A Framework for Exchange-Based Trading of Cloud Computing Commodities

Dissertation

an der Fakultät für Mathematik, Informatik und Statistik der Ludwig-Maximilians-Universität München

eingereicht von

Johannes R. Watzl

13. Dezember 2013

1. Gutachter: Prof. Dr. Dieter Kranzlmüller
2. Gutachter: Prof. Manish Parashar, Ph. D.

Tag der Einreichung: 13. Dezember 2013

Tag der Disputation: 7. April 2014

Contents

List of Figures

List of Tables

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Name, Vorname

....	....
Ort, Datum	Unterschrift Doktorand/in

Formular 3.2

Abstract

Cloud computing is a paradigm for using IT services with characteristics such as flexible and scalable service usage, on-demand availability, and pay-as-you-go billing. Respective services are called cloud services and their nature usually motivates a differentiation in three layers: Infrastructure as a Service (IaaS) for cloud services offering functionality of hardware resources in a virtualised way, Platform as a Service (PaaS) for services acting as execution platforms, and Software as a Service (SaaS) representing applications provided in a cloud computing way.

Any of these services is offered with the illusion of unlimited scalability. The infinity gained by this illusion implies the need for some kind of regulation mechanism to manage supply and demand. Today's static pricing mechanisms are limited in their capabilities to adapt to dynamic characteristics of cloud environments such as changing workloads. The solution is a dynamic pricing approch compareable to today's exchanges. This requires comparability of cloud services and the need of standardised access to avoid vendor lock-in. To achieve comparability, a classification for cloud services is introcuced, where classes of cloud services representing tradable goods are expressed by the minimum requirements for a certain class.

The main result of this work is a framework for exchange-based trading of cloud computing commodities, which is composed of four core components derived from existing exchange market places. An exchange component takes care of accepting orders from buyers and sellers and determines the price for the goods. A clearing component is responsible for the financial closing of a trade. The settlement component takes care of the delivery of the cloud service. A rating component monitors cloud services and logs service level agreement breaches to calculate provider ratings, especially for reliability, which is an important factor in cloud computing.

The framework establishes a new basis for using cloud services and more advanced business models. Additionally, an overview of selected economic aspects including ideas for derivative financial instruments like futures, options, insurances, and more complex ones is provided. A first version of the framework is currently being implemented and in use at Deutsche Börse Cloud Exchange AG.

Abstract - Deutsch

Cloud Computing repräsentiert eine neue Art von IT-Diensten mit bestimmten Eigenschaften wie Flexibilität, Skalierbarkeit, ständige Verfügbarkeit und nutzungsbezogene (*pay-as-you-go*) Abrechnung. IT-Dienste, die diese Eigenschaften besitzen, werden als Cloud Dienste bezeichnet und lassen sich in drei Ebenen einteilen: Infrastructure as a Service (IaaS), womit virtuelle Hardware Ressourcen zur Verfügung gestellt werden, Platform as a Service (PaaS), das eine Ausführungsumgebung darstellt und Software as a Service (SaaS), welches das Anbieten von Applikationen als Cloud Dienst bezeichnet. Cloud Dienste werden mit der Illusion unendlicher Skalierbarkeit angeboten. Diese Unendlichkeit erfordert einen Mechanismus, der in der Lage ist, Angebot und Nachfrage zu regulieren. Derzeit eingesetzte Preisbildungsmechanismen sind in ihren Möglichkeiten beschränkt sich auf die Dynamik in Cloud Umgebungen, wie schnell wechselnde Bedarfe an Ressourcen, einzustellen. Eine mögliche Lösung stellt ein dynamischer Preisbildungsmechanismus dar, der auf dem Modell heutiger Börsen beruht. Dieser erfordert die Standardisierung und Vergleichbarkeit von Cloud Diensten und eine standardisierte Art des Zugriffs. Um die Vergleichbarkeit von Cloud Diensten zu erreichen, werden diese in Klassen eingeteilt, die jeweils ein am Börsenplatz handelbares Gut darstellen.

Das Ergebnis dieser Arbeit ist ein Rahmenwerk zum börsenbasierten Handel von Cloud Computing Commodities, welches aus vier Kernkomponenten besteht, die von existierenden Börsen und Rohstoffhandeslplätzen abgeleitet werden können. Die Börsenkomponente nimmt Kauf- und Verkaufsorders entgegen und bestimmt die aktuellen Preise der handelbaren Cloud Rohstoffe. Die Clearing Komponente stellt die finanzielle Abwicklung eines Geschäftes sicher, das Settlement ist für die tatsächliche Lieferung zuständig und die Rating Komponente überwacht die Cloud Dienste im Hinblick auf die Nichteinhaltung von Service Level Agreements und vor allem deren Zuverlässigkeit, die einen wichtigen Faktor im Cloud Computing darstellt.

Das Rahmenwerk begründet eine neue Basis für die Cloudnutzung und ermöglicht fortgeschrittenere Geschäftsmodelle. In der Arbeit wird weiters ein Überblick über ökonomische Aspekte wie Ideen zu derivaten Finanzinstrumenten auf Cloud Computing Commodities gegeben. Dieses Rahmenwerk wird derzeit an der Deutsche Börse Cloud Exchange AG implementiert und bereits in einer ersten Version eingesetzt.

Chapter 1

Introduction

This chapter provides an overview of development and utilisation of cloud computing until today and motivates the creation of a framework for exchange-based trading of cloud computing commodities. A general problem of cloud computing, the lack of a regulation mechanism for supply and demand of cloud commodities, is introduced, and relevant terms are defined before the research questions addressed by this thesis are specified.

1.1. Evolution of cloud computing

Information Technology (IT) has become a ubiquitous part of our everyday life, be it work or leisure, from high performance computing systems in industry and academia to location-aware applications running on handheld devices. Today, smart phones are a common utility and people everywhere use so-called *apps* to utilise services provided over the Internet. This was unthinkable ten years ago. Even the Internet as we see it today was only a dream 25 years ago. The technological development has enabled new ways of interacting with the computer through new kinds of applications.

In a simplified view, in any of these applications, consumers use IT resources in the form of hard- and software via so-called IT-services. More recently, a new quality of IT-services has emerged under the term *cloud computing*, which is covered in the media almost every day. Due to its characteristics and possibilities, it is of massive interest for business, industry, academia, and even personal usage.

Cloud computing has happened due to the availability of today's IT infrastructures and in practice due to two main technological triggers:

- Access to ubiquitous broadband network
- Virtualisation of resources

Over the last few decades, an enormous growth in network speed and bandwidth has been observed. Following Nielsen's law, network bandwidth has been doubled every 24 month [63]. This steady growth has initiated a number of benefits for the consumers, originally starting from the model of a host-based system offered by a provider to the broad availability of Application Service Providing (ASP). During the 1990s, the availability of fast networks enabled the provisioning of IT-services similar to the electrical power grid in an approach called *grid computing* [43]. Simplified, the general idea behind grid computing is resource sharing between (mostly) scientific institutions and their resource providers. This has initiated many successful projects in science and research and drove the evolution of our knowledge society. Cloud computing is, in many aspects, a simplified and more abstract version of grid computing, more user-centric and originating from the business domain.

The second key breakthrough for cloud computing is virtualisation, which is the ability to share physical hardware between multiple (virtual) machines on one (physical) host. Virtualisation requires a software layer (hypervisor) to realise provisioning of multiple instances of virtual hardware and the translation between the physical hardware and the virtual machines. The original idea of virtualisation emerged with the usage of emulators to execute machine code on hardware different from the original hardware it was written for. Partial virtualisation enables the address space virtualisation that provides a separate address space for virtual machines and acts as the basis for time-sharing systems of mainframes such as the IBM 360 [71]. Full virtualisation (hypervisor runs on the physical hardware) and para-virtualisation (hypervisor runs on top of an operating system) has been introduced with the availability of more powerful hardware. The definition and a detailed overview of the of virtualisation and its different types can be found in [58]. Today's hardware is so powerful that multiple instances of virtual machines can be executed in parallel. As such, virtualisation plays a crucial role, especially in computing centres, when considering the space and power needed for physical machines. With virtualisation, multiple virtual machines and especially also storage can be provided on a limited number of physical instances. (Typically, large cloud providers utilise one physical machine for around 20 virtual machines. [12])

With broadband networks and virtualisation, cloud computing can be realised as the *Everything-as-a-Service* landscape observed today. The intention to offer services for everything - from resources to personal interactions - is referred to as Everything-as-a-Service as stated in [16] enables the dynamic usage of virtual hardware infrastructures, platforms, and applications and pricing based on the actual consumption.

However, there are additional benefits for consumers of cloud computing, especially when considering the financial aspects. Cloud Computing offers the ability to minimise capital expenditures (CAPEX) and turning them into operational expenditures (OPEX). This means that the entry level for using IT-services is much smaller compared to earlier days, where possibly huge investments in IT-infrastructures were needed at the startup of a new endeavour. This became even more important during the financial crisis in 2007, and consequently, cloud computing has profited a lot of the need for IT-resources when budgets for investments are limited. In fact, the financial crisis can be seen as an accelerator for the global cloud adoption [66]. In [62] as well as in the cloud computing definition (see Section 1.3) published by BITKOM, the business aspect of cloud computing is outlined in more detail.

The interest in cloud computing is also visible in the Gartner hype cycle of 2009 [1], where cloud computing appears on the very top of the curve - the peak of inflated expectations. The Gartner hype cycle describes the development of technologies over time, from the technology trigger to the plateau of productivity [57]. Today, Gartner offers even a separate cycle for cloud computing and associated technologies [8]. As such, we see cloud computing not only as one technology but as a number of different services and applications provided to consumers from all domains. In fact, not only newly created IT-services have been introduced on the cloud market. Instead, many existing online applications or IT-services have renamed themselves under the cloud umbrella. This renaming is often referred to as *cloud washing* [68] and can be observed to be performed by various companies e.g. Oracle Exalogic Elastic Cloud [51].

Remark: The reason for not taking into account academic cloud computing resources in this work is that publicly funded resources in general (whether scientific or not) are often not allowed (by law) to be sold and therefore not allowed to be traded. Grid infrastructures - as an example - are usually set up in scientific environments which are publicly funded, and grid resources are mostly *exchanged* freely without costs. Therefore an adoption of a business-centric marketplace is not possible for the academic domain today.

1.2. Motivation and problem statement

With clouds available today and the continued expectations in clouds from both, providers and consumers, we can identify a number of issues, which have not or at least not sufficiently been addressed by today's cloud environments:

- Load balancing and regulation of cloud supply and demand
- Static and inflexible pricing models of cloud services
- Absence of interoperability standards and vendor lock-in

The strong relation between the usage of cloud computing and its financial aspects, both as a source of costs (for the consumer) and as a source of potential income (for the provider), has generated a lot of interest from economics. The provisioning and utilisation of cloud resources needs to be optimised for a variety of different aspects. On the one hand, resource providers are forced to perform over-provisioning to ensure elasticity. This means that they have to keep spare capacity in order to fulfil a consumer's resource request. An example is the supply of web-shops during Christmas time [18] (e.g. Christmas peak at Amazon). As a consequence, providers are often not able to fully utilise and monetise idle cloud resources, which are waiting for customers but still consume energy and cooling. On the other hand, the increased cloud adoption may generate bottlenecks for consumers, when more resources are requested than available. The scalability of clouds seem to be infinite to the consumer, while in practice they are limited by the amount of physical hardware in the providers' resource centres. Providers are able to meet the consumers' demands as long as there are enough resources available. Finding the right balance between provisioning and consumption is a difficult task.

Related to this, today's cloud providers mostly use static pricing models. This means that resources have a pre-defined and fixed price, for example a certain price for running a virtual machine or a fixed price for a certain amount of storage per time unit. The price does not change depending on parameters given by the environment of the cloud provider. Depending on the interest and needs of consumers, cloud providers may need more or less resources to fulfil the demand. Additionally, operating the resources depends on a number of environmental parameters, e.g. costs for power and cooling, which may vary over time. In practice, cloud providers may want to adapt the pricing corresponding to specific characteristics of their environment and the actual demands from the consumers.

