Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 2892

Provisioning of Software as a Service Applications in the Cloud

Christoph Alexander Fehling

Course of Study:	Computer Science
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dipl.-Inf. Ralph Mietzner
Commenced:	01. January 2009
Completed:	02. July 2009
CR-Classification:	H.4.1, K.1, K.6.2, K.6.3, K.6.4, D.2.11

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction

Software as a Service is a Software deployment method. Vendors employing it license customers to access the software on demand over the Internet. While accessing software over a network is already widely accepted, Software as a Service introduces two new characteristics. First it allows that computing resources are shared between customers without them noticing. Second it respects that costumers have different requirements by enabling them to adjust the properties of the application. This customization ranges from altering application internal processes to specifying individual service level agreements (SLA)[JMS02, p. 3].
Software as a Service vendors are therefore able to extend their target market drastically since the demands of a much larger consumer group can be met.

1.1 Challenges

To fulfill the additional characteristics of Software as a Service a number of challenges have to be addressed. Customizable qualities such as availability or security are strongly interleaved with the computing environment. Since fine nuances of these qualities are required for good consumer targeting, Software as a Service vendors have to manage multiple computing environments.
Integration of these environments is very important for customization. It empowers customers to select quality of services not only for the application as a whole but with a finer granular. Application components vital to a customer may be deployed in high available environments while others may reside in less powerful ones.
Load balancing over homogeneous resources has been researched in great detail [Kop02]. However with many environments being managed load balancing over heterogeneous resource pools is required. Consumers requesting a low quality service could also be served by high quality services. In case the high quality service displays a bad utilization this delegation might result in a reduced cost for the overall system.
Finally the system is required to flexibly and quickly react to changing user numbers. Technologies enabling such a dynamic allocation of resources are commonly summarized as Cloud Computing. Central elements are the virtualization of resources [DR07, p. 6 ff.] and access through them over a network. The later indicates that the Software as a Service application itself can be part of a Computing Cloud. Two types of Computing Clouds are differentiated. Public Computing Clouds are accessible to a wide range of users. A private Computing Cloud is a private data center which is managed using Cloud Computing technologies.

1.2 Goals

Within the scope of this thesis methods addressing the mentioned challenges are researched. Integration of multiple computing environments and the load balancing of users according to their requested quality of services is enabled.
To achieve this an abstract data model is defined describing the resources constituting the managed system as well as its users. Optimization of the user distribution is performed based on the information contained in this model. After the complexity of the optimization problem is analyzed, algorithms solving it are defined and their performance is measured. At last a system architecture able to reflect the results of this optimization process is developed. It allows automatic provisioning of needed resources in the managed computing environments and reflects the optimized distribution of users by routing their requests to the assigned resource pools.

Document Structure

The document is divided into the following chapters.

Chapter 2 – Background and Related Work describes the used technologies and the current research state.

Chapter 3 – Resource Model defines an abstract model which is used to describe the entities part of the managed Software as a Service application.

Chapter 4 – Resource Usage Optimization analyses the complexity of optimizing the user distribution among multiple Computing Clouds and introduces algorithms performing it.

Chapter 5 – Performance of Optimization Algorithms verifies the performance of defined algorithms using several test cases.

Chapter 6 – Implementation describes a system architecture which utilizes the information obtained from optimization processes and handles provisioning and request routing.

Chapter 7 – Summary concludes the work by summarizing the results and gives an outlook on further improvements.

2 Background and Related Work

In oder to comprehend the challenges of optimization techniques introduced in Chapter 4, it is important to have some fundamental knowledge about the evolution of the utilized computing paradigms and technologies. This chapter also covers the economic opportunities arising from Software as a Service in greater detail and it describes the current state of used technologies.

2.1 Economic Aspects

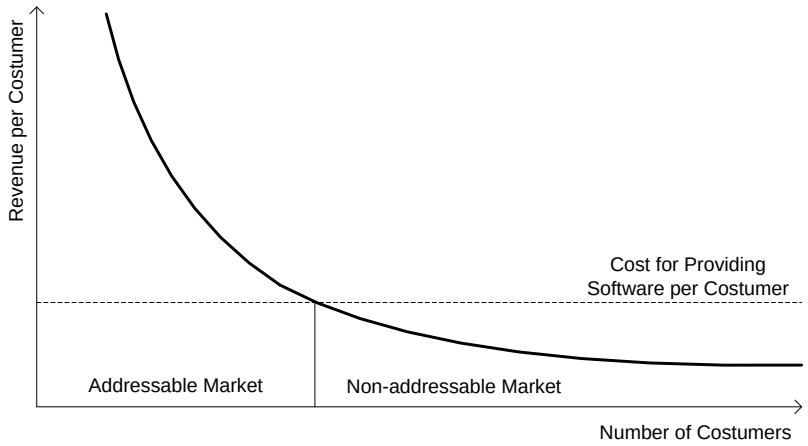

Figure 2.1: Power law distribution of user demands [CC06, p. 9]

Shifting the responsibility for the IT infrastructure from the user to the provider results in new market segments becoming economically seizable. Managing soft and hardware critical to a companies business is by far not a trivial task [ADC05]. While the downtime of a personal computer is annoying at best in a business environment this can lead to significantly greater damage. Having to run a complex IT infrastructure lead to companies moving away from their core competencies and is only profitable if the company has reached a critical size. Even though a small business would indeed benefit from using professional software the costs for management are often not justified.
Exploiting Software as a Service offerings affects this calculation significantly [CC06, p. 7]. Instead of just hosting hardware for a customer moving the management responsibility to providers, Software as a Service Applications offer a complete solution. And since these applications are tenant aware [CC06, p. 11] computing resources like databases are shared

between tenants. This reduces the needed revenue per costumer drastically.
The customizability is another important economical aspect of Software as a Service. The client is empowered to specify exactly what is needed and only pays for exactly these qualities. Additionally the structure of the application may be altered to support the client's business processes. Software matching the requirements of various tenants enlarges the target market for a Software as a Service application. This effect is commonly referred to as catching the long tail [CC06]. Demand for sold goods such as Software tends to follow a power law distribution as depicted in Figure 2.1.
One of the reasons why the on-line retailer Amazon.com is so successful is the fact that it may serve a user demand regular book stores cannot. Since it has virtually an endless amount of storage space available the pure amount of different books offered by Amazon.com is significantly higher than in regular book stores. While the revenue for best selling books is very high, according to Amazon.com most of its profit comes from the combined amount of less popular books being sold [CC06, p. 9]. The same opportunity lies within the Software as a Service business model. It does not only allow to provide applications to existing costumers at cheaper prices. It also targets new markets which could not be addressed before. Demands of costumers in these markets were too specific leading to the need for expensive customized software.
Another market served by Software as a Service is the one time demand of IT Infrastructure for a specific task. The cheapest virtual machine in Amazon's Computing Cloud costs about 72$ per month [Ama]. However for the same amount one can also buy 720 machines for one hour. This way everyone has access to computing power whenever needed. In 2007 the New York Times decided to make their old articles available as PDF files. The original articles were in the form of scanned images. Since a newspaper company hardly has the computing resources at hand for such a format transformation Amazon's Computing Cloud was used. Using 100 instances of virtual machines the 4 TB of image data for 11 million articles were transformed into PDF files in under 24 hours [Got07] and may now be searched and accessed in the New York Times article archive [New]. If the New York Times had to perform this task on their own IT infrastructure a lot more time and technical expertise would have been required.

2.2 Clouding of the Computer Industry

Cloud Computing allows that IT resources are moved out of corporate data centers to external providers and are accessed over the Internet at variable costs [RH08]. But it may also be incorporated into the design of private data centers. One therefore refers to them as public and private Clouds respectively. In the early ages of computer most programming was centered around mainframes. If one wanted more processing power a more powerful machine had to be bought. With the upcoming of cheap desktop computers the first computer clusters were born. Being a tightly integrated group of homogeneous machines a cluster was organized and set up to be perceived as one system. With the growing Internet distributed computing resources were interconnected and organized in computing grids. Instead of being a homogeneous federation of resources, as clusters are, those grids developed to be

very heterogeneous and highly distributed. Being often used for research purposes grids are resource and computing centered. Users are able to access resources for specific computing tasks for a certain amount of time.
Within the business world similar processes could be observed. Instead of hosting their own IT infrastructure locally specialized companies started to offer hosting of applications or physical servers. Cloud Computing shows several similarities to these traditional business models. The key aspects differentiating it are its dynamic, and the hiding of IT idiosyncrasies. This can be accomplished since Clouds are not resource centered but service centered. So instead of requesting concrete resources for a task Cloud users utilize services offered according to certain service level agreements. The amount of resources needed to fulfill these qualities is left inarticulate. The dynamic of Cloud Computing regarding the quick allocation and release of used resources was enabled by the evolution of hardware virtualization [DR07]. Instead of provisioning physical hardware every time a new system was needed virtual machines, merely emulating a full featured computer, are run on physical hardware. This allows for dynamic distribution of virtual machines among physical ones leading to better utilized data centers and reduced costs. Users may share computing resources of a physical machine without affecting each other.
A Computing Cloud combines these positive aspects of the evolution in virtualization as well as service orientation [Erl05, p. 34] to provide the flexibility and obfuscation of resource management. In general when computing functionality is accessed over the Internet the term

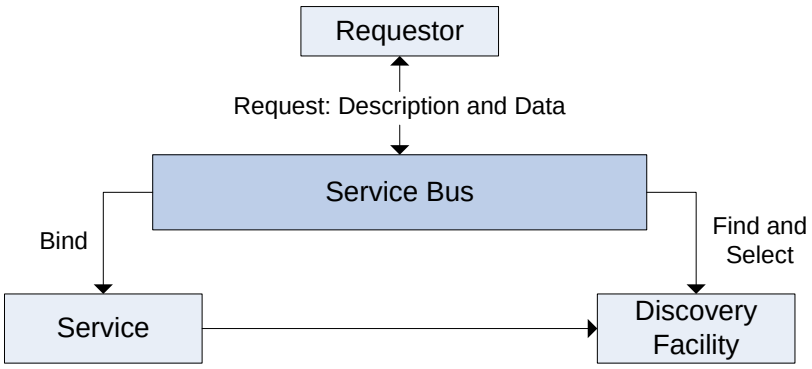

Figure 2.2: Role of the service bus in a SOA [WCL+05, p.20]

service is used often. It is regardless whether the resource offering the service is managed independently, in a Grid, or a Cloud and usually there is no way the user could tell. Access to services using common interface descriptions is the central idea behind a Service Oriented Architecture (SOA) [Erl05, p. 31 ff.]. The SOA Triangle depicted in Figure 2.2 shows the major players in such an environment. The service provider offers a service, describes its interface in a common language, and publishes this information to a discovery facility. The service user queries this discovery facility to find a service fulfilling his needs and binds to it. This as well as other mediating and decoupling functionality [Erl05, p. 34] is often realized by a special software, the Enterprise Service Bus (ESB). It is a distributed software often consisting of many Service Bus instances running at different locations. When accessing services different terminologies exist which are discussed next.

2.2.1 On Demand / Utility Computing

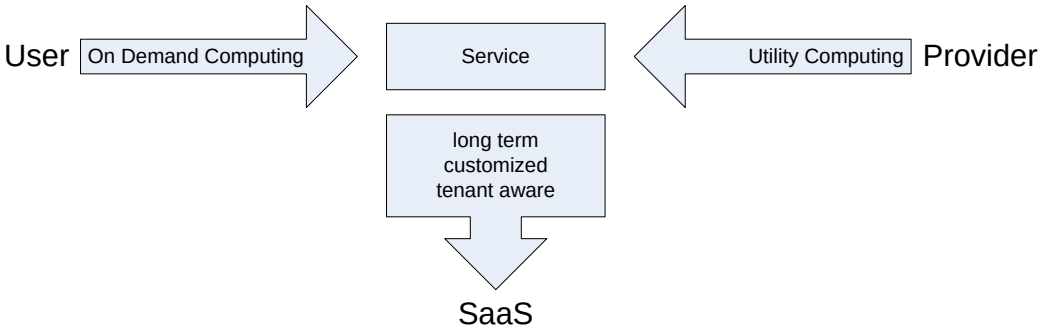

Figure 2.3: Views on services

Services offered by a provider are accessed on demand. Similar to utilities like water or power they guarantee an always on semantic. This functionality is referred to with two different terminologies depending on the point of view as depicted in Figure 2.3. Providers offer their services with similar qualities as traditional utilities so they refer to it as Utility Computing [Rap04]. The users on the other hand use them On Demand [HS05] meaning that all of their computing requirements are met as they arise. The concerns about performance and location of used resources is moved out of the scope of the user to the service provider.

2.2.2 Software as a Service (SaaS)

The client of a Software as a Service application is referred to as tenant. Tenants register a certain amount of users with the application. For example a tenant could be a company registering its employees as users for an application.
While Cloud Computing defines how resources are accessed the term Software as a Service is used to refer to additional characteristics of the used services. The central aspects being customizability and tenant awareness [CC06, p. 11]. Another main difference between it and On Demand Computing is the time for which a client binds to a service provider. Services are not subscribed to on a per request basis but for longer amount of times. Often a concrete software package is offered by the service provider which is customized for every client during the registration process.
A Software as a Service application being tenant aware means that multiple tenants may use one instance of the software but the software is represented to every tenant as if he were the only one accessing it. This quality is vital to the success of Software as a Service vendors since it allows tenants to share application instances reducing the overall costs. If other functionalities than software are offered using this paradigm more specific terms are used. They will be discussed next. The term Software as a Service will be used in general throughout this document.

Infrastructure as a Service (IaaS)

Services of this type show a lot of similarities to classic server hosting. But instead of requesting a physical server which is provisioned for every user a virtual machine is started somewhere on the providers infrastructure. The main difference is its dynamic since virtual machines may be requested and released within minutes. The user also often loses come control over the physical location at which the virtual machine is deployed.

Platform as a Service (PaaS)

Moving up in the infrastructure stack the provider offers an environment in which programs provided by the tenant are executed. Often concerns such a availability and scalability are handled by the service provider.

Composition as a Service (CaaS))

A very new concept is to allow restructuring of application internal processes. This allows the tenant to adjust the Software as a Service applications to support individual business processes. Processes running on the tenants own IT infrastructure may even be integrated. This results in composition of specified functionalities into a custom application and is referred to as Composition as a Service [LM09].

2.2.3 Layering the Computing Cloud

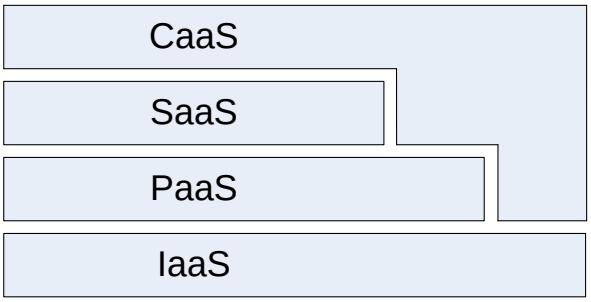

Figure 2.4: Layers of the Computing Cloud

Designing a Software as a Service application does not mean automatically that Cloud Computing is used. But since these applications profit greatly from a dynamic environment they have a high affinity towards Cloud Computing. One very powerful architecture is to model multiple layers of the system architecture as services. This results in empowering the tenant to choose services from all layers of the resource stack to integrate with its own applications or to combine them freely into a custom Software as a Service application. The resulting layers and their dependencies are depicted in Figure 2.4.

Utilizing such an architecture the Software as a Service vendor is able to not only sell an customizable application but also the services this application is build upon. This results in a very good utilization of data centers and a strong market position.

2.3 Alignment of Cloud Providers and the IT Infrastructure Stack

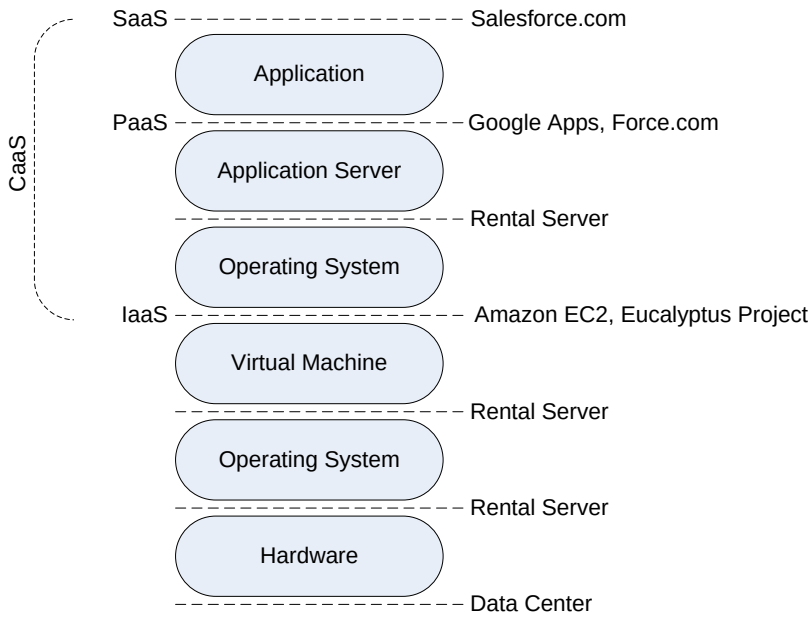

Figure 2.5: Exemplary IT infrastructure stack

Most IT infrastructures tend to be supported by similar or a subset of the resources depicted in Figure 2.5. All components above the respective lines are controlled by the user of the system. Therefore when running a local data center all system components are within the management scope of the user. On the left side the infrastructure stack is aligned to the corresponding layers of the Computing Cloud.
Next a couple of providers for the different service types are covered.

Rental Server

The first level of outsourcing [WCL+05, p. 8] would be to rent machines from another company. This step is done quite frequently since running a high available server is a lot more complicated than it seems [ADC05]. As seen in Figure 2.5 rental servers are offered on different levels of the infrastructure stack. In case of the lowest cut only the hardware would be rented. On higher levels the service provider would also install and maintain the operating system. Of course renting a managed operating system can also be done using virtual machines which is represented by the third cut labeled as rental server.

