2013

Service Quality and Profit Control in Utility Computing Service Life Cycles

Heckmann, Benjamin

BENJAMIN HECKMANN

SERVICE QUALITY AND PROFIT CONTROL IN UTILITY COMPUTING SERVICE LIFE CYCLES

Doctor of Philosophy, September 2012

SERVICE QUALITY AND PROFIT CONTROL IN UTILITY COMPUTING SERVICE LIFE CYCLES

by

BENJAMIN HECKMANN

RESEARCH WITH PLYMOUTH UNIVERSITY

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

September 2012 – version 1.1

ABSTRACT

SERVICE QUALITY AND PROFIT CONTROL IN UTILITY COMPUTING SERVICE LIFE CYCLES

BENJAMIN HECKMANN

Utility Computing is one of the most discussed business models in the context of Cloud Computing. Service providers are more and more pushed into the role of utilities by their customer's expectations. Subsequently, the demand for predictable service availability and pay-per-use pricing models increases. Furthermore, for providers, a new opportunity to optimise resource usage offers arises, resulting from new virtualisation techniques. In this context, the control of service quality and profit depends on a deep understanding of the representation of the relationship between business and technique.

This research analyses the relationship between the business model of Utility Computing and Service-oriented Computing architectures hosted in Cloud environments. The relations are clarified in detail for the entire service life cycle and throughout all architectural layers. Based on the elaborated relations, an approach to a delivery framework is evolved, in order to enable the optimisation of the relation attributes, while the service implementation passes through business planning, development, and operations.

Related work from academic literature does not cover the collected requirements on service offers in this context. This finding is revealed by a critical review of approaches in the fields of Cloud Computing, Grid Computing, and Application Clusters. The related work is analysed regarding appropriate provision architectures and quality assurance approaches.

The main concepts of the delivery framework are evaluated based on a simulation model. To demonstrate the ability of the framework to model complex pay-per-use service cascades in Cloud environments, several experiments have been conducted. First outcomes proof that the contributions of this research undoubtedly enable the optimisation of service quality and profit in Cloud-based Service-oriented Computing architectures.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

ASP Application Service Provision

API Application Programming Interface

BPEL Business Process Execution Language

BSLA Business Service Level Agreement

CMDB Configuration Management Database

CPU Central Processing Unit

CRM Customer Relationship Management

CS Computer Science

GRASP Grid-based Application Service Provision

GUI Graphical User Interface

HA High-Availability

IaaS	Cloud Infrastructure as a Service
IT	Information Technology
ITIL	IT Service Management Standard
MI	Million Instructions
MIPS	Million Instructions Per Second
NIST	National Institute of Standards and Technology, United States of America
OGSA	Open Grid Service Architecture
OSI	Open Systems Interconnection
PaaS	Cloud Platform as a Service
QoBiz	Quality of Business
QoE	Quality of Experience
QoS	Quality of Service
SaaS	Cloud Software as a Service
SLA	Service Level Agreements
SMF	Simulation Model Framework
SOA	Service-oriented Architectures
SOAP	Simple Object Access Protocol
SOC	Service-oriented Computing
TCO	Total Cost of Ownership
UC	Utility Computing
UML	Unified Modeling Language
VM	Virtual Machine
XML	Extensible Markup Language

ACKNOWLEDGMENTS

This thesis introduces the results collected over six and a half years of research. In this long period of time, many people contributed to my point of view on life in general and science in particular. My thanks go to all of them. Without some of whose assistance and knowledge this thesis would not have been successful - they therefore should be mentioned specifically.

In memory of and in gratitude to Prof. Dr. Günter Turetschek. He led my way into science as my initial Director of Studies.

Thanks to Prof. Dr. Christoph Wentzel for the energy he put into my supervision. Beyond his role as Director of Studies, he always cared about the personal aspects connected to the studies.

I would like to thank Prof. Dr. Ronald C. Moore for adopting the role of second supervisor. Our discussions significantly improved the comprehensibility of my results.

Special thanks to Prof. Andrew D. Phippen. Andy has always been the light at the end of the tunnel.

I would like to show my gratitude to Prof. Dr. Klaus Kasper. I have no idea, where he found the time - but he has been always there for long discussions about my research, science in general, and sometimes life in particular.

I would like to express my sincere thanks to Sebastian Abt. In his case, I know where he took all the time for our long discussions about life in general, my research, and science in particular.

Also to mention is Björn Bär. He dragged me through the last weeks of writing up. Thanks.

Thanks to Prof. Dr. Udo Bleimann for always seeing the positive site and for his gift of speech.

Special thanks go to Dr. Ingo Stengel for being an example for mastering the hard path.

I would like to express my sincere thanks to Marcus Zinn. He has been walking the path with me.

Special thanks to my girlfriend. Growing beyond herself, she is an example for me and her two kids.

I am indebted to my family for supporting me.

To my friends, especially those who still remember me: now I have time. Thanks for your patience.

The set of icons used is provided by http://dryicons.com. Thanks.

AUTHOR'S DECLARATION

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Graduate Committee.

Relevant scientific seminars and conferences were regularly attended at which work was often presented; external institutions were visited for consultation purposes. Several papers were published in the course of this research project, details of which are listed in the appendices.

Word count of main body of thesis: 48713 words

Signed: _____
Benjamin Heckmann

Date: _____

Part I

FIELD OF RESEARCH

1 INTRODUCTION

> "[...] since a service delivery process covers design, delivery, and operation, a common model of the customer's service business may be helpful to track the variations from any of the three phases and analyze their impacts on other phases."[1]
> — Liang-Jie Zhang (2007, p. 328)

The ease of offering and using Information Technology (IT) services is rising with the availability and bandwidth of the Internet in general. The more consumers rely on the availability of service offers, the more the providers of those services are pushed into the role of utilities; not intentionally, but driven by their customer's expectations qualified by the daily use of public utilities. Specifically in the B2B[2] markets, the demand for predictable service availability increases and has become one of the key properties of service contracting.

Another key property of contracting is service pricing. Understanding computing as a utility leads the customer to expect pay-per-use pricing models.

Service providers are thereby motivated to improve service offers in case of availability on the one hand and to provide pay-per-use pricing models on the other hand. Both of these varying goals can be reduced to the optimisation of the relation between customer, service and resource, as shown in Section 4.2. While the relation itself is unchangeable, its representation in service life cycles can be improved, as shown in Section 5.5.

The current representation of the customer-service-resource relation within a service life cycle reveals gaps in service level agreeing, life cycle information exchange, support for make-or-buy decisions, and economic-efficient quality control. These gaps prevent economic availability management and effective pay-per-use pricing models. This research introduces an approach to a delivery framework to narrow these gaps. The framework enables the development, examination, and operation of pay-per-use pricing models at the technical edge of business administration and software engineering.

1 Zhang stated this open research question aiming at service delivery platforms for multi-customer service offers. In the context of this thesis, the customers are service providers.

2 business to business

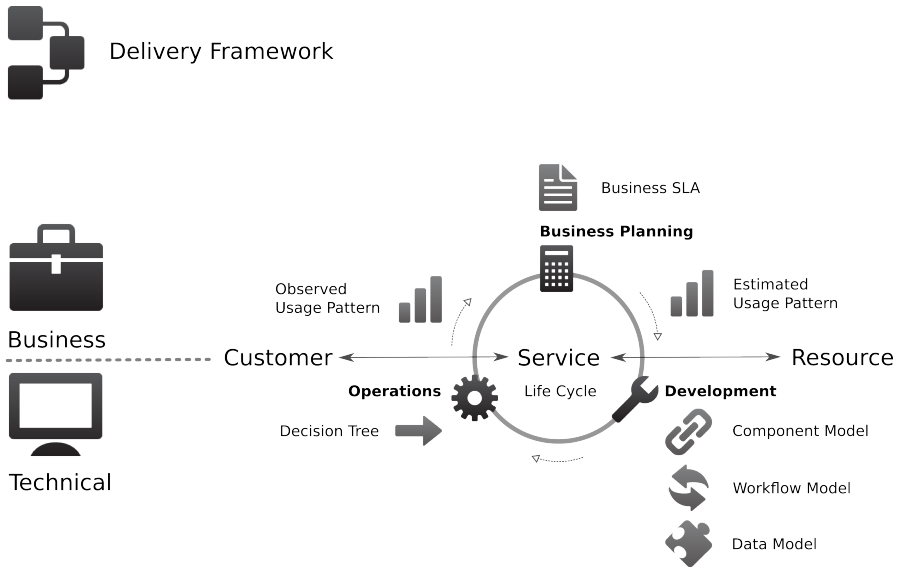


Figure 1: Delivery Framework to Support the Mapping of the Customer-Service-Resource Relation Onto the Life Cycle of Utility Computing Services

The framework consists of the components shown in Figure 1. The delivery framework supports the business planning with a data model for *business* Service Level Agreements (SLA), introduced in Section 5.3.4, and *usage patterns*, described in Section 5.3.2. The design of software architectures is supported with a *component model*, *workflow model*, and corresponding *data model*, evolved in the Sections 5.2.2, 5.2.3, and 5.4. In Section 5.3.3, the framework is completed with a *decision tree* to enable economic-efficient service request routing.

The research demonstrates the capabilities of the evolved delivery framework based on a simulation model. The simulation results are able to represent a set of common test cases for Cloud Computing environments.

1.1 ABOUT THE AUTHOR

The author has an experience of more than 13 years in IT consulting, specialised in the representation of business processes in Service-oriented Architectures (SOA) and in the operation of virtualised data centres. The author's professional experience started in 1999 as an IT administrator in a small company for Enterprise Resource Planing software. Afterwards, he worked as software engineer for a company specialised on Product Data Management software. In this context, the author was engaged in solutions for automated data exchange in business processes spanning multiple companies.

Since 2003, the author has been head of an applied research team at the University of Applied Sciences Darmstadt. The team conducted consulting and software development projects in the field of SOA. Concurrently, he built and operated a data centre in the research institute as head of operations. The data centre was the first to introduce virtualisation technologies in the university.

In 2006, the author was engaged in the foundation of a company for Business Process Management in the context of SOA. In 2009, he co-founded a business for IT consulting, in order to provide professional services regarding the research results gained during this thesis. Since 2012, the author has been co-founder of a data centre spin-off for Cloud Computing consulting.

During this work, some requirements and challenges started to occur more frequently. More and more business managers started to worry about the scalability of their service offers. But on the other hand, they often failed to specify properly and to communicate their usage expectations. For IT architects, it became more and more obvious that common architectural patterns for IT services would not scale in large dimensions. And in addition, IT architects often did not have this usage dimension clearly specified. The prediction of resource demands for the calculation of IT budgets started to make operations management getting heavily complex, with applications suddenly being able to allocate resources on their demand, while competing on the resource pool. A standardised usage description in the service life cycle could have improved all of the previously addressed problems.

During the business planning of IT service offers, business managers often struggle with the degree of freedom they have in the conception of service levels and pricing. This struggle typically results in concepts with expectations towards technology that are unrealisable or in service offers not being innovative, as they do not exhaust the possibilities. In typical discussions with IT architects, the planning of the scalability of service architectures often gets controversial. The ideas, which dependencies between actors in a component model are to be expected, often differ. Further elaborating this though, a lack of deeper knowledge about the data these actors have to exchange often leads to controversy. Operations managers often complain about not having a comprehensive insight into running services, contracted service levels, and usage expectations. There is a general uncertainty of what data should be collected to optimise resource demand predictions. A usage-centred data model could improve all of the previously addressed problems.

When it comes to the calculation of costs of service development and operations, most business managers draw back to the market

leaders. Often, because the comparison of provision solutions for Utility Computing services is difficult without the knowledge of a minimal set of features that should be claimed. Operations managers are often in the same position when it comes to the examination of alternatives to already productive provision solutions. In discussions with IT architects of provision solutions, it has often been discussed what kind of features should be implemented as part of a minimal set and in which architecture this should be done. A core provision model for Utility Computing services could improve all of the previously addressed problems.

Offering diverse levels of service quality for a bunch of services affects the whole service life cycle. Business managers often aim at the introduction of economic characteristics as metrics for automated decisions about resource usage. For IT architects, this introduces the need for a deeper knowledge about the interdependencies among infrastructure services in the horizontal application layer and vertical throughout all Open Systems Interconnection (OSI) layers. This view on service quality gets even more complex in operations, as the number of services rises. A model for usage-centred assurance of service quality could improve the previously addressed problems.

The previously implied problem of the complexity of service cascades in relation to infrastructure continues when the complexity of service cascades is analysed on the level of service interrelations. For business managers, this often leads to weak estimations of costs for resource demands and incorrect decisions on the buying-in of external service offers. From the point of view of IT architects, quality assurance for service cascades with different service levels has often been omitted as too complex. As detailed analyses of demands are often estimated as much too complex to handle, operations managers often simplify resource demands, resulting in generous over-provision.

In this context, in 2006 it became visible, that the idea of Service-oriented Computing (SOC) and virtualisation technologies would lead to a new paradigm of how resources in data centres are to be utilised. There was a strong indication, that these innovations would lead to new pay-per-use business models, demand advanced architectural models, create new hosting models, and that this would massively influence the service life cycle as a whole.

1.2 UTILITY COMPUTING SERVICE LIFE CYCLE

Business Model

Utility Computing (UC) (cf. Section 2.2) is introduced as a generic business model (cf. Section 2.2) in this thesis. It addresses IT services

that are offered to a broad range of customers corresponding to services provided by public utilities (cf. Section 2.2). The model reflects the core strategical aspects of IT service offers:

- Multiple differentiated customer groups;
- Resource scalability;
- Usage-centred pricing models;
- Reliability of quality agreements.

In a wide range of aspects, this definition relies on implicit properties inherited from business models of public utilities (e.g., necessity, usability, exclusivity; cf. Section 2.2).

As a business model blueprint, UC does not refer to any specific architectural or hosting model. Any IT service provided corresponding to the UC definitions can be referred to as UC service (parallel computations, object storage, mail service, web applications, virtual machines (cf. Section 2.4), etc.).

Architectural Model

SOC (cf. Section 2.3) is based on the approach of SOA (cf. Section 2.3) which introduces a loose coupling between providing and consuming components[3] in software architectures. SOC is an extension to the approach of SOA. Here, loose coupling is also introduced within the context of an architectural component itself. This leads to the concept of composite services represented by SOC. The composition is implemented using a workflow language as composition descriptor. In the end, the consequent realisation of building services out of existing services leads to a network of highly meshed interdependent services called *service cascade* (cf. Section 2.3).

Hosting Model

The provision of the following types of IT services is specified by the term *hosting* of infrastructure, platform or software in this thesis:

- Basic IT infrastructure (e.g., storage, computing resources);
- Application platforms like VMware's Cloud Foundry (VMware, 2011a), Microsoft's Windows Azure (Microsoft, 2011), or Amazon's Elastic Beanstalk (Amazon, 2011);

3 Logical group of functionalities within a software architecture, see Section 2.3 for a detailed definition.

- Maintained software instances (mail services, web applications, etc.).

Cloud Computing (cf. Section 2.4) subsumes these hosting characteristics and targets an extensive automation of the corresponding management operations in order to provide those services. Specific technical features for cloud-enabled provision platforms are not specified in the Cloud Computing model.

Service Life Cycle

Assuming the position of a service provider, each new offer of services like a hosted Customer Relationship Management (CRM) system, similar to Salesforce.com (2011), starts with the planning of its expected business potential. This *business planning* leads to detailed assumptions about expected provision costs and estimated service usage. Based on these expectations, during *development* the service is implemented as a software application. After reaching the aimed technical quality, the service is deployed to its operations environment. In *operations* the service is maintained continuously until market concerns imply changes to service features. Then the service's life cycle restarts in a new phase of business planning.

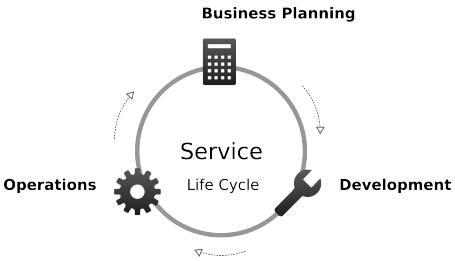


Figure 2: Simple Generic Service Life Cycle

In summary, a basic service life cycle comprises the phases:

- Business planning;
- Development;
- Operations.

The introduced life cycle is based on the IT Service Management Standard (ITIL) (cf. Section 2.5).

Life Cycle of UC Services in Service-oriented Computing, Hosted in Cloud Environments

This research addresses the mapping of the customer-service-resource relation into the life cycle of Utility Computing services in Service-

oriented Computing Architectures hosted on Cloud Computing platforms assuming the position of service providers.

As a process of continuous increase of division of labour, industrialisation of the information technology industry carries on. Due to its success, the Internet provides the infrastructure to offer IT services to a broad range of customers, reaching from international enterprises to private consumers. Service providers are easily enabled to offer their services within different markets. Being successful with this strategy is forcing providers into the role of a utility; pushed by their customer's expectations. Market success broadens the range of how service offers are used and relied on with a dynamic driven by the service consumers like implied by O'Reilly's statement: "customers are building your business for you"[4] (Batelle and O'Reilly, 2004). Therefore, this research examines Utility Computing as relevant future business model for service providers.

IT services scaling available resources dynamically matching customer demands and metered by usage might be implemented based on a broad range of architectural patterns (e.g., monolithic application, client–server model, peer-to-peer model). Keeping the ongoing IT industrialisation in mind, reliable offers - like characterised by UC - encourage providers to increase the level of service reuse, including services from other vendors. Complex service cascades depend on software architectures able to reflect these interdependencies. To meet this requirement, SOC is chosen as architectural model for the implementation of UC services in this research. (Marks and Lozano, 2010)

Hosting multiple services based on SOC architectures leads to specific requirements on the underlying hosting model. Multiple UC services overlap in their feature sets regarding dynamic resource scaling and usage metering. This enables optimisation potential within the overall provision architecture. By relocation of UC-specific features into the underlying hosting layer, this optimisation is achieved. Cloud Computing enables the dynamic resource scaling as feature subset and is therefore chosen as hosting model in this thesis.

Service provision is about optimal resource utilisation in order to maximise service profits assuming the position of a service provider. After the decision to offer services conforming to the business model of UC, efficiency estimations have to be made. This part of the business planning involves estimations for service development as well as its future operational cost. To calculate the Total Cost of Owner-

4 Which was originally stated in the context of the Web 2.0 discussion, but is carried on to web-based service offers in general in this thesis.

ship (TCO)[5] and oppose it to the estimated sales revenue, the service's entire life cycle has to be taken into account.

For a more detailed setting of the IT scenery for service life cycles in this thesis see Section 2.6.

1.3 PROBLEMS IN SERVICE QUALITY AND PROFIT CONTROL

The problems described in the following refer to the research context stated in Section 1.2.

The basic properties of service offers conforming to the business model of Utility Computing are the ability to serve differentiated customer groups, to scale resources on demand, to price customers by usage, and to keep reliable quality agreements. All this properties target the relation between customer, service, and resource (see Figure 1). Thus, in the following this relation is referred to as the *core relation* of Utility Computing. Analysis of this relation reveals the following three main problems (cf. Section 4) introduced as:

- P_1 — Description of the customer-service-resource relation;
- P_2 — Control of service quality;
- P_3 — Analysis of complex service cascades.

Description of the Customer-Service-Resource Relation

The relation between customer, service, and resource is only indirectly represented within the life cycle of an IT service. Analysing the phases of a life cycle from business planning, through development, to service operations, three local problems within the characteristic of the relation occur. These problems prevent a direct relation between customer, service, and resource and, in summary, describe the first main problem, introduced as P_1, for providers of UC services. The below listed properties are missing in the relation between customer, service, and resource.

- P_{1a} — Standardised usage description for data exchange between phases of a life cycle

  In business planning, the complexity of service cascades in UC scenarios requires an effective foundation for the analysis of price and cost models in alternative usage and resource scenarios. Such an analysis can be accomplished using specialised software (e.g., simulation frameworks) assumed a standardised de-

5 TCO indicates the total cost related to the provision of a service. It includes the direct and indirect cost spread over the service's life cycle. (Ellram, 1995)

scription of the service usage exists. During development, missing usage descriptions obstruct significant performance analysis to verify the estimated cost of operations.

Usage descriptions are also essential in the context of service quality control and usage accounting.

- P_{1b} — Comprehensive usage-centred data model

  The basis for analysis of the customer-service-resource relation is its formal description. Without such a description, neither the data exchange between phases of the life cycle nor a core provision model for UC services are technically feasible. As formal form of description of the customer-service-resource relation, a data model is chosen. A usage-centred data model enables data exchange by offering the overall set of relevant data. Also, a provision model requires a data model as anchor for its provision workflows.

- P_{1c} — Core provision model for Utility Computing services

  Commonly, services are not developed from scratch but based on existing application platforms (e.g., J2EE). The decision for an adequate application platform is complicated through the missing description of the minimum requirements on UC provision. This leads to a weak basis for cost estimations in the development of UC services.

Control of Service Quality

Current monitoring of service quality is mainly based on technical thresholds (e.g., processor load, memory consumption). Economic characteristics, like the profit per service request, are not considered during runtime. In addition, there is no continuous combined economic and technical control of service quality for UC environments (Rust, 2009). Necessarily, this control would have to reach through all OSI layers (Zimmermann, 1980) from a consumer to a resource. This control is currently also missing due to the fact that in practice there is no detailed knowledge about the dependencies of services among one another. Detailed control of delivery quality cannot be implemented without providing a continuous approach to economic and technical control, introduced as second main problem P_2 in the UC core relation.

Two essential requirements on UC are high service availability and scalability. In essence, these properties are based on the ability to monitor and control service quality in detail on all technical layers from customer to processing resource. Known technical monitoring

solutions[6] do not completely cover all architectural layers (Rust, 2009), also due to the fact that previous to this thesis there was no suitable approach to a core provision model for Utility Computing services available.

Internally, providers make use of overbooking of resource on purpose, taking contracted penalties into account. None of the known technical solutions consider economic characteristics at runtime to control provision quality in cases of resource overbooking (cf. Section 4.3). Instead, economic efficiency has to be the central benchmark for control of service quality in Utility Computing scenarios.

In addition, feasibility of business processes based on SOC architectures cannot be reliably estimated based on approaches that focus on technical monitoring, as shown in Section 4.3.2.

Analysis of Complex Service Cascades

In strict service-oriented software architectures complex usage relations arise quickly among the involved services, even with a small number of services given. For a service cascade [7] involving n services, the maximum number K_n of edges is calculated as $K_n = \frac{n(n-1)}{2}$ [8] (West, 1999). For a service cascade involving 20 services, this results in K_n= 190 possible edges[9]. This results in a maximum complexity of $\mathcal{O}(n^2)$ for a fully meshed service cascade.

Continuing the example, it is assumed that the number s_u of actually used service relations - out of all possible edges K_n - is only one third of K_n. It is also assumed that at least two kinds of services can be distinguished, in this example called business and basic services. Basic services offer highly reused essential features to business services. It is assumed that the number s_u of basic services in the example cascade is also on third of K_n. Also, it is assumed that basic services are by the factor of r_b = 2 more often invoked than business services.

In the example, there are g = 2 user groups with distinct usage behaviour regarding their type of service usage. The resulting number of service relations in this example can be calculated as $R = \sum (K_n s_u, K_n s_b r_b)\ g$. For the estimated 20 services this results in a service cascade involving 380 service relations. The complexity of service cascades states the third main problem in the UC core relation, introduced as P_3.

6 Amberpoint, Progress Actional, SOA Manager Service Manager, Oracle Enterprise Manager SOA Management Pack and OpTier CoreFirst

7 A service cascade is defined as connected vertices in a graph, where each connection represents a distinct type of service usage (cf. Section 2.3).

8 for complete graphs with n vertices

9 representing usage relations of services

The more complex such cascades become, the more complicated becomes the analysis of their interdependencies. The more services from other providers are embedded into the own service cascades, the more rises the complexity for business managers, as those have to consider individual pricing models or alternative standby providers. The more scenario alternatives have to be compared, the more complexity rises, too.

The manual analysis of the economic efficiency of a single service cascade or a complete service-oriented software architecture is expected to be very complex, as described in the example above. Assuming consistent usage-based accounting including bought-in services and redundant providers, such complex scenarios cannot be analysed manually. If additionally multiple usage scenarios have to be analysed, the complexity of an analysis rises again.

The control of service quality and its planning in the described scenarios is challenging. Concerning this complexity, fine itemised service quality classes can neither be manually implemented during planning nor managed at runtime.

1.4 RESEARCH APPROACH

Within the research context stated in Section 1.2, the research focuses on the representation of the core relation of Utility Computing (defined in Section 1.3) within a generic service life cycle. The research raises the question, whether this customer-service-resource relation indicates specific life cycle properties. The existing knowledge in this area suggests that the UC context has a significant impact on certain life cycle properties. Deduced from the introduced position, the following research was conducted. As research methods, simulation in conjunction with modelling are chosen (cf. Section 2.1).

A. Analysis of Utility Computing requirements on service life cycles

   The results of the analysis is presented in Chapter 4. Based on the analysis of the essence of UC in the context of this research, three identified main problems $P_{1\text{-}3}$ are analysed. Within a generic life cycle for SOC services, the UC-specific relations are identified and elaborated. The UC-specific requirements on the control of provision quality are characterised assuming the position of service providers. In addition, this chapter introduces results of the analysis of the primary requirements on UC provision platforms. These results are summarised at the end of the analysis and result in detailed representation of requirements

on the development of approaches to resolve the previously identified problems in service life cycles.

B. Elaboration of a usage-centred provision approach

In Chapter 5 the elaborated approach to an improved representation of the core relation of Utility Computing in service life cycles is presented. For the elaboration the previously identified requirements are evaluated and evolved into three concepts. The combined approach covers a core provision model, a concept for usage-centred assurance of service quality, and a usage-centred data model. Within the chapter it is demonstrated how these concepts interact with a generic life cycle.

C. Evaluation of a simulation model for service cascades in Utility Computing

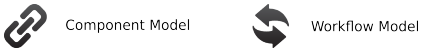
Based on the previously evolved approach in Chapter 6 the simulation model for service cascades in Utility Computing is build. Therefore, the core provision model, the concept for usage-centred assurance of service quality, and the usage-centred data model are incorporated into a single simulation model for multi-tier service architectures. Beside model building, the scientific background, capabilities and restrictions of the simulation model are discussed. In Chapter 7, the model is evaluated in comparison to a set of common test cases for Cloud Computing environments.

Concluding this research, its contributions, limitations, and a scope for further work are presented.

1.5 CONTRIBUTIONS

In order to advance the representation of the core relation of Utility Computing (defined in Section 1.3) in service life cycles, a delivery framework is introduced in this research as a combined approach, as shown in Figure 1. The delivery framework consists of six contributions consolidated in three concepts (see Chapters 5 and 6). The evolved approach focuses on UC services in Service-oriented Computing architectures hosted on Cloud Computing platforms. Assuming the position of service providers, the approach contributes three key findings to support UC service offers in a generic service life cycle, listed below.

- Technology-independent[10] core provision model for Utility Computing platforms



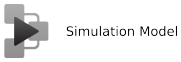
  The provision model is implemented as generic technology-independent component architecture. The model describes the minimum necessary functionalities and dependencies among function providers in an operations environment for UC services within this research context. Beside the components, representing logically grouped functions, the model contains corresponding basic workflows describing the minimum necessary interaction of these components to provide a UC operations environment. The model has to be incorporated into an architectural pattern to prepare a software/provision architecture for UC services.

- Concept for usage-centred assurance of service quality

  In the concept of usage-centred assurance of service quality the monitoring and control of provision quality does no more rely on technical criteria in the first place. The concept is determined by a usage-centred quality specification approach. To enable a continuous quality specification based on usage descriptions for services an adequate data model has been evolved. Embedded into the data model, a description pattern - introduced as *usage pattern* - covers the quantitative description of the usage relations in service cascades. Based upon, business SLAs are introduced as alternative to contract on service usage rather than on technical resource limits. Attendant, on the qualitative side of the usage relation, the concept demonstrates a decision tree for runtime quality control, bringing together the influencing factors to optimise economic provision efficiency.

10 Definition of technology-independence in this thesis: In which way an IT architect chooses to integrate the functionalities and workflows the model demands is dependent on the service's context. For a single service offer the integration into a simple 2-tier software architecture might be appropriate, as known from script-based web applications. For multiple service offers the integration into the architecture of the provision platform might be appropriate. In this sense, the model is independent of technologies and corresponding architectural patterns of service or platform implementations.

- Technology-abstracted[11] resource and cost simulation model

  The formerly introduced contributions form the basis for the building of the evolved simulation model. The simulation model has been built having chosen a generic multi-tier architectural pattern as example architecture. Simulation runs enable the analysis of service cascades for their interactions in case of resource demand, price and cost evaluation, and structural weaknesses (e.g. loops, single points of failure). The simulation model permits analysis of multiple service providers, each offering multiple services hosted in multiple data centres, and analysis of multiple consumer groups.

The introduced simulation model represents the essence of the core provision model and the concept for usage-centred assurance of service quality. In Chapter 7, this research evaluates the results of simulation runs in comparison to a set of common test cases for Cloud Computing environments. First outcomes of the evaluation show the ability of the simulation model to reflect essential characteristics of the common test cases and prove the applicability of the introduced approach in the given scenarios.

1.6 OUTLINE OF THE THESIS

The thesis is structured in eight chapters. The following list illustrates the order of the chapters and their objectives.

1. Introduction

   Within this chapter, the scene for the conducted research is set. The life cycle of Utility Computing services for Service-oriented Computing Architectures hosted in the cloud is briefly described as research context. Within this life cycle, the main problems in service quality and profit control are presented. In addition, the conducted research approach is specified. As contributions of this research, the combined approach of delivery framework including a technology-independent core provision model for Utility Computing platforms, a concept for usage-centred assurance of service quality, and a technology-abstracted resource and cost simulation model is characterised.

11 Definition of technology-abstracted in this thesis: The simulation model can be used to simulate any service cascades matching a multi-tier architectural pattern. In this sense, it is independent of the technical implementation of the services, but bound to the chosen architectural pattern.

2. Background

   The scientific and engineering foundations of the research context are introduced and corresponding terms, used in this thesis, are defined within this chapter. The chapter starts with the description of scientific research methods in the field of computer science. Afterwards, the research context is defined in detail.

3. Related work

   The works of other authors in the scientific fields related to this research project are presented and demarcated to the research of the author. The chapter begins with the introduction of approaches for provision models for utility computing platforms. It is distinguished by utility computing/cloud models, Grid models, and application cluster models. Concluding, the chapter introduces approaches for the usage-centred assurance of service quality.

4. Requirements concerning a generic service life cycle

   The specific requirements of Utility Computing services on a generic service life cycle are introduced within this chapter. Presented are the UC-specific relations within a service life cycle, the results of the analysis of the service quality assurance, and the evolved primary requirements on UC provision platforms.

5. Usage-centred provision approach

   For the requirements on Utility Computing services, elaborated in Chapter 4, this chapter introduces an approach to advance the representation of these requirements on UC service life cycles. As research results, a combined approach to a technology-independent core provision model for Utility Computing platforms, a concept for usage-centred assurance of service quality, and a usage-centred data model are described.

6. Simulation model framework

   In this chapter, a technology-abstracted resource and cost simulation model based on the research results provided in Chapter 5 is evolved. The implementation of the model as a simulation model framework for multi-tier architectures hosting UC services is described. As part of the description, it is shown how the previous research results are embedded in the framework. The chapter also includes an overview of computer simulation and a discussion of the features and limitations of the developed framework.

7. Evaluation of the simulation model framework

   The test cases, experimental setup, and results of several simulation runs are presented in this chapter. The simulation model framework evolved in Chapter 6 is evaluated on its capability to model basic and advanced cloud environments. It is shown that the framework is able to comply to this challenges.

8. Conclusions

   This chapter summarises the objectives, research approach, and contributions of this thesis. In addition, the limitations of the thesis are discussed. Concluding, potentials for future work is presented. The research is concluded with a technology review.

2 BACKGROUND

2.1 RESEARCH METHODOLOGIES IN THIS THESIS

This chapter illustrates which understanding of science leads to the outcomes in this thesis. Introduced is the scientific background (cf. Section 2.1.5) of the chosen scientific methods (cf. Section 2.1.1). In addition, the chapter gives a short and simplified definition of computer science (cf. Section 2.1.4) and corresponding scientific methods (cf. Section 2.1.3, 2.1.2).

2.1.1 *Research Objectives and Corresponding Scientific Methods*

In the applied computer science fields related to Utility Computing (cf. Section 2.2, 2.3, 2.4, and 2.5), the previously introduced research objectives (cf. Section 1.3) have been examined. These objectives represent the observed consequences in the provision of Utility Computing services (cf. Section 4).

The hypothesis is stated that an improved representation of the core relation of Utility Computing within a generic service life cycle (cf. Section 1.2) leads to an all in all more cost efficient service provision. Further, it is assumed that a core provision model, a concept for usage-centred assurance of service quality, and a resource and cost simulation model (cf. Section 5) are capable of achieving such goals.

As combined approach, the scientific methods of modelling and computer simulation (cf. Section 2.1.2) are chosen to elaborate the proposed models and concept. To test the hypothesis, a simulation model based on the findings is implemented (cf. Section 6). In comparison to a set of common test cases for Cloud Computing environments, simulation runs are evaluated (cf. Section 7) as proof of concept.

2.1.2 *Overview of Scientific Methods in Computer Science*

Referring to Dodig-Crnkovic (2002), in Computer Science (CS) four varying areas can be distinguished: modelling, theoretical, experimental, and simulation computer science, as listed below. Modelling is specific in the sense that it may precede the other methods in most cases.

- Modelling

  In order to be studied, the phenomenon of interest must be simplified in one sense. This process is called modelling. Modelling always occurs in science as a first step of abstraction. A model simplifies the phenomenon while taking the relevant features of the phenomenon into account. Simplified models provide a sort of description in some symbolic language. Models enable predictions of observable/measurable consequences of changes in a system.

  The methods left - theory, experiment, and simulation - all address models of phenomena in more or less detail.

- Theoretical computer science

  Theoretical computer science adheres to the traditions of logic and mathematics. It follows the very classical methodology of building theories as logical systems, which means it uses stringent definitions of objects (axioms) and operations (rules) for deriving/proving theorems. Theoretical computer science seeks to develop general approaches to problem solving and the understanding of computational paradigms. Theoreticians distil knowledge acquired through conceptualisation, modelling and analysis.

- Experimental computer science

  Although the subject of inquiry in the field of computer science is information, it makes no difference in the applicability of the traditional scientific method. Experimental computer science seeks to understand the nature of information processes by observation of phenomena, formulating explanations and theories, and testing them.

- Computer simulation

  Computer-based modelling and simulation has become the third research methodology within computer science. This method completes theory and experiment. Based on computer simulation, it is possible to investigate regimes that are beyond current experimental capabilities. In the realm of science, computer simulations are guided by theory as well as experimental results. The results often suggest new experiments and theoretical models.

2.1.3 *A Quick Classification of Scientific Methods*

Two types of methods are distinguished by the evaluated data: primary and secondary research. Primary research is focused on the collection and evaluation of original data, while opposing secondary research evaluates the existing data (generated by previous primary research) to gather new findings. (Clarke, 2005)

In addition, research methods can be classified by their process of theory construction. Distinguished are quantitative and qualitative methods. Quantitative research aims to collect data corresponding to a structured and standardised way from a preferably large random sample. In opposite, in qualitative research data is collected based on a preselected rather small characteristic sample. Data in qualitative research is gathered by relevance. Data acquisition is continuously being adapted based on previous research outcomes. If required, various research methods can be mixed depending on the research objectives. (Winter, 2000)

As an example for a common scientific method, the basic quantitative method of science derived from the Socratic method is presented in the following.

1. Pose a new question in the context of existing knowledge.
2. Formulate a tentative answer as new hypothesis.
3. Deduce consequences and predict results.
4. Test the new hypothesis in experiments and/or theoretically.
   The new hypothesis must prove its precision by fitting into the existing world-view. Position 2., 3., and 4. are to be repeated in a loop with modifications of the hypothesis until 5. is obtained. If major discrepancies are found, the process must restart at 1. with a more accurate question.
5. When consistency is obtained, the hypothesis becomes a theory.
   Theories provide a coherent set of propositions that define a new class of phenomena. Or theories introduce a new theoretical concept. At this stage, a theory gets subject of the process of natural selection among competing theories. Then a certain theory becomes a framework in which observations/theoretical facts are explained or predictions are made.

In this thesis, a scientific method is defined as a structured process that leads from a question to a proven and generic answer.

2.1.4 *Computer Science & Engineering in a Nutshell*

The definition of computer science and its demarcation from other domains of science and engineering has been discussed since its beginnings. Still it is only possible to give a vague overview of the state of this discussion.

Studies in the field of computing[1] commonly involve scientific and engineering parts. The scientific part deals with the investigation of research objectives and broadens the knowledge within the affected area of computing. Research in the scientific parts of computing is defined as computer *science* in this thesis. In opposite, the engineering part of studies is focused on the construction of reliable solutions based on already given knowledge. Consequently, the engineering part is defined as computer *engineering* within this thesis. (Dodig-Crnkovic, 2003; Hansson, 2009)

Many fields within computer science have important links to other fields such as Mathematics, Psychology, Cognition, Biology, Linguistics, Philosophy, Behavioural, and Brain Sciences among many others. One of the major qualities of computer science is to combine knowledge from all these fields. This also includes the possibility to operate with the scientific methods (cf. Section 2.1.3) related to these specific sciences. (Dodig-Crnkovic, 2003)

2.1.5 *Science, a Short Definition*

φιλοσοφία[2] can be literally translated as *love of knowledge*. Philosophy is the mother of all science.

The beginnings of Western philosophy are credited to the Greeks of the classical period. They believed that human reason is competent on its own account - not dependent on religious or mystical principles - to formulate the right questions, and to seek answers to them, concerning every matter of interest or importance to humanity. Starting with this perspective, the Greeks of the classical period inquired freely into all aspects of the world and humankind. (Grayling, 1999)

From this nucleus, philosophy started as a method (cf. Section 2.1.3) to explore humans and their surrounding world. The more knowl-

1 The German, French and Italian languages use the respective terms 'Informatik', 'Informatique' and 'Informatica' (Informatics in English) to denote Computing. It is interesting to observe that the English term 'Computing' has an empirical orientation, while the corresponding German, French and Italian term 'Informatics' has an abstract orientation. This difference in terminology may be traced back to the tradition of 19th-century British empiricism and continental abstraction, respectively. The view that information is the central idea of Computing/Informatics is both scientifically and sociologically indicative. Scientifically, it suggests a view of Informatics as a generalisation of information theory that is concerned not only with the transmission/communication of information but also with its transformation and interpretation. Sociologically, it suggests a parallel between the industrial revolution, which is concerned with the utilising of energy, and the information revolution, which is concerned with the utilising of information. (Dodig-Crnkovic, 2003)

2 philosophia, Greek for philosophy (Liddell and Scott, 2011)

edge[3] and specific methods were collected, the more expert fields within philosophy evolved to discrete sciences with their characteristic sets of proven methods of exploration and description of their certain domain of being.

Philosophy has given birth to modern sciences over the last centuries. In the seventeenth century, natural science detached from philosophy, growing into a fully qualified field of science on its own. In the eighteenth century, the same happened to psychology as well as to sociology and linguistics in the nineteenth century. In the twentieth century, philosophy has played a large part in the development of computer science, cognitive science, and research into artificial intelligence. (Grayling, 1999)

While the origin of science is sufficiently clear, it remains difficult to describe the nature of science based on one generic model. De Jong and Betti offer an approach to a generic definition that refers to science based on the classical model[4] of science. Here, a science is a system S of propositions and concepts (or terms) which satisfies the conditions listed below:

1. All propositions and all concepts (or terms) of system S address a specific set of objects or deal with a certain domain of being(s).
2. There are a number of so-called fundamental concepts (or terms) in system S.
3. All other concepts (or terms) occurring in S are composed of (or are definable from) the fundamental concepts (or terms) defined in 2..
4. There are a number of *fundamental propositions* in system S.
5. All additional propositions of S follow from or are grounded in (or are provable or demonstrable from) these fundamental propositions.
6. All propositions of S are true by definition.
7. All propositions of S are universal and necessary.
8. All non-fundamental proposition are known to be true through their proof in S.

3 In the scientific context and in this thesis, knowledge is built out of theoretical representations that serve as abstract images of the objective world. (Dodig-Crnkovic, 2004) Beyond the theoretical models, the meaning of the term *knowledge* may consist of more dimensions like experience.

4 The term *classical model* refers to the model of the Greeks of the classical period.

9. All concepts (or terms) of S are adequately known. Non-fundamental concepts are adequately known through their compositions (or definitions).

(Jong and Betti, 2008)

Despite the linear definition in this chapter, "there is no general account of science and scientific method to be had that applies to all sciences at all historical stages in their development" (Chalmers, 1999).

Dodig-Crnkovic (2009) recapitulates that "the Classical Model of Science is a reconstruction a posteriori and sums up the historical philosopher's ideal of scientific explanation".

In summary, in this thesis a science incorporates a well defined, but not closed, set of specific concepts, propositions, and methods to describe and explore a certain domain of being.

2.2 UTILITY COMPUTING AS BUSINESS MODEL

Utility

Utility Computing is concerned with offering computing as a utility. Thereby, the term *utility* refers to the field of industry. In industry, the term public utility (Britannica Online Encyclopedia, 2011) is used for enterprises that offer certain classes of services to a wide and heterogeneous range of consumers (e.g., consumers of electric utilities). (Rappa, 2004) defines six common criteria to characterise utilities, listed below.

- Necessity

  Consumers depend on utility services to fulfil their day-to-day needs.

- Reliability

  The service provided by a utility must be readily available when and where the consumer needs it.

- Usability

  No matter how technologically complex they may be on the production end, utility services are characteristically simple at the point of use.

- Utilisation

  Utilities are driven by the need to carefully manage utilisation rates. User demand for utility services can fluctuate widely over time and across the service region.

- Scalability

  Utilities are commodity businesses. Therefore, utility services can exhibit significant economies of scale that prefer larger providers to smaller ones.

- Exclusivity

  The economies of scale in a utility can benefit from a monopolistic provision of services.

The provision of Utility Computing services in this thesis is focused on the utility criteria necessity, reliability, and utilisation. Usability is estimated as a criterion of a specific service implementation in this thesis. Scalability and exclusivity, on the other hand, are abstract economic criteria in the context of Rappa. Their impact on service delivery is not examined in this thesis.

Business Model

As a first definition, Osterwaelder elaborates the term *business model* based on an analysis of the combined expressions *business* and *model*. This approach leads to the simple definition of "a representation of how a company buys and sells goods and services and earns money" (Osterwalder, 2004; Seppanen and Makinen, 2005). As a more precise definition, Osterwaelder states:

> "A business model is a conceptual tool that contains a set of elements and their relationships and allows expressing a company's logic of earning money. It is a description of the value a company offers to one or several segments of customers and the architecture of the firm and its network of partners for creating, marketing and delivering this value and relationship capital, in order to generate profitable and sustainable revenue streams." (Osterwalder, 2004; Seppanen and Makinen, 2005)

Also, Osterwaelder mentions that there are ongoing discussions on the demarcation of the terms *business strategy*, *business model*, and *business process model*. Business models describe more concrete short-term strategies for business development compared to business strategies. Business strategies describe long-term strategies for business development. Business process models specify the processes necessary to implement the business described in a business model and guided by a business strategy. As simplification, it is defined that strategy, business models, and process models address similar problems on different business layers. (Osterwalder, 2004; Seppanen and Makinen, 2005)

Utility Computing

As implicated by its name, Utility Computing depicts the vision of IT-based service offers comparable to service offers of common public

utilities. In this thesis, *Utility Computing* is defined as generic business model (Rappa, 2003; Foster et al., 2008) for service provision of IT-based services that are offered to a broad range of customers and accounted by service usage (Rappa, 2004).

From the point of view of a service provider, UC is about service offers that are, due to real-time fluctuations in consumer demands (Bunker and Thomson, 2006), enabled to scale dynamically and that are accounted based on the actual resource usage occurring during the processing of related service requests (Neel, 2002; Mendoza, 2007). The core strategical aspects of UC offers that are inherited from the implicit properties in business models of public utilities (cf. Section 2.2) are:

- Multiple differentiated customer groups;
- Resource scalability;
- Usage-centred pricing models;
- Reliability of quality agreements.

Taking the position of a service consumer, "the reduction of IT-related operational costs and complexity" (Yeo et al., 2006) is in focus. Both sides, provider and consumer, target the optimisation of their generally underutilised IT resources (Andrzejak et al., 2002; Heap, 2003). UC does not refer to any specific technical properties for service architectures or hosting environments.

Service Contract

In this thesis, it is assumed that customers and providers conclude legal compliant contracts before service offers are used. It is assumed that such contracts include agreements on usage-centred service pricing, service billing, and processing quality. (Chee and Franklin, 2010; Chou, 2011)

The agreement on processing quality incorporates the below listed properties, defined in this thesis.

- Availability

  The availability is determined from the provider's point of view as the time interval a service offer is technically usable conform to the contracted conditions.

- Response time

  Response time is determined from the provider's point of view as the time interval between initial service request receipt and

the terminal service request response drop out on the provider's perimeter.

- Data security

  The properties of data security define the physical and organisational access restrictions to prevent unauthorised data access. To prevent data loss, disaster recovery measures are agreed upon.

- Resource allocation limits

  Minimum[5] and maximum[6] limits for the allocation of processing resources on the provider side of the agreement are defined to enable reliable cost estimations for both contracting parties, while enabling the scaling of resource demands. The limitation defines a range of resource occupation during the processing of service requests (e.g., network bandwidth, processing cycles, memory, and storage).

In addition, for processing quality violations monetary penalties are agreed upon between customer and provider.

The properties concerning processing quality are explicitly not addressed as SLA in this thesis. An approach to handle SLA is presented in Section 5.3.4.

Usage-Centred Pricing Model

As pricing model, in this thesis, an algorithm for the calculation of a service's gross price is defined. For example, a price model for coffee P_c could be determined as the coffee's price per weight p_w multiplied with the actual number of weight units consumed c_w. The simple price model for the example is then $P_c = p_w \, c_w$. More complex models might take into account certain discounts (e.g., for large quantities, specific service levels, or strategic customers). Pricing models are build on corresponding pricing strategies, which are not considered in this thesis. (Monroe, 2003)

Based on to the previous definition of pricing models the usage-centred pricing models are introduced as a specific subgroup of models focused on dynamic pricing. Dynamic pricing is classified as a class of models with complex gross price algorithms that represent a direct relation between the gross price and the consumed amount of resources during processing. (Bitran and Caldentey, 2003; Maglaras and Meissner, 2006)

5 the customer's point of view
6 the provider's point of view

Dynamic models with market-adaptive pricing, where gross prices of competitors are part of the pricing algorithm (e.g., in auctions), are not considered in this thesis. (Cohen et al., 2008)

The term *pricing model* is used as a synonym for usage-centred pricing models in this thesis.

Service Billing

In this thesis, the term billing is generically defined as the binding information about the gross price of a processed service request transferred in real-time from the service provider to the service consumer. In this context, it does not include the generation and delivery of a legal compliant invoice (e.g., on a monthly basis).

2.3 SERVICE-ORIENTED COMPUTING AS ARCHITECTURAL MODEL

Services in Utility Computing

In computer science the term *service* is used differently in many fields. A common definition does not exist, but there are domain-specific definitions given. In the context of Utility Computing, the definition of a service has to integrate the two viewpoints of economics and IT. As Utility Computing is defined as business model, any IT service whose technical implementation is enabled to be delivered as UC service (in sense of scalability, pay-per-use and quality assurance) is addressed by the service definition in this thesis. Therefore, the following, rather abstract, service definition is given.

The term *service* represents a type of relationship-based interaction between service consumer and service provider to achieve a certain solution objective (Liang-Jie Zhang, 2007). In other words, a service represents a single unique interaction between consumer and provider. Although this definition has been evolved from a technical point of view, it conforms to the economically motivated definition of Fitzsimmons and Fitzsimmons (2006) and to the one given by Groenroos (2000) from a marketing point of view.

To narrow the field of research, SOAs are chosen as reference software architecture in this thesis. SOA introduces an approach to create software architectures out of loosely coupled service components. Software architectures that are based on the paradigm of loosely coupling support the scalability and quality assurance characteristics of UC. Each service interconnection implies a point of contact to distribute service requests to additional service instances or to interact otherwise with the service request flow driven by predefined quality criteria.

Service-Oriented Architectures

SOAs (Erl, 2005; Bieberstein, 2006) introduce loose coupling of service consumers and corresponding providers (Maximilien and Singh, 2005; Oasis, 2006) as paradigm for software architectures. Thereby, SOA requires that the functionality a software application provides is logically grouped into components on the level of software architecture. Those architectural components are represented by independent source codes on the level of software implementation. To achieve loose coupling within a software architecture, the service consuming component looks up a service offering component in the service registry, where a service provider publishes its service offers.

The term *service* is not clearly defined in the domain of SOA. Most authors think of services as a synonym for software components (D Souza and Wills, 1998) that are accessible via web service interfaces (Haas and Brown, 2004). Other authors use the term more strictly to exclusively describe IT-based representations of business services on the conceptual level of software architectures (Humm, 2008).

The following listing states the key principles that determine the interface characteristics of SOA service offer.

- Abstraction

  Services hide the details of their implementation behind abstract interfaces.

- Autonomy

  Services do not require any dependencies to other services to be respected by their consumers.

- Reusability

  The abstraction of service interfaces ensures its context-independent reuse.

- Statelessness

  Consumers are not required to respect the internal state of a service offer.

Also, SOAs are not bound to a specific technical framework. One of the most common is the Simple Object Access Protocol (SOAP)-based web services framework (Booth et al., 2004). When SOA is referenced within this thesis, SOAP-based web services are addressed.

Service-Oriented Computing

SOC (Papazoglou, 2003; Singh and Huhns, 2005) is an extension to the model of Service-oriented Architectures. SOC consequently continues

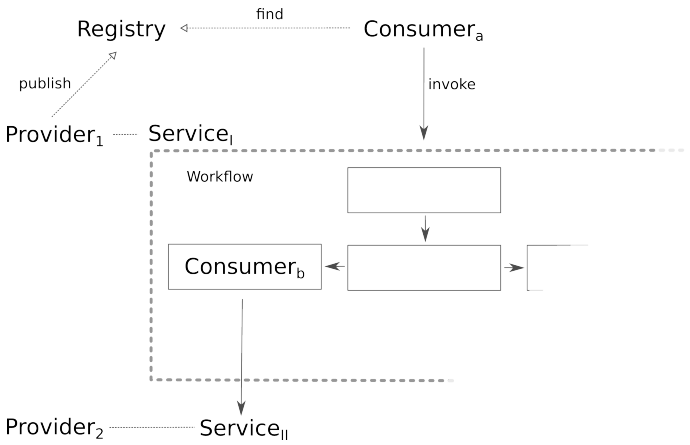


Figure 3: Service-Oriented Computing Uses Workflow Languages for Service Composition

the idea of SOA's loose coupling. The loose coupling is continued within a service component through the composition of services to provide the service functionality. To enable service composition, SOC introduces the abstraction layer of workflow languages to orchestrate existing services to composite services like the Business Process Execution Language (BPEL) (Andrews et al., 2003). Figure 3 illustrates how services can be meshed in SOC illustrating a basic service cascade, for example.

In this thesis, *service cascades* are defined as connected vertices in a graph, where each connection represents a distinct type of service usage, as illustrated in Section 1.3. They define a network of (most times incompletely) meshed services that rely on each other to provide their services.

2.4 CLOUD COMPUTING AS HOSTING MODEL

2.4.1 *NIST's Cloud Definition*

Cloud Computing "is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (Mell and Grance, 2010)

Essential Characteristics

The cloud model of National Institute of Standards and Technology, United States of America (NIST) promotes availability and is com-

posed of five essential characteristics that are listed below. (Mell and Grance, 2010)

- On-demand self-service

  Service customers are able to allocate resources for service offers without requiring human interaction with the service provider.

- Broad network access

  Service offers are available through standard mechanisms over common networks like the Internet.

- Resource pooling

  Resource allocation takes place on an abstract layer. The service consumer is not able to bind service processing to a specific resource. Instead, resources are allocated on-demand out of a resource pool to serve service requests.

- Rapid elasticity

  Resources can be allocated quickly to enable fast scale in or out of service capabilities.

- Measured service

  Resource usage is monitored, (automatically) controlled, and reported for service customers and the provider.

Service Models

Service offers based on Cloud Computing can be classified by their offered type of service. NIST provides definitions for three types of service, as listed below. (Mell and Grance, 2010)

- Cloud Infrastructure as a Service (IaaS)

  Offering hardware resources located in data centres (e.g., servers, storage, network) based on virtualisation technologies (e.g., VMware (2011b), Xen (Citrix, 2011)) is defined as IaaS in this concept. Virtualisation enables the separation of hardware resources into smaller fractions, whereby each fraction offers the same virtual hardware interfaces as an actual hardware. In this thesis, IaaS focuses on server virtualisation. Such a server fraction is called Virtual Machine (VM). Hardware resources can be allocated to VMs when demanded, depending on the features of the virtualisation technology used. In case of IaaS, the service interfaces are the virtual hardware interfaces of the VMs.

- Cloud Platform as a Service (PaaS)

  Platforms for software applications may be deployed based on an IaaS layer (Marks and Lozano, 2010). Deployed are instances of application servers combined to a cluster (cf. Section 2.4.2), in order to balance the load of service requests. In case of PaaS, the service interface is the administrative interface of the application server (cluster).

- Cloud Software as a Service (SaaS)

  Software applications may be deployed based on an IaaS or a PaaS layer (Marks and Lozano, 2010). This strategy can ease further deployment and scaling of multiple parallel instances of such applications. In case of SaaS, the service interface for consumers is the native interface provided by the deployed software applications.

Deployment Models

The definition of NIST offers four types of deployment models. They differ by the access rights for cloud usage. Private clouds are only accessible by a closed consumer group restricted to a single organisation. Community clouds are used by closed consumer groups overspanning multiple organisations. Public clouds are open to be accessed by the public. Hybrid clouds are a mixture of the previously introduced deployment models (e.g., an organisation operates a private cloud and, in addition, uses public clouds to carry peaks on demand). (Mell and Grance, 2010)

2.4.2 *Classification by Workload Model*

In addition to the definition of NIST, Cloud Computing can be classified by the processed workload. This addition refers to the origin of the Cloud Computing definition by Greg Boss et al. (2007), Weiss (2007), and Hayes (2008) as a pool of virtualised resources. In these pools, two types of workloads are distinguished. This thesis introduces a definition for such workload types, listed below.

- Parallel workloads

  Parallel workloads are defined as service requests with a relation between processing algorithm and data set to be processed as 1:n. In parallel workloads, a rather big amount of data sets have to be processed and the result of the processing of a single data set does not influence the processing of other data sets.

- Sequential workloads

  Sequential workloads are defined as service requests with a relation between processing algorithm and data set to be processed as 1:1. In sequential workloads, a single data set is processed and the result of the processing of this set may influence the processing of other data sets.

This definition can be compared to the one blogged by Harris (2008) and cited by Stanoevska-Slabeva et al. (2009).

Parallel Computing for Parallel Workloads

The processing workloads that are addressed by Parallel Computing are classified as *parallel workloads* in this thesis. Parallel workloads are long-running collections of similar computing jobs running without user interaction, also known as batch jobs. (Andrzejak et al., 2002)

In the beginning, Parallel Computing was focused on the simultaneous distribution of computing operations onto the processors of a single host computer (Almasi and Gottlieb, 1989). Hereby, large problems are divided into smaller sub-parts which can be processed simultaneously, thereby reducing the overall time to solve the problem. This basic concept of simultaneously distributing computable problems was evolved to Cluster Computing (Bader and Pennington, 2001). In Cluster Computing, the processors of multiple computers within the same administrative domain are used for simultaneous calculations of sub parts of larger problems.

To overcome administrative domain boundaries and to interconnect heterogeneous computing resources, Cluster Computing had been evolved to Grid Computing (Foster and Kesselman, 2003). Grids use the open standards of SOA to offer platform-independent interfaces for the management of parallel workloads in heterogeneous environments.

Application Clusters for Sequential Workloads

In this thesis, the processing of *sequential workloads* is performed by *application clusters*[7]. In the daily business context, sequential workloads often occur as short-running requests to SOA services. (Andrzejak et al., 2002)

Application servers (e.g., JBoss (2011), GlassFish (Oracle, 2011)) are defined as technology-specific software frameworks offered by multiple vendors that are used to provide SOA services. These services

7 In this thesis, a cluster is defined as a group of servers that each offers identical services and service requests are equally distributed among the servers.

are deployed within application servers to provide business logic to service consumers. (Natis et al., 2008)

Multiple application server instances can be joined as High-Availability (HA) or load-balancing clusters. HA clusters provide a cluster of application servers, where a primary server processes service requests and, in case of a failure, one of the redundant application servers takes over the processing of incoming requests. In contrast, load-balancing clusters provide the distribution of the overall request load to all available application servers. (Bourke, 2001)

In this thesis, application clusters are used as synonym for load-balancing application server clusters.

2.5 SERVICE LIFE CYCLE

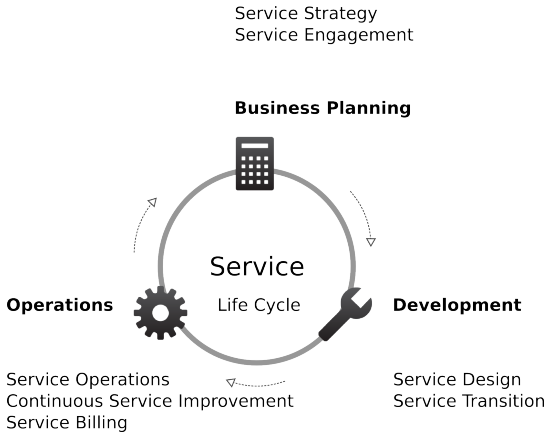


Figure 4: Generic Service Life Cycle

Service life cycles aim to represent the evolutionary steps necessary to evolve a service offer from a first business model to a mature and widely used service implementation. Thereby, the description of the evolutionary process as a cycle highlights the gap between the time of decision - often based on incomplete market information - during business planning and the time of service usage during operations, where actual market feedback is gathered. Summarised, service life cycles are feedback loops that enable continuous business improvement.

In this thesis, a basic life cycle is required as generic origin for the analysis of intersections to UC's customer-service-resource relation.

The life cycle definitions of Liang-Jie Zhang (2007) and ITIL (Beard, 2008; APM Group et al., 2011) are consolidated to a generic representation of a basic service life cycle. Resulting, the basic life cycle

contains three main cycle phases, as shown in Figure 4 and further detailed in the following listing.

Business Planning

The life cycle phase of business planning addresses the service strategy and service engagement, as defined in the following listing.

- Service strategy

  Service strategy deals with the questions of where and how service offers have to compete with other service offers. In this process, costs and risks for a targeted service portfolio have to be identified. Corresponding, for each service offer it has to be determined how value can be created. This includes decisions about the offered quality levels and expected consumer behaviour in terms of service usage.

- Service engagement

  Service engagement addresses make-or-buy decisions for supportive services within a service cascade. Service offers from other, maybe competing, service providers can be used to backup or load-balance proprietary service offers. Or external service offers can be used during service composition for feature completion.

Development

In the life cycle phase of development, the design and realisation of a service's implementation are in focus. In addition, this cycle phase includes the management of the transition from a completed service implementation to a deployed and running service instance. Both process steps are defined corresponding to the following listing.

- Service design

  Service design deals with the design of a software architecture to enable the realisation of targeted service features. This architecture is then transferred into a corresponding service implementation. The design, transfer, and implementation process considers the requirements on information security, service levels, service continuity, service availability and provision capacity.

- Service transition

  Service transition subsumes all tasks which ensure that subsequent service operations reach the targeted service quality. This

includes the tasks of service validation and testing, inclusion in knowledge, asset and change management, and deployment of service releases.

Operations

The life cycle phase of service operations is focused on economically efficient and technically effective service delivery. Incorporated into this phase of the service life cycle is the support for consumers of service offers. The management of the continuous improvement of the overall service life cycle is also included. In addition, the management of service billing is part of service operations, as defined in the listing below.

- Service operations

  The subject of operations includes a range of activities to ensure the technical accessibility and availability of service offers. This includes a service desk to support consumers when experiencing problems during service usage. Traditionally, operations is concerned with the technical management of all physical resources (e.g., network, servers). The operations management addresses the daily business activities to operate the IT environment like backups or updates. Service operations also includes application management. It is focused on the support of the service life cycle with knowledge about design, implementation, and maintenance of service implementations.

- Continuous service improvement

  The continuous service improvement addresses the overall service life cycle. As the name suggests, this step targets the ongoing improvement and rising maturity of service offers and business processes of the service provider. The step's effectiveness is dependent on a reliable service measurement and reporting that continuously monitors and manages the targeted service quality.

- Service billing

  Service billing includes billing-related activities reaching from accounting of service usage to the corresponding invoicing of consumed efforts.

  In this thesis, the consideration of service billing is simplified, as described in Section 2.2.

2.6 SUMMARISING THE RESEARCH BACKGROUND

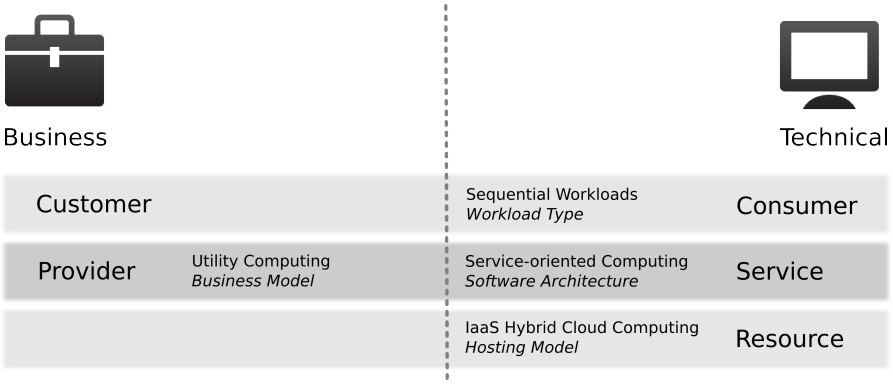


Figure 5: Setting of the Scenery for Service Life Cycles in This Thesis

This thesis is concerned with the mapping of the customer-service-resource relation in a generic life cycle for UC service offers in SOC architectures hosted on IaaS (cf. Section 1.2). This thesis examines both - technical and business - concerns of service providers regarding their service offers.

From a business point of view, services are offered by providers that choose the business model of Utility Computing (cf. Section 2.2). Complementary, customers willing to consume the offered kind of services exist. Customers and providers want to contract on usage-centred service pricing, billing constraints, contract penalties, and processing quality (cf. Section 2.2). It is assumed that there are multiple groups of customers classified by their behaviour in service usage, pricing expectations, and requirements on processing quality.

From a technical point of view, customers are consumers of service offers. In this thesis, services are considered as part of a SOC architecture (cf. Section 2.3). This includes the possibility that services are build dependent on other services and also that other services rely on the provided services. In business applications, mainly services are used which process sequential workloads (cf. Section 2.4.2), as shown in an example service cascade by Bodendorf and Schobert (2003) including, among others, service offers for route planning, car rental, and claims notification.

The so far described service offers are deployed on an IaaS hybrid Cloud Computing environment and are offered as public SaaS (cf. Section 2.4). The IaaS environment may extend over multiple data centres. Beside provider-owned private data centres, some of the demanded resources may be integrated using public IaaS from third-party suppli-

ers. The resources provided by the IaaS may be used to provide more than one public or private[8] SaaS.

The previous text introduces UC service offers in SOC architectures hosted on IaaS as subject-matter of service providers. In this thesis, the term *service* is used as a synonym for this subject-matter.

Service providers create and offer services following a life cycle that describes the process of service creation in terms of business planning and development and service offering in terms of operations. From provider to provider, life cycles can considerably vary in their level of detail in which individual process steps are implemented. In this thesis, the chosen detail level of life cycle description is high (cf. Section 2.5) to generate a generic view of life cycles that may support most common use cases. Figure 4 depicts the resulting generic life cycle for this thesis. The implications of the provision of UC service offers in SOC architectures hosted on IaaS for this generic life cycle are analysed in Chapter 4.