A major problem for the consumers of cloud services is the lack of standards and interoperability between clouds. This makes it difficult to compare different cloud offers and especially switching between different cloud providers. Examples are missing standards for describing cloud services for measuring cloud resource performance, for the usage of the cloud infrastructure itself, and for service level contracts. In practice, individually different cloud services offer different descriptions with different performance characteristics, using different contracts which require long negotiation processes. This incompatibility is further increased once a specific and proprietary offer has been chosen. At this point, cloud consumers experience a vendor lock-in, which means that today it is nearly impossible to switch between cloud providers without huge efforts in data transfer and data format conversion.

Historically, a similar situation can be observed in the trading of agricultural goods in the mid of the 19th century between different cities and countries. In [34] Doering describes different measures for basic measurands like weight, length, and currency. At that time measurands have even been applied for certain products or goods with similar chemical properties. Every city had its own rules for square measure, the amount of liquids of different types, and wood among several others and provided conversion tables. For example, 100 lb in Munich equalled 99.998 lb in Vienna or even 119.782 lb in Frankfurt. In an environment like this the trading of goods is hindered by the overhead of conversion and the complexity of comparing goods with different origin. The standardisation of these measures reduced the overhead necessary for the conversion and therefore enabled a faster and transparent trading. The complexity of cloud services hinders the creation of unique and well-defined conversion tables.

Comparing this historical example with today's cloud computing landscape, we identify a vision for future cloud computing with the creation of an exchange for cloud computing commodities. In practice, we need a trading platform for cloud resources based on standardised cloud commodities and a model similar to the commodity market. With such a platform, providers and consumers as buyers and sellers are able to meet at a marketplace to trade standardised, comparable and compatible cloud resources. Market participants can establish standardised contractual relationships easily on the fly, replacing the current static and inflexible pricing models with dynamic supply and demand based pricing models. This vision defines the goal of the thesis as the creation of a framework to regulate supply and demand within a cloud market and to allow more flexible trading mechanisms for cloud computing commodities. This framework needs to provide the following functionality:

- Trading of standardised cloud commodities between multiple market participants
- Regulation of cloud resource consumption based on supply and demand

The scientific questions associated with this functionality of a framework for exchange-based trading of cloud computing commodities can be formulated as follows:

1. How to achieve comparability of cloud services across different providers?
2. How to define dynamic pricing of cloud services based on quality and quantity?
3. How to regulate cloud service offerings corresponding to supply and demand?
4. How to enable trading of cloud service offerings at an exchange-based marketplace?
5. How to introduce additional parameters such as provider ratings into the pricing model?

These questions are addressed throughout this thesis from a computer science point of view, abstracting the economics side of the questions as much as possible. The intention is to provide an IT-based framework, which enables the above mentioned behaviour for different economic scenarios. The implementation of a framework as described in this thesis represents a proof-of-concept for the theoretical model from the standpoint of a computer scientist, not from the view of economic studies which are required additionally.

1.3. Defining cloud computing

With the above mentioned problem statement in mind, we need to define cloud computing and cloud computing services. With the current hype, it is very difficult to find a generally valid and useful definition. In fact, there are various definitions available for both terms, sometimes even conflicting and distorting the picture, especially for cloud consumers.

A number of exemplary definitions in literature are: [15], [74], [75]. In this work we want to focus on two important definitions: a more business-centric and a more technical definition.

The more business-centric definition is provided by BITKOM [1] [62]:

- *Cloud computing describes a form of flexible and tailored usage of IT resources to fit market needs.*
- *These resources are provided in real-time as a service over the internet utilising a pay by use billing model.*
- *Cloud computing enables the turning of capital expenditures to operational expenditures.*
- *The IT resources regarding this definition are related to*
  - *applications*
  - *platforms for application development and execution*
  - *basic infrastructure*

A more technical definition - covering most of the BITKOM definition except the economic aspects (fit to market needs and the ability to turn capital expenditures into operational expenditures) - is provided by NIST [2] (National Institute of Standards and Technology). It seems most useful and will therefore be used in the remainder of this work. It can be found in Appendix A. From the NIST definition [60], we identify the following essential characteristics for our work: Cloud computing is

- provided as three different instantiations Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).
- offered in an on-demand self-service, always available/accessible, and flexible (scale in, scale out) way billed in a pay-as-you-go model where only the actually consumed resources are paid.

[1]Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (http://www.bitkom.org/)
[2]National Institute of Standards and Technology (http://www.nist.gov/)

The three different instantiations of cloud computing are the main representatives of the *everything-as-a-service* landscape introduced above. IaaS means the provider offers virtual hardware resources, such as space to run virtual machines or storage capacity, PaaS describes an execution environment for applications, and SaaS is a way to provide applications .

The essential characteristics of clouds as defined by NIST provide more details on the capabilities of cloud computing. On-demand self service describes the way a consumer can manage the cloud resources. Cloud resources feature broad network access making the cloud resources available to different kinds of platforms. Providers make use of resource pooling to be able to serve multiple consumers.

Consumers are able to scale in/out cloud resources at any time. Cloud computing services are measured services. This means they implement mechanisms to meter resource usage and react appropriately in the case of changes. This capability also allows to provide reports to providers and consumers.

For cloud computing, different deployment models exist. These can be traced back to security concerns or requirements related to security, policy or compliance. The most restrictive deployment model is the *private cloud*. The cloud resources are only available to one organisation. A more open model is the model of *community clouds*, where cloud resources are shared among certain communities of consumers (organisations) with similar requirements. Cloud resources provided as *public cloud* are openly accessible, whereas *hybrid clouds* describe a combination of two or more of the previously mentioned deployment models in order to e.g. cover peak loads.

With cloud computing described as above, we identify cloud computing services as a special form of a standard IT-service corresponding to its usage in cloud computing environments. A useful definition of the term IT-service is provided by the IT Infrastructure Library (ITIL). ITIL represents a set of books showing best practices, and describing processes and tasks for managing IT infrastructures and align the infrastructures and their management with business needs. The ITIL IT-Service Management Framework can be seen as one of the standards for IT-Service Management at the moment.

In the words of ITIL, an IT-service is defined as as:
A set of related functions provided by IT systems in support of one or more business areas, which in turn may be made up of software, hardware and communications facilities, perceived by the customer as a coherent and self-contained entity. An IT service may range from access to a single application, such as a general ledger system, to a complex set of facilities including many applications, as well as office automation, that might be spread across a number of hardware and software platforms. ([9])

A cloud service is then an IT-Service as described above with the essential characteristics of the cloud computing - on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service - within one of the three service models of cloud computing - IaaS, PaaS, SaaS, and deployed as private, community, public or hybrid cloud. Cloud services are offered with a certain Service Level Agreement (SLA). A very important part of SLAs are Key Performance Indicators (KPIs) and corresponding thresholds.

1.4. Research contributions

By developing the framework for exchange-based trading of cloud computing commodities in order to regulate supply and demand of cloud resources, we contributed the following findings to the domain of distributed systems research:

- Quantification of cloud services: We developed a way to describe cloud services using specific characteristic parameters. The result is a meta-model describing parameters such as technical specification, performance, and location which allows to compare offerings of cloud services between different providers.
- Classification of cloud services: Using the quantitative parameters, we are able to compare cloud service offerings from different providers. The classification itself provides a possibility to order services according to their specific characteristics, making it able to trade them on an exchange.
- Establishment of an exchange-based trading framework: With standardised cloud commodities available, we developed a platform for trading and the required components in this framework. The exchange is comparable to typical commodity markets but using cloud commodities as tradable goods.
- Rating of cloud services: The quality of cloud offers and their conformance to SLA parameters allows rating of cloud computing providers. This can be used as a basis for consumers to assess the quality of different offers and to use this information as a selection criteria. The rating mechanism itself is a dedicated part of the framework.

The basis of this thesis is the introduction of comparability of cloud services. This is used for creating classes of different cloud computing services, the so-called cloud computing commodities, which are described by a set of parameters with measurable guaranteed performance and pre-defined service levels. The cloud computing commodities can then be traded on the exchange similar to traditional models, using a secure way for processing transactions and sophisticated mechanisms for rating of providers and their cloud offerings. The centralised rating mechanism is complemented by a peer-to-peer trust mechanism, which takes into account feedback from consumers. With the cloud computing definition introduced in this work, the comparability of cloud services offered by different providers is ensured, the use of standards is enforced, and the trading on an exchange is enabled.

Remark: C^2EX Compute Commodities Exchange - US Patent

Parts of the concept as formulated in this thesis has been filed as part of a US patent. This patent covers the concept of the framework including the standardisation of cloud computing commodities, the possibility to create complex financial instruments based on cloud computing commodities, and provider ratings. It has been filed as US patent 13/864,880 [42] together with Shawn P. Findlan, West Avenue Capital, LLC by ST. ONGE STEWARD JOHNSTON & REENS LLC (Attorneys for Applicants: Wesley W. Whitmyer, Jr., Registration No. 33,558 and Benjamin J. Lehberger, Registration No. 56,217).

In order to protect the intellectual property, no scientific papers or other publications have been published on this subject before this work. ST. ONGE STEWARD JOHNSTON & REENS LLC keeps a track record of the work performed (including email and conversation logs) from the beginning as an evidence for being able to claim an earlier invention date.

As patents have the intention to cover the field of the invention in a very broad way, the general concepts of the idea have been filed. The scientific aspects of the framework are covered by this thesis, whereas the patent focuses on saving the idea in case of a future commercialisation.

To provide transparency in terms of plagiarism, a number of claims referring to parts of this work are listed in the following, starting with a citation of the first claim - *field of the invention*.

- *Claim [0001] The present invention relates generally to a computing commodities exchange, and more specifically, relates to a cloud computing resource commodities exchange where suppliers and consumers can contract for available cloud computing resources. (cited from [42]*

- Claim [0009] introduces the idea of an exchange based market for cloud computing resources.

- Claim [0014] mentions the possibility to trade OTC (Over The Counter) with or without negotiation.

- In Claim [0020] the core components of the framework, namely exchange, clearing, settlement, and rating are listed. These are described in detail together with the informations exchanged between them in Section 3.2.

- Brokers are introduced in Claim [0015].

- Claim [0024] points out that the settlement delivers credentials. The process of secure clearing and settlement is shown in Section 3.2.6.

- The need of standardised commodities and parameters for their description is mentioned in Claim [0027]. Chapter 4 defines cloud computing commodities from a scientific perspective introducing a meta model for the parameters to describe these commodities.

- The trading of benchmark units are introduced in Claim [0028]. An approach for measuring cloud computing commodities based on benchmarks can be seen in Chapter 4.
- Claim [0030] mentions the different types of exchange cloud computing commodities or derivatives. A detailed description and a relation to energy markets is presented in Chapter 3 and Section 3.1.5.
- Claim [0037] and Claim [0044] introduce ratings of providers and effects on prices.
- Claim [0045] introduces the concept of subcommodities. For provider ratings a model combining centralised rating agencies with peer-to-peer ratings from consumers is presented in Section 4.4. The effect on prices of cloud computing commodities is shown in Section 4.4.3 together with the introduction of subcommodities.

1.5. Thesis roadmap

The thesis is structured in seven chapters. Chapter 1 motivates the work and gives an overview of the research field. Definitions for the terms used in the thesis are given and the research questions based on the problem the thesis intends to solve are introduced.

In Chapter 2, scenarios motivating the creation of a framework for exchange-based trading of cloud computing commodities are presented starting with a basic cloud usage scenario and specialised scenarios that can be derived from the basic cloud usage scenario. After the formulation of the requirements for the framework, Chapter 2 gives an overview of related work approaches and explains their drawbacks by analysing the requirements fulfilled by each of the presented approaches.

Chapter 3 starts with a detailed introduction of exchanges and provides an overview of the different types of exchanges currently in place and the products, which can be traded on these exchanges. The core part of Chapter 3 presents the model for exchange-based trading of cloud computing commodities. The different roles and entities are described in detail, as well as the four core components of the framework exchange, clearing, settlement, and rating. Diagrams are shown to illustrate the relationships and information flows between the entities of the model.