Amazon.com EC2 and the Eucalyptus Project

There are multiple service providers offering virtual machines. Amazon.com however is the first one offering its infrastructure as a service. As mentioned the user looses some control over the location where the virtual machine is running. In the so called Elastic Compute Cloud (EC2) [Muro8, p. 161 ff.] a virtual machine image is uploaded and a general region is selected where the machine will be deployed. Costs arise for the time the machine is running and the amount of data transferred [Ama]. Unlike many traditional hosting services there is no minimum fee.
The eucalyptus project [NWG+08] offers tools to realize a Computing Cloud in a private data center. It also allows to exchange virtual machines with EC2 [NWG+08, p. 2].

Google App Engine and Force.com

One exemplary provider for this level is the Google App Engine [Ciu09, p. 1]. It leaves even fewer management tasks to the user since it offers a complete running environment for Python scripts. Concerns about scalability and availability are moved out of scope of the user.
Another provider of Platform as a Service is Force.com [Sala]. It started out as a platform were users of the Software as a Service application of Salesforce.com [Salb] could develop custom plug-ins. But soon independent services were deployed exceeding the capability of plug-ins. They may also be integrated into processes completely independent of Salesforce.com.

Salesforce.com

At the Software as a Service Layer of the Resource Stack all IT infrastructure management concerns are taken away from the user. Salesforce.com offers such an application which may be used for Costumer Relationship Management [Salb].

2.4 Service Quality and Functionality

Services offered by Software as a Service vendors come with different service level agreements specifying quality of services. Those range from a certain encryption used to the availability guaranteed. Tenants may customize Software as a Service applications to display the qualities needed. When integrating services into their own applications additional concerns about matching interfaces and service protocol semantics arise as well. Several efforts were made to standardize the description of these characteristics. Considering the optimization and routing processes discussed in Chapter 4 it is important to know the functionality and limitations of the industry standards used.

2.4.1 Description of Service Interfaces

The Web Service Description Language (WSDL) [W3C01] evolved to be the de facto standard for describing service interfaces. "WSDL is an XML vocabulary to describe Web services. It allows service authors to provide crucial information about the service so that others can use it" [WCL+05, p. 40]. It does so by describing Web services on two distinct levels. The abstract level defines messages the service understands and the operations used to exchange them. The concrete level lets the user of the services know where the service resides and what communication protocol shall be used to interact with it. Such a concrete description is called a binding [W3C01]. A Web service may display several of such bindings representing different ways to access it.

2.4.2 Description of Service Policies

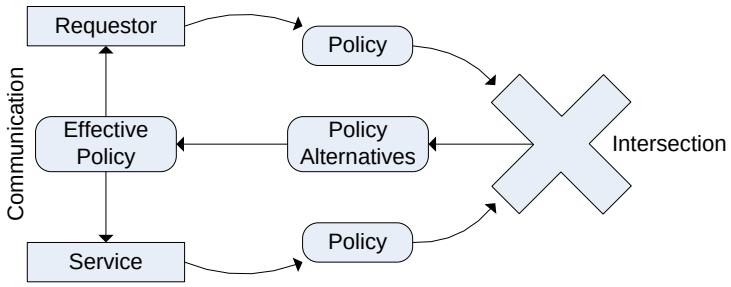

Figure 2.6: The process computing the effective policy [LK08]

WSDL has been designed with extensibility in mind. Therefore concepts describing characteristics of a service not covered by the standard may be added to the specification. Such an extension is WS-Policy [W3C07]. "It allows to define meta-data about services often related to data management, system configuration, or security areas by providing an interoperable, standardized representation of non-functional capabilities" [WCL+05, p. 127 - 128]. Policies are attached to Web service descriptions as well as to service discovery requests. During service discovery and binding the policies are intersected to find a set of policies acceptable for both parties. This set is called policy alternatives. From these alternatives one is selected as the effective policy. The process is shown in Figure 2.6.
For example a Web service could accept requestors demanding different levels of encryption. It would then specify all the encryption levels it offers as separate policies. The service requestor would do the same for the request. These policies are then intersected removing all characteristics not accepted by both communication parties. From the resulting alternatives one is selected as effective policy for the communication.

2.4.3 Description of Service Semantics

Additionally to a description of the service interface, semantics are very important. They describe the processes running behind the interfaces. The order in which these processes expect the exposed operations to be invoked is specified. During the development of WSDL semantic descriptors were still at research state [WCL+05, p. 106].
As part of another service description language, the Soap Service Description Language (SSDL) [PW05] [TDGS06], semantic descriptions may be formulated. A set of different specification languages is provided for this task ranging from automatons to the π-calculus. Another approach is to describe the interaction with a Web service as a business process [LR99, p. 32 ff.]. The Business Process Execution Language (BPEL) [OAS07] is a widely used standard to describe these processes. It may be attached to Web service definitions to specify message exchange patterns [NLv08].

2.4.4 Service Resolution

In theory policy and semantic descriptors may be used to find services matching requestors demands. Matching based on interfaces has been supported for a long time. However semantic matching efforts are still experimental. For policy matching a WS-Policy aware Service Bus is already coming forward [MvW+09]. However for the scope of this document it is important to realize that all matching processes search for equal specifications. There neither is a notion of better or worse policies nor of more or less powerful interfaces and semantics. But in general a service requestors would also accept more than what is needed. A requestor asking for a certain interface would also be satisfied if the service offered additional operations as long as the semantics are equivalent. Considering policies a requestor demanding an availability of 99.99% would also accept a service assuring one of 99.999%.
Until now it was usually not reasonable to provide superior services to requestors needing and paying only for weaker ones. But in a Software as a Service environment this approach could be used to ensure a better utilization of the overall system which would lead to cost reduction. This is covered in more detail in Chapter 4.

3 Resource Model

In order to fully exploit the optimization potential introduced by Software as a Service environments several tasks have to be taken care of. All system elements, their interaction with, and dependency on each other has to be represented by an abstract *Resource Model*. This ranges from hard and software resources to tenants signing up for the application. The collected information is used to perform optimization of the provisioned system resources to reduce overall running cost and to enable an automated provisioning process. After this optimization the model provides information to route tenant requests to the handling resources.
The algorithms utilized operate on different views of this Resource Model each showing only the necessary informations required for the current task. The view showing all system components is the *System Modeling View* (Section 3.1). From there a *Condensed View* (Section 3.2) is derived which mediates between the System Modeling View and the *Optimization View* (Section 3.3). It is also the view on which the user assignment necessary for routing will take place.
Information is partly redundant in this model. Depending on the implementation it might be efficient to remove this redundancy. The main characteristic affecting such a decision is the distribution of model information among IT systems. If all the information is concentrated in a single database parts of the model may be summarized. This approach has been chosen in Chapter 6.

3.1 System Modeling View

This view is the most intuitive representation because it describes every aspect of the system. The System Modeling View is also the most persistent one since it represents the current system state. It would be very unintuitive to model the other views manually and those are therefore derived only temporarily until the processes performed with them are completed.
The central element of this view and also of the following ones is the *Offering Group*. It is a set of resources offering functional equivalent services with identical quality of services. The granularity attribute specifies how many users one resource of this Offering Group may serve if run in a ideal environment. The actual number of users served by a resource may depend on other factors as described in Section 3.4.4. It is the only property which may be set for all Offering Groups. The others may only be set for *Atomic Offering Groups* discussed in Section 3.1.3.

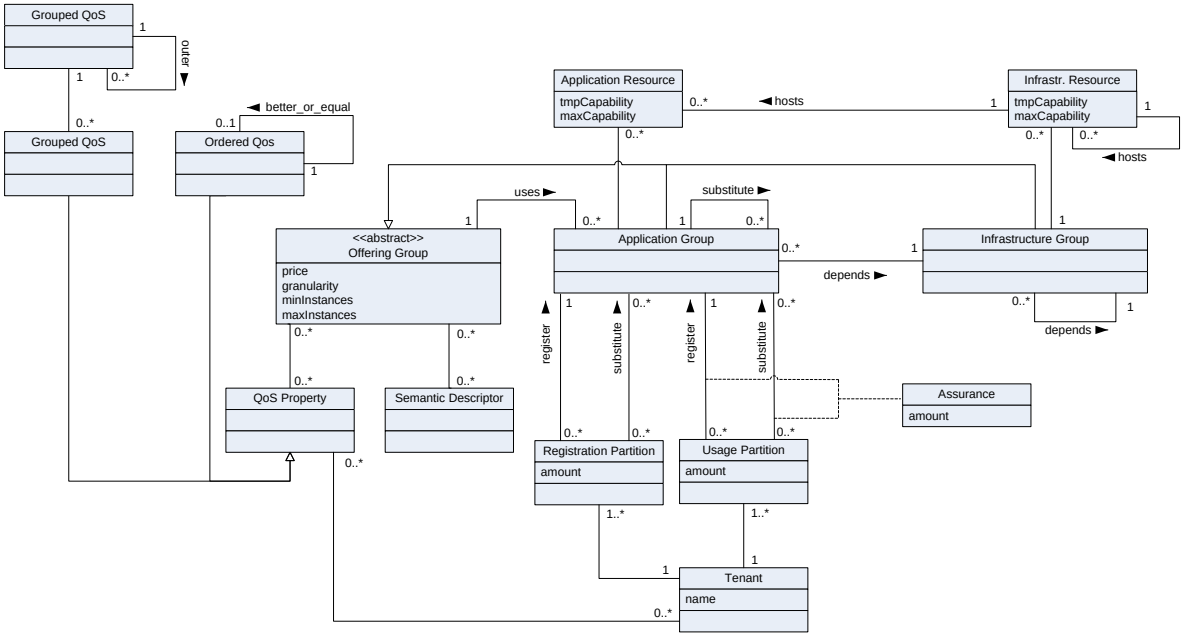

Figure 3.1: Components of the Resource Model on the System View Level

3.1.1 Application Groups

Application Groups summarize any piece of software offering a service. These services may then be used by tenants as well as other Application Groups. Consider an e-mail and an address book service for example. The e-mail service uses the address book service as part of its functionality and tenants may sign up for both.
Application Groups consist of *Application Resources* which are instances of a specific piece of software. These resources have a maximal and a temporary capacity. The maximal value states how many concurrent users can be served by this resource in an ideal environment and is therefore equivalent to the granularity. This number is always larger or equal to the number for the temporary capacity which states how many users this instance can handle with the computing power provided by the underlying *Infrastructure Group*.

3.1.2 Infrastructure Groups

These groups consist of system components on which Application Resources are deployed. For example this could be an application server. Only Application Groups are accessible by tenants. If a tenant shall have access to application servers in an Infrastructure Group at least one Application Group would have to be defined containing the services needed to access the application server's functionality. This could be a Web service allowing the tenant to upload code which shall be executed. Instead of using the resources of an Infrastructure Group the dependent resources are hosted on them. Those hosted resources may be part of Application as well as Infrastructure Groups. In the example given the application servers

might be hosted on virtual machines part of another Infrastructure Group. While Application and Infrastructure Groups may use multiple Application Groups both may have only one Infrastructure Group on which they depend.
The scenario in which Infrastructure Groups use services of Application Groups may not be that obvious. An example could be a logging service to which application servers send information.

3.1.3 Atomic Offering Groups

At the end of the dependency hierarchy there is always one Offering Group which does not depend on any Infrastructure Group. This group is called an Atomic Offering Group. Instances of this Offering Group are directly affecting the running cost. Limiting resource numbers of non atomic groups is contra productive to the optimization efforts as pointed out in Section 3.4.2.
In most scenarios the Atomic Offering Groups will be Infrastructure Groups but in some cases there may be Application Groups which do not depend on any Infrastructure Groups. This would be the case if Platform as a Service providers are used. Therefore in general the term Atomic Offering Group will be used.
Only Atomic Offering Groups have values set for the other properties additional to the granularity. The price property specifies the costs for one resource of this Atomic Offering Group. An Atomic Offering Group may also have an upper and a lower limit on present resources. They are used to regulate the automated provisioning process so that the minimum of resources is always provisioned and the maximum is never exceeded.
This is necessary since some Offering Groups might be cheap but not be very dynamic. Resources in a local data center are likely to display these characteristics. While they might be cheaper than external Cloud services provisioning them takes much longer. However if the load in a local data center is just exceeded a couple of days per month or during a certain time of the year, utilizing more expensive Cloud services is likely to be preferred. Lower and upper boundaries are needed to take these characteristics of Offering Groups into account while optimizing the user distribution.

3.1.4 Tenants

As introduced in Section 2.2.2 it shall be differentiated between tenants and users. A tenant is a party, often a company, requesting usage rights for a number of users. The tenants part of the Resource Model own *Usage* and *Registration Partitions* for at least one Application Group. A Usage Partition represents the amount of users a tenant has registered for an Application Group. They also identify the Application Groups into which the users may be substituted. These substitutions are a subset of the substitution links (Section 3.1.7) displayed by the Application Group registered for.
Registration Partitions have similar semantics as Usage Partitions. They are needed to differentiate between users which have already been provisioned in the system at least once and those a tenant has just requested. This is needed for request based optimization as

described in Section 4.6. During this process Registration Partitions are transformed into Usage Partitions.
Within the scope of the following sections only Usage Partitions are considered. It is shown afterwards how Registration Partitions are used.

3.1.5 Semantic Descriptors

A semantic descriptor specifies the interaction protocol supported by services part of an Application Group. It may be used to identify other Application Groups which may serve as substitutes as described in Section 3.1.7. Possible languages describing the semantics have been introduced in Section 2.4.3.

3.1.6 Quality of Services

To automatically compare quality of services some kind of ordering is needed and those qualities which cannot be ordered still have to be organized in such a way that enables comparison. Starting from the initially requested Offering Groups possible substitutions may then be computed automatically. The model introduced now does this by differentiating between *Ordered* and *Grouped Quality of Services*. Ordered Qualities are of ordinal scale [Ste46] and thus allow a partial ordering. Grouped Qualities are of nominal scale [Ste46] but still a way to compare them is introduced. A formal definition of this model and a formal algorithm using it to find possible substitutes is given in Section 4.1.1.

Ordered Quality of Services

An example of this type would be the availability of a system. A system having an availability of 99.9% is better than a system only having an availability of 99.0%. This relation is expressed using the better than link in the Resource Model. These links may be flat or hierarchical as also described for substitution links in Section 3.1.7.

Grouped Quality of Services

Not all quality of services can be ordered. A good example would be the position of systems. One might be located in Stuttgart and another one in London. It is unclear which one is better but tenants might be required to ensure that used systems reside in Germany. Since no order can be given according to capacities those quality of services will be assigned to *Quality of Service Groups*. Such a group may again be part of other Quality Groups. Considering the example Stuttgart and London would be included in the Quality Groups "Germany" and "United Kingdom" respectively. Those groups would then be included in the Quality Group "Europe" as shown in Figure 3.2.

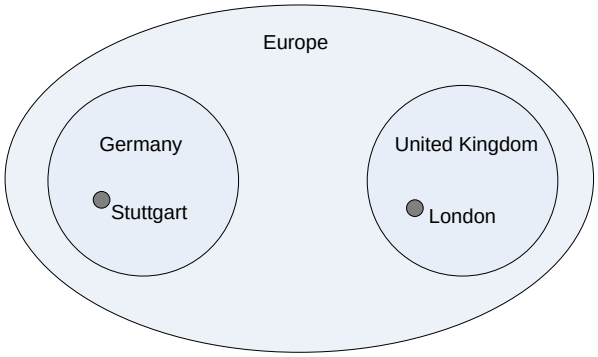

Figure 3.2: Exemplary Quality Groups for system location

Inheriting Quality of Services

Quality of services are inherited upwards the dependency hierarchy. Additional quality of services may be added but none removed or improved. Considering a usage relationship between Application Groups it has to be verified carefully if the used groups will affect the using group.
For example consider an information service which publishes monitoring data to an internal application. This information service might be quite unreliable but also not very important. It should not influence the quality of services of the Offering Groups monitored.

3.1.7 Substitution Links

A pressing question is how substitution links are created during the design time. Application Groups may exhibit multiple substitution links which play a major role in optimization efforts. They are used to indicate to what other Application Groups users may be delegated. Therefore these links point to Application Groups offering identical or superseding functionality and quality of service. For example an Application Group may provide a high priority e-mail service. Resources in this Application Group are provisioned but show a bad utilization. Users requesting another Offering Group with low-priority e-mail services could now be delegated into the high-priority Application Group to reduce the number of resources needed in the low-priority Group which would reduce the cost of the overall system.
In general substitution links can either be flat or hierarchical as depicted in Figure 3.3 and 3.4. In the case of hierarchical identification of possible substitutions, algorithms using the model might become more complicated. However an algorithm flattening the substitution links is very simple. So for ease of modeling hierarchical substitution links are allowed but they are flattened before any other of the described algorithms is run.
Substitution links may reside between Application Groups as well as between Usage and Registration Partitions and their corresponding Application Group. Intuitively it seems redundant since all substitution links of the Application Group registered for could be copied to the Usage or Registration Partition. However a substitution link between Applications

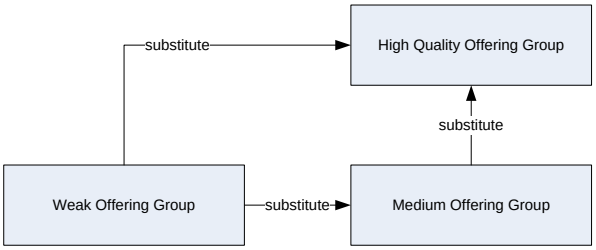

Figure 3.3: Flat substitutions

Figure 3.4: Hierarchical substitutions

Groups only indicates hat the Application Group substituted to has equal or extended semantics and better or equal Ordered Quality of Services. Grouped Quality of Services may lead to tenants not accepting the Application Group a substitution link references.
In order to create any substitution link automatically it would be necessary to algorithmically compare the semantics and quality of services of Application Groups. However it was pointed out in Section 2.4 that semantic comparison is still at research stage and that comparison of quality of services is not powerful enough since it only finds equal quality of services. For the creation of substitution links this is insufficient. Ordering and grouping of quality of services has been introduced conceptually to allow the generation of substitution links.
A formal specification of the method determining the set of possible substitution groups is given in Section 4.1.1. If the evaluation of quality of services and semantical descriptors were fully automated substitution links between Application Groups could be omitted totally. Links could be calculated completely during the registration process.

3.1.8 Assurances

For every registration and substitution link there is an Assurance. It stores the amount of users assigned to the Application Group referenced by the corresponding link. Assurances are used to route tenant requests to the handling Applications Groups according to the user distribution contained in the set of Assurances of a tenant's Usage Partitions.