8 Most service offers within a service cascade are not targeted on public use. They are only offered as private supportive services (e.g., file compression or checksum calculation), may be invoked out of several concurrent service cascades.

3 RELATED WORK

This chapter introduces related work and outlines why the results are not sufficient for the problem statement in this thesis. The chapter is divided into two parts. The first part 3.1 introduces core provision models for utility computing platforms. In the second part 3.2, concepts for the usage-centred assurance of service quality are presented.

3.1 PROVISION MODELS FOR UTILITY COMPUTING PLATFORMS

 Component Model  Workflow Model

In this chapter, three types of provision models are presented. The types are distinguished by their primary research domain. For each presented model, it is outlined what the major differences are compared to the evolved component (cf. Section 5.2.2) and workflow model (cf. Section 5.2.3) in this thesis.

3.1.1 *Utility Computing/Cloud Models*

Buyya's InterCloud - Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services

Buyya et al. introduce an approach to a service-oriented architectural framework consisting of clients brokering and coordinator services that support application scheduling, resource allocation, and migration of workloads, shown in Figure 6.

Buyya's Cloud Exchange (CEx) brings together service producers and consumers by acting as a market maker. The CEx aggregates infrastructure demands reported by the application brokers and compares these demands against the available resources published by the Cloud Coordinators. As part of the comparison process, the CEx supports competitive economic models - such as commodity markets and auctions - within the trading process of Cloud services. The use of CEx builds markets that are enabled to trade based on SLAs. In this case, an SLA specifies metrics, agreed upon by all parties, that address details of the service to be provided. To enable these markets, clients of federated platforms need to instantiate a Cloud Brokering service to dynamically establish service contracts with Cloud Coordinators

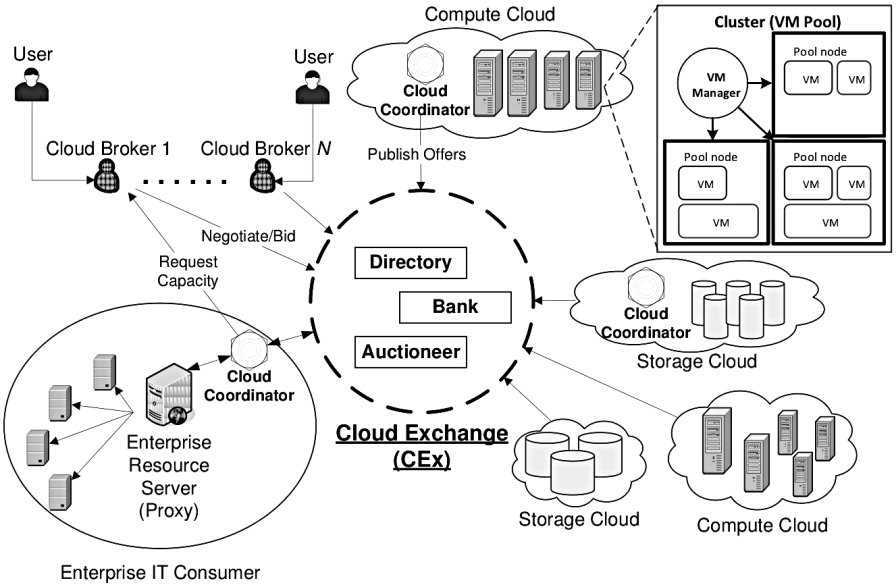


Figure 6: Buyya's Federated Network of Clouds Mediated by a Cloud Exchange (Buyya et al., 2010)

via the trading functions provided by the Cloud Exchange. (Buyya et al., 2010)

The approach of Buyya et al. represents a well elaborated IT architectural point of view on Cloud Computing. As open research questions that lead beyond the InterCloud approach, Buyya et al. list two issues. Buyya states that the information of SLAs should also be used by the automated cloud management to raise the energy efficiency of data centres. In addition, Buyya also states that in the future a solution is needed to allow "adaptive system management by establishing useful relationships between high-level performance targets (specified by operators) and low-level control parameters and observables that system components can control or monitor" (Buyya et al., 2010). This thesis offers an approach (Heckmann et al., 2012b) to these open questions, that is provided through the service broker (cf. Section 5.2.2) and its underlying data model (cf. Section 5.4).

In comparison, this thesis also offers a more detailed look at technical workflows (cf. Section 5.2.3) and functional components (cf. Section 5.2.2) for delivery frameworks.

Kertesz's SLA-Based Resource Virtualisation Architecture

Kertesz et al. present a unified service architecture that builds on the areas: agreement negotiation, brokering and virtualised service deployment. The suggested architecture is shown in Figure 7 (MN: meta-negotiator, MB: meta-broker, B: broker, ASD: automatic service

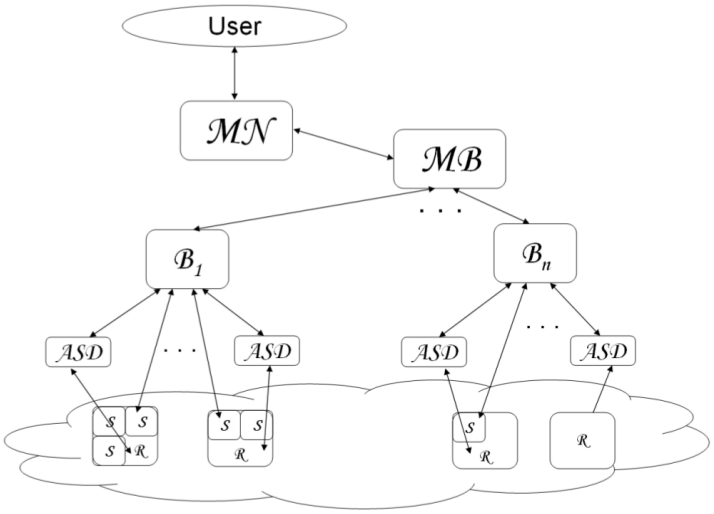


Figure 7: Kertesz's SLA-Based Resource Virtualisation Architecture (Kertesz et al., 2009)

deployment, S: service, and R: resource) and introduces an additional meta layer for negotiations of SLA.

In opposite to this thesis, the approach of Kertesz et al. supposes that service providers and service consumers meet on demand and usually do not know about negotiation protocols, document languages or infrastructure requirements of potential partners. (Kertesz et al., 2009)

In Chapter 4 this thesis defines its focus on provider-consumer-relations that involve non-automatic contracting before service usage.

Liu's Architecture for Green Data Centres

Liu et al. introduce an approach to consolidate workload to provide energy savings in cloud computing environments while guaranteeing real-time performance for performance-sensitive applications. The proposed architecture is shown in Figure 8. (Liu et al., 2009)

In opposite to this thesis, the approach of Liu et al. does not support pay-per-use service cascades nor enable the management of different service lines (cf. Section 5.3.5).

Marks & Lozano's Cloud Computing Technical Reference Architecture

The Cloud Computing Technical Reference Architecture is a minor part of the holistic Cloud Computing Reference Model of Marks and Lozano (2010, p. 188). The holistic model offers an extended logical cloud computing architecture that intents to broaden the tiers and logical layers of cloud computing. This layered logical cloud comput-

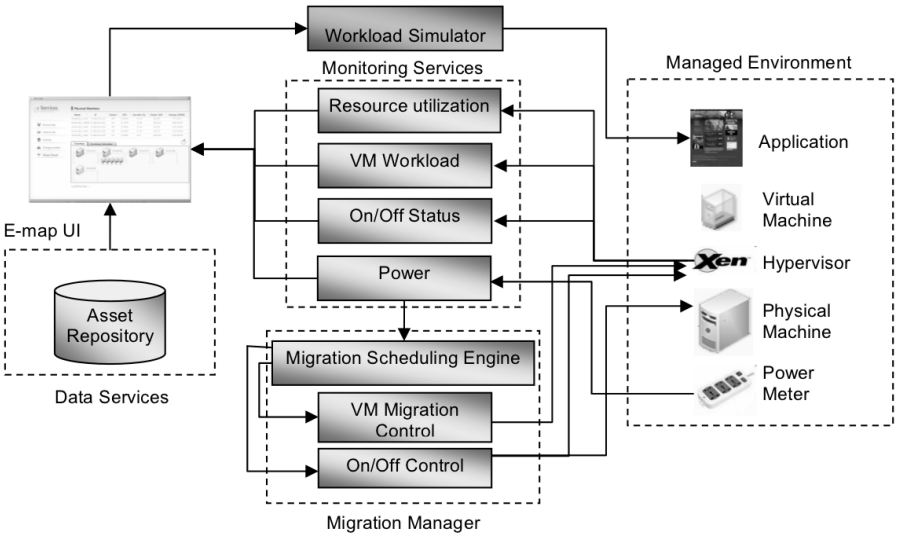


Figure 8: Liu's Architecture for Green Data Centres (Liu et al., 2009)

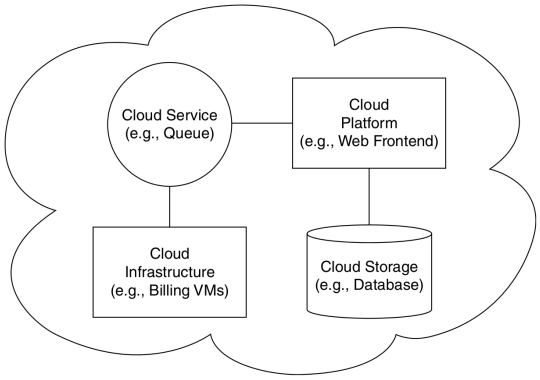


Figure 9: Marks & Lozano's Cloud Computing Technical Reference Architecture (Marks and Lozano, 2010)

ing model consists of layers of IT capabilities that must be virtualised in order to realise a cloud computing architecture. The corresponding Cloud Computing Technical Reference Architecture is introduced in Figure 9.

The technical reference architecture provided by Marks and Lozano demonstrates a basic[1] component architecture. The core provision model in this thesis offers more detailed information about component relations and considers more UC-specific functionalities.

1 as intended by the authors that focus on their much more detailed holistic Cloud Computing Reference Model

Mendoza's Software Application Service Framework

Mendoza defines the properties of Software Utility Applications (SUAs) (Mendoza, 2007, p. 107) as an extension to the definition of SaaS. The primary advantages of SUAs are multitenancy, maintainability, and measures to manage service quality (e.g., SLA, scalability, virtualisation).

Based on the definition of the Software Utility Applications Mendoza evolves a Software Application Service Framework (sasf) (Mendoza, 2007, p. 129).

Figure 10 shows how the sasf relates to the environment of a utility computing data centre. (Mendoza, 2007, p. 143)

> "The sasf provider will manage, maintain, and create SLAs for other service providers that will use some of the sasf-provided services. The sasf will incorporate Web services management capabilities, which will form the basis for managing Web services within the framework. [...] several SaaS (or SUA) applications use services offered by the sasf, such as metering, billing, and identity management. The figure also shows another group of SaaS applications offering Web services to other applications using the Web services gateway, which provides protection from malicious Internet attacks in the form of denial of service and unauthorized access. The sasf functions like any other software application. It uses all layers of the utility computing environment starting with the infrastructure services." (Mendoza, 2007, p. 143)

The approach of Mendoza represents a well elaborated technical point of view on service delivery laid out for software architectures based on SOAP[2]. Compared to the results of this thesis, providers need to deploy services that make active use of the sasf interfaces. The component and workflow model (cf. Section 5.2) of this thesis is defined to transparently offer UC features to services that are not explicitly implemented as UC services. In addition, load-balancing is considered as a major model property in this thesis.

Phan & Li's Vertical Load Distribution via Multiple Implementation Options

Phan and Li examine enterprise cloud computing, where enterprises may use an SOA to publish a streamlined interface of their business processes.

2 The SOAP (W3C, 2007) is one of many protocol families to implement a web service architecture (Booth et al., 2004).

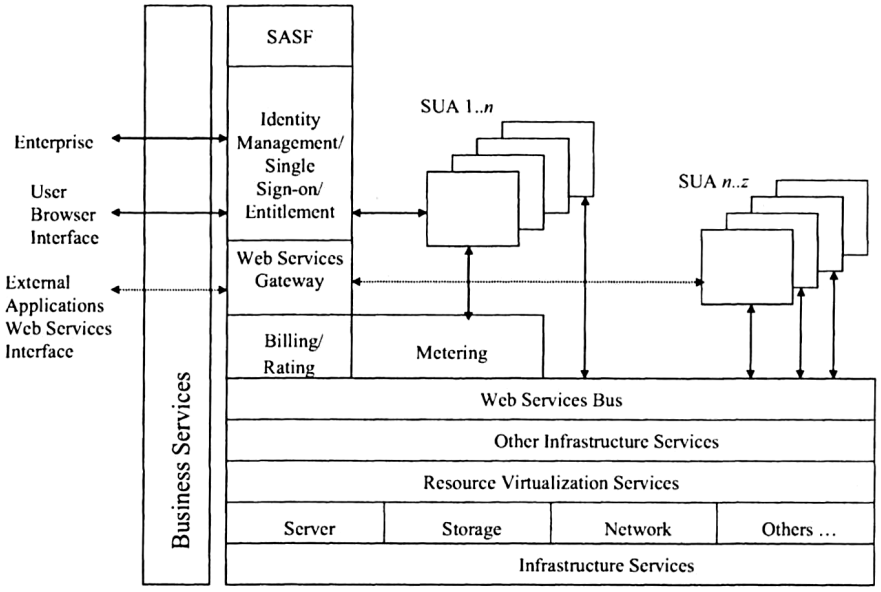


Figure 10: Mendoza's Software Application Service Framework (Mendoza, 2007, p. 143)

> "In order to scale up the number of business processes, each tier in the provider's architecture usually deploys multiple servers for load distribution and fault tolerance. Such load distribution across multiple servers within the same tier can be viewed as horizontal load distribution. One limitation of this approach is that load cannot be distributed further when all servers in the same tier are fully loaded. Another approach to providing resiliency and scalability is to have multiple implementation options that give opportunities for vertical load distribution across tiers. [Phan and Li] described in detail a request routing framework for SOA-based enterprise cloud computing that takes into account both these options for horizontal and vertical load distribution." (Phan and Li, 2010)

This approach - as shown in Figure 11 - is similar to the service line approach of this thesis (cf. Section 5.3.5). This thesis exceeds Phan and Li's approach in the abilities of its scheduler/service broker, which also takes economic aspects into account.

Sotomayor's Virtual Infrastructure Management in Clouds

Sotomayor et al. have developed a lease management approach to cloud resource schedulers. The implementation acts as a VM scheduler for a cloud management software or can be used on its own as a

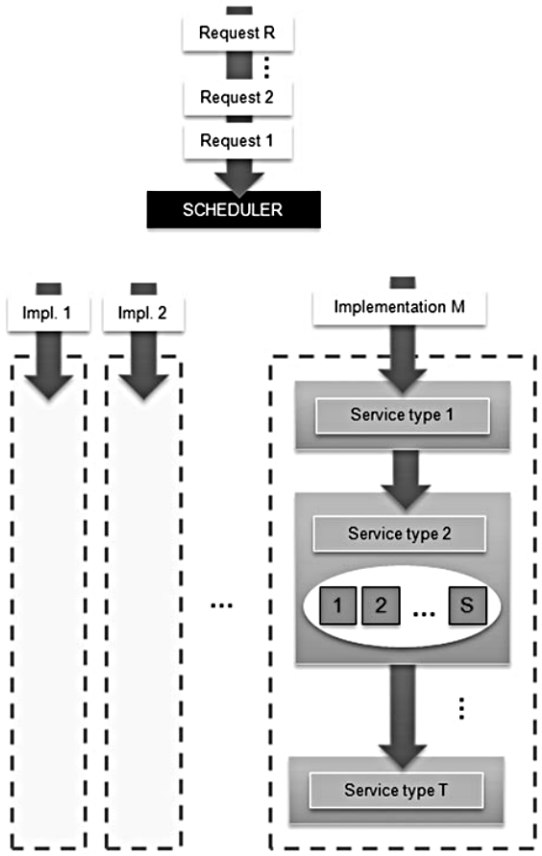


Figure 11: Phan & Li's Vertical Load Distribution via Multiple Implementation Options (Phan and Li, 2010)

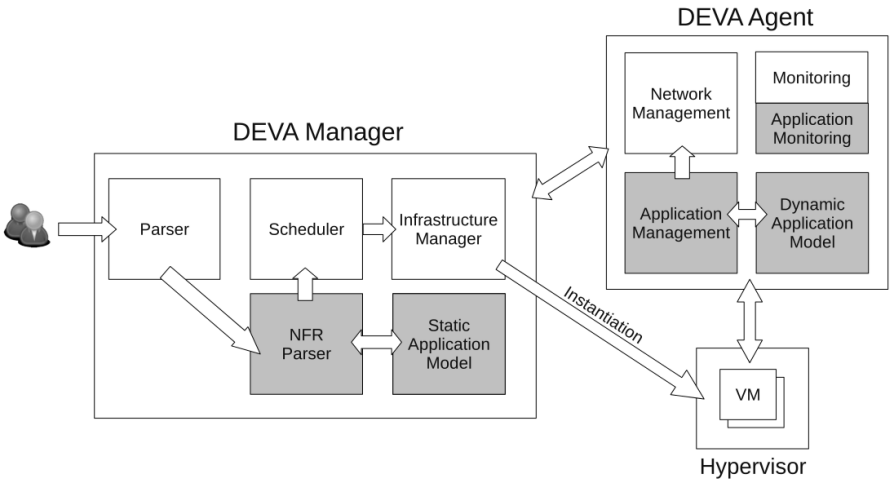


Figure 12: Villegas & Sadjadi's Architecture for the Mapping of Non-Functional Requirements (Villegas and Sadjadi, 2011)

simulator to evaluate the performance of different scheduling strategies over time. The approach supports advance reservation "leases in which resources must be available at a specific time; best-effort leases, in which resources are provisioned as soon as possible, and requests are placed in a queue, if necessary; and immediate leases, in which resources are provisioned when requested or not at all." (Sotomayor et al., 2009)

The approach of Sotomayor et al. could be classified as a functional subset of the service broker and load-balancer approach (cf. Section 5.2.2) in this thesis. The separation of service broker and load-balancer in this thesis offers additional economic scheduling of service requests and service request routing including billing. Due to its focus on sequential workloads, this thesis's component model does not deal with advance reservation of resources.

Villegas & Sadjadi's Mapping of Non-Functional Requirements to Cloud Applications

Villegas and Sadjadi evolve the design and implementation of an IaaS cloud manager in such a way that non-functional requirements determined during the requirements analysis phase can be mapped as properties for a group of virtual appliances running the application. The approach - shown in Figure 12 - aims to ensure that the expected quality of service is maintained during execution and that it can be considered during different development phases. (Villegas and Sadjadi, 2011)

The approach of Villegas and Sadjadi is focused on the life cycle phases of development and operations, disregarding business planning. The approach does also not consider fault-tolerance, execution cost or performance parameters such as processor, memory, network and disk usage either. This thesis exceeds Villegas and Sadjadi approach in these points (cf. Section 2.5, 5.2).

Zhang's Service Delivery Platform

Zhang et al. in detail describe their approach to deliver application clusters for SOAP-based SOAs. One of their contributions is a brief definition of a service delivery platform for SOAP services. As shown in Figure 13, the platform consists of six horizontal layers and two vertical layers. The Core Infrastructure Services layer offers extensive services for managing physical IT resources including supporting software (e.g., server, storage, network, operating system, database management system). The IT Service Management layer is based on the Core Infrastructure Services to help manage the IT infrastructure efficiently. (Liang-Jie Zhang, 2007, p. 312)

> "The Horizontal Services layer supports common IT services like Web application services, calendar services, collaboration services, etc. It also supports common business services including human resources services, logistic services, etc. The Vertical Business Services layer organizes and maintains applications that can be used to implement a specific business process or a solution for a specific industry. Some sample vertical services are loan services for the banking industry and claim services for the insurance industry." (Liang-Jie Zhang, 2007, p. 313)

The Services Partnership Manager is a function module to manage the relationships of the available service assets. The Value Added Services layer organises and manages service integration based on horizontal business services and vertical industry applications by leveraging the Services Partnership Manager. It provides services that are customised to a specific customer's need. The Service Membership Management layer is responsible for the enablement of portal access, business entity on-boarding, service provisioning and subscription. The Service Lifecycle Management layer is responsible for monitoring, metering, billing, and exception handling. (Liang-Jie Zhang, 2007, p. 313)

The approach of Zhang et al. represents a well elaborated business management point of view on service delivery laid out for SOAP-enabled software architectures. As open research questions, Zhang et

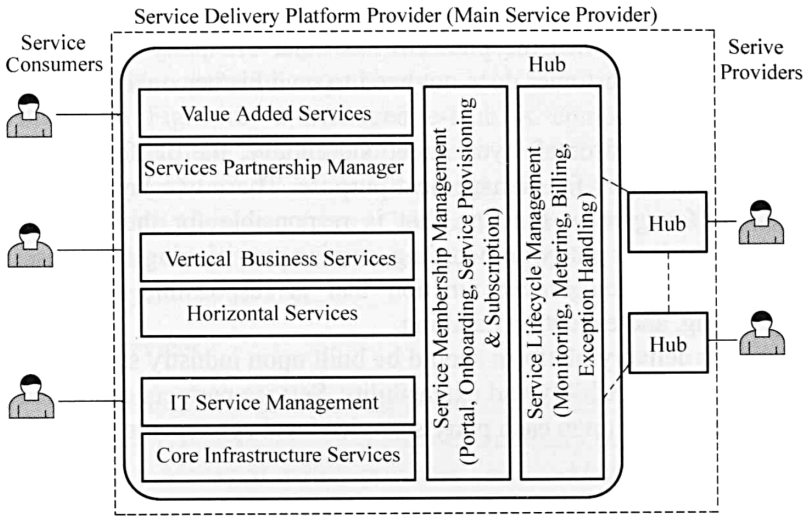


Figure 13: Zhang's Layered View of a Service Delivery Platform (Liang-Jie Zhang, 2007, p. 313)

al. indicate that "a service delivery platform should not only fulfil functional requirements but also non-functional requirements - different Service Level Agreements" (Liang-Jie Zhang, 2007, p. 328).

In comparison, this thesis offers a more detailed look at technical workflows and architecture. As a key feature, this thesis includes an specific approach to SLA.

3.1.2 *Grid Models*

Foster's Open Grid Service Architecture

The Open Grid Service Architecture (OGSA) (Foster et al., 2005) is introduced as approach to define an architecture for Grid[3] infrastructures based on the SOA paradigm. The main idea of OGSA is the exposition of Grid management interfaces as services for SOAs. Figure 14 presents a non-exhaustive overview of the OGSA framework capabilities.

While the framework itself does not relate to this thesis, the specification of OGSA is based on a detailed study of collected use cases (Kishimoto, 2003; Foster et al., 2004; Von Reich, 2004; MacLaren et al., 2006). These use cases are not based on a formal requirements analysis, but they have been accepted by the Grid community as basis for the OGSA standardisation. Altogether, they represent a wide spectrum of scenarios describing business and scientific use of distributed com-

3 Grids are defined as highly distributed infrastructures for parallel computing workloads (cf. Section 2.4.2). (Foster and Kesselman, 2003)

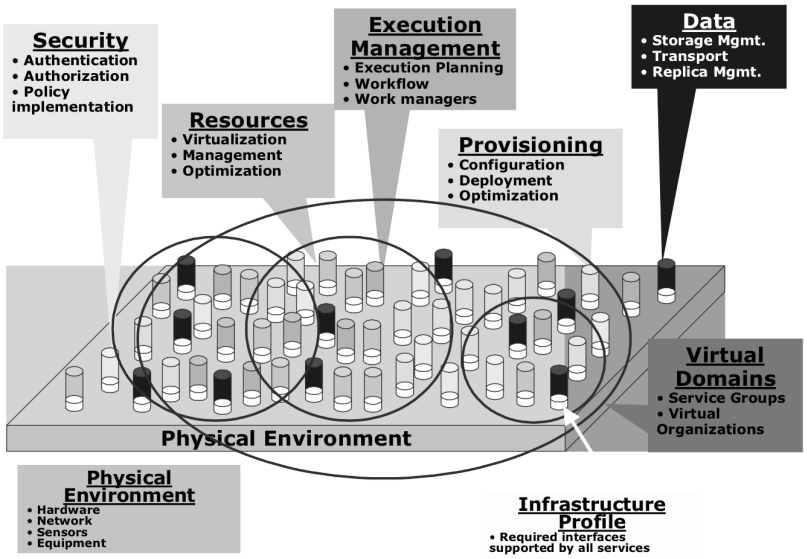


Figure 14: Foster's Open Grid Service Architecture Framework (Foster et al., 2005)

puting applications, not restricted to parallel computing workloads only. Section 4.4.1 analyses this use cases and extracts functional requirements for the development of a core provision model for UC service offers.

GRASP – An Architecture Enabling Grid-Based Application Service Provision

The Grid-based Application Service Provision (GRASP) project introduces a software architecture for Application Service Provision (ASP)[4] business models focused on Grid applications, as shown in Figure 15. The developed GRASP middleware is designed to provide a high level of scalability, reliability and security, advanced accounting functionality, quality of service, and resource management. (Dimitrakos et al., 2002, 2003; Wesner et al., 2004)

Due to the similarities between UC and ASP , GRASP offers a set of reusable requirement definitions for this thesis. Other aspects of the GRASP approach are not reused, as the projects results are limited to parallel computing workloads.

4 ASP can be seen as an predecessor of SaaS. It defines a 1:n relationship between a service provider and its customers. Contracts between both parties are long running and bounded to static pricing models, often flat rates. Like UC, ASP defines a business model. (Smith and Kumar, 2004; Kim and Paek, 2005)

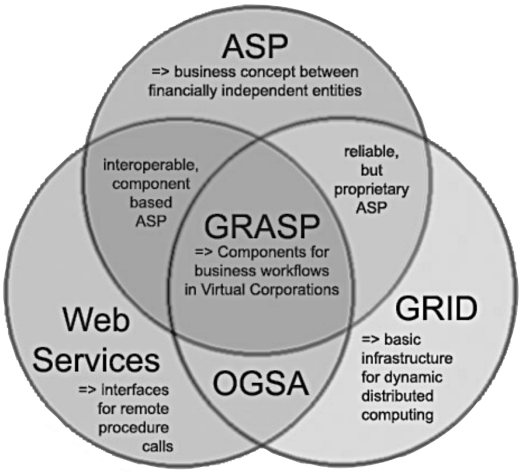


Figure 15: Underlying Concepts of GRASP (Dimitrakos et al., 2002)

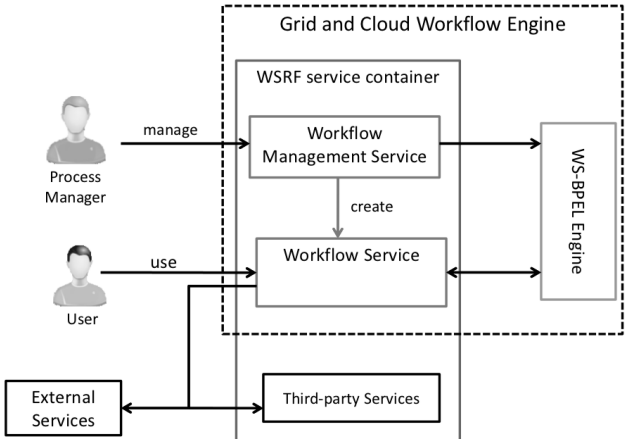


Figure 16: Höing's Architecture for Secure Workflow Orchestration for Cloud and Grid Services (Hoeing, 2010)

Höing's Architecture for Secure Workflow Orchestration for Cloud and Grid Services

Höing presents an orchestration architecture that facilitates the integration of stateful and stateless services across organisational boundaries. The approach aims to combine Web, Grid, and Cloud services in a single workflow. (Hoeing, 2010)

> "The architecture is based upon process modelling and execution technologies originating from the business domain. It extends a standard fully WS-BPEL-compliant workflow engine by services that provide additional functions during the actual workflow execution." (Hoeing, 2010)

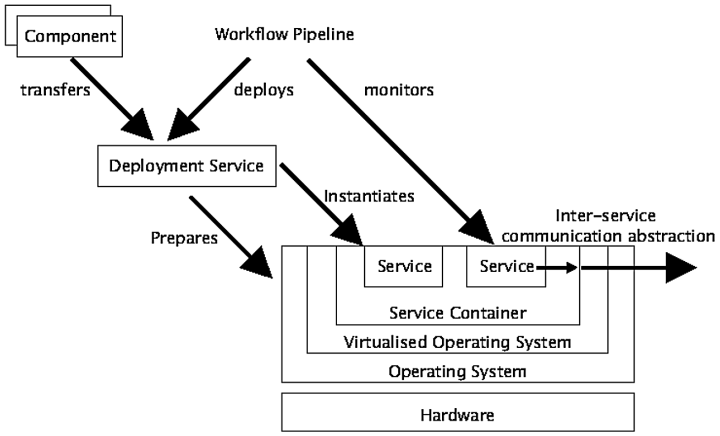


Figure 17: Architecture of the ICENI II Execution Environment (McGough et al., 2006)

Höing's approach offers a completely different solution architecture compared to this thesis (cf. Section 5.2.3), although both start in kin technical perspectives regarding SOA and the aspects of Cloud Computing addressed by Höing.

ICENI - Imperial College e-Science Networked Infrastructure

The ICENI project introduces a concept for high-level abstraction of scientific computing, as shown in Figure 17. To simplify the utilisation of Grid infrastructures, the project offers a framework consisting of three components: a graphical composition tool, distributed component repositories, and federating Grid middleware. ICENI also addresses pay-per-use features, as demanded by UC business models. (Cohen et al., 2006; Darlington et al., 2006; McGough et al., 2006; Cohen et al., 2008)

The presented approach is determined to process parallel computations. ICENI clearly defines itself as a distributed computation framework, it also touches interesting aspects with its pay-per-use approach to composite components/applications.

3.1.3 *Application Cluster Models*

Arsanjani's Service-Oriented Reference Architecture S3

Arsanjani et al. have evolved an SOA reference architecture based on the evaluation of SOA projects conducted at IBM from 2003 to 2006. As reference architecture Arsanjani et al. introduced a model of logical layers, where each layer represents a business value perspective, as shown in Figure 18. (Arsanjani et al., 2007)

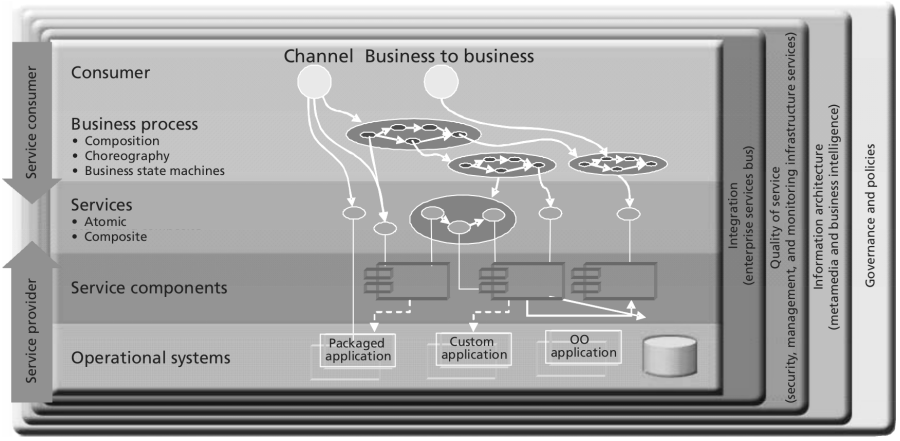


Figure 18: Arsanjani's Service-Oriented Reference Architecture S3 (Arsanjani et al., 2007)

The approach of Arsanjani et al. is a good example for the requirements of service cascades in SOA. The presented reference architecture, compared to this thesis, is rather abstract referred to its level of detail (cf. Section 5.3.5). Also, it does not address service billing.

Urgaonkar's Analytical Model for Multi-Tier Internet Services and its Applications

Urgaonkar et al. introduce an analytical model that should be able to represent the behaviour of multi-tier Internet applications (Urgaonkar et al., 2005a). The model is based on a network of queues, where each queue represents one of the tiers within a multi-tier architecture. The approach can be used to simulate session-based workloads, concurrency limits and caching at intermediate tiers.

In addition, Urgaonkar et al. have evolved a hosting platform architecture for multi-tier Internet services (Urgaonkar et al., 2005b) based on the previously introduced analytical model. The hosting platform architecture is shown in Figure 19 (Sentry: binding of sessions to an application's server pool including SLA-based flow control, Capsule: component of an application, Nucleus: component for Capsule's performance monitoring, Control Plane: dynamic provisioning of servers to individual applications).

With the presented approach Urgaonkar et al. address simple UC scenarios. The approach does not take into account providers with multiple data centres, service billing or service cascades.

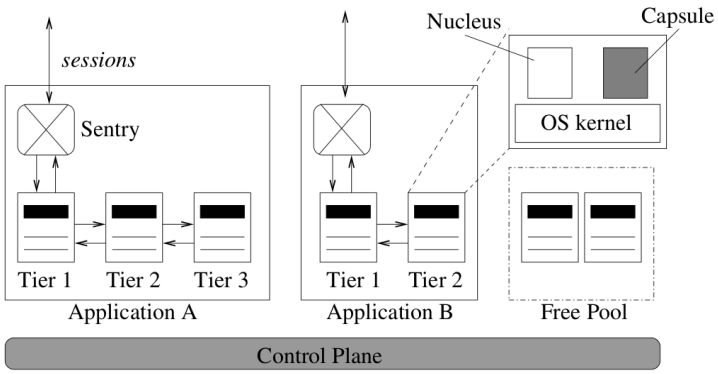


Figure 19: Urgaonkar's Hosting Platform Architecture (Urgaonkar et al., 2005b)

Older Models

Ninja (Gribble et al., 2001; von Behren et al., 2002; Welsh and Culler, 2003) is a project that evolved an approach to application server clusters. The approach enhances the idea of application clusters and introduces features to host distributed and composed services.

Océano (Appleby et al., 2001; Fong et al., 2002; Pazel et al., 2002) introduced an approach to dynamic load-balancing in physical server clusters for multi-customer hosting. The approach focuses on the handling of individual SLA per customer.

3.2 USAGE-CENTRED ASSURANCE OF SERVICE QUALITY

The specification of quality of service offers can be done on various abstraction levels. This thesis aims to assure service quality on all OSI layers, herein defined as horizontal layers. With its data model, Business Service Level Agreement (BSLA), usage pattern, and a decision tree (cf. Section 5), this thesis discusses and offers a vertical approach to service quality assurance. Related work discussed in this chapter reach from the categorisation of service quality to the control of service quality in related fields of computer science.

3.2.1 *Categorisation of Quality of Service, Experience, and Business*

A categorisation for quality metrics for IT services is introduced by Van Moorsel (2001); Machiraju et al. (2002). The approach is based on the assumption that technical metrics, such as availability, are not sufficient to evaluate service quality. As enhancements, Van Moorsel et

al. propose additional metrics that are based on user experience and business measures. All quality metrics are distinguished into three categories, as listed below.

- Quality of Service

  The Quality of Service (QoS) category is defined from the point of view of service providers. QoS involves all technical metrics to ensure that resource usage complies to given technical thresholds. For example, this includes metrics concerning (provider) availability over time or performance indicators like Central Processing Unit (CPU) load. In addition, Van Moorsel et al. also distinguish technical metrics monitored at operating system layer (e.g., memory capacity or availability) and process layer (e.g., memory usage or availability).

  The motivation for Van Moorsel's et al. sub-classification of operating system layer and process layer is not clear. The given examples seem to be rather arbitrary and trying to avoid intersections of metrics, while intersections might be useful. Given, intersections are useful, sub-classification might not be.

- Quality of Experience

  The Quality of Experience (QoE) category is defined from the point of view of service consumers. Like QoS, QoE is concerned with the monitoring of technical metrics to track given technical thresholds in order to ensure that service usage complies to certain quality demands. For example, this includes metrics concerning (consumer) availability over time or average response time. In addition, QoE also includes usability metrics.

  Van Moorsel's et al. inclusion of usability metrics seems inconsistent. Moorsel's et al. do not offer any advice how these usability metrics should be monitored.

- Quality of Business

  The Quality of Business (QoBiz) category is defined from the point of view of service providers. QoBiz metrics are focused on economic thresholds like request price, resource costs, or request failures.

  Van Moorsel's et al. classification of request failures as QoBiz seems inconsistent. As a technical measure, it should be classified as QoS.

Figure 20 illustrates the relations between QoS, QoE, and QoBiz from the point of view of a service provider. Service providers expose their

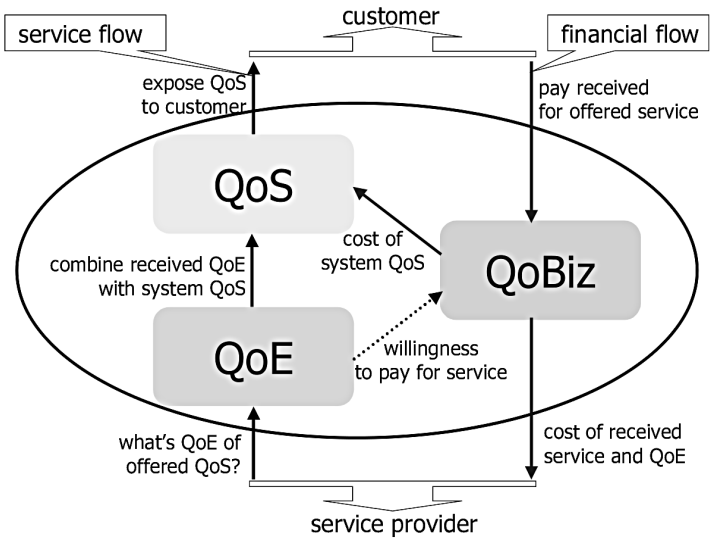


Figure 20: QoS, QoE, and QoBiz from the Point of View of a Service Provider (Van Moorsel, 2001)

QoS metrics to service consumers, called customers in Figure 20. Unless consumers pay for service consumption, they propagate their QoE metrics to the service provider. QoBiz relies on the gathered costs and cash flow metrics to calculate its economic metrics. QoS and QoE are analysed to adapt future QoS offers.

This thesis considers the given categories for quality metrics for services. In Section 4.3.1, this thesis improves Van Moorsel's et al. categorisation and offers a detailed definition of service quality in the work's context. The work does not cover aspects of QoS/SLA ontologies (Dobson and Sanchez-Macian, 2006).

3.2.2 *Quality Assurance in Cloud Computing Environments*

This research offers an approach (cf. Section 5) to technically converge the quality-related ontologies of service, experience, and business, as introduced by Moorsel et al. (cf. Section 3.2.1). None of the further discussed related work offers such a convergence approach to QoS, QoE, and QoBiz. The following chapter introduces approaches for quality assurance in addition to the models presented in Section 3.1.

Quality Assurance in Cloud Computing

Nathuji et al. (2010) introduce an approach to a cloud system that is designed to provide quality assurances for application performance deployed in virtualised machine pools. The quality control should be gained by the use of a multi-input multi-output model that captures

interference effects to drive a closed loop resource management controller. The controller meets specified performance levels for each VM by interpreting application feedback. Applications can specify levels of QoS, paying more for higher QoS levels. Nathuji et al. predict that this approach will lead to an overall improved cloud efficiency and utilisation. (Nathuji et al., 2010)

Stantchev and Schroepfer (2009) present a three step approach to SLA assurance in business processes hosted in cloud environments. The first step of their approach is the formalisation of the business process requirements, on the one hand, and of the service capabilities on the IT infrastructure side, on the other hand. In their approach, Stantchev and Schröpfer use a predefined service level objective structure and predefined non-functional property terms. The second step is the negotiation of IT infrastructure capabilities. Therefore, different load hypotheses and the performance metrics of the individual technical services are used in order to find a fitting resource for service deployment. In the last step of the approach, at runtime the SLA is controlled on the IT infrastructure level using transparent parallelisation based on multiple service instances. This replication ensures service levels regarding response time, transaction rate, throughput and availability, respectively reliability. (Stantchev and Schroepfer, 2009)

Both evolved models, the one evolved by Nathuji et al. and the model of Stantchev and Schröpfer, can be compared to the Service Broker approach (cf. Section 5.2.2) in this thesis. The work of Nathuji et al. more consistently works out a decision model for its resource management controller. Stantchev and Schröpfer work offers a more detailed definition for performance metrics of individual technical services. In comparison to both approaches, the benefit of the Service Broker is its embedding in the overall Core Provision Model (cf. Section 5.2) introduced within this thesis.

Quality Assurance in Grids

In Grid Computing large amounts of data have to be distributed over several computing systems so that parallel calculations on the data slices accelerate the overall processing of the data compared to the processing on a single computing system. To control the distribution, processing, and result consolidation, most grid architectures provide an architectural component called *broker*. The broker sometimes also offers billing or marketplace features like auctions and bidding. Different approaches for these brokers are known to accelerate processing of multiple parallel and/or sequential workloads[5], for example

5 Logical group of calculations, also referred to as *job*.

using simple resource reservation or considering the problem as a queueing system.

Afzal et al. (2008) propose a scheduling algorithm for Grid Computing frameworks. This approach claims to minimise the execution costs of workflows while ensuring that their associated QoS constraints are satisfied. The algorithm handles the resources as a queueing system, seamlessly routing the workflows through the resource network. The approach promises to not require performance prediction nor negotiation of advance reservations for every stage of the workflow. The predictions should not only be available for tested applications, but also for future workloads of new applications. The algorithm aims to guarantee QoS within required confidence bounds for the end-to-end execution of workflows. (Afzal et al., 2008)

In comparison to this thesis, the approach of Afzal et al. and other grid broker approaches (cf. Section 3.1.2) does not sufficiently cover the relation between resources, services, and consumers introduced in this thesis (cf. Section 4.2).

Quality of Service in Web Service Middleware

On the application layer, web service orchestration, as introduced in Section 2.3, deals with the challenge of quality control in multi-tier service cascades.

Zeng et al. (2004) present an approach to a QoS-enabled middleware supporting quality driven web service compositions. The approach comprehends a service quality model to evaluate the overall quality of (composite) web services. In addition, the approach offers two service selection approaches for composite service execution. Zeng et al. have implemented a platform based on their approach that supports the definition of service ontologies and the specification of composite services using statecharts. (Zeng et al., 2004)

In comparison to this thesis, the introduced QoS approach of Zeng et al. is not able to assure service quality per consumer in meshed service cascades. It only offers a global QoS driven service selection approach that is able to control the average service quality of a service cascade.

Quality of Service in Networks

Examining approaches in the field of computer networks, the understanding of QoS is restricted to control attempts on the network layer itself. Typical examples for this perspective are the works of Shigang and Nahrstedt (1998), Wolski et al. (1999), or Xipeng and Ni (1999). As an exception to the network focus, the project MDCSim (Lim et al.,

2009) offers an approach to a multi-tier data centre simulation, but focuses its outcomes onto the comparison of Infiniband and 10 Gigabit Ethernet network technologies.

Quality in the Context of Green IT

In Green IT, the focus lies on the reduction of power consumption in data centres. Green IT attempts to achieve this goal through the consolidation of hardware servers. While optimising the number of hardware servers, Green IT has to assure service quality to process given workloads.

Approaches for load prediction of servers in a single data centre are shown by Speitkamp and Bichler (2010) using historical data analysis. Bi et al. (2010) perform load prediction using a non-linear optimisation model. Another approach for single data centres is shown by Kusic et al. (2008) based on a limited lookahead control framework. Wang introduces an approach (Wang and Wang, 2010) to combine server consolidation and dynamic voltage and frequency scaling. Freitas et al. (2010) show an approach to service level management in distributed infrastructures, including QoS translation and support for self-adaptation. Load balancing on the level of data centres within and between client devices is introduced by Peoples et al. (2011).

3.3 SUMMARISING THE RELATED WORK

Related work divides into the two main sections, one addressing *provision models* in Section 3.1 and the other containing related approaches for *usage-centred assurance of service quality* in Section 3.2.

Section 3.1 introduces related provision models from the fields of research in Utility Computing and Cloud Computing. In addition, models from Grid Computing and Application Clustering are presented.

The models from the field of Utility Computing and Cloud Computing offer well elaborated business management or IT architectural points of view on Cloud Computing. The open research questions raised in the publications of the authors address the detailed exploration of the non-functional information of SLAs as one alternative to establish a useful relationships between high-level performance targets and low-level infrastructure metrics. Other main aspects related work does not address are the modelling of pay-per-use service cascades, the management of different service lines, or support for economically prioritised load-balancing.

In the field of Grid Computing, the models do not address sequential workloads, but are analysed due to their load-distribution simi-

larities. Here, fundamental requirements can be gained by analysing the publications about model building of established authors.

Models of application clusters are analysed due to their history in SOA. Analysed models do not cover multiple data centres, service billing, or service cascades.

Section 3.2 introduces a categorisation of quality of service, experience, and business. In addition, the models of quality assurance in the fields of Cloud Computing, Grid Computing, web service middleware, networks, and green data centres are analysed.

Analysis of related work in the field of quality assurance in Cloud Computing environments reveal that the known approaches can be compared to the Service Broker approach beside its embedding in the overall provision model. Approaches in the field of Grid Computing do not cover the relation between resources, services, and consumers addressed in this thesis. The work analysed in the field of web service middleware is not able to assure service quality per consumer in meshed service cascades. Work in the field of networks is restricted to control attempts on the network layer itself. The focus of green data centres lies on the reduction of power consumption through the consolidation of hardware servers. To reach this goal, approaches for load prediction, load balancing, or service level management are described in different publications. None of the analysed approaches is able to cover the relation between resources, services, and consumers, as introduced in this thesis.

Summarising the analysis of related work, publications of other authors do not address the usage of non-functional information detailed in SLAs to establish useful relationships between high-level performance targets and low-level infrastructure metrics, in order to cover the relation between resources, services, and consumers.

Part II

CONTRIBUTIONS

4

REQUIREMENTS CONCERNING A GENERIC SERVICE LIFE CYCLE

Regarding its service offers, the basic properties of the UC business model (cf. Section 2.2) are to serve differentiated customer groups, to scale resources on demand, to price customers by usage, and to keep reliable quality agreements. These properties target the core relation of UC between customer, service, and resource.

This chapter analyses the requirements of the UC business model concerning a generic life cycle for SOC service offers hosted on IaaS (cf. Section 2.6). Also, it introduces the specific requirements of the UC-specific relations inside a generic service life cycle (cf. Section 4.2), the UC-specific requirements concerning provision quality control (cf. Section 4.3), and the primary requirements on provision platforms (cf. Section 4.4). In Section 4.5, the results of the chapter are summarised.

4.1 RESEARCH METHODOLOGY

As introduced in Section 2.1.1, this thesis examines the hypothesis that an improved representation of the core relation of Utility Computing within a generic service life cycle leads to an all in all more cost efficient service provision. To achieve such a goal, a core provision model, a concept for usage-centred assurance of service quality, and a resource and cost simulation model are modelled.

As preparation of the modelling, this chapter analyses the requirements in the context of this thesis. In Section 4.2, requirements of the relations inside a service life cycle are collected based on related work and the project expertise of the author, previously introduced in Section 1.1. The conception of service levels and pricing for a diverse set of services affects the hole service life cycle. Business managers, IT architects, and operations managers need a common ground to exchange information about planned and deployed service offers. Out of this initial position, the publications addressing life cycles and experiences with service life cycles are analysed for their internal relations relevant in the context of this work.

In Section 4.3, the control of provision quality for SOC service offers hosted on IaaS is analysed. Based on the outcomes of the analysis of related work in Section 3.2.1 and in cooperation with business partners, the section further more analyses service quality and service

level agreements in the context of Utility Computing. The analysis of service level agreements starts with the definition of SLA in the context of this thesis based on related work and project expertise of the author. The feasibility of business processes is analysed based on requirements from business partners.

In Section 4.4, the requirements on provision platforms are analysed based on related work in the fields of Grid Computing and known industry standards. The functional requirement extraction is based on the work of the OGSA community and the GRASP project. The industry standards ITIL and COBIT are analysed for requirements from the operations perspective.

4.2 RELATIONS INSIDE A SERVICE LIFE CYCLE

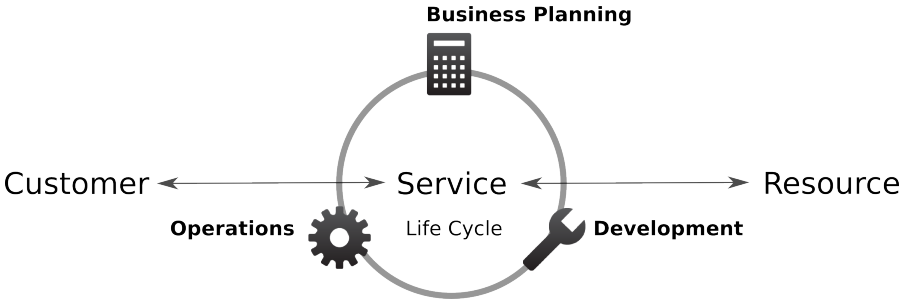


Figure 21: Overview of the Relation Between Customer, Service, and Resource

This chapter analyses the mapping of the customer-service-resource relation within a generic service life cycle for SOC service offers hosted on IaaS, as shown in Figure 21. To converge to the current representation of the UC-specific relations in a generic life cycle, defined in Section 2.5, the Section 4.2.1 analyses the relations in the cycle phase of business planning from the point of view of a business manager. The analysis is continued in Section 4.2.2 from the point of view of an IT architect in the cycle phase of development. In Section 4.2.3, the customer-service-resource relation is analysed from the point of view of an operations manager in the corresponding cycle phase of operations. Section 4.2.4 summarises the results of the previous sections and outlines the customer-service-resource relation giving an example.

4.2.1 *Cost-Price-Customer Relation*


Business Planning

The life cycle of UC service offers is analysed from the point of view of business managers in the cycle phase of business planning. From this economic point of view, the pay-per-use pricing models in UC business models increase the interdependence between costs of service delivery, achievable price for the service, and the expected number of customers in conjunction with their estimated usage intensity.

In the following, the results of the analysis of the cost-price-customer relation, as depicted in Figure 22, are presented.

In opposite to other business models, UC depends on price calculations at runtime (Mendoza, 2007, p. 5) based on the customer's resource consumption, shown as part of relation R_{CS} between customer and service. For service providers it might be interesting to distinguish three cases during price calculation (Liang-Jie Zhang, 2007, p. 178), defined as:

Relation R_{CS}

- Per-standard-use price

  In this case, the service request has been processed successfully with a service request response providing the expected results, below the defined maximum response time. Also, the contracted maximum amount of simultaneously consumable resources, defined as standard-conform use, is not exhausted. The allocated resources during processing are billed based on the contracted price scale for standard-conform service use.

- Per-overrun-use price

  This case is similar to the previous per-standard-use price case. The service request is also successfully processed. In opposite to the per-standard-use price case, the contracted maximum amount of simultaneously consumable resources is exhausted during request processing. In result, the allocated resource during processing are billed due to the contracted price scale for non standard-conform service use.

- Service level penalties

  In this case, the processing of the service request failed. Service requests can fail in terms of request loss, violation of maximum latency, or other metrics agreed in the SLA (Buyya et al., 2011, p. 305). Optionally, the causes of failure can be provided with

individual penalties. In this thesis it is estimated that monetary penalties are issued to the customer.

Correspondingly, during price scale building a suitable unit of consumption has to be defined, shown as part of relation R_{SR} between service and resource. A unit of consumption represents a measurable fraction of a certain resource (e.g., bytes of memory, cycles of processor time). Corresponding, for each unit of consumption, costs per unit have to be calculated. Therefore, the costs per physical host have to be determined. The resulting relation attributes are defined as:

Relation R_{SR}

- Per-unit-of-consumption costs;
- Per-physical-host costs.

Essential for the effective operation of the relations R_{CS} and R_{SR} are the knowledge, characterisation, and transfer of the customer's usage behaviour between acting parties within the service life cycle (Mendoza, 2007, p. 59). For price scale building business managers have to characterise the estimated usage behaviour of future customers introduced as UP_1. Otherwise, it will not be possible to make proper cost predictions resulting from estimations on resource consumption by IT architects and/or operations managers. Without cost estimations, scale effects cannot be considered adequately. For contracting - especially when separate overrun price scales or service level penalties are agreed - the contracted amount of standard usage referred to as UP_2 has to be characterised. In the case of billing, the actual amount of consumed resources has to be characterised as usage and transferred to the billing instance. This observed usage is referenced as MU_1. The relevant types of usage in the cost-price-customer relation are defined as:

Types of usage

- Estimated usage as UP_1;
- Contracted usage as UP_2;
- Observed usage as MU_1.

Both usage characterisations, UP_1 and UP_2, describe expected usage behaviour. In addition, MU_1 describes monitored usage behaviour including resource consumption metrics. Both types of usage characterisation are defined as:

Definition of expected usage as UP

- Usage complexity classifications as UCC

  For the classification of usage behaviour the central aspect is the ability to describe the complexity of usage, as this indicates the prospective resource consumption during processing. For example, a provider could define two classes of file upload behaviour to his document management service. Let UUC_{e1} be the behaviour of a group of file uploaders[1] that only upload large text documents and let UUC_{e2} be the behaviour of a group of file uploaders that mainly upload small binary files like pictures. The amount of processing resources used for indexing significantly varies between UUC_{e1} and UUC_{e2}.

- Average usage frequency per interval as UF

  Beside the ability to describe the complexity of usage, the frequency of this usage is important for the description of usage behaviour. For example[2], given the previously introduced classification UUC_{e1}, the group of file uploaders showing the usage complexity UUC_{e1} use the document management service once per minute in mean, assuming normal distribution, referred to as UF_{e1}.

- Usage schedule of UFs associated with UCCs

  As usage behaviour may vary over time, it can be described by utilising a schedule that consolidates the usage complexity classification and the usage frequency for a certain group of users showing varying usage behaviour over time. For example[3], given a group of users that use the example document management service with the usage complexity UUC_{e1}, this group use the service within the working hours with the usage frequency UF_{e1}. The hour before lunch and the hour before leaving work are peak hours with the usage frequency UF_{e2} with five uploads per minute in mean.

Definition of monitored usage MU

- Service requests processed

  Similar to UP's UF, *service requests processed* describes the quantitative usage behaviour for a given interval. MU describes the number of processed service requests. In addition, when usage

1 Users that send files to a service.
2 The example is a basic abstraction from the services of the business partners.
3 The example is an advanced abstraction from the services of the business partners.

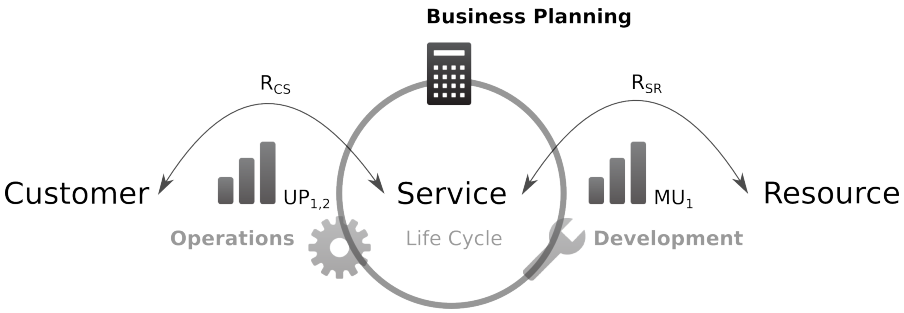


Figure 22: Relations Between Price, Cost, and Customer in a Generic Service Life Cycle

behaviour varies over time a schedule can be extracted from the monitored information and added to the MU.

- Resource consumption per request

  In addition to the request amount, MU contains the metrics of the actual resource consumption per service request for the given interval. Typical resource metrics for monitoring can be processing cycles, memory use, disk allocation, or power consumption (Buyya et al., 2011, p. 421). This metrics are used for service billing and cost calculations (Mendoza, 2007, p. 132).

The previous analysis reveals two issues in the cost-price-customer relation. At runtime, the relation R_{CS} depends on the analysis of a formal description of MU_1 for its price calculations and optional economic resource capacity planning. At time of business planning, relation R_{CS} depends on the reliable prediction of MU_1 based on a formal description of UP_1 for cost estimations. At time of contracting, relation R_{CS} depends on a formal description of UP_2. At this point, it can be stated that a formal description of service usage is an essential, but missing part of the cost-price-customer relation.

4.2.2 *Consumer-Service Relation*

The life cycle of UC service offers is analysed from the point of view of IT architects in the cycle phase of development. From this technical point of view, the application of UC business models increases the probability of services to be used by a higher fluctuating number of users with more inhomogeneous usage behaviour, while the complexity of service cascades and the claim for the reliable prediction of resource demands increases driven by business manager demands. To enable pay-per-use pricing models, the key factors in UC service

development are to keep predicted resource demands and to comply with planned service levels avoiding monetary penalties.

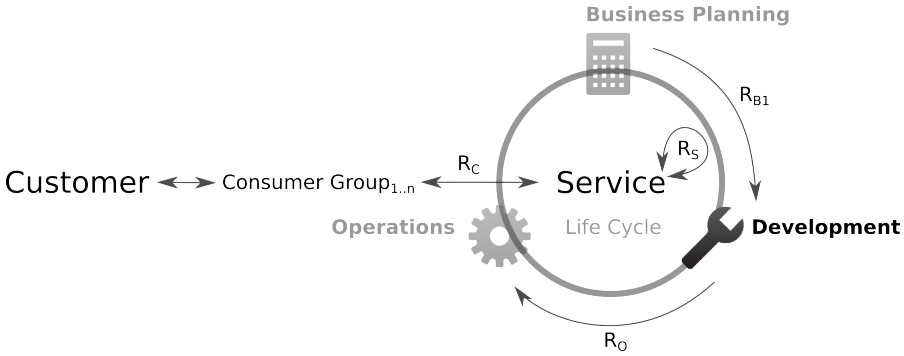


Figure 23: Relations Between Consumer and Service in a Generic Service Life Cycle

In the following, the results of the analysis of the consumer-service relation, as depicted in Figure 23, are presented.

The analysis of the relation between a service and its consumer starts at the core relation, referred to as R_C. Consumers are grouped corresponding to their usage behaviour (Liang-Jie Zhang, 2007, p. 3). It is assumed that service contracts consist of four major parts that define the legal conditions, the service functionalities, the non-functional technical properties, and the service pricing (Buyya et al., 2011, p. 601). Relation R_C represents the non-functional technical contract properties per consumer group, introduced as *service level*. The specification of the classification of service levels is defined[4] as:

4 The definition is based on the requirements analysed in the context of the business partners.

Service level classification

- Maximum latency

  Specifies the maximum acceptable latency for the interval between the receipt of a service request and the return of the processed service request response by a service provider.

- Average service availability per interval

  Specifies the minimum experienced average availability of a service offer conform to the terms specified for a certain service level.

- Physical location restrictions

  Specifies identifiers of the allowed physical locations for the processing of service requests corresponding to legal requirements of the service customer.

- Backup specifications

  Specifies the demanded procedures for the preparation and storage of data copies designated for permanent storage on the resources of the service provider for future disaster recovery.

The relation R_S represents the relations between services among each other. R_S reflects the interdependencies of a given service with the service itself as root of the service cascade. The relation R_S is determined by the software architecture of a service offer.

As second phase of the service life cycle, development retrieves its requirements from the previous phase of business planning.

> "[...] new data, which includes usage patterns, will be added to the list of things to be considered [by IT architects]. By looking at both functional and usage requirements, a design for an on-demand IT infrastructure can be implemented to eliminate problems such as underutilized resources and low return on investments." (Mendoza, 2007, p. 228)

The estimated usage UP_1 is passed on from business planning to development in relation R_{B1}.

Further in the life cycle, development passes on its service implementation as deployment sets and a generic usage description as part of the relation R_O, defined as:

Relation R_O

- Allocation of deployment sets

  A deployment set represents all technical resources (e.g., binary files, configuration files, data collections) that are necessary to instantiate the technical service offer.

- Description of generic usage as GU_C for the developed service cascade

  – Usage complexity classifications as UCC

    Abstract description of the load-dependent resource demand during the processing of a service request.

  – Generated usage intensity

    Description of the quantitative and qualitative usage of supplying service offers as $UP_{Sm} \quad m = \{1..r\}$.

The key issue in service development is the inclusion of estimated usage behaviour into the processes of quality assurance, as previously stated by Mendoza (2007). This inclusion of usage behaviour enables the testing of actual resource demands continuously during development, and therefore provides the basis for early revision of cost estimations in business planning.

4.2.3 *Service-Resource Relation*


Operations

The life cycle of UC service offers is analysed from the point of view of operations managers in the cycle phase of operations. From this technical point of view, UC service offers have less predictable resource demands due to the increasing probability of services to be used by a higher fluctuating number of users with more inhomogeneous usage behaviour[5]. While the complexity and interdependencies of and in between service cascades rise, it becomes more challenging to comply with planned service levels avoiding monetary penalties.

In the following, the results of the analyses of the service-resource relation, as depicted in Figure 24, is presented.

5 Due to the requirements gathered by the business partners.

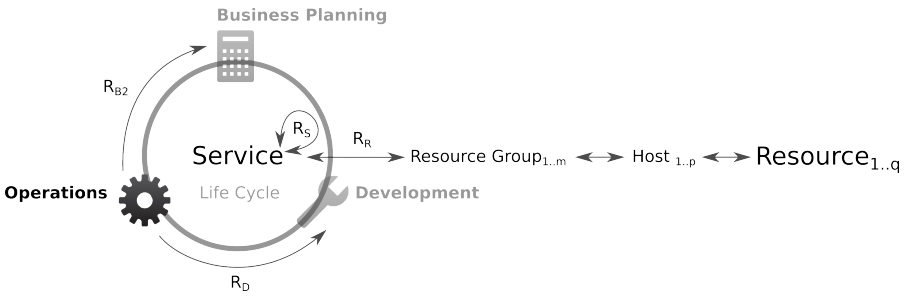


Figure 24: Relations Between Service and Resource in a Generic Service Life Cycle

The analysis of the relation between a service and its available resources starts at the core relation, referred to as R_R. Service instances are deployed on physical hosts of certain resource groups. Resources of hosts are grouped by their non-functional operations properties, defined in the service level classification in the following. Hosts are treated as group of resources like processing time, memory, or storage space.

Service level classification

- Average maximum latency

  Specifies the average of the maximum experienced latencies of service request responses hosted on a resource for a certain interval.

- Average service availability per interval

  Specifies the average of the minimum experienced availability of a resource for a certain interval.

- Physical location

  Represents an identifier for the physical location of a resource.

- Backup properties

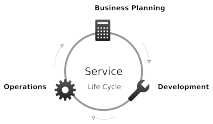
  Specifies the available procedures for the making and storage of copies of data disposed for permanent storage on the resources of the service provider for future disaster recovery.

The relation R_S represents the relations between services among each other, as introduced in Section 4.2.2. The analysis of the service-resource relation adds the global interdependencies of service cascades to R_S. This includes the interdependencies among service offers as well as the interdependencies among service levels, as both are competitors for the available resources.

The observed usage MU_1 is passed from operations to business planning in relation R_{B2}.

Backward in the life cycle, operations offers a technical interface for provision platform interaction to enable service deployment. The provision of this interface is represented by relation R_D.

4.2.4 *Outlining the Relation Between Customer, Service, and Resource*



This section subsumes the previous results giving an example (Heckmann and Phippen, 2010). The example addresses the previous distinctly analysed views on the customer-service-resource relation (cf. Section 4.2.1, Section 4.2.2, and Section 4.2.3) and merges them into a consistent view on this relation in a generic life cycle for UC offers.

Let a service provider be planning to offer a new service. The service should be provided based on web service technologies (Haas and Brown, 2004) via the Internet. The business model for the service offer conforms to the model of UC. During business planning, the business manager expects three market segments the service could be offered to economically successful. After the analysis of typical usage behaviour of representative customers within the market segments, these usage expectations are given as UP_{1m} $m = \{a, b, c\}$.