Cloud computing commodities are defined in Chapter 4 diversifying them from cloud services. The model for specifying cloud computing commodities by different parameters is presented in addition to details of the parameters necessary. To be able to measure the performance of a cloud computing commodity, methods to quantify cloud services are shown before introducing a concept for rating cloud providers and their services in a centralised and a peer-to-peer manner utilising web-of-trust ideas.

Chapter 5 presents the implementation of the framework that has been done as a proof-of-concept.

In Chapter 6 the simulations run are presented together with an evaluation and discussion of the results gained by the simulations.

The final chapter - Chapter 7 - concludes the thesis providing an outlook and an overview of future work together with a reflection of further economic aspects as economic outlook of the thesis.

Chapter 2

Cloud Usage Scenarios and Related Work

This chapter describes usage scenarios in current cloud computing infrastructures, providing an overview of state-of-the-art approaches in this field and motivates the development of a framework for exchange-based trading of cloud computing commodities.

Starting from a generic cloud computing usage scenario, specialised scenarios are derived including the usage of aggregated services, the usage of improved services, resource balancing between data centres (cloud providers), and a scenario of switching cloud providers. After that, the requirements for a system able to solve the problem presented in Chapter 1 are given.

Related work provides different ways to find partial answers to the questions identified in Section 1.2, combining distributed computing and economic models. The state-of-the-art work shown is complemented by the description of Amazon Spot Instances - a commercial approach in this area but with the limitation of taking into account only one provider. The presented approaches are discussed and matched to the requirements identified previously. The chapter closes with a discussion of this thesis in comparison to related work.

2.1. Cloud computing usage

Before presenting the usage scenarios, the terms *provider* and *consumer*, as used in the scenarios, are explained together with a specification of the types of contracts. In the context of cloud computing, these two parties are necessary to establish a cloud usage relationship:

- A provider offers a cloud service: Hardware resources or services necessary to offer the cloud service are owned by the provider. Providers charge the consumers for the cloud service usage.
- A consumer uses a cloud service: The consumer connects to the provider's resources. Consumers usually have to pay the costs generated by using the service.

Remark: Payment for cloud services in the academia or research domain may be different, because dedicated service providers may be established for scientific utilisation, such as a university's own computing centre. In this case, the payment arrives via the funding clauses of the computing centre.

A cloud usage relationship between a provider and a consumer is a contract based relationship. Currently, a differentiation can be made between two types of contracts for the usage of cloud services. This differentiation relates to the way the contract is established.

- Provider defined contracts
- Individually defined or negotiated contracts

At the moment (public) cloud services are offering a contract defined by their provider, who defines the terms relevant for using the cloud service including a service level agreement (in general very simple), which has to be accepted by the consumers. If the consumer does not agree with the terms, the SLA or parts of it, the consumer cannot use the service.

Individually defined or negotiated contracts are set up bilaterally between the provider and the consumer. An individual set of terms including an individual SLA is defined for the usage of a certain service. Normally, this type of contract is established with consumers intending to use a large number of resources for a longer time period. The process of negotiating this form of contract can take weeks or even months.

Contracts may contain certain information such as parameters describing the service, legal definitions, pricing models, service level requirements including service quality and penalties in case a service quality requirement is not met.

2.1.1. Cloud usage scenarios

Contract based cloud usage relationships between provider and consumer are presumed for the following cloud usage scenarios representing different ways of how cloud computing is used at the moment. The six scenarios considered in this chapter are:

1. *Basic scenario:* One provider is offering a cloud service used by one consumer.
2. *Aggregated services scenario:* A cloud service is aggregated by other cloud services. In this case providers can also be consumers of cloud services.
3. *Enhanced services scenario:* Cloud providers can extend the functionality of services from other providers with own resources and/or functionality.
4. *Execution platform for component-based applications scenario:* A cloud service offers

functionality to compose applications by combining application components. The resulting composed applications are executed on a PaaS.

5. *Resource balancing scenario:* Data centres can run into resource bottlenecks when more capacity is requested than the capacity available.
6. *Provider switching scenario:* The responsibilities in terms of data transfer after switching cloud providers are not clearly defined.

The scenarios can be categorised showing different views. Scenarios 1-4 offer both, a provider- and a consumer-centric view. Scenario 5 represents the view of providers offering cloud services hosted in their data centres and Scenario 6 shows a view on the consumers' data and its transfer.

1. Basic scenario

The basic scenario shows a setup with a provider offering a cloud service that is used by a consumer. In Figure 2.1 this scenario is depicted with a consumer on the left accessing a cloud service *S1* offered by a provider (on the right hand side of the figure). The provider runs a data centre (illustrated by the grey servers) and offers two cloud services *S1* and *S2*. In general, a provider might offer more than one service. In the case of this scenario, the provider owns the resources necessary to run the cloud services offered.

An example for this scenario is Amazon Web Services (AWS) offering compute and storage resources as EC2 and S3 .

The consumer sees the services offered by the provider and might have information on the provider's resources (e.g. data centre location). In general, the consumer has very limited information on the resources behind the cloud service.

The provider sees the consumer making use of its resources offered through the cloud service *S1* and *S2* and in this case owns the resources.

The name *basic scenario* has been chosen as the following scenarios can be derived from this scenario as special cases.The following scenario introduces providers who do not own resources. Instead, they are buying resources from other providers in order to offer them.

2. Aggregated services scenario

Cloud services can be composed by aggregating cloud services that might be also offered by other providers - e.g. in order to avoid or limit own resource costs. Cloud services from other providers can also be aggregated with own cloud services (Figure 2.3). Figure 2.2 shows a setup with five cloud providers. Providers 1, 3, and 4 run their offered cloud services on their own resources, whereas Provider 2 and Provider 5 do not own any resources. Provider 2 and

5 aggregate (depicted by clouds instead of servers in the figure) cloud services to new cloud services. Provider 2 uses Provider 3's service $S_3 1$ and combines it with service $S_4 1$ to offer service $S_2 1$. Provider 5 is able to provide the consumer on the left of the figure with service $S_5 1$ composed by service $S_1 1$ and service $S_2 1$. The Provider 1, 2, and 5 - in this scenario - are both, consumers and providers. The consumer is linked to Provider 5 in order to access the service $S_5 1$ and is linked transitively to Provider 1, 2, 3, and 4.

In this scenario, the consumer interacts with Provider 5 consuming service $S_5 1$. The consumer has no information about the resources behind the service unless Provider 5 informs the consumer. Actually, the consumer is using resources offered by Provider 1, 3, and 4.

Provider 1, 2, and 4 are providers in the sense of the basic cloud usage scenario. They are owning resources. In this set-up, Provider 1 has one customer, namely Provider 5, who is using service $S_1 1$. Provider 3 Provider 2 as customer using service $S_3 1$, who is also customer of Provider 4 consuming service $S_4 1$.

Provider 2 and 5 are both, providers and consumers of cloud services. Provider 2 consumes service $S_3 1$ and $S_4 1$, and Provider 5 consumes $S_1 1$ and $S_2 1$. Provider 2 offers service $S_2 1$ to Provider 5, who is offering $S_5 1$ to the consumer.

An example for an aggregated service could be a cloud service offering redundant storage with resources residing at at least two locations. The provider offering this service does not own storage resources but buys storage resources from two other providers, which will be synchronised.

Providers who also act as consumers can not only offer the services acquired by other providers to their consumers transitively (in an aggregated form). The following scenario introduces enhancements of cloud services - functionality providers add on top.

3. Enhanced services scenario

Aside from the aggregation of cloud services, these can also be improved by extensions or enhancements resulting in new cloud services. An enhancement of a cloud service can be described as adding value on top of other services. Enhancements can also be combined with aggregations of cloud services. In Figure 2.3 Provider 1 combines the services $S_2 1$ and $S_3 1$ offered by Provider 2 and 3 and add the functionality of the components *A* and *B* on top to create the service $S_1 1$ offered to the consumer.

An example of a cloud service, which is an enhancement of another service, is the cloud storage service Dropbox that is using Amazon S3 storage cloud extended with own functionality [35].

In this case, the consumer interacts with Provider 1 through $S_1 1$.

Provider 2 and 3 are providers in the sense of the basic cloud usage scenario.

Provider 1 is a consumer of $S_2 1$ and $S_3 1$ offered by Provider 2 and 3 and a provider of $S_1 1$. Service $S_1 1$ is composed by adding the two components *A* and *B* on top of $S_2 1$ and $S_3 1$.

Enhancing cloud services with a certain functionality leads to the next scenario where software components are linked together to an application that is executed in an execution environment represented by a PaaS cloud.

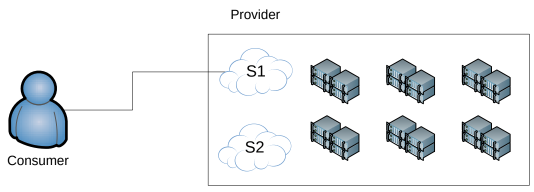


Figure 2.1.: Basic cloud usage scenario

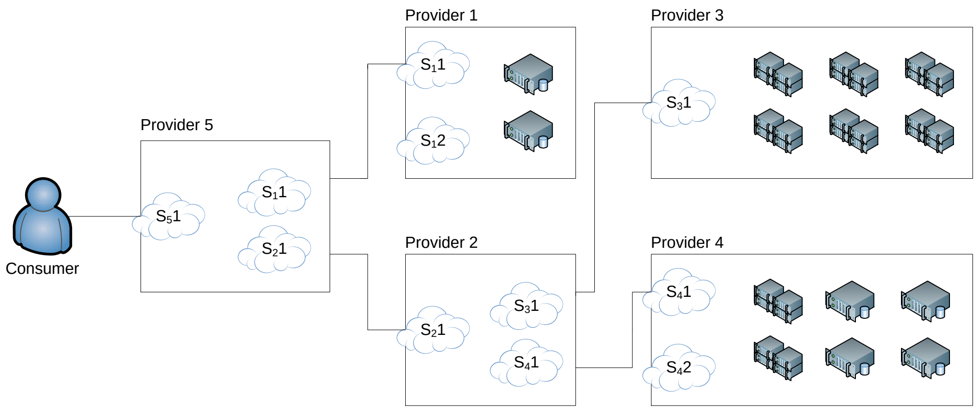


Figure 2.2.: Usage of aggregated cloud services

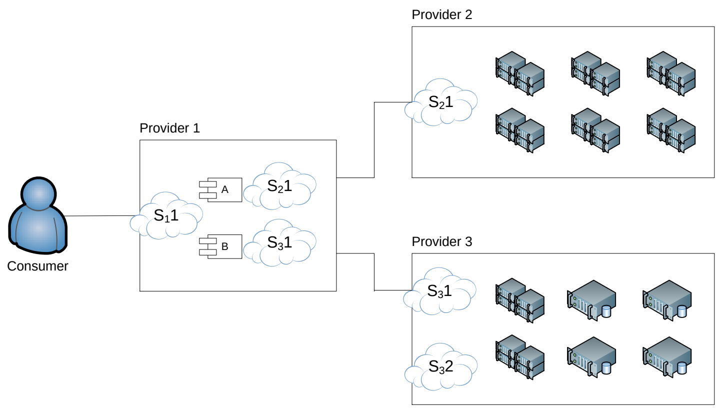


Figure 2.3.: Usage of an enhanced cloud service

4. Execution platform for component-based applications scenario

In contrast to aggregated or enhanced services, this scenario describes a cloud-based execution environment for applications composed by application components. As an execution environment a Platform as a Service (PaaS) offer is chosen. The scenario consists of three main components (illustrated in Figure 2.4):

- a PaaS provider (might enhance IaaS resources acquired over the exchange with PaaS functionality)
- an algorithm directory
- an application linker

The core component is the application linker responsible for accepting an application description containing information about the components the composed application is built of. The application linker provides functionality for the composition of the application and the execution of the composed application on the PaaS offered by the PaaS provider (bottom of the figure). The algorithm directory holds application components (referred to as algorithms and identified by capital letters). Initially, the consumer is interacting with the application linker by sending an application description. The linker checks this description and composes the application by choosing the components/algorithms from the algorithm directory. In the next step, the component-based application is executed on the PaaS cloud.