3.2 Condensed View

The Modeling View contains a lot more information than what is needed for optimization. Filtering this information during the runtime of algorithms would result in a cluttered definition and therefore error prone code. It would also result in a reduced performance

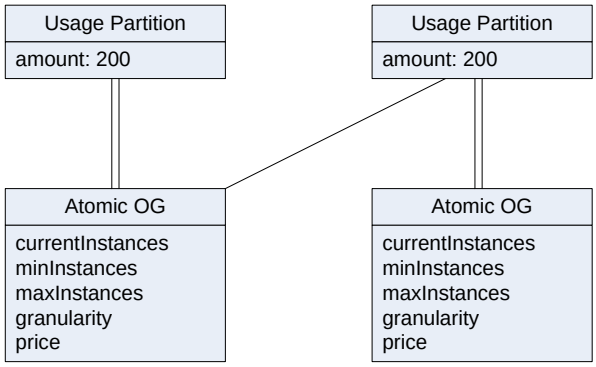

Figure 3.5: Condensed View of the Resource Model

since additional operations would have to take place in each iteration of the algorithm. Since only Atomic Offering Groups affect the price of the system the Condensed View focuses on these groups and their users. It forms the basis for the *User Distribution algorithm* introduced in Section 4.5. It also facilitates the transition between the System Modeling View and the following Optimization View. These transformations are called deflation and inflation of the Resource Model and are discussed in more detail in Section 3.4.3 and 3.4.4.
In the Condensed View the registration links may be represented by a double line and the substitution links by single lines. Resources are not modeled directly anymore but are now only a property of the Offering Groups.

3.3 Optimization View

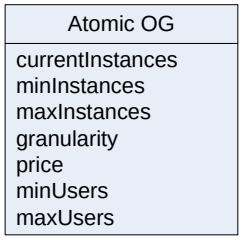

Figure 3.6: Optimization View for an Atomic Offering Group

The Condensed View still contains a notion of different tenants or at least their Usage Partitions. This is needed to distribute users and create routing rules for every tenant. But the optimal number of resources is not affected by the number of Usage Partitions only by the number of users per Offering Group.
The algorithms introduced in Section 4.2 therefore operate on an even simpler model which only consists of Offering Groups with properties for the minimal and maximal allowed users and resource instances. This model emerges from the Condensed View using a simple process. The minimal amount of users is the sum of all users in Usage Partitions registering

for an Offering Group having no substitution links. The maximal amount is the sum of all users in Usage Partitions linked to an Offering Group regardless whether by registration or substitution link.

$$\text{minUsers} = \sum \text{users in Usage Partitions registered having no substitution links}$$
$$\text{maxUsers} = \sum \text{users in Usage Partitions connected by registration or substitution link}$$

The minimal and maximal values of users are separate from the minimal and maximal number of instances and not linked semantically.
This model facilitates the definition of very efficient algorithms. Since it loses the notion of tenants and their users completely the algorithms will return an optimal number of instances per Offering Group. The User Distribution algorithm uses the Condensed View to distribute users among those instances to enable routing.

3.4 Using the Resource Model

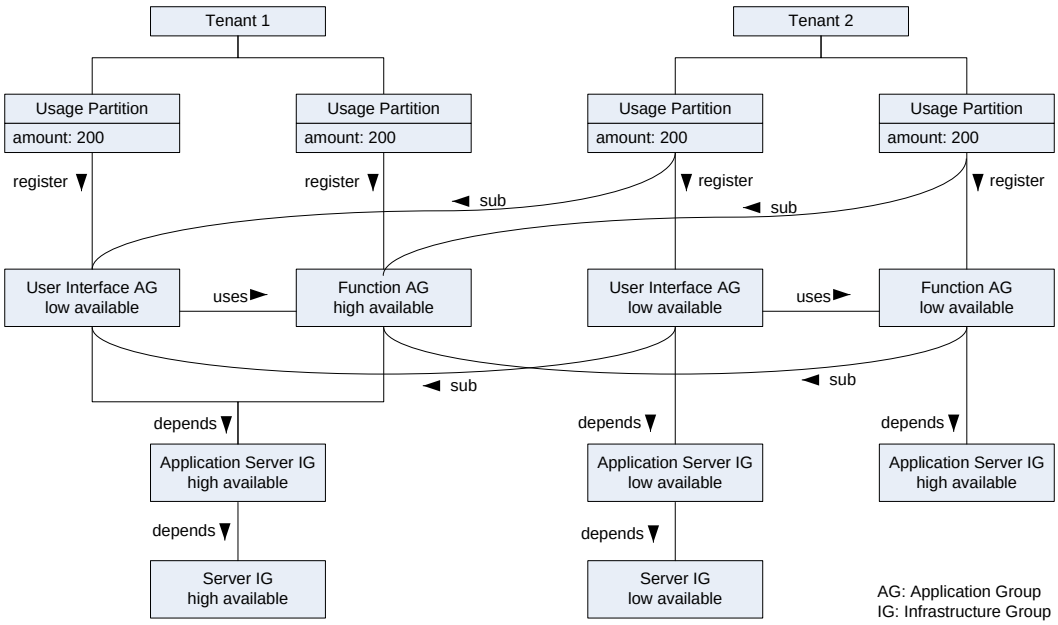

Figure 3.7: An exemplary setup of the Resource Model

The processes deflating and inflating the Resource Model are best explained using a concrete scenario. Consider an exemplary setup as depicted in Figure 3.7. The concrete modeling of quality of services, semantic descriptors, granularities, and prices has been omitted. The application consists of two functionally different Application Groups offering a user interface and some other functions with the later being used by the former one. Both service types are offered with two different quality of services. The high available services could be considered to run in a local, high available data center. Whereas the services with low availability are run partly on less powerfull local hardware and partly at an off-site application service

provider.
Furthermore there are two tenants registered with the application. One requesting the high available and the other requesting the low available services. Each Usage Partition identifies possible substitutions. The quality of services required for Usage Partitions of the tenant registering for the low available Application Groups are met by the high available ones. Therefore substitution links are created for both Usage Partitions of that tenant.

3.4.1 Tenant Registration Process

When a tenant signs up a set of initial Application Groups is selected and Usage Partitions are created for them. A tenant registering for a service like a whole e-mail system is likely to register for one Application Group only. A tenant wanting to integrate services of the provider into its own infrastructure is likely to sign up for several Application Groups whose services will then be orchestrated by the tenant. Another possibility is that a tenant wants to split up Usage Partitions to exploit locality of service in case his users are distributed over a large area or some other regulations require him to do so.
Starting from this initial request additional Usage Partitions are created automatically. Following the given example:

1. The second tenant chooses to register for the User Interface Service.
2. Since the User Interface Application Group identifies one possible Substitution Group, it is verified that the quality of service of that group also meets the tenant's demands. If so it is added as substitute for the concrete Usage Partition.
3. Next the usage links are resolved. Since the User Interface Application Group uses the Function Application Group a second Usage Partition is generated for that group.
4. Again the possible substitutions are evaluated.

After this process a tenant owns Usage Partitions for every Application Group which services are needed to serve his requests. The tenant himself only chooses a small subset of those groups and specifies required quality of services. The whole amount of Application Groups needed may then be calculated.
However quality of service and semantic comparison is experimental so currently the tenant is likely to be involved in this process identifying possible substitutions. The prototype implementation discussed in Chapter 6 assumes that a tenant will specify all accepted Application Groups or that they are computed at some earlier point during the registration process. How this computation might be conducted is defined formally in Section 4.1.1.

3.4.2 A naive Optimization Attempt

Consider a new example depicted in Figure 3.8. One naive attempt to optimize is to minimize instances of all Application Groups and therefore minimizing the required instances of underlying Infrastructure Groups. However due to the fact that Application Resources

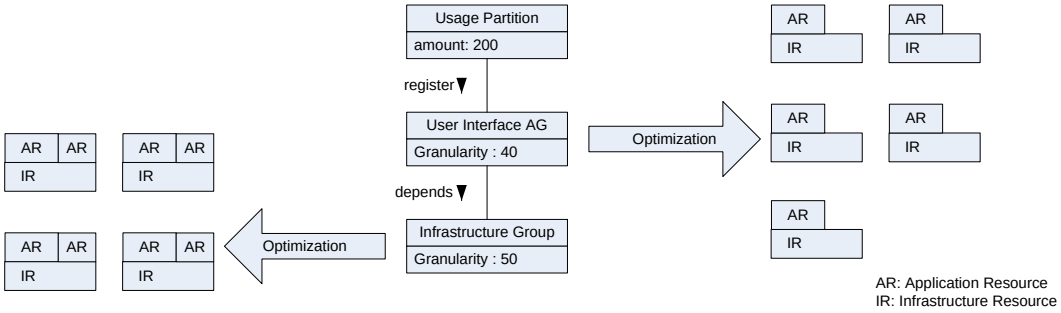

Figure 3.8: A naive optimization attempt and resulting number of instances

do not directly generate costs there can be circumstances under which this approach is unsuccessful.
Minimizing the number of provisioned Application Resources would result in the decision that five of them shall be provisioned to serve 200 users. To provision those five instances and grant each one of them the full capability to serve 40 concurrent users would also require five Infrastructure Resources. But four Infrstructure Resources with two Application Instances provisioned to each would be sufficient. The criterion to optimize is therefore to have a minimum of instances which actually affect costs and to utilize them to their full potential. These instances are part of Atomic Offering Groups which is why all others will be masked when deflating the model into the Condensed View.

3.4.3 Deflating the Resource Model

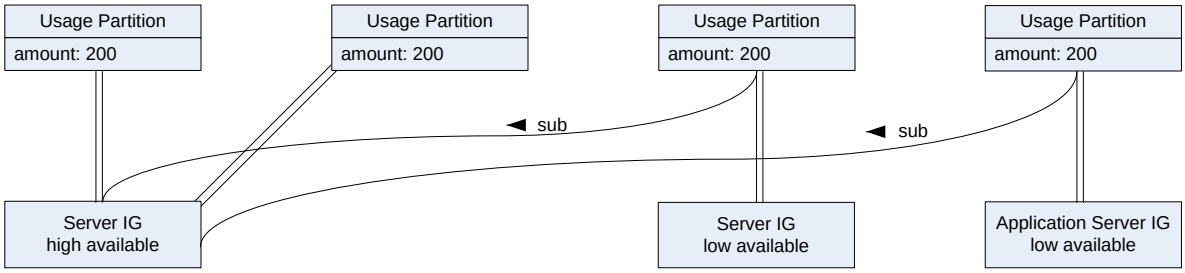

Figure 3.9: Condensed View of the exemplary Resource setup

The Resource Model is deflated by following the registration and substitution links of a Usage Partition and then moving down the dependency hierarchy until an Atomic Offering Group is found. All other Offering Groups visited are masked. This results in the Condensed View depicted in 3.9 for the example given in Figure 3.7. From there the Optimization View is computed.
The algorithms introduced in Section 4.2 operate on this Optimization View shown in Figure 3.10. They will specify how many instances are required in each Atomic Offering Group. The User Distribution algorithm will then assign parts of the Usage Partitions to concrete registration and substitution links resulting in the set of Assurances.

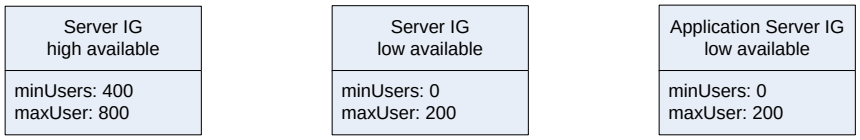

Figure 3.10: Optimization View of the exemplary Resource setup

3.4.4 Inflating the Resource Model

In order to use the optimization results for load balancing the model has to be inflated again since routing is performed on the Application Group level. Through unmasking all Offering Groups this information is already available.
Now resources have to be provisioned respecting the Infrastructure Resources they are hosted on. A simple process very similar to the User Distribution algorithm defined in Section 4.5 solves this assignment: An iteration over all instances of the Atomic Offering Groups is performed provisioning resources of the depending Offering Group while assuring them as large capabilities as possible. The order in which depending Offering Groups are provisioned does not affect the result. Furthermore since the number of instances in Atomic Offering Groups is optimal all Offering Groups are guaranteed to end up with sufficiently powerful resources. According to the ratios of those capabilities the *Load Balancer* (Section 6.1.1) may now route requests to them.
It has been seen that the naive optimization attempt allowing all Application Resources to run at full capacity was unsuccessful. Having Application Resources with fewer temporary capacities allows them to be better distributed among underlying Infrastructure Resources.

4 Resource Usage Optimization

It has been covered how different views of the Resource Model are created and how the concrete number of resources may be calculated and provisioned. The following algorithms optimize this number of Atomic Resources. Two optimization goals are differentiated.
Global optimization functions as if all tenants had just signed up to the system and no resources have been provisioned jet. Request based optimization respects that a system grows over a period of time. Therefore it leaves parts of an optimized system untouched only optimizing the distribution of a new tenant's users. At first the problem which has to be solved by optimization algorithms is formally defined as the *Festival Problem*. After the complexity of this problem is analyzed several optimization algorithms are specified and compared to each other. It is pointed out how the algorithms are used for global and request based optimization. At the end of this chapter several unsuccessful optimization attempts are described.

4.1 The Festival Problem

The Resource Model along with its transformations would result in a complex formal definition moving the focus away from what is needed for complexity analysis. The Festival Problem shows similar challenges while using a more general terminology omitting the idiosyncrasies of computer resources and their users and allowing a more general discussion.
Consider the organization of a music festival. There are camping grounds for attendees to sleep at. A special service shall be offered so that groups of campers may bring their own beds and sleep in tents put up for them. Tents come in multiple types which differ by the amount of campers they can occupy, their price, and other characteristics. Neglecting the wish of camper groups to sleep together, groups may be split up between different tents. Since one physical tent may host multiple campers tents may not be fully utilized if there are not enough camper groups signing up for this specific tent type. So the optimization potential is equal to that of the Resource Model and aligns well with what has to be solved for the Condensed View. Similar problems could be defined for distributing travelers among buses or planes. The significant difference between this problem and other distribution problems like Bin Packing [SLBC05, p. 34] is that the items to be distributed, the camper groups, may be split up into single units.
For a formal definition of the Festival Problem consider the set of tent types (Tt):

$$Tt = \{t_1, t_2, \ldots, t_\tau\}$$

Every tent type has additional characteristics. One is the number of campers a tent may host, the capacity (κ).

$$\kappa : Tt \rightarrow \mathbb{N}$$

Additionally every tent type has a price (p) which is the amount it costs to put up one tent.

$$p : Tt \rightarrow \mathbb{R}^{+}$$
$$p(t_i) = \text{price for one tent of this type}$$

In order to be closer to the problem of optimizing the Resource Model let there be a minimal availability (min) for a tent type. This is the number of tents which are put up always. In the scope of the festival problem this could be because the tent building starts while campers are still signing up.

$$min : Tt \rightarrow \mathbb{N}$$

The same holds for a maximum availability (max), the maximal number of tents which may be put up.

$$max : Tt \rightarrow \mathbb{N} \cup \{\infty\}$$
$$max \geq min$$

Let there be camper groups (C) characterized by a distinct identifier (i). They request a certain number of beds (b) for a specific tent type and they would also accept a set of substitution tent types (CT).

$$C_i = \{i, b_i, t_i, CT_i\}$$
$$CT_i = \{t_{i1}, t_{i2}, \ldots, t_{in}\}$$

The algorithm optimizing this scenario has to find a correlation (g) which assigns every bed to a tent type for a specified set of camper groups $C_1, C_2, \ldots, C_m$.

$$b_{ij} : \text{bed number } j \text{ of } C_i$$
$$g : b_{ij} \rightarrow t_k$$
$$g(b_{ij}) = \text{tent type assigned to } b_{ij}$$

Now a function can be defined to calculate the number of tents required to serve the users assigned by this correlation.

$$required : Tt \rightarrow \mathbb{N}$$
$$required(t_i) = \left\lceil \frac{\left| \{b_{ij} | g(b_{ij}) = t_i\} \right|}{\kappa(t_i)} \right\rceil$$

Furthermore the following condition must hold for the resulting correlation.

1. Acceptance Check: The assigned tent type must be accepted by the assigned camper.

$$\forall b_{ij} : g(b_{ij}) = t_i \vee g(b_{ij}) \in CT_i$$

2. Availability Check: It must be possible to put up enough tents of every type.

$$\forall t_i \in Tt : required(t_i) \leq max(t_i)$$

3. Cost Optimization: Utilizing a function (*cost*) calculating the cost for every tent type the overall costs (G) shall be minimized with respect to g.

$$cost : Tt \rightarrow \mathbb{R}^+$$
$$cost(t_i) = \begin{cases} required(t_i) \cdot p(t_i) & \text{if } required(t_i) > min(t_i) \\ undefined & \text{if } required(t_i) > max(t_i) \\ min(t_i) \cdot p(t_i) & \text{else} \end{cases}$$
$$G = \sum_{t_i \in Tt} cost(t_i) \text{ is minimal with respect to g}$$

Notice that within the scope of the availability check the minimal availability has not been considered. This is due to the fact that campers assigned to a tent type do not have to fully utilize the minimal put up tents. It is just not very efficient since even if the minimum amount is not utilized it still has to be paid for. This is reflected in the cases of the cost function.