IT architects and operations managers can now provide their estimations of resource consumption as MU_1. This resource consumption is the basis for cost estimations on the planned service offer. This enables business managers to calculate price scales for the identified market segments. In addition to the price scale for each market segment, two service levels per market segment are defined as SL_{mn} $n = \{1, 2\}$. Summarised, the standard scenario for service introduction is $S_{st} = \{UP_{1m}, SL_{mn}\}$. In addition, business managers want to analyse a best case scenario, introduced as S_{bc}, and a worst case scenario, referred to as S_{wc}, with constant SL_{mn}, but varying UP_{1m}.

During development of the service cascade, the IT architect verifies and optionally adapts his previous resource consumption estimation MU_1. This enables early quality assurance and prevents from undetected rising of future operations costs (e.g., in cases where the processing of service requests requires more resources or in cases where other service offers are reused in the service cascade and therefore new costs and/or resource dependencies arise).

During operations, the operations manager needs to manage the provision quality of all concurrent service offers hosted by the service

provider. The management of the provision quality at runtime is subject to Section 4.3. Beside the runtime management, also a continuous capacity planning is essential to keep provision quality on the long term. As previously done in business planning, $S'_p \; p = \{wc, st, bc\}$ are estimated per service offer as future scenarios. In addition to S, S' also takes UP_2 as contracted usage into account.

The example reveals the importance of an integrated view on the customer-service-resource relation for UC service providers. The missing of a well documented description of such a consistent view is introduced as problem P_{1b} in Section 1.3. This thesis proposes to evolve a comprehensive usage-centred data model in order to map the gained analysis results in a reusable description of the relation between customer, service, and resource. In addition, the example shows the need for a standardised usage description for data exchange between phases of a life cycle. This problem was introduced in Section 1.3 as problem P_{1a}.

The key results of the analysis presented in Section 4.2 are also published in (Heckmann et al., 2008, 2009; Heckmann and Phippen, 2010).

4.3 REQUIREMENTS ON PROVISION QUALITY CONTROL

This section analyses the control of provision quality for SOC service offers hosted on IaaS.

As the analysis of related work in Section 3.2.1 reveals, there are no continuous approaches for the provision of UC service offers in SOC architectures hosted on IaaS (cf. Section 2.6) to assure service quality throughout all OSI layers and converge the quality-related ontologies of service, experience, and business.

The findings show that the monitoring of service quality responds to technical thresholds, classified as QoS. Monitoring of service quality during runtime does not take economic characteristics into account, as classified in QoBiz. Current work does not distinguish between the views on service quality from the point of view of the service provider (QoS) and the service consumer, classified as QoE. For UC service offers, overbooking of resources on purpose depends on the economic weighting of the committed QoE and the available QoS that can be offered to the competing service requests. Thus, there is no continuous combined economic and technical control of service quality through all OSI layers from a consumer to a resource in UC scenarios. This is one aspect, why control of service quality is introduced in Section 1.3 as problem P_2.

To prepare the identification of an approach to close this gap, service quality is analysed from the point of view of a UC service provider in Section 4.3.1.

One of the most common uses of SOC service offers is the technical representation of business logic to represent parts of business processes. In complex service cascades with redundant service offers, as introduced in Section 1.3, the estimation of the feasibility of business processes based on SOC architectures is an open research question, as the insufficient technical approaches like shown in Section 3.2.1 and by Heckmann et al. (2011, 2012a) illustrate. Therefore, a generic multi-tier SOC architecture is elaborated in Section 4.3.2 to enable the identification of an approach to close this gap.

4.3.1 *Service Quality and Service Level Agreements in Utility Computing*

Tied to the classification of QoS in Section 3.2.1, in this section an analysis of service quality from the point of view of a UC service provider is presented. In UC, service quality has an important role. Constitutive to the definition of UC in Section 2.2, necessity, reliability, usability, utilisation, scalability, and exclusivity are the main criteria to characterise utilities. All of these criteria directly or indirectly address the quality of the service provision.

Service Level Agreements

The examination of service quality starts with its agreement. The definition of suitable quality criteria for a service offer is done as part of an SLA. As additional part of an SLA, the type of monitoring for these quality criteria is specified. This thesis addresses probabilistic SLAs (Buyya et al., 2011, p. 174) on the level of technical agreements about service quality. The level of functional correctness is not examined. *Technical* agreements are defined as: all technical measurable requirements that are relevant for the attended service response properties beside functional correctness. In opposite, service offers are functional correct, when the service processing of service requests results in the expected behaviour, conform to a given business logic including changes in non-volatile business data and corresponding service request responses.

To illustrate the definition of functional correctness, a short example is provided in the following. Let a service method invocation with consistent method parameters be conform to the specified parameter value ranges. Such a method invocation must result in an accurate result set consisting of the expected business data.

In this thesis, response time is considered as the only primary SLA criterion from the UC service consumer's point of view. Other possible primary SLA criteria (e.g., continuity or security) are not considered in this thesis. From the point of view of a UC service consumer, response times must be independent of the overall provider load. This requirement implies the abdication of directly contracted resource reservations of any kind. Otherwise, service providers cannot achieve the targeted dynamic scaling of their underlying resource architecture and the dynamic optimisation of resource utilisation in general.

From the point of view of a UC service provider, response time of service request responses as primary SLA criterion depends on secondary SLA criteria. In case of UC service offers, these secondary criteria are usage complexity and usage frequency per interval (cf. UUC and UF in Section 4.2.1). The introduced primary and secondary criteria are subject to individual SLAs between service consumer and provider.

Response Time Classification

In addition to service quality criteria for service offers, corresponding classes of value ranges for such criteria must be given for their successful technical monitoring. Corresponding actions must be defined within SLAs during contracting. For the primary criterion of response time this is done in the following. Aberrations to the contracted maximum response time could be classified as follows.

- RT_{st-} — Below contracted minimum

  Classifies service response times below the minimum as RT_{min} of the determined acceptable response time.

- RT_{st} — Within contracted range

  Classifies service response times between RT_{min} and the maximum as RT_{max} of the determined acceptable response time.

- RT_{st+} — Above contracted maximum

  Classifies service response times above RT_{max} of the determined acceptable response time.

- RT_{np} — Unprocessed request

  Classifies service requests that are technically and functionally conform to the processing requirements of a certain service offer, with the service request response not sent back to the service consumer (e.g., due to technical errors or request dropping in cases of resource overload).

Quantitative and Qualitative Aspects of Service Quality Control

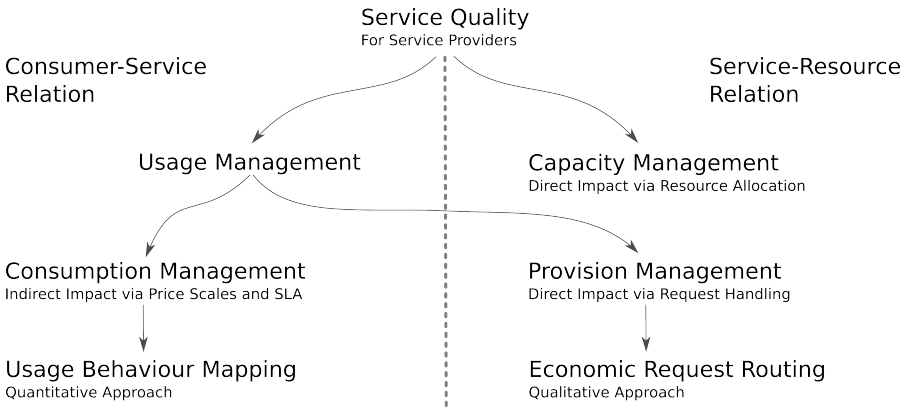


Figure 25: Overview of Quantitative and Qualitative Aspects of Service Quality Control

The analysis of the management of service quality introduces the quantitative and qualitative aspects of quality control for UC services. Beside the previous analysis on the criteria for service quality and the agreeing on service quality, the management of service quality is of interest to service providers. The *capacity management* is the classical approach to manage service quality in the service-resource relation. In capacity management, the goal is to dedicate a certain amount of resources for a specific service offer to achieve a certain QoE.

For UC service offers, the management of service quality is analysed further. Quality cannot only be managed by capacity provision, but also through the management of the usage itself. Combining both approaches offers a more granular management of service quality to service providers.

Usage management consists of *provision management* and *consumption management*, as shown in Figure 25. While provision management also addresses the service-resource relation, consumption management is concerned with the customer-service relation. Provision management offers a qualitative approach to the management of service quality in the service-resource relation. Provision management monitors, analyses, and controls the service request routing and resulting utilisation of processing resources. In opposite, consumption management describes a quantitative approach to the management of service quality in the customer-service relation. Consumption management uses price scales and SLAs as the provider's instruments to indirectly influence the usage behaviour of its service customers.

Summarising Service Quality

The analysis of service quality and SLAs in UC from the point of view of a UC service provider reveals the missing standardisation of the usage description for the quantitative usage behaviour mapping in the consumption management. This problem is previously introduced as P_{1a} in Section 1.3.

The analysis also shows that SLA criteria for UC service offers have to be limited to the response time of service request responses. To provide a more granular management of service quality to UC service providers, usage management has to be in the focus in UC service quality control. These findings allow approaches that focus on a combined economic and technical control to assure service quality throughout all OSI layers and converge QoS, QoE, and QoBiz. These are additional aspects that made control of service quality necessary to be introduced in Section 1.3 as problem P_2.

The results of the analysis presented in Section 4.3.1 are also published by Heckmann and Phippen (2010).

4.3.2 *Feasibility of Business Processes Operated on SOC Architectures Hosted on IaaS*

From the point of view of business managers, the feasibility of business processes is essential for economic success. But not only is the state of feasibility of interest, but also the workload of processes. Certain workloads may reach critical technical or organisational thresholds. From the point of view of technical operations, offering such state information for business processes can become a complex challenge, especially for business processes based on service-oriented architectures. The complexity of this challenge arises from highly meshed service cascades and redundant alternate service offers, as previously introduced in Section 1.3.

Multi-Tier SOC Infrastructure Analysis

To enable the development of an approach to estimate the feasibility of business processes operated on SOC architectures hosted in IaaS, a generic model for IaaS infrastructures for the hosting of SOC services is described in the following.

The generic architecture this thesis introduces consists of eight horizontal layers, as shown in Figure 26. As entry layer, business processes represent the abstract description of steps to be processed to produce a business value. The automated steps within such a business process are represented by technical workflows/processes in the

underlying layer. The business functionality within these technical workflows is provided by the orchestration of the available service offers on the layer of service instances (e.g., web services). The service instances are hosted on the application infrastructure layer (e.g., within database systems or application servers). The software components of the application infrastructure layer are deployed on the operating system layer. Thereby, each operating system instance runs in a virtual machine on the virtualisation layer. The resources allocated on the virtualisation layer are retrieved on the physical systems layer. On the final layer, the network services connect the systems on the physical systems layer. The connections are implemented relying on resources such as routers, switches, or domain name services.

The previously described horizontal multi-tier SOC infrastructure is accompanied by a vertical layer, with the business process layer as exception. This vertical layer is the technical monitoring, which tracks and evaluates technical measuring points on the horizontal layers (e.g., running processes, log file analysis, network stack availability, processing load, memory, or storage usage).

The complexity of the management of such infrastructures rises when additional requirements have to be considered, defined as:

- Handling of complex service cascades with redundant service offers;
- Integration of internal and external service providers;
- Support for an intermediate logic that changes the invocation target of a service request at runtime (e.g., to seamlessly switch between redundant service offers).

The complexity of the introduced generic multi-tier SOC architecture illustrates the last aspects that made control of service quality necessary to be introduced in Section 1.3 as problem P_2. In addition, the shown multi-tier architecture offers the basis for the identification of a suitable approach to the estimation of business process feasibility.

The presented generic architecture disclaims technical details to ensure service availability for individual layer services like redundant failover layouts for servers or network components. In this thesis, it is estimated that these technics are applied in implementations of those architectures as individually appropriate.

The results of the analysis presented in Section 4.3.2 have been evolved in cooperation with a business partner and are also published by Heckmann et al. (2011, 2012a).

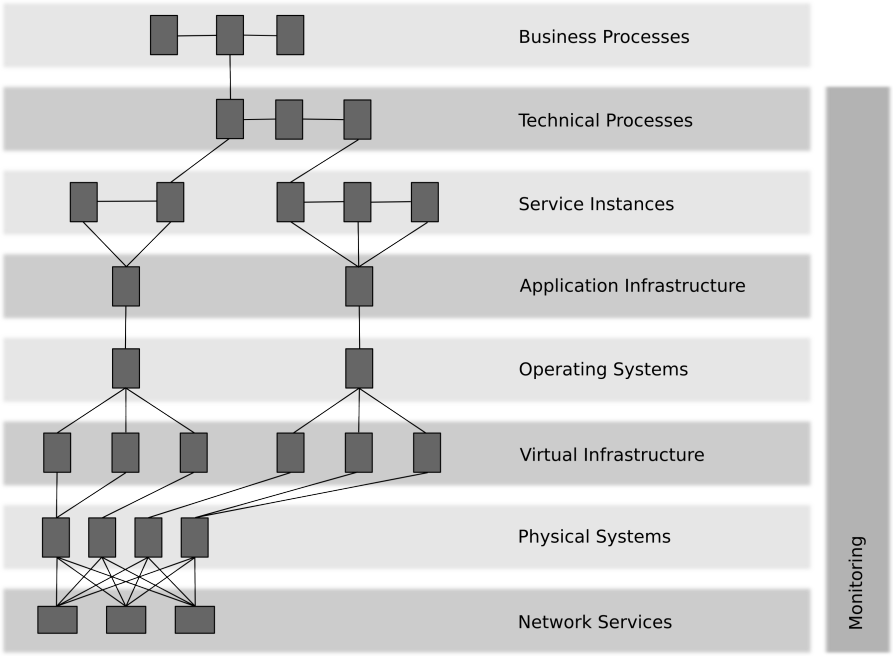


Figure 26: Generic Multi-Tier SOC Architecture Including Redundant Service Offers

4.4 REQUIREMENTS ON PROVISION PLATFORMS

This section aims at evolving a set of basic functional requirements on provision platforms from the point of view of a service provider. These requirements should enable the constitutive development of a core model for UC-conform provision platforms, as introduced in Section 1.3 as problem P_{1c}.

To determine the basic functional requirements three sources are analysed. The main set of requirements is extracted from the collection of use cases of the OGSA community in Section 4.4.1. This set is enriched with two minor sources for functional requirements. Section 4.4.2 analyses qualified industry standards as basis for the extraction of requirements. In Section 4.4.3, this thesis analyses the mediation conditions specific for Utility Computing service offers.

4.4.1 *Functional Requirement Extraction*

As introduced in Section 3.1.2, during the specification of the OGSA the Grid community collected a wide spectrum of scenarios describing business and scientific use of distributed computing applications, not restricted to parallel computing workloads. This use case collec-

tion is analysed for its functional requirements on a core provision model for UC service offers.

A rough overview of the following results of the analysis, presented in Section 4.4.1, are also published by Heckmann (2007).

Functional Requirement Constraints

This thesis does not aim to evolve a comprehensive provision model for UC service offers. Targeted is a core provision model for UC service offers to clarify the relation between service consumer and resources and, in addition, to enable decisions about technical frameworks for UC service provision and, thereby, to enable cost estimations for service development and operations. The minimum requirements to evolve such a core provision model derived from the UC definition in Section 2.2 and the thesis scene setting in Section 2.6 are defined as:

- Provision of SOC service offers;
- Scalability of service offers;
- Management of provision quality;
- Basic accounting model.

The management of transactions that are overarching service offer is excluded in this thesis, as it is not estimated as a minimum requirement for UC service offers.

Functional Requirements for Commercial Data Centres

The following functional requirements are extracted from the OGSA use case *Commercial Data Center* (CDC) (Foster et al., 2004):

- Discovery

  To enable the operations of a CDC, at least one reference to this CDC has to be also published in one or more discovery services.

- Authentication, authorisation, and accounting (AAA)

  The CDC authenticates a customer and authorises its job requests. In this process, the CDC also identifies the customer's policies including, but not limited to, SLA, security, scheduling, and brokering policies.

- Advanced reservation

  Job requests contain a date that describes the favoured start time for the job processing. For job requests without such timing information, the CDC's reservation service should choose

a suitable start time for request processing. Such an advanced reservation feature is useful in the context of shared parallel computing systems to optimise resource utilisation over time.

In this thesis, this requirement is excluded, as it does not apply to sequential workload scenarios.

- Brokering

  The brokering service in a CDC identifies suitable resources for the processing of a job request with a given start time. This resource selection takes access control and quotas into account. The identified resource is reserved and its identifier is returned to the customer.

- Data sharing

  A job request contains the information about the required data access during processing (e.g., database and/or files). Access to required data should also be considered during resource identification during brokering.

- Provisioning

  A sufficient time period prior to the planned start time of the job request processing, the CDC initiates the provision of the application and data to the reserved resources.

- Scheduling

  At start time the processing of the job request is initiated by the CDC.

  In this thesis this requirement is excluded, as it does not apply to sequential workload scenarios.

- Metering and accounting

  During the processing of the job, a metering service monitors the resource usage. This monitoring data is passed on to an accounting service.

- Fault handling

  The fault handling process is designated through fault management policies per customer to ensure the job submitter is informed about errors during job processing.

- Policy

  Policies should be available to determine specific job properties. For example, a brokering policy should define the resource usage quotas per customer. Error and event policies should be

available to enable autonomous management including provisioning and failover.

Functional Requirements for Interoperation

The following functional requirements extend the previous use case and are extracted from the OGSA use case *Inter Grid* (Foster et al., 2004):

- Discovery and brokering

  It should be possible to discover and broker services across organisations with various levels of security policies being considered.

- Metering and accounting

  It should be possible to access heterogeneous storage systems with support for varying system interfaces. Requirements for accounting include the handling of multiple accounts and/or accounting systems.

- Monitoring

  Monitoring should offer a cross-organisational view on resources with focus on life cycle management and fault handling. Automated actions should be supported in reaction to monitored events. The monitored data should be accessible via Application Programming Interface (API).

- Provisioning

  The provisioning service should be compatible with non-Grid provisioning systems. One solution could be the specification of APIs for Grid provisioning service interaction.

  In this thesis, this requirement is excluded, as it does not apply to sequential workload scenarios.

- Resource collision resolution

  In mixed Grid and non-Grid environments, resources could be used by both types of workload at the same time, but this is considered as undesirable. This should be prevented by a collision management service.

  In this thesis, this requirement is excluded, as it does not apply to pure sequential workload scenarios.

- Usability

  It should be possible to use resources for both types of access, batch and interactive.

- Management

  The management and monitoring of resource usage including the detection of SLA violations should be supported.

- Load-balancing

  Load-balancing should take security policies into account.

- Legacy application management

  Legacy applications are owned software systems that cannot be changed, but are too valuable to be replaced, or too complex to be reimplemented. In the integration of such legacy applications into the CDC, it has to be considered that the applications should still be usable in addition to the integration realisation.

- Administration

  The administration of a CDC should support automation of standard maintenance tasks. CDCs should be able to self-organise and self-describe, managed by low-level configuration details based on higher-level configurations and management policies, specified by administrators.

  Administration should also support the migration of non-SOA services to SOA services to be deployable in a CDC.

  Software updates for the CDC framework software should not enforce any service unavailability.

- Programming model

  A guideline for software development of CDC applications including a guide for the integration of legacy applications.

  In this thesis, this requirement is excluded, as it is estimated to exceed the minimum requirements of UC service provision.

- Program execution

  It should be possible to specify the range of tolerated processing (e.g., priorities for the operating system scheduler to provide real time processing of applications) and network delays.

- Logging

  The logging of CDC events should be detailed enough to perform reliable troubleshooting. This could include varying levels of logging, as required by the administrators.

- Policy

  There should be policies available for the specification of security and identity properties.

- Collaboration requirements

  Various levels of security should be available to control the access to resources or services.

- User interfaces

  To submit, monitor, and control the job requests, interfaces should be available to CDC users and administrators. In addition, for administrators an interface for the monitoring of the CDC performance should be provided.

Functional Requirements for Resource Resellers

The following functional requirements are extracted from the OGSA use case *Grid Resource Resellers* (Foster et al., 2004):

- Discovery and brokering

  Resellers operate brokers to distribute the workloads to available resources. To decide about the distribution, information about the job preferences and resource policies must be given. A broker can use metrics like processing performance, network delays, or resource costs to select specific resources for processing. Resellers should be able to identify the original resource owner, even when the resource is resold by other resellers. And end-users should be able to identify resellers.

- Metering and accounting

  The resellers should bill end users based on the actual consumed resources during job processing.

- Monitoring

  The resource owner should monitor resource usage per individual end-user to track resource abuse, even when the resource is sold through multiple resellers.

- Policy

  End-users and resource owners will probably have complicated policies, as well as the resellers might have. Resellers must not be able to resell resources in a way that it violates the resource owner's policies.

- Extended SLA

  SLAs should additionally carry cost information and penalties.

Functional Requirements for Resource Usage Services

The following functional requirements are extracted from the OGSA use case *Resource Usage Service* (RUS) (Foster et al., 2004):

- Discovery and brokering

  The RUS should offer discovery mechanisms to locate sources for resource metrics.

- Monitoring

  The monitoring fabric should provide methods to collect usage metrics.

- Policy

  A policy service should manage the configuration and orchestration of RUS instances.

- Security

  A security service should offer the capability known from common authentication, authorisation, and accounting (AAA) (Treadwell, 2005) systems. For some commercial scenarios the RUS should be able to store account identifiers from an AAA system along with consumption metrics.

Functional Requirements for IT Infrastructure and Management

The following functional requirements are extracted from the OGSA use case *IT Infrastructure and Management* (Von Reich, 2004):

- Authentication

  A system should authenticate its users.

- Authorisation

  Users should obtain their credentials from the local virtual organisation (VO) (Treadwell, 2005) or optionally, in order to access remote applications, from the remote VO.

- Fault tolerance

  Possible sources of error lie the process of queue lookup, errors in transmission, or automated routing of job requests. These should be prevented by measures like redundant provision components and exception handling.

- Registry

  A registry should provide the service to enable the lookup of the location of available applications including the information whether this applications run local or remote.

- Resource specification

  The specification of resources should provide an inventory of applications on a physical host and corresponding records in the registry.

- Notification/Messaging

  The registry should be notifiable about changes in the application inventory.

- Resource selection

  The requirements for resource matching should be known. In addition, the demand for software licences to execute applications should be determined.

- Policy schema

  Relevant infrastructure and management properties should be specified through policies.

- Brokering and arbitration

  If multiple licence types would apply to one application, a given licence scheme should be evaluated together with given policies to pick a suitable licence.

- Reservation

  It should be possible to reserve licences demanded by applications in advance.

- Logging

  Status events should be logged.

- Hosting environment

  Hosting environment should be available and initialised.

- Data migration

  There should be a service that migrates applications to physical hosts for execution.

- Monitoring

  A service for monitoring should report installation progress, job status, and resource consumption.

- Metering

  A service for metering should record resource usage in terms of resource occupation and occupation duration. In addition, the usage of licences should be recorded.

- Auditing

  Usage and application profiles should be audited on physical hosts.

- Billing

  Users should be billed based on the metered data.

Functional Requirements for ASPs

The following functional requirements are extracted from the OGSA use case *GRID based ASP for Business* (Von Reich, 2004) and on the key results extracted from the GRASP project (cf. Section 3.1.2):

- Discovery

  Discovery is one of the core functions of an ASP infrastructure. Discovery is done by the locator subsystem. Whether ASP users search for suitable grid services or other ASPs are looking for grid services, in order to resell those, a locator subsystem is demanded to provide search functionality within the provided service offers. Beside the obvious option to search for services by functionality, the business domain of ASP demands the consideration of other search criteria such as price or specific service properties.

  It is proposed to redefine SLAs as a superset of functional, technical and business requirements, searchable by service users. Such SLAs should then represent a unique contract between a customer and a service provider. Providers publish SLA templates in the context of the locator subsystem. These SLA templates could be either be fixed or be adaptable to the customer's needs. The locator subsystem should be able to process such SLAs and return appropriate service endpoints to a user.

- Instantiation

  VOs should be built based on hosting environments (HE) (Treadwell, 2005) containing a gateway host and a pool of physical hosts. Users of service offers should not be enabled to directly invoke the service factory, but should be forced to be served by an instantiator subsystem. The instantiator subsystem should respond to requests for service instantiation by executing the request on behalf of the user respecting the given SLA. Other than the locator subsystem, the instantiator subsystem should consider the current resource load of the HE. Therefore, the instantiator subsystem interacts with multiple other subsystems

to gather information about the HE state, like with the monitoring or security subsystem.

- Load-balancing

  HEs usually comprise more than one physical host that is able to execute a grid service conform to a certain SLA. Obviously, this strongly depends on the current load on the physical host, the character of the grid service and the requirements resulting from the related SLA. Therefore, some kind of load-balancing system is required. Such a load-balancing system should generate a prioritised list of the physical host, able to process a certain request conform to its SLA. This list is processed by the instantiator subsystem to choose a suitable physical host.

- Metering and monitoring

  Load-balancing depends on the ability to measure certain system metrics. Beside common system metrics, an extensible metering and monitoring system should provide service-related metrics. Such service-related metrics could be the number of service invocations or the service performance. Other subsystems like accounting or monitoring could require these metrics.

- SLA management

  The use of sophisticated SLAs introduces additional requirements for SLA management systems. One requirement is that there should be a common SLA description language available. Another requirement is the demand for a corresponding parsers to process an SLA described in a common SLA description language. For example, the locator subsystem might do a fuzzy search on the SLA, while the load-balancing system looks up single metrics. As last requirement, an SLA management system should monitor service instances for SLA violations. In case of SLA violations, the SLA management system should either inform the accounting subsystem regarding the consideration of penalties or decide to destroy the service instance.

- Accounting

  Services executed on behalf of a customer should be accounted. The accounting subsystem should be able to handle scenarios, with services executed under varying SLAs. Varying SLAs can cause scenarios, with varying metrics considered for price calculations. In addition, the accounting subsystem should be able to consider penalties and manage composite services.

- Orchestration

  Different business models for ASPs should be supported. Beside simple models, where providers extend their legacy applications by service offers, a reseller model should be supported, where providers resell orchestrated services in an OGSA compliant way. Thus, the infrastructure should be able to expose orchestrated services as new service.

- Security

  Security issues - such as authorisation or authentication - should be considered by an infrastructure approach. This includes the demand to support multiple security-related roles such as service consumer or service provider.

- Deployment

  There should be a subsystem to autonomously manage the deployment, update, and removal of service instances on hosts including their registration at the other subsystems (e.g., the locator subsystem).

- Notification

  The complexity of service cascades and, in addition, of the subsystem infrastructure itself implicates the demand for a dedicated notification system to exchange state information between service instances and/or subsystems. Simple request-response approaches should not be considered in this case.

- Legacy application management

  To support ASPs that depend on the integration of the functionalities of legacy applications, tools and libraries should be provided to support this integration into service cascades.

Functional Requirements for Monitoring Architectures

The following functional requirements are extracted from the OGSA use case *Grid Monitoring Architecture* (Von Reich, 2004):

- Discovery and brokering

  There should be mechanisms available that enable service consumers to discover services, including the validation of the consumer's access rights.

- Data sharing

  To provide access and management of data and corresponding meta-data, mechanisms like replication, archiving, and caching should be available.

- Policy

  As providers and consumers conclude a contract on the conditions of service usage, the containing information should be used for - in terms of event and/or error policies - self-management within the infrastructure and/or failover of the monitoring system.

4.4.2 *Qualified Industry Standards*

Properties Derived From the ITIL Service Delivery Specification

The following raw properties relevant for provision platforms are extracted from the ITIL v2 IT service management set *Service Delivery* (Elsaesser, 2006; APM Group et al., 2011):

- Service level management (SLM)
  - Client service relation (CSR);
  - Operation level agreement (OLA);
  - Service achievement service catalogue;
  - Service charter including service level agreements;
  - Service quality plan including key performance indicators (KPI);
  - Service level requirement (SLR) including service specification sheet (SSS);
  - Underpinning contract.
- Availability management
  - Availability;
  - Reliability;
  - Maintainability;
  - Serviceability;
  - Uptime;
  - Downtime;
  - Operation time average downtime (ADT);
  - Annual failure rate (AFR);

  - Single point of failure (SPOF);
  - Mean time between failures (MTBF);
  - Mean time to repair (MTTR).

- Financial management
  - Budgeting;
  - Charging;
  - Accounting.

    In this thesis the financial management properties are excluded, as it is estimated to exceed the minimum requirements of UC service provision.

- Capacity management

  Capacity management should calculate the requirements for IT resources, measure the utilisation of the infrastructure, and plan the necessary infrastructure revisions.

- Continuity management

  Continuity management should offer disaster planning considering common failure reasons, common countermeasures to prevent data loss, and service failure[6]. The following properties should be included:
  - Cold standby;
  - Immediate standby;
  - Manual recovery;
  - Warm standby.

Properties Derived From the COBIT Framework Specification

The following raw properties relevant for provision platforms are extracted from the COBIT 4.0 framework (ISACA, 2005).

- Plan and organise
  - Define a strategic IT plan;
  - Define the information architecture;
  - Determine technological direction;
  - Define the IT processes, organisation, and relationships;
  - Manage the IT investment;

6 See also the *ITSCM process* within the ITIL specification.

  - Communicate management aims and direction;
  - Manage IT human resources;
  - Manage quality;
  - Assess and manage IT risks;
  - Manage projects.
- Acquire and implement
  - Identify automated solutions;
  - Acquire and maintain application software;
  - Acquire and maintain technology infrastructure;
  - Enable operation and use;
  - Procure IT resources;
  - Manage changes;
  - Install and accredit solutions and changes.
- Delivery and support
  - Define and manage service levels;
  - Manage third-party services;
  - Manage performance and capacity;
  - Ensure continuous service;
  - Ensure systems security;
  - Identify and allocate costs;
  - Educate and train users;
  - Manage service desk and incidents;
  - Manage the configuration;
  - Manage problems;
  - Manage data;
  - Manage the physical environment;
  - Manage operations.
- Monitor and evaluate
  - Monitor and evaluate IT performance;
  - Monitor and evaluate internal control;
  - Ensure regulatory compliance;
  - Provide IT governance.

4.4.3 *Mediation Conditions in Utility Computing*

Cost Domains

In this thesis, it is assumed that there may exist providers that need to distinguish between more than one cost domain. The term *cost domain* herein is defined as an abstract group of resources grouped by the criterion of the provider's costs that occur for the processing of a service request. Cost domains therefore represent economical boundaries.

Cost domains are often reflected in technical boundaries. Such boundaries can be implied by the physical location of resource in different data centres, significantly different hardware performance (e.g., in scenarios where older and newer hardware are operated simultaneously within the same data centre), or differing technologies for the operation of resources. But cost domains also can be reflected in legal boundaries like the location of technically comparable data centres in different countries.

The results of this analysis are also published by Heckmann et al. (2012b).

Provisioning Factors

The management of service requests at runtime - based on active measures to control the flow of service requests - aims to gain control over the resource utilisation by controlling the routing of service requests to their processing resources. Besides the continuous monitoring of the utilisation of processing resources, the decision about the route of requests is the core of provision management (cf. Section 4.3.1). As a basis to calculate the decision of request routing, technical and economical criteria of the later request processing have to be considered. In this thesis, these technical and economical criteria are called *Provisioning Factors*. Provisioning Factors aim to represent the qualitative aspect of the service-resource relation and are defined in the following.

Provisioning Factors are introduced as a group of three complement factors:

- Processing factor

  The processing factor aims at calculating the costs for the processing of a given service request on provider-owned resources. As basis for these cost calculations, the fixed costs for service hosting (e.g., for server acquisition, housing, and administrative personnel) and corresponding dynamic costs (e.g., for cooling and power) are taken into account. Not part of this thesis

is the identification of the individual combination of these fixed and dynamic costs. The calculation of the processing factor is not useful, till the availability of sufficient resources for request processing is ensured. This calculation must include all costs for sub-requests invoked by the initial request. This thesis estimates that a detailed analysis of complex service cascades, performed in order to find the optimum costs or in order to calculate the exact resource demand, may fail at runtime. In such cases, this thesis suggests the calculation of approximations instead.

- Outsourcing factor

  There are scenarios conceivable, where it can be an economical alternative to forward requests to other service providers for processing. For all layers of a service cascade such outsourcing decisions can be appropriate. For example, conceivable outsourcing scenarios may reach from the processing of customer requests on competitor sites in times of peak loads up to the dynamic processor picking for back-end services such as the retrieval of geological information. The outsourcing factor aims at calculating these costs for external request processing.

- Neglecting factor

  In opposite to the previously introduced factors, the neglecting factor aims at calculating the costs for an intentional violation of the SLA agreed upon between customer and service provider. The worst conceivable violation is the intentional drop of a request. But also, other aberrations from a corresponding SLA may might be possible. For all contracted variations of service level aberrations, the costs must be taken into account. The neglecting factor introduces an additional option, that increases the flexibility of routing decisions.

All elaborated Provisioning Factors calculate costs. In combination with the information about a consumer's contracted price list, the profit or loss of the possible routing decision can be calculated. The mentioned costs may vary over time on individually contracted factors (e.g., time of day or discounts on request amounts). The specified factors can be interpreted as proposals, from the point of view of this thesis. In other contexts, it is possible to add or remove criteria as needed.

The results of this analysis are also published by Heckmann and Phippen (2010).

4.4.4 *Summary of the Requirements on Provision Platforms*

Section 4.4 introduced a wide set of functional requirements for provision platforms. In Section 4.4.1, a partly overlapping set of requirements extracted from the use cases of the OGSA community has been extracted. This set includes requirements for commercial data centres, interoperation, resource resellers, resource usage services, IT infrastructure and management, application service providers, and monitoring architectures. Section 4.4.2 enriches this set of requirements with two also overlapping requirement sources. The properties relating to IT infrastructures defined in ITIL and COBIT are analysed and interpreted as requirement sources for UC-conform provision platforms. This requirement collection is enriched by the analysis of the mediation conditions specific for Utility Computing service offers. This collection of overlapping requirement sets enables the constitutive development of a core model for UC-conform provision platforms in Section 5.2 to evolve a solution approach to problem P_{1c} (cf. Section 1.3).

4.5 SUMMARISING THE REQUIREMENTS

Chapter 4 analyses in detail the settings of the problems stated in Section 1.3. This analysis reveals certain specific requirements of the UC business model on a generic life cycle for SOC service offers hosted on IaaS (cf. Section 2.6).

The analysis addresses the problems $P_{1..3}$. The settings for problem P_{1a}, that is concerned with a missing standardised usage description for data exchange between phases of a life cycle, are detailed in the Sections 4.2 and 4.3. Section 4.2 examines the relation between customer, service, and resource while elaborating the importance of a standardised usage description throughout the service life cycle. For the collected analysis results Section 5.3.2 evolves a solution approach to such a usage description.

The settings of problem P_{1b}, that is concerned with a comprehensive usage-centred data model, are detailed in Section 4.2. Section 4.2 elaborates the importance of a consistent view on the customer-service-resource relation. Section 5.4 evolves a solution approach to a data model resolving this issue.

The settings of problem P_{1c}, that is concerned with a core provision model for Utility Computing services, are detailed in Section 4.4. Section 4.4 examines the functional requirements for UC-conform provision platforms, while elaborating the importance of a generic

provision model for Utility Computing services. Section 5.2 evolves a solution approach to such a provision model.

The settings of problem P_2, that is concerned with the control of service quality in the context of SOC service offers hosted on IaaS, are detailed in Section 4.3. Section 4.3 examines the role of service level agreements, response times, and the quantitative and qualitative control aspects. In addition, this chapter examines the settings to evaluate the feasibility of business processes based on SOC service offers. Section 5.3 evolves corresponding solution approaches for the usage-centred assurance of service quality.

Summarising the previously introduced utility computing-specific requirements, the example in Section 4.2.4 reveals the importance of continuous analysis of the complex estimations about markets and their possible usage behaviour for UC service offers throughout the entire life cycle, in order to keep track of changes at the cost side of the business planning. This finding reflects the problem P_3, that is concerned with the analysis of complex service cascades. The results of the analysis in this chapter can be summarised as an additional indicator - beside the example given in Section 1.3 - for the complexity of UC service cascades. Chapter 6 evolves a simulation model to enable a corresponding analysis of complex UC service cascades.

It is known, that the control of the cost side of business planning is currently also missing due to the fact that in practice there is no detailed knowledge about the dependencies of services among each other.

5 USAGE-CENTRED PROVISION APPROACH

The requirements on Utility Computing services are elaborated in Chapter 4. Based on the previous elaborations, this chapter introduces approaches to the representation of these requirements on UC service life cycles. Section 5.2 evolves an approach to a technology-independent core provision model for Utility Computing platforms. In addition to such a core provision model, Section 5.3 evolves a set of approaches to enable a concept for usage-centred assurance of service quality. A corresponding usage-centred data model is evolved in Section 5.4. Section 5.5 demonstrates the interaction between the previously introduced approaches and the life cycle of a UC service offer.

5.1 RESEARCH METHODOLOGY

In Section 2.1.1, the hypothesis is introduced that an improved representation of the core relation of Utility Computing within a generic service life cycle leads to an all in all more cost efficient service provision. In this chapter, a core provision model and a concept for usage-centred assurance of service quality are modelled. Chapter 6 adds a resource and cost simulation model based on these outcomes.

As preparation of the modelling, chapter 4 analyses the requirements in the context of this thesis.

Section 5.2 consolidates the requirements on provision platforms retrieved in Section 4.4 as base for a core provision model. These primary requirements are to be used afterwards to derive provision components as logical functional groups. The core workflows between these provision components are then to be derived from the primary requirements in interaction with the functionalities of the introduced components.

In Section 5.3, a concept for usage-centred assurance of service quality is elaborated based on the requirements gathered in Section 4.3. Based on the work of Liang et al. (2005, 2006), the level of usage is introduced as starting point for the characterisation of usage in the context of provision quality. The quantitative approach of usage pattern is complemented with an approach for a decision tree, in order to enable economic request routing. These approaches are continuously evolved into an approach for business service level. The

evolved approaches offer the ability to introduce a new approach for the feasibility rating of service cascades. This approach is evolved based on the previous work in this section.

Section 5.4 introduces an approach for a usage-centred data model based on the evolution of the work in Section 5.3 and feedback from business partners. The core of the model is represented by the previously introduced business service levels. Also, the model links the approaches evolved in Section 5.2 and Section 5.3 logically with each other.

5.2 CORE PROVISION MODEL

Based on the requirements defined in Section 4.4, Section 5.2.1 consolidates the primary requirements providing a single source for further processing. Resulting, two use cases are derived from the collected requirements. One, describing the service consumption in provision platforms, the other, specifying the requirements on service provision for UC platforms.

The basis for Section 5.2.2 are the previously generated use cases. Within the section, a minimum set of abstract provision components, representing the actors of the evolved core provision model, is derived.

In Section 5.2.3, a minimum set of interactions between these provision components is described. The interactions are grouped into three workflows that specify the essentially necessary interactions between the defined provision components. These workflow descriptions complete the core provision model for UC platforms.

The evolved approach to a core provision model offers a generic provision model for Utility Computing services, as demanded, in order to solve problem P_{1C}, specified in Section 1.3.

5.2.1 *Consolidation of Primary Requirements on Provision Platforms*

In the following, the requirements introduced in Section 4.4 are consolidated into a single source of requirements on UC-conform service provision.

Service Consumption Use Case

- Discovery
  - Req_{01}: Looking up service providers by service types;
  - Req_{02}: Selecting a reference to a certain service provider;

  - Req_{03}: Providing geographical information about the processing location (e.g., local data centre, remote data centre, trusted partner service, or public third-party service);
  - Req_{04}: Providing price scales for request processing.

- Brokering and load-balancing
  - Req_{05}: Looking up service providers by service instances;
  - Req_{06}: Finding the most suitable resources for a service request, respecting processing costs;
  - Req_{07}: Queuing requests when forwarding is currently not possible;
  - Req_{08}: Applying access control and quotas for a selected resource;
  - Req_{09}: Forwarding requests to their selected resources and authenticate requests at a resource;
  - Req_{10}: Preventing SLA violations by dynamically forwarding requests to SLA-conform resources;
  - Req_{11}: Preventively invoking additional service instances to increase the pool for SLA-conform request processing;
  - Req_{12}: Destroying underutilised service instances.

- Orchestration
  - Req_{13}: Exposing orchestrated services as new services;
  - Req_{14}: Providing accounting information for orchestrated services;
  - Req_{15}: Providing SLA information (e.g., monitoring information) for orchestrated services;

- Authentication and authorisation
  - Req_{16}: Authenticating service consumers submitting service requests;
  - Req_{17}: Authorising submitted service requests;
  - Req_{18}: Selecting the corresponding policies (e.g., event, error, security, and brokering policies) associated with a specific service customer;
  - Req_{19}: Enabling the storage of authentication information to access third-party services.

- Monitoring, metering, and accounting
  - Req_{20}: Considering monetary penalties for SLA violations;
  - Req_{21}: Monitoring resource usage on processing systems for runtime analysis:
    - Overall system processor, memory, storage, and network usage;
    - Processing duration, processor, memory, storage, and network usage per service request;
    - Deployed service types and running instances;
  - Req_{22}: Monitoring the life cycle of service instances per service type:
    - Overall feasibility per service type;
    - Individual service instance deployment, state, and destruction;
  - Req_{23}: Calculating price based on resource usage per request;
  - Req_{24}: Storing gathered information for offline analysis;
  - Req_{25}: Attaching billing information to request responses;
  - Req_{26}: Providing an interface for access to the gathered information.
- Fault handling and logging
  - Req_{27}: Providing failure notification to service consumers;
  - Req_{28}: Handling instructions given by the error policies;
  - Req_{29}: Logging error information for troubleshooting.
- Policies
  - Req_{30}: Brokering-specific policy properties defining resource usage quotas per service customer (including the SLA properties to specify the tolerances in transport and processing delays and enable the control of the service level throughout all parties in a service cascade);
  - Req_{31}: Error-specific policy properties defining actions for the autonomous management including provisioning and failover, whereby the following types of errors should be supported:
    - Request failed unexpected (request was well formed, but its processing failed);

    - Request format unsupported (request was not well formed);
    - SLA violation (request was executed, but at least one SLA property has been violated during processing);
    - Resource unavailable (request was executed, but ran out of some resource, e.g. memory or storage);
    - Consumer unreachable (request was successfully executed, but response is undeliverable);
    - Request timeout (provider-side queuing time expired, without processing resource being available);
  - Req_{32}: Event-specific policy properties defining actions for autonomous management including provisioning and load-balancing like resource overused and resource underutilised;
  - Req_{33}: Security-specific policy properties defining location, authentication, and authorisation for customers and their service requests, as listed below:
    - Authentication: Consumer known and trusted;
    - Authorisation: Request invoked by an authenticated consumer and consumer allowed to use a specific service offer;
    - Location: Request processing restricted to classified resources (e.g., provider owned resources, trusted partner resources, or public resources).

The model does not aim to support service request scheduling (e.g., submitting jobs to the model with timing information attached).

Submitted service requests are processed as soon as possible with respect to their policies.

The model does not aim to support mixed legacy application usage with the possibility to use the same resources in SOA and legacy fashion.

The model does not aim to support licence management for third-party services or embedded legacy applications.

Service Provision Use Case

- Data access

  Req_{34}: Providing access to all necessary user data for deployed service instances, in order to enable processing of service requests.

- Provisioning
  - Req_{35}: Deploying of at least one service instance prior to actual demand through incoming service requests, with the deployment including resource selection, instantiation based on the given files, and insurance of user data access;
  - Req_{36}: Announcing of a service instance in the service directory after deployment;
  - Req_{37}: Deployed service instances being removable at runtime;
  - Req_{38}: Service implementations supporting different deployment states like *online* (ready for incoming requests), *offline* (service is ready to be removed form a resource), and *standby* (service is ready for transfer to the online state).
- Embedded legacy applications

  Req_{39}: Enabling the ability to include non-model conform application functionalities, in order to support mixed environments with existing legacy applications.
- Synchronous and asynchronous usage

  Req_{40}: Service offers being available for synchronous or asynchronous usage by service consumers.
- Administration
  - Req_{41}: Interfaces enabling full automation of all actions demanded for administration, in order to connect autonomous management systems enabling features like self-organisation;
  - Req_{42}: Giving interfaces, libraries, or instructions to enable the migration of service offers from non-SOA environments to SOA service offers;
  - Req_{43}: Assuring that upgrades of software providing the environment does not lead to service outages.
- Policies

  Req_{44}: Deployment-specific policy properties defining the automated resource selection for service deployment, like the number of online or standby instances per service level and/or geographical location, including corresponding rules to determine these numbers, and actions to ensure the adequate deployment.

5.2.2 *Derivation of Provision Components*


Component Model

This section examines the collected requirements in Section 5.2.1 for their qualification for a core provision model. As selection criteria, the requirement has to enable the service provision conform to the definition of UC service offers in SOC architectures hosted on IaaS, defined in Section 2.6. To balance the acceptance of a requirement as part of the minimal set, the necessity of the requirement is also checked in comparison to the workflows evolved in Section 5.2.3. As there is no comparable predefined minimal set, this thesis introduces the selection given in the following as proposal for such a minimal set of requirements on UC-conform services.

After their selection, the requirements are grouped by functionality within an abstract network of interacting components. This structures the requirement collection without binding the provision components to a specific technical environment.

The derived provision components presented in the following are also published by Heckmann (2007).

Service Instance

- Functionalities
  - Instance of a service type that can handle multiple service requests simultaneously;
  - Exists within the context of a certain service host;
  - Applies SLA quotas (implements Req_{08});
  - Supports the states online, standby, and offline (implements Req_{38}).
- Properties
  - Processor usage of instance (implements Req_{21});
  - Memory usage of instance (implements Req_{21});
  - Storage usage of instance (implements Req_{21});
  - Network usage of instance (implements Req_{21});
  - Events like errors or state changes (implements Req_{22}).

Service Host

- Functionalities

  Hosts can only host one service instance of a certain service type at a time.

- Properties
  - Processor usage of host (implements Req_{21});
  - Memory usage of host (implements Req_{21});
  - Storage usage of host (implements Req_{21});
  - Network usage of host (implements Req_{21});
  - Service instance states (implements Req_{21});
  - Service types deployed (implements Req_{21});
  - Events like errors or state changes (implements Req_{22}).

Service Type

Service types offer the functionality to define a certain class of service with distinctive business functionality and a standardised public interface.

Service Consumer

- Functionalities

  Invoking service instances by sending service requests.

- Properties

  Offering authentication information for service requests.

Service Request

- Functionalities
  - Invoking a service instance;
  - Including an associated synchronous or asynchronous service request response (implements Req_{40}).
- Properties
  - Service request bill[1] (implements Req_{25});
  - Service request events like error messages or event messages (implements Req_{27});
  - Service request response time[2] (implements $Req_{21,\ 30}$);

1 Service request responses should contain the request-specific costs that are going to be billed by the service provider.

2 Time period between the first occurrence of an incoming service request within the provision environment of a service provider and the outgoing service request response finally leaving the provision environment

  - Service request execution time[3];
  - Service request queuing time[4].

Service Registry

- Functionalities
  - Authenticating service consumers (implements $Req_{16,\ 33}$);
  - Authorising service requests (implements $Req_{08,\ 09,\ 17}$).
- Properties
  - Service types available (implements $Req_{01,\ 36}$);
  - Service type price scales (implements Req_{04});
  - Service type broker (implements Req_{02});
  - Third-party service type usage information like authentication data (implements $Req_{05,\ 19}$);
  - Service type deployment files (implements Req_{35});
  - Service type load-balancers like local, trusted-partner, public third-party (implements Req_{03});
  - Service load-balancer costs information.

Service Broker

The functionalities provided by service brokers can be summarised as economical load-balancing and accounting management, defined as:

- Forwarding service requests to the economically most suitable service load-balancer or third-party service broker (implements $Req_{06,\ 09,\ 10,\ 18,\ 30}$; implements *Provisioning Factors*, cf. Section 4.4.3), respecting all given policies;
- Generating service request bill per request, including third-party service utilisation costs and SLA penalties (implements $Req_{14,\ 20,\ 23}$).

Service Load-Balancer

- Functionalities
  - Representing a single cost domain (implements *cost domains*, cf. Section 4.4.3);

3 Time period for the service request processing at a service instance including generation of a service request response

4 Cumulated time periods in which a service request is being queued on its way through the provision environment towards and back from a service instance

  - Queueing service requests, if necessary (implements Req_{07});
  - Forwarding service requests to the technically most suitable service instances, respecting all given policies (implements $Req_{08,\ 10}$);
  - Managing (e.g., deploying, activating, deactivating, or removing) the service instances on the controlled service hosts on demand (implements $Req_{11,\ 12,\ 28,\ 31,\ 32,\ 37}$).
- Properties
  - Service instances running;
  - Service instances deployed;
  - Service hosts managed.

Service Monitoring

Service monitoring offers the functionality to monitor policy-related metrics per service request. The following properties are expected.

- All usage metrics from other model components - historical and real time metrics - retrievable from the service monitoring archive (implements $Req_{24,\ 26,\ 29}$);
- Usage metrics of service types, accumulated as index of the overall resource usage per service type for all registered service types, including third-party offers (implements Req_{15}).

Network Connector

Network connectors offer the functionality of an undirected connection between two components of the model. As property, the network connector offers the usage metrics of the network layer.

Service Network

A service network offers the functionality of grouping a finite set of connected model components.

Brokering Policy

The properties provided in the brokering policy are defined per service consumer:

- Information to access the user data (implements Req_{34});
- Maximum latency for service request responses (implements Req_{30});
- Maximum number of concurrent service requests.

Error Policy

The properties provided in the error policy are defined per service consumer (implementing Req_{31}):

- Unexpected failure actions;
- Request format failure actions;
- SLA violation actions;
- Resource unavailability action;
- Consumer unreachable action;
- Request queuing timeout action.

Event Policy

The properties provided in the error policy are defined per service type (and implement Req_{32}). The properties specify the two types of actions to take when resources are underutilised or overloaded.

Security Policy

The properties provided in the error policy are defined per service consumer (and implement Req_{33}). There is one property that specifies the authorised service consumers. And there is another property that specifies the allowed processing and storage locations for service requests and corresponding user data.

Storage Network

User data is stored near its creation or processing location. User data is accessible directly through its storage location.

Excluded Requirements

As they do not represent parts of the minimal set of requirements necessary to provide UC-conform services, the below listed requirements are not considered for the core provision model:

- Embedded legacy applications (Req_{39});
- Administration-related requirements ($Req_{41\text{-}43}$);
- Deployment policies (Req_{44}).

5.2.3 *Derivation of Core Workflows Between Provision Components*

This section describes a minimal set of interactions between the provision components introduced in Section 5.2.2, that enables the provision of UC services in a core model. The interactions are analysed corresponding to and grouped by the three workflows that this thesis estimates as essential for the process of service provision. In this context, the minimal set of interactions is defined as these interactions which enable the provision of service offers conform to the primary requirements defined in Section 5.2.1.

The derived provision interactions presented in the following are also published by Heckmann (2007).

1_SSC — Simple Service Consumption Workflow

The *simple service consumption* workflow describes the minimal set of interactions between the provision components, that processes an atomic service request with a single resource domain given.

- Initial broker lookup

  Name: s-01

  Request flow: service consumer → service registry

  Request data: service consumer authentication and service type

  Response data: service broker and service price scales

- Request transmission

  Name: s-02

  Request flow: service consumer → service broker → service load-balancer → service instance

  Request data: service consumer authentication and service request

  Response data: service request state, service request response, and service request bill

- Request authentication

  Name: s-03

  Invoked by: s-02

  Request flow: service broker → service registry

  Request data: service consumer authentication and service request

  Response data: service authorisation and brokering policy

- Load information retrieval

  Name: s-04

  Invoked by: s-02

  Request flow: service load-balancer → service monitoring →service host

  Response data: host usage of *processor, memory, storage, and network*, deployed service types, and service instance modes

- Load information delivery

  Name: s-05

  Invoked by: s-02

  Request flow: service instance → service monitoring

  Request data: instance usage of *processor, memory, storage, and network* and instance events

- Service bill archiving

  Name: s-T

  Invoked by: s-02

  Request flow: service broker → service monitoring

  Request data: usage information of third-party services and service bill

Workflow step s-01 is optional.

Workflow step s-T terminates each evolved workflow (1_SSC, 2_CoSC, and 3_CaSC) as closing step.

Workflow 1_SSC is illustrated in Figure 27.

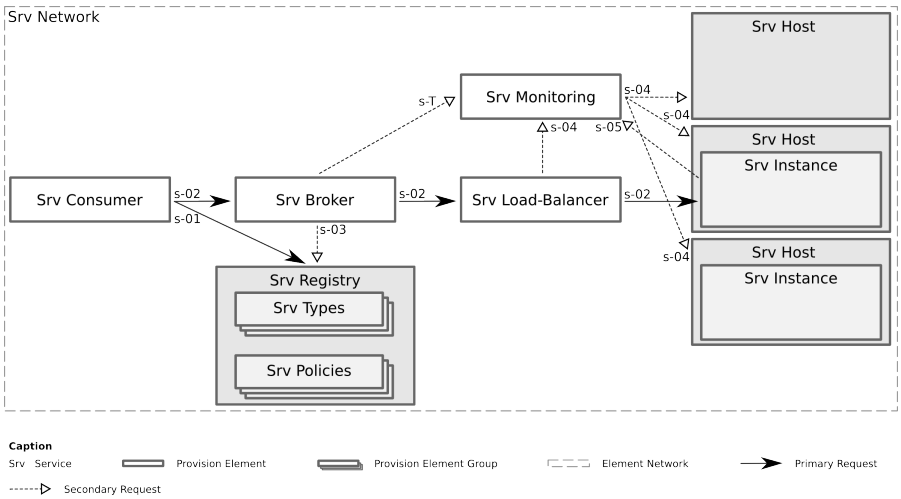


Figure 27: Illustration of Workflow 1_SSC

2_CoSC — Complex Service Consumption Workflow

The *complex service consumption* workflow describes the minimal set of interactions between the provision components, that processes an atomic service request with multiple resource domains given.

The 2_CoSC expands the 1_SSC by one workflow step - named s-06 - invoked by the second 1_SSC step, s-02. The step determines the utilisation of each given resource group represented by a load-balancer and is called *determine domain utilisation*. Its request flow starts at the service broker and ends at a service load-balancer. The data given with the request is the service type of the service request to be processed. The data expected as response is the service type utilisation per load-balancer.

Workflow 1_SSC is illustrated in Figure 28.

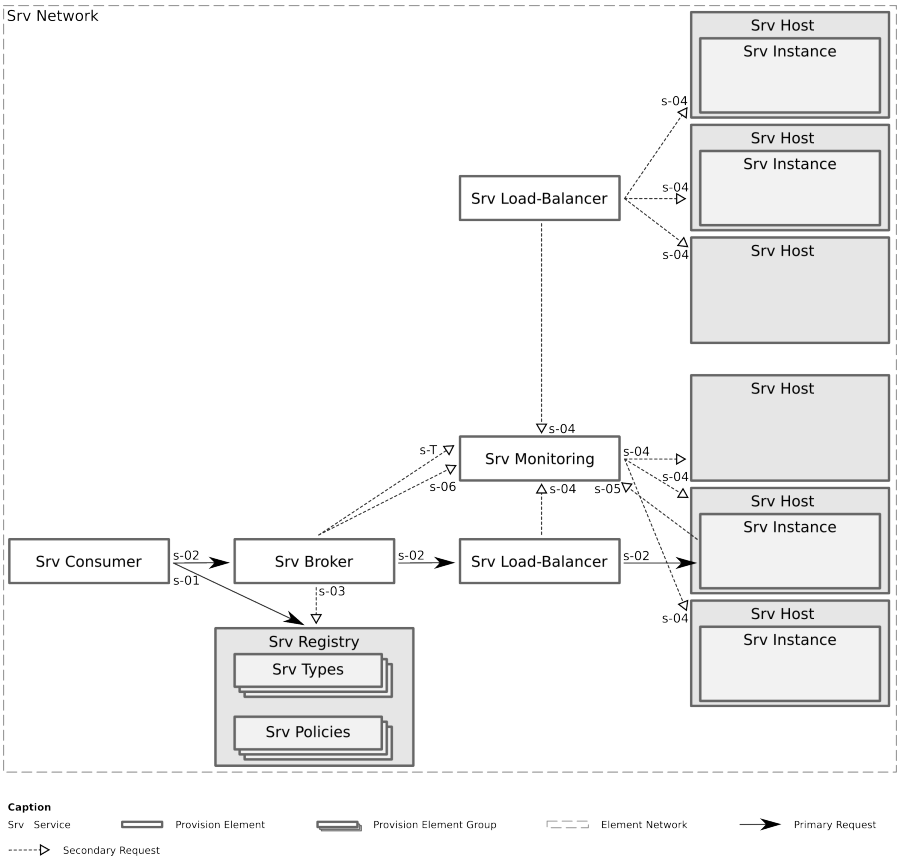


Figure 28: Illustration of Workflow 2_CoSC

3_CaSC — Cascaded Service Consumption Workflow

The *cascaded service consumption* workflow describes the minimal set of interactions between the provision components, that processes a cascaded service request with multiple resource domains given.

The 3_CaSC expands the 2_CoSC by two workflow steps invoking sub-requests to external service offers. The scene setting assumes that performing sub-requests to external service offers is more complex compared to the invocation of requests addressing internal service offers. Further, it is assumed that being able to handle the more complex external sub-requests also enables to handle internal sub-requests.

- External sub-request transmission

  Named: s-07

  Invoked by: s-02

  Request flow: service instance (internal) → service broker (internal) → service broker (external) → service load-balancer (external) → service instance (external)

  Request data: service consumer authentication (added by internal service broker) and service sub-request (invoked by internal service instance)

  Response data: service request state, service request response, and service request bill (removed by internal service broker)

- External load information retrieval

  Named: s-08

  Invoked by: s-07

  Request flow: service broker (internal) → service broker (external) → service monitoring (external)

  Request data: service type

  Response data: utilisation metric[5]

Workflow 1_SSC is illustrated in Figure 29.

5 The provision of a utilisation metric shall enable the internal service broker to determine the available service capacity to be included in its forwarding decision. There might by scenarios given where service providers may not intend to provide such metrics. Therefore, the provision of a utilisation metric is optional.

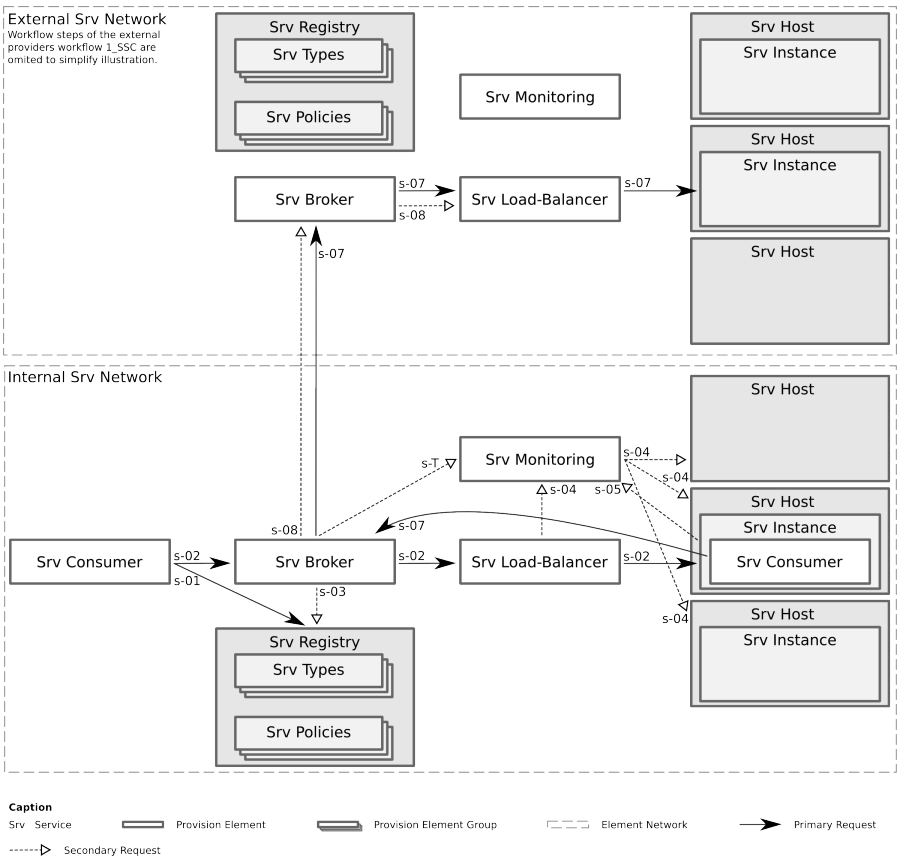


Figure 29: Illustration of Workflow 3_CaSC

Indirect Sub-Request Invocation

The 3_CaSC workflow implies that service instances do not invoke sub-requests directly. The indirect invocation of sub-requests mediated by the service broker prevents the model from losing control over changes in service providers or authentication credentials. Pay-per-use billing of service offers based on service cascades would be more error-prone and rise the complexity of service implementation, if billing information from sub-requests was managed by service instances themselves. In addition, the complexity of the control of load-balancing between redundant service offers would rise, if service instances invoked sub-requests directly. The model offers the service broker as component to handle the overhead of billing management and abstraction level for sub-request service providers.

5.3 USAGE-CENTRED ASSURANCE OF SERVICE QUALITY

Based on the levels of usage evolved by Liang (cf. Section 5.3.1), this section describes approaches to specify usage behaviour (cf. Section

5.3.2), decide on request routing (cf. Section 5.3.3), determine the feasibility of service cascades (cf. Section 5.3.5), and specify service levels for UC services (cf. Section 5.3.4). These approaches are consolidated in Section 5.3.6.

5.3.1 *Levels of Usage*

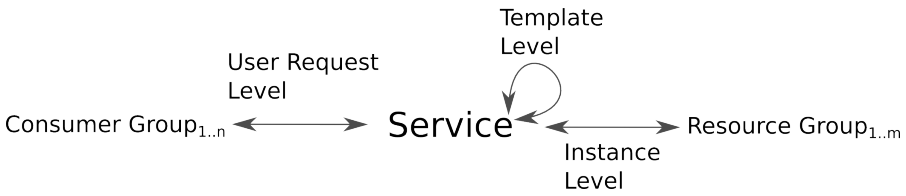


Figure 30: Liang's Levels of Usage

As an entry to the discussion on usage in the service context, the work of Liang et al. (2005, 2006) is introduced. Liang et al. evolve three points of view on service usage for the research on data mining in the field of web services.

The three levels of usage are specified as follows.

- User request level

  The user request level is concerned with the outer view on composite services. The level describes how composite services are used by consumers. On this level there is no awareness of the implementation of the service itself. Therefore, the potential complexity of a corresponding service cascade providing the service is hidden.

- Template level

  The template level of service usage deals with the inner view on composite services. The level describes the correlation of services representing an underlying service cascade of a service offer. Liang et al. characterise service cascades as flows of service dependencies which lead to a consumer-satisfying output.

- Instance level

  The instance level is concerned with the dependencies of services regarding their runtime environment. Such constraints restrict service implementation, deployment, and migration.

The three introduced levels of usage and their relation to service consumers and providers are depicted in Figure 30.

The previously introduced results of the analysis of the work of Liang et al. is also published by Heckmann (2009); Heckmann and Phippen (2010).

5.3.2 *Usage Patterns*

Usage Pattern

In Section 4.2.1, the demand for an approach to characterise the customer's usage behaviour and enable its transfer between acting parties within the service life cycle is described. In addition, in Section 4.3.1 it is illustrated that the secondary SLA criteria in UC also depend on a characterisation of customer's usage behaviour.