The provider of the application linking service and the provider of the algorithm directory are dependent on the PaaS provider as well as the consumer in a transitive way over application linking service and algorithm directory provider. The usage of the service is limited to just one provider offering the PaaS. An improvement of the infrastructure could be the utilisation of different PaaS providers. This would prevent capacity shortages and would enable the ability to run the applications on the cheapest PaaS provider's resources.

In the four scenarios presented so far, consumers of cloud services are completely dependent on their provider and therefore are at the mercy of the provider in terms of the interface to access the cloud service, the availability of the service and pricing models, the raising of prices, and last but not least the financial health of the provider (e.g. What happens to valuable company data stored in a cloud provider's data centre when the provider becomes insolvent?).

The consumer interacts with the Application Linker by sending the application description without knowing where the application is executed in the end.

The Application Linker could be offered by a provider as a cloud service in the sense of the previous scenario or could be hosted at the consumer's premises.

The PaaS Provider is a provider in the sense of the basic cloud usage scenario owning resources.

An example describing a similar system for mobile applications is described in *Der Cloud-Broker: dynamische Orchestrierung von Cloud-Diensten zu Smart Mobile Apps* [33].

The PaaS provider could run in the situation of a shortage of resources, when consumers request more capacity than available. The next scenario describes how providers can balance resources between different providers and data centres.

5. Resource balancing scenario

This scenario effects a provider in the sense of the basic cloud usage scenario. If the demand of resources increases, providers may not be able to serve the consumers' needs and the scalability of the cloud service (often referred to as flexibility or elasticity) cannot be guaranteed any more.

In cloud computing one problem is the limited capacity of a provider (Figure 2.5), in contrast to the infinite scalability of cloud services. Consumers might request more resources than available at the provider's data centre at the moment (Figure 2.6). In this case one consumer's cloud service utilisation is a periodical usage depicted as a sine wave like function. The provider's capacity is shown by the blue graph. In Figure 2.5 the resources of the provider are almost fully utilised. If a consumer requests more resources than available in the provider's data centre, the request cannot be fulfilled. The data centre is not able to guarantee the functionality of the cloud service. At the moment, cloud providers have two choices: to extend their data centres or to establish an individual contract with another provider to cover the peak performance. This other provider may charge any price for the resources at any service level.

As consumers today are not able to switch easily and fast between providers and utilise the resources of other providers, the consumers might suffer from a shortage and find themselves locked in by the providers who often offer access to the cloud services over proprietary interfaces only.

A possible solution for the problem occurring in this scenario is a service enabling the exchanging of cloud services between providers to balance the loads (Figure 2.7). After requesting more instances than available at the provider's data centre, the provider is able to cover this peak load by acquiring resources from another data centre and serve the user's need.

In the case a provider is also a consumer of cloud services and needs to move large amounts of data to another provider's data centre to balance the load, the following scenario shows issues arising in this case from the perspective of a cloud service consumer.

6. Provider switching scenario

This scenario describes a consumer with the desire to switch from cloud provider A to another e.g. motivated by the lower costs of the service provided by other providers. In this scenario, the consumer is able to compare the service offers from cloud provider A to other cloud providers (in Figure 2.8 – B, C, D).

Assuming comparable cloud services and a consumer deciding to switch to the cheapest one. The consumer is running computations in the cloud service of provider A and needs a certain amount of data stored in provider A's data centre. If the consumer selects cloud provider B as the new provider and wants to transfer the data needed in provider B's data centre there is no standard way to deal with responsibilities and costs for the transfer of the data from provider A to provider B. Currently, the consumer is responsible for the transfer and has to cover the costs.

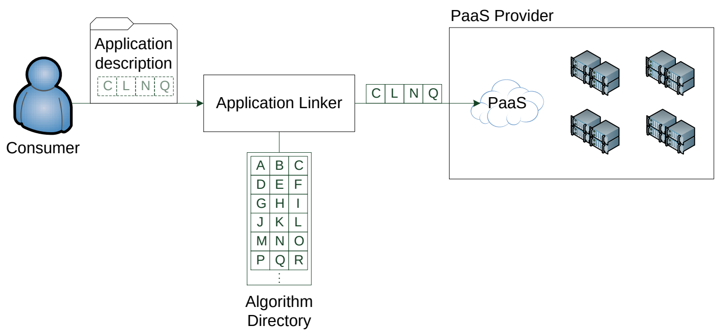


Figure 2.4.: Executing component-based applications on a PaaS cloud

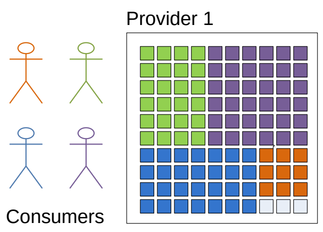


Figure 2.5.: Provider with resource bottleneck

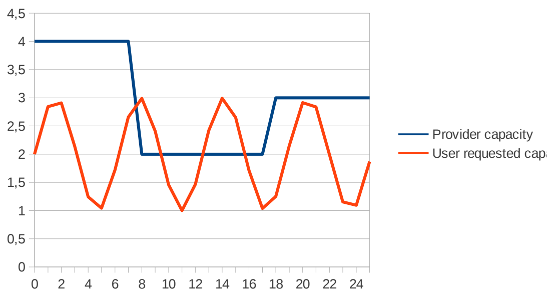


Figure 2.6.: Requested Capacity exceeds the provider capacity

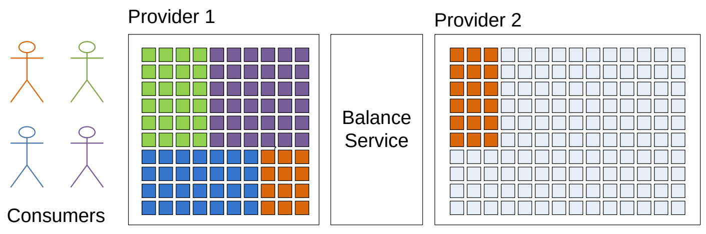


Figure 2.7.: Resource balancing between data centres

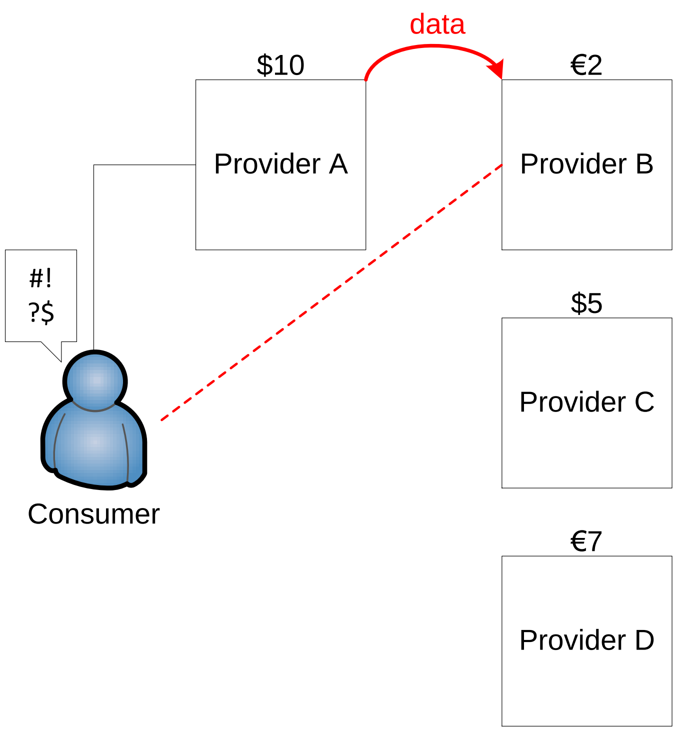


Figure 2.8.: Switching cloud providers

2.1.2. Scenario analysis and functionality extraction

From the scenarios presented above a list of functions can be derived, which helps to enhance the scenarios. This list acts as a basis for the requirements analysis and the functionality of the framework for exchange-based trading of cloud computing commodities.

1. **A mechanism to regulate supply and demand:** The goal is to enable a market bringing together n providers and m consumers of resources ($m, n \in \mathbb{N}$). The current landscape is dominated by large providers with each of them serving a certain group of consumers.

2. **A mechanism to trade standardized buckets of cloud computing resources:** To create tradable commodities resources have to be described in a standardised way by different parameters together with information about a standardised way to access it. A minimum tradable set of resources has to be defined. A marketplace has to be established to trade these standardised buckets of compute commodities.

3. **A mechanism to enable the comparability of cloud resources:** Relations to compare cloud resources of the same type offered by different providers have to be defined.

4. **A mechanism to guarantee constant quality of cloud resources:** An entity has to constantly perform measurements of the resource quality.

5. **A list of the types of cloud resources/services available to trade:** A directory of which cloud service types and their properties has to be published, updated, and made available for (future) providers and consumers.

6. **A dynamic pricing mechanism based on actual supply and demand:** The static pricing mechanisms currently used by cloud providers suffer from several problems. Balancing available resources and consumer demand is only possible if supply and demand is taken into account designing pricing models.

7. **A decoupling of trading from resource delivery:** The selling and buying of cloud resources has to be independent from the providers' resources and technical implementation. A minimum set of information necessary for the trading has to be identified and integrated into the model.

8. **A mechanism to guarantee prices for both, providers and consumers of resources for future usage:** Consumers need to reserve cloud resources in advance and guarantee prices for these advance reservations. Providers need to set a minimum desired price for resources they are offering at the moment and in the future.

2.2. Requirements analysis

In this section, the requirements for an exchange-based model for trading cloud computing commodities are given, as derived from the scenarios and the problem statement. The scenarios have been analysed and in order to identify mechanisms, which are essential to solve the issues and drawbacks. The previous list acts as a basis for deriving the requirements from the scenarios. The identified requirements are given, distinguishing between functional requirements and non-functional requirements.

Functional requirements are requirements representing the functions and methods, the framework has to fulfil in order to achieve the goal – the creation of a framework for exchange-based trading of cloud computing commodities and the non-functional requirements describe the ways these functions and methods have to be implemented.

In the upcoming Section 2.3, these requirements are matched with the approaches presented.

Functional requirements

In the following, the capabilities and mechanisms a system for exchange-based trading of cloud computing commodities has to provide, are presented.

1. The system has to be able to regulate supply and demand.
2. The system has to define a method to limit the access to a group of (verified) users.
3. The system needs to accept requests in a pre-defined form from consumers and providers.
4. The system has to provide information about available resources or classes of resources.
5. The system has to be able to bring together a certain number of consumers with a certain number of providers.
6. Computing services have to be classified or grouped based on quantity, quality, and performance parameters.
7. The system has to provide the capability to compare service offers from different providers.
8. A method to provide information about the quality and reliability of computing services and their providers has to be established (rating mechanisms).
9. The model needs to take into account quantitative, qualitative, and reliability parameters to price computing services.
10. A mechanism for the pricing of cloud computing resources based on the current supply and demand have to be provided.

Non-functional requirements

The following list presents requirements on how the mechanisms shown previously need to be developed.

1. Standards have to be used to define the service classes.
2. Legal constraints (for data privacy) have to be met by the infrastructure and computing service classes.
3. The system should be designed in a way to avoid performance bottlenecks.
4. Users should be able to identify matching computing services without complex brokering.
5. Rating mechanisms should be designed in a not-only centralised approach.
6. A decoupling of trading from the resource delivery.
7. A mechanism to guarantee prices for both, providers and consumers of resources for future usage.

2.3. Related work

This section outlines related work in the field of this thesis. The idea of markets for compute resources has been discussed previously for utility computing [72], grid computing [43], and also cloud computing [15].

The related work approaches have been chosen based on their ability to solve the problem stated in the first chapter. The selected approaches combine distributed computing environments with economic methods responsible for regulating supply and demand of certain resources and the price.

In short, the following related work approaches are considered:

1. The POPCORN Market: At the POPCORN Market CPU cycles can be sold and bought, which can be consumed by Java applications.
2. The GridEcon Project: The GridEcon project describes a marketplace for grid computing resources and introduces an exchange model for certain types of such resources.
3. Standardising Products Based on an SLA-Matching Approach: An electronic market sells products standardised through an approach based on SLA templates that are adapted based on the SLAs offered by sellers and SLAs requested by buyers.
4. Market-Oriented Grid and Utility Computing, Market-Oriented Cloud Computing: Another approach for a marketplace for grid, utility and cloud computing resources is shown in these works.
5. Revenue Management Systems: This approach is trying to optimise the utilisation of a limited set of resources originally emerging from hotel and airline reservation systems.
6. Amazon EC2 Spot Instances: Amazon has introduced a dynamic pricing model based on the current supply and demand in the Amazon owned data centres. There is only one seller, which implies a very limited market without opponents.