4.1.1 Utilizing Quality of Services

Until now every camper group is required to specify a list of tent types which are also accepted. Not only for the Festival Problem this seems cumbersome. Also in Software as a Service environments tenants should not have to manually specify all Application Groups to which their users may be delegated. The Resource Model already introduced quality of services and it has been shown that substitution links alone are not sufficient to delegate users. This is why quality of services are specified individually for every tenant and have to be evaluated during the registration process.
Within the scope of the Festival Problem it is now shown formally how Ordered and Grouped Quality of Services can be used to calculate the set of also accepted tent types (CT_i). To achieve this let there be a set of qualities (Q) describing the features of tent types.

$$Q = \{q_1, q_2, \ldots, q_\pi\}$$

A subset of those qualities is assigned to every tent type through the function Ψ.

$$\Psi : Tt \rightarrow Q$$
$$\Psi(t_i) = \{q_{i1}, q_{i2}, \ldots, q_{im}\}$$

Just like the quality of services defined for the Resource Model, Q consists of Ordered Qualities (OQ) and Grouped Qualities (GQ).

$$Q = OQ \cup GQ$$

Also as in the Resource Model the set of Ordered Qualities is ordered partially and Grouped Qualities are an element in a Quality Group (QG) which again may be part of another Quality Groups.

$$GQ = \{QG_1, QG_2, \ldots, QG_h\}$$
$$QG \subseteq GQ$$
$$QG \subseteq QG'$$
$$\forall q_i \in GQ : q_i \in QG_j$$

The list of tent types a camper group also accepts a may now be obtained from the quality of services of the initially requested type. At first a camper group only specifies this initial tent type.

$$C_i = \{i, b_i, t_i, \{\}\}$$

The initial tent type has grouped and Ordered Qualities associated with it.

$$\Psi(t_i) = GQ_i \cup OQ_i$$
$$GQ_i \subseteq GQ$$
$$OQ_i \subseteq OQ$$

This set of qualities is now extended to obtain the set of all qualities accepted by C_i which shall be called CQ_i. The extension is performed using two functions (σ and ν).

$$\sigma : OQ_i \rightarrow OQ_i'$$
$$OQ_i' = \{q_i | q_i \in OQ_i \vee q_j \geq q_i,\ q_j \in OQ,\ q_i \in OQ_i\}$$

$$\nu : GQ_i \rightarrow GQ_i'$$
$$GQ_i' = \{q_i | q_i \in GQ_i \vee q_i \in GQ_j,\ GQ_i \subseteq GQ_j\}$$
$$CQ_i = GQ_i' \cup OQ_i'$$

This set of accepted qualities may now be used to find the set of also accepted tent types for C_i.

$$CT_i = \{t_j | \Psi(t_j) \subseteq CQ_i\} \setminus t_i$$

4.1.2 Complexity Analysis

Before algorithms solving the Festival Problem are introduced in Section 4.2 it is shown that the problem at hand is np-hard. Therefore no algorithm exists always finding the optimal solution in polynomial time unless $p = np$ [Puno8, p. 11-11]. Quite commonly this is shown by reducing a problem to another one which is already known to be np-hard. Therefore a special case of the Festival Problem which is much simpler than the problem itself is transformed into a representation into which the Knapsack Problem [Man89, p. 327] known

to be np-hard may also be transformed.
Let there be only one camper group which accepts all tent types present.

$$C = \{b, t, CT\}$$
$$CT = \{t_1, t_2, \ldots, t_\tau\}$$
$$\forall t_i \in Tt : t_i \in CT$$

Availabilities also introduce further complexity to the Festival Problem so that for complexity analysis the minimal availability shall be zero. And all tent types shall be available infinitively.

$$\forall t_i \in Tt : min(t_i) = 0$$
$$\forall t_i \in Tt : max(t_i) = \infty$$

A notion of the physical tents (it) being put up is also needed. Let there be a set (T) containing all physical tents. All of the physical tents can be correlated with a tent type using the function (Γ).

$$T = \{it_1, it_2, it_3, \ldots, it_\mu\}$$
$$\Gamma : T \rightarrow Tt$$
$$\Gamma(it_i) = \text{Tent type of this physical tent.}$$

Therefore an algorithm solving the Festival Problem would have to find the amount of physical tents for which the cost is minimal while still providing enough places to sleep.

$$T_{minimal} = \{it_1, it_2, \ldots, it_r\}$$
$$\sum_{j=1}^{r} p(\Gamma(it_j)) \text{ is minimal}$$
$$\text{with respect to } \sum_{j=1}^{r} \kappa(\Gamma(it_j)) \geq b$$

The Knapsack Problem will be transformed into an equivalent form. It optimizes the packing of a knapsack having a certain carrying capability (G). Also there is a set of n items each of which is characterized by two natural numbers g_i and w_i specifying its weight and worth respectively. Now an optimal subset (J) of those items is searched which maximizes the carried worth while still fitting into the knapsack. Considering the camper group as being a knapsack which has to be "filled" with tents those two problems are analogous. For simplicity let the amount of available tents be finite but always sufficient which will not affect the generality of this proof.
The original definition of the Knapsack Problem is:

$$J = \{i_1, i_2, \ldots, i_r\}$$
$$\sum_{j=1}^{r} w_{ij} \text{ is maximal}$$
$$\text{with respect to } \sum_{j=1}^{r} g_{ij} \leq G$$

Already this seems similar to the Festival Problem with the difference that the Knapsack Problem aims to maximize the worth while the Festival Problem minimizes the cost. Now let K be the complementary subset of J:

$$K = \{i_i, i_2, \ldots, i_n\} - J$$

The weight of all available items shall be A which is the sum of weights in J added to the sum of weights in K.

$$A = \underbrace{\sum_{j=1}^{r} g_{ij}}_{\text{Weights in } J} + \underbrace{\sum_{k=1}^{n-r} g_{ik}}_{\text{Weights in } K}$$

Originally the condition was that the weight of all items in the knapsack (J) shall not extend the carrying capability. Equivalently one may say that the weight of all items not in the knapsack (K) shall be greater than the overall weight subtracted by the carrying capability of the knapsack:

$$\sum_{j=1}^{r} g_{ij} \leq G \equiv \sum_{k=1}^{n-r} g_{ik} \geq A - G$$

Analogously the maximization can be formulated as a minimization since maximizing the worth in J is equivalent to minimizing the worth in K. Therefore the general definition of the Knapsack Problem could also be stated as follows without affecting its complexity.

$$\sum_{j=1}^{r} w_{ij} \text{ is minimal}$$

$$\text{with respect to } \sum_{j=1}^{r} g_{ij} \geq G$$

This representation is equivalent to the special case of the Festival Problem which is therefore reduced to the Knapsack Problem. Since this proves that the Festival Problem is np-hard local search algorithms will now be defined searching for an optimal user distribution for a Condensed View of the Resource Model.

4.2 Local Search Methods

When performing a local search [Noro3, p. 110] one operates on a multidimensional search scope of elements forming the valid solutions to a problem. Elements are arranged in this search scope according to their properties. An intuitive search scope considering a number of users distributed over three Offering Groups is depicted in Figure 4.1. Section 4.2.2 however shows that the following algorithms have to use a slightly different search scope. Offering Groups have one characteristic which is the number of users assigned to it. The set of elements surrounding a specific element are called neighborhood. In case of this search scope the size of such a neighborhood is fixed and it is therefore called discrete. The opposite

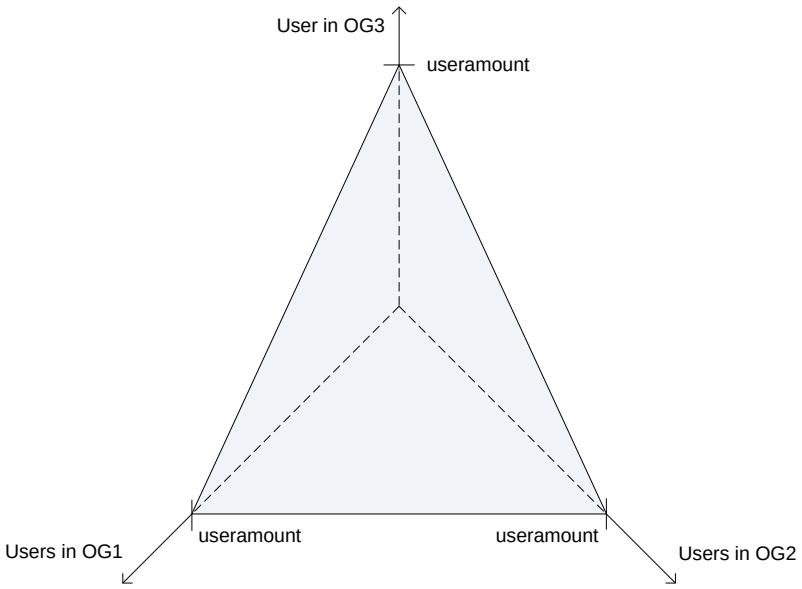

Figure 4.1: An intuitive search scope using the user distribution

would be a continuous search scope in which every element would have an infinite number of neighbors [Noro3, p. 42]. This would also be the case here if fractions of users could be assigned to Offering Groups.
In order to make elements of this search scope comparable a fitness function is defined. The goal of a local search algorithms is then to find an element with a minimal or maximal value depending on the nature of this fitness function. This is done by starting at one element in the search scope and iteratively selecting neighbors according to selection strategies described in Section 4.2.3. The alternative would be to calculate the fitness value for every element of the search scope which is only feasible for small scenarios. An algorithm using this approach is defined in Section 5.1 and will be used to evaluate the performance of the following local search algorithms.

4.2.1 Fitness Function

The fitness function has to assign a value to the elements of the search scope making them comparable. In case of the Festival Problem the only property which has to be optimized is the overall cost. In other environments the elements of the search scope often have different characteristics which would have to be weighted but this is not the case here. It is covered later how this simplicity of the search scope may be exploited to smartly move around in it. The cost function incorporates the cost for an Offering Group as well as its utilization. This is the main advantage over greedy approaches discussed in Section 4.7.1 which may only focus on optimizing cost or utilization. Since the local search methods consider the overall cost the optimization of Offering Group cost and their utilization is performed implicitly. Consider the simple scenario depicted in Figure 4.2. It is comprised of two Atomic Offering Groups

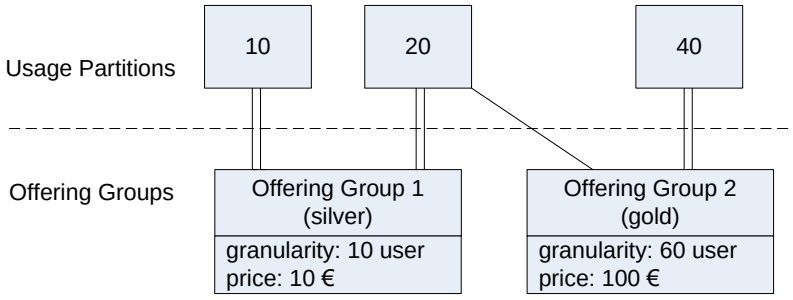

Figure 4.2: An examplary scenario of Tenants registering for two Offering Groups.

providing lesser and better quality of services. This results in users of tenants requesting the silver Offering Group being delegatable into the gold Offering Group. Only the Usage Partition containing 20 users actually allows this while the other one containing 10 users demands provisioning into the silver Offering Group. It is perceived easily that in this scenario all of the 20 users should be delegated into the gold Offering Group since the 40 users requesting its service leave a free slot for exactly 20 users. The resulting cost function respecting all possible user distributions for this scenario is shown in Figure 4.3. It is a step

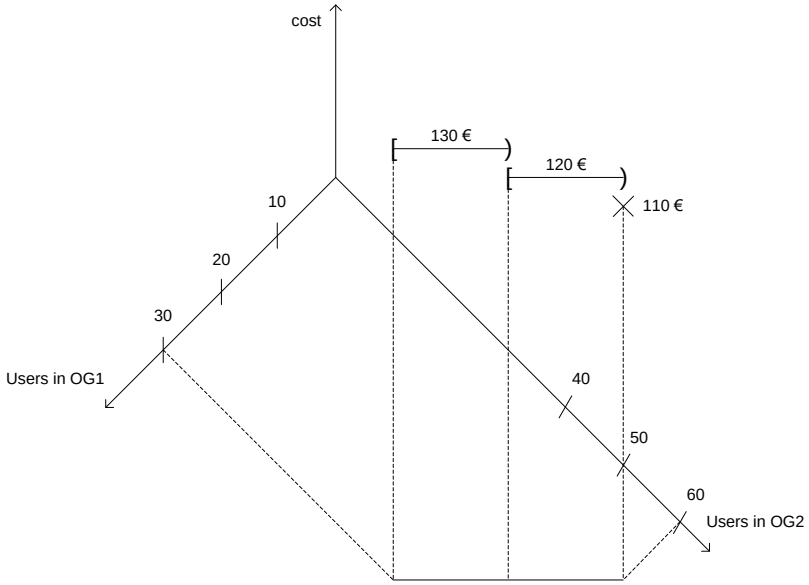

Figure 4.3: Possible user distribution and cost for this scenario.

function since costs only raise if a new instance is needed and are constant until this instance is fully utilized.
The fitness function is constrained by valid user ranges for the Offering Groups as well as by the total user amount. In areas where these constraints are not met the cost function is undefined. If the number of users were not constrained the cost function would be more like a plane with steps in it having its lowest point at the origin. This is so due to the fact that the

cost is always minimal if no users are present in the Offering Groups. While the following fitness function may utilize the cost function defined for the Festival Problem in Section 4.1 it has to respect these additional constraints for the user numbers. These numbers are contained in the Optimization View of the Resource Model.
In every setup there is a total number of users.

$$U_{total} \in \mathbb{N}$$

Furthermore there is a set of Offering Groups (OG) which is the equivalent for the set of tent types (Tt).

$$OG = \{OG_1, \ldots, OG_n\}$$

Each Offering Group has a certain Granularity (G) defining the number of users which can be serviced using one instance. This is equivalent to the tent capacity (κ).

$$G_i \in \mathbb{N}$$

Just as a tent type each Offering Group has a price (P) which has to be payed for one instance.

$$P_i \in \mathbb{R}^+$$

The additional characteristic of the Offering Groups are the ranges of users which may be assigned to them (k_i). In general k_i is an interval of positive integers:

$$k_i = [l, \ldots, m]\ l, m \subset N,\ l < m$$

Due to the setup of the scenario the silver Offering Group evidently can be filled with anything between 10 and 30 users while the gold Offering Groups may service 40 to 60 users which can be calculated from the users actually singing up for the Offering Groups and their possible distribution between them.

$$k_1 = [10...30]$$
$$k_2 = [40...60]$$

A function $User(OG_i)$ will return the number of users currently assigned to an Offering Group.

$$User(OG_i) = k_i$$

Considering all these factors the fitness function ($fitness$) can be formulated as such:

$$fitness = \begin{cases} \sum_{i=1}^{|OG|} cost(OG_i) & \text{if } \sum_{i=1}^{|OG|} User(OG_i) = U_{total} \\ undefined & else \end{cases}$$

Within the scope of the following algorithms two procedures are utilized to calculate the fitness of an element in the search scope. The cost function shown in Algorithm 4.1 calculates the price for a given number of resource instances in the optimized Offering Groups. The next section shows why it is more efficient to use instances than user numbers. Another procedure shown in Algorithm 4.2 evaluates before hand if an element is in fact valid. It ensures that the additional constraints respecting user numbers are met.

Algorithm 4.1 Cost Function for an Instance Vector respecting the price of minimal instances.

```
procedure COST(instances[ ], min[ ], prices[ ])
    price ← 0
    for i ← 0, size of instances[ ] do
        if instaces[i] ≥ min[i] then
            price ← price + instances[i] · price[i]
        else
            price ← price + min[i] · price[i]
        end if
    end for
    return price
end procedure
```

4.2.2 Moving to Neighbors in the Search Scope

One step for the local search methods is to choose another neighbor from the neighborhood of the current element. In case of three Offering Groups which all may service the total amount of users a possible search scope was shown in Figure 4.1. If Offering Groups have minimal and maximal utilizations due to tenant's demands the search scope would be a subset of the depicted one with fringed edges and holes.
In order to perform a step the elements of the neighborhood have to be comparable with respect to the fitness function. When using the naive approach using the user distribution for the search scope a neighbor would be equivalent to the current element in all but two user values for Offering Groups between which one user has been exchanged.
However the cost function for the scenario shown in Figure 4.3 is not continuous due to the fact that the cost of an Offering Group will not decrease nor increase if one user is removed or added respectively unless this alteration results in fewer or more instances required in it. Imagining more than two Offering Groups in the search scope the cost function would show multidimensional planes in which all neighboring elements will have the same cost unless moving to the neighbor results in moving over the edge of a plane of the cost function. When utilizing the user distribution as search scope these planes drastically hinder the effectiveness of local search methods since neighbors cannot be compared effectively anymore. The next step should always move into another plane which is very hard to achieve. More precisely not the neighbors of an element should be compared but the neighboring planes of the one in which the cost of the current element resides.
Therefore instead of manipulating the user distribution the number of instances should be used to define an element's neighborhood since addition or subtraction of an instance will definitely result in moving to a different plane in the cost function. Unfortunately instances cannot be interchanged between Offering Groups as easily as users. Removing an instance in one Offering Group does not necessarily result in adding exactly one in another Offering Group due to different granularities. In the strict sense this would result in many possible distribution of freed users. However the number of computed neighbors can be reduced while still allowing the algorithm to find good results.

The following algorithms will therefore consider an instance vector ($\overrightarrow{i}$) as an element of the search scope. Each index of this vector defines how many instances are provisioned for the corresponding Offering Group.

$$\overrightarrow{i} = \begin{pmatrix} \text{Instances in } OG_1 \\ \vdots \\ \text{Instances in } OG_n \end{pmatrix}$$

In order to determine the neighbors of such an element the following alterations are performed for every index of the current instance vector.

- Remove one instance from the current index and iterate over the other indexes. For every index a new neighbor is calculated by adding as many instances as are necessary to handle the amount of users which were removed.
- Create a new neighbor by separately removing one instance from all indexes but the current one adding as many instances as are needed to handle the freed users to the current index.
- Remove one instance from the current index and distribute freed users equally among the other Offering Groups.
- Individually remove an instance in each index since users might fit completely into free space in other Offering Groups.

These alterations are not considering the constraints for user ranges in Offering Groups nor the total amount of users. Therefore every computed neighbor has to be validated against these constraints before adding it to the neighborhood. The procedure performing this check is shown in Algorithm 4.2. Also important is the handling of minimal and maximal boundaries for resources in the Offering Groups. It has to be verified that a valid neighbor will not assign so many users so that an Offering Group would require more than its maximal number of resources to handle them. It is not verified that the assigned number of users fully utilizes the minimal number of resources since a neighbor not utilizing them is still valid. In fact there may be no element in the search scope totally utilizing the minimal resources and therefore this check must not be included here.
But user distributions leaving those resources unused will not be considered as good results by the algorithms due to their high price. At the beginning one valid element is selected to start from. In general this could be the obvious result for which all tenants are provisioned in the initially requested Offering Group with no substitution taking place at all.