Based on Liang's user request and template levels, this section describes an approach to specify usage behaviour in the context of this thesis. This approach is called *usage pattern.*

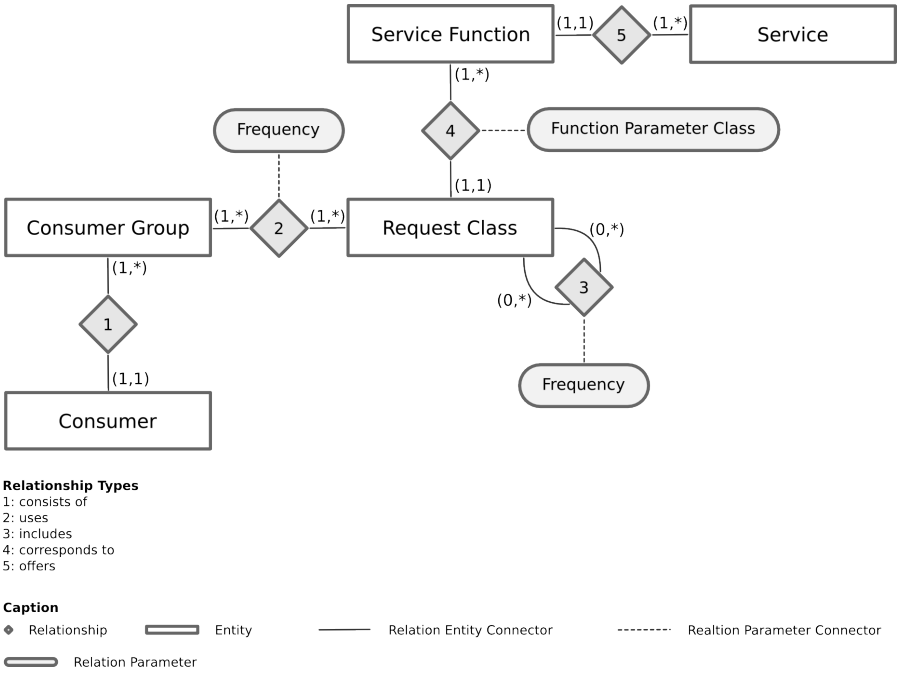


Figure 31: Usage Behaviour Description by Usage Pattern

A single usage pattern defines a quantitative approach to usage behaviour mapping (cf. Section 4.3.1) between an unlimited number of service consumers and a particular service offer by a unique service provider. In usage patterns, consumers are grouped corresponding to their usage behaviour. The behaviour groups are represented by one or more request classes. The corresponding relation includes the request frequency as attribute with equal distribution assumed. Each

request class describes a certain usage behaviour regarding one function of a service offer.

Behaviour is represented by an abstract function parameters class. Each class represents a characteristic combination of function parameter value ranges, that imply a deterministic function call behaviour. This approach assumes that for most functions the resource demand for processing a function call can be deducted from given parameter values. It is known that there are function implementations where this assumption fails.

As an example for a non-deterministic function, the calculation of the total amount of a bank account is given. Given the account number as a function parameter, it is not possible to estimate the resource demand of this calculation by evaluating the account number.

The request class may relate to any number of sub-request classes. For this recursive relation a request frequency attribute is provided with equal distribution assumed. This feature enables the specification of service cascades. It is known that the use of this feature breaks the paradigm of service abstraction (Erl, 2007). Therefore, the use of the recursive relation is optional.

The introduced relations - which altogether instantiate a usage pattern - are shown in Figure 31 using an entity relationship diagram (Chen, 1976).

The previously introduced approach to specify usage behaviour is also published by Heckmann (2009); Heckmann and Phippen (2010).

The evolved approach of the usage patterns offers a standardised usage description for data exchange between phases of a service life cycle as demanded to solve problem P_{1a} specified in Section 1.3.

5.3.3 *Decision Tree*

➡ Decision Tree

In Section 5.2.2, the service broker is introduced as essential component in UC provision platforms. This component enables the routing of UC service requests based on economic criteria in addition to resource load criteria. To enable such an economic request routing (cf. Section 4.3.1), a decision tree[6] [7] is proposed as quantitative approach.

The direct management of service requests at runtime aims to gain control over the processing resource utilisation by managing the forwarding of service requests. This thesis assumes that, next to the continuous monitoring of the utilisation of processing resources, the decision on the route of a request is the core of UC provision management. To decide on a request's route, measurable criteria that represent technical and economical aspects of the request processing must be evaluated. These criteria are collected and sorted in a decision tree. This

6 In previous work of Heckmann this approach has been called *Provisioning Factors*.

7 Despite the use of the term in data mining, machine learning, operations research, or statistics, this thesis reuses the term in its own context.

decision tree is a qualitative approach to enable provision management in the service-resource relation of UC (cf. Section 4.3.1).

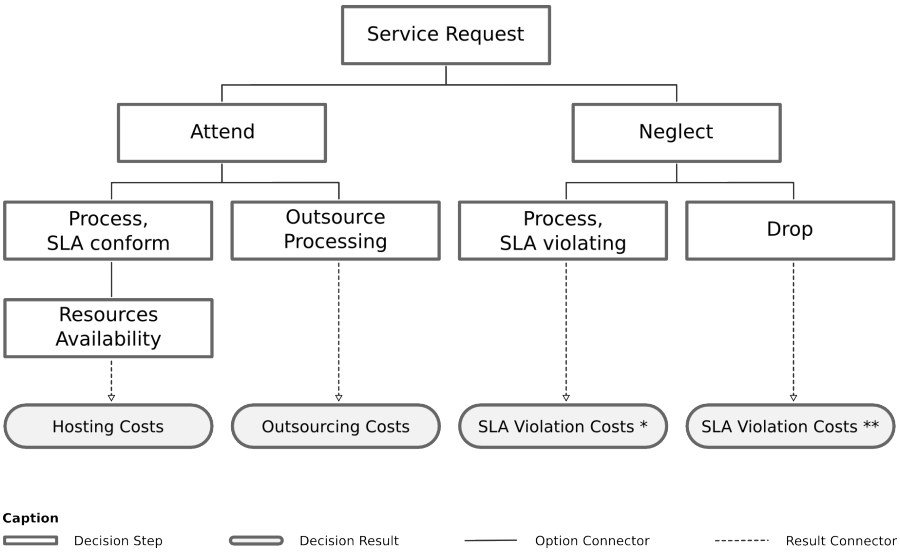


Figure 32: Decision Tree for Utility Computing Service Request Routing

The decision options in the tree can be grouped into three categories, as follows.

- Processing

  The decision option *processing* aims at calculating the costs of the processing of a specific service request on provider-owned resources. These costs are derived from fixed costs for service hosting (e.g., for server acquisition, housing, and administrative personnel) and the dynamic costs (e.g., for cooling and power consumption). Not part of this thesis is the identification of the individual combination of these fixed and dynamic costs. Previous to the decision calculation, the availability of sufficient resources to process the specific service request must be ensured. If resources are available, the costs for request processing can be estimated. This calculation also includes costs for all subrequests initiated by the specific request.

  This thesis assumes that the detailed analysis of large service cascades in order to find the optimum costs at runtime may fails in complex provisioning scenarios. This also applies to the calculation of the exact resource demands. In case of complex provisioning scenarios, this thesis suggests calculating approximations instead.

- Outsourcing

  Another decision option for the processing of service requests is the *outsourcing*. In this case, the request is not processed on provider-owned resources. Scenarios are conceivable, where it is an economical alternative to forward requests to other service providers for processing. Request outsourcing can be appropriate for all layers of a service cascade. Such scenarios can reach from the processing of customer requests on competitor sites in times of peak loads up to dynamic processor picking for back-end services (e.g., retrieval of geographical information). Regardless of the scenario, the costs for the external request processing are estimated.

- Neglecting

  In opposite to the previous groups, the decision option of *neglecting* a service request aims to calculate the costs for an intentional violation of the SLA. The intensity of the violation can range from exceeding the maximum agreed request response time up to simply dropping the incoming service request. To decide about the best option, the costs for all contracted variations of service level aberrations must be taken into account.

In all introduced decision groups, costs are calculated. In combination with the estimation of the price for a specific service request, the profit or loss of a routing decision can be estimated. All addressed costs may also vary over time corresponding to individually contracted factors (e.g., time of day or discounts on request amounts). The evolved groups are proposals fitting in the context of this thesis, but can be adapted by adding or removing options in other contexts, as needed.

The introduced relations - which altogether instantiate a usage pattern - are shown in Figure 32 using an entity relationship diagram (Chen, 1976).

The previously introduced approach to a decision tree is also published by Heckmann (2009); Heckmann and Phippen (2010).

5.3.4 *Business Service Level Agreements*


Business SLA

In Section 4.3.2, the analysis of the view points of business managers and technical operations introduces the demand for a simplification of the agreement about feasibility of service offers between business and IT. Common approaches for service level agreements

focus on technical metrics resulting from the contracting of resource reservations throughout the OSI layers (cf. Section 3.2.2). The complexity of highly meshed service cascades including redundant alternate service offers makes these approaches inefficient.

The idea of BSLAs introduces a new level of service agreements, which loosens the coupling between business and IT.

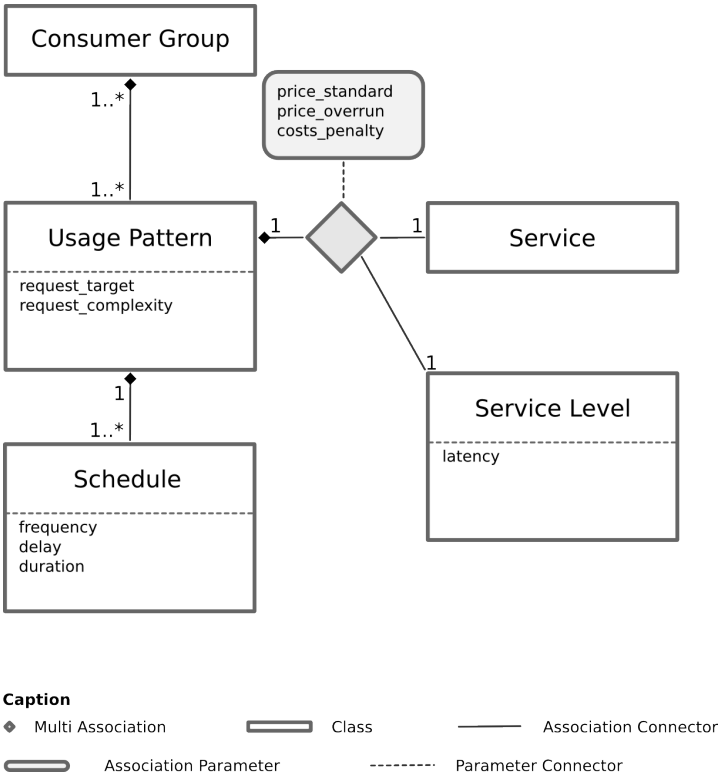


Figure 33: Simple Business Service Level Agreement

The core of BSLAs is the turn-away from the description of the committed QoS from the point of view of service providers. BSLAs specify the expected QoE from the point of view of a service consumer. To achieve this goal, BSLAs start with the description of the expected usage behaviour of a service consumer. For example, a customer consists of a group of ten service consumers. Approximately, each consumer sends one service request per second. In this simplified example, also the request complexity is approximately constant throughout the consumer group. These quantitative service quality metrics can be represented using the previously introduced usage pattern (cf. Section 5.3.2). The remaining relevant mandatory technical service level metric is the maximum latency of service request responses, as introduced in Section 4.3.1. The combination of usage pattern and specification of the maximum tolerated latency of ser-

vice request responses represent the core of the BSLA approach. This core can optionally be enhanced by the declaration of maintenance windows, maximum downtimes, fines, prices, or other service level attributes.

The approach of BSLA changes the focus of the agreement on service levels from technical thresholds to the usage behaviour description as relevant metric. Thus, BSLAs introduce a new abstraction layer by reducing the demand for technical metrics in service level agreements to the specification of the tolerable response time limit. Figure 33 illustrates a simple characteristic of BSLAs in a Unified Modeling Language (UML) diagram (OMG, 2011). The diagram details the BSLA relation between a consumer group and a service. This relation consists of price and cost attributes combined with a usage pattern class and a service level class. The usage pattern class offers attributes for the requested service type and method, called *request_target*, and an abstract indicator for the request complexity regrading its processing. The service level class offers a latency attribute. To specify the timely distribution of the service usage, the usage pattern relates to the scheduling class. This class offers attributes to detail the request sequences in terms of request frequency, delay between requests, and sequence duration.

The previously introduced approach to Business Service Level Agreements is also published by Heckmann et al. (2011, 2012a).

5.3.5 *Feasibility Rating for Service Cascades*

Based on the analysis of multi-tier SOC infrastructures operating common business processes in Section 4.3.2, this section evolves an approach to rate the feasibility of such infrastructures hosted on IaaS.

Rating feasibility starts with an agreement. The previously evolved approach to agree upon service levels based on the description of the expected usage of service offers - BSLA - is suitable to agree on the feasibility of service offers. Using BSLAs to contract on service offers introduces *contracted usage* as metric to analyse feasibility. As opposite metric to the contracted usage, the *monitored usage* reflects the IT infrastructure's current request load and resource utilisation per group of consumers addressed by the given BSLA. Comparing the monitored with the contracted usage indicates the workload of the contemplated consumer group, assuming the technical availability of all demanded infrastructure components. Therefore, the resulting metric can be used to lead back the workload of business processes that are dependent on the contracted service offer.

Contracted usage consists of the specification of the planned request load that is recorded in the BSLA and a corresponding estimation of the expected resource demand that is related to a specified service request type and complexity. This resource demand has to be estimated by the service provider. Approaches to ease the estimation of the resource demand of service requests are not examined in this thesis.

Monitored usage depends on the monitoring of resource utilisation, which can be done using common monitoring systems, and the monitoring of the current request load per consumer group. This thesis proposes the previously introduced service broker (cf. Section 5.2.2) as component to implement the monitoring of request load.

The service broker consolidates the information, in order to calculate the contracted and monitored usage metrics and the resulting workload metric. Based on this information combined with the technical availability of the demanded components, the service broker can estimate the feasibility of service offers for certain consumer groups. This approach eliminates the demand to track down the exact resource consumption for each dedicated service request, in order to estimate the feasibility of service cascades.

To enable the estimation of technical availability of the demanded components, the relevant components have to be identified and their technical monitoring information has to be consolidated. To reflect the functional dependencies between components in an IT infrastructure, a *topology graph* is introduced. Topology graphs consist of the infrastructure components retrieved from a Configuration Management Database (CMDB)[8].

If a CMDB is not available, other sources - like plain Extensible Markup Language (XML) files (Bray et al., 2012) - can be used to represent the IT infrastructure. For example, a prototype implementation to evaluate the introduced approach is based on a simple CMDB mapped in a plain XML file (cf. Heckmann et al. (2011, 2012a)).

Redundant service offers are mapped as *service lines* within a topology graph. Service lines are logical groups of infrastructure components that together provide a service offer. Service offers are represented by service cascades on the service instance layer in Figure 34.

In IT infrastructures, *component categories* are logical groups of components with identical functionalities, such as application servers providing the function of hosting service instances for example. Component categories enable the consolidation of resource utilisation across service lines, as illustrated in Figure 34.

To estimate the resource utilisation within a topology graph, the nodes within the graph are enriched by technical monitoring information. The enriched graph is introduced as *availability graph*. The granularity of the added monitoring information can vary. The levels of variation are evolved in the following.

8 A CMDB is a repository of all IT infrastructure components referred to in the context of ITIL.

- State-based availability

  The state-based availability of a graph node is derived from the interpretation of state-related technical monitoring data. The information about the state of a represented infrastructure component could - for example - be retrieved by analysing the ICMP ping[9] history stored in a technical monitoring system. The interpretation of state-based data is limited to two simple states: available or non-available. State-based availability is technically simpler to be imposed than the subsequent variations, but is less significant for the deduction of the feasibility of a component.

- Load-based availability

  Load-based availability of a graph node is derived from the comparison between its current resource utilisation and its maximum resource capacity. The resource utilisation metric is calculated based on a customisable combination of load-related technical monitoring data (e.g., for server hardware the combination of processor load, memory capacity, and storage space metrics are proposed in this thesis). The interpretation of the resulting utilisation metric enables the provision of proportional result values reflecting the current availability of a specific infrastructure component (e.g., 20 % resource utilisation of a server). Load-based availability is technically more complex to be imposed than the state-based variant, but enables more significant deduction of the feasibility of a component.

- Mixed-mode availability

  In mixed-mode graphs the availability of a node can be determined by either the state-based or the load-based variant. Mixed-mode graphs offer a customisable compromise of administration effort and feasibility deduction precision by combining both previous variants.

A service broker estimates a service cascade as feasible, when all state-based nodes of at least one service line are available in the availability graph and when the resource utilisation of all load-based component categories offer sufficient reserves to process the *estimated usage*. Estimated usage is introduced as a representation for the expected usage in a given time frame, which results from the difference between the monitored usage and contracted usage. In availability graphs that are

9 Ping provides the ability to monitor the technical network interface of a remote system. Ping is specified in the Internet Control Message Protocol (ICMP) (Postel, 1981).

exclusively state-based, only the request load is taken into account when calculating the estimated usage. In other respects, the resource utilisation is also incorporated.

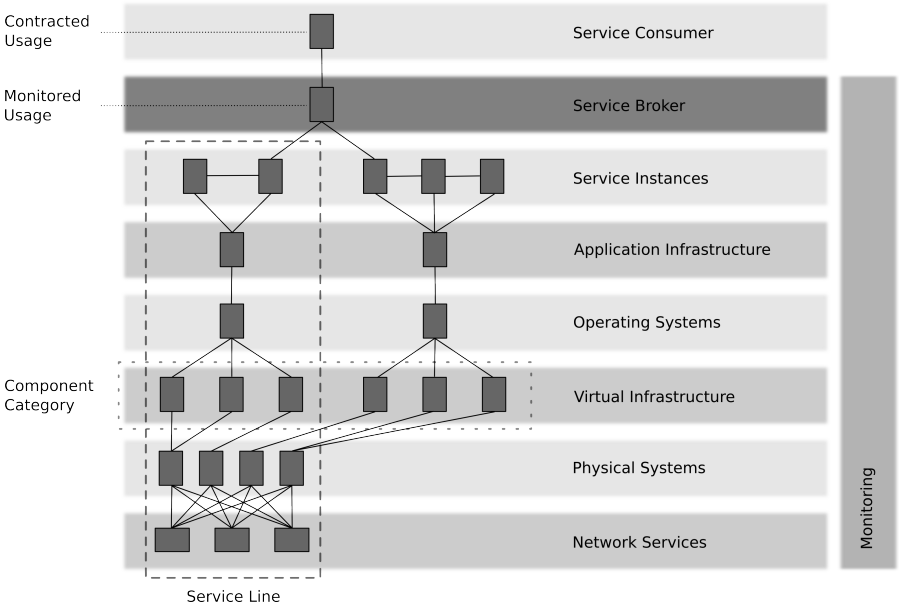


Figure 34: Feasibility Rating for Service Cascades in Generic Multi-Tier SOC Architectures Including Redundant Service Offers

In summary, this thesis proposes the integration of a new abstraction layer for advanced request routing into multi-tier infrastructures, as shown in Figure 34. The service broker enables the consolidation of different kinds of metrics. Based on the consolidated information, the feasibility of a service cascade can be estimated. This approach releases the customers from the necessity to estimate metrics outside their subject areas, like the memory consumption during service usage. On the other hand, the approach introduces potential for service providers to develop approaches for an efficient use of their resources, while maintaining contracted service levels.

The previously introduced approach to feasibility rating for service cascades and dependent business processes is also published by Heckmann et al. (2011, 2012a).

5.3.6 *Concept for Usage-Centred Assurance of Service Quality*

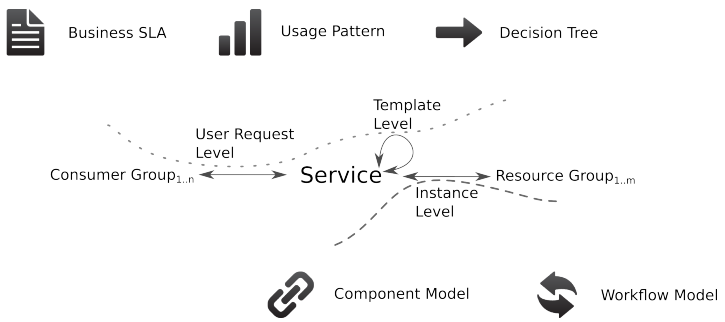


Figure 35: Coverage of Liang's Levels of Usage

Summarising Sections 5.2 and 5.3, the concept for usage-centred assurance of service quality proposed in this thesis consists of five elements. Beginning from Liang's levels of usage presented in Section 5.3.1, the user request level and the template level of service usage are addressed by the approach of BSLA (cf. Section 5.3.4), including usage pattern (cf. Section 5.3.2), and the evolved decision tree approach (cf. Section 5.3.3). BSLAs are building the basis for the agreement and control of service quality in UC environments (cf. Section 5.3.5). Usage patterns are the core of BSLA. Together they provide the ability to map usage behaviour for consumption management in UC, as demanded in Section 4.3.1. In addition, the decision tree enables economic request routing as basis of the provision management.

The component model (cf. Section 5.2.2) and the corresponding workflow model (cf. Section 5.2.3) address the instance level of service usage. The combined models describe the minimum set of properties a platform has to offer for services, in order to enable core UC features (cf. Section 5.2.1).

All these approaches combined, the agreement and control of service quality is enabled, as demanded in Section 4.3. The mapping of the previously summarised findings to the levels of usage of Liang is illustrated in Figure 35.

The evolved concept for usage-centred assurance of service quality offers control of service level agreements, response times, and the quantitative and qualitative control aspects in the context of SOC service offers hosted on IaaS, including the feasibility rating of service cascades, as demanded to solve problem P_2 specified in Section 1.3.

5.4 USAGE-CENTRED DATA MODEL


Data Model

This section evolves a usage-centred data model as basis for the documentation and analysis of the customer-service-resource relation. The data model enables the data exchange between phases of the life cycle and a technical implementation of the previously introduced core provision model for UC services. Thus, the introduced data model acts as an anchor for the provision workflows between the components of the core provision model. The data model enables the technical or economical optimisations of resource usage at runtime.

To evolve the data model, this section transforms the findings in the concept for usage-centred assurance of service quality (cf. Section 5.3) in the context of the core provision model (cf. Section 5.2) into data elements and corresponding interrelations in Section 5.4.1.

Section 5.4.2 introduces a concept for the integration of the data model into the core provision model.

5.4.1 *Data Model Specification*

The relationships between a certain service provider and its customers are modelled with a data structure using a UML diagram. Such a data structure is subsequently used as the basis for optimisations and must be modified only when the relationships between the service provider and its customers change.

Customers can have one or more contracts with a service provider. A contract covers one or more consumer groups (e.g., branches or departments) within the customer organisation. Each consumer group relates to one or more usage patterns, which can be reused for the description of the behaviour of other consumer groups. A usage pattern consists of the attributes service request target and complexity and is associated to one or more schedules, in order to time the request frequency. Beside the request frequency, a schedule includes attributes for the delay prior to a request sequence and for the duration of the sequence. Each usage pattern is associated with one service and two service levels. This association includes the pricing information (per unit) for standard request processing and for cases of overrun processing. Also, the association between usage pattern and service contains an attribute for the contract penalty (per unit). Thereby, the first service level in the association relates to the standard pricing, penalty costs, and contracted usage pattern. The second service level specifies the commitment for service requests over and above the contracted

usage pattern. The second service level relates to the pricing for overrun scenarios, while penalties are excluded in this case.

Services act as connector between the resource demands expressed by usage patterns and one or more resource groups. Also, resource groups may host an unlimited number of services. In addition, a service is associated to a deployment set, a backup specification, and service levels. Deployment sets contain the necessary files and configuration properties to deploy service instances. The additional association between one or more service levels enables the modelling of services at differing service levels. Service levels serve as abstract categorisation, in order to differentiate varying level of service quality to be expected when using a certain service offer. To enable this, a service level includes an attribute to specify the maximum tolerated latency for service request responses. In addition, service levels are associated - beside the previously introduced relations - to one or more geographical location, an availability class, and one or more backup specifications.

Resources are organised in groups. The primary grouping criterion for resources is technical (e.g., the virtualisation software used). As second criterion, the geographical location (e.g., the hosting data centre) is used. Resource groups include an attribute to specify the costs of the resource usage (per unit). Linked to a resource group are one or more technical interfaces for its administration and monitoring (e.g., virtualisation management APIs). Additionally, each resource group is associated with an availability class. Availability classification enables an abstract categorisation of resources corresponding to their estimated level of technical availability.

The previously described relations are elaborated in the usage-centred data model in Figure 36.

The evolved data model includes the previously introduced approach of BSLA, defined in Section 5.3.4.

The previously introduced approach to a usage-centred data model is also published by Heckmann et al. (2012b).

The evolved approach to a comprehensive usage-centred data model offers a consistent view on the customer-service-resource relation, as demanded, in order to solve problem P_{1b}, specified in Section 1.3.

5.4.2 *Core Provision Model Integration*

The data model is stored in the service registry component of the previously introduced component model (cf. Section 5.2.2). For service providers the information extractable from the data model enables the economical optimisations of resource usage. This optimisation can

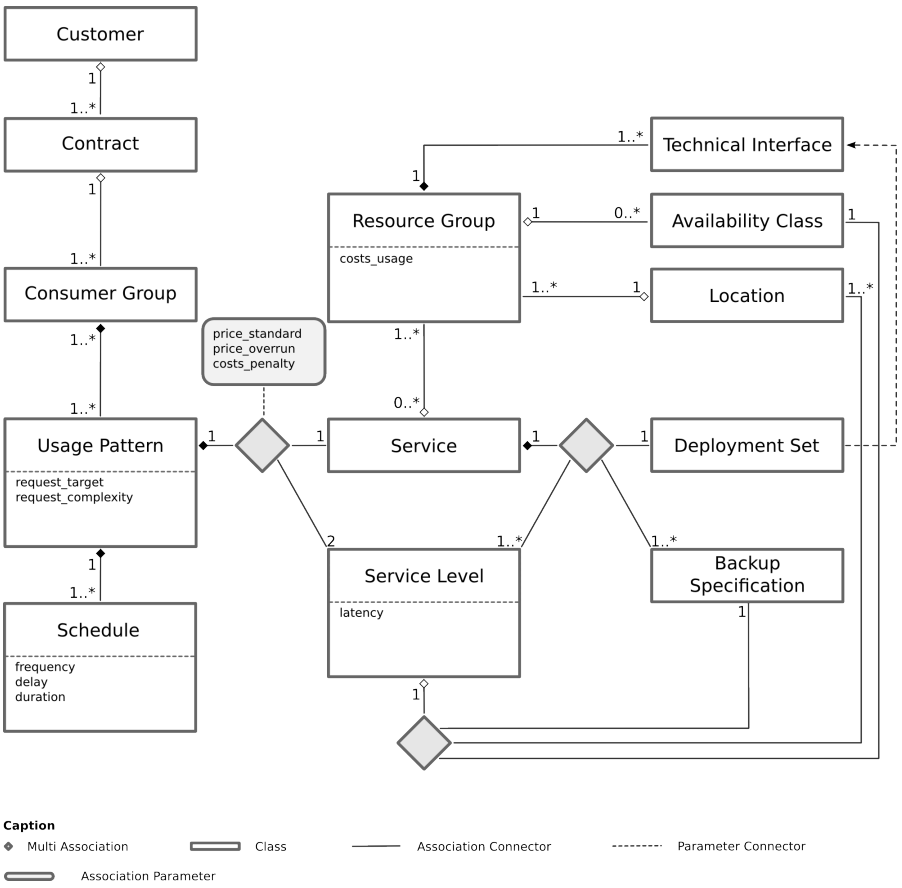


Figure 36: Usage-Centred Data Model

be done at time of business planning based on estimated data or at runtime based on monitored data. In case of runtime optimisation, the data model has to be evaluated by the service broker. The service broker is previously introduced in Section 5.2.2 as a component to decide about the economical routing of service requests. This routing decision comprehends incoming and outgoing requests from the point of view of a service provider. The evolved data model only addresses the relations between stakeholders, in order to handle incoming service requests. To enable the economical runtime optimisation of incoming service requests, the usage-centred data model is integrated into the service broker together with the decision tree evolved in Section 5.3.3. Enabled by the decision tree and the information extractable from the data model, the matching between a customer's service request and a suitable resource is done in the following six steps.

Preconditions are a given service request and at least two resource groups.

1. Service type, service consumer and the service level corresponding to the service request are determined.

   Postcondition 1: identifiers for service type, service consumer, and service level are known.

   Precondition 2: service request and service type are known.

2. Resource demand for the service request is estimated.

   Postcondition 2: service request's resource demand is known.

   Precondition 3: service request's resource demand, service type, and service level are known.

3. Pools of resource groups are selected by available resources and matching service level.

   Postcondition 3: two pools of resource groups are known, where each resource group offers enough resources for request processing and one pool complies with the demanded service level and the other does not.

   Precondition 4: service request's resource demand, service type, and service consumer are known.

4. The estimated revenue per pooled resource group for request processing is calculated.

   Postcondition 4: per given resource group the estimated revenue is known.

   Precondition 5: service request's resource demand, service type, service consumer, and service level are known.

5. Estimated costs for service level violation (latency exception and request failure) are calculated.

   Postcondition 5: estimated costs for latency exception and request failure are known.

   Precondition 6: two pools of resource groups with sufficient processing resources distinguished by service level compliance, estimated revenue per pooled resource group and estimated costs for latency exception and request failure are known.

6. The most efficient opportunity out of the following actions is selected:

   - Request is processed by a service level conforming resource group;
   - Request is processed by a non-conforming resource group;

- Request is not processed.

Postcondition 6: action for further request processing determined.

5.5 DEMONSTRATION OF THE LIFE CYCLE INTERACTION

Chapter 5 evolves approaches to enable the mapping of requirements of the UC business model on a generic life cycle for SOC service offers hosted on IaaS (cf. Section 2.6), detailed in Chapter 4.

The contributions offer approaches for solutions to the problems $P_{1..2}$. The approach of a standardised usage description is introduced in Section 5.3.2 as a solution for problem P_{1a}, that is concerned with a missing standardised usage description for data exchange between phases of a life cycle. The usage pattern approach is based on the analysis in Section 4.2, which examines the relation between customer, service, and resource.

Chapter 5.4 evolves an approach to a comprehensive usage-centred data model as a solution for problem P_{1b}, that is concerned with a consistent view on the customer-service-resource relation. The usage-centred data model is also based on the analysis in Section 4.2.

For problem P_{1c}, that is concerned with a core provision model for Utility Computing services, this thesis proposes a generic provision model for Utility Computing services, evolved in Section 5.2. The provision model approach is based on the analysis in Section 4.4, which examines the functional requirements on UC-conform provision platforms.

The approaches for the usage-centred assurance of service quality are introduced in Section 5.3 as a combined solution for problem P_2. The evolved approaches are based on the analysis in Section 4.3, which examines the role of service level agreements, response times, and the quantitative and qualitative control aspects in the context of SOC service offers hosted on IaaS. The approaches combine into one approach to feasibility rating of service cascades consisting of SOC service offers in Section 5.3.5.

This section demonstrates the interaction of the evolved approaches with a generic life cycle, as introduced in Section 2.6. To enable such examinations, the delivery framework is composed out of the introduced approaches in Section 5.5.1. Sections 5.5.2, 5.5.3, and 5.5.4 examine the gathered approaches from the point of view of the life cycle phases business planning, development, and operations.

5.5.1 *Delivery Framework Composition*

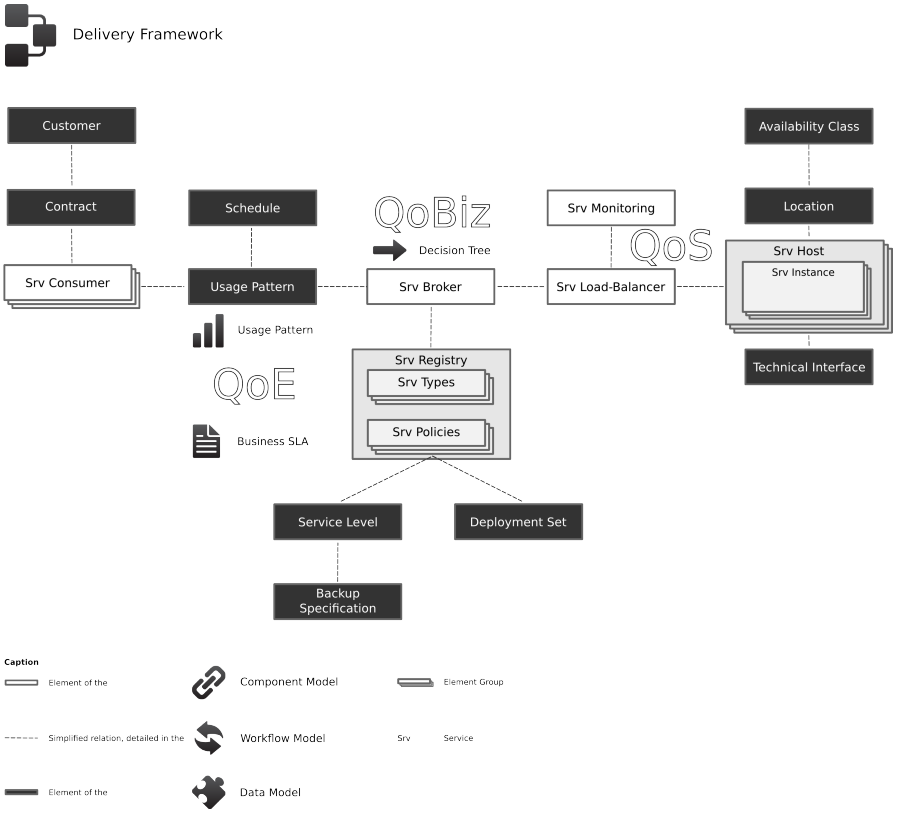


Figure 37: Illustration of the Delivery Framework as Composition of the Contributions on the Component and Data Level

Figure 1 on page 4 introduces the contributions of this chapter from the point of view of the service life cycle. While this illustration enables the understanding of how these findings relate to the phases of a service life cycle, a component and data level illustration supports the visibility of the interrelation of the evolved approaches. This illustration of the delivery framework from the point of view of the component and data level is shown in Figure 37 as composition of the contributions of this chapter.

The core of the interrelation between the evolved approaches in this thesis constitutes the BSLA approach. BSLAs provide an approach to specify the QoE as the service consumers expectation of usage experience related to a certain usage behaviour. To enable the description of usage behaviour, BSLAs apply the approach of usage pattern. In Figure 37, the BSLA is represented by an element group which consists of the service consumer components introduced in the core provision model and which relates to a usage pattern, an associated schedule,

and the service broker representing the service relation. The representation of the BSLA is completed by the service level element related to the service broker through its embedment into the service registry as part of the service policies.

The service consumer group relates to the contract and customer elements of the delivery framework, as described in the data model.

The service broker grants the opportunity for economical load-balancing, in order to enable the control of QoBiz. To support runtime decisions on the economical routing of service requests, the service broker implements the decision tree. The information pool to enable economical decisions is stored in the service registry within the data model and corresponding service policies.

QoS is controlled on the level of the service load-balancers. Based on the data collected in the service monitoring, the service load-balancer can route service requests optimised by technical criteria to a suitable resource within the resource group it controls. Service hosts aggregate resources like processing capacity or memory. Each service host shares the availability class, geographical location, and technical interface for resource management with its resource group. Service instances are hosted on service hosts. For the instantiated service types, the service registry provides corresponding deployment sets and backup specifications.

The illustration of the delivery framework simplifies the actual relations between the involved approaches. The delivery framework illustration misses a detailed representation of the workflows evolved in the workflow model. Not included into the delivery framework illustration are the following components of the component model: service network, service request, network connector, and storage network. Despite of the simplifications of the illustration, the evolved delivery framework comprehends the full feature set of all introduced approaches.

5.5.2 *Business Planning*

Business Planning

Business managers address the economical optimisation of planned or revised service offers. Therefore, the focus of business planning lies on the customer as data model element. Also in the primary focus lies the contract as a data model element. The service consumer group from the component model completes the primary focus by introducing its usage pattern and corresponding schedule.

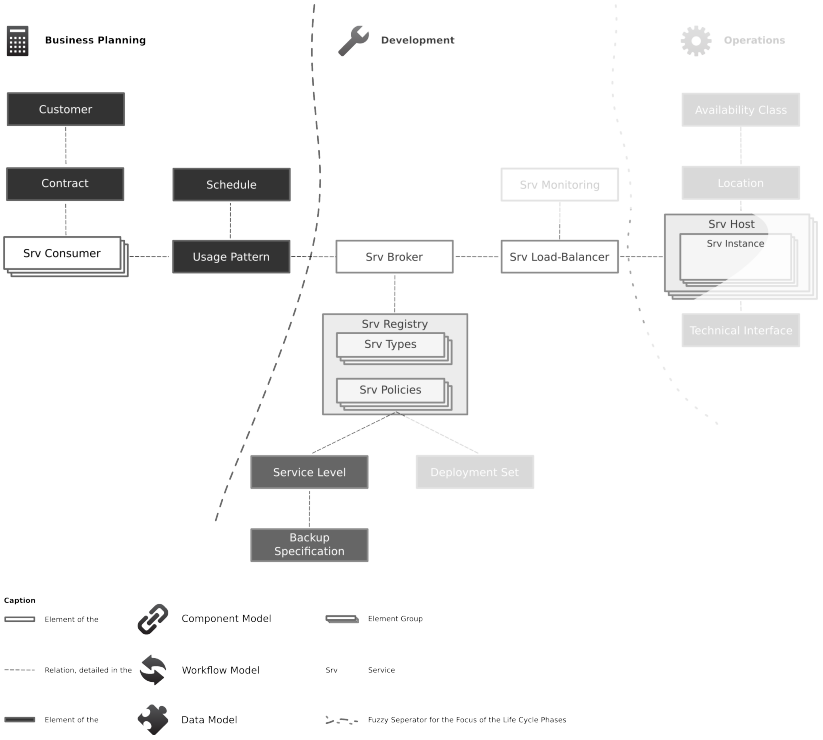


Figure 38: Field of Interests of Business Planning on the Component and Data Level in the Delivery Framework

Beside the primary focus of business managers, business planning also relies on information about the costs of resource provision and the technical boundaries of business model implementation. Therefore, the secondary focus of business planning goes up to the service host component of the component model. It also includes the service registry component, especially because service level and backup specification are relevant data model elements for customer trends.

Business planning demands the continuous analysis of concurrent market scenarios for a certain service offer, respecting its cost structure, as formulated as part of problem P_3. As business planning focuses on most of the elements of the evolved delivery framework, a simulation model - as evolved in Chapter 6 - is representing the delivery framework.

5.5.3 *Development*

IT architects are assigned to keep the balance between the cost estimations from the business planning, regarding both, costs of implementation and costs of future operations. On the one hand, the focus

of development lies on the usage pattern, the corresponding schedule, and service levels as data model elements. On the other hand, the primary focus lies on the resource consumption induced by the service instances in the service host components of the component model.

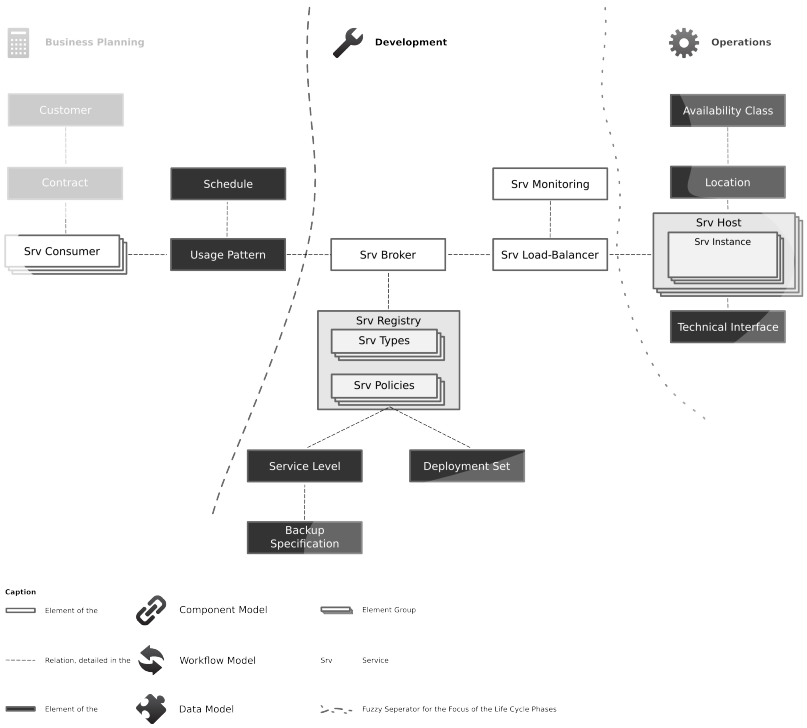


Figure 39: Field of Interests of Development on the Component and Data Level in the Delivery Framework

Beside the primary focus of IT architects, development also relies on the abilities of a provision platform to control service quality and support the orchestration of service offers into service cascades regarding pay-per-use support. Therefore, the secondary focus of IT architects reaches up to the service broker, service load-balancer, and service registry components of the component model. Also, development contributes to the deployment set. The IT architecture is affected by backup specifications and the technical interfaces of the resource management.

Development demands the continuous analysis of resource usage and performance limitations of the service implementation, as formulated as part of problem P_3. As deployment focuses on most of the elements of the evolved delivery framework, a simulation model - evolved in Chapter 6 - is representing the delivery framework.

5.5.4 *Operations*

Operations managers are concerned with the continuous quality of the provisioned service offers. Therefore, the focus of operations lies on the service monitoring, service broker, and service load-balancer from the component model. The primary focus also includes the service consumer from the component model, the usage pattern, and the schedule as data model elements, in order to enable the comparison of contracted versus monitored usage.

Beside the primary focus of operations managers, operations also relies on information stored in the service registry from the component model. Access to the service level, the backup specification, and the deployment set as data model elements is demanded to enable automated decisions of model components. This information demand also includes the availability class, location, and technical interface as data model elements related to the resource pools consisting of service hosts from the component model. Also important for the control of continuous service quality is the information about the distribution of service instances among the service hosts (e.g., to decide about automated deployment for load-balancing).

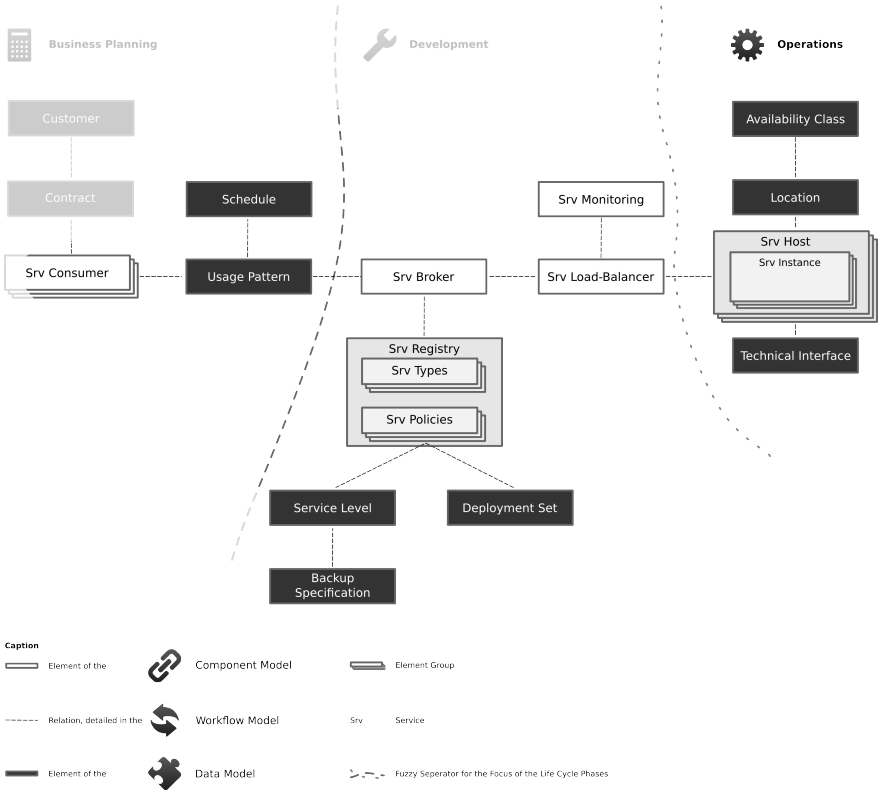


Figure 40: Field of Interests of Operations on the Component and Data Level in the Delivery Framework

Operations demands the continuous analysis of resource usage and performance limitations of the service cascades, as formulated as part of problem P_3. As operations focuses on most of the elements of the evolved delivery framework, a simulation model - evolved in Chapter 6 - is representing the delivery framework.

6 SIMULATION MODEL FRAMEWORK

This chapter describes the transfer from the delivery framework, introduced in Chapter 5, into a simulation model framework. The evolved simulation model framework offers an approach to solve problem P_3, which is concerned with the demand for an approach to analyse UC's complex service cascades (cf. Section 1.3, 4.5). The presented simulation model framework enables the analysis of concurrent market scenarios, resource usage, and performance limitations of the service cascades. Within the limits of this thesis introduced in Section 2.6, the framework implementation enables the analysis of SOC service cascades. This analysis is restricted to multi-tier IT architectures, as introduced in Section 4.3.2. The chosen IT architecture significantly effects the overhead, induced by the management communication between the components of the delivery framework. In addition, differing IT architectures imply different approaches for the implementation of simulation model frameworks. These constraints require the restriction of the type of analysed IT architecture. Due to their distribution in the hosting of SOC service cascades, multi-tier IT architectures have been chosen.

Section 6.1 briefly introduces the principles of model building and simulation focused on discrete-event models. Related work in the field of combined resource and costs simulation of UC service cascades in SOC architectures hosted on IaaS are analysed in Section 6.2. The evolved delivery framework is mapped into a simulation model in Section 6.3. Section 6.4 highlights the major implementation details of the simulation model framework, while Section 6.5 discusses its capabilities and restrictions.

6.1 BACKGROUND OF MODEL BUILDING AND SIMULATION

6.1.1 *Simulation*

Representing the dynamic behaviour of a system by another system is the goal of the simulation process. *What if?* questions are the most typical problem statements simulation models are used to answer. Origin for the process of simulation is a real world system. Its behaviour is measured and analysed in terms of the systems interaction with its environment. Based on these findings, a simulation model is built, in

order to represent the major characteristics of the original system. In case of simulation models, the major characteristics are those which relate to the analysed system's behaviour.

The results of a simulation run, as the product of the simulation process, are approximations of the corresponding behaviour results of the real world system under identical environmental conditions. Such a set of environmental conditions, as input data for a simulation run, together with the desired output data define an experiment.

Modelling begins after the experiment is defined. Depending on the nature of the real world system to be represented, different types of simulation models can be chosen. The most common diversification property for systems is their dynamic-change behaviour of the system variables. Here it is distinguished between continuous-variable models and discrete-variable models.

Continuous-variable models are composed of mathematical expressions for the modelling of the contentious changes of their system variables.

Discrete-variable models - commonly known as discrete-event models - describe the change of system variables in discrete steps, neglecting optional transients between the states. In discrete-event models, queueing of messages between the entities of a model is a characteristic aspect.

> "Classical queueing theory relationships can be used to represent only a limited range of model complexity and variability. To transcend this range, a simulation model involving numeric and symbolic variable representation and manipulation is employed."[1] (Ralston et al., 2003, p. 1204)

Simulation models are implemented as computer programs, in order to enable the representation of complex dynamic processes. For this purpose, specific high level programming languages for the implementation of simulation models exist.

Simulation models and their implementation must be validated, in order to ensure the reliability of the simulation results. As simulation is an approximate process to analyse the validity of a result, the relative accuracy for the gathered output data has to be determined. The analysis of the relative accuracy therefore requires an understood criterion of validity.

This summary of simulation refers to the work of Roth (1987); Banks (1998); Ralston et al. (2003).

1 Queueing theory deals with the description and analysis of connected queues as limited message stores for the analysis of performance measures of the modelled system. (Cooper, 1981)

6.1.2 *Building Discrete-Event Models*

The evolved delivery framework represents a system of interacting entities, where few of the variables change at discrete and mostly random points of time. The result data of framework simulation runs should be analysed in terms of resource usage, request and queue timing, and historical data. The previous described properties indicate the delivery framework representation as discrete-event model.

Discrete-event models consist of a set of basic elements. These elements are users, resources, queues, and demands. Users consume resources. Resources represent entities that are offered as consumables. Users and resources interact based on events. Queues hold, optionally order, and forward events. Demands are scheduled events that wait for the processing conform to the model description.

Software is used to implement discrete-event models. Often, this is done using a high level programming language in conjunction with a simulation framework. In common, simulation frameworks consist of a simulation executive, a clock, random number generators, a result collection, and an application programming interface to enable the interaction between the simulation model implementation and the framework.

A simulation executive controls the time advance during the simulation runs. The executive coordinates the events between the simulation model entities. The central clock is used to track the time. Random number generators enable the simulation of stochastic behaviour. The result collection stores model-specific metrics for analysis.

Discrete-event models are built without relying on a specific formal method or description. Depending on the author in literature, the recommendations vary from the ad hoc mixture of numeric and symbolic notation, the flowcharting, or the conclusion that the model is the source code of the simulation software itself.

This summary of discrete-event model building refers to the work of Ball (2001); Ralston et al. (2003); Banks et al. (2009, p. 1204).

6.2 RELATED SIMULATION MODEL FRAMEWORKS

This section introduces related work in the field of combined resource and costs simulation of UC service cascades in SOC architectures hosted on IaaS. Other approaches for cloud computing simulation are introduced, and it is demonstrated why these approaches are not sufficient in the context of this thesis. The framework also includes an approach for a cloud broker, which enables the negotia-

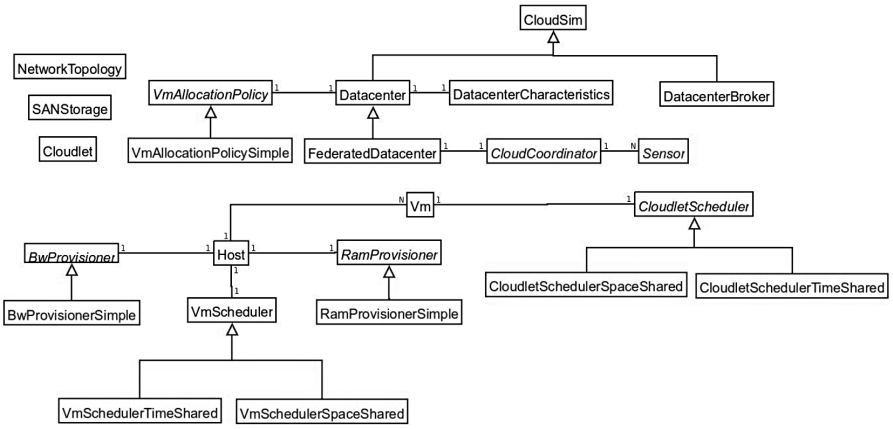


Figure 41: Buyya's Cloud Simulation Model Framework Class Design Diagram (Calheiros et al., 2011a)

tion on services or resources of other service providers based on QoS parameters.

Buyya's CloudSim

CloudSim (Buyya et al., 2009; Wickremasinghe et al., 2010; Calheiros et al., 2011a; Garg and Buyya, 2011) is a simulation model framework for cloud computing platforms. It offers an approach to model the behaviour of physical resources spread among data centres. Supported is the modelling of network, processing, memory, and storage resources. CloudSim supports the modelling of a virtualisation layer for physical resources. Based on the specified resources, cascading cloud services can be modelled. The framework offers a monitoring interface to model individual behaviour like power consumption. The classes of the simulation model framework are illustrated in Figure 41, in order to provide an overview of the core feature set.

Buyya's CloudSim - in comparison to the approach of this thesis - offers a more detailed resource abstraction. CloudSim enables the modelling of a virtualisation layer and the modelling of the power consumption of server hardware. The basis of CloudSim is Buyya's InterCloud model for cloud computing, introduced in Section 3.1. Resulting, the CloudSim inherits its discussed limitations. Compared to the simulation model (cf. Section 6.3) of this thesis, the CloudSim model misses a service broker which supports economical load-balancing (cf. Section 5.2.2) and the approach of BSLA (cf. Section 5.3.4).

Ostermann's GroudSim

GroudSim (Ostermann et al., 2011b,a) is a simulation model framework for Grid and cloud platforms. SimEngine represents the core of the framework. It takes care of the time advance and holds the future event list. The framework offers different types of GroundEntities (e.g., CloudSite or GridSite). Workloads for resources can be modelled using GroudJobs. Costs for resource consumption can be modelled either on a charge per processing time unit or in fixed time intervals independent from the actual processing amount. To enable the evaluation of simulation results, tracing can be modelled as either entity state or event-based tracing. To support the use of probability distributions for simulated workloads and the failure behaviour of resources, GroudSim offers a range of predefined distributions (e.g., exponential or logarithmic). For each GroundEntity, its failure behaviour can be modelled. In addition, GroudSim can simulate background load on resources based on correspondingly modelled load descriptions.

The primary aim of GroudSim is the improvement of runtime performance of simulation runs. It explicitly competes with the previously introduced CloudSim framework in terms of runtime performance. It is not based on a superior cloud computing model. Therefore, compared to CloudSim or the simulation model framework introduced in this thesis, GroudSim offers very basic building blocks to model UC service cascades in SOC architectures hosted on IaaS.

Nunez's iCanCloud

iCanCloud (Castane et al., 2012; Nunez et al., 2012, 2011a,b) is a simulation model framework for Grid and cloud platforms with a focus on scientific Grid applications. Figure 42 presents an overview of the architecture of iCanCloud. The cloud hypervisor can be modelled with a variable list of queues for waiting and finished jobs. The hypervisor comprises a list of users, job scheduling, brokering policies, and cost policies. The VM map enables the modelling of sets of virtual machines, in order to model multiple cloud applications. Each modelled VM consists of a network system, a CPU system, a memory system, a list of application instances, and an API.

One of the goals of iCanCloud - as of GroudSim - is the improvement of the runtime performance for the calculation of simulation runs representing the behaviour of very large cloud platforms. In almost the same manner as GroudSim, iCanCloud explicitly competes with CloudSim. As a result of this competition, iCanCloud aims at

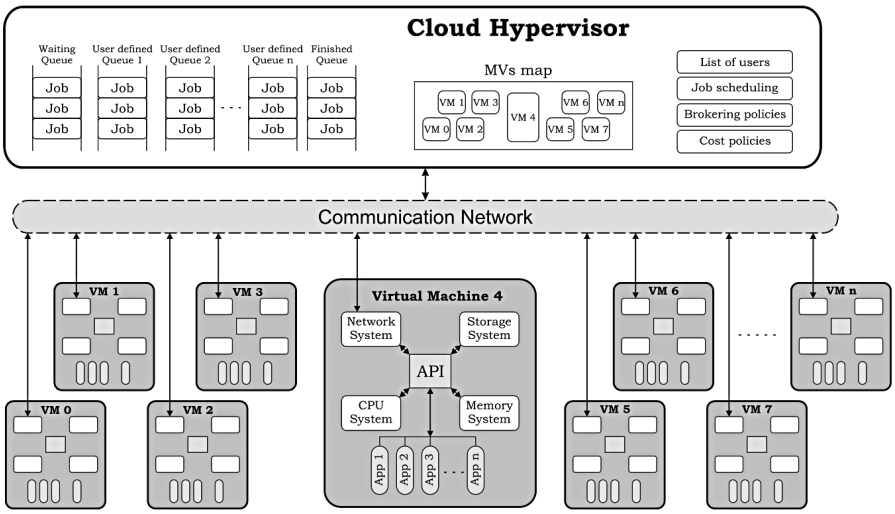


Figure 42: Basic Schema of the Nunez's iCanCloud Architecture (Nunez et al., 2012)

exceeding CloudSim in terms of Graphical User Interface (GUI) and runtime performance. Compared to the simulation model (cf. Section 6.3) of this thesis, the framework offers just very basic building blocks, in order to model UC service cascades in SOC architectures hosted on IaaS, as trade-off.

Kliazovich's GreenCloud

GreenCloud (Kliazovich et al., 2010b,a) is a simulation model framework for cloud platforms. The framework enables the modelling of cloud platforms, in order to support the estimation of their energy consumption. GreenCloud uses a network-centric modelling approach and is built based on a network simulation model framework. Therefore, the simulation model framework offers detailed properties to configure and analyse the network behaviour of simulation models.

Figure 43 presents an overview of the architecture of GreenCloud. The framework supports the modelling of three-tier network architectures for data centres. Cloud users can be modelled as workload generators based on workload trace files. The generated traffic enters into the modelled data centre. The centre consists of task schedulers, a three-tier network architecture, task completion agents, and a server pool. The data centre entity can be modelled including data centre characteristics. The first network layer represents the core network of the data centre. Depending on the setup, the switches of the core network can be modelled as 10 gigabit or 100 gigabit ethernet switches. Each switching layer can be modelled with a certain energy model.

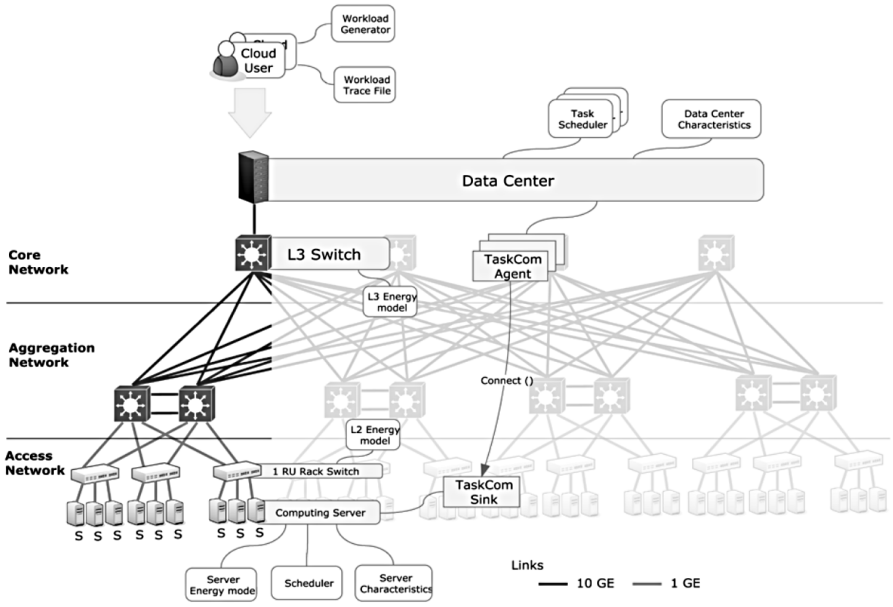


Figure 43: Kliazovich's GreenCloud Architecture (Kliazovich et al., 2010b)

The second network layer represents the aggregation network. The access network of the data centre is represented by the third network layer. This layer connects to the servers within the pool. Each computing server consists of a server energy model, a scheduler, server characteristics, and a task completion sink.

GreenCloud does not offer the abilities necessary to map UC service cascades in SOC architectures hosted on IaaS, as it has not been designed for this purpose. The offered building blocks are not sufficient to map the delivery framework properties (cf. Section 5.5.1) into a simulation model, beside the basic modelling of the relation between consumer, service, and resource.

6.3 SIMULATION MODEL ELABORATION

Component Model

The introduced component model (cf. Section 5.2.2) is completely mapped to the simulation model. Service networks (external service providers) are mapped directly to the simulation model. Service consumers are mapped as consumer groups referring to the data model. Service requests are initiated by consumer groups. The service monitoring, service broker, and service registry are mapped directly to the simulation model. The service types and service policies (brokering, error, event, and security) are consolidated as single service type policy per service. Sites represent groups of resources (e.g., often

grouped by their geographical location). One service load-balancer, a storage network, and a group of service hosts are mapped as a site to the simulation model. Each service host is modelled, in order to contain a set of service instances able to act as service consumer, corresponding to the specification of the component model. Each simulation model component is modelled, in order to be enabled to connect to another component by using a network connector.

 Workflow Model

Consumer groups use the service broker through workflow step s-02, in order to initiate service requests. Workflow steps s-04 and s-05 are modelled, in order to store and retrieve load information for the service broker and the service load-balancers. The overall load state of a site is determined through step s-06. Other service networks are used by the internal service broker through workflow step s-07. The information of step s-T is modelled, in order to be stored in the service request response.

Steps s-01, s-03, and s-8 are omitted due to the experimental setup.

 Decision Tree

The decision tree is modelled as part of the service broker of the simulation model. The decision tree enables the service broker to take economical aspects for its routing decision into account. In opposite to this behaviour, the service load-balancer forwards the incoming requests exclusively by technical metrics. To simplify the modelling of the experimental setup, the economical load-balancing is modelled as an prioritised queue. As priority criterion for the queueing, service requests are modelled with a priority metric to indicate their affiliation to a specific economical class. The economical classification of service requests is done manually during the configuration of the simulation runs.

 Data Model

Service requests consolidate information from usage pattern, service level, contract, and schedule classes from the data model. The class *service* is represented by a service type policy. The association parameters *price_standard*, *price_overrun*, and *costs_penalty* are substituted by a simpler modelling of resource pricing due to the experimental setup. A service type policy specifies the costs per unit of resource usage per site. In addition, a percentaged margin can be added to these costs. Alternatively, a fixed price per request can be specified. This pricing implementation also represents the relevant contracting information from the data model. The location of resource groups is

represented as part of a service type policy. The resource group of the data model is modelled as service host in the simulation model.

Omitted are the classes *customer*, *technical interface*, *availability class*, *deployment set*, and *backup specification*. The associated relations of these classes are omitted, too. The functionalities represented by these classes are only relevant for the actual implementation and operation of provision platforms.

An overview of the components and associated data of the introduced simulation model is presented in Figure 44.

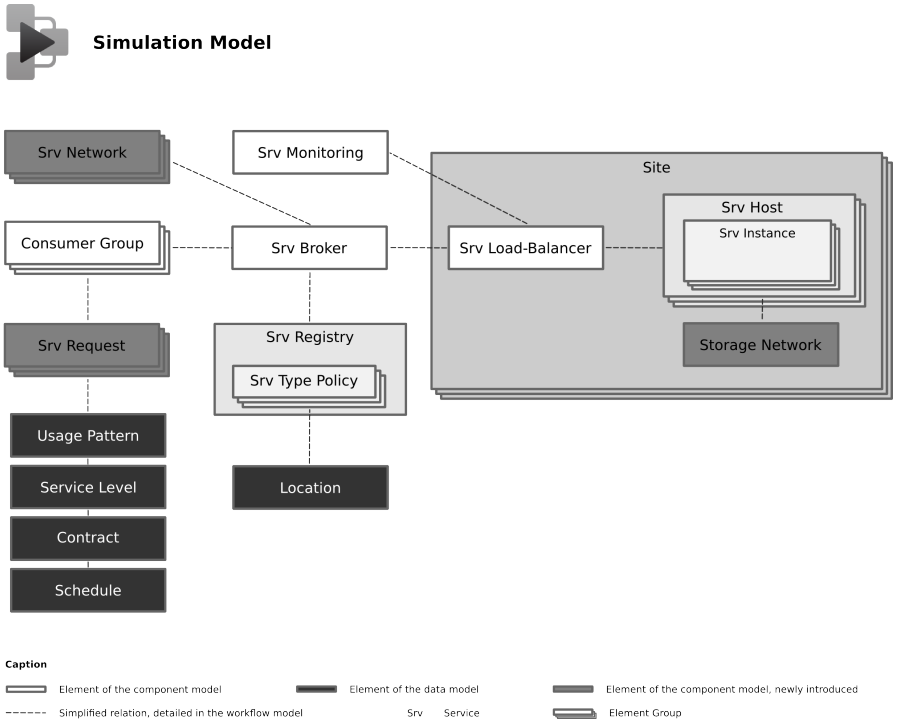


Figure 44: Mapping of the Delivery Framework Into the Simulation Model

The evolved delivery framework is modelled, in order to represent IT service provision in a multi-tier architecture. This architecture is depicted in Figure 45. The elements service request and service type policy of the previously introduced simulation model are not illustrated in the multi-tier architecture. It is not possible to assign these elements to a specific architectural layer.