2.3.1. The POPCORN Market – POPCORN

An early approach for a supply and demand based market for a very basic good, namely CPU cycles, is presented in *The POPCORN market. Online markets for computational resources* [67]. Regev and Nisan propose an infrastructure for a globally connected virtual parallel computer building on previous ideas around “cycle-stealing” which is not driven by economic aspects. The fact the approach is based on is the large number of idle processors. These processors could be utilised by applications with a large workload. Both, buyers and sellers meet at a market – the trusted intermediate – responsible for matching the bid and ask offers. The commodity “traded” at the POPCORN market is CPU time. There are three main components in the POPCORN market.

1. The application – written making use of the so called POPCORN programming paradigm. This means, the program can be split into "computelets" – sub-computations of the main application that can be distributed over the internet to processors taking part in the POPCORN ecosystem. The computelets are distributed by the POPCORN market to the CPU time sellers.

2. The "provider" part – CPU time is provided by visiting a certain website. The execution of an applet takes care of the "selling" of the commodity.

3. The market – bringing together buyers and seller acting as the matchmaker for bid and ask offers.

The POPCORN market commodity are *JOPs* (Java OPerations) – an equivalent to FLOPs (FLoating Point Operations). In the sense of the POPCORN market JOPs are defined as a specific mix of computations. Attached to each computelet is a benchmark to measure the actual consumption of JOPs. The price for a computelet is proportional to the consumed JOPs. The parallel program has to specify a price for every computelet or JOP included in a *contract* object provided by each computation.

The POPCORN market introduces a currency called *popcoins*. Each participant in the POPCORN market has to have a popcoin account. The market in terms of POPCORN is the bottleneck of the system. Despite a marketplace to bring together buyers and sellers each computation has to be carried out over the market. Each result has to be communicated over the market and each financial transaction has to be handled by the market as well.

Additionally, the POPCORN system offers a philanthropist-approach for websites to include the *POPCORN-logo* in their website. The users visiting the website are informed that parts of their CPU will be used for computing POPCORN computelets as long as the website is opened. These users neither need a popcoin account nor earn any popcoins.

The basic structure of the POPCORN market concept is depicted in Figure 2.9. A user (left hand side of the figure) sends an application containing or consisting of computelets to the POPCORN Market. The POPCORN Market is composed of a Marketplace component, a Distribution/Collection of Computelets/Results component, and a Popcoin Management component. Each computelet includes the price the user is willing to pay to run the computelet. The POPCORN Market selects suitable resources and distributes the computelets accordingly. After successful computations, the results are collected and sent back to the user in a combined form. The Popcoin Management component takes care of the popcoin transactions between the participants of the transaction.

2.3.2. The GridEcon Project – GridEcon

An exchange-based approach in the area of grid computing is described by GridEcon. In the project GridEcon, funded by the European Commission in the 6th Framework Program (FP6) finished in 2009, economic aspects of grid computing have been analysed. Focused on grid computing, the project established an approach for a *Grid Market* and took also into account the first movements in the field of cloud computing. During the project duration, several scientific papers have been published. For the requirements analysis in this work *GridEcon: A market place for computing resources* [14], *GridEcon–The Economic-Enhanced Next-Generation Internet* [13], *Pricing Resources on Demand* [30], and *Market mechanisms for trading grid resources* [29] have been taken into account. GridEcon set up an infrastructure for trading grid resources including virtual infrastructures. The example for the commodity being traded (referred to in the published work) is based on the concept of VMs. In the GridEcon project on demand dynamic pricing for resources (bandwidth) has been analysed. One of the core goals of GridEcon was the definition of a matching algorithm for bids and asks for the VM commodity depending on time constraints. As the name implies, GridEcon is a project dealing with grid computing but tried to integrate virtualisation, which is one of the building blocks for cloud computing. GridEcon's Grid Marketplace consists of (Figure 2.10)

- Grid Market Subsystem – contains bid and ask queues, the matching module, and a fragmentation module
- User Management Subsystem – to allow users and providers to take part in the marketplace
- Scheduler Subsystem – responsible for the management of the computational elements and the binding of the resources
- Security Subsystem – performs checks to guarantee the security of the system; interfaces with accounting and logistics subsystems
- Accounting/Logistics Subsystem – accounting and logistics management
- Directory Services Subsystem – organisation and advertisement of resource leases
- Notification Subsystem – sends notifications to the users
- Scheduler Subsystem – handles the execution of tasks

GridEcon focused on grid computing and did therefore neither classify commodities (despite of VMs), nor provide a mechanism to help users to find a match of their needs to available resources or commodities in the sense of cloud computing.

2.3.3. Standardising products based on an SLA-Matching approach – SLA-Matching

Based on the findings of GridEcon, the paper *Creating standardized products for electronic markets* [22] recognises the need for standardised products in electronic markets. It represents an adaptation of the work created by GridEcon with the SLA Mapping approach presented in [69].

Altmann, the former head of GridEcon, et al. state that is is necessary to standardise products which are tradable on a marketplace for computing resources. The standardisation is achieved by an SLA template model. With this model it is possible to create adaptive standardised cloud products. Public SLA templates represent the products traded on an exchange and have fixed parameter values, whereas private SLA templates are either requests form users or providers and their parameter values are given as a range. Before a product can be traded an SLA mapping has to be computed (e.g. to calculate different metrics). Private SLA templates can be grouped and based on these groups new public SLA templates are created, replacing the previous public SLA templates the private SLA templates of the group are based on. Figure 2.11 shows the setup of an electronic market combined with adaptive standardised products based on SLA matching. Consumers and providers need a SLA Mapping Middleware and a private SLA template. The middleware might need to communicate for the purpose of negotiation. Over the SLA Mapping Middleware both, consumers and providers can either send a publish request or query the electronic market for public SLA templates. The Electronic marketplace holds a database storing the public SLA templates and implements a monitoring and adaption component responsible for updating or replacing existing public SLA templates.

2.3.4. Market-oriented grid and utility computing – Gridbus

In the papers *The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report* [26] and *Economic Models for Management of Resources in Grid Computing* [23] Buyya et al. describe the idea of utilising economic models for the allocation and brokering of grid resources. Buyya et al. outline the potential of economic models to increase the resource utilisation and balance supply and demand in grid environments. With economic models the price for resources can be negotiated based on the demand together with resource parameters, priority, and the available budget. Therefore, the user does not have to pay the highest price. In [23] Buyya and Stockinger present a model with Grid Service Providers (GSPs) – the sellers, Grid Resource Brokers (GRBs) – the buyers/consumers, Grid Resource Trading Services (GRTs), and a Grid Market Directory (GMD). In this model each user has an own broker acting as an agent responsible for selecting the “best” resource matching the user’s job request (Figure 2.12). In the paper, different market models are discussed. The Gridbus Toolkit is a framework for the realisation of grid cooperation (sharing, exchange, selection, and aggregation of resources) driven by economic requirements. Problems concerning scheduling and resource management in grid environments result from different parameters including usage policies, cost models, and varying load. The main components of the Gridbus Toolkit are:

- Gridbus Grid Service Broker – responsible for scheduling decisions based on user requests and the characteristics of available resources.
- Grid Market Directory – a registry for service publication and discovery
- Gridbank – an accounting and micro payment handling system; managing the accounts of buyers and sellers

- Gridscape – enabling the creation of interactive testbeds without programming effort
- Alchemi – providing a runtime environment for the execution of .NET based applications
- Libra – an economy driven scheduling system
- GridSim – a simulator for heterogeneous resources with a large variety of configurations

The idea has been enhanced to deal with cloud resources, lacking the core requirement of a exchange-based approach the predefined commodities. Buyya reuses the concepts developed for grid and utility computing in cloud computing infrastructures *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities* [27]. The motivations are regulating supply and demand, provide feedback about economic incentives, and provide quality of service based resource allocation mechanisms. Cloud providers have to meet different quality of service parameters, which are negotiated with each customer in a specific service level agreement.

At the core of the paper *Cloudbus toolkit for market-oriented cloud computing* [24] and *Market-Oriented Cloud Computing and the Cloudbus Toolkit* [25] is the Cloudbus Toolkit – an update of the Gridbus Toolkit described previously. Cloudbus' intention is to create a marketplace for heterogeneous cloud resources. The basic model of the approach presented by Buyya is to connect cloud resources of all kinds to an "exchange". In his work, Buyya does not clearly distinguish between market maker, meta broker, and cloud exchange. This "exchange" is responsible for aggregating infrastructure demands from application brokers and matching on the published supply. The meta brokering service is selecting the best option available on the cloud marketplace based on the user requirements. The model makes use of *Cloud Coordinators* and *Cloud Brokers* (Figure 2.13). Every participating provider has to deploy a cloud coordinator component responsible for publishing currently available resources. Each participant of the consumer side needs a cloud broker capable of establishing contracts with the cloud coordinators. In the model users have the provide their requirements and quality of service parameters to the broker. The Cloudbus Toolkit represents a collection of technologies enabling the presented model. The core components are

- Aneka – a Platform-as-a-Service (PaaS) Solution
- Broker – the Grid Service Broker, responsible for the access to distributed physical and virtual resources, optimal matching of job to compute resources and their selection, monitoring and job execution, access to remote data, and presenting the results
- Workflow Engine – enables applications to be represented as a workflow
- Market Maker/Meta-broker – intermediate between cloud users and cloud providers; responsible to find a suitable match of cloud resources for a specific user request sent by the cloud brokers

- MetaCDN – offering a storage cloud composed of resources of multiple Infrastructure-as-a-Service (IaaS) providers
- CloudSim – a modelling and simulation engine for clouds

A core requirement for any exchange market is the availability of standardised commodities. This requirement is not fulfilled by this model.

2.3.5. Revenue management systems – RMS

Revenue Management Systems (RMS) are not related to exchange models but offer a way to regulate supply and demand of a resource with the maximisation of the utilisation as one of the goals. The idea of using revenue management systems in combination with cloud computing, especially for the pricing of cloud computing resources, has been presented in *Analysing the Applicability of Airline Booking Systems for Cloud Computing Offerings* [76]. This approach shows how revenue management systems used e.g. by airlines or the hotel industry could be used for selling cloud computing resources. A basic assumption for revenue management systems is a limited set of resources. An aircraft has a limited number of seats available on a flight. The same holds for a hotel that has a certain number of rooms available per night. The paper presents a matching of cloud service requests to flight bookings and outlines the differences between static pricing and the dynamic revenue management pricing approach. The paper introduces technical requirements to enable the trading of cloud resources and therefore establishes ties to this thesis.

As stated above, the use of RMS is motivated by limited resources of any kind and the desire to maximise their utilisation. An example for revenue management system in distributed computing environments can be observed in systems for the maximisation of HPC resource utilisation and the flattening of peaks in requested super computer capacity.

2.3.6. Amazon EC2 Spot Instances – SpotEC2

A special case of an exchange-based approach for the pricing of cloud computing resources is the concept of Amazon EC2 Spot Instances. Amazon has introduced these so called *EC2 Spot Instances* for their Elastic Compute Cloud (EC2) in 2009. The spot instances are priced based on the current supply and demand. The concept is that users can set a maximum price they are willing to pay for a resource. If the price is higher, the resource will be e.g. suspended and resumed when the price decreases to the set maximum.

In the paper *Deconstructing Amazon EC2 Spot Instance Pricing* [19] Ben-Yehuda et al. describe their way to find out the pricing mechanism of the spot instance concept which is not only based on supply and demand but on random preserve prices.

Amazon EC2 Spot Instances are not really a market approach since only one provider is taking part in the marketplace but it is an example for a market-based idea in cloud computing.