4.2.3 Selection Strategies

Local search algorithms usually employ one of the following selection strategies [Nor03, p. 112 ff.].

Algorithm 4.2 Validation of a neighbor element

```
procedure isValidNeighbor(neighbor[ ], granularity[ ], min, max, maxInst, useramount)
    neighboramount ← 0                                  // amount of users in this neighbor
    for i ← 0, size of neighbor[ ] do
        capability ← neighbor[i] ∗ granularity[i]       // possible amount of users
        if capability < min then
            return false                                // cannot handle enough users
        end if
        if capability > max then                        // can handle more users than necessary
            if capability − granularity[i] > max then
                return false                            // do not allow completely empty instances
            end if
            neighboramount ← neighboramount + max
        else
            neighboramount ← neighboramount + capability
        end if
        if maxInstances is specified then
            if ⌈neighbor[i]/granularity[i]⌉ > maxInst then
                return false                            // would require too many instances
            end if
        end if
    end for
    if neighboramount < useramount then
        return false
    end if
    return true
end procedure
```

- Steepest Ascent: all neighbors are evaluated and the best one is chosen. This will result in a deterministic behavior but also in a longer search since the whole neighborhood has to be evaluated before the next step.
- First Choice: evaluate neighbors only until a better one is found and move to that one. While this will also result in a deterministic behavior search time is reduced.
- Stochastic: choose a random better element from the neighborhood or just a random one. The later will be performed by the Simulated Annealing algorithm.

4.2.4 Local and Global Optima

A local optimum is a result for which none of its neighbors have a better value for the fitness function [Nor03, p. 110]. A global optima is also a local optima but additionally there is no other valid element in the whole search scope which has a better fitness.

Algorithm 4.3 The Hillclimbing Algorithm

```
procedure HillClimbing(userdistribution[ ], min[ ], max[ ], granularity[ ], prices[ ])
    instances[ ]                          // Initialized accordant to the user distribution.
    price ← cost(instances[ ], min[ ], prices[ ])
    boolean foundbetter
    while true do do
        N ←calculateNeighborhood( )
        foundbetter ← false
        for neighbor ∈ N do
            tmpprice ← cost(neighbor, min[ ], prices[ ])
            if neighbor.prize < tmpprice then
                result ← Copy of neighbor
                foundbetter ←true
            end if
        end for
        if foundbetter == false  then
            break
        end if
    end while
    return neighbor
end procedure
```

While moving through the search scope the only way to tell whether a local or a global optimum was found is to evaluate all elements. This is one of the major problems when exploiting local search techniques. Even though the costs will steadily degrade when moving towards results with more users in the cheaper Offering Groups the cheapest results will sometimes not have the cheapest Offering Groups provisioned to their maximum. Next the local search algorithms Hill Climbing [Noro3, p. 111] and several forms of Simulated Annealing [Noro3, p. 115] are discussed.

4.3 Hill Climbing

This is a straight forward method which finds local optima very quickly but is then stuck in them. It examines the neighborhood of the current element and always selects a better one. It will stop if there is no better value in the direct neighborhood. One attempt to circumvent the problem with local optima is introduced by Random-Restart Hill Climbing [Noro3, p. 113]. It starts a series of Hill Climbing searches from a set of random elements in the search scope. Hill Climbing usually shows good results if started from multiple elements each favoring one Offering Group if the Offering Group substitution graph is strongly connected. However in other scenarios the problem with local optima is more eminent.

Algorithm 4.4 The Simulated Annealing Algorithm

```
procedure SimA(userdistribution[ ], min[ ], max[ ], granularity[ ], prices[ ], cooling)
    instances[ ]                          // Initialized accordant to the user distribution.
    state ← instances[ ]
    stateprice ← cost(state, min[ ], prices[ ])
    result ← state
    resultprice ← stateprice
    probability ← 1
    while probability > 0 do
        probability ← probability − cooling
        neighborhood ← calculateNeighborHood( )
        neighbor ←random element ∈ N
        tmpprice ← cost(neighbor, min[ ], prices[ ])
        if tmpprice < stateprice then
            state ← neighbor
            stateprice ← tmpprice
            if tmpprice < resultprice then
                result ← state
                price ← stateprice
            end if
        else
            random ← random number between 0 and 1
            if random < probability then
                state ← neighbor
                price ← tmpprice
            end if
        end if
    end while
    return result
end procedure
```

4.4 Simulated Annealing

The reason why the Hill Climbing approach gets stuck in local optima is that it never takes a step to a weaker element. The values for the fitness function have to increase steadily.
Imagine you have a huge metal bowl and a marble. The bowl is old and rugged and therefore has a lot of dents in its surface. If you let the marble slide in from the rim it will get stuck in one of the dents instead of rolling all the way to the center. The center could be considered the global optimum while the dents represent local optima. If the bowl is shaken a little bit the marble jumps out of the dents and eventually comes to rest in the center of the bowl.
Simulated Annealing functions similarly allowing to escape local optima. It has its origin in statistical mechanics [KGV83, p. 672] where it was observed that metal which has been cooled slowly or was exposed to several heating and cooling phases showed a more regular

atomic structure than metal which was cooled down rapidly. This was due to the fact that slow cooling and re-heating gave the atoms more time to move into less energetic states. Due to this origin the factor determining the behavior of the Simulated Annealing algorithms is commonly referred to as cooling rate. Algorithm 4.4 is almost the original version of Simulated Annealing [Nor03, p. 116]. The longer the algorithm runs the more unlikely it gets that a weaker element is chosen. This behavior leads to the very commonly used derivation from the original algorithm. Instead of returning the current state when finished the best state visited is stored separately. Otherwise the movement to a weaker element close to the end of the runtime would lead to a weaker result than what has actually been computed before. Another alteration introduced here is to utilize a linear decreasing cooling rate while the original approach decreases exponentially.
The challenging aspect when using this algorithm is to choose an adequate cooling rate since it is the only parameter affecting the runtime. Reducing the probability too quickly will result in non optimal results.
Due to the nature of the Festival Problem another heuristic can be used to further improve performance. The cost function is the only criterion which has to be optimized so search algorithms should lean towards the cheaper results. Hill Climbing certainly is doing this to the highest degree since it will only move to better elements. Chapter 5 shows that even with this rigorous technique very satisfying results are computed in most cases. Therefore the Simulated Annealing algorithms should also exploit the simplicity of the search scope. Two alterations are defined, the *Smart Simulated Annealing* and the *Smarter Simulated Annealing* algorithm. Both adapt the search for a better element from Hill Climbing at some point during their runtime.

4.4.1 Smart Simulated Annealing

Smart Simulated Annealing is very close to Hill Climbing since it starts each iteration by searching for a better element. Only if none is found it chooses a random one from the weaker elements and moves to it with a certain probability. To avoid moving right back to the better element moved away from, the elements stepped down from are stored and never visited again.
Since the search scope is of such a simple nature only few downward steps will be performed. However Smart Simulated Annealing is suspected to have additional problems in scenarios with Offering Groups having the same granularity. In such cases the algorithm is likely to stay in the area around a local optimum visiting the surrounding elements and stepping down from each one of them too. So to leave such a local optimum will take the algorithm more steps.
This effect is further examined in Section 5.2.2 where this algorithm is compared to Smarter Simulated Annealing.

Algorithm 4.5 The Smart Simulated Annealing Algorithm

```
procedure SmartSimA(userdistribution[ ], min[ ], max[ ], granularity[ ], prices[ ], cooling)
    instances[ ]                          // Initialized accordant to the user distribution.
    Initialized as Simulated Annealing
    S ← ∅                                 // Set of elements stepped down from.
    while probability > 0 do
        probability ← probability − cooling
        N ← calculateNeighborHood( )
        for neighbor ∈ N do
            tmpprice ← cost(neighbor, min[ ], prices[ ])
            if tmpprice < stateprice then
                if neighbor ∉ S then
                    state ← neighbor
                    stateprice ← tmpprice
                    if tmpprice < resultprice then
                        result ← state
                        resultprice ← stateprice
                    end if
                end if
            else
                random ← random number between 0 and 1
                if random < probability then
                    S ← S ∪ neighbor
                    state ← neighbor
                    stateprice ← tmpprice
                end if
            end if
        end for
    end while
    return result
end procedure
```

4.4.2 Smarter Simulated Annealing

Whereas Smart Simulated Annealing was quite similar to Hill Climbing, Smarter Simulated Annealing is more like the original version of Simulated Annealing despite the additional operation shown in Algorithm 4.6. It computes a random element and will choose it right away if it is better and with a certain probability if it is a worse element just like the original version. However in case a worse element is not chosen it will iterate over the whole neighborhood searching for a better one. This behavior combines the positive aspects of both the Hill Climbing and the Simulated Annealing approaches since it allows moving down out of local optima at the beginning of the search and moves strongly towards an optimum during its end. This also nullifies the need to remember elements stepped down from since

Algorithm 4.6 The Smarter Simulated Annealing Algorithm

```
procedure SmarterSimA(userdistribution[ ], min[ ], max[ ], granularity[ ], prices[ ], cooling)
    ...
    if random < probability then
        state ← neighbor
        stateprice ← tmpprice
    else
        for neighbor2 ∈ N do
            tmpprice ← cost(neighbor2, min[ ], prices[ ])
            if tmpprice < resultprice then
                result ← neighbor2
                result ← tmpprice
            end if
        end for
    end if
    ...
end procedure
```

the random move is done before the iterative search. Therefore the problems considering the same granularities and local optima will not be of significance until late in the search sequence. At this point the algorithm is much more likely to be in the general area of the global optimum.
It can already be suspected that Smart Simulated Annealing will be more affected by the additional computations than Smarter Simulated Annealing since it has to perform them in every iteration. Smarter Simulated Annealing is likely to omit the additional search at the beginning of its runtime. Close to the end it is better to calculate more during an iteration than performing no movement at all as the original version is likely to do. This is also examined further in Chapter 5.

4.5 User Distribution Algorithm

Using any of the local search methods the result will be a vector indicating how many instances shall be provisioned into each Offering Group. The local search algorithms use minimum and maximum user numbers as constraints for the Offering Groups instead of using a concrete notion of tenants. While this is totally sufficient to calculate the close to optimal number of instances the routing algorithm needs concrete rules applied to each tenant request.
The arising problem is once more to distribute a tenant's users among different Offering Groups. However running a local search before leads to another major assumption which could not be made before: The optimized system can definitely support the number of users. Therefore a linear algorithm can be defined which distributes users across the Offering Groups. It roughly consist of the following two steps.

- Create a spanning tree of the substitution graph.
- Traverse through the tree and assign users sorted by number of substitution links of their Usage Partitions. Do so starting with the one having the fewest links.

4.5.1 Calculating the Spanning Tree

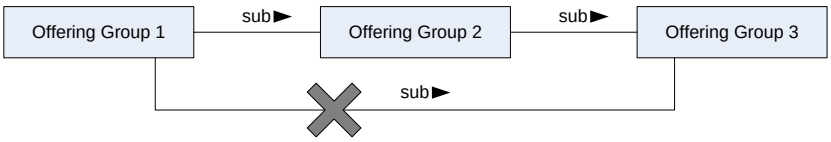

Figure 4.4: Obtaining the minimal substitution tree

Traversing a tree is a lot more efficient than traversing a graph since a concrete direction is known and nodes are never visited twice. The substitution graph is therefore transformed into a tree. Since the substitution edges are not weighted a simplified version of the Kruskal algorithm [Man89, p. 567] may be used for this task. Link removal to obtain the substitution tree is depicted in Figure 4.4. It can also be seen, that this algorithm ensures that all substitution links are hierarchical and performs the reversed function of the flattening process discussed in Section 3.1.7.

4.5.2 Correctness of the Distribution Algorithm

It has to be proven that there will never be any users left after traversing the tree once. Otherwise the algorithm would have to include backward or recursive steps which would hurt performance badly. Since the number of instances is guaranteed to handle all users the only potential problem is that users of a tenant are assigned to an Offering Group which could also be used by another tenant and the later tenant cannot use any other vacant space. Let there be two Usage Partitions (u_1 and u_2) sorted by number of substitution links. They are assigned to Offering Groups starting with the smallest one:

$$u_1 > u_2 \equiv u_1 \text{ has more substitution links than } u_2$$

Assume a scenario in which u_1 blocks space which must be used by u_2. Three cases for the relationship between those two Usage Partitions have to be considered.

1. $u_1 > u_2$: u_2 always gets assigned to Offering Groups before u_1 and therefore u_1 can never consume space u_2 must use.
2. $u_1 < u_2$: u_1 may indeed take up space which could also be used by u_2. However u_2 can always use some other space since it accepts more Offering Groups than u_1 and it is ensured that there is enough room in the overall system.

Algorithm 4.7 The User Distribution Algorithm

```
procedure distributeUsers( )
    OG                                                  // Set of all Offering Groups.
    E                                                   // Set off all Edges.
    ST ← ∅                          // Set to remember which nodes are in the spanning tree.
    R ← ∅                                // Set containing possible roots of the spanning tree.
    for edge ∈ E do        // edge.to and edge.from return the Offering Groups at the ends.
        if edge.from ∈ ST ∧ edge.to ∈ ST then
            E ← E \ edge
        else
            if edge.from ∉ ST then
                ST ← ST ⋃ edge.from
            end if
            if edge.to ∉ ST then
                ST ← ST ⋃ edge.to
            end if
        end if
    end for
    for og ∈ ST do
        if number of incoming edges for og == 0 then
            R ← R ⋃ og
        end if
    end for
    for og ∈ R but one do
        provision(og)
    end for
    traverseAndProvision(lastog)                // lastog is the one not yet provisioned.
end procedure
```

3. $u_1 = u_2$: in this case both Usage Partitions actually compete over identical or at least the same amount of Offering Groups. This could lead to a dead lock but only if there were not enough resources. Since the system is guaranteed to be able to handle all users the order in which u_1 and u_2 are assigned is irrelevant and even though they are competitors both their demands will be met always.

Due to these facts the distribution algorithm is guaranteed to find one legal distribution of users among the number of instances in Offering Groups. A various number of possible distributions exist. Which one is found is mainly affected by the order in which Usage Partitions having an equal number of substitution links are assigned.

4.6 Performing Global and Request based Optimization

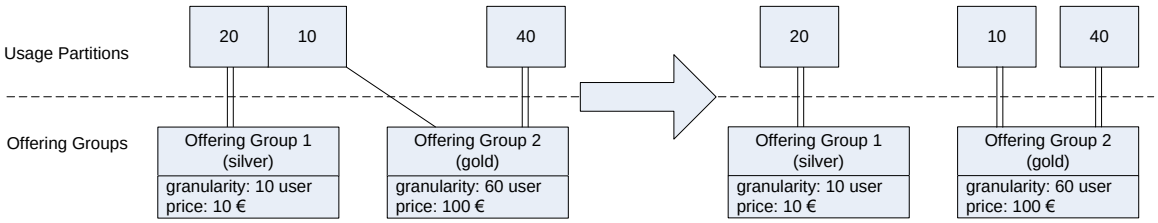

Figure 4.5: Deriving a temporary Condensed View for Request Optimization.

As mentioned earlier the local search algorithms have to enable global as well as request based optimization. In general moving users and resources handling their requests might take up significant time. Considering users of a tenant being served completely in a local data center, moving them to an external provider might take a lot of time and should therefore not be done frequently or might not even be possible at all. The later may be handled simply by deleting the substitution links of a tenants Usage Partitions once they have been provisioned. The former case is solved using request based optimization which allows parts of a system to be left untouched until moving users becomes profitable. The result of both optimization methods may be calculated and if a globally optimized system becomes sufficiently cheaper than a request based optimized system the whole system may be restructured. In other cases only small provisioning tasks will have to be completed.
The usage of the algorithms discussed in Section 4.2 resembles global optimization since all Usage Partitions have been distributed as if just requested. Within the scope of request based optimization a subset of the Usage Partitions has already been distributed and provisioned. The other users have just been requested and are therefore part of Registration Partitions.
The algorithms defined so far have to be utilized to distribute those new users to the Offering Groups leaving already distributed users untouched. Only few alterations have to be made to the global optimization process to achieve this functionality. When the model has been deflated into the Condensed View before transforming it into the Optimization View the already provisioned Usage Partitions are split up. For the registration link and for every substitution link of a temporary Usage Partition is created which only has a registration link for the registered or substituted Offering Group and contains the corresponding amount of users. The Registration Partitions are then transformed into regular Usage Partitions and added to the Condensed View. This model is then going through the normal optimization process.
During the user distribution and the inflation of the model only the new Usage Partitions are considered. Using this approach unused capabilities of already provisioned resources are considered during the optimization process for the new Usage Partitions and already distributed users are left unchanged.

4.7 Failed Optimization Attempts

Before the local search algorithms were defined several unsuccessful optimization attempts were conducted. Those attempts and their flaws are described in this section.

4.7.1 Greedy Optimization Algorithms

Greedy algorithms [Man89, p. 370 ff.] often provide very fast solutions but need some optimization criterion which can be evaluated during every step of a single iteration.
When calculating the optimal amount of instances required in every Offering Group iterating over these groups provisioning instances according to an optimization criterion would be the mode of operation for greedy algorithms. Therefore the success of greedy algorithms depends on the intelligent determination of the order in which the Offering Groups are visited and the art how the visited Offering Group is provisioned. In order to make these decisions a few more assumptions about the problem are made:

1. Offering Groups with higher quality of services have more incoming substitution links.
2. With raising quality of services the Price per user also increases.

The first assumption can be considered to hold most of the time since users by definition may only be delegated to Offering Groups with equal or better quality of services. The second assumption may be violated in case Offering Groups with lesser quality own Grouped Quality of Services such as location. In that case it may not be possible to provision all users into the more powerful and cheaper Offering Group due to these additional attributes. No assumption can be made about granularity of Offering Groups.
If separately sorting the Offering Groups by number of substitution links, price per user, and granularity always results in the same order a distinct provisioning order is found. In this special case a greedy algorithm could be used to find the optimal distribution. Unfortunately this case will almost never occur in practice. But it indicates how the next Offering Group to be provisioned should be selected:

- It should have many incoming substitution links and therefore unused capabilities are likely to be substituted to.
- It should be cheap.
- It is unlikely that following Offering Groups will generate possibilities to substitute for the current one.