The service broker layer, given in the reference architecture and presented in Section 4.3.2, is extended by the service registry. The layer is renamed to *service providers*, in order to reflect this addition. The application infrastructure and the virtual infrastructure layers are obsolete in the simulation model. Their prior functionalities are

merged into the service instances layer of the evolved model. In addition to the reference architecture, the simulation model assumes, that service instances are hosted in VMs which include the functionalities previously provided by the application infrastructure and the virtual infrastructure layers. The operating system layer is merged into the service hosts on the physical systems layer. The network services layer is dispensed in the simulation model. The layer services (e.g., routers, switches, or domain name services) are not considered in the simulation model. The only considered feature of the network services layer of the generic architecture is the network bandwidth of the connections between the entities. The network bandwidth is modelled in the network connections of the simulation model. Introduced as new layers in the multi-tier architecture of the simulation model are the storage networks and the service processors layer. The storage networks layer hosts the storage network entity of the simulation model. The service processors layer hosts the link to the external service networks and the internal service load-balancers. The construct of service lines, introduced in Section 5.3.5, is represented by the *site* entity in the simulation model.

The key model is also published by Heckmann et al. (2009).

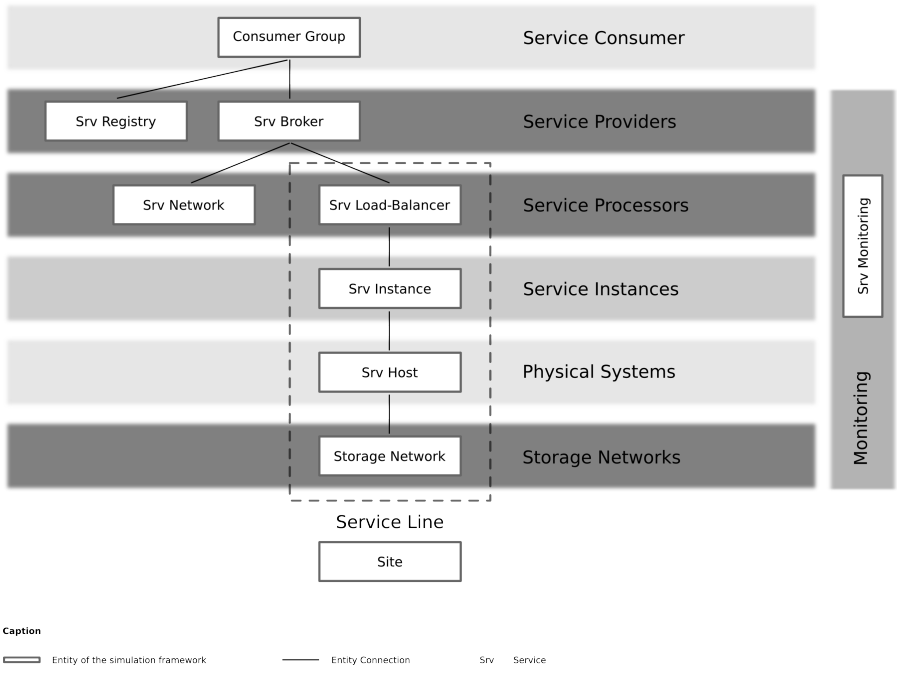


Figure 45: Multi-Tier Architecture View on the Simulation Model

6.4 IMPLEMENTATION OF THE SIMULATION MODEL

6.4.1 *OMNeT++ Simulation Library & Framework*

OMNeT++ (Varga, 2001; Varga and Hornig, 2008) serves as basis for the implementation of the evolved simulation model, presented in the previous chapter. OMNeT++ offers a combined simulation library and framework that is extensible, modular, and component-based. The entire framework is implemented in the programming language C++ JTC1/SC22/WG21 (2012). This brings forth C++ as preferred programming language for simulation models implemented using OMNeT++. The simulation library and framework can be used to implement network simulation models like wired and wireless communication networks, on-chip networks, queueing networks, and more. There are many domain-specific model frameworks that enable the extension of OMNeT++, in order to provide further - often more detailed - functionality (e.g., sensor networks, wireless ad-hoc networks, or Internet protocols). One of the major benefits of OMNeT++ is its graphical runtime environment. It enables fast debugging and clear understanding of simulation models through visual model exploration.

Simulation models build using OMNeT++ consist of three main parts. The central building block for simulation models in OMNeT++ is the model structure defined in the NED language. Using the NED language, the model components and their interrelations are defined. Relating components can communicate by exchanging messages through channels. The conditions for the message flow can be determined per channel. Two types of components can be defined. Simple modules represent active components, which contain own logic specifying their behaviour. The logic of each simple module is implemented in C++ code, based on the OMNeT++ simulation kernel and class library. Simple modules cannot host nested sub-modules. Compound modules are used to group simple modules or other compound modules. Compound modules are passive components, as it is not possible to embed logic directly. At the end of a modelling process, OMNeT++ simulation models are represented in a hierarchy of interrelated modules called network.

As previously introduced, each simple module is implemented in C++ code. These individually implemented logic components represent another main part of the OMNeT++ framework. Each individual C++ component inherits from the basic component class provided by the simulation library and registers itself at the simulation kernel. From this point on, the component can send and receive messages

over the specified channels and access the available kernel and library features of OMNeT++.

The last building block for simulation models in OMNeT++ is the simulation run configuration. Beside the initialisation of the component parameters, the configuration comprehends time-limits for simulation runs, the definition of iterations, constraint expressions, differing seeds, or the initialisation of different random number generators. In one simulation run configuration, multiple simulation runs can be specified with differing parameters.

While the configuration for simulation runs can easily be changed editing the corresponding text files, changes to the model structure or the C++ components require the simulator[2] to be recompiled.

6.4.2 *Model Structure in NED Language*

The network module (root element) of the model structure (component hierarchy) for the simulation model framework of this thesis consists of a set of consumer groups and one or more service provider modules. While the service providers and their interrelations are specified directly in the model structure, the number and behaviour of consumer groups can be configured through the simulation run configuration. This enables the simple variation of service consumer behaviour (usage patterns) for varying simulation runs, without the need to recompile the simulator before each simulation run. Listing 1 introduces the corresponding NED code that is using two service providers as example setup.

Listing 1: Example of a Network Module Definition in NED Code

```
//Definition of the root module
module UCNet
    parameters:
        consumerGroups: numeric const;
    submodules:
        consumerGroup: SrvConsumer[consumerGroups];
        provider1: UCSrvProv;
        provider2: UCSrvProv;
    connections:
        for i=0..sizeof(consumerGroup)-1 do
            consumerGroup[i].srvRequest --> InternetLastmile -->
                provider1.srvRequest++;
            consumerGroup[i].srvResponse <-- InternetLastmile <--
                 provider1.srvResponse++;
        endfor;
```

2 In this thesis, the compiled executable program of a simulation model is called simulator.

```
        //Use this loopback to access services provided within
            your own network
        provider1.ucServicesOut++ --> provider1.srvRequest++;
        provider1.ucServicesIn++ <-- provider1.srvResponse++;
        //Service provider interconnections
        provider1.ucServicesOut++ --> provider2.srvRequest++;
        provider2.ucServicesIn++ <-- provider1.srvResponse++;
        provider2.ucServicesOut++ --> provider1.srvRequest++;
        provider1.ucServicesIn++ <-- provider2.srvResponse++;
endmodule

//Registration of the root module
network ucnet : UCNet
endnetwork
```

Consumer groups are represented by the simple module *SrvConsumer*. The module basically consists of one incoming and one outgoing gate to interconnect the module with service providers in the network. Listing 2 shows the corresponding NED code.

Listing 2: Example of a Service Consumer Group Definition in NED Code

```
//Definition of the service consumer module
simple SrvConsumer
    parameters:
        queuePriority: numeric const,
        srvRequestConfig: xml;
    gates:
        in: srvResponse;
        out: srvRequest;
endsimple
```

In opposite, service providers are represented by the compound module *UCSrvProv*. This module is composed of a service broker, service registry, service monitoring, and a set of sites. The service provider module offers four sets of gates for interconnections. There are two sets of gates for incoming messages and two sets for outgoing messages. At any one time, one pair of incoming and outgoing gate sets is used to interconnect one communication direction. Consumer groups only demand one communication direction, from the consumer to the provider. Service providers among each other are connected in both directions. In this way, providers are able to share services between each other. Listing 3 introduces the corresponding NED code.

Listing 3: Example of a Service Provider Definition in NED Code

```
//Definition of the service provider module
module UCSrvProv
    parameters:
        srvPolicies: xml,
        sites: numeric const;
    gates:
        in: srvRequest[];
        in: ucServicesIn[];
        out: srvResponse[];
        out: ucServicesOut[];
    submodules:
        broker: SrvBroker;
        site: Site[sites];
        registry: SrvRegistry;
        monitoring: SrvMonitoring;
    connections:
        for i=0..sizeof(srvRequest)-1 do
            srvRequest[i] --> broker.srvRequestIn++;
            srvResponse[i] <-- broker.srvResponseOut++;
        endfor;
        for i=0..sizeof(site)-1 do
            broker.srvRequestOut++ --> InternalBackbone --> site[
                i].srvRequest;
            broker.srvResponseIn++ <-- InternalBackbone <-- site[
                i].srvResponse;
            monitoring.lbOut++ --> site[i].lbMonIn;
            monitoring.lbIn++ <-- site[i].lbMonOut;
            registry.lbOut++ --> site[i].lbRegIn;
            registry.lbIn++ <-- site[i].lbRegOut;
            monitoring.hstIn++ <-- site[i].conMonOut;
        endfor;
        for i=0..sizeof(ucServicesIn)-1 do
            ucServicesIn[i] --> broker.ucServicesIn++;
            ucServicesOut[i] <-- broker.ucServicesOut++;
        endfor;
        broker.monOut --> InternalBackbone --> monitoring.brkIn;
        broker.monIn <-- InternalBackbone <-- monitoring.brkOut;
        broker.regOut --> InternalBackbone --> registry.brkIn;
        broker.regIn <-- InternalBackbone <-- registry.brkOut;
endmodule
```

The NED code used for the simulation model framework of this thesis comprehends more modules. Beside the previously introduced model components, the following modules are specified: *Site*, *SrvHost*, *SrvBroker*, *SrvLoadbalancer*, *SrvInstance*, *StrgNetwork*, *SrvMonitoring*, and *SrvRegistry*. In addition, a *MsgConcentrator* and a *SrvHostSwitch* module are defined as helper modules, in order to enable the consoli-

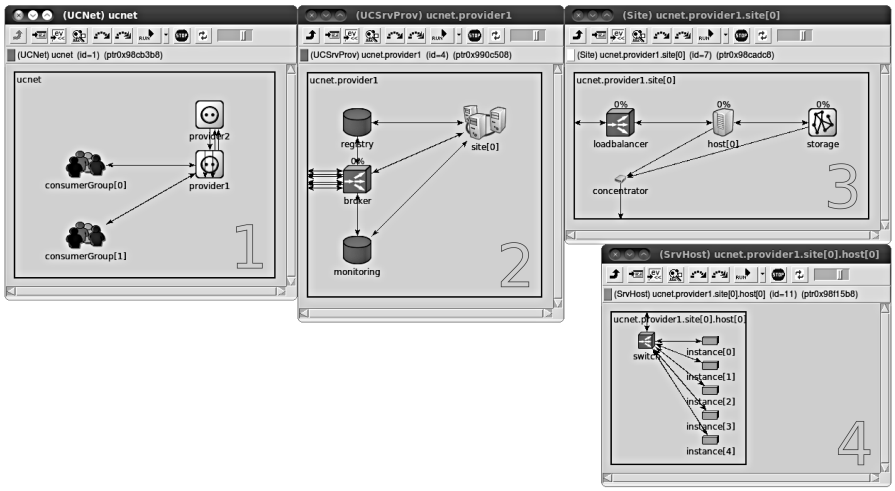


Figure 46: Screenshot of the Simulation Model Framework Implementation

dation of message flows over single connections. As channels for interconnections, *InternetLastmile*, *InternetBackbone*, and *InternalBackbone* are specified to provide different classes of bandwidth for service request transport.

6.4.3 *Model Logic as C++ Components*

The behaviour of simple modules defined in the model structure is implemented in C++ components using the OMNeT++ simulation library and kernel.

Figure 46 shows a screenshot of the simulation model framework implementation based on OMNeT++. The window marked with 1 shows the top of the model hierarchy. Shown are the service consumers, service providers and their connections. Window 2 shows, as example, one service provider as the next hierarchy level of the simulation model. The provider level consists of the service broker, registry, monitoring, and sites. One level deeper, window 3 shows the first site of service provider 1 with its service load-balancer, hosts, and storage network. Window 4 shows the deepest hierarchy level. On this level, the window shows the internals of the service host of site 1. Primarily, the host consists of a fixed number of empty sockets for service instances.

Some restrictions in the implementation of the model logic were made and are detailed further.

The service monitoring and service registry are implemented as structural stubs. Due to the experimental setup, the entities are modelled within the framework, in order to verify their embedding in

the implementation. The mechanisms for load prediction instead, are not implemented within the components. Access to the service type policies is implemented directly through the service broker.

The workflow step s-T is replaced by the logging of the underlying simulation package.

The basic functionality of the decision tree is represented by an ordered queue. The primary order criterion is the service level of service requests. Requests of the same type are ordered by their arrival time at the service broker. If the resources of the service broker to queue and process service requests are exhausted, further incoming requests are dropped.

Identical implementations of the queueing are used in the service broker and service load-balancer.

6.4.4 *Configuration of Simulation Runs*

The simulation model framework is configured using a text file in an OMNeT++ specific format. In addition, this framework uses the ability of OMNeT++ to access files in the XML format. The main properties of a simulation run that are configurable are introduced below.

- Resources

  Resource must be configured for all modules, except for the service consumers. Each module offers attributes for the specification of abstract resources for processing, memory, and local storage. Listing 4 shows the configuration of the host resource as an example.

  Listing 4: Example of a Resource Configuration for Service Hosts

```
#Processing resources
**.site[*].host[*].res_cpu = 100
#Memory resources
**.site[*].host[*].res_mem = 100
#Local storge resources
**.site[*].host[*].res_hdd = 100
```

  For each module offering resource attributes, a corresponding abstract reserve property (*res_reserve*) must be specified. The reserve refers to the processing resource and clarifies that systems cannot be kept responsive, when they are used to 100 percent.

  For each site, a representation of a storage-area-network is provided. To indicate the latency for its access, the property *procDurFac* can be used. The property is used as multiplier of the

processing duration of a service request, in order to calculate the latency of the storage-area-network.

- Resource grouping

  As specified in the component model, service hosts provide resources to process service requests. Hosts are grouped into sites. A service provider can have multiple sites, specified in the *ucnet.<service-provider>.sites* property. Each site can have multiple hosts, defined in the *ucnet.<service-provider>.site[<number>].hosts* property .

- Instances

  Service hosts need to be configured with their maximum capacity for service instances (***.site[*].host[*].ld_maxInstances*). Per service type, only one service instance per service host can be run. Service instances, by default, are deployed and instantiated by the service load-balancer. After a configurable timeout (***.site[*].loadbalancer.garbageCollectionDelay*), during which the instance is not used, service instances are removed from hosts.

- Load-balancing

  For load-balancing, two different modules are used. The service broker conducts economical load-balancing between the available sites and external service providers. Per site, a service load-balancer distributes service requests to the available service hosts. The scheduling strategy used tries to process the given request flow with the minimal number of hosts. This enables the deactivation of unused hardware to save energy costs.

  To enable the comparison of different scenarios, the queueing behaviour can be specified. Supported are first-in-first-out[3] or priority[4] scheduling. Further, the queueing delay can be configured. The number of requests simultaneously retrieved from the queue can be configured.

  Per site, a delay for the deployment of new service instances to service hosts can be specified. In addition, the interval to remove unused service instances can be defined.

3 The first arriving service request is also the first request to leave the queue.
4 Prioritised by service level class.

Listing 5: Example of a Load-Balancing Configuration

```
#Queue sorting (0 = FIFO; 1 = by priority)
**.queuePriority = 0
#Delay of message before leaving the queue
**.queueDelay = 0.1s
#Number of messages retrieved from queue at once
**.queueFactor = 3
#Delay of instantiation of service instances on hosts
**.site[*].loadbalancer.deploymentDelay = 0.1s
#Interval for garbage collection
**.site[*].loadbalancer.garbageCollectionDelay = 120s
```

Service Request Configuration

The behaviour on service requests for groups of service consumers is specified using a link (*ucnet.consumerGroup[<number>].srvRequestConfig = xmldoc("<srvRequest_consumerBehaviour.xml>")*) to a corresponding XML file from within the OMNeT++-specific text configuration.

Via the XML configuration, a service type is specified. The resource demands for request processing and the resource demands during the network transport of the request are defined. This includes the bandwidth demand as well as the resource demand for load-balancing.

For service requests, a service level class and a corresponding maximum latency are specified. Also, a multiplier for margin calculation is defined in the XML file. The consumer cost (or price) of a service request is the product of the provision costs multiplied with the margin multiplier.

The usage pattern of the consumer group is defined in the XML file, too. To specify a schedule for service requests, the initial delay prior to the first action of a service consumer can be specified. A delay between the sending of two service requests can be defined. Correspondingly, the behaviour of the delay can be controlled in more detail. Two types of delay behaviour are offered. The time interval between the requests can either be fixed or some random deviation can be added. The degree of deviation can be specified. The duration of a request sequence can either be defined as absolute number of requests or as time interval. In addition, it can be specified how often a complete request sequence should be repeated.

At last, it can be configured that a request is to be sent after the response of the previous request has returned. Or it is sent after a certain timeout.

Listing 6 introduces the approach to configure service cascades. Service request definitions can include other service request definitions as sub-requests.

Listing 6: Example for the Configuration of Service Request Cascades

```
<srvRequest>
    <param id="<uid>" value="<foo>"/>

    <requestCascade>

        <!-- EMBEDDED REQUEST-->
        <srvRequest>
            <param id="<uid>" value="<foo>"/>

            <requestCascade>
            </requestCascade>
        </srvRequest>
        <!-- EMBEDDED REQUEST-->

    </requestCascade>

</srvRequest>
```

Service Policies Configuration

The properties of the service type provision by service providers is specified using a link (*ucnet.<provider>.srvPolicies = xmldoc ("srvPolicy_providerBehaviour.xml")*) to a corresponding XML file from within the OMNeT++-specific text configuration.

Via the XML configuration, several service types can be specified. Each service type includes the definition of properties for cost calculation and priority specification. Cost of service request of a specific service type can be fixed per request or calculated based on the resource consumption. To calculate the dynamic price per request, the costs per resource and a multiplier to calculate an additional margin for the service provider are specified.

In the case different service providers offer the same service type, a weight can be specified to enforce a selection order.

6.5 CAPABILITIES AND RESTRICTIONS OF THE FRAMEWORK

Modelling of Consumer, Provider & Service Relations

The framework enables the modelling of an unlimited[5] number of consumer groups and service providers. Each consumer group always relates to one specific service provider. Consumers of one group

5 Restricted by the resources and capabilities of the underlying software and hardware.

use a single service type. The given scheduling properties allow the modelling of simple request distribution over time. It is possible to define a sequence of requests. The distribution of requests within the sequence can be either fixed or randomly varied within a given time frame. Previous to a sequence, a delay can be defined. Sequences can be repeated. More complex distributions can be achieved by combining multiple consumer groups, in order to represent complex behaviour. Behaviour beyond the combination of multiple consumer groups for the representation of complex behaviour is not supported (e.g., non-linear, fully randomised, or standard distributions).

For the analysis of complex service cascades, a loop detection is not implemented. Service requests invoking sub-requests to themselves will not be detected.

Modelling of Costs & Pricing

The modelling of pricing for resource usage is basically supported. Service usage can be charged by a fixed price per request or by resource usage during processing. In case of the pricing by resource usage, the resource costs are calculated and a percentaged provider margin is added. Within service cascades, the cost information is collected from sub-requests and added to the total costs of a request, if the request is charged by usage.

Complex pricing models (e.g., discounts per consumer group or per request quantity) cannot be represented using the simulation model framework.

Part III

EVALUATION

7

EVALUATION OF THE SIMULATION MODEL FRAMEWORK

In this chapter, the previously introduced Simulation Model Framework (SMF) (cf. Section 6) is evaluated, in order to determine its qualification to enable the modelling of SOC service offers hosted on IaaS (cf. Section 2.6). The SMF is build on the contributions of this thesis. The evaluation of the framework also evaluates these outcomes.

Section 7.1 evaluates the basic features of cloud platforms based on test cases well known in the research community. In Section 7.2, the framework is evaluated based on an independent test case, in order to analyse quality and profit assurance in UC service cascades.

7.1 EVALUATION OF BASIC CLOUD SCENARIOS

7.1.1 *Test Cases Derivation*

To evaluate the output of the evolved SMF, at least one service provision scenario and corresponding reference values for the comparison with simulation run results are necessary. To gain such scenarios and reference values, the published experiments of Buyya's CloudSim framework (cf. Section 6.2) are used. As Calheiros et al. aim at modelling cloud platforms, the publications introducing CloudSim include the description of experiments to evaluate its qualification, in order to enable the modelling of SaaS and IaaS scenarios.

For the evaluation of the SMF, three test cases are extracted from the publications of the CloudSim framework. As basic test case, Calheiros et al. refer to the federation of cloud computing components between available resource pools (cf. Section 7.1.2). As second test case, the framework is evaluated on its ability to model hybrid cloud environments (cf. Section 7.1.3). Section 7.1.4 evaluates the ability of SMF to model high loads in Cloud environments.

7.1.2 *Federation of Cloud Components*

Test Case

The ability of CloudSim to model scenarios of the distribution of cloud computing components on given resource pools is evaluated in a small test case. Calheiros et al. describe an experimental setup

(Calheiros et al., 2011a) with three providers of cloud resources offering their resources to a single service consumer. Each data centre is modelled providing 5[1] computing hosts, each offering a capacity of 10 GB of memory, 2 TB of storage, and 1000 MIPS of processing resources.

The service consumer requests 25 VMs with each requiring 256 MB of memory, 1 GB of storage, and one processor. Associated to each VM is a cloudlet with a resource demand of 1800000 MI. The request is sent to the primary data centre. The other centres are provided as secondary resources for the usage through the primary centre.

Two scenarios are being compared. In the first scenario, the primary data centre does not rely on the resources in the secondary centres for request processing. In the second scenario, in addition to the primary resources, the secondary resources are used to processes the workload.

Experimental Setup

The previously introduced test case is modelled in the SMF as a single service consumer group to service provider relation. A service consumer group invokes 25 service requests of the same service type. The requests are scheduled with a delay of one second between each request. The resource demand of the requests is modelled, corresponding to the test case with 256 MB memory, 1 GB storage, and 100 % processor usage for 1800 seconds.

The resources of the service provider are equally split on three sites. Each site consists of five hosts, each with a resource offering of 10 GB memory, 2 TB storage, and 1000 MIPS processing capacity.

The processing effort for request transport on the service broker and service load-balancer is set to 1 MI. The network traffic generated by a single service request is configured to 10 MB. The available network bandwidth between the service consumer group and the service

1 In the original publication, the number of hosts is specified to 50.
The analysis of the experimental setup reveals that taking 50 hosts into account does not provide a necessary bottleneck that could be solved by federation. For the specified example workload, 25 hosts are sufficient to calculate all cloudlets in parallel. To gain an observable effect, the number of hosts per data centre must be smaller than 25.
Calculating backwards, the results of Calheiros et al., for a processor with 1000 Million Instructions Per Second (MIPS), a cloudlet with 1800000 Million Instructions (MI) would take 1800 seconds as makespan. For the 25 specified cloudlet instances this results in an overall sequential makespan of 45000 seconds. Divided by the published overall makespan of 8405 seconds, this results in approximately 5.4 parallel calculating cloudlet instances in mean.
The issue has been discussed with R. N. Calheiros, as the corresponding author of the publication, who acknowledges the issue. Therefore, this thesis continues to work with the number of 5 hosts per data centre.

	Average turn around time (s)	Makespan (s)
CS-1S	4700.1	8405
CS-3S	2221.13	6613.1
SMF-1S	5471.8	9094.1
SMF-3S	2958.5	5490.5

Table 1: Result Comparison Between CloudSim (CS) and the Simulation Model Framework (SMF) in Cases *Without Federation* (1S) and *With Federation* (3S)

broker is specified to 50 Mbit per second and a delay of 10 milliseconds. The interconnections between the service provider's internal components are configured to 1 Gbit per second and a delay of 0.1 milliseconds. The SMF implements prioritised queues on the service broker and service load-balancer. Another delay of 10 % of the transport processing effort is added, in order to model the forwarding process on the broker and load-balancer. Another time consuming factor is the deployment delay for the initialisation of new service instances on a service host. This delay is set to 60 seconds. In addition, the delay for the interaction of a service instance with the storage network has to be taken into account as well. This overhead consists of the network delay between the hosts and the storage network and the processing of the storage access in the storage network.

Costs, pricing, and service levels are not configured.

First Outcomes

Examined are the average turn around time of the service requests and the makespan[2], in order to retrieve all requests, corresponding to the analysis published by the CloudSim project. The turn around time is defined as the timespan between the sending of a service request and the receiving of a corresponding response.

Results that are about 5 % slower, compared to the results of CloudSim, have been expected. The overhead should exist due to the modelled queueing in the service broker and service load-balancer, for service deployment delays on the service hosts, and due to the modelling of the network bandwidth demands.

The first outcomes, illustrated in Table 1, indicate that the SMF is able to model the federation of cloud components. The deviations of the results between SMF and CloudSim are acceptable. They can be

2 The term *makespan* is defined by Calheiros et al. as the total timespan between the sending of the first service request and the receiving of the last response of all sent requests.

explained by the imprecision in the mapping of the test case. The previously highlighted model-specific deviations (e.g., modelling of queueing, deployment, and networking) impact the results. Due to the problem in the interpretation of the original resource setup, the number of hosts is only an approximation to the original resource offering. The two frameworks use differing scheduling strategies to route requests to the available sites. The SMF additionally models the delay for accessing storage networks.

7.1.3 *Hybrid Cloud Provisioning*

Test Case

The ability of CloudSim to model scenarios with public and private clouds is evaluated in a small test case. Calheiros et al. describe an experimental setup (Calheiros et al., 2011a), where two providers of cloud resources offer their resources to a single service consumer. The private data centre of provider A is modelled providing 100 computing hosts, each with a capacity of 2 GB of memory, 10 TB of storage, and 1000 MIPS of processing resources. The public data centre of provider B is modelled providing a varying number of computing hosts, each with a capacity of 1.7 GB of memory, 160 GB of storage and 1000 MIPS processing resources. For different simulation runs, the number of hosts in the public data centre is varied between 10 and 100 hosts.

The service consumer sends 10000 requests as cloudlets with a mean resource demand of 1260000 MI (1000 MIPS used for 21 minutes). The resource demand of the corresponding VM for the cloudlet is not explicitly specified, beside that one processor is allocated per VM. The requests are sent to the private data centre. The public data centre is used by the private centre in cases of peak demands.

Eleven scenarios are being compared. In the first scenario, the private data centre does not rely on public resources for request processing. In the other scenarios, in addition to the private resources, the public resources are used to processes the workload. As differing scenarios, simulation runs are setup with 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 hosts.

Experimental Setup

The previously introduced test case is modelled in the SMF as a single service consumer group to service provider relation. A service consumer group invokes 10000 service requests of the same service type. The requests are scheduled with a delay of one millisecond be-

tween each request. The resource demand of the requests is modelled corresponding to the previous test case with 256 MB memory, 1 GB storage, and 100 % processor usage for 1260 seconds.

The resources of the service provider are located on two sites. The first site represents the private cloud resources. It consists of 100 hosts, each with a resource offering of 2 GB memory, 10 TB storage, and 1000 MIPS processing capacity. The second site represents the public cloud resources. It consists of 10 to 100 hosts, dependent on the simulation run configuration. Each public cloud host offers 1.7 GB of memory, 160 GB of storage, and 1000 MIPS processing capacity.

The processing effort for request transport on the service broker and service load-balancer are set to 1 MI. The network traffic generated by a single service request is configured to 10 MB. The available network bandwidth between the service consumer group and the service broker is specified to 50 Mbit per second and a delay of 10 milliseconds. The interconnections between the service provider's internal components are configured to 1 Gbit per second and a delay of 0.1 milliseconds. The SMF implements prioritised queues on the service broker and service load-balancer. Another delay of 10 % of the transport processing effort is added, in order to model the forwarding process on the broker and load-balancer. Another time consuming factor is the deployment delay for the initialisation of new service instances on a service host. This delay is set to 60 seconds.

Costs, pricing, and service levels are not configured.

First Outcomes

Examined are the makespan for all initialised requests and cost prediction for the respective public cloud usage, corresponding to the analysis published by the CloudSim project. For the cost predictions, it is estimated that the costs of a utilised VM are charged with U$ 0.10 per hour of usage.

As in the previous test case, the results are expected to be 5 % slower compared to the results of CloudSim.

The first outcomes, illustrated in Table 2, indicate that the SMF is able to model scenarios comprehending public and private clouds. The deviations of the results between SMF and CloudSim are acceptable. Just like the previous deviations, they can be explained by the imprecision in the mapping and model-specific deviations.

	Makespan (s)	Cost (U$)
CS-P	127155.77	0.00
CS-P+10	115902.34	32.60
CS-P+20	106222.71	60.00
CS-P+30	98195.57	83.30
CS-P+40	91088.37	103.30
CS-P+50	85136.78	120.00
CS-P+60	79776.93	134.60
CS-P+70	75195.84	147.00
CS-P+80	70967.24	160.00
CS-P+90	67238.07	171.00
CS-P+100	64192.89	180.00
SMF-P	126568.1	0.00
SMF-P+10	115211.9	32.00
SMF-P+20	106232.1	60.00
SMF-P+30	98501.3	84.00
SMF-P+40	91109.7	104.00
SMF-P+50	84928.7	120.00
SMF-P+60	79817.5	132.00
SMF-P+70	75788.9	147.00
SMF-P+80	70993.2	160.00
SMF-P+90	68154.6	171.00
SMF-P+100	64371.9	180.00

Table 2: Result Comparison Between CloudSim (CS) and the Simulation Model Framework (SMF) in a Private (P) and Public (+<hosts>) Cloud Setup

7.1.4 *Cloud Computing Environments Under High Load*

Test Case

The ability of CloudSim to model scenarios with QoS-scheduled cloud resources is evaluated in a small test case. Calheiros et al. describe an experimental setup (Calheiros et al., 2011b), with one provider of cloud resources offering its resources to a group of service consumers. The providers' data centre is modelled providing 1000 computing hosts, each with a capacity of 16 GB of memory and 1000 MIPS of processing resources. The storage capacity is not modelled in this test case.

The group of service consumers sends their requests with a random variation of the delay. The request workload is modelled based on a simplified version of the Wikipedia access logs published by Urdaneta et al. (2009), representing a typical web workload for highly frequented websites. The workload varies between 500 and 1200 requests per second on Tuesday to Friday. The deviation on Saturday to Monday is in its maximum 400 to 900 requests per second. The resource demand of a request is specified to occupy the VM resources for 100 milliseconds. The corresponding VM for the cloudlet uses 2 GB of memory and offers a processing capacity of 125 MIPS.

Compared are six scenarios, varying in the number of VMs allocated to process service requests and in the scheduling approach to allocate VMs. For scheduling five static and one self-evolved dynamic approach are being compared. For the static approach, the number of VMs is varied from 50, 75, 100, 125, up to 150 instances serving requests. In case of the dynamic approach, the VM demand is calculated and dynamically adapted by the scheduler. The maximum tolerated turn around time of a service request is specified to 250 milliseconds. Higher response times are assessed as SLA fails.

Experimental Setup

The previously introduced test case is modelled in the SMF based on a thirteen service consumer groups to one service provider relation. The service consumer groups are used to model the variation of the request workload over time. Each group continuously sends service requests with a constant frequency over a specified period of time. For each group, a varying delay is specified before the request sequence is initialised. Table 3 lists the corresponding configuration. Figure 47 illustrates the resulting mean request distribution over time for a 24 hours period. Modelled is a period with high workload between 500 and 1200 requests per second.

	Request Frequency	Initialisation Delay (s)	Period (s)
G-01	500	0	86400
G-02	91	3600	79200
G-03	90	7200	72000
G-04	87	10800	64800
G-05	82	14400	57600
G-06	76	18000	50400
G-07	69	21600	43200
G-08	60	25200	36000
G-09	51	28800	28800
G-10	40	32400	21600
G-11	29	36000	14400
G-12	18	39600	7200
G-13	6	43200	3600

Table 3: Configuration of the Modelled Service Consumer Groups (G)

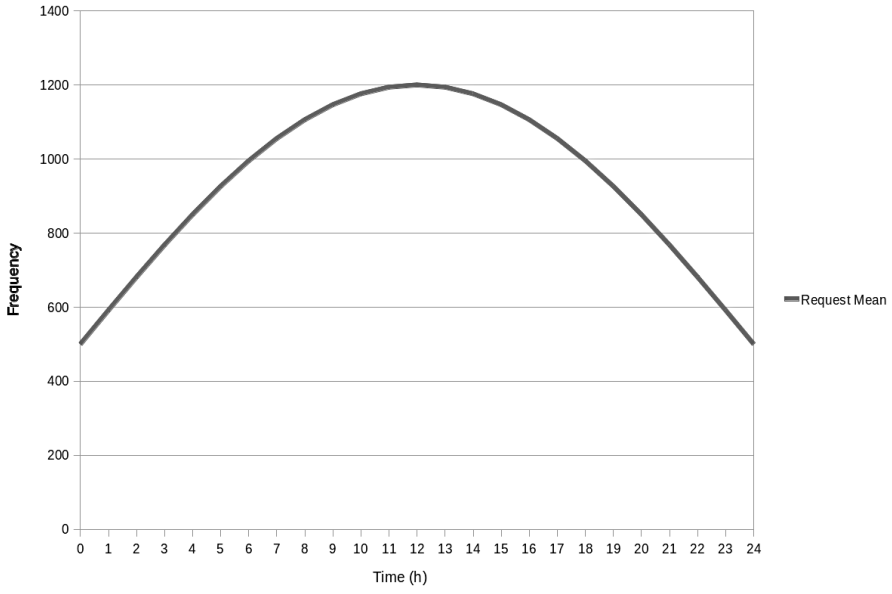


Figure 47: Mean Request Distribution of the Service Consumer Group

The resource demand of the requests is modelled with 2 GB of memory, 1 MB of storage, and 100 % processor usage for 100 milliseconds.

The resources of the service provider are located on one site. As scenario, 19 hosts are chosen, comparable to the setup of Calheiros et al. using 150 instances. The quantity of hosts is calculated using the number of VMs as reference. For 150 VMs with a VM processing capacity of 125 MIPS and a host VM capacity of 1000 MIPS, then this enables 8 VMs per host and results in a demand of 18.75 units of host capacity or 19 hosts. The original number of hosts is insignificant. Each host offers 16 GB of memory, 10 TB of storage, and 1000 MIPS processing capacity.

The processing effort for request transport on the service broker and service load-balancer are set to 1 MI. The network traffic generated by a single service request is configured to 1 KB. The available network bandwidth between the service consumer group and the service broker is specified to 1 Gbit per second and a delay of 10 milliseconds. The interconnections between the service provider's internal components are configured to 2 Gbit per second and a delay of 0.1 milliseconds. The SMF implements prioritised queues on the service broker and service load-balancer. Another delay of 10 % of the transport processing effort is added, in order to model the forwarding process on the broker and load-balancer. The service instances on the service hosts are preinitialised before requests are sent, in order to save the deployment delay.

Costs, pricing, and service levels are not configured.

First Outcomes

Examined are the average response time and corresponding standard deviation for all requests, the resource utilisation in percent, and the percentage of requests failing the SLA. These metrics correspond to the analysis published by the CloudSim project.

In comparison with CloudSim, results that have a 5 % higher average response time, a comparable standard deviation of the response times, a significant higher resource utilisation, and comparable SLA fails have been expected.

The outcomes of Calheiros et al. are depicted as part of Figure 48. Figure 48 depicts two of the variants analysed by Calheiros et al.: CS-Adp as a variant with an adaptive scheduling strategy for resource allocation and CS-150 as a variant modelling fixed resource allocation.

The first outcomes indicate that the SMF is able to model high request loads. The deviations of the results between SMF and CloudSim are acceptable, as it can be explained due to the previously high-

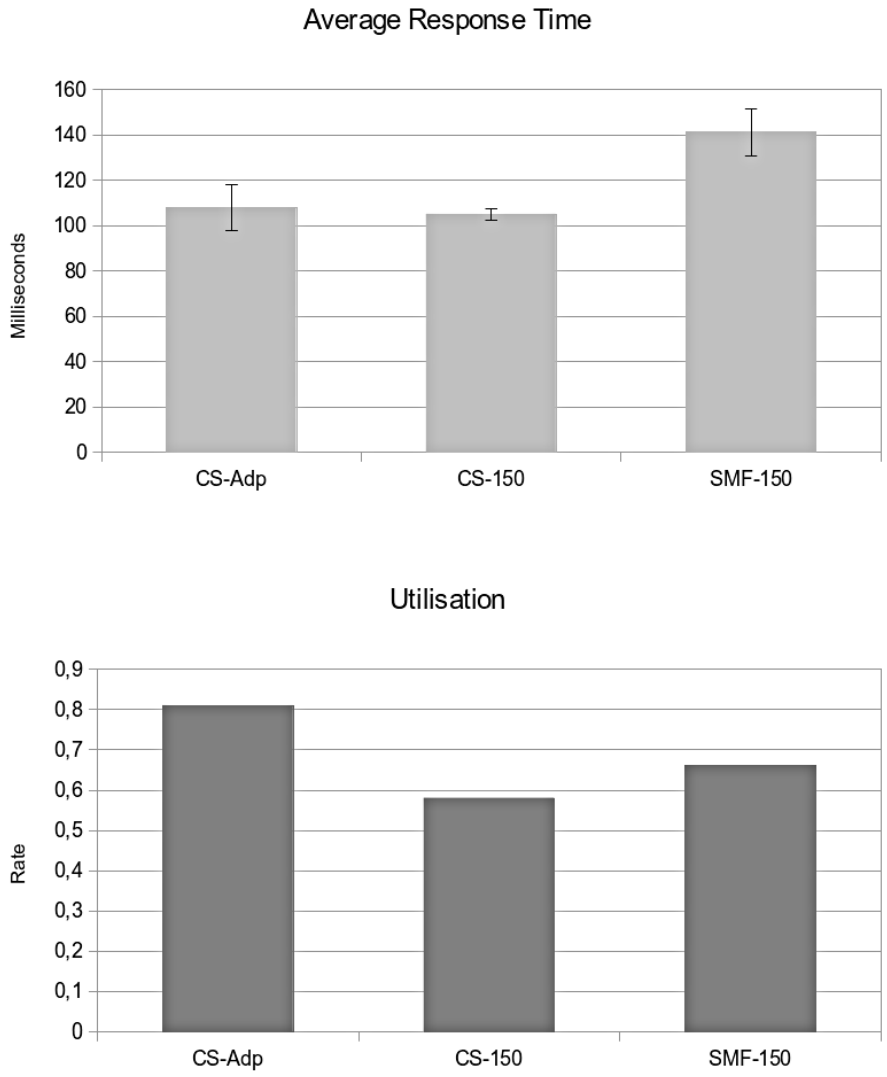


Figure 48: Result Comparison Between CloudSim (CS) and the Simulation Model Framework (SMF) in High Load Scenarios

lighted model-specific deviations (e.g., modelling of queueing, deployment, and networking). The shorter times for request processing allow greater influence for the queueing model and network delays. For both frameworks, the rate of SLA fails is insignificant (0,03 % for SMF).

7.2 EVALUATION OF ADVANCED CLOUD SCENARIOS

The test cases in the previous section demonstrate clearly that the SMF and its underlying model are capable of modelling basic cloud scenarios. In this section, scenarios enabling the quality and profit analyses in UC service cascades are evaluated.

7.2.1 *Simple Service Cascades*

Experimental Setup

This experimental setup shows the ability of SMF to model cloud scenarios including differing SLA, different cost domains, and service cascades using external service offers. As a basis, a modified version of the experimental setup introduced in Section 7.1.4 is used.

One cloud provider offers a single service to two differing consumer groups. The first group contracts a lower service level than the second group. For both groups, the service level criterion is the time-frame between the sending of a service request and the receiving of its processed response. For group one, the maximum tolerated response time is 500 milliseconds. For group two, it is specified to 300 milliseconds. Higher response times are assessed as SLA fails.

The workload modelled for each group is derived from the previously introduced simplified Wikipedia workload. To enable faster simulation runs, only one hundredth of the previously described workload is used. The workload for the first group is modelled as $2/3$ of the request frequency at a time of the workload while, for the second group, the workload is modelled as $1/3$ of the frequency at a time. Corresponding to the mapping in the previous section, each workload modelled in the SMF is based on thirteen service consumer groups, in order to represent the variation of the request workload over time. Figure 47 illustrates the resulting mean request distributions over the simulation time of 24 hours.

All service requests of group one are sent with the lower service level 2. Requests from group two are sent using the better service level 1.

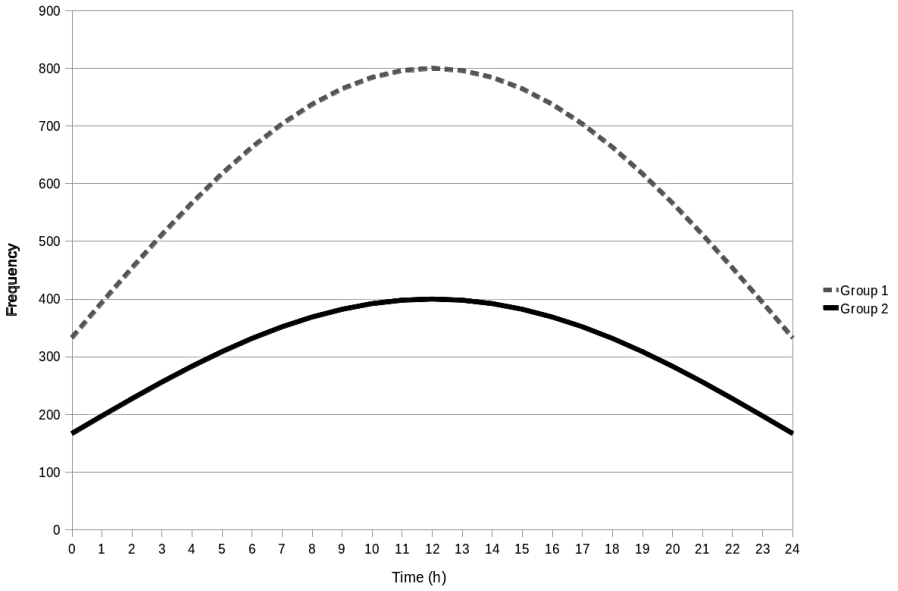


Figure 49: Mean Request Distributions of the Service Consumer Groups

Differing from the previous case, a request to the service offer of provider one includes two sub-requests. The service offer of provider one is introduced as service type A. Both sub-requests, introduced as service types B and C, are each offered by one external provider. Service type A can be described as route planning service for car travelling. It uses service type B as service for traffic density information. Service type C offers information about the regional weather regarding a planned route. This scenario is based on the scenario introduced by Bodendorf and Schobert (2003). All service requests are modelled with an identical resource demand, as introduced in Section 7.1.4.

The data centre of provider one consists of two sites. One site represents a rack as cost domain, which contains newer hardware with a lower demand in electrical power and cooling capacity. Resulting, the cost of operation for hosts in site one is lower than on site two. The second site models a rack containing older hardware with a higher demand in electrical power and cooling capacity. Resulting, site two has higher cost of operations. Service types A is hosted on these sites as demanded, dynamically adapted by the routing decisions of the service broker. Both external providers are each modelled with one site containing sufficient resources to offer their individual service types.

Each external service provider is equipped with identical resources. The resources (10 hosts) are sufficient to process all service requests, even on peak load. The service provider is modelled with two sites. One site represents the older rack with a single host, the other site

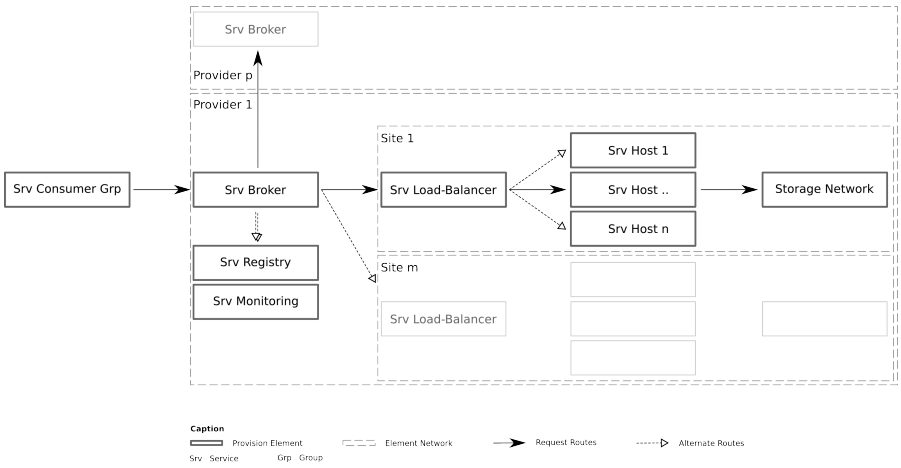


Figure 50: Overview of the Experimental Setup Architecture

represents the newer rack with one or two hosts, dependent on the scenario analysed. Each host offers a capacity of 16 GB of memory, 10 TB of storage, and 125 MIPS processing resources. This setup is derived from the *Static-125* setup in Section 7.1.4.

Compared are the results of four simulation runs. The runs differ in the configuration of the prioritisation of the request queues and in the number of hosts available. Run *FIFO-2* models two hosts as resources for service A using a first-in-first-out queue, with incoming requests being forwarded in the order of their arrival. Run *PRIO-2* uses a prioritised queue instead. The prioritisation criterion is the service level of a request. Run *FIFO-3* and *PRIO-3* use an additional host in the newer rack, while differing in their queue prioritisation.

The effort for transport and the network resources are modelled corresponding to the configuration chosen in Section 7.1.4. Service instances are deployed to and removed from hosts on demand with a deployment delay of 1 millisecond. Costs are basically configured to enable routing decisions for the service broker. The cost configuration demonstrates a symbolic cost difference of 1 cost unit between site 1 and 2.

An overview of the architecture of the specified experimental setup is illustrated in Figure 50.

First Outcomes

Examined are the average response time, corresponding standard deviation, percentage of requests failing the SLA, percentage of dropped requests, resource utilisation in percent, and total costs for resource usage. The average response time, corresponding standard deviation, percentage of requests failing are calculated per SLA group. These

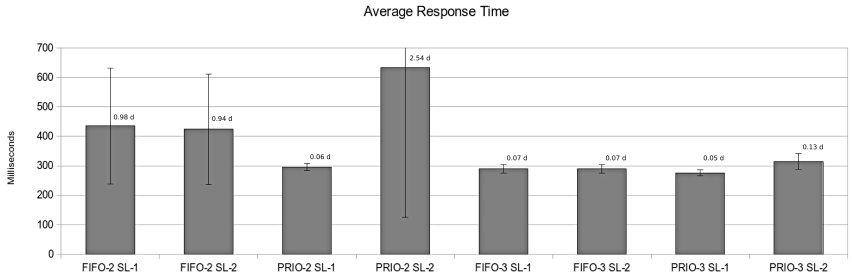


Figure 51: Average Response Times of Runs FIFO-2, PRIO-2, FIFO-3, and PRIO-3 Differentiated by Service Level (SL) and Corresponding Standard Deviation (d)

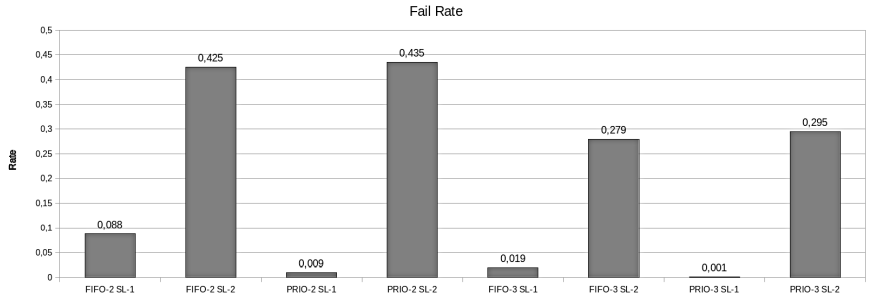


Figure 52: Fail Rate of Runs FIFO-2, PRIO-2, FIFO-3, and PRIO-3 Differentiated by Service Level (SL)

metrics have been chosen, in order to correspond to the analysis from Section 7.1.4.

Figure 51 depicts the average response time and corresponding standard deviation for the runs FIFO-2, PRIO-2, FIFO-3, and PRIO-3. For the purpose of a better comprehension of the relations, the illustration of the standard deviation is scaled up by the factor 200. The original deviation displayed near the illustration is identified by *d*. The average response time is differentiated by the two service levels modelled.

Figure 52 illustrates the percentage of requests failing the SLA. This rate reflects processed requests that failed the maximum allowed latency. It excludes dropped requests. The percentage of dropped requests is depicted in Figure 53.

The resource utilisation in percent and the total costs for resource usage are not varying between the two used queue techniques. Figure 54a depicts the variation in the resource utilisation between the scenarios with two or three hosts. The variation in the total costs for the resource consumption during the processing of all service requests, including the failed requests, is illustrated in Figure 54b. This excludes the dropped requests, as they have not been processed.

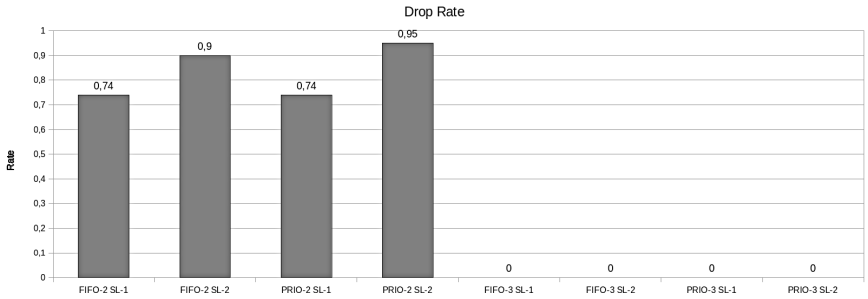
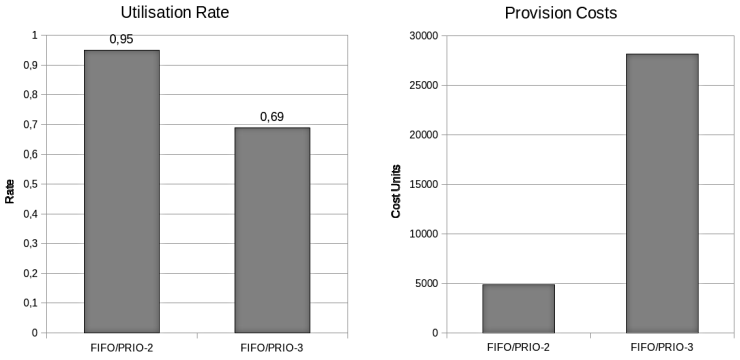


Figure 53: Drop Rate of Runs FIFO-2, PRIO-2, FIFO-3, and PRIO-3 Differentiated by Service Level (SL)



(a) Utilisation Rate (b) Provision Costs

Figure 54: Utilisation Rate and Provision Costs of Runs FIFO-2, PRIO-2, FIFO-3, and PRIO-3

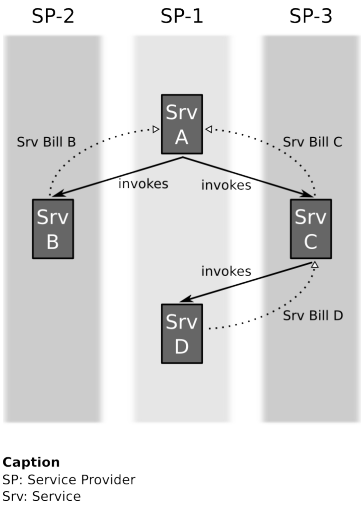


Figure 55: Service Cascade Including Service Bill Flow

The first outcomes indicate that the SMF is able to model cloud scenarios including differing SLAs, different cost domains, and service cascades using external service offers. The outcomes also indicate that the results of SMF enable the analysis of service quality and cost aspects of such cloud scenarios.

7.2.2 *Complex Service Cascades*

Experimental Setup

This experimental setup is a variation of the previous setup in Section 7.2.1. As extension to the previous setup, the service cascade is enhanced to include another service offer. The new service offer models a sub-request of service type C. In addition, the setup is varied, in order to use binding of service instances to sites (service type A is bound to site 1 and 2; service type D is bound to site 3 and 2) and service billing throughout the service cascade. Figure 55 illustrates the modelled service cascade. The figure also depicts the flows of service bills within the cascade.

As variation to the previous setup, the service provider for service type A is modelled with an additional site. Each provider site now offers seven hosts.

First Outcomes

As the previous section already analysed metrics like response time, standard deviation, request fails and drops, resource utilisation, or processing costs, this section introduces a closer look on an exemplary

selection of data retrievable from SMF. Figure 56a illustrates the mean of the processor load on one of the service hosts located on site 1. The graph shows that the host is constantly under load processing requests for a single service type. For comparison, Figure 56b shows the mean of the processor load on one of the service hosts located on site 2. As site 2 is used by two service types, the graph depicts the load share of both service instances. Figure 56c illustrates the mean of the processor load on the service broker. Corresponding, Figure 56d and 56e show the mean of the processor load on the service load-balancers on site 1 and 2. In Figure 56f, the mean of the costs for the request processing of service consumer group 1 is depicted.

7.3 EVALUATION SUMMARY

In Section 7.1, this chapter presents common cloud scenarios. The SMF is used to model such scenarios and to compare the results of the simulations runs with the outcomes of Calheiros et al.. The evaluation shows the ability of SMF to model common cloud scenarios with comparable outcomes to the works of Calheiros et al..

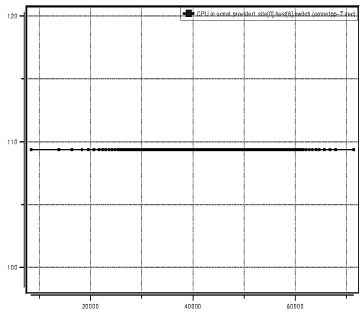
Section 7.2 evaluates the ability of SMF to model cascaded SOC service offers hosted on IaaS. The analysed scenarios are enhancements of a scenario used by Calheiros et al.. The scenario is expanded by a service cascade introduced by Bodendorf and Schobert. To further clarify the potential of SMF, the service cascade is advanced to a cascade including internal and external service offers, cascaded sub-requests, resource binding, and cost forwarding in service cascades. The first outcomes of these simulation runs show the ability of SMF to model advanced cloud scenarios with complex pay-per-use service cascades.

In comparison to the models introduced in Section 6.2, the SMF offers a more detailed resource abstraction compared to Buyya's CloudSim. The CloudSim model misses a service broker with support for economical load-balancing and the modelling of the BSLA approach.

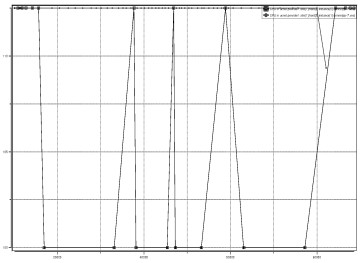
GroudSim offers very basic building blocks to model UC service cascades compared to the detail level achievable using SMF.

Another related work introduced in the previous chapter is GreenCloud. In comparison to SMF, GreenCloud is not sufficient to map UC service cascades in the context of this thesis. The model's building blocks are able to basically model the relation between consumer, service, and resource. More complex UC characteristics, as introduced in Section 2.6, can not be modelled.

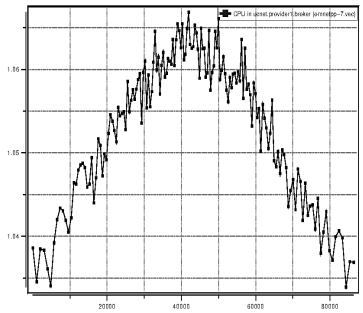
Summarising the related work, using SMF permits the modelling of economical load-balancing based on BSLA by using complex UC service cascades in a simulation framework, in order to analyse economi-



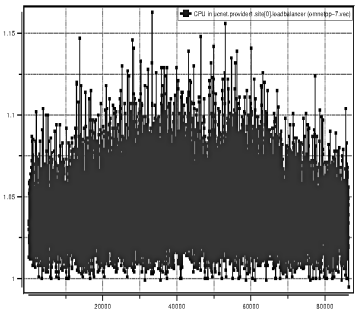
(a) Mean of Processor Load on One of the Service Hosts on Site 1



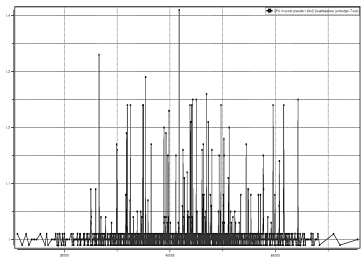
(b) Mean of Processor Load on One of the Service Hosts on Site 2



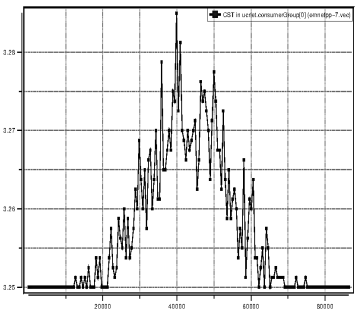
(c) Mean of Processor Load on the Service Broker



(d) Mean of Processor Load on the Service Load-Balancer of Site 1



(e) Mean of Processor Load on the Service Load-Balancer of Site 2



(f) Mean of Request Processing Costs for Service Consumer Group 1

Figure 56: Examples of Analysable SMF Metrics

cal and technical hypothetical questions of service providers throughout a service's life cycle.

The SMF has the ability to model an infinite number of consumer groups and service providers with complex and unlimited service usage relations among these actors. The scheduling properties allow the modelling of request distribution over time, up to complex distributions that can be achieved by the combination of multiple consumer groups.

The modelling of pricing for resource usage is supported as fixed price per request or by resource usage during processing. The evaluation has only rudimentary made use of these features, as the selected scenarios did not offer the ability for a useful inclusion of payment analyses in the given service cascades. Further details about the modelling of costs and pricing are given in Section 6.5.

As the most elaborated part of this thesis, the evaluation demonstrates the successful interrelation of the thesis outcomes in an executable model that enables control and prediction on service quality and profit.

The implementation of the SMF limits the use to simulation models that address multi-tier architectures. For the modelling of architectures that do not correspond to the proposed data model the SMF is not suitable.

Summarising, this chapter strongly indicates the capabilities of the thesis contributions to represent the core relation of UC, in order to enable the analysis, development, and operation of all in all more cost efficient SOC service offers. This optimisation of cost efficiency is enabled by the underlying delivery framework, which is the basis for improved control and prediction of service quality and costs.

As lessons learned from the first outcomes of the evaluation, six topics are identified:

- The data model should be extended to include power consumption per service host. This would induce a more comfortable analysis of the simulation outcomes;
- The SMF implementation should be extended to:
  - Support an additional abstraction layer, in order to map time varying schedules in the service consumer component. This would improve the overview of complex request models;
  - More efficiently handle large request amounts by reimplementing often used code sequences. This would speed up the calculation of complex simulation runs;

- Support a more granular configuration of the logging of the result vectors. This has been implemented already. The configuration enables smaller log files, what speeds up the analysis and leads to less system load during simulation runs;
- Enable parallel calculations of the simulation runs, in order to speed up simulation runs for complex models with long simulation times.
- Include fixed costs per time unit for each resource offer.

Concluding, the first outcomes of the evaluation of the SMF and underlying delivery framework clearly demonstrate a novel degree in the clarification and optimisation of the association between the UC business model and SOC architectures hosted in Cloud environments.

8 CONCLUSIONS

The research aims to optimise the control of service quality and profit for SOC service offers based on Utility Computing business models. The elaborated delivery framework helps service providers optimise future UC service provision throughout their entire life cycle.

To evolve an effective strategy to enable the optimisation of control of service quality and profit in service life cycles, related work and corresponding service provision requirements are collected and analysed.

The following approaches from related work are reviewed:

- Provision models for UC platforms to model either UC, Cloud, Grid, or application cluster models;
- Usage-centred assurance of service quality in Cloud Computing.

The review shows that no existing approach addresses all the requirements identified.

The collection of the service provision requirements comprehend the analysis of:

- Utility Computing-specific relations inside a generic service life cycle;
- Utility Computing-specific requirements on provision quality control;
- Primary requirements on provision platforms for SOC service offers based on UC business models.

Concerning the specific requirements of SOC service offers based on UC business models, the analysis reveals the demand for approaches to improve:

- Description of the customer-service-resource relation in corresponding service life cycles;
- Service quality control concerning such service offers;
- Analysis of complex service cascades.

Corresponding approaches are elaborated in a combined usage-centred provision approach introduced as delivery framework. The included approaches model the requirements, control structures, and information demands, in order to enable the optimisation of control of service quality and profit in the considered service life cycles.

8.1 CONTRIBUTIONS

This thesis offers discrete approaches for the following problems. The analysis of the relations within service life cycles reveals a demand for an improved representation of the customer-service-resource relation. Particularly, service level agreeing, life cycle information exchange, support for make-or-buy decisions, economic-efficient quality control, and the analysis of complex service cascades are identified as problems that prevent economic availability management and effective pay-per-use pricing models.

Due to Liang-Jie Zhang (2007, p. 328), a common model for UC service life cycles would be helpful, in order to track the variations and analyse the impacts among the life cycle phases.

Due to Mendoza (2007, p. 228), a key issue in service development is the consideration of estimated usage behaviour to improve quality assurance.

This research provides approaches to close these gaps. Some aspects of these findings challenge current thinking regarding the pay-per-use management in service cascades, the merge of *business, consumer, and technical* quality views, and the indirect feasibility rating for service cascades. This research highlights the relations within a service life cycle and links them to the specific requirements imposed for service provision quality and corresponding provision platforms. Therefore, the achievements of this research lie beyond the evolved approaches in the progressing of a deeper understanding of how UC service quality is related to a service's life cycle.

As comprehensive outcome, this thesis introduces an executable model for a delivery framework that enables control and prediction on service quality and profit for its user.

As distinct outcomes, the following approaches are introduced:

- Technology-independent core provision model for Utility Computing platforms

   Component Model  Workflow Model

  The approach models a generic technology-independent component architecture for incorporation into an architectural pattern, in order to prepare a software/provision architecture for UC services. The model allows its users the consistent integration of the identified UC requirements into software/provision architectures.

- Concept for usage-centred assurance of service quality

   Data Model  Business SLA  Usage Pattern  Decision Tree

  The concept of usage-centred assurance of service quality merges the *business, consumer, and technical* quality views, as a significant aspect of this research. By introducing a usage-centred quality specification, it enables the monitoring and control of provision quality contracted on service usage rather than on technical resource limits.

- Technology-abstracted resource and cost simulation model

   Simulation Model

  A simulation model for multi-tier UC service architectures has been built based on the previous findings. Constitutive simulation runs are valuable tools to enable the analysis of service cascades in terms of resource demand, price and cost evaluation, and structural weaknesses.

The major contributions of this thesis have been published to, refereed by and discussed with the community in the Cloud Computing field of research. The positive feedback from reviewers indicates that this research is making appropriate and useful contributions to this field.

Beside the scientific impact, portions of the contributions already have significant impact on industrial partners of this research. Three business partners[1] sponsored particular parts of the research project and used the outcomes to evolve new business models for their future Cloud Computing strategies.

1 rh-tec AG, Löhne, Germany; secco advanced GmbH, Grossostheim, Germany; T-Systems International GmbH, Frankfurt, Germany.

8.2 LIMITATIONS

Limitations of this thesis are discussed from two points of view regarding the quality of method and the quality of findings.

Analysing the chosen research method, the evaluation of the findings based on the comparison of simulation results of two simulation model frameworks can be discussed. Although, a commonly accepted simulation model framework is chosen and the relevant test cases have been rebuilt, the comparison of the simulation results to a real world system would improve the generalisability of the gathered results. Nevertheless, the gathered results are sufficient as first outcomes, in order to enable a resilient focus on the quality of the findings.

Although the evaluation has been calculated on on average 6 cores over four weeks, parallel simulation implementation on significantly more cores is required to enable the simulation of real world systems modelling more than 36 hours of service usage, complex service cascades, and multiple large scale data centres.