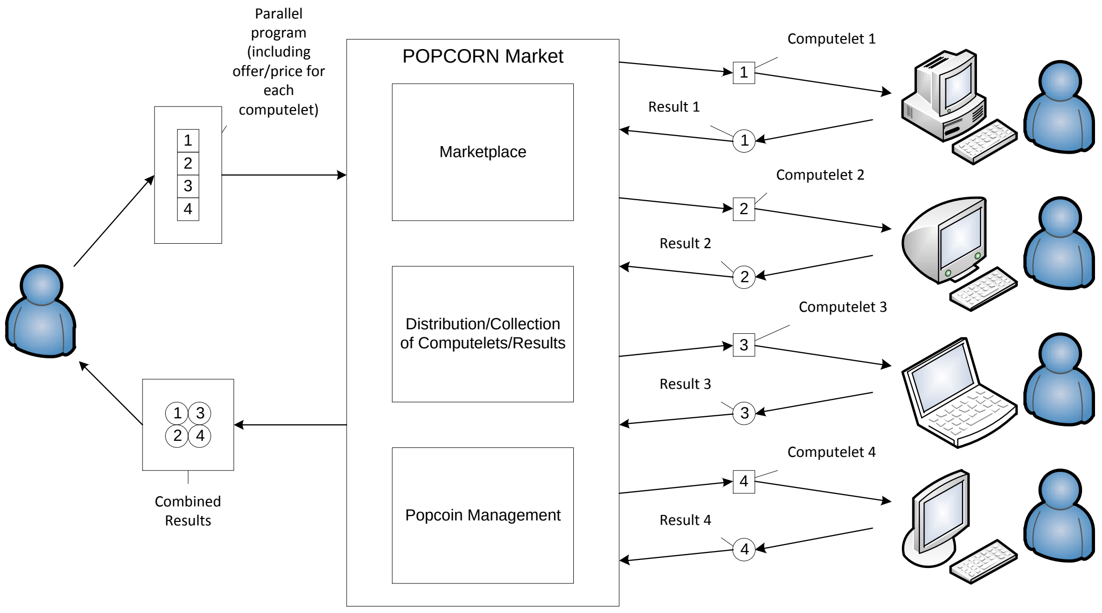


Figure 2.9.: The POPCORN Market

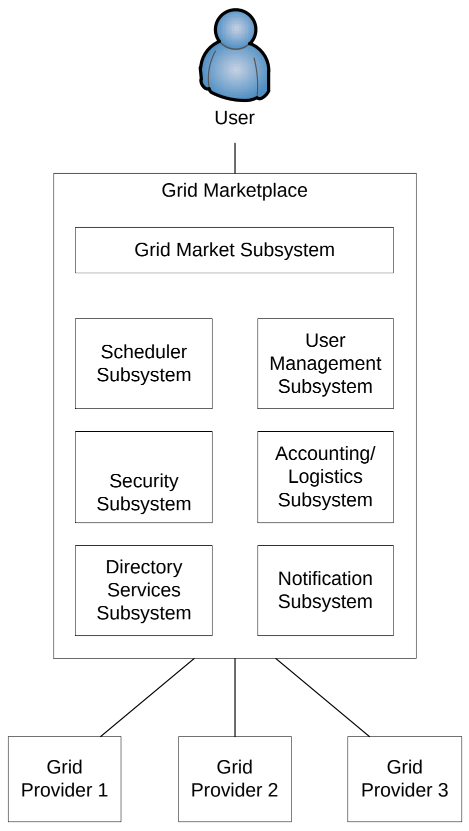


Figure 2.10.: The grid marketplace introduced by the GridEcon Project

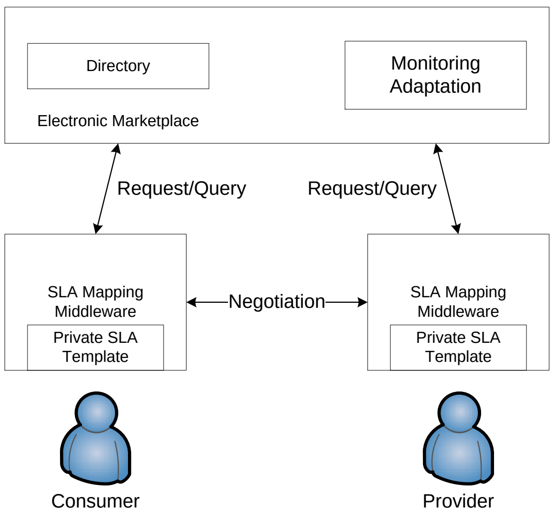


Figure 2.11.: An SLA Matching approach

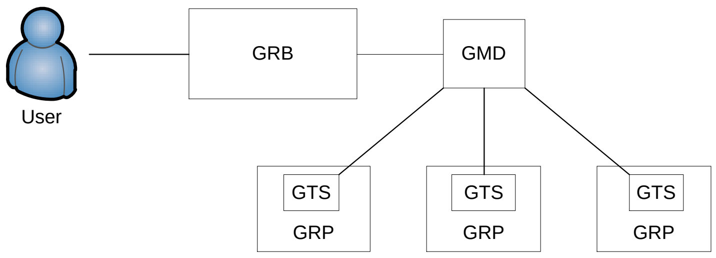


Figure 2.12.: Buyya's market-oriented approach for grid computing

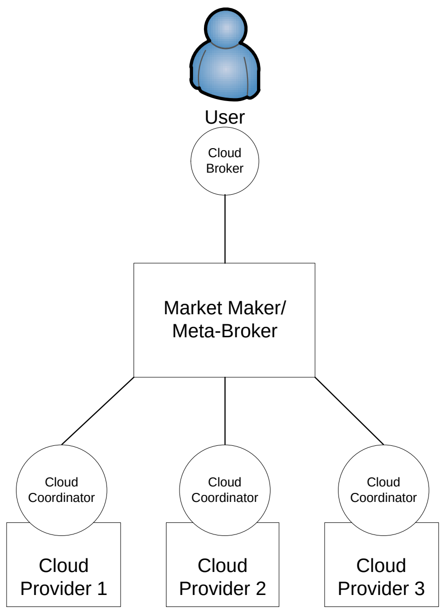


Figure 2.13.: Buyya's market-oriented approach for cloud computing

2.4. Discussion

This section shows where the related work approaches presented can be positioned in the corresponding research fields and provides information how the research done in this work correlates with the state-of-the-art work.

2.4.1. Positioning related work

The research field combining distributed computing with economic models is a very broad one reaching on the distributed computing side from hardware related topics to cloud services. In distributed computing environments, aspects ranging from CPU cycles to virtualised infrastructure services, execution platforms, and even applications can be taken into account. Related to economic models, the research field reaches from markets with only one seller to revenue management systems for optimising the price for a limited set of resources and to exchange-based market models.

IT resources appropriate for a combination with economic models have to be measurable in terms of quantity and quality. Resources should be either normalised or comparable. These resources include:

- Hardware resources: Hardware resources that are measurable and accessible over a certain mechanism (e.g. CPU time, CPU cycles, storage space, HPC resources)
- Virtualised hardware resources: Virtual "Hardware" resources provided over a certain middleware (e.g. hypervisor).
- Execution platform resources: These provide the functionality to develop and run applications. The applications can make use of certain additional capabilities such as automated scaling or databases. An example for execution platforms offering various extra functions (e.g. transfer of large files, virtual organisations) are grids.
- Application resources: These resources provide certain applications available for online usage.

Economic models that are suitable for combining with IT resources include static and dynamic pricing models with the goal to either maximise the revenue of the resource offering entity and/or to regulate supply and demand. These economic models include:

- Static pricing models: Prices for resource usage are pre-defined. The prices may be changed from time to time to meet business requirements.
- Monopolistic dynamic pricing models: Based on the current supply and demand in the context of one provider, the prices are adapted to a certain extent.
- RMS models: These have their origin in the airline and hotel industry. These models

are tightly linked to advanced reservation. The goal of revenue management models is to maximise the revenue of a provider based on certain criteria including the time of booking in advance, certain booking classes with or without restrictions (e.g. restricted airline ticket vs. semi-flexible vs. flexible ticket). For the consumers the pricing is usually not transparent.

- Exchange-based market models: Exchanges for commodities or stocks have the goal to enable the trading of standardised products. The price of a product is deduced from the current supply and demand.

An overview of the types of IT resources and economic models is shown in Table 2.1.

2.4.2. Requirements fulfilled by related work

In this section the previously identified requirements are matched to related work. In the Table 2.2 the following abbreviations for the related work approaches are used: “POPCORN” for The POPCORN Market, “GridEcon” for The GridEcon Project, “Gridbus” for Market-oriented Grid and Utility Computing, “RMS” for Revenue Management Systems, and “SpotEC2” for Amazon EC2 Spot Instances.

Functional requirements

1. All presented approaches intend to regulate supply and demand by the introduction of marketplaces with the capability to bring together a certain number of consumers with a certain number of providers. In the case of Amazon’s EC2 Spot Instances the regulation effect is limited as there is only one provider and the price of the spot instances is not only based on the spot market but also on the demand of resources in Amazon’s static cloud pricing model.

2. Access restrictions to a limited group of users is not part of the POPCORN market (e.g. all viewers of a website implementing the POPCORN donation technology “donate” background CPU time while watching the site). The grid computing-centric works GridEcon and Buyya’s market-based grid include the restriction to a certain user group as grid computing implements strong access control methods. Buyya’s economic cloud work points out some possible security issues but does not integrate them in the system. Revenue management systems are in general designed for a certain resource. In the case of an HPC reservation system the user group able to use the resource are the users of the supercomputer. Amazon restricts it’s users to users with an Amazon AWS account and a credit card (or bank account).

3. The POPCORN market defines a description of the so called computelets that is sent to intermediate from the users. The grid works and Buyya’s market-oriented cloud presented in the state-of-the-art section accept requests from the participants in a certain format.

Approach	**IT resources**	**Economic models**
The POPCORN Market	Hardware resources	Commodity market idea, currency, resource donation model
The GridEcon Project	Virtualised hardware resources, Execution platform resources	Normalised grid resources traded on an exchange-based market
Standardising Products Based on an SLA-Matching Approach – SLA-Matching	Resources described by SLA templates that are adapted continuously,	Traded on an exchange-based market
Market-Oriented Grid and Utility Computing	Execution platform resources	Normalised grid and cloud resources traded on an marketplace over a brokering component
Revenue Management Systems	Hardware resources, Virtualised hardware resources	Revenue Management systems similar to hotel and airline reservation systems
Amazon EC2 Spot Instances	Virtualised hardware resources	Monopolistic dynamic pricing models

Table 2.1.: Assignment of state-of-the-art approaches to IT resources and economic models

Functional Requirements						
	POPCORN 2.3.1	GridEcon 2.3.2	SLA-Matching 2.3.3	GridBus 2.3.4	RMS 2.3.5	SpotEC2 2.3.6
1	✓	✓	✓	(✓)	(✓)	(✓)
2	×	✓	(✓)	(✓)	?	✓
3	✓	✓	✓	✓	(✓)	(✓)
4	✓	✓	✓	✓	✓	(✓)
5	✓	✓	✓	✓	×	×
6	×	×	(✓)	×	×	×
7	×	×	(✓)	×	×	×
8	×	✓	×	×	×	×
9	×	×	×	×	✓	✓
10	✓	✓	✓	✓	✓	✓

Non-Functional Requirements						
	POPCORN	GridEcon	SLA-Matching	Gridbus	RMS	SpotEC2
1	×	×	✓	×	×	×
2	×	×	×	×	×	×
3	×	×	×	×	?	?
4	✓	×	×	×	×	✓
5	×	×	×	×	×	×
6	×	×	×	×	×	×
7	×	×	×	×	×	×

Table 2.2.: Matching of requirements and state-of-the-art work

The SLA Matching approach accepts requests based on a certain SLA template. Revenue management systems and Amazon EC2 Spot instances are limited to one provider but accept user requests in a pre-defined format.

4. This requirement is fulfilled by all the presented approaches as each of these publishes information about the resources available to buy or sell. The problem with the presented approaches is the heterogeneity of the resources. Commodities, which can be traded on an exchange need to be standardised.