Comparing these qualities to the assumptions made over the system shows the dilemma of greedy optimization algorithms. An Offering Group which has many incoming substitution links is unlikely to be cheap. But when iterating over Offering Groups starting with the most expensive one delegating to cheaper Offering Groups often is impossible. The ordering would have to represent an optimal ratio between price and number of substitution links. This is why the following algorithms will always perform well only under certain conditions.

They cannot be used in general for optimization. However Section 5.2.3 shows that they can serve well to find a good starting element for local search methods.

Unavoidable Evil

In general if users request high quality, high quality has to be provisioned. Expensive Offering Groups are considered to have few substitution links, so they are likely to provision all users in themselves.

- Sort Offering Groups by price per user.
- Iterate over this list starting with the most expensive one and provision users in it which directly registered for this Offering Group.
- While provisioning check if instances can be omitted by delegating users to free space in other Offering Groups.

Using this approach expensive Offering Groups are provisioned first which is likely to result in many substitution possibilities for following Offering Groups. It is further guaranteed that these substitution possibilities will be used by the next most expensive users. However it will never find a user distribution exploiting better fitting granularities in other Offering Groups.

Cheap is good

This approach does the opposite to the one just introduced. It considers provisioning cheap resources as being better than provisioning expensive ones.

- Sort Offering Groups by price per user.
- Iterate over this list starting with the cheapest one and provision as many users as possible. Delegate them if possible.

This tactic will only lead to good results if expensive resources can actually substitute into cheaper ones.

Tear down what is not needed

This algorithm is very similar to the one just described.

- Sort Offering Groups by price per user.
- Provision in each Offering Group the users directly registered for it.
- Iterate over the Offering Groups starting with the cheapest. Try to remove provisioned instances by delegating users over substitution links.

4.7.2 Gradient Descent

In Section 4.2.1 a cost function was introduced to determine the fitness of visited elements in the search scope of local search algorithms. The gradient descent [Sny05, p. 34 ff.] is a method to find extreme points of functions. Instead of visiting the neighbors of a given starting point it calculates the gradient of the fitness function in this point and determines the next element utilizing this information.
The gradient descent requires a legal result or a point on the function to start from. A possible result shall be a vector ($\overrightarrow{u}$):

$$\overrightarrow{u} = \begin{pmatrix} User(OG_1) \\ \vdots \\ User(OG_n) \end{pmatrix}$$

Now the derivations of the cost function for each component of the vector is calculated. When derivating for a concrete OG_i all other parts of the sum will vanish leaving a factor dependent on the granularity and cost of the component for which the derivation was performed. This factor is larger the more expensive an Offering Group is. The values of the derivations for each component of $\overrightarrow{u}$ in the concrete point of the cost function is the gradient vector ($\overrightarrow{u}'_i$).
This vector is subtracted from the current point to move towards a minimum. In this case this would result in the following alteration of $\overrightarrow{u}$:

$$\overrightarrow{u}_{k+1} = \overrightarrow{u}_k - \overrightarrow{u}'_k$$

Utilizing only this alteration the gradient descent would always come to the conclusion that having no users at all leads to the cheapest price. Therefore the constraints defined separately for the fitness function would have to be represented in the function itself.
The main problem is that the gradient descent is used for optimization in continuous search scopes. The cost function however in not continuous but a step function. Therefore the direction in which to move cannot be determined from the gradient which is always zero for all points on a plane. This complicates the usage of the gradient descent extremely and it is not expected that it performs any better than the local search methods since it is just as likely to get stuck in local optima as the Hill Climbing algorithm defined in Section 4.3.

5 Performance of Optimization Algorithms

The optimization algorithms defined in Chapter 4 are now evaluated in several test cases. At first they are compared to an algorithm which computes all possible results and thus is guaranteed to find the global optimum. Obviously this may only be done for small scenarios. Well performing algorithms are then compared to each other using larger test cases. Afterwards a method to find good cooling rates and heuristics for Smarter Simulated Annealing is covered.

5.1 Total Computation Algorithm

There are multiple approaches by which to calculate all possible results. The most trivial one is the *Total Computation* algorithm 5.1 which iterates over the possible minimum and maximum values for the user distribution and validates each time if the distribution contains the specified number of users. A similar algorithm could be defined for the possible number of instances in the Offering Groups since their minimum and maximum values can be derived easily from the minimum and maximum users and the Offering Groups' granularities. This approach would result in fewer steps and a slightly more complicated test for the total amount of users. However this reduces the complexity only by a static factor

Algorithm 5.1 Total Computation Algorithm for three Offering Groups

```
procedure computeBestResult(min[ ], max[ ], useramount)
    price ← ∞
    for og1 ← min[0], max[0] do                    // Valid domain of first Offering Group
        for og2 ← min[1], max[1] do                // Valid domain of second Offering Group
            for og3 ← min[2], max[2] do            // Valid domain of third Offering Group
                if og1 + og2 + og3 == useramount then
                    if price for current distribution < price then
                        price ← price for current distribution
                    end if
                end if
            end for
        end for
    end for
    return price
end procedure
```

and the algorithm will perform equally for larger numbers of users and Offering Groups. Complexity of both approaches is exponential:

$$O(og^u) \text{ with: } og: \text{ number of Offering Groups, } u: \text{ number of Users}$$

However a lot of computed distributions will not be evaluated to being valid in the innermost for statement. So instead of considering all possible distributions an algorithm would perform much better if it moves around the search scope as the local search algorithms. Instead of looking for an optimum it would ensure total coverage of the search scope. It has been shown that this is hard to achieve in the instance search scope in Section 4.2.2 but in the user search scope all legal user distribution could be computed by moving around in it. This way the algorithms complexity would be reduced to the size of the search scope. Computing this size and all its elements has the same complexity as the following mathematical problem.

5.1.1 Estimating the Search Scope Size

Given the set of numbers having the length (n) and the base (g) the following formula determines the amount(A) of numbers having a given cross sum (q) [Roh48].

$$A_{n,q}^{(g)} = \sum_{v=0}^{\infty} (-1)^v \binom{n}{v} \binom{q - gv + n - 1}{n - 1}$$

The possible user distribution among Offering Groups can be mapped to this problem. The length of the numbers is the number of Offering Groups. Considering that all users may be in every Offering Group the base of these numbers equates to the total number of users which is also equal to the required cross sum. Even though the user number boundaries are not reflected in this formula it allows to estimate the size of the search scope for a given scenario.
Since A grows extremely quickly it shows that the number of comparisons needed to evaluate the complete search scope are significantly higher than those of the local search methods even for small scenarios. Therefore none of the possible total computation algorithms is a good candidate for productive use.

5.2 Test Cases

All test cases are constituted on random values for the minimal and maximal user numbers as well as for granularities and prizes. Thus the possible distribution of substitution links resulting in these values can also be considered random. Even though the real world most likely allows additional assumptions to be made, an algorithm performing well in a randomized environment will also perform well in such a real world environment. Random number generators for all test cases have been initialized with the same seed. This means that they will produce the same sequence of pseudo random numbers which allows results in the graphics to be compared with each other.
The testing environment was an Intel dual core CPU running at two GHz.

5.2.1 Performance Test

This test case computes the global optimum for each case using the total computation algorithm. Since this is only possible for small scenarios it consists of three Offering Groups with a maximum of 1500 users each. 1000 test cases were performed comparing the best result to that one found by the examined optimization algorithm. Test case results show the number of the test case on the X-axis. On the Y-axis the deviation by which the result found is more expensive is plotted. For example if the optimal price were 100 and the optimization algorithms ends up with a price of 120 the deviation would be 20 %. It is computed using the following formula:

$$\text{deviation} = \left(\frac{\text{result}}{\text{bestresult}} - 1 \right) \cdot 100$$

Hill Climbing

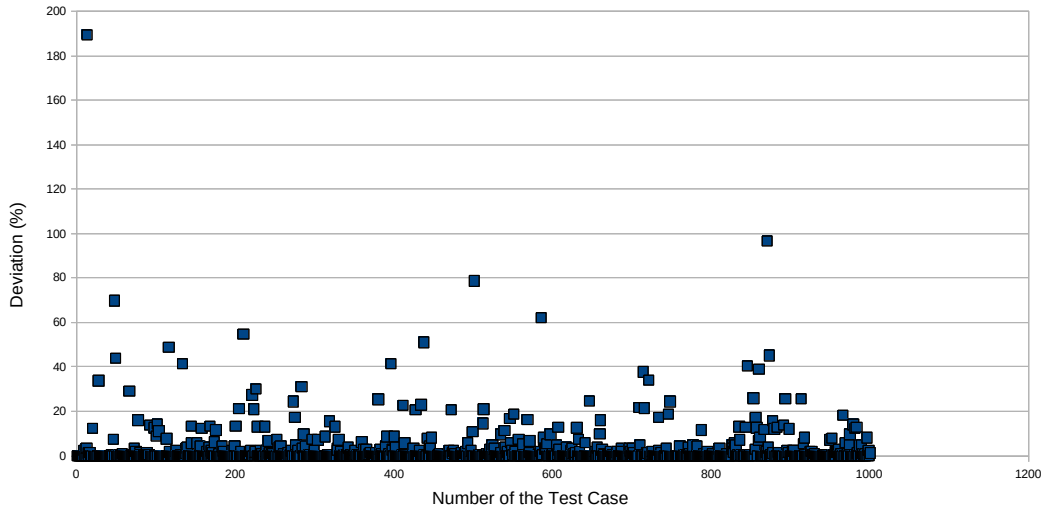

Figure 5.1: Performance of Hill Climbing

Since the environment is so small Hill Climbing is performing very well in most cases as shown in Figure 5.1. However there are some bad results ranging in 60% to 80% and one extreme result is even almost twice as expensive as the optimum. For larger test cases the number of such bad results is expected to increase since the number of local optima will also.

Original Simulated Annealing

All Simulated Annealing approaches are examined. Their cooling rates are set to allow 10000 iterations. While normally a lot more iterations would be allowed, for testing purposes this would not lead to significant results. With enough steps all Simulated Annealing algorithms

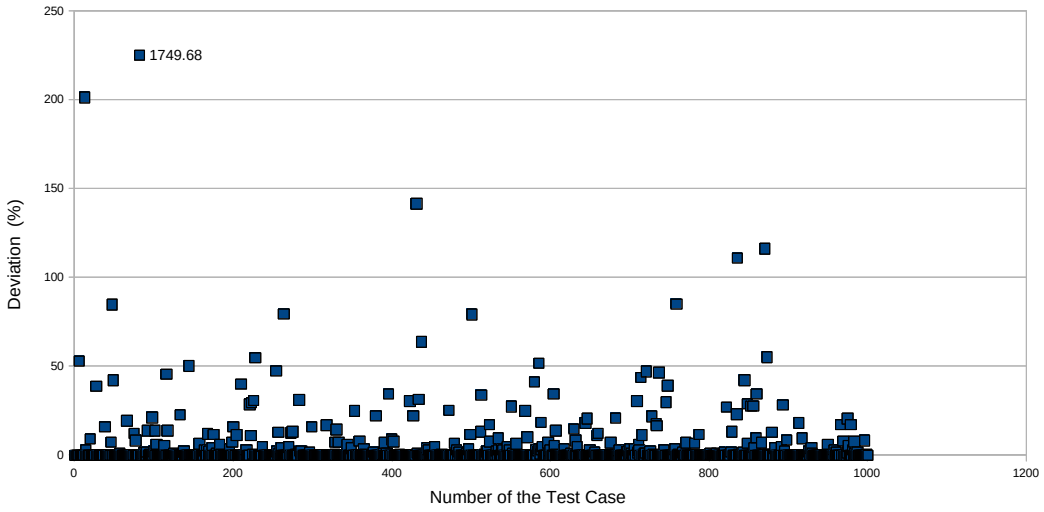

Figure 5.2: Performance of Simulated Annealing

would lead to very good results.
The original Simulated Annealing algorithm is not performing so well. Since it only selects one random neighbor in each iteration the simplicity of the search scope is not exploited. The chance that a worse element is examined randomly is quite high. Therefore the algorithm is likely to perform no step at all at the end of its runtime when it is unlikely that the randomly examined and worse element is actually selected.

Smart Simulated Annealing

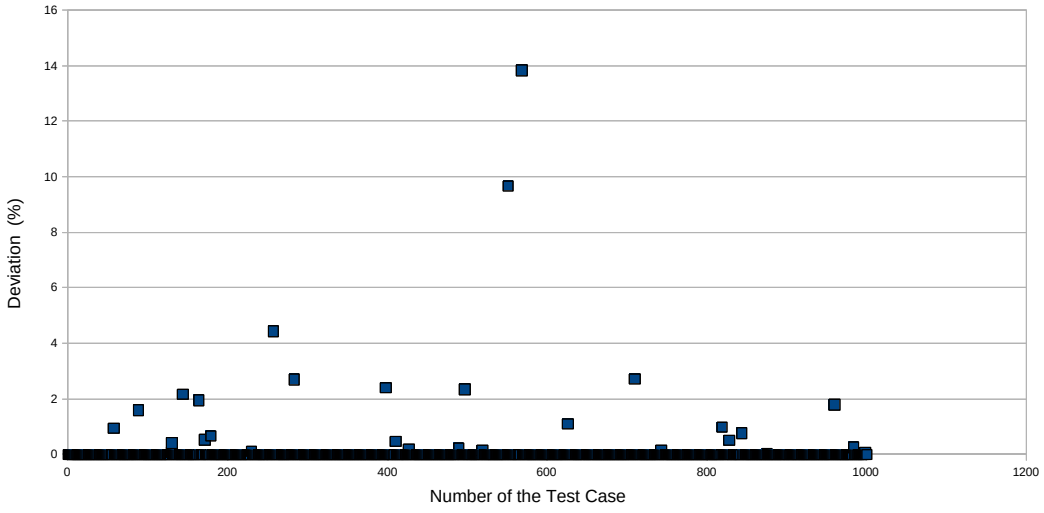

Figure 5.3: Performance of Smart Simulated Annealing

Smart Simulated Annealing performs better with the same cooling rate as the original algorithm for two reasons. First it exploits the simplicity of the search scope by moving into a discrete direction and it is much more likely to perform steps during the end of its runtime. However this comes with a price. While Hill Climbing, Simulated Annealing and the following Smarter Simulated Annealing needed less than one second for one of the test cases Smart Simulated Annealing needed five. This quite long considering the simplicity of the test case. But Smart Simulated Annealing performs more checks in each iteration and therefore needs fewer iterations by design to lead to similar results as the other approaches.

Smarter Simulated Annealing

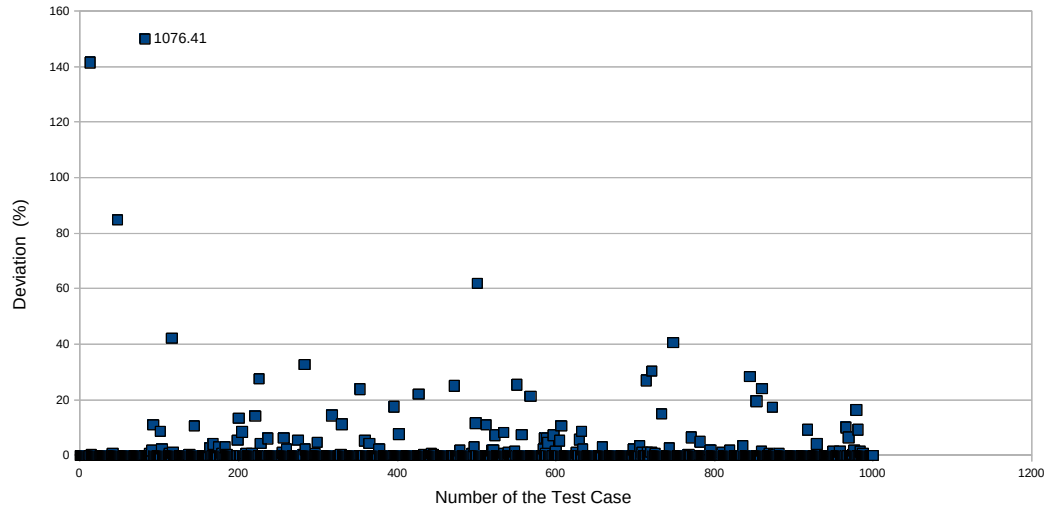

Figure 5.4: Performance of Smarter Simulated Annealing

Smarter Simulated Annealing also finds better results than the original approach. It can be seen in the extremely expensive results that it is more similar to the original approach than Smart Simulated Annealing which behaved more like Hill Climbing. One improvement is that no noticeable performance cutbacks resulting from the additional operations could be observed on the test machine.
Before the extreme results are discussed in Section 5.2.3 Smart and Smarter Simulated Annealing are compared to each other.