Analysing the findings of this thesis, some limitations can be identified. The usage-centred data model only considers the stakeholder relations of incoming service requests, so that relations to third-party service providers are excluded. The simulation model framework only addresses multi-tier architectures. The component model excludes some non-minimal requirements that are considerable in this context: embedding of legacy applications (Req_{39}), administration-related requirements ($Req_{39\text{-}43}$), and deployment policies (Req_{44}).

The work presented should be reviewed on a regular basis to reflect new versions of ITIL and COBIT as they appear.

8.3 FUTURE WORK

In order to build on the contributions of this thesis, a number of research aims are being proposed in the following:

- Estimation of resource demand of service requests

  To enable more efficient simulation models in the future, the estimation of the resource demand of individual service requests has to be optimised. To increase the precision of predictions, the individual resource demand of service requests of differing consumer groups in real world systems has to be gathered. Simple metering of request amounts and a general rough estimation of processing demands do not offer the aimed precision.

- Racks as new grouping component

  It might improve the modelling of real world systems when racks as additional layer for the grouping of resources are considered. This might be helpful to map existing data centres, where energy consumption per rack is a relevant issue.

- Representation of power consumption & cooling capacities

  The expansion of the data model to directly map the power consumption of resources is considered as useful. This would enable a more efficient analysis of the results of simulation runs. Also included in the expansion should be a maximum power capacity per rack and/or site. In addition, the power capacity could be linked to boundaries for cooling capacity, if applicable.

- Virtualisation layer as expansion

  The expansion of the core provision model to enable the modelling of a virtualisation layer should be verified. As it is not considered to be useful in the SOC context, it might broaden the usability of the delivery framework in other service provider contexts.

- Runtime estimations of resource usage

  It could be useful to verify, whether a specialised implementation of the simulation model framework can be used at runtime, embedded in a service broker implementation. This might improve future routing decisions and subsequent resource utilisation.

8.4 TECHNOLOGY REVIEW

Concluding this research, the contributions are reviewed in a broader context. The initial aim to support service providers that are pushed into the role of utilities by their customer's expectations, this thesis unquestionable offers a suitable approach to adapt their current service life cycles. The contributions offer a reliable foundation for the future development of Cloud Computing in general.

In future IT Clouds, approaches to enable real-time service provision will gather a growing impact. This demand will be pushed by the growing extension of autonomous heterogeneous business processes that are based on Cloud services. The underlying service composition mechanisms will be based on adaptation patterns relying on quality metrics. The monitoring and prediction of service quality in Cloud Computing will become a major issue due to the increase of

new Cloud layers, decentralisation, and deployment dynamic. This induces the demand for techniques to correlate monitoring data and utilisation predictions from different sources in differing formats.

On the other hand, there will be a rising demand for service consumers to adequately monitor their SLA. For accurate cost predictions, the measuring of service quality and the prediction of expectable service quality for heterogeneous service-oriented systems has to work closer together. This includes the demand to correlate short-term and long-term predictions on both sides, service consumers and providers.

Data management on large scale will demand new approaches, when federated in the Cloud, specifically when certain security policies should be considered (e.g., geographical distribution, competitor sealing).

For all these presented future challenges, the evolved delivery framework offers a serious basis for service provision.

BIBLIOGRAPHY

Afzal, A., McGough, A. S., and Darlington, J. (2008). Capacity planning and scheduling in grid computing environments. *Future Generation Computer Systems*, 24:404–414. ACM ID: 1350010. (Cited on page 57.)

Almasi, G. S. and Gottlieb, A. (1989). *Highly Parallel Computing*. Benjamin-Cummings, Redwood City, CA. (Cited on page 33.)

Amazon (2011). AWS elastic beanstalk. http://aws.amazon.com/elasticbeanstalk/. (Cited on page 7.)

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. (2003). *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. (Cited on page 30.)

Andrzejak, A., Arlitt, M., and Rolia, J. (2002). Bounding the resource savings of utility computing models. Technical report, Hewlett-Packard Laboratories. (Cited on pages 26 and 33.)

APM Group, TSO, and Cabinet Office (2011). ITIL. http://www.itil-officialsite.com. (Cited on pages 34 and 91.)

Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalantar, M., Krishnakumar, S., Pazel, D. P., Pershing, J., and Rochwerger, B. (2001). Oceano-SLA based management of a computing utility. In *2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings*, pages 855–868. IEEE. (Cited on page 53.)

Arsanjani, A., Zhang, L.-J., Ellis, M., Allam, A., and Channabasavaiah, K. (2007). S3: A service-oriented reference architecture. *IT Professional*, 9(3):10–17. (Cited on pages x, 51, and 52.)

Bader, D. A. and Pennington, R. (2001). Cluster computing: Applications. *The International Journal of High Performance Computing*, pages 181–185. (Cited on page 33.)

Ball, P. (2001). Introduction to discrete event simulation. http://masters.donntu.edu.ua/2006/kita/kondrakhin/library/art6.htm. (Cited on page 139.)

Banks, J. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice: Modelling, Estimation and Control*. John Wiley & Sons, 1. auflage edition. (Cited on page 138.)

Banks, J., Carson, J. S., Nelson, B. L., and Nicol, D. M. (2009). *Discrete-Event System Simulation*. Prentice Hall, 5 edition. (Cited on page 139.)

Batelle, J. and O'Reilly, T. (2004). Opening welcome, the state of the internet industry, web 2.0 conference, san francisco. http://itc.conversationsnetwork.org/shows/detail270.html. (Cited on page 9.)

Beard, H. (2008). *Cloud Computing Best Practices for Managing and Measuring Processes for On-Demand Computing, Applications and Data Centers in the Cloud with Slas*. Lulu.com. (Cited on page 34.)

Bi, J., Zhu, Z., Tian, R., and Wang, Q. (2010). Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 370–377. (Cited on page 58.)

Bieberstein, N. (2006). *Service-oriented architecture compass: business value, planning, and enterprise roadmap*. FT Press. (Cited on page 29.)

Bitran, G. and Caldentey, R. (2003). An overview of pricing models for revenue management. *Manufacturing & Service Operations Management*, 5(3):203–229. (Cited on page 27.)

Bodendorf, F. and Schobert, A. (2003). Integration von web-services in ein kundenportal. *HMD - Praxis Wirtschaftsinform.*, 234. (Cited on pages 37 and 170.)

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web services architecture. http://www.w3.org/TR/ws-arch/. (Cited on pages 29 and 43.)

Bourke, T. (2001). *Server load balancing - help for network administrators.* O'Reilly. (Cited on page 34.)

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2012). Extensible markup language (XML) 1.0 (Fifth edition). http://www.w3.org/TR/xml/. (Cited on page 122.)

Britannica Online Encyclopedia (2011). public utility. http://www.britannica.com/EBchecked/topic/482523/public-utility. (Cited on page 24.)

Bunker, G. and Thomson, D. (2006). *Delivering Utility Computing: Business-driven IT Optimization*. John Wiley & Sons. (Cited on page 26.)

Buyya, R., Broberg, J., and Goscinski, A. M. (2011). *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, 1. auflage edition. (Cited on pages 65, 68, 69, and 75.)

Buyya, R., Ranjan, R., and Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In *Proceedings of the 7th High Performance Computing and Simulation Conference*, Leipzig, Germany. IEEE Press, New York, USA. (Cited on page 140.)

Buyya, R., Ranjan, R., and Calheiros, R. N. (2010). InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. In Hsu, C., Yang, L., Park, J., and Yeo, S., editors, *Algorithms and Architectures for Parallel Processing, Pt 1, Proceedings*, volume 6081, pages 13–31. Springer-Verlag Berlin, Berlin. WOS:000279552800002. (Cited on pages x and 40.)

Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., and Buyya, R. (2011a). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software - Practice and Experience*, 41(1):23–50. (Cited on pages xi, 140, 160, and 162.)

Calheiros, R. N., Ranjan, R., and Buyya, R. (2011b). Virtual machine provisioning based on analytical performance and QoS in cloud computing environments. In Gao, G. R. and Tseng, Y.-C., editors, *ICPP*, pages 295–304. IEEE. (Cited on page 165.)

Castane, G. G., Nunez, A., Filgueira, R., and Carretero, J. (2012). Dimensioning scientific computing systems to improve performance of map-reduce based applications. *Procedia Computer Science*, 9:226–235. (Cited on page 141.)

Chalmers, A. F. (1999). *What Is This Thing Called Science?* Open University Press, 3rd edition. (Cited on page 24.)

Chee, B. J. and Franklin, C. J. (2010). *Cloud Computing: Technologies and Strategies of the Ubiquitous Data Center*. CRC Press, 1 edition. (Cited on page 26.)

Chen, P. P.-S. (1976). The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36. (Cited on pages 117 and 119.)

Chou, T. (2011). *Introduction to Cloud Computing*. Active Book Press, LLC. (Cited on page 26.)

Citrix (2011). Xen hypervisor. http://www.xen.org/. (Cited on page 31.)

Clarke, R. J. (2005). Research methodologies. (Cited on page 21.)

Cohen, J., Darlington, J., and Lee, W. (2008). Payment and negotiation for the next generation grid and web. *Concurrency and Computation: Practice and Experience*, 20(3):239–251. (Cited on pages 28 and 51.)

Cohen, J., Lee, W., Darlington, J., and McGough, A. S. (2006). A service-oriented utility grid architecture utilising pay-per-use resources. In *First International Conference on Communication System Software and Middleware, 2006. Comsware 2006*. IEEE. (Cited on page 51.)

Cooper, R. B. (1981). *Introduction to Queuing Theory*. George Washington University, 2nd edition. (Cited on page 138.)

D Souza, D. F. and Wills, A. C. (1998). *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*. Addison-Wesley Professional, 1 edition. (Cited on page 29.)

Darlington, J., Cohen, J., and Lee, W. (2006). An architecture for a next-generation internet based on web services and utility computing. In *15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE '06*, pages 169–174. IEEE. (Cited on page 51.)

Dimitrakos, T., Gaeta, M., Ritrovato, P., Serhan, B., Wesner, S., and Wulf, K. (2002). Grid based application service provision. Oxford, UK. (Cited on pages x, 49, and 50.)

Dimitrakos, T., Randal, D. M., Yuan, F., Gaeta, M., Laria, G., Ritrovato, P., Serhan, B., Wesner, S., and Wulf, K. (2003). An emerging architecture enabling grid based application service provision. In *EDOC*, pages 240–251. IEEE Computer Society. (Cited on page 49.)

Dobson, G. and Sanchez-Macian, A. (2006). Towards unified QoS/SLA ontologies. In *IEEE Services Computing Workshops, 2006. SCW '06*, pages 169–174. IEEE. (Cited on page 55.)

Dodig-Crnkovic, G. (2002). Scientific methods in computer science. In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*. (Cited on page 19.)

Dodig-Crnkovic, G. (2003). Shifting the paradigm of philosophy of science: Philosophy of information and a new renaissance. *Minds and Machines*, 13(4):521–536. 10.1023/A:1026248701090. (Cited on page 22.)

Dodig-Crnkovic, G. (2004). Theory of science. Technical Report MRTC report ISSN 1404-3041 ISRN MDH-MRTC-64/2001-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University. (Cited on page 23.)

Dodig-Crnkovic, G. (2009). Theory of science 1. (Cited on page 24.)

Ellram, L. M. (1995). Total cost of ownership: an analysis approach for purchasing. *International Journal of Physical Distribution & Logistics Management*, 25(8):4–23. (Cited on page 10.)

Elsaesser, W. (2006). *ITIL einfuehren und umsetzen: Leitfaden fuer effizientes IT-Management durch Prozessorientierung*. Carl Hanser Verlag GmbH & CO. KG, 2., erweiterte auflage edition. (Cited on page 91.)

Erl, T. (2005). *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall. (Cited on page 29.)

Erl, T. (2007). *SOA Principles of Service Design*. Prentice Hall International, 1 edition. (Cited on page 117.)

Fitzsimmons, J. A. and Fitzsimmons, M. J. (2006). *Service Management: Operations, Strategy, Information Technology*. Mcgraw-Hill Professional, 5th ed. edition. (Cited on page 28.)

Fong, L. L., Kalantar, M., Pazel, D. P., Goldszmidt, G. S., Fakhouri, S., and Krishnakumar, S. (2002). Dynamic resource management in an eUtility. In *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, pages 727–740. IEEE. (Cited on page 53.)

Foster, I., Gannon, D., Kishimoto, H., and Von Reich, J. (2004). Open grid services architecture use cases. Information Document. (Cited on pages 48, 81, 83, 85, and 86.)

Foster, I. and Kesselman, C. (2003). *The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2 edition. (Cited on pages 33 and 48.)

Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J., and Von Reich, J. (2005). The open grid services architecture, version 1.0. Informational Document. (Cited on pages x, 48, and 49.)

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. (Cited on page 26.)

Freitas, A. L., Parlavantzas, N., and Pazat, J.-L. (2010). A QoS assurance framework for distributed infrastructures. In *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, MONA '10, pages 1–8. ACM ID: 1929567. (Cited on page 58.)

Garg, S. K. and Buyya, R. (2011). NetworkCloudSim: modelling parallel applications in cloud simulations. In *UCC*, pages 105–113. IEEE Computer Society. (Cited on page 140.)

Grayling, A. C. (1999). *Philosophy 1: A Guide through the Subject*. Oxford University Press, USA. (Cited on pages 22 and 23.)

Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall (2007). Cloud computing. *IBM DeveloperWorks*. (Cited on page 32.)

Gribble, S. D., Welsh, M., von Behren, J. R., Brewer, E. A., Culler, D. E., Borisov, N., Czerwinski, S. E., Gummadi, R., Hill, J. R., Joseph, A. D., Katz, R. H., Mao, Z. M., Ross, S., and Zhao, B. Y. (2001). The ninja architecture for robust internet-scale systems and services. *Computer Networks*, 35(4):473–497. (Cited on page 53.)

Groenroos, C. (2000). *Service Management and Marketing: A Customer Relationship Management Approach*. Wiley & Sons, 2 edition. (Cited on page 28.)

Haas, H. and Brown, A. (2004). Web services glossary. http://www.w3.org/TR/ws-gloss/. (Cited on pages 29 and 73.)

Hansson, H. (2009). Research methods in CS. (Cited on page 22.)

Harris, D. (2008). Why 'Grid' doesn't sell. http://www.hpcinthecloud.com/hpccloud/2008-03-24/why_grid_doesnt_sell.html. (Cited on page 33.)

Hayes, B. (2008). Cloud computing. *Communications of the ACM*, 51(7):9–11. (Cited on page 32.)

Heap, D. G. (2003). Taurus - taxonomy of actual utilization of real UNIX and windows servers. Technical report, IBM Corporation. (Cited on page 26.)

Heckmann, B. (2007). Service provision in a utility computing environment. In *Proceedings of the Third Collaborative Research Symposium*

on Security, E-Learning, Internet and Networking, pages 185–198, Plymouth, UK. Lulu.com. (Cited on pages 81, 105, 110, and 201.)

Heckmann, B. (2009). Service provision in an utility computing environment. Technical report, Science and Technology, Computing and Mathematics, University of Plymouth. (Cited on pages 116, 117, and 119.)

Heckmann, B. and Phippen, A. (2010). Quantitative and qualitative description of the consumer to provider relation in the context of utility computing. In *Proceedings of the Eighth International Network Conference (INC 2010)*, pages 335–344, Heidelberg, Germany. (Cited on pages 73, 74, 78, 95, 116, 117, 119, and 201.)

Heckmann, B., Phippen, A. D., Moore, R. C., and Wentzel, C. (2011). Agreeing on and controlling business service levels in service-oriented architectures. *International Transactions on Systems Science and Applications*, Vol. 7(No. 3/4):173–178. (Cited on pages 75, 79, 121, 122, 124, and 201.)

Heckmann, B., Phippen, A. D., Moore, R. C., and Wentzel, C. (2012a). Agreeing on and controlling service levels in service-oriented architectures. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Service Science*, pages 267–270, Porto, Portugal. INSTICC - Institute for Systems and Technologies of Information, Control and Communication. (Cited on pages 75, 79, 121, 122, 124, and 201.)

Heckmann, B., Stengel, I., Phippen, A., and Turetschek, G. (2009). Utility computing simulation. In *ESM'2009 The 2009 European Simulation and Modelling Conference*, pages 175–180, Leicester, United Kingdom. EUROSIS-ETI. (Cited on pages 74, 146, and 201.)

Heckmann, B., Turetschek, G., and Phippen, A. (2008). A technology-abstracted approach to a utility computing simulation framework. In *Proceedings of the Fourth Collaborative Research Symposium on Security, E-learning, Internet and Networking*, pages 155–165, Wrexham, UK. (Cited on pages 74 and 201.)

Heckmann, B., Zinn, M., Phippen, A. D., Moore, R. C., and Wentzel, C. (2012b). Economic efficiency control on data centre resources in heterogeneous cost scenarios. In *ICITST-2012 Proceedings*, pages 675–679, London, UK. Infonomics Society, UK. (Cited on pages 40, 94, 127, and 201.)

Hoeing, A. (2010). *Orchestrating Secure Workflows for Cloud and Grid Services*. PhD thesis, Technische Universitaet Berlin, OPUS. (Cited on pages x and 50.)

Humm, B. (2008). Was ist eigentlich ein service? In *Softwaretechnik-Trends 28*, pages 8–11. (Cited on page 29.)

ISACA, editor (2005). *COBIT 4.0*. United States of America. (Cited on page 92.)

JBoss (2011). JBoss application server. http://www.jboss.org/. (Cited on page 33.)

Jong, W. R. and Betti, A. (2008). The classical model of science: a millennia-old model of scientific rationality. *Synthese*, 174:185–203. (Cited on page 24.)

JTC1/SC22/WG21 (2012). ISO/IEC JTC1/SC22/WG21 - the c++ standards committee. http://www.open-std.org/jtc1/sc22/wg21/. (Cited on page 147.)

Kertesz, A., Kecskemeti, G., and Brandic, I. (2009). An SLA-based resource virtualization approach for on-demand service provision. In *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, VTDC '09, pages 27–34, New York, NY, USA. ACM. (Cited on pages x and 41.)

Kim, H. J. and Paek, K. H. (2005). Promoting the application service provision (ASP) model. In *DEEC*, pages 95–102. IEEE Computer Society. (Cited on page 49.)

Kishimoto, H. (2003). OGSA usecase matrix. (Cited on page 48.)

Kliazovich, D., Bouvry, P., and Khan, S. (2010a). DENS: data center energy-efficient network-aware scheduling. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 69 –75. (Cited on page 142.)

Kliazovich, D., Bouvry, P., and Khan, S. (2010b). GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, pages 1–21. (Cited on pages xii, 142, and 143.)

Kusic, D., Kephart, J., Hanson, J., Kandasamy, N., and Jiang, G. (2008). Power and performance management of virtualized computing environments via lookahead control. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 3–12. (Cited on page 58.)

Liang, Q., Chung, J.-y., Miller, S., and Ouyang, Y. (2006). Service pattern discovery of web service mining in web service registry-repository. In *2006 IEEE International Conference on e-Business En-*

gineering (ICEBE'06), pages 286–293, Shanghai, China. (Cited on pages 99 and 115.)

Liang, Q., Miller, S., and Chung, J.-Y. (2005). Service mining for web service composition. In *Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on.*, pages 470 – 475. (Cited on pages 99 and 115.)

Liang-Jie Zhang (2007). *Services Computing: Core Enabling Technology of the Modern Services Industry*. Tsinghua University Press; Springer, Beijing; Berlin; New York. (Cited on pages x, 3, 28, 34, 47, 48, 65, 69, and 180.)

Liddell, H. G. and Scott, R. (2011). Philosophy, a greek-english lexicon, perseus. http://www.perseus.tufts.edu. (Cited on page 22.)

Lim, S.-H., Sharma, B., Nam, G., Kim, E. K., and Das, C. (2009). MDCSim: a multi-tier data center simulation, platform. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1–9. (Cited on page 57.)

Liu, L., Wang, H., Liu, X., Jin, X., He, W. B., Wang, Q. B., and Chen, Y. (2009). GreenCloud: a new architecture for green data center. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, ICAC-INDST '09, pages 29–38, New York, NY, USA. ACM. (Cited on pages x, 41, and 42.)

Machiraju, V., Rolia, J., and Van Moorsel, A. (2002). Quality of business driven service composition and utility computing. *COMPUTING, SOFTWARE TECHNOLOGY LABORATORY, HP LABORATORIES PALO ALTO*. (Cited on page 53.)

MacLaren, J., Newhouse, S., Haupt, T., Keahey, K., and Lee, W. (2006). Grid economy use cases. (Cited on page 48.)

Maglaras, C. and Meissner, J. (2006). Dynamic pricing strategies for multi-product revenue management problems. *Manufacturing & Service Operations Management (MSOM)*, 8(2):136–148. (Cited on page 27.)

Marks, E. A. and Lozano, B. (2010). *Executive's Guide to Cloud Computing*. Wiley, 1 edition. (Cited on pages x, 9, 32, 41, and 42.)

Maximilien, E. M. and Singh, M. P. (2005). Toward web services interaction styles. In *2005 IEEE International Conference on Services Computing*, volume 1, pages 147– 154 vol.1. IEEE. (Cited on page 29.)

McGough, A. S., Lee, W., and Darlington, J. (2006). ICENI II. In *First International Conference on Communication System Software and Middleware, 2006. Comsware 2006*. IEEE. (Cited on pages x and 51.)

Mell, P. and Grance, T. (2010). The NIST definition of cloud computing. (Cited on pages 30, 31, and 32.)

Mendoza, A. (2007). *Utility Computing Technologies, Standards, and Strategies*. Artech House Inc. (Cited on pages x, 26, 43, 44, 65, 66, 68, 70, 71, and 180.)

Microsoft (2011). Windows azure platform. http://www.microsoft.com/windowsazure/. (Cited on page 7.)

Monroe, K. B. (2003). *Pricing*. McGraw-Hill/Irwin, Boston, Mass., 3. ed., internat. ed edition. (Cited on page 27.)

Nathuji, R., Kansal, A., and Ghaffarkhah, A. (2010). Q-clouds: managing performance interference effects for QoS-aware clouds. In Morin, C. and Muller, G., editors, *EuroSys*, pages 237–250. ACM. (Cited on pages 55 and 56.)

Natis, Y. V., Pezzini, M., Iijima, K., and Favata, R. (2008). Magic quadrant for enterprise application servers, 2Q08. http://www.gartner.com/id=655409. (Cited on page 34.)

Neel, D. (2002). The utility computing promise. http://www.infoworld.com/d/networking/utility-computing-promise-807. (Cited on page 26.)

Nunez, A., Castane, G., Vazquez-Poletti, J., Caminero, A., Carretero, J., and Llorente, I. (2011a). Design of a flexible and scalable hypervisor module for simulating cloud computing environments. In *2011 International Symposium on Performance Evaluation of Computer Telecommunication Systems (SPECTS)*, pages 265 –270. (Cited on page 141.)

Nunez, A., Vazquez-Poletti, J., Caminero, A., Carretero, J., and Llorente, I. (2011b). Design of a new cloud computing simulation platform. In Murgante, B., Gervasi, O., Iglesias, A., Taniar, D., and Apduhan, B., editors, *Computational Science and Its Applications - ICCSA 2011*, volume 6784 of *Lecture Notes in Computer Science*, pages 582–593. Springer Berlin / Heidelberg. (Cited on page 141.)

Nunez, A., Vazquez-Poletti, J., Caminero, A., Castane, G., Carretero, J., and Llorente, I. (2012). iCanCloud: a flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209. (Cited on pages xii, 141, and 142.)

Oasis (2006). Reference model for service oriented architecture. *Public Review Draft 2*. (Cited on page 29.)

OMG, O. M. G. (2011). Unified modeling language. http://www.omg.org/spec/UML/. (Cited on page 121.)

Oracle (2011). GlassFish application server. http://glassfish.java.net/. (Cited on page 33.)

Ostermann, S., Plankensteiner, K., and Prodan, R. (2011a). Using a new event-based simulation framework for investigating resource provisioning in clouds. *Scientific Programming*, 19(2-3):161–178. (Cited on page 141.)

Ostermann, S., Plankensteiner, K., Prodan, R., and Fahringer, T. (2011b). GroudSim: an event-based simulation framework for computational grids and clouds. In *Euro-Par 2010 Parallel Processing Workshops - HeteroPar, HPCC, HiBB, CoreGrid, UCHPC, HPCF, PROPER, CCPI, VHPC, Ischia, Italy, August 31-September 3, 2010, Revised Selected Papers*, volume 6586 of *Lecture Notes in Computer Science*, pages 305–313. Springer. (Cited on page 141.)

Osterwalder, A. (2004). *The Business Model Ontology - a proposition in a design science approach*. PhD thesis, University of Lausanne, Switzerland. (Cited on page 25.)

Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *4th International Conference on Web Information Systems Engineering (WISE'03)*, pages 3–12. (Cited on page 29.)

Pazel, D. P., Eilam, T., Fong, L. L., Kalantar, M., Appleby, K., and Goldszmidt, G. (2002). Neptune: A dynamic resource allocation and planning system for a cluster computing utility. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002*, pages 57–57. IEEE. (Cited on page 53.)

Peoples, C., Parr, G., and McClean, S. (2011). Energy-aware data centre management. In *Communications (NCC), 2011 National Conference on*, pages 1–5. (Cited on page 58.)

Phan, T. and Li, W.-S. (2010). Vertical load distribution for cloud computing via multiple implementation options. In Furht, B. and Escalante, A., editors, *Handbook of Cloud Computing*, pages 277–308. Springer US. 10.1007/978-1-4419-6524-0_12. (Cited on pages x, 44, and 45.)

Postel, J. (1981). Internet control message protocol - RFC 792. http://tools.ietf.org/html/rfc792. (Cited on page 123.)

Ralston, A., Reilly, E. D., and Hemmendinger, D., editors (2003). *Encyclopedia of Computer Science*. Wiley, 4 edition. (Cited on pages 138 and 139.)

Rappa, M. (2003). Business models on the web. http://digitalenterprise.org/models/models.html. (Cited on page 26.)

Rappa, M. A. (2004). The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42. (Cited on pages 24 and 26.)

Roth, P. F. (1987). Discrete, continuous and combined simulation. In *Proceedings of the 19th conference on Winter simulation*, WSC '87, pages 25–29, New York, NY, USA. ACM. (Cited on page 138.)

Rust, G. (2009). Internal technical report, secco advanced GmbH, grossostheim, germany. (Cited on pages 11 and 12.)

Salesforce.com (2011). CRM software & online CRM system. http://www.salesforce.com. (Cited on page 8.)

Seppanen, M. and Makinen, S. (2005). Business model concepts: a review with case illustration. In *Engineering Management Conference, 2005. Proceedings. 2005 IEEE International*, volume 1, pages 376– 380. IEEE. (Cited on page 25.)

Shigang, C. and Nahrstedt, K. (1998). An overview of quality of service routing for next-generation high-speed networks: problems and solutions. *IEEE Network*, 12(6):64–79. (Cited on page 57.)

Singh, M. P. and Huhns, M. N. (2005). *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley Computer Publishing, New York. (Cited on page 29.)

Smith, M. A. and Kumar, R. L. (2004). A theory of application service provider (ASP) use from a client perspective. *Information & Management*, 41(8):977–1002. (Cited on page 49.)

Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. T. (2009). Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22. (Cited on page 46.)

Speitkamp, B. and Bichler, M. (2010). A mathematical programming approach for server consolidation problems in virtualized data centers. *Services Computing, IEEE Transactions on*, 3(4):266–278. (Cited on page 58.)

Stanoevska-Slabeva, K., Wozniak, T., and Ristol, S., editors (2009). *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. Springer, 1 edition. (Cited on page 33.)

Stantchev, V. and Schroepfer, C. (2009). Negotiating and enforcing QoS and SLAs in grid and cloud computing. In Abdennadher, N. and Petcu, D., editors, *Advances in Grid and Pervasive Computing*, volume 5529, pages 25–35. Springer Berlin Heidelberg, Berlin, Heidelberg. (Cited on page 56.)

Treadwell, J. (2005). Open grid services architecture glossary of terms. http://www.ggf.org/documents/GFD.44.pdf. (Cited on pages 86 and 88.)

Urdaneta, G., Pierre, G., and van Steen, M. (2009). Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11):1830–1845. (Cited on page 165.)

Urgaonkar, B., Pacifici, G., Shenoy, P. J., Spreitzer, M., and Tantawi, A. N. (2005a). An analytical model for multi-tier internet services and its applications. page 291. ACM Press. (Cited on page 52.)

Urgaonkar, B., Shenoy, P. J., Chandra, A., and Goyal, P. (2005b). Dynamic provisioning of multi-tier internet applications. pages 217–228. IEEE. (Cited on pages xi, 52, and 53.)

Van Moorsel, A. (2001). Metrics for the internet age: Quality of experience and quality of business. *5TH PERFORMABILITY WORKSHOP*. (Cited on pages xi, 53, and 55.)

Varga, A. (2001). The OMNeT++ discrete event simulation system. In *ESM'2001 Proceedings of the 15th European Simulation Multiconference*, Prague, Czech Republic. (Cited on page 147.)

Varga, A. and Hornig, R. (2008). An overview of the OMNeT++ simulation environment. In Molnar, S., Heath, J., Dalle, O., and Wainer, G. A., editors, *SimuTools*, page 60. ICST. (Cited on page 147.)

Villegas, D. and Sadjadi, S. M. (2011). Mapping non-functional requirements to cloud applications. In *SEKE*, pages 527–532. Knowledge Systems Institute Graduate School. (Cited on pages x and 46.)

VMware (2011a). Cloud foundry. http://cloudfoundry.org/. (Cited on page 7.)

VMware (2011b). VMware virtualization. http://www.vmware.com/virtualization/. (Cited on page 31.)

von Behren, J. R., Brewer, E. A., Borisov, N., Chen, M., Welsh, M., MacDonald, J., Lau, J., and Culler, D. E. (2002). Ninja: A framework for network services. In *ATEC '02: Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*, pages 87–102, Berkeley, CA, USA. USENIX Association. (Cited on page 53.)

Von Reich, J. (2004). Open grid services architecture: Second tier use cases. Draft. (Cited on pages 48, 86, 88, and 90.)

W3C (2007). SOAP specifications. http://www.w3.org/TR/soap/. (Cited on page 43.)

Wang, Y. and Wang, X. (2010). Power optimization with performance assurance for multi-tier applications in virtualized data centers. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 512–519. (Cited on page 58.)

Weiss, A. (2007). Computing in the clouds. *netWorker*, 11(4):16–25. (Cited on page 32.)

Welsh, M. and Culler, D. E. (2003). Adaptive overload control for busy internet servers. In *USENIX Symposium on Internet Technologies and Systems*. (Cited on page 53.)

Wesner, S., Serhan, B., Dimitrakos, T., Randal, D. M., Ritrovato, P., and Laria, G. (2004). Overview of an architecture enabling grid based application service provision. In Dikaiakos, M. D., editor, *European Across Grids Conference*, volume 3165 of *Lecture Notes in Computer Science*, pages 113–118. Springer. (Cited on page 49.)

West, D. B. (1999). *Introduction to Graph Theory*. Prentice Hall, 2nd ed. edition. (Cited on page 12.)

Wickremasinghe, B., Calheiros, R. N., and Buyya, R. (2010). CloudAnalyst: a CloudSim-Based visual modeller for analysing cloud computing environments and applications. In *Proceedings of the 24th International Conference on Advanced Information Networking and Applications*, pages 446–452, Perth, Australia. IEEE Computer Society. (Cited on page 140.)

Winter, S. (2000). Quantitative vs. qualitative methoden. http://imihome.imi.uni-karlsruhe.de/nquantitative_vs_qualitative_methoden_b.html. (Cited on page 21.)

Wolski, R., Spring, N. T., and Hayes, J. (1999). The network weather service: a distributed resource performance forecasting service for

metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768. (Cited on page 57.)

Xipeng, X. and Ni, L. M. (1999). Internet QoS: a big picture. *IEEE Network*, 13(2):8–18. (Cited on page 57.)

Yeo, C. S., de Assuncao, M. D., Yu, J., Sulistio, A., Venugopal, S., Placek, M., and Buyya, R. (2006). Utility computing and global grids. *cs/0605056*. (Cited on page 26.)

Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J., and Chang, H. (2004). QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327. (Cited on page 57.)

Zimmermann, H. (1980). OSI reference model – the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425– 432. (Cited on page 11.)

PUBLICATIONS

A. Agreeing on and Controlling Business Service Levels in Service-Oriented Architectures,
(Heckmann et al., 2011);

B. Agreeing on and Controlling Service Levels in Service-Oriented Architectures,
(Heckmann et al., 2012a);

C. A Technology-Abstracted Approach to a Utility Computing Simulation Framework,
(Heckmann et al., 2008);

D. Economic Efficiency Control on Data Centre Resources in Heterogeneous Cost Scenarios,
Heckmann et al. (2012b);

E. Quantitative and Qualitative Description of the Consumer to Provider Relation in the Context of Utility Computing,
(Heckmann and Phippen, 2010);

F. Service Provision in an Utility Computing Environment,
(Heckmann, 2007);

G. Utility Computing Simulation,
(Heckmann et al., 2009).

siwn.org.uk

International Transactions on Systems Science and Applications
Volume 7 Number 3/4 December 2011 pp. 173-178

iTSSA

Agreeing on and Controlling Business Service Levels in Service-Oriented Architectures

B. HECKMANN [a], A. D. PHIPPEN [b], R. MOORE [a] and C. WENTZEL [a]

[a] *University of Applied Sciences Darmstadt, Germany*
[b] *University of Plymouth, Great Britain*

Abstract: This paper introduces Business Service Levels (BSLs) as a generalised concept to agree on feasibility and workload of business processes hosted in service-oriented architectures. BSLs loosen the coupling between technical service provision and business service consumption by offering an alternative to technical service level agreements. To controll the BSL compliance at runtime an technical approach is introduced and implemented as proof-of-concept. It retrieves state or load information from a technical monitoring system and technical topology information from a CMDB. Based on those, this component estimates a business process's feasibility and workload. To maintain the contracted BSLs it actively controlls the request flow towards the services.

Keywords: availability, business, monitoring, reliability, services.

1. Introduction

State of the art management of service levels in service-oriented architectures aims to track and keep certain levels of technical measurements. These objectives are formalising as service level agreements (SLAs) consisting of technical thresholds, corresponding actions to keep them and penalties when failing.[1]

From a business perspective, feasibility and workload of business processes are the only relevant measurements. In this paper the workload of a business process reflects the relation of planned to actual workflow actions in a certain time frame, whereby it is estimated that all actions considered rely on IT services to be conducted. Accordingly the feasibility of a process predicts the availability of the underlying IT resources for a certain time frame and includes its workload state. This paper introduces a new abstraction layer in order to agree on the feasibility and workload of a business process instead of technical thresholds of the underlying technology. This layer is called the Business Service Level (BSL). BSLs establish a black box around service capacity and technical implementation, thus loosening the coupling between technical service provision and business service consumption on the level of service agreements.

This paper introduces Business Service Levels and offers an approach to agree on and keep them from a provider perspective.

2. Background

Starting from the business perspective, the feasibility of business processes is essential for economical success. But not only is the state of feasibility of interest, but also the workload of a process, as

[1] in the context of ITIL

sai: itssa.2011.12.041 ISSN 1751-1461 (print) ISSN 1751-147X (cd-rom)

certain workloads may reach critical technical or organisational thresholds. From the perspective of technical operations, offering such state information for business processes based on service-oriented architectures, can become a complex challenge. This complexity arises from highly meshed service cascades and redundant alternate service offers (e.g., for load balancing). Committing to technical thresholds in classical SLAs unnecessarily restricts both business and technical operations. The introduction of the BSL as a new level of service agreement loosens this coupling.

In this context the research objectives of this paper are as follows:

- Identify qualified technical indicators that clearly relate to the feasibility or workload of the modeled business process as basis for Business Service Level Agreements (BSLA).
- Provide a sufficient description that represents the indicators identified for BSLAs.
- Introduce a technical approach to monitor and enforce BSLAs during technical operations.

3. Related Work

Related works mainly address technical perspectives on service levels (SL) in service-oriented architectures (SOA). Starting with the IT architecture perspective, publications target SL description in the context of performance modelling [1], SLA-driven development [2] and dependability throughout the life cycle [3].

The main focus within the IT architecture perspective is SL operations management. The most common approach for operations management could be described as distribute-and-enforce. In [4,5, 6,7,8] highly detailed SLAs are defined, distributed and then enforced on each member of the service cascade. The complexity of this approach increases with the complexity of the given cascade.

Other approaches, such as [9] or this paper, decouple the complexity of SL operations management from the complexity of the managed service cascades. The advantage of this paper's approach is that it is embedded in a whole life cycle concept [10,11].

Beside the IT architecture perspective, technical operations is another perspective concerned with service levels. Technical operations focus on keeping track off certain levels of technical measurements which can be grouped together under the term *technical monitoring*. Three types of monitoring can be distinguished: active, passive and agent-based [12].

Other approaches focus on the monitoring of business process workloads. This kind of monitoring can be referred to under the term *workflow monitoring*. Its focus is on the workflow state rather than underlying technical measurements [13]. In contrast, [14] and this paper aim to provide a non-intrusive workflow monitoring approach combined with active SL management. This paper broadens this approach to incorporate technical monitoring data and address general IT services based on IP networks.

4. Research Approach

This research was conducted in cooperation with secco[2] as business partner, introducing the problem statement. The joint research aims at providing a concept to monitor the feasibility and workload of business processes, hosted using on multi-tier IT service provision infrastructures, without active technical monitoring. The following steps were taken to reach this goal:

1. Analyse of a typical multi-tier IT service provision infrastructure operated in the context of our business partner.
2. Design of a generalised concept to track feasibility and workload of business process hosted based on multi-tier infrastructures.
3. Implementation of a BSLA monitoring framework as proof-of-concept.

[2] secco advanced GmbH, Grossostheim, Germany

5. Multi-tier Infrastructure Analyses

In the context of secco, SOA infrastructures consist of seven horizontal layers, see the filled layers in Figure 1. Business processes are represented by technical workflows as entry layer. Business functionality within these workflows is provided by the orchestration of application layer services, such as web services. These services are hosted on the application infrastructure layer (e.g., within database systems or application servers). All software components from upper layers are deployed on the operating system layer, each instance running in a virtual machine on the virtualisation

layer. The virtual hardware is mapped to resources on the physical systems layer. As the final layer, the network services connect these systems relying on resources such as routers, switches or domain name services (DNS). Complementary to the previously described horizontal multi-tier SOA infrastructure, there is the vertical technical monitoring layer. It collects, tracks and evaluates technical measuring points of the given horizontal layers, such as running processes, log file analysis, network stack availability, CPU load, RAM or storage usage.

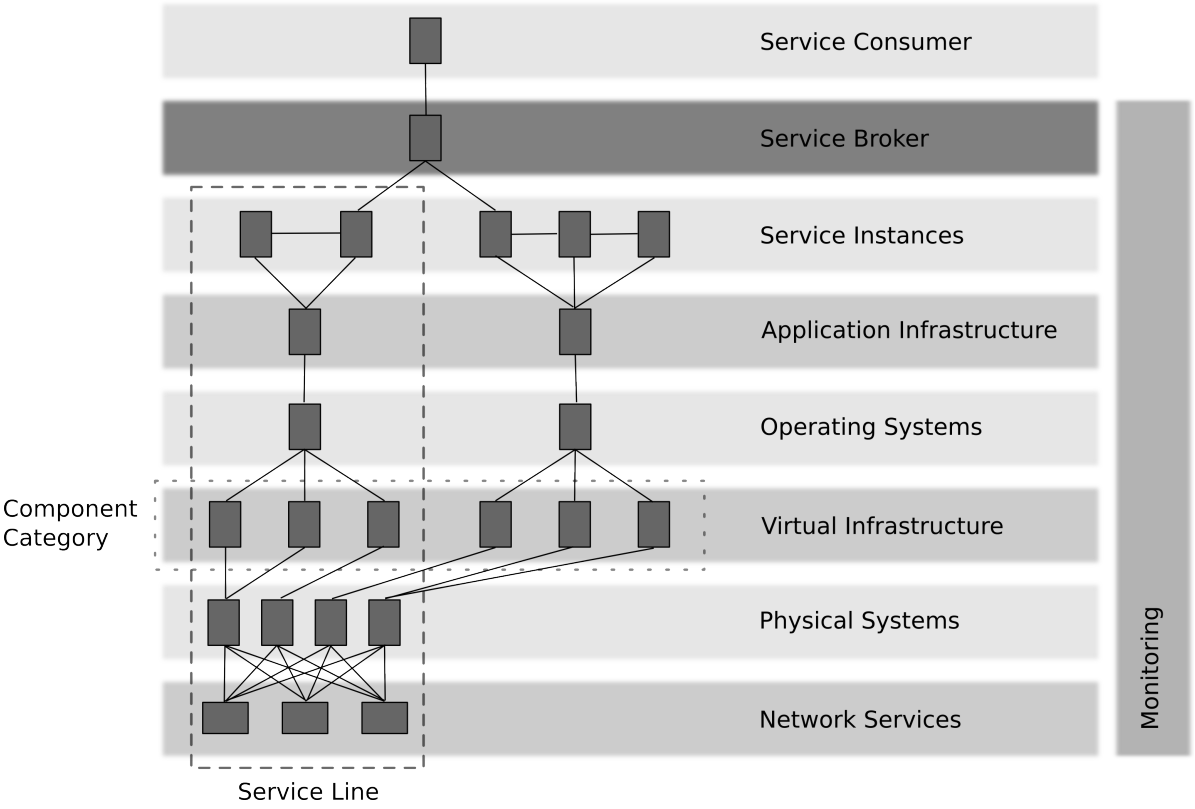


Figure 1. Multi-tier architecture including Service Broker, Service Line and Component Category as example

To manage these complex infrastructures the following major points must be considered:

- Handling of service cascades with redundant service offers.
- Seamless integration of internal and external service providers.
- Support for dynamic coupling[3] between service consumer and provider.

The technical monitoring solutions Amberpoint, Progress Actional, SOA Manager Service Manager, Oracle Enterprise Manager SOA Management Pack and OpTier CoreFirst evaluated by [15] do not offer sufficient information to gather quantifications of failure impacts and reliable conclusions on the feasibility of the implemented business processes in the given SOA infrastructures. Specifically

[3] intermediate logic that changes the invocation target of a service request at runtime

the analysis of the actual feasibility of a business process in scenarios with redundant service offers fails.

6. Solution Design

To agree on feasibility and workload of business processes this paper proposes Business Service Level Agreements as abstraction layer to ease the definition of the level of service between consumer and provider. The core of a BSLA is the description of the expected usage behaviour. It is optionally enriched by the declaration of maintenance windows, maximum downtimes, fines, prices or other service level attributes. For the description of the usage behaviour this paper proposes the use of Usage Patterns [10]. Usage Patterns address the description of the quantitative consumer-provider-relation in terms of request frequency and complexity. Applying BSLA to contract on service offers provides the starting point for business process feasibility and workload analyses by specifying the *contracted usage*.

In opposite to the contracted usage, the *monitored usage* reflects the current request amount and resource utilisation within the IT infrastructure. The business process's workload is then determined by comparing its contracted and monitored usage, assuming all demanded infrastructure components are technically available. To enable monitoring of the request amount this paper proposes the use of a centralised request routing component, named Service Broker [16] (see Figure 1). The Service Broker provides a measuring point for request amounts per business process, which represent the process's workload, taking the contracted usage as reference. The business process's feasibility is lead back from its workload combined with information about the technical availability of all infrastructure components hosting the process.

The aggregation of technical monitoring information in service cascades hosting business processes is addressed by the *topology graph*. The topology graph is introduced by this paper to reflect the functional dependencies between the components in an IT infrastructure. To build the topology graph infrastructure components can be retrieved from a CMDB[4], if available. To represent redundant service offers within a topology graph *service lines* are introduced. A service line is a logical group of infrastructure components that are necessary to provide an application layer service. To enable the aggregation of the resource utilisation of service line spanning resources this paper introduces the term *component category*. Component categories logically group infrastructure components that provide similar functionalities (e.g., application servers, which provide the hosting of application layer services as functionality), see Figure 1 as example.

To calculate the resource utilisation, each topology graph node is enriched with interpreted technical monitoring information. This enriched graph is introduced as *availability graph* in this paper. Thereby different detail levels of monitoring data interpretation are possible:

- State-based availability
  In case of state-based analyses the availability of a graph node is lead back by interpreting state-related technical monitoring data of the represented infrastructure component, such as ICMP ping states retrieved from a technical monitoring system. Interpretation of this data limits its results to two simple states: available and non-available. This variant is simpler to impose, but is less significant when deducting the feasibility of constitutive business processes.
- Load-based availability
  For load-based analyses the availability of a graph node is estimated using a comparison of its current resource utilisation with its maximum resource capacity. Thereby the resource utilisation is calculated based on a customisable combination of load-related technical monitoring data. This paper proposes the use of CPU load, RAM capacity and disk space. The interpretation of such load-related data enables the provision of proportional result values reflecting the

[4] referring to the context of ITIL

current availability of the represented infrastructure component (e.g., 20 % resource utilisation of the DNS server). The administrative overhead of this variant is to be estimated as higher, but enables a more significant deduction of the feasibility of constitutive business processes.

- Mixed-mode availability
  In mixed-mode graphs the availability of a node can either be interpreted based on state or load-related data. This variant combines both previous variants to a customisable compromise of administration effort and feasibility deduction precision.

The Service Broker estimates a business process as feasible if in the availability graph all state-based nodes of at least one service line are available and the resource utilisation of all load-based component categories offer sufficient reserves to process the *estimated usage*. The estimated usage for a given time frame is calculated by subtracting the monitored usage for a given business process from its contracted usage. In strictly state-based availability graphs only the process workload is taken into account when calculating the estimated usage, otherwise also the resource utilisation is incorporated.

The Service Broker also provides the ability for load- and/ or cost-based routing decisions at runtime to internal or external service providers. Cost-based routing at runtime is in detail described in [10]. Load-based decisions include the ability to drop requests to prevent overloading the underling IT infrastructure.

7. Proof-of-Concept Implementation

As proof-of-concept an application was implemented representing the availability graph of a small exemplary business process. Due to the specifications of the business partner a state-based availability graph is realised. As technical monitoring system the implementation bears on the Open Source Software Zabbix [17]. The application depends on a given XML configuration file representing the topology graph. The given example topology graph consists of nine nodes representing two service lines. Each node description is enriched with a reference to its Zabbix database identifier. The application extracts the nodes' state values from the underlying monitoring system. It autonomously determines the service lines and aggregates their availability states. The validation of the determination reliability is subject of future research.

As first outcomes, the implementation shows the ability to realise state-based availability graphs based on technical monitoring data. It further introduces an approach to autonomously identify service lines in topology graphs.

8. Further Work

Complementary to the proof-of-concept implementation of state-based availability graphs the proof for load-based availability implementations is currently being developed in cooperation with another business partner. This research also aims to include cost-based decisions at runtime or time of deployment. For highly meshed service cascades the Service Broker could rely on simulation to forecast the business process feasibility for a certain time frame (e.g., based on [11]). It could be useful to periodically simulate the behaviour of service cascades at runtime in short time frames. For example the results could represent the expected resource utilisation within the next 15 minutes. The Service Broker could take these results into account for its routing decisions. This could be especially useful in scenarios with long running service requests.

9. Conclusions

This paper introduces Business Service Levels as new abstraction layer to agree on feasibility and workload of business processes between business and IT management. This eliminates the need for

classic service level agreements based on technical thresholds. The research objectives in this context have been to identify qualified technical indicators that clearly relate to the feasibility and workload of business processes, provide a sufficient description for BSLAs and introduce a technical approach to monitor and enforce BSLAs during operations. Related works mainly address the classical technical perspectives onto service level management. To close this gap, this paper analyses a typical multi-tier IT service provision infrastructure operated by the project's business partner. It designes a generalised concept to track feasibility and workload of business processes hosted in such infrastructures. Concluding, a proof-of-concept implementation demonstrates the capabilities of this approach.

References

[1] P. C. Brebner, "Performance modeling for service oriented architectures", in Companion of the 30th international conference on Software engineering, Leipzig, Germany, 2008, p. 953-954.

[2] V. Muthusamy and H.-A. Jacobsen, "SLA-driven distributed application development", in Proceedings of the 3rd workshop on Middleware for service oriented computing, Leuven, Belgium, 2008, p. 31-36.

[3] V. Stantchev and M. Malek, "Addressing Dependability throughout the SOA Life Cycle", IEEE Transactions on Services Computing, Bd. 99, Nr. PrePrints, 2010.

[4] Y. Chen, S. Iyer, D. Milojicic, and A. Sahai, "A systematic and practical approach to generating policies from service level objectives", in Integrated Network Management, 2009. IM âĂŹ09. IFIP/IEEE International Symposium on, 2009, p. 89 -96.

[5] V. Muthusamy, H.-A. Jacobsen, T. Chau, A. Chan, and P. Coulthard, "SLA-driven business process management in SOA", in Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research, Ontario, Canada, 2009, p. 86-100.

[6] Chih-Hao Hsu, Yun-Wei Liao, and Chien-Pang Kuo, "Disassembling SLAs for follow-up processes in an SOA system", in 2008 11th International Conference on Computer and Information Technology, Khulna, Bangladesh, 2008, p. 37-42.

[7] C. Raibulet and M. Massarelli, "Managing Non-functional Aspects in SOA through SLA", in 2008 19th International Conference on Database and Expert Systems Applications, Turin, Italy, 2008, p. 701-705.

[8] Guijun Wang u. a., "Service Level Management using QoS Monitoring, Diagnostics, and Adaptation for Networked Enterprise Systems", in Ninth IEEE International EDOC Enterprise Computing Conference (EDOCâĂŹ05), Enschede, The Netherlands, 2005, p. 239-250.

[9] V. Stantchev and C. Schroepfer, "Techniques for service level enforcement in web-services based systems", in Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, Linz, Austria, 2008, p. 7-14.

[10] B. Heckmann and A. Phippen, "Quantitative and Qualitative Description of the Consumer to Provider Relation in the Context of Utility Computing", in Proceedings of the Eighth International Network Conference (INC 2010), Heidelberg, Germany, 2010, p. 335-344.

[11] B. Heckmann, I. Stengel, A. Phippen, and G. Turetschek, "Utility Computing Simulation", in ESMâĂŹ2009 The 2009 European Simulation and Modelling Conference, Leicester, United Kingdom, 2009, p. 175-180.

[12] A. Utlik and N. Alexeyev, "Comparative analysis of Service Level Agreement monitoring methods", in Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET), 2010 International Conference on, 2010, p. 346 -346.

[13] T. Ou, W. Sun, C. Guo, and J. Li, "Visualized Monitoring of Virtual Business Process for SOA", in Proceedings of the 2008 IEEE International Conference on e-Business Engineering, 2008, p. 767-770.

[14] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL", in Proceeding of the 17th international conference on World Wide Web, Beijing, China, 2008, p. 815-824.

[15] G. Rust, "Internal technical report, secco advanced GmbH, Grossostheim, Germany", Juni-2009.

[16] B. Heckmann, "Service Provision in a Utility Computing Environment", in Proceedings of the Third Collaborative Research Symposium on Security, E-Learning, Internet and Networking, Plymouth, UK, 2007, p. 185-198.

[17] A. Vladishev, "Zabbix :: An Enterprise-Class Open Source Distributed Monitoring Solution", Juli-2010. [Online]. Available: http://www.zabbix.com/. [Accessed: 21-Juli-2010].

AGREEING ON AND CONTROLLING SERVICE LEVELS IN SERVICE-ORIENTED ARCHITECTURES

Benjamin Heckmann[1], Andrew D. Phippen[2], Ronald C. Moore[1] and Christoph Wentzel[1]
[1]*University of Applied Sciences Darmstadt, Haardtring 100, 64295, Darmstadt, Germany*
[2]*University of Plymouth, Drake Circus, PL4 8AA, Plymouth, U.K.*
benjamin.heckmann@gmx.de

Keywords: Availability, Business Processes, Monitoring, Reliability, SOA.

Abstract: Business Service Level Agreements (BSLAs) are introduced as a generalised concept to agree on feasibility and workload of business processes hosted in service-oriented architectures as an alternative to technical SLA. Based on BSLAs an according approach to control feasibility at runtime is presented.

1 INTRODUCTION

In Service-oriented Architectures (SOA) Service Level Agreements (SLAs) (Group et al., 2011) are used to specify technical thresholds, corresponding actions to keep them and penalties when failing. From a business point of view, only the feasibility of business processes is of interest for economical success. Business processes are feasible if a given workload is processed in a certain maximum time frame. Workloads of processes are the sum of all current actively processed process instances. Determining whether a business process based on a SOA is feasible can be complex for highly meshed service cascades including redundant alternate service offers (e.g., for load balancing).

This research is conducted in cooperation with secco[1] as business partner, introducing the problem statement. The applied research aims at providing a concept to monitor the feasibility and workload of business processes, hosted on multi-tier IT service provision infrastructures for SOA services, without the need for active technical monitoring. To elaborate an solution approach, a typical multi-tier IT service provision infrastructure operated in the context of our business partner is analysed in section 3. A generalised concept to track feasibility and workload of business processes hosted based on multi-tier infrastructures is elaborated in section 4. In section 5, a BSLA monitoring framework is implemented as a proof-of-concept.

[1]secco advanced GmbH, Grossostheim, Germany, http://www.seccoadvanced.de.

2 RELATED WORKS

This paper offers an approach to technically converge the quality-related ontologies of service, experience, and business as introduced in (Van Moorsel, 2001; Dobson and Sanchez-Macian, 2006). Most other authors address technical perspectives on SLA in SOA. From the IT architecture point of view, authors deal with SLA descriptions of performance modelling (Brebner, 2008), SLA-driven development (Muthusamy et al., 2009) or dependability throughout the life cycle (Stantchev and Malek, 2010). In operations management SLA are mostly enforced through an distribute-and-enforce tactic. By (Hsu et al., 2008; Raibulet and Massarelli, 2008; Chen et al., 2009; Muthusamy and Jacobsen, 2008) highly detailed SLAs are defined, distributed and then enforced on each member of a service cascade. The complexity to manage such approaches increases with the complexity of the given cascade. (Stantchev and Schroepfer, 2008) decouples SLA operations management from the complexity of a service cascade. This paper presents a similar approach and advances it by embedding BSLA in a whole life cycle concept (Heckmann and Phippen, 2010). Technical operations is focused on the *technical monitoring* of technical resource thresholds. Three types can be distinguished: active, passive and agent-based (Utlik and Alexeyev, 2010). Other authors propose the *workflow monitoring* of business process workloads. It is focused on the workflow state rather than underlying technical measurements (Ou et al., 2008). In contrast, (Moser et al., 2008) aims to provide a non-intrusive workflow mon-

itoring approach combined with active SLA management. This paper broadens this approach to incorporate technical monitoring data and address general IT services based on IP networks.

3 MULTI-TIER INFRASTRUCTURE ANALYSIS

In the context of secco, SOA infrastructures consist of 7 horizontal layers, shown in Figure 1. Business processes are represented by technical workflows acting as service consumers on the top layer. Business functionality is provided by the orchestration (Andrews et al., 2003) of application layer services, for example web services (Haas and Brown, 2004). These service instances are hosted on the application infrastructure layer (e.g., within database systems or application servers). All software components from upper layers are deployed on the operating system layer, each instance running in a virtual machine on the virtual infrastructure layer. The virtual hardware is mapped to resources on the physical systems layer. As the final layer, the network services connect these systems relying on resources such as routers, switches or domain name services. Complementary to the previously described horizontal multi-tier SOA infrastructure, there is the vertical technical monitoring layer. It evaluates technical measuring points of the horizontal layers, such as network availability, CPU load, memory consumption or storage usage.

Analysed service cascades include redundant service offers and the support for dynamic coupling[2] between service consumer and provider should be considered. The technical monitoring solutions Amberpoint, Progress Actional, SOA Manager Service Manager, Oracle Enterprise Manager SOA Management Pack and OpTier CoreFirst do not offer sufficient information to gather quantifications of failure impacts and reliable conclusions on the feasibility of the implemented business processes in the given SOA infrastructures. Specifically analyses of the current feasibility of business processes in scenarios with redundant service offers fail due to the evaluation of secco[3].

4 SOLUTION DESIGN

To agree on feasibility and workload of business processes Business Service Level Agreements (BSLA) are proposed as abstraction layer for the contracting of service quality between service consumer and service provider. BSLAs are focused on the description of the estimated usage behaviour, extended by the declaration of the maximum allowed response time for service requests and can optionally be enriched by the declaration of maintenance windows, maximum unplanned downtimes, fines, pricing or other non-functional properties. BSLAs are aimed at replacing SLAs. In BSLAs the consumer's usage behaviour is described by *Usage Patterns* (Heckmann and Phippen, 2010), which offer an approach for the description of the quantitative consumer-provider-relation in terms of request frequency and processing complexity. BSLAs enable analyses on business process feasibility and workload by specifying the *contracted usage*. In opposite, the *monitored usage* reflects the current request amount and resource utilisation within IT infrastructures. The business process's workload is determined by comparing its contracted and monitored usage, assuming all infrastructure components are technically available. To enable monitoring of the request amount this paper proposes the use of a centralised request routing component, named Service Broker[4] (see Figure 1). The Service Broker provides a measuring point for request amounts per business process, which represent the process's workload, taking the contracted usage as reference. The business process's feasibility is lead back from its workload combined with information about the technical availability of all infrastructure components hosting the process.

The aggregation of technical monitoring information in service cascades hosting business processes is addressed by a *topology graph*. The term topology graph is introduced to reflect the functional dependencies between the components in an IT infrastructure. To build the topology graph infrastructure components can be retrieved from a configuration management database (CMDB) (Group et al., 2011). To represent redundant service offers within a topology graph *service lines* are introduced. A service line is a logical group of infrastructure components that are necessary to provide an application layer service. To aggregate resource utilisation of service line spanning resources the term *component category* is introduced. Component categories logically group infrastructure components that provide similar functionalities (e.g., application servers, which provide hosting of application layer services), see Figure 1 as example. To calculate the resource utilisation, each topology graph node is enriched with interpreted technical monitor-

[2]Intermediate logic that changes the invocation target of a service request at runtime.

[3]Based on an internal technical report in June 2009.

[4]The Service Broker acts as a economical load-balancer for cloud infrastructures (Heckmann, 2007).

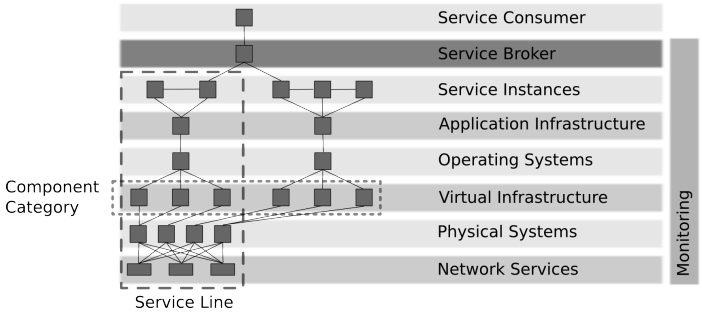


Figure 1: Multi-tier architecture including service broker, service line and component category as example.

ing information. The enriched graph is introduced as *availability graph*. Considered are two levels of monitoring data interpretation.

In case of *state-based* analyses the availability of a graph node is lead back by interpreting state-related technical monitoring data of the represented infrastructure component, such as ping states[5] retrieved from a technical monitoring system. Interpretation of this data limits results to two simple states: available and non-available. This variant is simpler to impose, but is less significant when deducting the feasibility of constitutive business processes. For *load-based* analyses the availability of a graph node is estimated comparing current resource utilisation with its maximum capacity. The utilisation is calculated based on load-related technical monitoring data like CPU load. This enables the provision of proportional metrics reflecting the current availability of the represented resource (e.g., 20 % utilisation of a DNS server).

The Service Broker estimates a business process as feasible if in the availability graph all state-based nodes of at least one service line are available and the resource utilisation of all load-based component categories offer sufficient reserves to process the *estimated usage*. The estimated usage for a given time frame is calculated by subtracting the monitored usage for a given business process from its contracted usage. In strictly state-based availability graphs only the process workload is taken into account when calculating the estimated usage, otherwise also the resource utilisation is incorporated.

5 PROOF-OF-CONCEPT IMPLEMENTATION

As proof-of-concept an application was implemented representing the state-based availability graph of an exemplary business process. For technical monitoring the implementation bears on Zabbix[6]. The application uses a given XML configuration file representing the topology graph. Its topology graph consists of nine nodes representing two service lines. Each node description is enriched with a reference to its Zabbix database identifier. The application extracts the state values for all nodes from the monitoring system. It autonomously determines the service lines and aggregates their availability states. As first outcomes, the ability to realise state-based availability graphs based on technical monitoring data is presented. The validation of the determination reliability is subject of future research.

[5]Ping enables the monitoring of the technical network interface of a remote system and is specified in the Internet Control Message Protocol (ICMP) (Postel, 1981).

6 CONCLUSIONS

This paper introduces a alternate abstraction layer in order to agree on the feasibility and workload of a business process instead of technical thresholds of the underlying technology as known from common SLA. This layer is called Business Service Level Agreements (BSLA) and establishes a black box around service capacity and technical implementation, thus loosening the coupling between technical service provision and business service consumption on the level of service agreements. Based on the identified qualified technical indicators, the paper evolves that BSLA approach. Corresponding, an approach for the technical monitor and enforcement of BSLAs during operations is presented. Concluding, a proof-of-concept implementation demonstrates the capabilities of these approaches.

REFERENCES

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D.,

[6]http://www.zabbix.com

Thatte, S., Trickovic, I., and Weerawarana, S. (2003). *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems.

Brebner, P. C. (2008). Performance modeling for service oriented architectures. In *Companion of the 30th international conference on Software engineering*, pages 953–954, Leipzig, Germany. ACM.

Chen, Y., Iyer, S., Milojicic, D., and Sahai, A. (2009). A systematic and practical approach to generating policies from service level objectives. In *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on*, pages 89 –96.

Dobson, G. and Sanchez-Macian, A. (2006). Towards unified QoS/SLA ontologies. In *IEEE Services Computing Workshops, 2006. SCW '06*, pages 169–174. IEEE.

Group, A., TSO, and Office, C. (2011). ITIL. http://www.itil-officialsite.com.

Haas, H. and Brown, A. (2004). Web services glossary. http://www.w3.org/TR/ws-gloss/.

Heckmann, B. (2007). Service provision in a utility computing environment. In *Proceedings of the Third Collaborative Research Symposium on Security, E-Learning, Internet and Networking*, pages 185–198, Plymouth, UK. Lulu.com.

Heckmann, B. and Phippen, A. (2010). Quantitative and qualitative description of the consumer to provider relation in the context of utility computing. In *Proceedings of the Eighth International Network Conference (INC 2010)*, pages 335–344, Heidelberg, Germany.

Hsu, C., Liao, Y., and Kuo, C. (2008). Disassembling SLAs for follow-up processes in an SOA system. In *2008 11th International Conference on Computer and Information Technology*, pages 37–42, Khulna, Bangladesh.

Moser, O., Rosenberg, F., and Dustdar, S. (2008). Non-intrusive monitoring and service adaptation for WS-BPEL. In *Proceeding of the 17th international conference on World Wide Web*, pages 815–824, Beijing, China. ACM.

Muthusamy, V. and Jacobsen, H. (2008). SLA-driven distributed application development. In *Proceedings of the 3rd workshop on Middleware for service oriented computing*, pages 31–36, Leuven, Belgium. ACM.

Muthusamy, V., Jacobsen, H., Chau, T., Chan, A., and Coulthard, P. (2009). SLA-driven business process management in SOA. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 86–100, Ontario, Canada. ACM.

Ou, T., Sun, W., Guo, C., and Li, J. (2008). Visualized monitoring of virtual business process for SOA. In *Proceedings of the 2008 IEEE International Conference on e-Business Engineering*, pages 767–770. IEEE Computer Society.

Postel, J. (1981). Internet control message protocol - RFC 792. http://tools.ietf.org/html/rfc792.

Raibulet, C. and Massarelli, M. (2008). Managing non-functional aspects in SOA through SLA. In *2008 19th International Conference on Database and Expert Systems Applications*, pages 701–705, Turin, Italy.