5. This requirement is fulfilled by the POPCORN market, the GridEcon project, the market-oriented grid and cloud approaches by Buyya. The requirement is not fulfilled by revenue management systems and Amazon's Spot market for EC2 instances as these bring together consumers and only one provider.

6. This requirement is not fulfilled by the presented approaches and fulfilled in a limited way be the SLA-Matching approach. Resources are tried to be normalised on different levels or in the case of revenue management systems and Amazon Spot instances there is only one "good" or commodity.

7. With the presented state-of-the-art work it is not possible to compare offers from different providers. The parameter space is multidimensional and not equal for every offer.

8. A method to provide information about the quality and reliability of computing services and their providers has to be established (rating mechanisms).

9. All the approaches do not take into account all three, namely quantitative, qualitative, and reliability parameters when pricing computing services.

10. The presented economic driven models for computing resources implement pricing strategies as a core element.

Non-Functional Requirements

1. None of the approaches uses service classes; therefore, no approach makes use of standards in defining those. The SLA-Matching approach defines SLA templates based on certain standards.

2. The handling of legal constraints is not covered by the presented the state-of-the-art work.

3. Bottlenecks have been identified in the presented work except revenue management systems and Amazon EC2 Spot Instances. There is no information in the case of revenue management systems as these are designed for a certain resource or set of resources. There is no information about the implementation of Amazon EC2 Spot Instances.

4. This requirement is fulfilled by models making use of revenue management systems and Amazon EC2 Spot Instances as there resource offers are very limited (might be only one).

In the case of the other approaches, brokers are necessary to find matching resources.

5. None of the state-of-the-art works implements rating mechanisms – neither centralised nor non-centralised or a combination of both.
6. None of the presented approaches decouples the marketplace from the resource delivery
7. The GridEcon approach has capabilities to deal with certain types of derivatives.

2.4.3. Comparison to related work

The state-of-the-art work shows approaches to combine economic models and distributed computing environments. Marketplaces have been introduced for CPU cycles, compute grids, and cloud computing. Compared to state-of-the-art approaches, this work also aims to increase the utilisation of computing resources and manage scarcity of resources – in this case cloud computing resources and regulates the supply and demand with exchange mechanisms – but other than the related work this thesis introduces cloud computing commodities, which enable the possibility to compare and harmonise cloud computing service offers from different providers and therefore realising the commoditisation of cloud computing services. In other approaches, a brokering instance is responsible for the matching of complex user requirements to a pool of heterogeneous resources or the matching of offered and requested SLAs. This matching is both, a bottleneck and faces the problem of non unique solutions, what means the broker is not able to find *the* best match. Usually, only a solution set can be given as a result. In these cases the determination of a price is very hard as the available and requested resources change constantly, which makes it very hard or even impossible to create derivatives on certain underlyings.

In the approach for a framework for exchange-based trading of cloud computing commodities a way users can select a commodity by given parameters is shown. These parameters include information about the performance of certain applications executed on the cloud services offered as a certain cloud computing commodity.

The state-of-the-art work shows marketplaces integrated with cloud management, VM Management, and other components like accounting and billing mechanisms. The way this work sets up the marketplace is the decoupling (as much as possible) of the exchange – responsible for accepting orders and their matching and the pricing of the commodities, the clearing – for handling the financial transaction, the settlement – responsible for the delivery of the good (the credentials to access the cloud computing service), and the rating agencies – performing monitoring of transactions, service level violations, and other aspects to be able to provide ratings for cloud services and their providers. The trading of resources is virtual. There is no need to integrate components for the management of resources into the trading environment. The result of a successful trade is a contract allowing to use a certain cloud computing commodity for a certain timespan together with the access information to use the service.

Chapter 3

Conceptual Model for an Exchange

This chapter introduces a conceptual model for an exchange for the trading of cloud computing commodities. In the first part, the analogy to traditional exchange markets is outlined, and a general picture of how exchanges work is shown. After an overview of the roles involved in an exchange marketplace, additional aspects related to the trading of cloud computing resources are discussed. The core element of this chapter is the model for enabling the exchange-based trading of cloud computing resources. Each of the components and roles is described together with a way for the secure handling of transactions.

3.1. Analogy to traditional exchanges

This section explains different types of exchange markets together with the analogy to a cloud computing commodities exchange. The link of the newly created exchange market to traditional markets is established and the specific aspects of trading cloud computing resources are discussed.

3.1.1. Exchange markets

Exchanges have been established to bring together buyers and sellers of goods (both real and virtual goods) and regulate supply and demand. Buyers are not necessarily the consumers of a good and sellers are not necessarily the providers or producers of a good. Buyers and sellers place orders containing the good's name or identifier, the quantity, and the price as the core information. The exchange is responsible keeping track of buy and sell orders. For this, the exchange makes use of order books that contain a list of buy and sell orders. With the help of matching algorithms orders are searched with matching quantity and price. If matching orders are found this results in a trade. The relation between supply and demand of a good determines its price.

Two main requirements of an exchange are the standardisation of the traded goods and the standardisation of the contracts between buyers and sellers.

Exchanges exist for various kinds of commodities, stocks, and derivative financial instruments. The first exchanges in Europe have been constituted in the early medieval mostly for agricultural goods. A well known example of an exchange for agricultural goods and derivatives is the tulip market in Amsterdam during the Dutch golden age, which led to a crash of this market in the years 1636 and 1637 [44]).

3.1.2. Types of exchanges

In the following, a brief overview of stock and commodity exchanges representing two types of exchanges are given as an example.

- Commodity exchanges enable the trading of various goods registered on these exchanges. The traded commodities range from agricultural goods to metals, oil, gas, and electricity. Usually, commodities are traded as future contracts, so the existing commodities exchanges are commodity futures exchanges.

  A special case of commodities is electric energy [17], [41], [46]. Energy contracts are traded in blocks of one or more time units on derivative and spot markets (more details in Section 3.1.4).

- Stock exchanges enable companies to raise additional money – a motivation for a company to get listed on an exchange (e.g. to expand the business). Stock exchanges bring together buyers and sellers of company shares listed on the exchange.

Products traded at an exchange can either be delivered immediately (spot market) or delivered at a date in the future (derivatives market). Derivatives are financial instruments based on an underlying real or virtual good. Examples of derivatives are standardised future contracts or options. Derivative markets enable the trading of these financial instruments.

3.1.3. A generic exchange model

As an introduction to exchange markets this section presents a basic general model of an exchange market. A generic exchange model as depicted in Figure 3.1 involves:

- exchange
- buyer
- seller
- broker (between buyers or sellers and the exchange)
- market maker

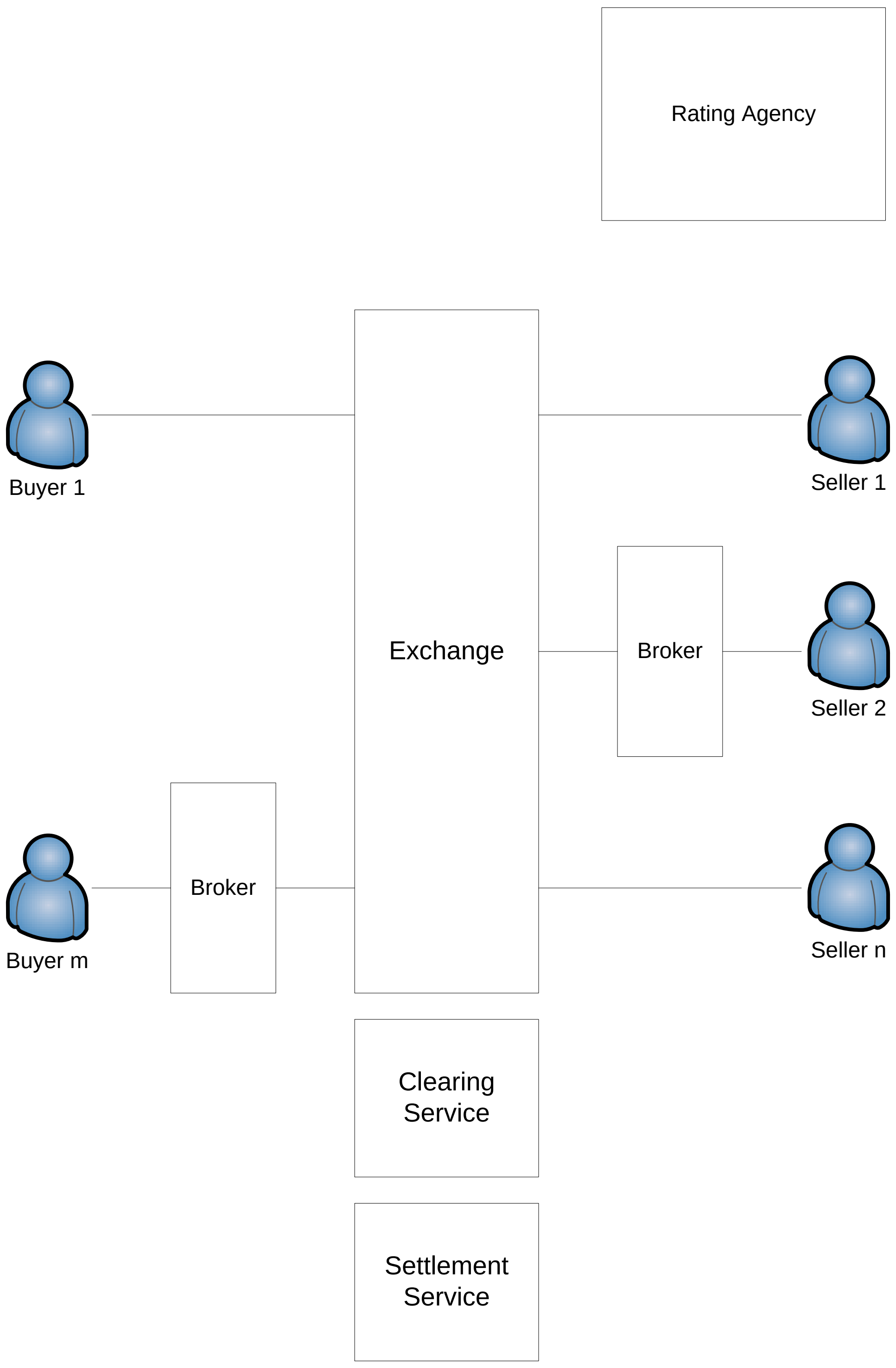


Figure 3.1.: Generic exchange model

- clearing
- settlement
- rating agency

In the following, these terms are explained based on the definitions given in the *Rules of the London Stock Exchange* [40] with two additional roles, namely consumer and provider as they are necessary in the framework later in this chapter.

Exchange The exchange is responsible for accepting offers and requests from buyers, sellers and/or brokers. The requests and offers are inserted in an order book, and the matching, auctioning and determining of prices is performed. The exchange provides a description of each good listed on the exchange and in case of commodity markets also to ensure comparability of offers and a consistent quality of the exchange traded products.

Remark: Exchanges in today's world are subject to regulatory frameworks controlled by supervisory authorities. This work does neither provide insights of the legal situation nor any analysis regarding the legal requirements and implications related to an exchange for cloud computing resources.

Buyer Buyers place a buy order for a certain tradable good on the exchange.

Seller Sellers offer a certain quantity of an exchange listed good at a (minimum) desired price for the offered good. *Seller* can also refer to a reseller in the case a buyer sells a previously bought good (e.g. a buyer who acquired too much capacity can resell the overcapacity)

Remark: Both, buyer and seller represent marketplace roles. They interact directly with the exchange or transitively via a broker. In the case of direct interaction with the exchange buyers select the exchange traded product they intend to consume, the quantity, and the price they are willing to pay. Sellers select the exchange traded product they would like to sell, the quantity, and the minimum price they want to get for the offer. This information (order information) is forwarded as a buyer or seller request to the exchange in a request form (buy order, sell order) defined by the exchange. The order information contains at least the identifier of the tradable good, the quantity, the price, and the ID of the buyer or seller.

Broker A broker acts as an entity between buyers, sellers, and the exchange. It accepts buy requests from buyers and sell requests from sellers. The requests from buyers and sellers to the broker may not be provided in the form of orders defined by the exchange and may also contain weaker descriptions of the goods. The broker generates orders with the information provided in the requests and places them to the exchange.