5.2.2 Comparison of Simulated Annealing Algorithms

While both alterations from the original Simulated Annealing algorithm displayed improvements Smart Simulated Annealing seems to be performing better since the extremely expensive results were not present. However it has already been suspected in Section 4.4.1 that Smart Simulated Annealing runs into problems when several Offering Groups have the same granularity. In such a case it is suspected that it takes Smart Simulated Annealing

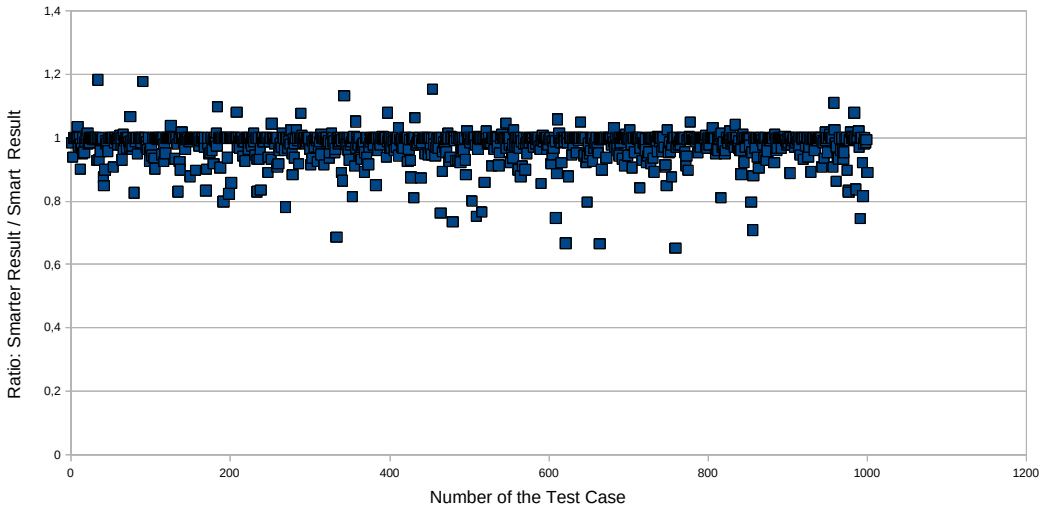

Figure 5.5: Comparison of Smart and Smarter Simulated Annealing

a lot longer to move away from a local optimum. This problem becomes only significant if a critical number of Offering Groups is present. Such a test case may not be computed totally in a reasonable amount of time. Therefore the results of Smart and Smarter Simulated Annealing are compared directly with each other.
The test case has a maximum of 15,000 users distributed among six Offering Groups with random but equal granularity. As said before Smart Simulated Annealing requires more time per iteration. To reduce the running time of this test case the number of iterations was reduced to 1000 for both algorithms. This should favor Smart Simulated Annealing since it performs more tests than Smarter Simulated Annealing in one iteration.
To display which approach lead to better results a ratio was computed:

$$\text{ratio} = \frac{\text{smarter result}}{\text{smart result}}$$
$$\text{smarter result} > \text{smart result} \quad \Leftrightarrow \text{ratio} > 1$$
$$\text{smarter result} < \text{smart result} \quad \Leftrightarrow \text{ratio} < 1$$

Figure 5.5 shows that Smarter Simulated Annealing despite being significantly faster also leads to better results. Its disadvantage in this test case resulting from the fewer tests performed in each iteration was clearly outweighed by the factor Smart Simulated Annealing was slowed down due to equal granularities.

5.2.3 Extreme Results and Heuristics

From the tests performed so far it became apparent that Smarter Simulated Annealing in general is the optimization algorithm of choice since it incorporated the advantages of Hill Climbing and those of the original Simulated Annealing. Only if granularities differ greatly in a use case Smarter Simulated Annealing may be considered for optimization.

However there were several extreme results in which Smarter Simulated Annealing performed very badly. As said earlier the cooling rate significantly affects the performance of Simulated Annealing. In general a smaller cooling rate will lead to better results but also to a longer runtime. The results of the performance test allowing one million iterations is depicted in Figure 5.6. Using this cooling rate Smarter Simulated Annealing needed

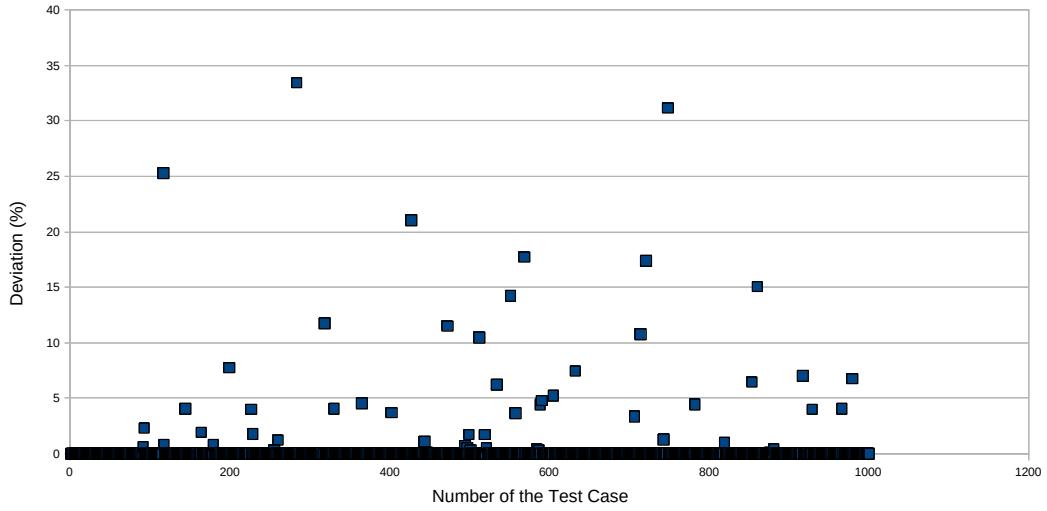

Figure 5.6: Performance of Smarter Simulated Annealing with more iterations

approximately five seconds for each case just like Smart Simulated Annealing using the original cooling rate. While this improved the results they are still not as good as those of Smart Simulated Annealing.
Analysis of the cases in which bad results were computed shows that they display a characteristic which is quite unlikely to occur in realistic setups. But since the test cases used here were generated randomly without any further semantic, scenarios as depicted exemplary in in Figure 5.7 were computed. They include Usage Partitions registering expensive Offering Groups allowing complete substitution into cheaper ones. Even though it could be argued

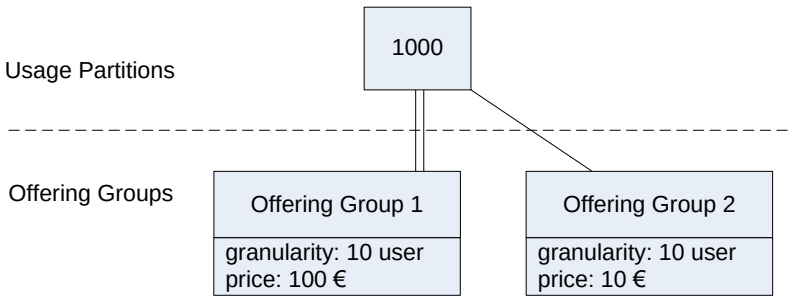

Figure 5.7: Scenario leading to long ways in the search scope

that this will never happen in practice there might be some irregular scenario in which it does. Assuming that this scenario might in fact be eligible substitution of large amount of

users away from the initially requested Offering Group results in long ways in the search scope between the starting element and the optimal element. Therefore Hill Climbing is likely to get stuck in local optima and Simulated Annealing runs out of steps.
This problem can be solved by using the greedy approach "Cheap is good" discussed in

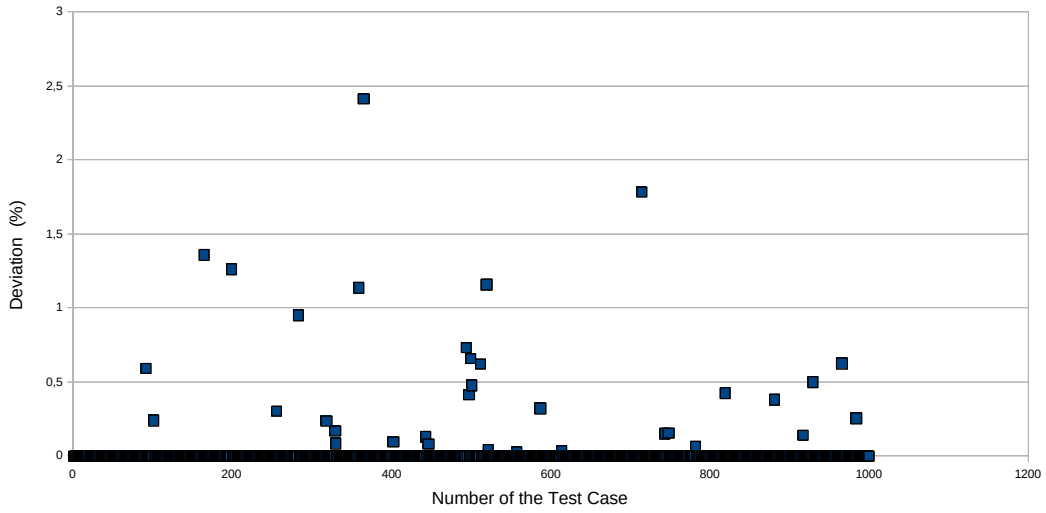

Figure 5.8: Performance of Smarter Simulated Annealing with more iterations and "Cheap is good"

Section 4.7.1 to find a more adequate starting point in the search scope instead of relying on the tenants' requests. The greedy algorithm assigns as many users as possible to cheap Offering Groups and then a local search can be started from that user distribution. Results are shown in Figure 5.8.
Their quality increased significantly and no more extreme results are present. However a greedy algorithm has to be selected carefully. "Cheap is good" will not improve the optimization if no substitution from expensive to cheaper Offering Groups is allowed which might be the case in realistic setups. Thus if the concrete scenario has an expensive Offering Group with a large granularity the only optimization opportunity might be to move users from cheaper groups into this expensive one. In such a case "Cheap is good" would move the starting element away from the optimum. A different greedy approach defined in Section 4.7.1 might be more adequate.
Due to these facts optimizing Smarter Simulated Annealing using greedy approaches exploiting certain heuristics has to be done on a per use case basis.

5.2.4 Cooling Rate and Load Test

Independent of any heuristics to find a good starting element the cooling rate may always be optimized to obtain good results. Since the the runtime of the Simulated Annealing algorithms is static selecting an adequate cooling rate for a given problem is very important. A trivial cooling rate may be found by measuring the time of one iteration and then allowing

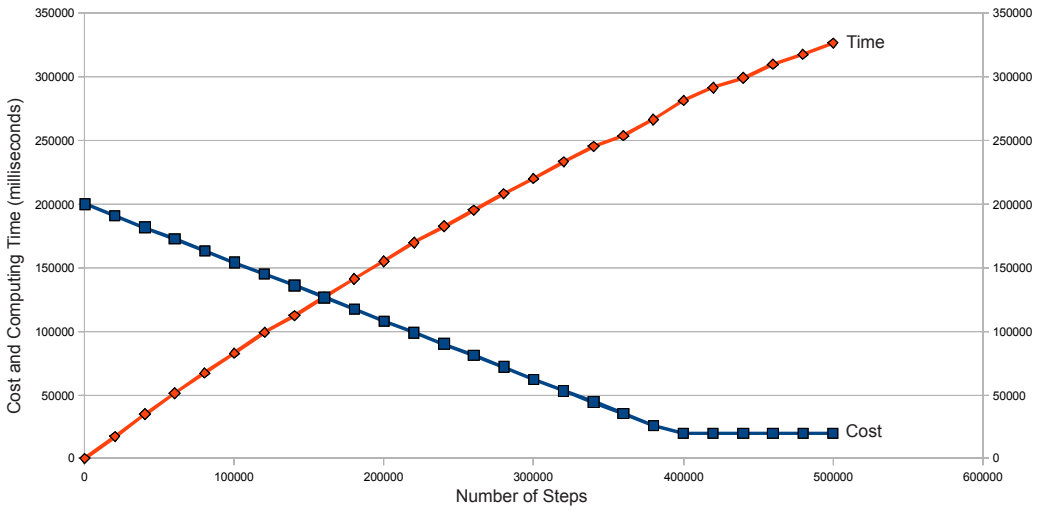

Figure 5.9: Determining an adequate cooling rate for Smarter Simulated Annealing.

as much iterations as the timing of the environment allows. So if one hour is acceptable for optimization a cooling rate could be selected so that the algorithm needs one hour.
However most likely a more sophisticated estimate is required. A good cooling rate may be obtained by a scenario which ensures extremely long ways in the search scope between the starting and the optimal element. Those scenarios are exactly what has lead to extreme results in earlier test cases. They can be build by assigning a large amount of users into the expensive Offering Groups which all can be substituted into the cheapest one. The cheapest Offering Group should be empty so that the maximum amount of steps has to be performed during optimization.
The concrete setup used here is such a scenario to put a maximum load on the Smarter Simulated Annealing algorithm. It includes 21 Offering Groups. All but one have price and granularity of one as well as 10,000 assigned users in the starting element. The last Offering Group is empty and also has a granularity of one but a price of only 0.1. Thus the optimal result is only found when all users are substituted into this Offering Group leading to a optimal price of 20.000. The algorithm was started multiple times increasing the number of iterations. The results are depicted in Figure 5.9. Running the algorithm once recording the results every few steps would not lead to meaningful results because Smarter Simulated Annealing does not behave equally during its runtime. As shown in Section 4.4.2 it behaves more like Hill Climbing at the end which would not have been respected if results were recorded during a single pass.
The optimal price was found after approximately 400,000 iterations. It only took the Smarter Simulated Annealing algorithm about five minutes to perform these steps on the test system for this complex scenario.

6 Implementation

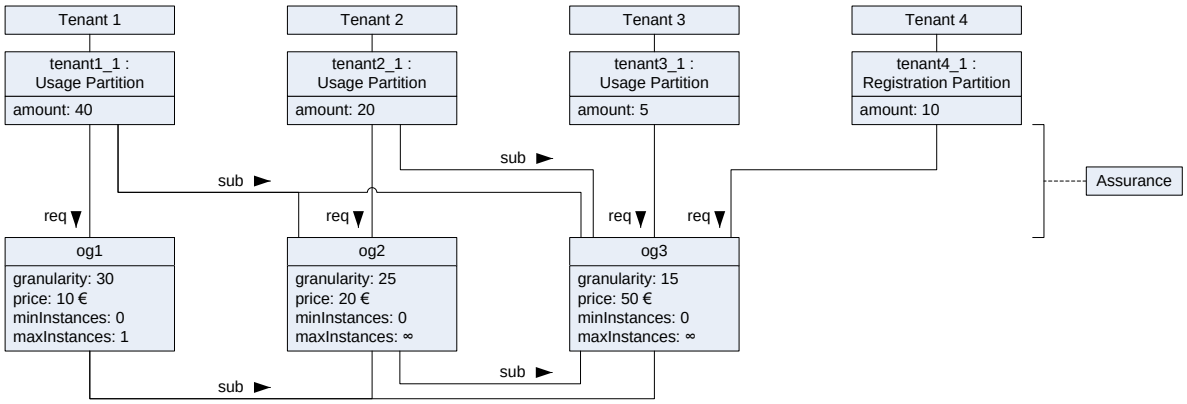

Figure 6.1: Implementation scenario

This Chapter shows how the information enclosed in the Resource Model is utilized. Since the main improvement is to dynamically provision heterogeneous pools of resources and load balance between them the implementation will focus on these aspects. It shows further how the registration and provisioning processes interact to create a runtime environment which is utilized optimally using tenant based routing.
The implemented scenario uses Atomic Application Groups since this is where load balancing and routing is performed. However since in more complex scenarios these Atomic Offering Groups could also be Infrastructure Groups the more general term Atomic Offering Group is used to refer to them. As seen in Figure 6.1 there are three Atomic Offering Groups. The substitution links between them indicate that Offering Group Two offers better quality of service than one and that Offering Group Three is better than both the others. The modeling of qualities themself as well as semantic descriptors has been omitted. They are assumed to be evaluated at some earlier point during the registration process resulting in the displayed substitution links between Offering Groups, Usage and Registration Partitions.
Furthermore the scenario contains three tenants currently signed up for the service and one tenant which has just initiated a request. By convention the names of Usage and Registration Partitions shall be the tenant id followed by a number. Before the individual optimization and provisioning processes are described a system architecture is introduced which supports them.

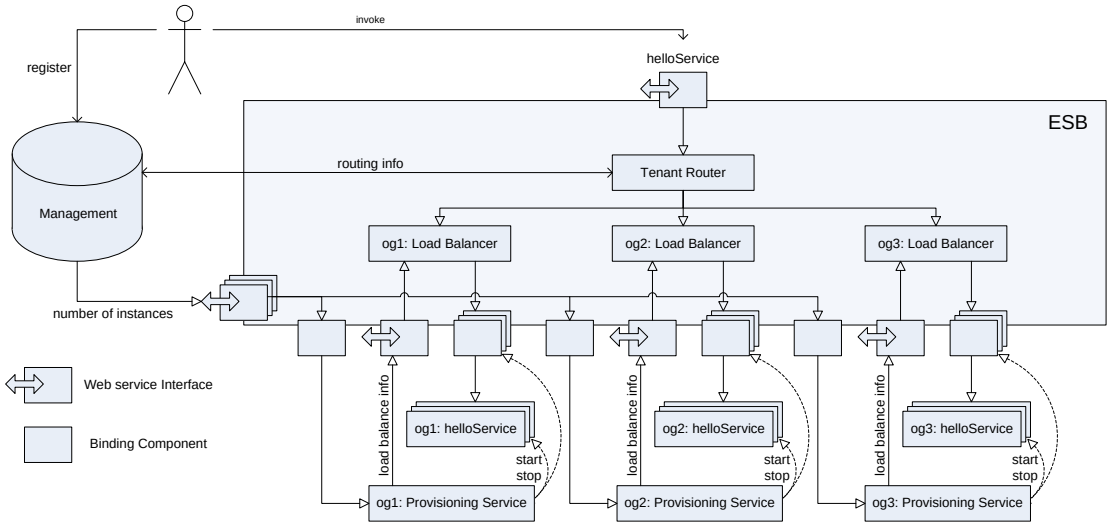

Figure 6.2: System architecture

6.1 System Architecture

All information contained in the Resource Model is stored in a central database. The tenants' registration requests are also stored in this database. Tenants interact with a simple echo service, called Hello Service. The *Tenant Router* will utilize the Assurances to route requests to Offering Groups based on the tenant identifier. The management application creates this information based on the result of the optimization process and requests the needed number of resources at the *Provisioning Services*. Those start and stop the Application Resources and send their addresses to the *Load Balancers*. Since all Offering Groups in this scenario are atomic the Load Balancers will perform a Round Robin selection [Kopo2, p. 28] of provisioned services. In Figure 6.2 the request flow between system elements is depicted. The response flow is omitted since it is the reverted request flow. The Provisioning Service and the Instances of the Hello Service in the separate Offering Groups may be accessed directly through Web service interfaces. Those are integrated into the Enterprise Service Bus through binding components and Web service interfaces offered by the Enterprise Service Bus itself. This way loose coupling [Joso7, p. 35] is guaranteed. It is especially important since it allows that all tenants access the same service offered by the Enterprise Service Bus and are routed according to their identifier to the Offering Groups handling their requests. A tenant does not know which concrete Offering Group handled the request.

6.1.1 Routing Components

The components responsible for routing are the Tenant Router and several Load Balancers. In Figure 6.2 it can already be suspected due to their interfaces that they utilize two different routing methods. In general they could be implemented using either one of the methods but both were implemented here for demonstration purposes.

The utilized Enterprise Service Bus was ServiceMix [Apa]. Internally it exchanges so called normalized messages [THW05, p. 13] between components allowing the usage of Enterprise Application Integration (EAI) patterns [HW04] to describe the behavior of the routing components.