Stantchev, V. and Malek, M. (2010). Addressing dependability throughout the SOA life cycle. *IEEE Transactions on Services Computing*, 99(PrePrints).

Stantchev, V. and Schroepfer, C. (2008). Techniques for service level enforcement in web-services based systems. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 7–14, Linz, Austria. ACM.

Utlik, A. and Alexeyev, N. (2010). Comparative analysis of service level agreement monitoring methods. In *Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET), 2010 International Conference on*, pages 346 –346.

Van Moorsel, A. (2001). Metrics for the internet age: Quality of experience and quality of business. *5th Performability Workshop*.

A technology-abstracted approach to an Utility Computing simulation framework

B.Heckmann[1,2], A.Phippen[1] and G.Turetschek[1,2]
[1] Network Research Group, University of Plymouth, Plymouth, United Kingdom
[2] aiDa Institute of Applied Informatics, University of Applied Sciences Darmstadt, Darmstadt, Germany
email: benjamin.heckmann@gmx.de

Abstract

This paper identifies the demand for a lightweight technology-abstracted UC model for SOA service provision. It introduces the UC models of Bunker, Mendoza and Zhang and points out why they are not suitable for the indicated target. Afterwards the paper introduces the model from (Heckmann, 2007). It is shown that the model and its simulation will be useful for IT architects, operations managers as well as business managers. The introduced discrete event simulation approach for the UC model is based on OMNeT++.

Keywords

SaaS, Cloud Computing, Utility Computing, Service-oriented Architecture, Service Billing, Service Provision, Quality of Service, Simulation, Virtual Infrastructure

1. Introduction

1.1. Project overview

The afterwards described approach for an Utility Computing (UC) simulation framework is part of the effort to define a technology-abstracted Utility Computing model, that describes the minimum features a technical UC platform must provide. This model will allow technical framework evaluations or the simulation of provisioning resource demands and costs early during the development. In prior works, the core of such a technology-abstracted, UC-conform, lightweight service provisioning model was introduced in (Heckmann, 2007). This paper adds an approach for a simulation framework based on this model.

1.2. Utility Computing

In this paper Utility Computing is defined as business model for service providers remotely offering IT-enabled business services for Service-oriented Architectures (SOA) (Singh, 2005) (Melzer, 2007) and charging customers per usage, according to (Rappa, 2004). From the provider's IT perspective UC is about service provision that is able to scale dynamically, based on the real-time fluctuations in demand (Bunker *et al.*, 2006). Additionally UC service provision offers its services equipped with the ability to charge service consumption per use (Mendoza, 2007).

From a consumer's perspective UC is about "the reduction of IT-related operational costs and complexity" (Shin Yeo *et al.*, 2006). Both perspectives, provision and consumption, have in common to target a better utilisation of generally underutilised IT resources (IBM, 2003) on both sides. In summary, UC claims an abstract description how IT resource utilisation, its total costs and service prices relate. Based on this relation, UC offers the ability to dynamically adapt prices described in associated pricing models.

1.3. A technology-abstracted Utility Computing model

For Utility Computing a key success factor is how to effectively and efficiently deliver software systems as services (Zhang *et al.*, 2007). Concurrently, building a software architecture to deliver UC-based services is vastly different from designing architectures of traditional licensed applications (Mendoza, 2007). Additionally, Mendoza clearly states that from a perspective of independent software vendors (ISVs) of traditional enterprise applications, little has been done to study what needs to be changed in traditional enterprise applications to transform them into viable applications offered as services. But in future, ISVs will want to transform their business model to accommodate an UC model. Also Mendoza points out that, if an ISV develops an UC-based application or transforms a traditional licensed application to an application delivered as a service, "the application architect needs to be aware of the attributes that make an application suitable to be delivered as a service". In summary, from the perspective of an ISV there is a new way of providing software functionality, but also the question: What does a provider need to be prepared for going down this road? Bunker and Thomson have seen this demand also and proposed a corresponding Utility Computing Reference Model (Bunker *et al.*, 2006). This model aims to provide a practical framework for the introduction of utility computing. It is based on experience gained in real-world projects. The model of Bunker and Thomson provides an overall IT strategy sight to the provision of UC-based services. It provides not enough details to be helpful for IT architects to design a suitable UC architecture for a specific service provision scenario.

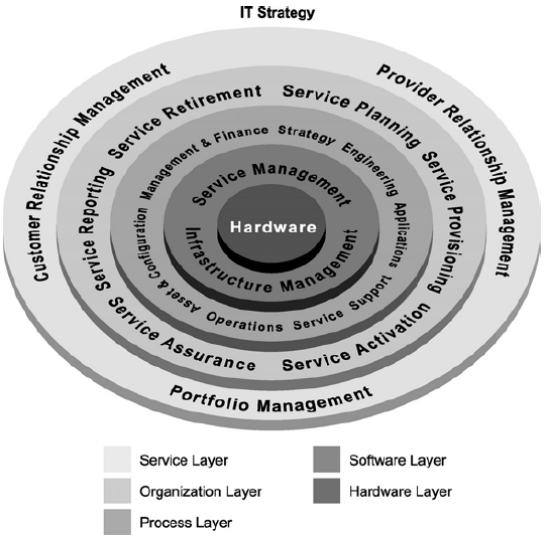


Figure 1: Utility Computing Reference Model by Bunker and Thomson

The UC model by Zhang, Zhang and Cai was developed from a business management perspective and is specified as "End-to-End Services Delivery Platform and Methodology" (Zhang *et al.*, 2007). With that, Zhang wants to provide an as complete as possible model for UC.

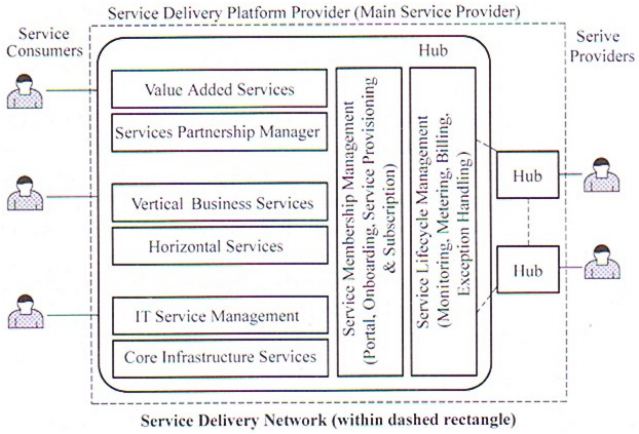


Figure 2: Layered view of a services delivery platform by Zhang

The End-to-End Services Delivery Platform description specifically aims for the provision of SOAP-based web services and describes in detail how SOAP web services should be provided. With its complexity the model addresses IT architects that have to handle large SOAP web service provision projects.

Vice versa comes the UC model of Mendoza (Mendoza, 2007), which was developed from a technological perspective. Mendoza starts his modelling with the definition of "software utility applications" and a detailed description of attributes that compose such UC services. Afterwards he describes a corresponding "software application services framework" that enumerates supporting services like metering or billing, which are essential for the implementation of UC services.

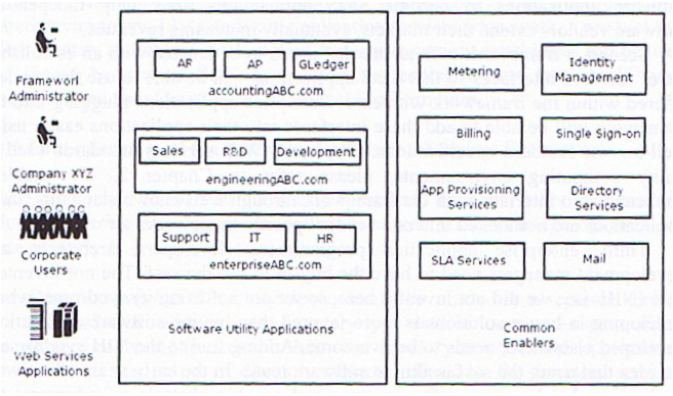


Figure 3: Conceptual Software Utility Framework by Mendoza

Like Zhang, Mendoza tries to deliver a complete model for UC. Although the model is confined to web service provision, it is the most interesting among the previously introduced UC models to be taken into account for IT architects. There are two things the introduced models have in common: They are rather complex and not clearly technology independent. And these properties are both not optimal for the currently fast advancing IT service technologies. But as

- a usable UC model is a key success factor (Zhang *et al.*, 2007),
- building a software architecture to deliver UC-based services is vastly different from designing architectures of traditional licensed applications (Mendoza, 2007),
- the application architect needs to be aware of the attributes that make an application suitable to be delivered as a service (Mendoza, 2007) (Bunker *et al.*, 2006),

there is a demand for a less complex UC model in small to medium projects, that describes what the core functions of a UC service architecture are and which is independent from the afterwards used implementation technology. Such a technology-abstracted Utility Computing core model was introduced in (Heckmann, 2007). In summary the model consists of eleven abstract elements, logically grouping demanded functionalities, and three basic workflows, which describe the minimum demanded interaction of those elements, listed below.

◊ **Service type**

Definition of a service class

◊ **Service instance**

Instance of a service type

◊ **Service host**

Host for service instances

◊ **Service consumer**

Sends service requests

◊ **Service request**

Invocation of a service instances including an associated service response (synchronous or asynchronous)

◊ **Service registry**

Authentication, directory and repository services

◊ **Service broker**

Authorization of service requests, cost-based routing

◊ **Service load-balancer**

Load-based routing, instance deployment

◊ **Service monitoring**

Resource consumption tracing

◊ **Policies**

Conditions for brokering (per service consumer), error and event handling (per service type) and security conditions (per service consumer)

The shown elements are abstract representatives of a group of functionalities. In a derived technical IT architecture these functional groups might be represented as standalone components, but could also be combined in other forms. Examples for IT architectures that show optional implementations of technical frameworks are shown below:

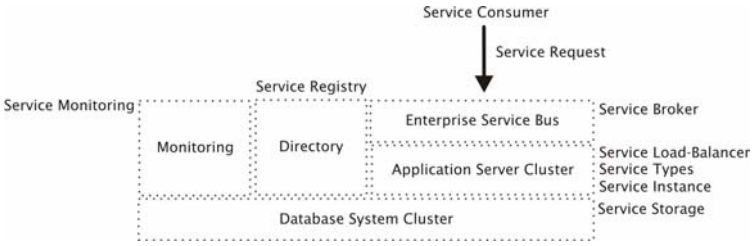


Figure 4: Implementation example based on classical SOA components

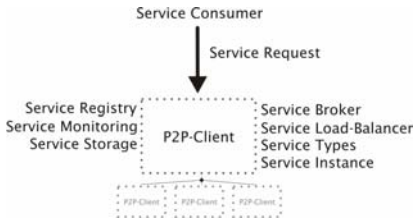


Figure 5: Implementation example implemented on a P2P architecture

1.4. Usage scenarios for a UC simulation

Utility Computing is basically a rather simple concept, where costs decrease while efficiency and effectiveness rise. But this simple message from theory misses its proof in practice, which makes it difficult to claim its use in real world projects. (Bunker, 2006) In prospection, the role of an IT architect will become more and more important to projects success. They will act as bridges between today's disconnected business analysts, application architects and infrastructure architects. For the IT architect's work, "new data, which includes usage patterns, will be added to the list of things to be considered. By looking at both functional and usage requirements, a design for an on-demand IT infrastructure can be implemented to eliminate problems such as underutilized resources and low return on investments." (Mendoza, 2007) In summary, there is a demand for an early proof of evidence in UC-driven development projects. But usage patterns are not only relevant to IT architects. There are other stakeholders who should take usage patterns into account, too: operations managers and executives. For all of those a UC model simulation could be a collective tool to work on usage patterns. A simulation could especially be useful for UC scenarios, where we expect service consumer groups which are strongly varying or complexly behaving in their usage patterns and in scenarios with highly meshed services. The association between the service's stakeholders and its usage patterns is shown in Figure 6. The UC model and its simulation can support this relationship at the following points:

(1) Development: IT architects

Based on the estimated usage, an IT architect can evaluate technical frameworks based on the technology-abstracted UC model. Additionally a UC simulation provides a way to analyse the characteristics of meshed services for service cascade optimisations. IT architects may also use the model as a base for design of individually developed technical frameworks.

(2) Operations: Operations managers

Based on the current usage, operations managers can use the simulation for resource prediction in the context of service operations (CPU, memory, storage, but also related factors like energy consumption for hardware or air conditioning systems).

(3) Executive: Business managers

Based on a subset of the usage data, business managers can use the simulation as a base for management decisions about IT investments (cost prediction) or price scales (per consumer group: pay per use, dynamic discounts, SLA fines, alternate provision locations, …).

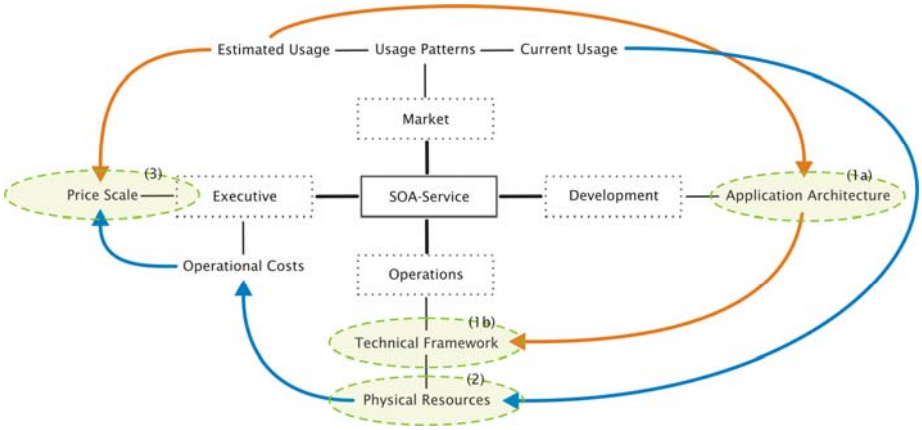


Figure 6: Relationship between service stakeholders & service usage patterns

2. Conceptual approach for the simulation

2.1. The simulation framework OMNeT++

OMNeT++ is an Open Source, modular and component-based simulation environment for discrete event simulations, like the simulation of communication networks. For this purpose it is widely used within the network simulation community. Because of its generic and flexible architecture, it has been successfully used in other areas, like the simulation of IT systems, queuing networks, hardware architectures and business processes as well. (omnetpp.org, 2008) It uses a flexible topology description language that combines modules that communicate via message passing. (Varga, 2001) (Varga, 1998) (Varga, 1997)

2.2. Technology-abstracted Utility Computing simulation framework

As a result of the mapping process from the introduced technology-abstracted UC model into a UC simulation, the already introduced elements of the Utility Computing model can basically be found within the simulation; see 1.2. for a short introduction of elements and Figure 7 for an overview how they relate in the simulation. The simulation currently provides the following functionalities:

- **Resource measurement and monitoring for CPU, memory and disk space**

  The hardware resources simulated and monitored are: CPU, memory and disk space. Not monitored, but taken into account, is the network traffic (bandwidth and delay). Additionally monitored are the load-balancer and broker queues and their message transport resource consumption. Also the overall resource consumption for the storage network is traced. Some management events on the hosts and corresponding instances are monitored as well.

◊ **Message billing to service consumers (Service Broker)**

To each request response a bill based on the processing sites CPU, memory and storage costs gets attached. The consumption of these resources during processing of the request is billed, and it is possible to add additional per site and per consumer margins.

◊ **Message routing by site costs (Service Broker)**

Messages get routed to a site with enough resources to process the request and the least costs for processing.

◊ **Message routing by resource demand (Service Load-Balancer)**

Messages are routed by a site's load-balancer to a host with enough resources, whereas hosts with already deployed instances are preferred.

◊ **Message queuing (Service Broker & Service Load-Balancer)**

Messages are temporarily stored within the service broker or service load-balancer when not enough resources for their processing are present. They are recalled from queue after a certain scheduling time and entered again in the scheduling sequence of either the service broker or the service load-balancer. In doing so the queuing consumes resources in the system, and if the system balancer runs out off resources, incoming messages are being dropped.

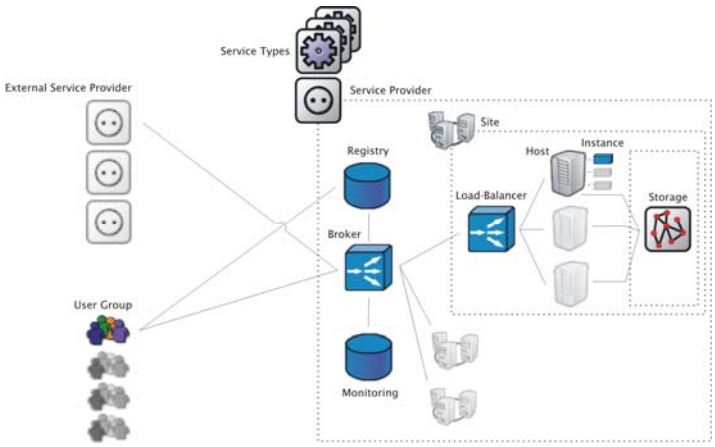


Figure 7: Utility Computing simulation overview

3. First outcomes of the simulation framework tests

First tests of the simulation have shown that it is possible to simulate all aspects of the formerly introduced UC model. The most complex scenario currently tested was based on 48 consumer groups, each sending a single request of the same service type

that includes two subrequests to independent external service providers. Additionally, one of the subrequests also invokes another subrequest to a service type provided by the original service provider for the initial request. This request cascade is illustrated in the Figure below.

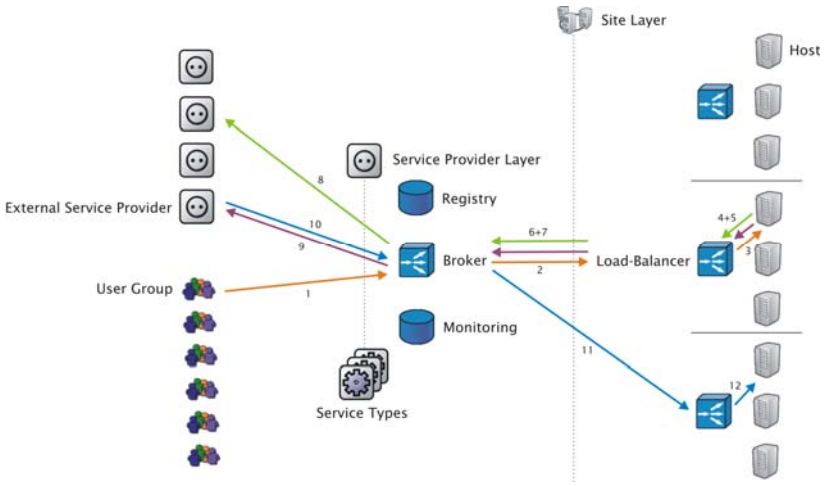


Figure 8: Service request example

1) Initial service request
2) Forwarding to the most cost-effective site with free resources
3) Forwarding to a free resource
4) Invocation of the first subrequest
5) Invocation of the second subrequest
6) Forwarding of the first subrequest to subrequest proxy
7) Forwarding of the second subrequest to subrequest proxy
8) Forwarding of the first subrequest to a suitable service provider
9) Forwarding of the second subrequest to a suitable service provider
10) Invocation of the third subrequest
11) Forwarding to the most cost-effective site with free resources
12) Forwarding to a free resource

As a result of each simulation run several values have been recorded. For all network elements CPU, memory and storage load are recorded. Additionally, for service brokers and load-balancers the queue length and outstanding messages and

subrequests are counted. As an example, Figure 9 shows the brokers CPU load, the outstanding messages forwarded by the broker and the brokers queue length.

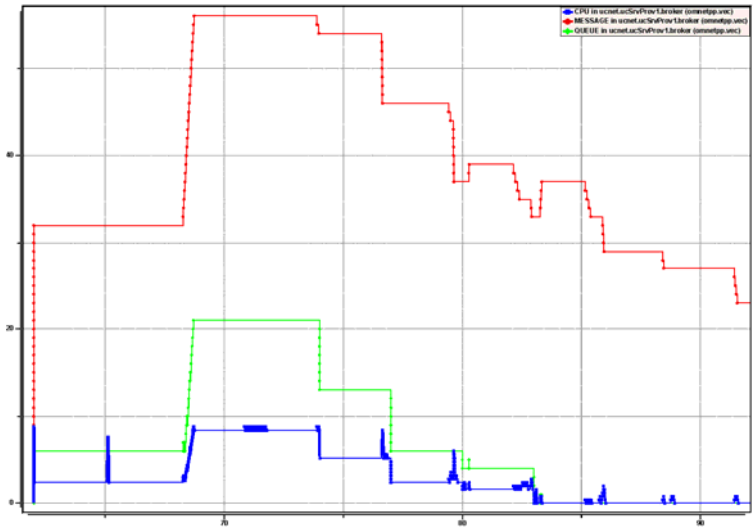


Figure 9: CPU load, outstanding messages and queue length of the service broker

For a whole site the CPU load can look as summarised in Figure 10. Here, the CPU loads of all elements of a certain site are shown over a period of time.

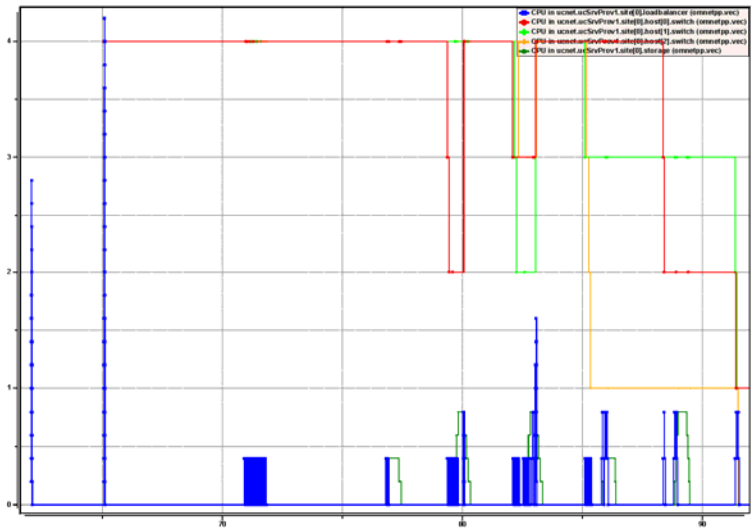


Figure 10: CPU load of a sites service load-balancer, hosts and storage network

These results only show the ability of the simulation to basically prove that the introduced UC model can be simulated. Future work will be focused on proving that the outcomes of a simulation run adequately represent a service's behaviour. One major aspect will be the calibration of the simulation to represent a real world scenario. Driven by these first results it is assumed that a technology-abstracted simulation could not only be used to predict the behaviour of SOAP web services, but also for RESTful services or simpler services, like web server clusters. Even virtual infrastructures, e.g. VMware or Xen, could be simulated.

4. Summary

This paper defines the term Utility Computing as on-demand service provision business model for Service-oriented Architectures. Based on this definition, it evolves the demand for a lightweight technology-abstracted UC model for future service provisioning projects. It shortly introduces the UC models of Bunker, Mendoza and Zhang. All three models do not meet the depicted standard for a lightweight technology-abstracted UC model, but, on the other hand, clearly show the demand for such a model. Subsequently, the paper elaborates, based on the works of Bunker and Mendoza, that service's usage patterns are one conjunctive common ground in future UC projects. Based on this finding it is shown that not only a lightweight technology-abstracted UC model is useful within this relationship. Also the simulation of such a model enables IT architects to optimize service cascades, operations managers to predict resource loads and business managers to obtain resilient information for IT investments and for the planning of price models. One conceptual approach for a UC simulation is introduced in the second chapter of this paper. As simulation framework OMNeT++ is used. As UC model, the model described in (Heckmann, 2007) is implemented within the simulation framework. The current state of implementation shows that all aspects of the model can be transferred into the simulation. Yet implemented features are: resource measurement and monitoring for CPU, memory and disk space, message billing to service consumers, message routing by site costs, message routing by resource demand and message queuing.

References

Bunker, G., Thompson, D., 2006. Delivering Utility Computing. Business-driven IT Optimization: Business-Driven IT Optimization.

Heckmann, B., 2007. Service provision in a utility computing environment. SEIN 2007, University of Plymouth, 14-15 June 2007.

IBM, 2003. A Taxonomy of the Actual Utilization of Real UNIX and Windows Servers. IBM white papers.

Melzer, I., Dostal, W., Jeckle, M., 2007. Service-orientierte Architekturen mit Web Services. Elsevier, Spektrum Akademischer Verlag.

Mendoza, A., 2007. Utility Computing Technologies, Standards, and Strategies. Artech House Inc.

omnetpp.org, 2008. OMNeT++ Community Site. http://www.omnetpp.org.

Rappa, M. A. 2004. The utility business model and the future of computing services. IBM Syst. J. 43, 1 (Jan. 2004), 32-42.

Singh, M., Huhns, M., 2005. Service Oriented Computing Semantics. John Wiley & Sons.

Yeo, C.S., Assunção, M.D., Yu, J., Sulistio, A., Venugopal, S., Placek, M., and Buyya, R., Utility Computing on Global Grids, Hossein Bidgoli (ed), The Handbook of Computer Networks, John Wiley & Sons, New York, USA, accepted in April 2006 and in print.

Varga, A., 1997. Flexible topology description language for simulation programs. SIMULATION IN INDUSTRY: 9TH EUROPEAN SIMULATION SYMPOSIUM 1997 (1997):225-229.

Varga, A., 1998. Parametrized topologies for simulation programs. PROCEEDINGS OF THE COMMUNICATION NETWORKS AND DISTRIBUTED SYSTEMS MODELING AND SIMULATION (CNDS'98) (1998):15-20.

Varga, A. 2001. The OMNeT++ Discrete Event Simulation System. In the Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic.

Zhang, L.-J., Zhang, J., Cai, H., 2007. Services Computing, Core Enabling Technology of the Modern Services Industry, published by Springer and Tsinghua University Press.

Economic Efficiency Control on Data Centre Resources in Heterogeneous Cost Scenarios

Benjamin Heckmann, Marcus Zinn,
Ronald C. Moore, and Christoph Wentzel
University of Applied Sciences Darmstadt
Haardtring 100, 64295, Darmstadt, Germany
benjamin.heckmann@gmx.de

Andrew D. Phippen
University of Plymouth
Drake Circus, PL4 8AA, Plymouth, U.K.

***Abstract*—Optimisation of resource selection in hybrid cloud data centres depends on the control of resource usage. The primary criterion for this resource selection is economic efficiency. The presented approach considers operational efficiency aspects in service providing and therefore focuses on technical criteria, such as resource load, as well as economic criteria, such as the costs of resource usage. When services are offered at different service levels the approach enables revenue optimisation in cases of excessive load. The concept is prepared to handle heterogeneous IaaS scenarios.**

***Keywords*—Business, Cloud, Efficiency, Services-oriented Architecture, Utility Computing**

I. Introduction

The following concept characterises an approach to optimise the resource selection in data centres. Both runtime and deployment time are considered as point of decision about the usage of resources. Primary criterion for this decision is economic efficiency.

The project was conducted as an applied research in the field of business informatics in close cooperation with a business partner [1]. The developed approach for efficient control on data centre resources in heterogeneous cost scenarios was also implemented as a proof-of-concept [2]. The concept is restricted to the following technical solutions for IT resource offers specified by our business partner.

IaaS: Offering hardware resources located in data centres (e.g., servers, storage, network) based on virtualisation technologies (e.g., VMware, Xen) is defined as Infrastructure as a Service (IaaS) in this concept. Virtualisation enables the separation of hardware resources into smaller fractions, whereby each fraction offers the same virtual hardware interfaces as an actual hardware. In this context the IaaS focus is on server virtualisation. These server fractions are called virtual machines (VM). Hardware resources can be allocated to VMs as demanded, depending on the features of the virtualisation technology used. IaaS thereby describes the basic management layer for data centre operations.

SaaS: Software applications can be deployed based on an IaaS layer. In this context deploying business software in one or several VMs to ease deployment and operation of multiple parallel instances of this software is called Software as a Service (SaaS). Thereby, SaaS describes the basic layer for the consumer interaction.

Hybrid Cloud: In this paper the provisioning of resources or IT services based on the paradigm of IaaS or SaaS is also called cloud-based provision, conforming to the *cloud* definition of the National Institute of Standards and Technology (NIST) [3]. In this paper hybrid clouds are compositions of clouds offering the same type of service while their operation technology may vary. The services analysed in this project are operated as a hybrid cloud hosted in several data centres across the world. A data centre may expose its resources as a single cloud, but more often as the sum of multiple clouds, each representing an individual technical solution grown over time.

II. Background

A. Cost Domains

Here it is assumed that in most cases the technical boundary of a cloud also reflects an individual cost domain. This is true when clouds reside in different data centres, even more obvious in different countries. Clouds can also differ in the applied technology for their operations. Distinguishable cost domains can also originate out of significantly different hardware performance, as in scenarios where older and newer hardware are operated simultaneously within the same data centre.

B. Utility Computing Service Life Cycle

Our concept takes major aspects of Heckmann et al. and extends them significantly. The works of Heckmann et al. reflect the characteristics of a service life cycle (business planning, development and operations) in the context of Utility Computing (UC). The business model of UC offers scalable IT-based services metered by usage.

The main contributions aggregated from these results are:

- Technology-independent Provision Model [4]
  The developed component architecture describes the minimum necessary functionalities and dependencies in an operations environment for a UC service. This architecture is used when services should be operated as part of a service-oriented architecture (SOA) and hosted on a cloud platform and are incorporated in an UC business plan. Those scenarios (SOA and cloud and UC) are called UC scenarios herein.

- Technology-abstracted Resource and Cost Simulation [5]
  When services are orchestrated [6] using other services and used in UC scenarios, complex service cascades are formed. These cascades can be complex both architecturally and economically. Both challenges can be addressed with a simulation framework to analyse the interaction between resource allocation and costs, service orchestration, service purchasing costs, and service pricing. A proof-of-concept implementation of such a simulation framework for multi-tier operations environments was implemented.
- Specification Paradigm for Service Quality [7]
  Within the introduced results a new approach on agreeing on service level for services in UC scenarios is described. This approach offers a specification paradigm for service quality description straight from a usage perspective. In this case service levels are no more defined by technical conditions. They are called *business service level* (BSL) and are specified by describing the quantity and quality of the consumers' behaviour in using a service.

C. Research Objectives

The research objectives are examined from the perspective of a service provider.

We assume a service provider with multiple data centres spread worldwide. The data centres are operated as a hybrid cloud with multiple clouds per data centre hosting SaaS offers for a multiplicity of varying customers. Each service is offered with more than one service level. Each cloud is considered to be its own cost domain. The research objective is to make potential savings accessible between different cost domains. We provide an approach for a technical solution, including a proof-of-concept implementation. This optimisation should be performed at the initial resource allocation during deployment of a service as well as continuously during its operations.

III. Related Work

This research offers an approach to technically converge the quality-related ontologies of service, experience, and business as introduced by Moorsel [8] or Dobson and Sanchez-Macian [9]. In the literature, three focuses on data centre control related approaches can be found: effective data distribution, quality of service (QoS) in networks [10] and reduction of power consumption. The focus on effective data distribution resides in the field of grid computing. Here large amounts of data have to be distributed over several nodes so that parallel calculations on the data slices accelerate the overall processing of the data. In most grid architectures there is an architectural component called *broker* [11]. This broker controls the distribution, processing and result aggregation, sometimes supplemented by billing or marketplace features, like auctions and bidding. Different approaches are known to accelerate processing, for example using resource reservation or considering the problem as a queueing system [12].

In networks, QoS approaches mainly are focused on the network layer. MDCSim [13] instead offers an approach for a multi-tier data centre simulation, but focuses their outcomes onto a comparison of Infiniband and 10 Gigabit Ethernet network technologies.

The focus on reduction of power consumption centers on server consolidation. Approaches for load prediction for servers in a single data centre are shown by Speitkamp [14] using historical data analysis, Bi [15] using a non-linear optimisation model or based on a limited lookahead control framework by Kusic [16]. Wang introduces an approach to combine server consolidation and dynamic voltage and frequency scaling [17]. An approach for service level management in distributed infrastructures, including QoS translation and support for self-adaptation, is shown by Freitas [18].

Load balancing on the level of data centres within and between client devices is addressed by Peoples [19].

None of these approaches sufficiently covers the relation between resources, services and consumers introduced in Section 5 of this paper.

IV. Research Approach

The following steps were taken to obtain the research objectives of making potential savings accessible between different cost domains for SaaS providers:

1) Analysis of the customer-service-resource relation in SaaS provision scenarios in the context of our business partner (see section V).
2) Design of a generalised concept to efficiently control data centre resources in heterogeneous cost scenarios based on the previous analysis (see section VI).
3) Implementation of the design as a proof-of-concept (see section VII).

V. Analysis of the Business Partner Context

A. Model of the Customer-Service-Resource Relations

The relationships between a SaaS provider and its customers are modelled with a data structure. This data structure is subsequently used as the basis for the optimisation, and must be modified only when the relationships between the provider and the customers change. The provider and the customers are represented by the nodes in a graph; the edges in the graph represent the services provided.

Customers can have one or more contracts with the provider. A contract applies to one or more consumer groups (e.g., branches) within the organisation of the customer. Each consumer group relates to one or more services of the provider. This relation incorporates the link to two service levels and one usage pattern. A Usage Pattern is a quantitative and qualitative description of the service usage behaviour of a consumer group [7].

The price (per unit) and the contract penalty (per unit) are stored attached to the link between the first service level, a service and a consumer group. Accordingly, the price (per unit) is also stored attached to the link affecting the second service level. A penalty for this relation is not necessary, as it reflects the service usage over and above the contracted usage pattern.

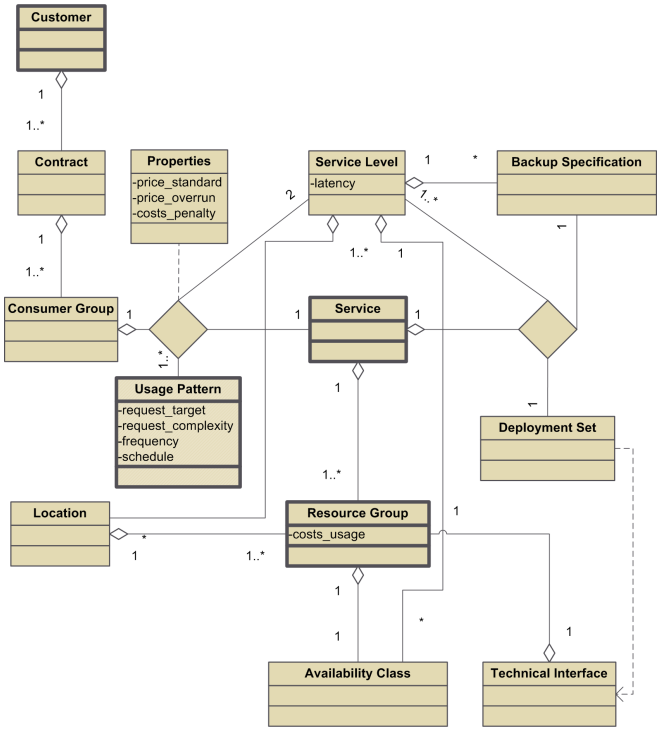


Figure 1. Relations Between Resources, Services and Consumers

Services, one or several, act as connector between provider and customer, more precisely between resource groups on the provider side and consumer groups on the customer side. Also a service relates to a backup specification and a technical deployment set. Such a set contains necessary files and configuration properties for deployment.

Resources are managed in groups. The primary grouping criterion is technical, for example the virtualisation software used. The secondary criterion is the geographical location, for example the hosting data centre. The cost of the resource usage (per unit) is an attribute of a resource. Linked to a resource is the according technical interface for its administration and monitoring (e.g., virtualisation management API). Additionally, an availability class is linked to a resource group. The availability classification enables an abstract categorisation to distinguish between different level of technical availability assurance.

Service level serve as abstract categorisation to differentiate between varying level of service quality. Beside their previously described relations, a service level links to one or more locations, one availability class and one backup specification.

The customer-service-resource relation is elaborated in the data model in Fig. 1.

B. Mediation Conditions

In the research context resource groups are only considered during resource selection when they conform to the required quality properties. Resource selection should respect the technical load of resource groups and customer constraints such as processing location. Only incoming service requests (e.g., from the consumer towards the service) should be considered.

VI. Solution Design

The required functionality for an efficient control on data centre resources in the analysed context is distributed among two architectural components, named *Service Broker Manager* and *Service Broker Gateway*.

The Service Broker Manager implements the elaborated data model described above and offers interfaces for interaction (e.g., graphical user interface (GUI), application programming interface (API)). Beside the storage of the data model the broker offers a method to match a service request from a certain customer with a suitable resource. The broker continuously analyses the monitoring data from all resource groups and redirects service requests, including service relocation, accordingly.

The matching between a customer's service request and a suitable resource is done in six steps. Preconditions are a given service request and at least two resource groups:

1) Service type, service consumer and the service level corresponding to the service request are determined.
   Postcondition 1: identifiers for service type, service consumer, and service level are known.
   Precondition 2: service request and service type are known.
2) Resource demand for the service request is estimated.
   Postcondition 2: service request's resource demand is known.
   Precondition 3: service request's resource demand, service type, and service level are known.
3) Pools of resource groups are selected by available resources and matching service level.
   Postcondition 3: two pools of resource groups are known, where each resource group offers enough resources for request processing and one pool complies with the demanded service level and the other does not.
   Precondition 4: service request's resource demand, service type, and service consumer are known.
4) The estimated revenue per pooled resource group for request processing is calculated.
   Postcondition 4: per given resource group the estimated revenue is known.
   Precondition 5: service request's resource demand, service type, service consumer, and service level are known.
5) Estimated costs for service level violation (latency exception and request failure) are calculated.
   Postcondition 5: estimated costs for latency exception and request failure are known.
   Precondition 6: two pools of resource groups with sufficient processing resources distinguished by service level compliance, estimated revenue per pooled resource group and estimated costs for latency exception and request failure are known.
6) The most efficient opportunity out of the following actions is selected:

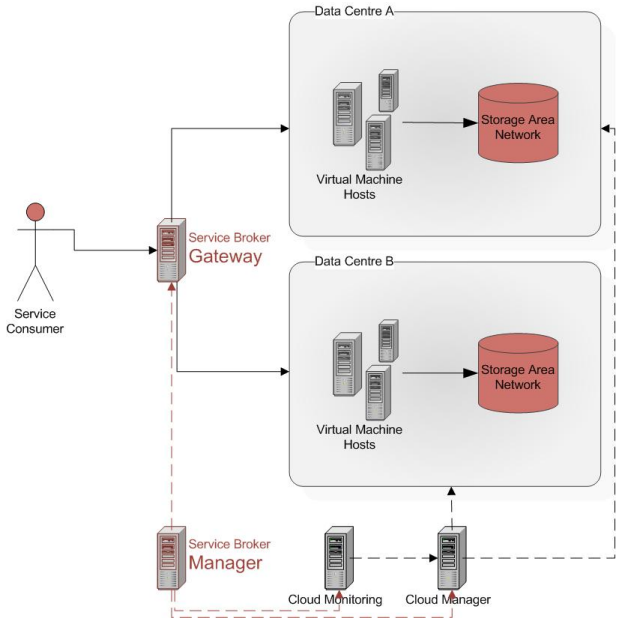


Figure 2. Service Broker Component Architecture

- Request is processed by a service level conforming resource group.
- Request is processed by a non-conforming resource group.
- Request is not processed.

Postcondition 6: action for further request processing determined.

The Service Broker Gateway acts as a load balancer on the network layer. It reroutes service requests to appropriate service instances, including the capability of dynamically shaping the traffic up to the blocking of certain requests. This is especially useful in cases of excessive load. Here requests can be forwarded (or blocked) based on economic efficiency.

This *Service Broker* concept enables resource selection and control on load distribution based on the elaborated relation. An overview on the component architecture is given in Fig. 2.

VII. Initial Results

As proof-of-concept the Service Broker Manager including a GUI, API and the request-resource matching method has been implemented. As a scenario for the evaluation of the request-resource matching method a database with four customers, each with two contracts affecting two consumer groups is defined. Five services are available, whereby each consumer group uses two services. As hosting environment two resource groups are provided, hosted in two data centres as varying cost domains. The self-service cloud portal of the business partner uses the Service Broker API to retrieve a suitable resource address during service deployment.

First tests using the self-service portal show the broker's ability to pick the most cost effective resource with enough load reserve. This leads to a significant overall change in load (and service instance) distribution among the two cost domains. The load distribution shifts in favour of the more cost-effective data centre. Without the broker-enriched self-service portal, the deployment of new service instances took about three weeks for the whole business process to terminate, due to internal measurements of the business partner. Using the broker-enriched portal the deployment time was *reduced to approximately 30 minutes*.

These first outcomes demonstrate the proof-of-concept's ability to efficiently control data centre resources in heterogeneous cost scenarios.

VIII. Further Work

Feasibility of Business Processes: Our concept creates an opportunity to also associate business process steps with our data model. A similar approach was introduced by Heckmann, but not elaborated to work based on resource load information.

Simulation-based Load Prediction: The Service Broker can be extended based on the simulation framework for Utility Computing elaborated by Heckmann [5]. Instead of retrieving the current load through a service for resource monitoring (referring to step three in Section 6) the broker can use load forecasts.

Utilisation of External Services: From a provider's perspective, at the current stage, the concept only addresses incoming service requests. In addition, the concept could also be extended to represent outgoing service requests to external service providers. This could expand the efficiency of the service provision one step further.

IX. Conclusions

This paper introduces and evaluates the *Service Broker* concept.

The Service Broker is an approach to optimise the resource selection in data centres. The concept enables the control of resource usage both at runtime and deployment time. In this research context, the primary criterion for resource selection and subsequent request forwarding is economic efficiency. The broker was evolved and evaluated in close cooperation with a business partner. The evaluation of the concept was done through a proof-of-concept implementation presented on CeBIT 2011 as an applied research in the field of business informatics.

The elaborated concept considers technical criteria, such as resource load, as well as economic criteria, such as the costs of resource usage. When services are offered at different service levels the broker enables revenue optimisation in cases of excessive load. Additionally, the concept is independent of the technical solution for resource management (e.g., virtualisation framework) and is prepared to also handle heterogeneous technical scenarios.

References

[1] P. Opper, "T-Systems International GmbH," 2011.
[2] M. Zinn, "Cebit 2011," 2011.
[3] P. Mell and T. Grance, "The NIST definition of cloud computing," Jul. 2010, (Access Date: 04/05/2011). [Online]. Available: http://csrc.nist.gov

[4] B. Heckmann, A. D. Phippen, R. C. Moore, and C. Wentzel, "Agreeing on and controlling business service levels in Service-Oriented architectures," *International Transactions on Systems Science and Applications*, vol. Vol. 7, no. No. 3/4, pp. 173–178, Dec. 2011. [Online]. Available: http://siwn.org.uk/press/sai/itssa0007.htm

[5] B. Heckmann, I. Stengel, A. Phippen, and G. Turetschek, "Utility computing simulation," in *ESM'2009 The 2009 European Simulation and Modelling Conference*. Leicester, United Kingdom: EUROSIS-ETI, Oct. 2009, pp. 175–180. [Online]. Available: http://www.eurosis.org

[6] T. Erl, *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, 2007.

[7] B. Heckmann and A. Phippen, "Quantitative and qualitative description of the consumer to provider relation in the context of utility computing," in *Proceedings of the Eighth International Network Conference (INC 2010)*, Heidelberg, Germany, Jul. 2010, pp. 335–344.

[8] A. Van Moorsel, "Metrics for the internet age: Quality of experience and quality of business," *5TH PERFORMABILITY WORKSHOP*, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.3810

[9] G. Dobson and A. Sanchez-Macian, "Towards unified QoS/SLA ontologies," in *IEEE Services Computing Workshops, 2006. SCW '06*. IEEE, Sep. 2006, pp. 169–174.

[10] R. Braden, D. Clark, and S. Shenker, "RFC 1633 - integrated services in the internet architecture: an overview," http://www.apps.ietf.org/rfc/rfc1633.html, Jun. 1994. [Online]. Available: http://www.apps.ietf.org/rfc/rfc1633.html

[11] S. Venugopal, R. Buyya, and L. Winton, "A grid service broker for scheduling distributed data-oriented applications on global grids," in *Proceedings of the 2nd workshop on Middleware for grid computing*, ser. MGC '04, 2004, pp. 75–80, ACM ID: 1028506.

[12] A. Afzal, A. S. McGough, and J. Darlington, "Capacity planning and scheduling in grid computing environments," *Future Generation Computer Systems*, vol. 24, p. 404414, May 2008, ACM ID: 1350010.

[13] S. Lim, B. Sharma, G. Nam, E. K. Kim, and C. Das, "MDCSim: a multi-tier data center simulation, platform," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009, pp. 1–9.

[14] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *Services Computing, IEEE Transactions on*, vol. 3, no. 4, pp. 266–278, 2010.

[15] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 370–377.

[16] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *Autonomic Computing, 2008. ICAC '08. International Conference on*, 2008, pp. 3–12.

[17] Y. Wang and X. Wang, "Power optimization with performance assurance for multi-tier applications in virtualized data centers," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, 2010, pp. 512–519.

[18] A. L. Freitas, N. Parlavantzas, and J. Pazat, "A QoS assurance framework for distributed infrastructures," in *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, ser. MONA '10, 2010, p. 18, ACM ID: 1929567.

[19] C. Peoples, G. Parr, and S. McClean, "Energy-aware data centre management," in *Communications (NCC), 2011 National Conference on*, 2011, pp. 1–5.

Quantitative and Qualitative Description of the Consumer to Provider Relation in the Context of Utility Computing

Benjamin Heckmann[1], Andrew D. Phippen[2]
In memoriam Günter Turetschek

[1]h_da – University of Applied Sciences Darmstadt, Germany
[2]Centre for Security, Communications and Network Research
University of Plymouth, United Kingdom
benjamin.heckmann@gmx.de

Abstract: Utility Computing service provision aims to control the service quality for a wide range of consumers. To closely control the desired service quality in each phase of the service operations lifecycle, it is essential to be able to describe the quantitative and qualitative relation between consumer and provider. This work introduces Usage Patterns as a description language for the planning of quantitative relations and Provisioning Factors to control the qualitative relations during runtime.

1 Utility Computing

This work is focused on the modelling and simulation of service usage in the context of Utility Computing (UC). The term *utility* thereby refers to the field of industry. Here a public utility [Bri10] describes an enterprise that provides certain classes of services to a wide range of consumers.

The name Utility Computing indicates the vision of IT-based services comparable to public utilities. In this work Utility Computing is defined as a business model for service providers offering IT-based services and charging service consumers per usage, according to [Rap04]. From the provider's IT perspective UC is about service provision that is able to scale dynamically, according to real-time fluctuations in demand [BT06]. Additionally, UC service provision offers its services equipped with the ability to charge service consumption per use [Nee02].

From a consumer's perspective UC is related to "the reduction of IT-related operational costs and complexity" [YdAY+06]. Both perspectives, provision and consumption, have in common to target a better utilisation of generally underutilised IT resources [AAR02] on both sides. In summary, UC implicitly claims an abstract description of how IT resource utilisation, its total costs and service prices relate (see Figure 1).

Thereby Utility Computing does not refer to a specific IT service definition. From a business perspective any IT service where, from an economical point of view, it makes sense to charge it by its usage is addressed by UC, e.g. flight scheduling, webspace offers or

others. Therefore a more abstract service definition is the most suitable for UC: A service represents a type of relationship-based interaction between a service provider and a service consumer to achieve a certain solution objective [Zha07]. From a technical perspective there are several types of services that fit this definition, e.g. web services (SOAP, RESTful and others), HTTP web servers or virtual infrastructures (Xen, KVM and others). Service types outside the UC scope are e.g. IT projects or hardware sales.

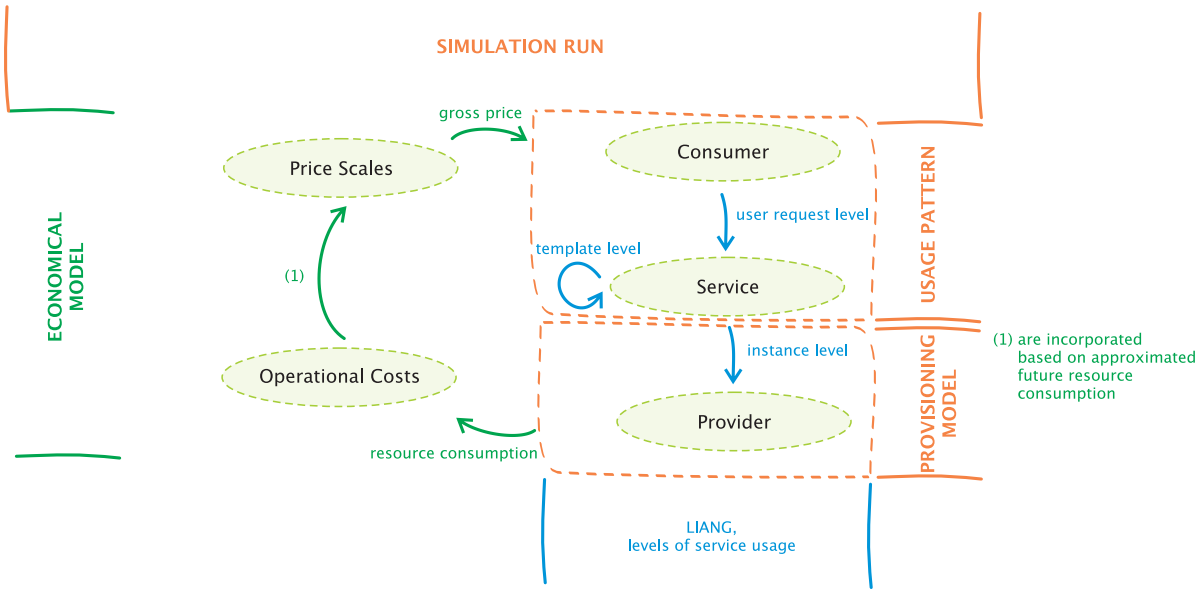


Figure 1: Levels of service usage in the context of this work

2 Service Quality

Utility Computing strongly addresses the aspect of service quality. This is implied by the vision of UC as a business model for IT service providers comparable to public utilities in case of necessity, reliability, usability, utilisation, scalability and exclusivity [Rap04]. All these attributes directly or indirectly address the quality of the service provision.

The definition about what quality criteria for a service offer are chosen and how they are monitored is subject to a service level agreement (SLA)[1] between the service provider and a service consumer. This work addresses SLA on the level of technical agreements of service quality, not the functional level. Thereby *technical* means: all technically measurable requirements relevant for the attended service response properties beside functional correctness. Functional correctness describes the accurate behaviour of a service on the layer of business logic. For example, a service method invocation with accurate method parameters, according to the specified parameter value ranges, must compute the accurate result set consisting of the expected business data.

[1] referring to the context of ITIL

In this work, response time is defined as the primary SLA criterion for the service consumer's perspective in the context of UC. Other possible primary SLA criteria, like continuity or security of service offers, are neglected. From the UC consumer perspective, request response times must be independent of the overall provider load. This implies the abdication of contracted resource reservations of any kind. Otherwise the targeted dynamic scale and optimised resource utilisation cannot be addressed by the UC provider. Response time is the outcome of the secondary SLA criteria on the service provider side, defined as request processing complexity and overall request amount. These criteria are subject to the individual SLA between provider and consumer. This definition differs from known definitions of SLA, which use resource centric criteria and therefore do not recommend for UC scenarios.

In addition to service quality criteria, according classes of observed criteria value ranges and corresponding actions must be specified during contracting. For the criterion of response time, aberrations could be classified as:

- Better – for service response times beyond the minimum specified acceptable period of response time.
- Within acceptable range – for service response times between the minimum and the maximum of specified acceptable periods of response time.
- Beyond unacceptable limit – for service response times beyond the maximum specified acceptable period of response time.
- Unprocessed requests – for technically and functionally accurate service requests that never got processed by the service provider, e.g. for dropped requests due to resource overload on the provider side.

Beside the definition of service quality criteria, the management of service quality is of interest to the service provider. A classical approach to manage the quality of service provision is the capacity management. Here the goal is to provide the necessary amount of resources for a certain service quality level at any time. This paper additionally proposes to manage the quality of service provision by managing the service usage. In this work *usage management* contains the indirect *consumption management* affecting the consumer side and the direct *provision management* on the provider side. Consumption management uses the provider's abilities such as SLA or price scales to indirectly influence the service consumer's usage behaviour. Therefore it addresses the quantitative aspect of the consumer-provider relation. For the qualitative aspect of the relation, provision management monitors, evaluates and directly controls service request routing and processing resource utilisation.

3 Research Objectives

The overall context of this work focuses on specific aspects of the service operations lifecycle (SOL) [HSPT09][2] for service offers based on the business model of Utility Computing. In the phase of service business planning this work refers to the corresponding service properties and service usage profiles resulting from the previous UC definition. During service development and the phase of service operations this work will focus on services in the technical context of Service-oriented Computing (SOC) [Pap03] corresponding to the paradigm of Cloud Computing as described by [BMQ+07].

In this context a description of the modifications necessary to transfer a standard service operations lifecycle into a UC SOL is missing. This includes the demand for an explicit definition of UC's core relation between IT resource utilisation, its total costs and service prices. Also specific attention must be given to the implications of complex UC usage scenarios.

The unidentified implications of complex UC usage scenarios considerably compromise the planning, development and operation of UC service offers. Under these conditions the prediction of resource utilisation and dependent operational costs, calculation of subsequent price scales, and subsequent runtime gross price calculations will fail.

4 Research Approach

The overall work starts from the business perspective, as technical requirements depend on the business requirements imposed. Therefore, a five step approach to find solutions for the specified objectives is proposed:

1. Describe the current state of service usage in the context of Utility Computing.
2. Elaborate a detailed definition for the relation between a service and its consumer.
3. Analyse the SOL of UC services.
4. Determine the implications of complex UC usage scenarios regarding SOL.
5. Deduct a corresponding strategy to handle the complexity.

This paper focuses on the elaboration of the quantitative and qualitative relation between service consumers and providers in the context of UC service provision quality management.

[2] SOL phases are defined as: business planning, development and operations

5 Usage Patterns

To be able to describe the core relation of Utility Computing, the following is assumed: UC implies a relation between IT resource utilisation, its total costs and service prices. This relation basically can be described by the quantitative usage relation between consumers and a service, enriched by metadata. All other variables are deducted from this usage relation, like cost and price calculations. This work proposes *Usage Patterns* as a description language for this quantitative usage relation.

For the IT architect's work, "new data, which includes usage patterns, will be added to the list of things to be considered" [Men07]. But the term Usage Pattern is not clearly defined in computer science. Some similar terms are used to describe traffic in computer networks or load in enterprise data centres. But none of these terms is applicable in the Utility Computing service provision context of this work.

As an entry point to service usage description the works of [LyCMO06] are introduced. Liang defines three perspectives of service usage and collects data on these levels as entry points for his usage data mining on web services. These levels of service usage are:

- User request level – The user request level of service usage addresses the outer view on composite services. This perspective focuses on how composite services are used by the consumer. This level is not aware of the optional complexity of service cascades or the diversity of providers within the cascade.
- Template level – The template level of service usage addresses the inner view on service correlations. A service template is defined as a flow of services, the final output of which can satisfy the consumer's need. At this level service usage concentrates on how services correlate.
- Instance level – The instance level of service usage addresses the constraints of the service runtime environment of the service provider. These constraints restrict how services are implemented and whether and how they can function.

Usage Patterns are concerned with the user request level and template level of Liang's definition of service usage, see Figure 1. Both of these perspectives describe relations between services and their consumers.

A Usage Pattern defines the quantitative usage relation between an unlimited number of service consumers and a particular service offered by a service provider. Thereby consumers are grouped according to their usage behaviour. This behaviour is expressed by one or more request classes, whereby the relation provides the request frequency as attribute with equal distribution assumed. Each request class describes a certain usage behaviour towards the function of a service.

This behaviour is described by an abstract function parameters class. Each class represents a characteristic combination of function parameter value ranges that imply a certain function call behaviour. It is assumed that for most functions the resource demand for processing a function call can be deducted from given parameter values. It is known that there are functions where this assumption fails, e.g. a function to calculate the total amount of a

bank account given the account number. It is not possible to estimate the resource demand of this calculation by evaluating the account number.

Also a request class may relate to any number of sub-request classes. For this recursive relation a request frequency attribute is provided, with equal distribution assumed. It is known that this ability to describe service cascades breaks the paradigm of service abstraction [Erl07]. Therefore this feature is optional.

The detailed relations which altogether instantiate a Usage Pattern are shown in Figure 2 using an entity relationship diagram [Che76].

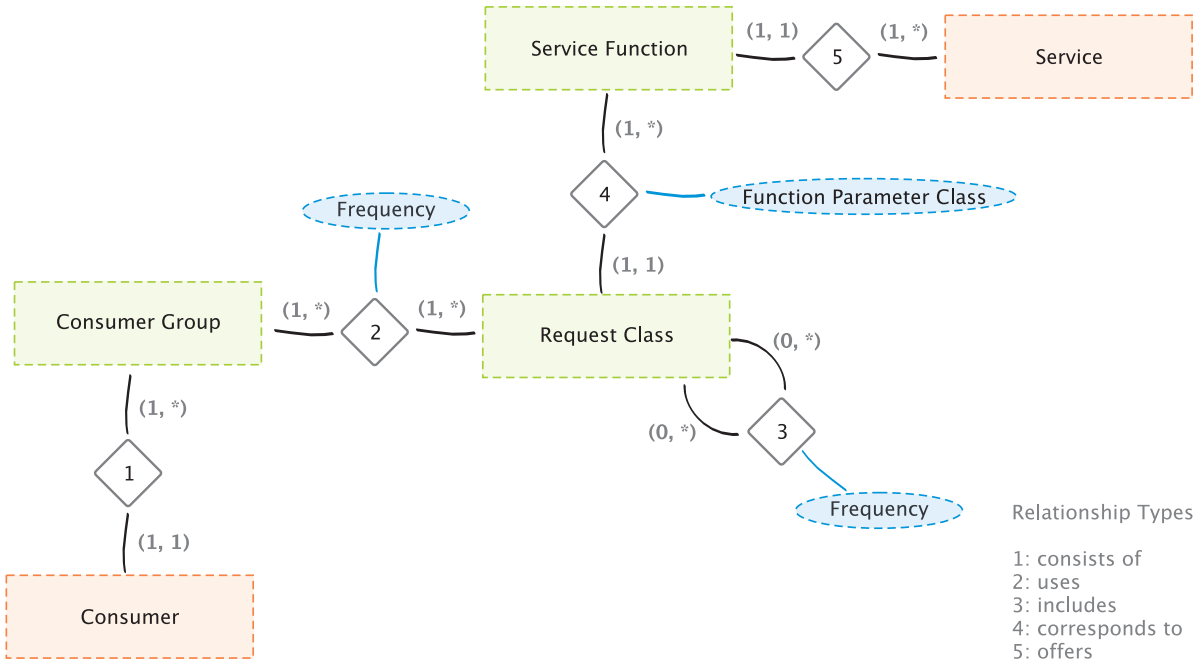


Figure 2: Usage Pattern as ER diagram

6 Service Operations Lifecycle Interaction

Imagine a service provider plans to offer a new web service [(W304] consistent to the business model of Utility Computing. In the first phase of the SOL, business planning for this service offer is conducted. In this example the executive expects three market segments in which the service could successfully be offered. In each segment different consumer numbers and usage behaviour are expected, due to the analyses of typical consumers. Based on these expectations a first simple Usage Pattern instance is derived.

Given this first pattern, IT architects and operations managers now jointly estimate the resource consumption of the future service architecture. This estimation is incorporated into the business planning by deducting the operational costs for the predicted resource consumption. This gives the executive the chance to validate planed price scales at an early stage. Beside the Usage Pattern instance representing the expected consumer behaviour, the executive elaborates a worst and a best-case scenario. In addition to the estimation

of the quantitative consumer behaviour, the executive specifies the service quality to be offered in each market segment.

As a base for their estimations IT architects will need a suitable provisioning model. Both will need a simulation of such a model to evaluate service quality and resource demand of service cascades. Both items, the model and its simulation, are described in [HSPT09].

In the development phase of the SOL the IT architect details the given Usage Pattern instances from the previous SOL phase. In this example it is estimated that the orchestration of another web service is reasonable. To accelerate the development reuse is chosen. Based on this extended Usage Pattern instance, conducting a simulation of the planned architecture helps the IT architect to validate previous resource demand estimations early. Even more importantly: he can validate service quality early and continuously during development.

After the new service is transferred to operations, the responsible manager needs to manage the service provision quality. Beside the indirect ability of consumption management, a continuous capacity planning is essential. Based on the estimation of future usage behaviour, provided by the executive, capacity demand for all offered services is simulated, respecting their interactions. This analysis is conducted as a worst-, standard- and best-case scenario to enable the executive to decide about future investments. Again, these quantitative conditions are described in Usage Patterns, thereby representing the base for simulation runs. Beside this continuous capacity planning in the SOL phase of service operations, service quality is guaranteed by provision management. Provision management ensures service quality by managing the routing of service requests. To calculate a routing decision, Provisioning Factors are used.

7 Provisioning Factors

The active management of service requests at runtime aims to gain direct control of the processing resource utilisation by controlling the routing of service requests. Besides the continuous monitoring of the utilisation of processing resources, the decision about the route of a request is the core of future provision management. To calculate this decision, measurable criteria, that both express technical and economical aspects of the request processing, must be defined. In this work these technical and economical criteria are called Provisioning Factors. These factors represent the qualitative aspect of the consumer-provider relation.

Provisioning Factors are segmented into three main factors:

- *Processing factor:* The processing factor aims to calculate the costs for the processing of a service request on provider-owned resources. These costs derive from the fixed costs for service hosting, e.g. for server acquisition, housing and administrative personnel, and corresponding dynamic costs, e.g. for cooling and power. The process to identify the individual combination of these fixed and dynamic costs is not part of this work. Before the calculation of the processing factor, resource

availability for request processing must be ensured. If the resources are available, the processing costs using the selected resources are calculated. This calculation includes all costs for sub-requests performed by the request. It is known that detailed analyses of large service cascades in order to find the optimum costs or to calculate the exact resource demand at runtime may fail in complex provisioning scenarios. In this case this work suggests calculating approximations instead.

- *Outsourcing factor:* Beside the option to process requests on provider-owned resources, scenarios are conceivable, where it can be an economical alternative to forward requests to other service providers for processing. Such outsourcing decisions can be appropriate for all layers of a service cascade. From processing customer requests on competitor sites in times of peak loads up to the dynamic processor picking for back-end services, such as the retrieval of geological information. The outsourcing factor aims to calculate the costs for external request processing.

- *Neglecting factor:* Instead of the two previous factors the neglecting factor aims to calculate the costs for an intentional violation of the SLA agreed with the consumer. Thereby the violation may at worst consist of a request drop, but also in other aberrations form the given SLA. To achieve this flexibility, the costs for all contracted variations of service level aberrations must be taken into account.

All three Provisioning Factors calculate costs. Combined with the consumer's contracted price list, the profit or loss of a request routing decision can be estimated. Note that all mentioned costs may vary over time on individually contracted factors, such as the time of day or discounts on request amounts. The introduced factors are only proposals used in this work. It is possible to add or remove criteria as needed in other contexts.

8 First Outcomes

First outcomes can be shown analysing an example scenario. In this example the profit of a service provider is analysed during peak demands, where significantly more resources to process all incoming service requests are necessary than are available. The example provider offers a single service to a certain range of consumers. The consumers can be grouped into three SLA groups. Each SLA group differs in maximum request response time, pricing and contractual penalty. It is assumed that in terms of request complexity, request frequency and request number each consumer behaves equally. The number of consumers in the highest and lowest SLA group is equal. The number of consumers in the medium SLA group is double the size of one of the other groups. The metered values in this scenario are the total number of requests and for each SLA group: number of request responses, mean of response duration, request drops, SLA fails and profit.

Compared are two scenario alternatives: classical vs. UC provision management during peak demands. In classical provision management resources are shared at a fixed ratio at runtime. During peak demands, this constraint also applies to more flexible classical resources sharing alternatives, where unused resources can be borrowed among other con-

sumers. For UC provision management it is assumed that request routing is adapted at runtime based on the Provisioning Factors introduced in this work.

The scenario is modelled as discrete-event model presenting a multi-tier IT architecture to process service requests. The model is implemented using the [HSPT09] UC simulation framework. Figure 3 shows the comparison between a classical and an UC simulation run. Both runs represent a simulation period of 30 minutes with 40 requesting consumers using random request invocations. Analysing the outcomes, the distribution and drop rates of requests between each SLA group are even. But there are significant improvements in means of response duration for the primary and secondary SLA group and in SLA fails for the primary SLA group. These shifts directly lead to a significant higher profit in the UC provision management scenario.

	Classic		UC		
[1] Total number of requests	15.594		15.738		
[2] Number of request responses					
Total	13.683	88% of [1]	13.704	87% of [1]	**Earnings per request**
SLA 1	3.539	26% of [2]	3.560	26% of [2]	0,25 €
SLA 2	6.963	51% of [2]	7.007	51% of [2]	0,05 €
SLA 3	3.181	23% of [2]	3.137	23% of [2]	0,01 €
[3] Mean of response duration					
Total	9,05 sec		10,46 sec		**SLA classes**
SLA 1	9,05 sec		7,40 sec		10,00 sec max.
SLA 2	9,05 sec		7,59 sec		15,00 sec max.
SLA 3	9,05 sec		16,38 sec		30,00 sec max.
[4] Request drops					
Total	1.911	12% of [1]	2.034	13% of [1]	**Drop fine per request**
SLA 1	573	30% of [4]	627	31% of [4]	0,50 €
SLA 2	1.129	59% of [4]	1.166	57% of [4]	0,02 €
SLA 3	209	11% of [4]	241	12% of [4]	none
[5] Request response SLA fails					
Total	1.729	11% of [1]	1.216	8% of [1]	**SLA fine per request**
SLA 1	981	57% of [5]	480	39% of [5]	0,25 €
SLA 2	726	42% of [5]	708	58% of [5]	0,01 €
SLA 3	22	1% of [5]	28	2% of [5]	none
[6] Provider profit					
Total	219,56 €		434,68 €	198% of classical [6]	
SLA 1	-35,50 €		179,75 €		
SLA 2	225,56 €		226,25 €		
SLA 3	29,50 €		28,68 €		

Figure 3: Analyses of a simulation run

9 Conclusions and Further Work

In this paper the quantitative and qualitative description of the consumer-to-provider relation in the context of Utility Computing is analysed. Derived from the demands of service quality management and the Utility Computing business model, Usage Patterns are introduced to address the definition of the quantitative consumer-to-provider relation. Usage Patterns are used to enrich the service operations lifecycle to enable the analyses of the qualitative consumer-to-provider relation during strategic planning. To address the qualitative consumer-to-provider relation at runtime, Provisioning Factors are introduced. Building on these factors, requests can be prioritised to optimise provider profits. The

introduced example uses a Usage Pattern to define its demand scenario. The simulation of the Usage Pattern shows that runtime request prioritisation raises profits, while higher service levels benefit from shorter response times. Further research aims to verify the simulation results analysing a real world scenario. The main aspect here will be the calibration of the simulation runs to reflect the current resource consumption of the simulated service requests.

References

[AAR02] Artur Andrzejak, Martin Arlitt, and Jerry Rolia. Bounding the Resource Savings of Utility Computing Models. 2002.

[BMQ+07] Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall. Cloud Computing. *IBM DeveloperWorks*, October 2007.

[Bri10] Encyclopaedia Britannica. public utility. http://www.britannica.com, 2010.

[BT06] Guy Bunker and Darren Thomson. *Delivering Utility Computing: Business-driven IT Optimization*. John Wiley \& Sons, 2006.

[Che76] Peter Pin-Shan Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[Erl07] Thomas Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, 2007.

[HSPT09] Benjamin Heckmann, Ingo Stengel, Andy Phippen, and Guenter Turetschek. Utility Computing simulation. In *ESM'2009 The 2009 European Simulation and Modelling Conference*, pages 175–180, Leicester, United Kingdom, October 2009. EUROSIS-ETI.

[LyCMO06] Qianhui Liang, Jen yao Chung, Steven Miller, and Yang Ouyang. Service Pattern Discovery of Web Service Mining in Web Service Registry-Repository. In *2006 IEEE International Conference on e-Business Engineering (ICEBE'06)*, pages 286–293, Shanghai, China, 2006.

[Men07] Alfredo Mendoza. *Utility Computing Technologies, Standards, and Strategies*. Artech House Inc, April 2007.

[Nee02] Dan Neel. The utility computing promise. http://www.infoworld.com/d/networking/utility-computing-promise-807, April 2002.

[Pap03] M.P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat. No.03CH37417)*, pages 3–12, Rome, Italy, 2003.

[Rap04] M. A. Rappa. The utility business model and the future of computing services. *IBM Syst. J.*, 43(1):32–42, 2004.

[(W304] World Wide Web Consortium (W3C). Web Services Architecture. http://www.w3.org/TR/ws-arch/, February 2004.

[YdAY+06] Chee Shin Yeo, Marcos Dias de Assuncao, Jia Yu, Anthony Sulistio, Srikumar Venugopal, Martin Placek, and Rajkumar Buyya. Utility Computing and Global Grids. *cs/0605056*, May 2006.

[Zha07] Liang-Jie Zhang. *Services Computing: Core Enabling Technology of the Modern Services Industry*. Tsinghua University Press ;;Springer, Beijing ;Berlin ;;New York, 2007.

Service provision in a Utility Computing environment

Benjamin Heckmann

Network Research Group, University of Plymouth, Plymouth, United Kingdom
email: benjamin.heckmann@gmx.de

Abstract

This project is motivated by the gap between technology-centred service provisioning frameworks and the business model Utility Computing. In the beginning this paper introduces the term 'Utility Computing' [UC] as an on-demand service provision business model for Service-oriented Architectures. It distinguishes Utility Computing from technology-originated terms such as Grid or J2EE.

The paper describes these technologies as possible frameworks to implement IT architectures for UC business models. And it determines that actual frameworks are not smart enough to fit the service provisioning demands of small to medium-sized businesses. Therefore a technology-independent and UC-conform service provisioning model is claimed, that enables framework evaluations and simulations of provisioning demands.

Subsequently, the basic structure for a technology-independent, UC-conform service provisioning model is described. As a first step towards such a model this paper introduces the general conditions for such a network, underlying use cases, derived network elements and appropriate workflows. With this as base the overall project aims to provide a technology-abstracted model for service provision and fundamentals for load characteristic simulations for UC environments.

Keywords

SaaS, On Demand, Utility Computing, Grid, Service-oriented Architecture, Service Billing, Web Service Provision, Quality of Service

1. Introduction

1.1. On Demand Service Provision for Service-oriented Architectures

Service-oriented Architectures [SOA] (MacKenzie *et al.*, 2006) are one of the most observed topics in IT today. More and more standard software products are delivered 'SOA-ready', which in most cases means additionally equipped with a webservice interface. This paper will focus on SOAP-based webservices (Booth *et al.*, 2004) as implementation technology for SOAs.

On the service consumer side 'SOA-readiness' means that the encapsulated functionality becomes accessible to each business process step separately. As a result, you can easily rearrange your business processes, while your backend software stays untouched.

On the service provisioning side this means that with standardised interfaces and firewall-friendly protocols, service provision could evolve to its next step: from locally deployed purchased software packages to remotely hosted pay-per-use services.

This business model of leasing remote services and charging the customers per usage is defined with the term 'Utility Computing' [UC] in this paper. It is also known under the terms Software-as-a-Service [SaaS] or On-Demand Computing. Additionally, this paper discusses

UC frameworks, networks, models and further elements that can act as part of the implementation of a Utility Computing business model.

1.2. Utility Computing frameworks for small and medium-sized businesses

Future works should be concerned with the question of whether there a suitable UC framework for the service provision in small and medium-sized businesses [SMB] based on open source software. Currently the estimated answer is: not yet.

To be able to latter follow up this question, an abstract model for a UC framework is required. This paper will attempt to define the basic elements and workflows for a UC network in preparation for designing a technology-abstracted UC model.

1.3. Meshed services and resource prediction

This project also aims to reach conclusions about basic questions linked with the operations of UC services. Simulations of service network behaviour, based upon the model to be elaborated, should shed light on the following questions:

- The behaviour of *highly meshed UC services* in response to service failures and in case of service loops.

- The ability for *resource prediction* for UC service providers to identify bottlenecks in the UC infrastructures including service clients, the UC service network and embedded foreign services.

  Planning of peak demand scenarios for provided services and basic operating figures for pricing strategies, which should be covered by information gathered for bottleneck identification in conjunction with variations of the simulated scenarios.

To recapitulate, this project should deliver a technology-independent, UC-conform model as a background for future UC framework evaluations and simulations of UC provisioning scenarios.

1.4. Outline of research

In the following second chapter the conceptual approach for the overall project is described. The chapter describes the three major steps towards a UC model that enables evaluations of technology-dependent UC implementation frameworks: context gathering, model building and confirmation of the model.

Afterwards the terms Utility Computing, Grid and J2EE are demarcated in the third chapter. The demarcation provides a better understanding for the coherence between UC as a business model and technology frameworks like Gird or J2EE.

As core subject of this paper the basic elements and workflows of a UC network are introduced in chapter four. The elements and corresponding basic workflows are based on a service consumption and a service provision use case. These use cases are derived from works from OGSA, GGF and industry best practices like ITIL.

2. Conceptual approach for the project

2.1. Pre-modelling context building

In the context phase, the project defines its basic terms and elaborates its background and related work. The most important term should be Utility Computing itself. As a business model it is technology-independent and focused on economic opportunities of the utility idea.

As a result, detailed analyses of provisioning costs or capacity demands pre to investments can not be made. Also, comparisons of different provision technologies are not possible. This is due to a missing technology-abstracted service provisioning model substantiating the business model.

In addition, possible current technologies to provide UC-conform services should be examined and demarcated. The gathered information should provide a basis for provisioning conditions.

2.2. Technology-independent, UC-conform service provisioning model

The model building phase of the project is separated into four consecutive steps:

(1) As an initial point for model building, the use cases collected in the context of the Open Grid Services Architecture [OGSA] and results of the EU GRASP project that aim to define an infrastructure for Application Service Provision [ASP] based on GRID technology will be reused.

Additionally, the model should consider the basic characteristics of industry standards like ITIL or CobiT. Based on these existing use cases and industry-class service delivery demands, new UC-centred uses cases must be derived.

(2) Taking the derived UC-centred use cases as a basis, the technology-independent and UC-conform service provisioning components must be identified.

(3) The basic workflows for the component interaction must be described. They should at least enable the model to provide services that are scalable over cost-domains and can be billed according to customer usage.

(4) A complete model must be built based on the defined service provisioning components and workflows. This model should be usable as a basis for simulation-based analyses of UC networks.

2.3. Simulation-based analyses

In the simulation phase of the project, the previously developed model will be utilised for the implementation of a simulation environment for UC-conform service provisioning.

The following activities are necessary:

(1) A suitable simulation framework for the developed model must be selected.

(2) The model must be implemented within the selected framework.

(3) First simulations should be accomplished. They should address the behaviour of highly-meshed UC services and provide the basis for the examination of peak demands within the simulated model.

3. Background and Related Work

As preparation for defining the Utility Computing model, this paper defines and demarcates the terms Utility Computing, Grid and J2EE.

3.1. UC, Grid and J2EE definition

◊ Utility Computing

Utility Computing describes a business model to offer software-based services in the future. While today we are becoming increasingly reliant on computer technology, an interesting question arises: "Is computing the next utility?" (Rappa, 2004)

To answer this question, the term 'utility' first should be defined. The difference in offering a 'service' to a customer or a customer who utilises a 'utility' is shaped by the underlying requirements on the consumer side: necessity, reliability, usability, utilisation, scalability and exclusivity. Additionally, the business model is based on the metering of usage combined with a 'pay as you go' approach. For more detailed description see (Rappa, 2004).

From the service consumer perspective, the most important advantages of Utility Computing are "the reduction of IT-related operational costs and complexity" (Shin Yeo *et al.*, 2006). The investments for the IT infrastructure are no longer static costs for technology and operating staff, but now depend on the usage of the utilised services. As a result the costs become variable.

On the other hand service providers can serve their resources to a wide spread number of users with diverse usage patterns. This increases the chance to minimise unutilised resources on the provider side. "Utility computing also enables providers to achieve a better Return On Investment (ROI) such as Total Cost of Ownership (TCO) […] ." (Shin Yeo *et al.*, 2006) For more detailed description see (Shin Yeo *et al.*, 2006).

Due to the 'pay-per-use' approach of UC there is a new direct relation between IT service provisioning costs and business process costs, especially in the context of Service Oriented Computing [SOC] (Munindar and Huhns, 2005). Following this approach, the costs for processes that utilise UC-based services are quite easy to comprehend. "The provider may be an organization's IT department or an external utility provider, and the service may be storage, computing, or an application." (Foster and Tuecke, 2005)

◊ Grid

Basically, a Grid "coordinates resources that are not subject to centralized control" (Foster, 2002). This means that it is a system that is able to dispose requests for a certain functionality under known resources, regardless of the administrative domain in which a resource is hosted.

One major aspect for achieving this ability is "using standard, open, general-purpose protocols and interfaces" (Foster, 2002) to build a Grid. The final needed characteristic of a Grid is that it must be able "to deliver nontrivial qualities of service" (Foster, 2002), which

implies that in a Grid "the utility of the combined system is significantly greater than that of the sum of its parts" (Foster, 2002). For more detailed description see (Foster, 2002) and (Foster and Kesselmann, 2004).

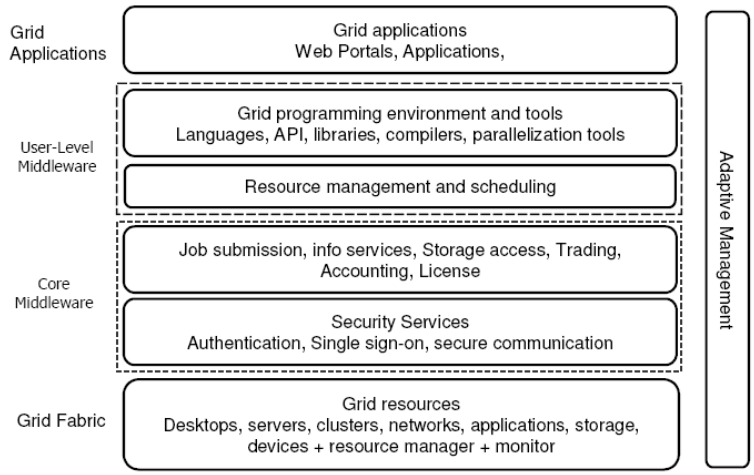


Figure 1: Grid layers (Shin Yeo *et al.*, 2006)

◊ **J2EE**

J2EE is an application model that supports applications that implement enterprise services. "Such applications are inherently complex, potentially accessing data from a variety of sources and distributing applications to a variety of clients." (Sun, 1999) The middle tier of this application model offers its deployed services to consumers. It handles properties like high availability, security and scalability, "to insure that business transactions are accurately and promptly processed" (Sun, 1999). To store the data processed by the middle tier services the EIS-Tier is used. For more detailed description see (Sun, 1999).

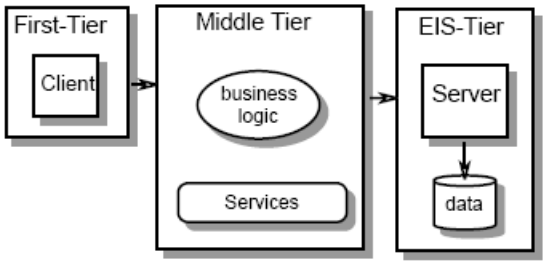


Figure 2: J2EE architecture (Sun, 1999)

3.2. UC, Grid and J2EE demarcation

◊ **UC vs. Grid**

Utility Computing as a business model requires a technical environment to offer its services. Grids have the potential to serve as an appropriate service host. Grids aim to enable resource sharing and problem solving on an infinite number of computing devices. As a result, multi-institutional virtual organizations can be built upon a wide range of computing devices that are logically coupled together and presented as a single unified resource. "The design aims and benefits of Grids are analogous to those of utility computing, thus highlighting the

potential and suitability of Grids to be used as utility computing environments." (Shin Yeo *et al.*, 2006) For more detailed description see (Shin Yeo *et al.*, 2006).

◊ **Grid vs. J2EE**

To implement Grid services, a specific hosting or execution environment is needed. This environment is characterised through certain development tools and programming languages that meet the Grid service semantics. Previous Grid applications are realised by relying on native operating system processes as their hosting environment.

Modern container- or component-based hosting environments such as J2EE can also be used to implement Grid services. These environments offer a framework to build complex applications that offers superior programmability, manageability, flexibility and safety. For more detailed description see (Foster *et al.*, 2002).

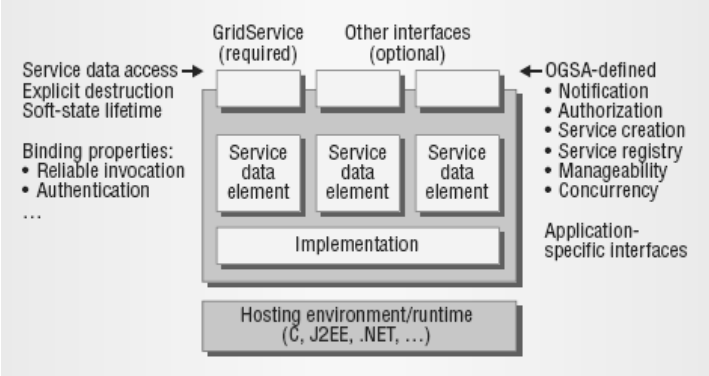


Figure 3: Grid architecture (Foster *et al.*, 2002)

◊ **UC vs. J2EE**

The common trend as described in 'Grid vs. J2EE' is using a Grid that is J2EE-based. For an example of building a J2EE-based Grid, see (Araki, 2004).

A standalone solution for J2EE-based UC is not known. J2EE-clusters are possible, but without billing and cross-side (and therefore cross-cost zones) load-balancing.

> Cluster definition: "group of machines working together to transparently provide enterprise services" (Kang, 2001)

3.3. Demarcation summary

Utility Computing can best be described as a business model for offering services within or to organisations. A Grid could be one technology to build and offer UC-based services. With Grid environments, however, "there is a fundamental gap between the technology and its users" (De Roure *et al.*, 2006). The targeted audience in this project are SMB. For this audience the technology is still too complex and requires too much knowledge commonly not available in-house. For more detailed description see (De Roure *et al.*, 2006).

J2EE as standalone technology is not able to offer UC-based services. Solutions for service-consumer billing or cross-side (cross-cost-domain) load-balancing are lacking within J2EE.

4. Definition of the basic elements of a UC network

4.1. Modelling properties and use cases

The goal for the model building is to at least fulfil the minimum requirements of Utility Computing, which are service provision 'on-demand' and 'pay-per-use' billing. The targeted properties are (currently excepted is the service transaction management):

- SOA service provision
- Extensive load-balancing
- Management of service quality
- Accounting
- Model complexity fitting for SMB (service provision and consumption side)

4.2. Underlying use cases and general conditions

The analyses of the functional requirements are based on the OGSA and GGF use cases (Foster *et al.*, 2004) (MacLaren *et al.*, 2006) (Von Reich, 2004):

- Commercial Data Centre
- Grid Resource Resellers
- Inter Grid
- Resource Usage Service
- IT Infrastructure and Management
- Grid-based ASP for Business
- Grid Monitoring Architecture

Additionally, it is based on the main results of the EU GRASP project that aims to define an infrastructure for Application Service Provision based on Grid technology (Dimitrakos *et al.*, 2004).

Complementing the basic requirements in the industry standard ITIL with focus on service delivery best practices are incorporated. Also basic requirements from the CobiT (ISACA, 2005) framework are included.

4.3. Derived use cases for the model

Starting from the underlying use cases and general conditions brought together previously, the following two use cases define the basic functional requirements the model should fulfil. Aggregating the service delivery requirements and matching them against the predefined goals for the model resulted in the subsequently-denoted use cases for UC service delivery operation.

◊ **Service consumption use case overview**

- Discovery
- Brokering and load-balancing
- Orchestration
- Authentication and Authorisation

- Monitoring, Metering and Accounting
- Fault Handling and Logging
- Corresponding Policies

◊ **Service provision use case overview**

- Data Access
- Provisioning
- Embedded legacy applications
- Synchronous and asynchronous usage
- Administration
- Corresponding Policies

4.4. Elements derived from the model use cases

◊ **Service type**

The element represents a definition of a service class with distinctive business functionality and a standardised public interface.

◊ **Service instance**

This represents an instance of a service type that can handle multiple service requests simultaneously, and exists as a subset of a service host and applies SLA quotas. The element supports standby, online and offline modes.

◊ **Service host**

The element represents a host for service instances that can only host one service instance of a service type at a time.

◊ **Service consumer**

The consumer invokes service instances by sending service requests.

◊ **Service request**

A request is an invocation of a service initiated by a service consumer. The invocation always includes the associated service response (synchronous or asynchronous).

◊ **Service registry**

The registry authenticates service consumers.

◊ **Service broker**

A broker authorises service requests, forwards service requests to the most suitable service load-balancer or third-party service broker (with respect to SLA and cost calculations) and creates service request bills (including third-party service type utilisation costs and SLA violations).

◊ **Service load-balancer**

The load-balancer represents a physical location or a cost class. It queues service requests (if necessary) and forwards service requests to most suitable service instances (with respect to SLA). Also it deploys, activates, deactivates or removes service instances on service hosts as necessary (e.g. for load-balancing or in case of failure).

◊ **Service monitoring**

This element monitors the SLAs per service request.

◊ **Policies**

These elements define the general conditions for brokering (per service consumer), error and event handling (per service type) and additionally security conditions (per service consumer).

Information is stored near their creation or consumption location. Information is provided directly through its storage location.

◊ **Variations to the derived use cases for the model**

Not incorporated in the element definition are the embedded legacy applications, administration and policies areas of the service provision use case.

The orchestration of existing services into new services is indirectly supported through the provision of new service types. This means that if you want to orchestrate existing service types to new service types, you must build a new service type and as internal functionality invoke and compose the existing services.

4.5. Workflows for the model

In workflow steps the '→' sign is read as 'requests' and marks a request track. Steps can be marked as optional to the initially requesting instance. Steps marked as '*TERMINATOR*' are always executed at the end of any workflow, regardless of the workflow type (e.g. 1_SSC, 2_CoSC, 3_CaSC).

◊ **1_SSC: Simple service consumption workflow**

The following workflow describes the simplest possible service request in the model:

1) Service consumer → Service registry *OPTIONAL*

Authenticated service consumers can request service type information including cost information and available service brokers.

2) Service consumer → Service broker → Service load-balancer → Service instance

   Authenticated service consumers can send service requests using a service broker. In response they get the service state and a request bill.

3) Service broker → Service registry

   This step is invoked by step 2 and transmits the service consumer authentication data and service type to get authorisation information and brokering policy.

4) Service load-balancer → Service monitoring → Service host

   This step is invoked by step 2 and collects the service hosts load data.

5) Service instance → Service monitoring

   This step is invoked by step 2 and reports the individual service request load during processing and according events.

6) Service broker → Service monitoring *TERMINATOR*

   This step is invoked by step 2 and closes a service request by reporting third-party service usage information and the issued service bill to the monitoring service.

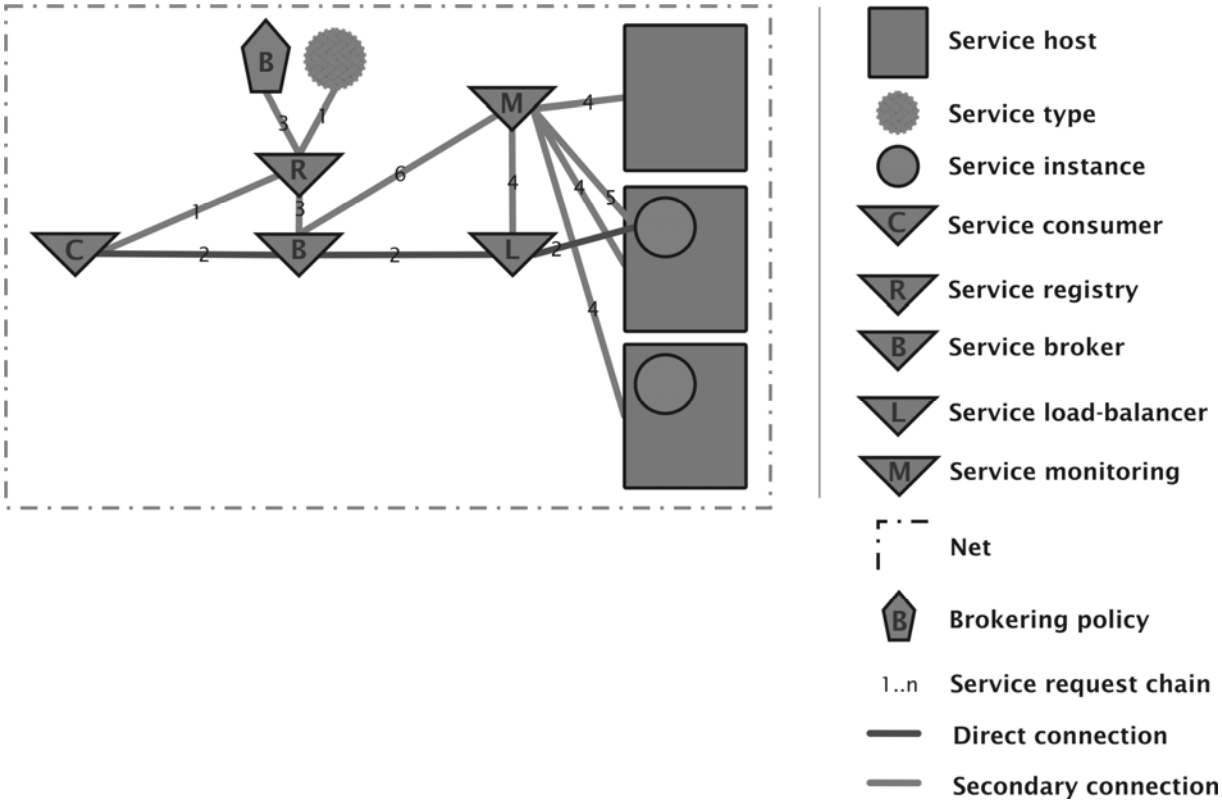


Figure 4: 1_SSC network view

◊ **2_CoSC: Complex service consumption workflow**

The workflow for complex service consumption expands the basic workflow for simple service consumption [1_SSC]. It describes a more complex workflow within the model by still providing a single service type.

7) Service broker → Service load-balancer

   This step is invoked by step 2 and collects the service type utilisation information per load-balancer.

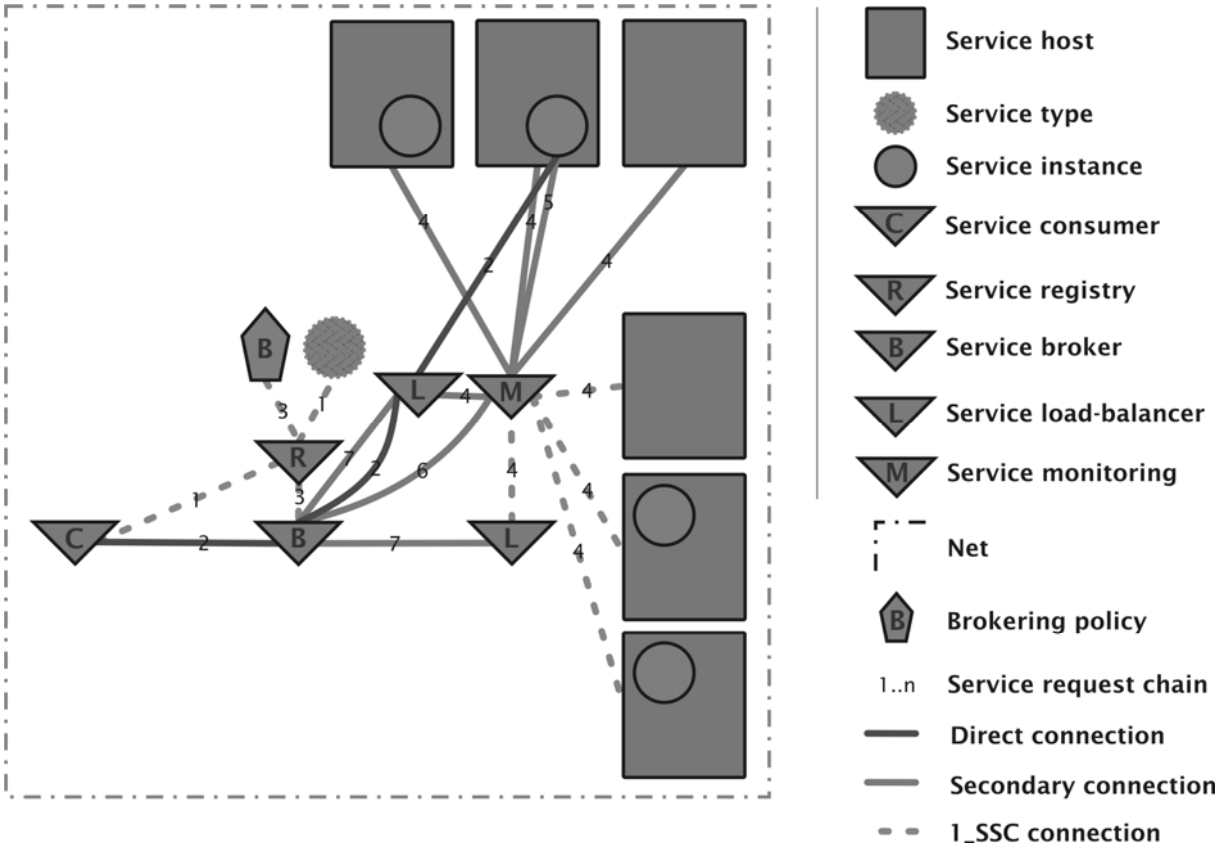


Figure 5: 2_CoSC network view

◊ **3_CaSC: Cascaded service consumption workflow**

The workflow for cascaded service consumption expands the workflow for complex service consumption [2_CoSC]. It describes a workflow that utilises an externally-provided service.

8) Service instance (local) → Service broker (local) → Service broker (third-party) → Service load-balancer (third-party) → Service instance (third-party)

   This step is invoked by step 2 and invokes a third-party service. The invoking instance sends its service request enhanced with service consumer authentication data by the service broker. In the responding data, the service request response and state are used by the service instance. The request bill is extracted by the service broker.

9) Service broker (local) → Service broker (third-party) → Service monitoring (third-party)

   This step is invoked by step 8 and collects the service type utilisation information for the third-party service.

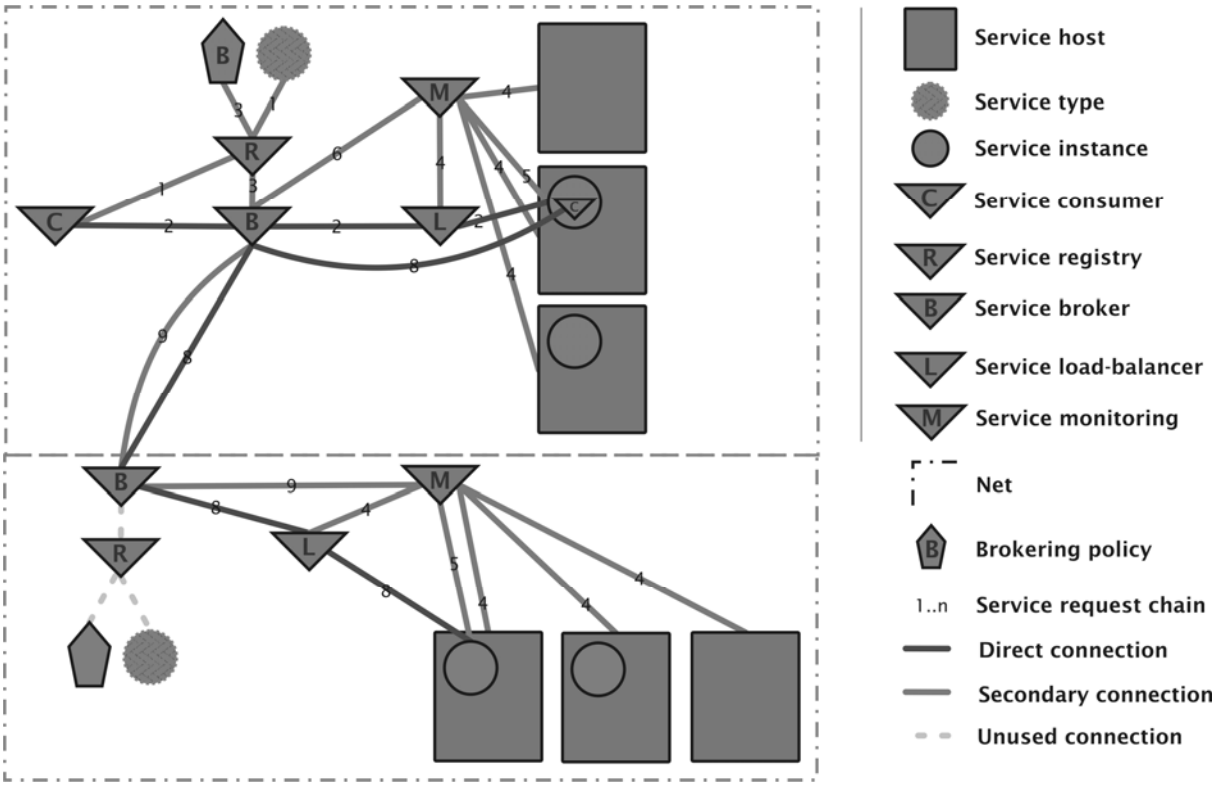


Figure 6: 3_CaSC network view

◊ **Service instances as service consumers**

Why service instances should not invoke their embedded service calls directly: Service instances need to act as service consumers, when they want to embed functionality provided by other services. If service instances would call their embedded services directly, the system would lose control over:

- Service provider changes
- Service billing
- Authentication changes
- Service load-balancing

With a centralised element as provided with the service broker, external service invocation will be handled by the broker. This introduces a new implementation strategy for software developers of service-oriented architectures.

4.6. Enhancements compared to plain SOA

The four main differences to service provision in basic service-oriented architectures are:

- *Pay-per-use base*, achieved through the optional usage of the service request bill

- *Internal active SLA-control*, achieved through the service type utilisation combined with the service load-balancer

- *External passive SLA-control*, optionally achieved through the service type utilisation combined with the service broker

- *Centralised service consumption management*, achieved through the service broker

  Thus not only is the service provision managed through a central proxy-like instance, but the service consumption is also managed centrally.

As an analogy, compare the evolved architecture with the IBM proposal for a utility computing architecture in (Kloppmann *et al.*, 2004).

5. Summary

This paper introduced Utility Computing as the on-demand service provision business model for Service-oriented Architectures. As a trigger for the project, the question for a suitable UC framework for small and medium-sized businesses was raised. Subsequently, a technology-independent, UC-conform service provisioning model was claimed as a precondition to answer this question.

The paper goes on to expresses the need to characterise the behaviour of meshed services and the necessity to provide room for resource prediction in UC networks. As a suitable solution, a simulation framework based on the previously demanded UC model is described.

To distinguish Utility Computing clearly from technologies such as Grid or J2EE these terms are defined and demarcated. The conceptual approach for the project is explained and the first steps towards a technology-abstracted UC model are presented. The gathering of the basic network elements including the general conditions for the network, its developed uses cases, the derived network elements and the appropriate workflows are introduced.

Based on these results, the project will now start to build a complete UC model as a precondition for the simulation phase of the project.

References

Abraham Kang, 08/03/01. J2EE clustering, Part 1. JavaWorld.com

Araki, T. 2004. *Autonomic WWW server management with distributed resources*. In Proceedings of the 2nd Workshop on Middleware For Grid Computing (Toronto, Ontario, Canada, October 18 - 22, 2004). MGC '04, vol. 76. ACM Press, New York, NY, 81-86.

C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, Booz Allen Hamilton, 2006. *Reference Model for Service Oriented Architecture 1.0*. OASIS Committee Specification 1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

Chair: K. Jeffery; Editor-in-Chief: D. De Roure, 2006. *Future for European Grids: GRIDs and Service Oriented Knowledge Utilities*. European Commission, published in January 2006.

Chee Shin Yeo, Marcos Dias de Assunção, Jia Yu, Anthony Sulistio, Srikumar Venugopal, Martin Placek, and Rajkumar Buyya, *Utility Computing on Global Grids*, Hossein Bidgoli (ed), The Handbook of Computer Networks, John Wiley & Sons, New York, USA, accepted in April 2006 and in print.

David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard, 2004. *Web Services Architecture*. W3C Working Group Note 11, http://www.w3.org/TR/ws-arch/.

Foster, I. and Tuecke, S. 2005. *Describing the elephant: the different faces of IT as service*. Queue 3, 6 (Jul. 2005), 26-29.

Foster, I. Kesselman, C. Nick, J.M. Tuecke, S., 2002. *Grid services for distributed system integration.* Computer, Volume 35, Issue 6, June 2002 Page(s):37 – 46

I. Foster, D. Gannon, H. Kishimoto, Jeffrin J. Von Reich, 2004. *Open Grid Services Architecture Use Cases*. GGF, http://www.ggf.org/documents/GFD.29.pdf.

Ian Foster, 2002. *What is the Grid? A Three Point Checklist*. Argonne National Laboratory & University of Chicago.

Ian Foster, Carl Kesselmann, 2004. *The Grid 2 – Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

ISACA, 2005. *COBIT 4.0*. Printed in the United States of America, 2005. ISBN 1-933284-37-4

J. MacLaren, S. Newhouse, T. Haupt, K. Keahey, W. Lee, 2006. *Grid Economy Use Cases*. GGF, http://www.ggf.org/documents/GFD.60.pdf.

Jeffrin J. Von Reich, 2004. *Open Grid Services Architecture: Second Tier Use Cases*. OGSA-WG GGF, http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-wg/docman.root.published_documents.use_cases_1_0/doc13574.

M. Kloppmann, D. Konig, F. Leymann, G. Pfau, D. Roller, 2004. *Business process choreography in WebSphere: Combining the power of BPEL and J2EE*. IBM Systems Journal, Volume 43 Issue 2.

Munindar P. Singh, Michael, N. Huhns, 2005. *Service Oriented Computing Semantics, Process, Agents*. Hrsg: John Wiley & Sons.

Rappa, M. A. 2004. *The utility business model and the future of computing services.* IBM Syst. J. 43, 1 (Jan. 2004), 32-42.

Simplified Guide to J2EE, 1999. Sun Microsystems, Inc.

T. Dimitrakos, D. Mac Randal, S. Wesner, B. Serhan, P. Ritrovato, G. Laria, 2004. *Overview of an architecture enabling Grid based Application Service Provision*. AxGrid '04 , Nicosia, 28-30 January, 2004.

UTILITY COMPUTING SIMULATION

Benjamin Heckmann
Ingo Stengel
Günter Turetschek
University of Applied Sciences Darmstadt
Haardtring 100
D-64295 Darmstadt, Germany
E-mail: benjamin.heckmann@gmx.de

Andy Phippen
University of Plymouth
Room 405a, Cookworthy Building, Drake Circus
Plymouth, Devon, PL4 8AA, UK

KEYWORDS

SaaS, Cloud Computing, SOA, Service Billing, Service Provision, QoS

ABSTRACT

Utility Computing (UC) misses an explicit definition of the core relation between IT resource utilisation, its total costs and service prices. Additionally, the implications of complex usage scenarios occurring in UC have not been examined for the service operations lifecycle. Missing those, UC service offers fail in: prediction of resource utilisation and dependent operational costs prediction, calculation of subsequent price scales, and subsequent runtime gross price calculations.

In this paper a strategy to handle UC's complexity proposing a simulation model to support each step in the service operations lifecycle is presented. The implementation approach for the model is based on OMNeT++. First simulation outcomes are presented.

INTRODUCTION

This paper starts with a short definition of the term Utility Computing as a business model. Afterwards a common Service Operations Lifecycle is defined that has been derived from ITIL. After setting the context, the research objectives and the related research approach are introduced. Subsequent the evolved strategy that is able to handle UC's complexity is outlined. This strategy includes the demand for an UC provisioning model and a corresponding simulation model. Both models and the implementation of the simulation of the UC provisioning model are introduced. First outcomes of simulation runs are shown. Corresponding conclusions and further works are discussed.

UTILITY COMPUTING

This work is focused on the modelling and simulation of service usage in the context of Utility Computing (UC). The term utility thereby refers to the field of industry. Here a public utility (Encyclopaedia Britannica, 2008) describes an enterprise that provides certain classes of services to a wide range of consumers.

The name Utility Computing indicates the vision of IT-based services comparable to public utilities. In this work Utility Computing is defined as a business model (Weill, 2001) for service providers offering IT-based services and charging service consumers per usage, according to (Rappa, 2004). From the provider's IT perspective UC is about service provision that is able to scale dynamically, according to real-time fluctuations in demand (Bunker *et al.*, 2006). Additionally, UC service provision offers its services equipped with the ability to charge service consumption per use (Neel, 2002).

From a consumer's perspective UC is related to "the reduction of IT-related operational costs and complexity" (Yeo *et al.*, 2006). Both perspectives, provision and consumption, have in common to target a better utilisation of generally underutilised IT resources (Andrzejak *et al.*, 2002) on both sides. In summary, UC implicitly claims an abstract description how IT resource utilisation, its total costs and service prices relate (see Figure 1).

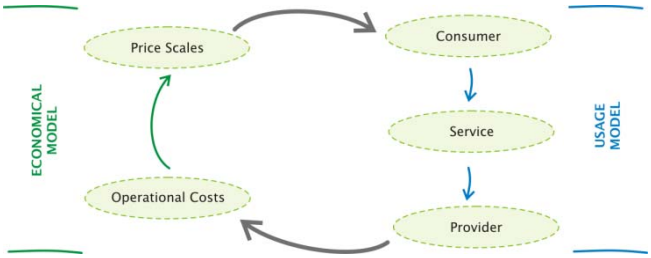


Figure 1: UC's resource – cost – price relation

Thereby Utility Computing does not refer to a specific IT service definition. From a business perspective any service that economically makes sense to be charged by its usage is addressed by UC. Therefore a more abstract service definition will be the most suitable for UC: A service represents a type of relationship-based interaction between a service provider and a service consumer to achieve a certain solution objective. (Zhang *et al.*, 2007) This definition considers the definitions of (Fitzsimmons *et al.*, 2006) from the economics perspective and (Gronroos, 2000) from the marketing perspective. From a technical perspective there are several types of services that fit into this definition, e.g. SOAP web services, HTTP web servers or Xen virtual infrastructures.

SERVICE OPERATIONS LIFECYCLE

In the context of Utility Computing service provision a lightweight definition of a service lifecycle is necessary to obtain an overview of lifecycle stakeholders and basic activities relevant for service provision. As a basis for the definition of a lifecycle in this work, the basic lifecycle

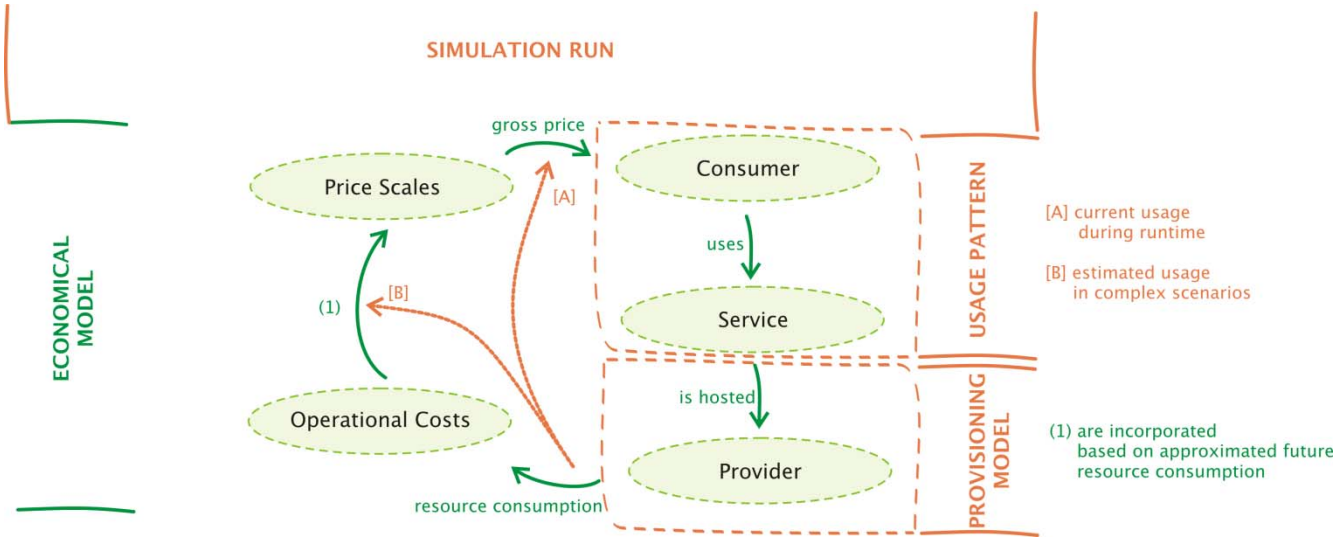


Figure 2: UC relations in the business planning

described in (Zhang et al., 2007) and the aggregation of the ITIL v3 service lifecycle described in (Beard, 2008) are used. Both descriptions can be aggregated to the three main lifecycle phases: Service business planning, service development and service operations.

The lifecycle phase of service business planning is addressing service strategy and service engagement to implement a business model. In classical IT business models, not based on the vision of UC, IT resource utilisation, its total costs and service prices only relate indirectly (see economical model in Figure 2).

During service development the service lifecycle is responsible for the design and implementation of services. This phase also includes the transition process from an implemented service to a deployed, ready for operations service. The phase service operations focuses on the provision of services. This addresses effectiveness and efficiency in delivery and support of services.

RESEARCH OBJECTIVES

The overall context of this work focuses on specific aspects of the service operations lifecycle (SOL) for service offers based on the business model of Utility Computing. In the phase of service business planning this work refers to the corresponding service properties and service usage profiles resulting from the previous UC definition. During service development and the phase of service operations this work will focus on services in the technical context of Service-oriented Computing (SOC) (Papazoglou, 2003) consistent to the paradigm of Cloud Computing as described by (Boss *et al.*, 2007).

In this context a description of the modifications necessary to transfer a standard service operations lifecycle into a UC SOL is missing. This includes the demand for an explicit definition of UC's core relation between IT resource utilisation, its total costs and service prices. Also specific attention must be given to the implications of complex UC usage scenarios.

The unidentified implications of complex UC usage scenarios, considerably compromise the planning, development and operation of UC service offers. Under these conditions the prediction of resource utilisation and dependent operational costs prediction, calculation of subsequent price scales, and subsequent runtime gross price calculations will fail.

RESEARCH APPROACH

The overall work starts from the business perspective, as technical requirements depend on the business requirements imposed. Therefore, a five step approach to find solutions for the specified objectives is proposed:

(1) Describe the current state of service usage in the context of Utility Computing.

(2) Elaborate a detailed definition for the relation between a service and its consumer.

(3) Analyse the SOL of UC services.

(4) Determine the implications of complex UC usage scenarios regarding SOL.

(5) Deduct a corresponding strategy to handle the complexity.

This paper focuses on the simulation of the UC model developed as part of the overall work. The simulation, as well as the UC model, is part of the developed strategy to handle the complexity of UC usage scenarios.

STRATEGY TO HANDLE UC'S COMPLEXITY

The overall work analyses the SOL to identify where modifications allow an optimised support for UC scenarios. Beginning with the phase of service business planning, the classical relation between resource utilisation, costs and prices is examined. Advanced relations for service provision in UC were elaborated as show in Figure 2. The relation marked with [A] adds a runtime relation between

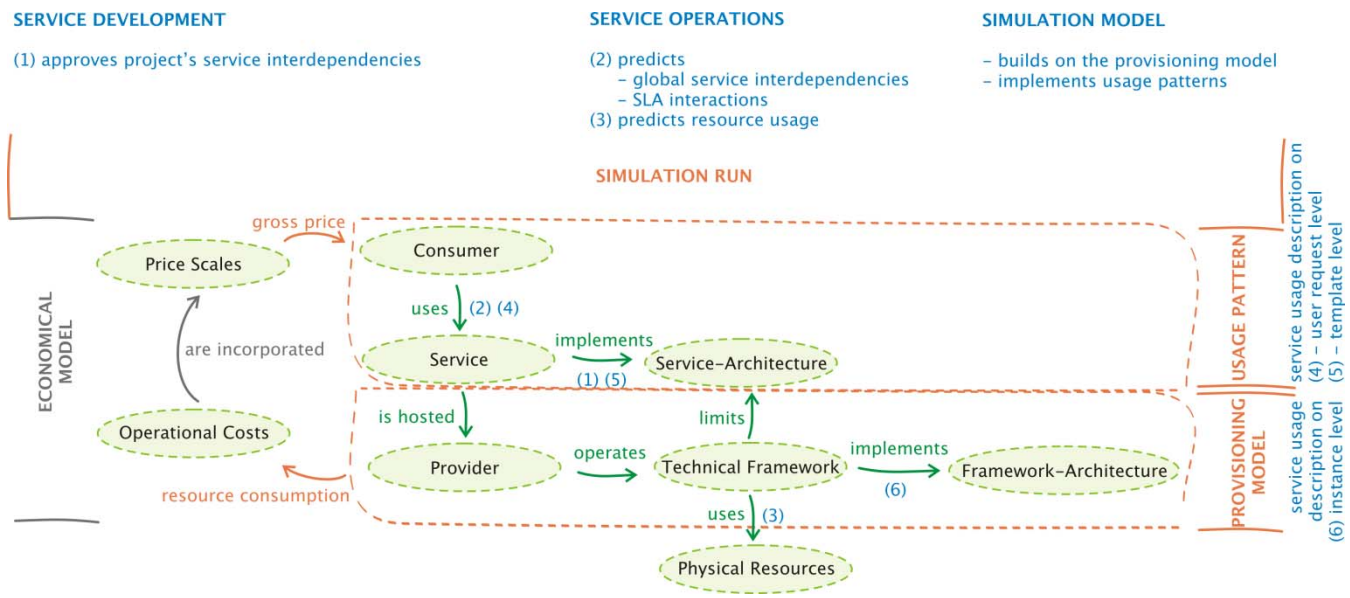


Figure 3: UC relations in service development and operations

the current usage and the gross price calculation that is essential to offer pay-per-use in UC scenarios. Beside this, service providers have to deal with complex usage scenarios, added by relation [B]. To enable the direct relation [A], constraints for UC service provision are necessary. These constraints must describe the requirements to enable this relation during service development and operation.

In the service development phase of the service lifecycle, new data including Usage Patterns specific for on-demand IT infrastructures need to be integrated into the development process (Mendoza, 2007) to improve service quality (Heckmann, 2009). To support the planning of framework architectures or the selection of framework implementations, the definition of relevant UC service provision constraints is necessary.

In the service operations phase, there are no adequate tools to evaluate service interdependencies between all hosted services. Also the Service Level Agreement (SLA) interactions between all hosted services cannot be estimated. Nor the resource planning for services to ensure contracted service levels, respecting resource consumption of other services hosted on shared resources, cannot be analysed without adequate tools for complex UC scenarios.

As a result, of the analyses of the service operations lifecycle, four major strategies for the reduction of the complexity of UC service provision can be identified:

- Define UC constraints for service architectures
- Enable the analysis of service interdependencies on development and operations level
- Permit the analysis of SLA interactions and resource prediction
- Support the proof of price scales

To implement these modifications the development of a technology-agnostic UC service provision model and a corresponding technology-abstracted UC simulation environment is proposed. See Figure 3 for a detailed overview of all previously addressed relations.

UC MODEL

As the previous strategy suggests, the overall work defined a technology-agnostic UC model (Heckmann, 2007). In summary the model consists of eleven abstract elements, logically grouping demanded functionalities, and three basic workflows, which describe the minimum demanded interaction of those elements.

The abstract elements are consumer groups requesting services with a certain member count, request frequency and characteristics, a broker to forward requests according to costs aspects, a load-balancer to forward respecting load aspects of a request, a host offering resources such as computing cycles, memory, storage and network, and service instances consuming offered resources. Additionally some elements to organise service provision: a registry, monitoring, and a service type element. In a derived technical IT architecture these functional groups can be represented as standalone components, but could also be combined in joint architectural elements. As basic workflows a simple service consumption workflow, a complex service consumption workflow, and a cascaded service consumption workflow where defined.

Other models in this context have been proposed by (Mendoza, 2007), (Zhang *et al.*, 2007) or (Bunker et al., 2006). The model of Bunker and Thomson is the most inadequate of them, since it provides too few details to be helpful for IT architects to design a suitable UC architecture for a specific service provision scenario. The model delivers only a quick overall IT strategy to the provision of UC-based services. The UC model by Zhang, Zhang and Cai was developed from a business management perspective. It specifically aims to the provision of SOAP-based web services and describes in detail how those should be provided. While the model of Mendoza uses a very efficient model building approach, starting from a

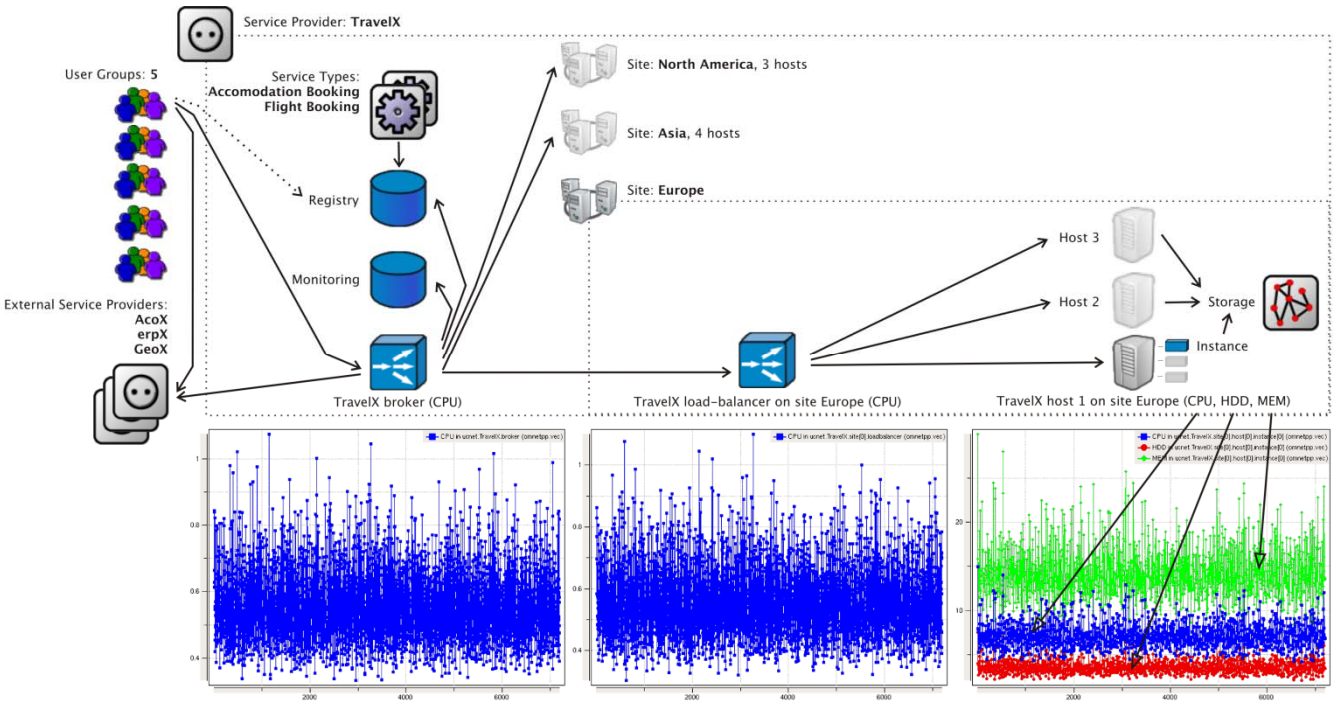


Figure 4: Simulation model as multi-tier architecture

technological perspective. Both of the afore mentioned models are very complex and technology-dependent. Therefore a custom model building was conducted, targeting a lightweight technology-agnostic solution.

SIMULATION MODEL

The simulation model represents a multi-tier architecture (see Figure 4) for the UC-conform provision of services in service-oriented computing. The functionalities described in the UC model have been transformed into the simulation model that implements this architecture. The current implementation is capable to simulate:

- Complex user behaviour (user group): Messages can be sent with random or fixed timeslots to control the amount of messages arriving at the broker. The resource consumption for transport and processing of the embedded service request can be determined separately. It is also possible to configure transport priorities. Each service request can include a free number of subrequests to represent service cascades.

- Resource measurement and monitoring for computing cycles, memory and disk space: The hardware resources simulated and monitored are computing cycles, memory and disk space. Network traffic (bandwidth and delay) is currently not monitored, but simulated. Additionally monitored are the load-balancer and broker queues and their message transport resource consumption. The overall resource consumption for the storage network is also traced.

- Message billing to service consumers (broker): To each request response a bill based on the processing sites computing cycles, memory and storage costs will be attached. The consumption of these resources during processing of the request is billed, and it is possible to add additional per site and per consumer margins.

- Message routing by site costs (broker): Messages get routed to a site with enough resources to process the request and the least costs for processing.

- Message routing by resource demand (load-balancer): Messages are routed by a site's load-balancer to a host with enough resources.

- Message queuing (broker & load-balancer): Messages are temporarily stored within the service broker or service load-balancer when not enough resources for their processing are available. They are recalled from queue after a certain scheduling time and entered again in the scheduling sequence of either the service broker or the service load-balancer. In doing so the queuing consumes resources in the system, and if the system balancer runs out of resources, incoming messages are dropped.

The simulation model is implemented based on the discrete event simulation environment OMNeT++ (Varga, 2001). For each simulation run it is possible to determine the number of user groups requesting services. It is possible to vary the total number of group members, the behaviour timing as well as the type of request in meanings of service type and request complexity individually per user group. It is possible to specify any service type and any number of service providers. Each provider may have several sites with any number of hosts. Each host can be individually equipped with computing power, memory and storage. Additionally each site has access to a storage network to

estimate storage network loads. For each user group and service provider the price relation to each service type can be individually adapted. This highly flexible configuration targets the necessity to represent complex UC scenarios.

FIRST OUTCOMES

The largest test scenario currently simulated represents a virtual travel booking processor with 2770 consumers in five groups, each sending a single request of the same service type including two subrequests to external service providers. Thereby each request's initialisation is randomly scheduled within a given timeframe. As outcome of each simulation run a data pool consisting of all values stated in the resource measurement and monitoring definition of the simulation model is provided. As examples for graphs based on parts of the outcomes, in Figure 4 the consumption of computing cycles characterised as the CPU utilisation is shown for the TravelX broker, a load-balancer and a host. The host's graph also shows the memory (MEM) and disk space (HDD) utilisation on the contemplated host.

Additionally to simulation runs testing a large amount of concurrent users, the implementation showed that it is able to simulate the behaviour of highly meshed service cascades. Even if it comes to special cases like looping service requests, where subrequest providers themselves use services provided by the original subrequest initiator.

Or in case of internal subrequests occurring, where the provision of a service involves requests to other internally provided services.

CONCLUSIONS AND FURTHER WORK

This paper identifies the modifications necessary to transfer the standard SOL into a UC SOL. As part of the presented strategy to handle UC's complexity a simulation model is introduced. First tests of this simulation model have shown that it is possible to represent complex scenarios. The prediction of resource utilisation, dependent operational costs and subsequent runtime gross prices has been shown in virtual scenarios. Further the simulation model mustbe validated analysing real world scenarios. The main aspect for adequate representation of the service's behaviour will be the calibration of the simulation runs to reflect the current resource consumption of service requests. Here further research has to be conducted.

Also part of future research must be the documentation of theoretical aspects of the simulation model building. This includes the relation between discrete event simulation and queuing concepts from queuing theory, the revision of relevant probability topics and the relevant background in stochastic processes.

It is assumed that the technology-abstracted simulation model can also be used for the simulation of RESTful or simple services, such as web servers. Here further research will be conducted.

BIOGRAPHIES

Benjamin Heckmann is a researcher at the aiDa research center in Dieburg and PhD student at the University of Plymouth, UK. He holds a M.Sc. in computer science. His research interests are in the areas of Utility Computing, Cloud Computing, Unified Communications and IT-Security.

Ingo Stengel graduated at the Cork Institute of Technology, Ireland. He is co-founder and Executive Director of the igdv-Centre for Advanced Learning, Media and Simulation at the University of Applied Sciences Darmstadt. His research interests are in the area of Multiagent-Systems, Simulation Software, IT-Security and Advanced Learning.

Andy Phippen received his PhD in the year 2001. He is Senior Lecturer in Business Enterprise and Ethics at the University of Plymouth. His research focuses on the impact of software development and learning & teaching in higher education. Further, he is the director of the IT liaison at the University of Plymouth.

Günter Turetschek is Professor for computer science at the University of Applied Sciences Darmstadt; co-founder and director of the Institute for Applied Informatics Darmstadt (aiDa). His research interests are in the area of Business Computing, Unified Communications and Utility Computing.

REFERENCES

Andrzejak, A., J. Rolia and M. Arlitt. 2002. "Bounding Resource Savings of Utility Computing Models". HP Labs Technical Report HPL-2002-339.

Beard, H. 2008. *Cloud Computing Best Practices for Managing and Measuring Processes for On-Demand Computing, Applications and Data Centers in the Cloud with Slas*. Emereo Pty Ltd.

Boss, G., P. Malladi, D. Quan, L. Legregni and H. Hall. 2007. "Cloud computing". IBM, developerWorks, WebSphere, High Performance On Demand Solutions.

Bunker, G.; and D. Thompson. 2006. *Delivering Utility Computing: Business-driven IT Optimization*. John Wiley & Sons.

Encyclopaedia Britannica, 2008. *public utility*. In Encyclopaedia Britannica Online, retrieved December, 2008.

Fitzsimmons, J.A.; and M.J. Fitzsimmons. 2006. *Service Management: Operations, Strategy, and Information Technology*. 5th Ed., Irwin/McGraw-Hill, Homewood, IL.

Gronroos, C. 2000. *Service Management and Marketing: A Customer Relationship Management Approach*. John Wiley & Sons.

Heckmann, B. 2007. "Service provision in a utility computing environment". SEIN 2007, University of Plymouth, 14-15 June 2007.

Heckmann, B. 2009. "Technology-agnostic definition of the Utility Computing service operations lifecycle". Transfer Report, University of Plymouth, April 2009.

Mendoza, A., 2007. *Utility Computing Technologies, Standards, and Strategies*. Artech House Inc.

Neel, D. 2002. "The Utility Computing Promise". InfoWorld, April 12, 2002.

Papazoglou, M.P. 2003. "Service-oriented computing: concepts, characteristics and directions". Web Information Systems

Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on 10-12 Dec. 2003 Page(s):3 – 12.

Rappa, M.A. 2004. “The utility business model and the future of computing services”. IBM Syst. J. 43, 1 (Jan. 2004), 32-42.

Yeo, C.S., M.D. Assunção, J. Yu, A. Sulistio, S. Venugopal, M. Placek and R. Buyya. 2006. “Utility Computing on Global Grids”. Hossein Bidgoli (ed), The Handbook of Computer Networks, John Wiley & Sons, New York, USA, accepted in April 2006 and in print.

Varga, A. 1997. “Flexible topology description language for simulation programs”. Simulation in industry: 9th European Simulation Symposium 1997:225-229.

Varga, A. 2001. “The OMNeT++ Discrete Event Simulation System”. In the Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic.

Weill, P. and M.R. Vitale. 2001. “Place to space: Migrating to eBusiness Models”. Boston, Harvard Business School Press.

Zhang, L.-J.; J. Zhang; and H. Cai. 2007. *Services Computing, Core Enabling Technology of the Modern Services Industry*. published by Springer and Tsinghua University Press.