Tenant Router

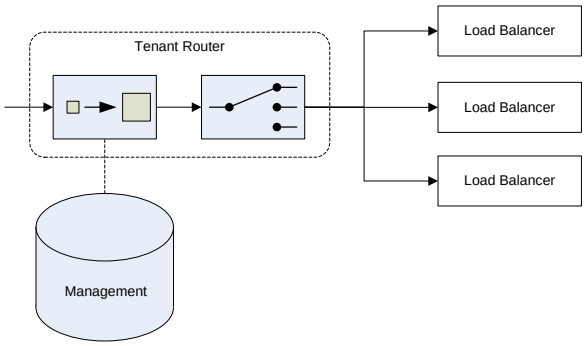

Figure 6.3: Tenant Router as EAI pattern

The Tenant Router follows the patterns for Content Enricher [HW04, p. 336 ff.] and Content Based Router [HW04, p. 230 ff.]. For every incoming normalized message the Tenant Router retrieves the Assurances for the corresponding tenant. Given the user distribution of this tenant's Usage Partition it calculates a random Offering Group to handle the request and enriches the header of the normalized message with that information. The content based router utilizes this header to route the message to the Load Balancer responsible for the target Offering Group. Such an implementation allows the Tenant Router to be stateless. Multiple instances may therefore be started to handle requests which is done automatically by ServiceMix. This results in a higher throughput but also in a non deterministic but randomized routing behavior. The ratio of user distributions stated in the Assurances will only be respected statistically for large number of requests.

Load Balancer

The Load Balancers follows the Dynamic Router pattern [HW04, p. 243 ff.]. They manage their routing information locally which is send to them using a Web service interface. The Provisioning Service provides the needed information after it started a service instance. If such an instance is stopped the address is removed respectively.
Since all resources are part of an Atomic Offering Group the Load Balancers use a Round Robin [Kop02, p. 28] distribution algorithm. In more complex environments a weighted distribution according to the temporal capabilities of managed resources has to be performed. Utilizing this approach the behavior of the Load Balancer is more predictable than that of the Tenant Router.

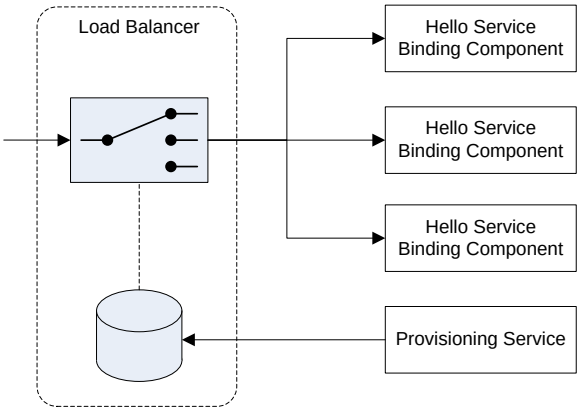

Figure 6.4: Load Balancer as EAI pattern

6.1.2 Service Interfaces

Listing 6.1 WSDL Port Type of the Hello Service

```
...
<wsdl:portType name="Hello">
       <wsdl:operation name="SayHello">
    <wsdl:input message="tns:SayHelloRequest"> </wsdl:input>
    <wsdl:output message="tns:SayHelloResponse"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
...
```

The interface of the Hello Service is very simple. It only offers one operation which has one input and one output message. Both messages only contain one element of type literal. The tenant identifier is used as input to this operation and routing is performed based on this information. The result string is used to send back information specifying which service instance in an Offering Group handled the request.

Listing 6.2 WSDL Port Type of the Provisioning Service

```
...
<wsdl:portType name="Set">
  <wsdl:operation name="setAmount">
    <wsdl:input message="tns:SetRequestMessage"> </wsdl:input>
    <wsdl:output message="tns:ResponseMessage"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
...
```

The interface of the Provisioning Service is very similar to that of the Hello Service. It is used to specify the necessary amount of resources. The Provisioning Service starts and stops them

and deploys their binding components to the Enterprise Service Bus. It only displays one operation to set the required amount of instances. Since it manages the amount of current instances internally it may compute the necessary actions from the input parameter.

Listing 6.3 WSDL Port Type of the Load Balancer Control Service

```
...
<wsdl:portType name="Add">
  <wsdl:operation name="addRoute">
    <wsdl:input message="tns:AddRequestMessage"> </wsdl:input>
    <wsdl:output message="tns:ResponseMessage"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="Remove">
  <wsdl:operation name="removeRoute">
    <wsdl:input message="tns:RemoveRequestMessage"> </wsdl:input>
    <wsdl:output message="tns:ResponseMessage"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
...
```

When the Provisioning Service started or stopped a Hello Service instance it must inform the Load Balancer since it manages addresses locally. Therefore the Load Balancer Control interface includes two operations to add and remove the addresses of the binding components. The response given by the Load Balancer Control Service states that the address was removed successfully or that it was not found in the rule database.

6.2 Management Life Cycle and System Component Interaction

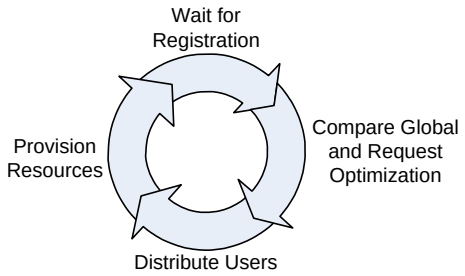

Figure 6.5: Management life cycle

The management system constantly performs a circular process which is depicted in Figure 6.5. After every registration global and request based optimization results are compared. To do so the optimization process extracts the Condensed View of the Resource Model from the database and transforms it into the Optimization View. This is then fed into the optimization algorithms for request as well as global optimization. Since a significant amount of time is needed to provision resources the result of global optimization will only be selected if a

certain threshold of cost difference is exceeded to avoid constant changes. After resources have been provisioned the users are distributed and the resulting Assurances are written back into the database. Considering that a global optimization has been performed prior to tenant four signing up the Assurances for the scenario are:

Tenant	Usage Partition	Offering Group	Amount
tenant1	tenant1_1	og1	30
tenant1	tenant1_1	og2	5
tenant1	tenant1_1	og3	5
tenant2	tenant2_1	og2	20
tenant3	tenant3_1	og3	5

Table 6.1: Assurances after first global optimization

The User Distribution algorithm ensured that the left over capabilities reside in the most expensive Offering Group by substituting as many users as possible into Offering Group Two. If a request based optimization is performed for the request of the fourth tenant one new instance in Offering Group Three will be provisioned. The distribution of already provisioned resources may not be changed and tenant four does not allow its users to be delegated. This optimization results in one more line being added to the Assurances:

Tenant	Usage Partition	Offering Group	Amount
tenant4	tenant4_1	og3	10

Table 6.2: Assurances after request optimization.

However one can easily see that users delegated into Offering Group Three should be moved to another Group since the users which have to be in Offering Group Three could be served by only one instance. Running another global optimization will result in users of Offering Group One being substituted into Offering Group Two since the maximum amount of resources for Offering Group One is reached. Therefore another instance is provisioned in Offering Group Two and one from Offering Group Three will be freed:

Tenant	Usage Partition	Offering Group	Amount
tenant1	tenant1_1	og1	30
tenant1	tenant1_1	og2	10
tenant2	tenant2_1	og2	20
tenant3	tenant3_1	og3	5
tenant4	tenant4_1	og3	10

Table 6.3: Assurances after second global optimization.

7 Summary

It has been pointed out how a Software as a Service vendor may successfully integrate and manage resources part of different computing environments. This is mandatory to offer customizable applications targeting different consumer markets. The need to optimize the user distribution in such a heterogeneous environment had to be addressed to reduce the cost of the overall system.
To achieve this goal an abstract Resource Model was defined which may be used to describe the entities part of a Software as a Service application. It enables the management of resources as well as the optimization of user distribution.
Finding an optimal distribution was observed to be an np-hard problem [Puno8, p. 11-11]. Local search algorithms based on Hill Climbing [Noro3, p. 111 ff.] and Simulated Annealing [Noro3, p. 115 - 116] were used to find an optimized distribution in an acceptable time frame. The complexity of the algorithms' input data was reduced by defining different views of the Resource Model. Information not needed for the specific tasks was hidden. Through these algorithms the optimal number of resources was computed. Concrete routing information was then obtained from an algorithm handling the user distribution.
Since local search methods do not guarantee to find the best result their performance was evaluated using a set of test cases. Small scenarios which allowed computing the best result by evaluating all possible user distributions were used. The best performing algorithms were examined further by comparing their results for larger test cases with each other. Smarter Simulated Annealing, an algorithms based on Simulated Annealing [Noro3, p. 115] with influences from Hill Climbing [Noro3, p. 111], performed best. As all other Simulated Annealing approaches, its performance is strongly affected by a static factor specified by the user. A method to determine this so called cooling rate was also covered.
The system architecture defined in this thesis allows to utilize the computed information for intelligent provisioning of resources and request routing. It is able to automatically allocate needed resources in the managed resource pools and routes customers' requests based on the optimized user distribution. During this process it is unknown to the costumer which resource handled his request.
A prototype was implemented using ServiceMix as an Enterprise Service Bus. It manages three resource pools. One service has been developed which is accessed by users. Custom routing components assign their requests to the resource pools.

7.1 Limitations and Outlook

While a lot of the challenges faced could be resolved there are some issues which may arise in the future. Additionally there are still some present limitations.

Automatic evaluation of service meta data It has been pointed out how an atomized comparison of service semantics and quality of services may be used to load balance users over a heterogeneous environment. Considering semantic comparison a lot of research is already being conducted. A quality of service model allowing automated evaluation has been formally defined. However to fully complete this task it is likely that this model has to be improved further.

Larger scenarios The Smarter Simulated Annealing algorithm has been shown to scale for up to 21 resource pools handling 200,000 users. In larger scenarios other search strategies might have to be evaluated. Allowing larger steps in the search scope might be one of them. Combining Simulated Annealing with a Bin Packing approach [FL87] [SLBC05, p. 34] might also lead to even more efficient algorithms.

More sophisticated cost model The cost function utilized is quite basic. Running costs such as for traffic or wearing of resources are not respected. A more complex cost model may however be integrated seamlessly since it would only affect the defined cost function.

Balancing the temporary load The distribution of users does not take timing into account. It could be reasonable that a tenant may specify time frames in which users access the Software as a Service application. This information could be respected during the optimization process. It could be used to even further improve the utilization of computing resources. Additionally the system size could be increased or decreased automatically for specific time frames.

Bibliography

[ADC05] ADC: *TIA-942 Data Center Standards Overview*. 2005 http://www.adc.com/Library/Literature/102264AE.pdf (Cited on pages 9 und 14)

[Ama] Amazon.com: *Elastic Compute Cloud EC2*. – http://aws.amazon.com/ec2/ (Cited on pages 10 und 15)

[Apa] Apache: *ServiceMix*. – http://servicemix.apache.org/home.html (Cited on page 67)

[CC06] Chong, F. ; Carraro, G.: Architecture Strategies for Catching the Long Tail. In: *MSDN Library, Microsoft Corporation* (2006). – http://www.ap-businessit.com/msforum/hdownload/Architecture%20Strategies_for_Catching_the_Long_Tail.pdf (Cited on pages 9, 10 und 12)

[Ciu09] Ciurana, E.: *Developing with Google App Engine*. Apress, 2009 (Cited on page 15)

[DR07] Dittner, R. ; Rule, D.: *Best Damn Server Virtualization Book Period: Including Vmware, Xen, and Microsoft Virtual Server*. Syngress Press, 2007 (Cited on pages 7 und 11)

[Erl05] Erl, T.: *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice-Hall, 2005 (Cited on page 11)

[FL87] Friesen, D.K. ; Langston, M.A.: Bin packing: On optimizing the number of pieces packed. In: *BIT Numerical Mathematics* 27 (1987), Nr. 2, S. 148–156. – Springer (Cited on page 72)

[Got07] Gottfrid, D.: *Self-service, Prorated Super Computing Fun!* New York Times Blog, 2007. – http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/ (Cited on page 10)

[HS05] Huang, K.W. ; Sundararajan, A.: Pricing models for on-demand computing. In: *Working Paper CeDER-05-26* (2005). – Center for Digital Economy Research Leonard N. Stern School of Business, New York University (Cited on page 12)

[HW04] Hohpe, G. ; Woolf, B.: *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley, 2004 (Cited on page 67)

[JMS02] Jin, L. ; Machiraju, V. ; Sahai, A.: Analysis on service level agreement of web services. In: *HP Technical Report* (2002). – http://www.hpl.hp.com/techreports/2002/HPL-2002-180.pdf (Cited on page 7)

[Jos07] Josuttis, N.: *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 2007 (Cited on page 66)

[KGV83] Kirkpatrick, S. ; Gelatt, C. D. ; Vecchi, M. P.: Optimization by Simulated Annealing. In: *Science, New Series* 220 (1983), Nr. 4598, 671–680. http://www.jstor.org/stable/1690046. – American Association for the Advancement of Science (Cited on page 44)

[Kop02] Kopparapu, C.: *Load balancing servers, firewalls, and caches*. Wiley, 2002 (Cited on pages 7, 66 und 67)

[LK08] Leymann, F. ; Karastoyanova, D.: *Lecture: Web-based application integration*. 2008. – http://www.iaas.uni-stuttgart.de/lehre/vorlesung/2008_ws/vorlesungen/wbai/ (Cited on page 16)

[LM09] Leymann, F. ; Mietzner, R.: *Applications in the Cloud*. ITPC Cloud Day, 2009 (Cited on page 13)

[LR99] Leymann, F. ; Roller, D.: *Production workflow: concepts and techniques*. Prentice Hall, 1999 (Cited on page 17)

[Man89] Manber, U.: *Introduction to algorithms: a creative approach*. Addison-Wesley, 1989 (Cited on pages 34, 48 und 51)

[Mur08] Murty, J.: *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly Media, Inc., 2008 (Cited on page 15)

[MvW+09] Mietzner, R ; van Lessen, T. ; Wiese, A. ; Wieland, M. ; Karastoyanova, D. ; Leymann, F.: Virtualizing Services and Resources with ProBus: The WS-Policy-Aware Service and Resource Bus. In: *Proceedings of the 7th International Conference on Web Services (ICWS) 2009*, IEEE Computer Society, Juli 2009 (Cited on page 17)

[New] New York Times: *Article Archive*. – http://www.nytimes.com/ref/membercenter/nytarchive.html (Cited on page 10)

[NLv08] Nitzsche, J. ; Leymann, F. ; van Lessen, T.: Formalising Message Exchange Patterns using BPEL Light. In: *IEEE International Conference on Services Computing* 1 (2008), S. 353–360. – http://doi.ieeecomputersociety.org/10.1109/SCC.2008.97 (Cited on page 17)

[Nor03] Norving, P.: *Artificial intelligence: a modern approach*. Prentice-Hall, 2003 (Cited on pages 36, 37, 41, 42, 43, 45 und 71)

[NWG+08] Nurmi, D. ; Wolski, R. ; Grzegorczyk, C. ; Obertelli, G. ; Soman, S. ; Youseff, L. ; Zagorodnov, D.: The Eucalyptus Open-source Cloud-computing System. In: *Proceedings of Cloud Computing and Its Applications* (2008) (Cited on page 15)

[OAS07] OASIS: *Standard: Web Services Business Process Execution Language Version 2.0*. 2007. – http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (Cited on page 17)

[Puno8] Puntambekar, A.A.: *Analysis And Design Of Algorithms*. Technical Publications, 2008 (Cited on pages 34 und 71)

[PW05] Parastatidis, S. ; Webber, J.: *The SOAP Service Description Language 1.3*. 2005. – http://www.ssdl.org/docs/v1.3/html/SSDL%20v1.3.html (Cited on page 17)

[Rap04] Rappa, M.A.: The utility business model and the future of computing services. In: *IBM Systems Journal* 43 (2004), Nr. 1 (Cited on page 12)

[RH08] Richard, M.J. ; Hoover, N.: *Guide To Cloud Computing*. 2008. – http://www.informationweek.com/story/showArticle.jhtml?articleID=208700713 (Cited on page 10)

[Roh48] Rohrbach, H.: Die Anzahl der Zahlen mit vorgegebener Quersumme. Helmut Hasse zum 50. Geburtstag. In: *Mathematische Nachrichten* 1 (1948), Nr. 6, 357–364. http://dx.doi.org/10.1002/mana.19480010606. – WILEY-VCH Verlag GmbH Co (Cited on page 56)

[Sala] Salesforce.com: *Force.com - Platform*. – http://www.salesforce.com/platform/what-is-it.jsp (Cited on page 15)

[Salb] SalesForce.com: *Salesforce CRM*. – http://www.salesforce.com/crm/ (Cited on page 15)

[SLBC05] Simchi-Levi, D. ; Bramel, J. ; Chen, X.: *The logic of logistics: theory, algorithms, and applications for logistics and supply chain management*. Springer, 2005 (Cited on pages 31 und 72)

[Sny05] Snyman, J.A.: *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Springer, 2005 (Cited on page 53)

[Ste46] Stevens, S.S.: On the theory of scales of measurement. In: *Science* 103 (1946), Nr. 2684, S. 677–680. – http://courses.essex.ac.uk/gv/gv200/stevensarticle.pdf (Cited on page 22)

[TDGS06] Taylor, I.J. ; Deelman, E. ; Gannon, D.B. ; Shields, M.: *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 2006 (Cited on page 17)

[THW05] Ten-Hove, R. ; Walker, P.: *Java Business Integration (JBI) 1.0-JSR 208 Final Release*. Technical report, Sun Microsystems, Inc., 2005. – http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html (Cited on page 67)

[W3C01] W3C: *Note: Web Services Description Language (WSDL) 1.1*. 2001. – http://www.w3.org/TR/wsdl (Cited on page 16)

[W3C07] W3C: *Recommendation: Web Services Policy 1.5 - Framework*. 2007. – http://www.w3.org/TR/2007/REC-ws-policy-20070904 (Cited on page 16)

[WCL+05] Weerawarana, S. ; Curbera, F. ; Leymann, F. ; Storey, T. ; Ferguson, D.F.: *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall, 2005 (Cited on pages 11, 14, 16 und 17)

All links were last followed on June 30, 2009.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Christoph Alexander Fehling)