Market maker Market makers guarantee the liquidity of the market. They act as both, buyers and sellers, and therefore represent a special case of buyer or seller.

Clearing The clearing represents a central entity between a buyer and seller of an exchange traded product. It is responsible for the management of a transaction until its settlement described in the next paragraph. A clearing service might offer settlement services. Usually, clearing services require the buyers and sellers to deposit a margin payment to reduce the risk of non-payment or non-delivery.

Settlement The settlement provides processes for the delivery of an exchange traded product for a certain amount of money. Simplified, *settlement* describes the actual delivery of a commodity.

Rating agency Rating agencies turn parameters representing the financial health of a provider and corresponding indicators into a rating. These ratings provide a measure of the risk of investing in the rated financial instrument. In the financial services world, rating systems have been established by the major rating agencies like Fitch, Standard & Poors or Moody's. More details of the financial rating systems can be found in *Prototype risk rating system* [31] and *Credit risk rating systems at large US banks* [73]. In Section 4.4 a rating system for service providers is introduced derived from existing financial rating systems.

These roles and components stay the same in a commodities exchange for cloud computing commodities. Adjustments concerning rating agencies have to be applied as described in Section 4.4.

3.1.4. Selected economic aspects

For a better understanding of the economic aspects related to marketplaces, this section gives an overview of commodity markets and commodity derivative markets with a focus on electricity markets, as there are similarities to cloud computing resources as well as penalties, insurances in case of non-delivery or non-payment, and a brief view on the risk. The information provided in this section is based on [41], [45], and [46].

Commodity and derivatives markets

Commodity markets have been established for various goods ranging from agricultural products, metals, oil, and gas to electricity.

Derivatives in commodity markets have been started to trade in the fourth millennium before Christ in Mesopotamia and the Byzantine Empire as Weber states in *A short history of derivative security markets* [77]. Futures exchanges in the basic form of those existing today have been established in Europe and the United States in the second half of the 19th century [61].

A derivative is a financial instrument which depends on one or more underlyings. The following list shows three basic representatives of derivatives that are considered as important in the context of a cloud computing commodity market.

- **Forward contracts**: A forward contract is an agreement between a buyer and a seller to deliver a defined amount of a commodity at a certain price at a certain date in the future. Forward contracts are not exchange-traded.
- **Futures contracts**: On the first sight future contracts seem equal to forward contracts but the difference is that future contracts are standardised, have standardised underlyings, and are traded on an exchange. Clearing houses require margin payments before orders can be placed on the exchange but through this the risks of non-delivery or non-payment (counterparty risk) is reduced by the clearing house. In terms of commodity markets the most prominent underlyings of futures can be grouped in:

  - Grains
  - Food
  - Metals
  - Energy
  - Others

  In Table 3.1 examples of commodities belonging to these groups are shown.

  Examples of exchanges for futures are the European Exchange (EUREX), Chicago Mercantile Exchange (CME), New York Mercantile Exchange (NYMEX), the Hong Kong Futures Exchange (HKEX) and the European Energy Exchange (EEX) in cooperation with EUREX.
- **Options**: Options are a financial security grants which give the option holder the right to buy or sell a certain underlying at a certain date or maturity (European option) of before a certain date (American option) for an agreed price (strike). There exist two types of options:

  - Call option: A call options issues the holder the right to buy a certain underlying at or before a certain date for a certain price
  - Put option: A put option gives the holder the right to sell a certain underlying for a certain price at or before a certain date

Apart from the previously mentioned European and American options, other types like Asian options are referred by the umbrella term *Exotic options*. For the calculation of the option price a formula has been developed by Fisher Black, Myron Samuel Scholes and Robert C. Merton – the *Black-Scholes Formula* or *Black-Scholes-Merton Formula* ([21]) – a stochastic differential equation (SDE) [56].

In the next section, the commodity electricity is presented. Electricity is a virtual commodity and shares some important characteristics with cloud resources.

3.1.5. Reference commodity: Electricity

The commodity electricity has been chosen as an important example commodity, which can act as a reference commodity when building cloud computing commodities. There are similarities between electrical power and cloud computing resources examined in more detail in this section. In general, electrical power is a non-storable good and locality matters since it demands a well balanced powergrid to make its way from the producer to the consumer. Electricity is traded in *multisettlement markets* ([46], [55]). An example how the trading is done in a multisettlement market gives the Pennsylvania-New Jersey-Maryland (PJM) market. This market consists of:

- **Day-ahead market** This market enables the trading of electricity for the following day on an hourly basis. Offers can be placed seven days in advance. The market for the following day closes at noon. The results of the day-ahead market are published at 4PM.
- **Day-of market** In the day-of market, electricity of the current rest of the day can be traded (per hour).
- **Hour-ahead market** The hour-ahead market enables trading of the electricity for the upcoming hour.
- **Real-time market** The real-time market acts as a reconciliation market to antagonise deviations in the markets explained before.

In energy markets, an independent entity called *Independent System Operator – ISO*, usually a non-profit organisation, is responsible for the operational health of the transmission system. For this, reserve capacity is needed. Reserve capacity is normally gained from providers or generators not delivering 100% of their full-load capacity. In the unlikely event of an outage these reserves can be utilized as ordered by the ISO. Reserve capacity is divided into different availability classes from instant reserve supply, reserve supply after 30 minutes, reserve supply to keep the voltage of the system constant, etc. Providers of reserve capacity usually have contract agreements established with the ISO. There exist markets also for reserve capacity.

The vast amount of electrical energy is traded mainly at the European Energy Exchange as forward and future contracts settled financially or both financially and physically. For electricity markets, locality is crucial because the transmission network might be too weak to transport the energy from the generator to the consumer.

In electricity (also in natural gas markets), certain contracts might be established allowing interruptibility, which gives the provider the right to interrupt the delivery of the commodity for reasons like system emergencies or when the price reaches a certain level.

The concept of a cloud computing commodity market introduced in Section 3.2 is a special from of a commodity market. There is a big overlap with the trading of electricity. The following main similarities can be identified:

- As in electricity, cloud computing resources are a non-storable commodity
- For cloud computing resources certain restrictions (legal constraints) exist which restrict the usage to certain legal domains (zones)

Despite the similarities, there are differences to electricity:

- There is no physical requirement to balance a network of cloud computing resources
- Cloud computing resources need to be described by more dimensions than electricity

Grains	**Food**	**Metals**	**Energy**	**Others**
Corn	Cocoa	Copper	Crude oil	Carbon certificates
Rice	Coffee	Gold	Electricity	Weather
Wheat	Pork bellies	Platinum	Natural gas	
	Orange juice	Silver		
	Sugar			

Table 3.1.: Examples of exchange traded commodities

3.2. A model for a cloud computing commodities exchange

Based on the concept of existing markets and state-of-the-art work as shown in Section 2.3, a model for an exchange for cloud computing commodities is presented in this section starting with the main advantages of trading cloud computing commodities on an exchange. The roles and entities of a cloud computing commodities exchange are presented together with their requirements. The conceptual model introduces the main components of the framework followed by the representation of the model in UML. After providing the information and data model and presenting a concept for secure clearing and settlement, the section closes with a matching of the requirements and the presented model.

The main advantages of using an exchange model for the trading of cloud services are listed in the following. These advantages are not restricted to the field of business management.

- With an exchange model for the trading of cloud computing commodities, supply and demand can be controlled.
- The cloud services traded on the exchange have to meet certain criteria defined by the exchange. Resources not meeting the criteria cannot be offered. This ensures comparability and a minimum standardised service level for each of the listed services.
- For each of the listed cloud computing commodities, a method of access is defined. This enforces standardisation – providers who want to offer a resource for a certain listed cloud service have to provide access to their resource over the defined and standardised access method. The usage of open standards should be a goal.
- A benefit for providers and consumers is the potential of guaranteed prices for certain periods of time. These guaranteed prices can be achieved by the usage of futures and options.
- There is no standardised way of insuring the case of non-delivery of resources (e.g. broken SLAs). There are neither standardised services nor a way of insurance against broken SLAs or outages. The exchange model allows the trading of standardised services (cloud computing commodities) enabling the creation of common insurance methods as well as more complex models based on financial instruments.
- With the introduction of a third party responsible for ratings, the quality (SLA conformance) of providers and their offered services can be monitored and logged. Providers will not be able to offer their services for certain prices any more or may even be excluded from trading if their reputation is at a low level.

3.2.1. Roles and entities in a cloud computing commodities exchange

In an ecosystem for the trading of cloud services, different players and roles can be identified. These roles can be mapped directly to the roles in traditional exchange markets (see Section 3.1.3). The definition and explanation of terms including the roles in traditional exchange markets is provided in *Rules of the London Stock Exchange* [40].

- Exchange: The central point responsible for accepting offers from sellers and requests from buyers.
- Buyer: An entity sending a request to the exchange with the intention of buying a resource (or derivative). This entity is not necessarily the consumer. Buyers may buy resources (or derivatives) just for the purpose of reselling them.
- Seller: An entity sending an order to the exchange with the intention of selling a resource (or derivative). This entity is not necessarily a provider.
- Broker: A broker in the sense of an exchange for cloud computing commodities is an entity executing trades on an exchange with respect to criteria requested by their customers. Brokers act between the exchange and a buyer or seller or consumer or provider from whom they are commissioned to place orders meeting certain criteria. From the perspective of the exchange, the broker acts as a buyer and seller. Brokers are not investigated in more detail in this work as they can be seen as buyers and sellers.
- Market Maker: In this work, market makers will not be examined further as they can be seen as a special form of buyers and sellers.
- Clearing: The clearing is responsible for the secure and anonymous processing of the financial transaction.
- Settlement: The settlement provides a mechanism for the delivery of the access information (not the actual cloud service). The access information or credentials are the *goods* exchanged between buyer and seller).
- Rating Agency: Rating agencies are responsible for monitoring the financial and operational health of the providers and the quality criteria of the services offered. Based on this information, rating agencies calculate ratings that are published.

3.2.2. Requirements for each role

Derived from the general requirements presented in the previous chapter, this section lists the requirements for each of the identified roles and entities (Section 3.2.1) in the framework for exchange-based trading of cloud computing commodities introduced in this work.

- Exchange
  - The exchange is required to provide a list of the cloud computing commodities that can be traded.
  - For each of the cloud computing commodities, the exchange has to define parameters including quality criteria, and performance criteria including benchmarks, and benchmark threshold values.
  - The exchange has to be able to accept offers from sellers and requests from buyers in a form defined by the exchange.
  - The exchange has to provide a method for determining the price for each of the services based on supply and demand (matching bid orders and ask orders, pricing algorithms).
  - The exchange has to offer the functionality of a certificate authority.
- Buyer
  - In case of direct interaction with the exchange: The buyer has to be able to formulate a request in the form the exchange accepts requests.
  - In case of interaction with the exchange over a broker: The buyer has to be able to formulate a request in a form the broker accepts requests.
  - Buyers are able to resell capacity acquired on an exchange.
- Seller
  - In case of direct interaction with the exchange: The seller has to be able to formulate an offer in the form the exchange accepts offers.
  - In case of interaction with the exchange over a broker: The seller has to be able to formulate an offer in a form the broker accepts offers.

- Broker
  - Brokers define a form they accept requests from buyers and/or sellers.
  - Brokers have to be able to accept requests from buyers and/or sellers in a predefined form.
  - Brokers have to be able to formulate orders when interacting with the exchange.
  - Brokers have to be able to find offers or requests matching the buyers' or sellers' requests.
- Clearing
  - The clearing needs a mechanism to check the validity of a trade.
  - The clearing has to ensure the anonymity of the market participants.
  - The clearing service is required to provide methods to ensure the security of the transaction.
  - The clearing has to be linked to the settlement in order to provide trusted account capabilities.
- Settlement
  - The settlement has to provide mechanisms to receive credentials in an encrypted form from the seller.
  - The settlement is required to store encrypted credentials in a database and provide mechanisms for looking up these credentials on request.
  - The settlement has to implement capabilities to send credentials in an encrypted form to the buyer.
- Rating Agency
  - Rating entities (or agencies) monitor the quality of cloud computing services for the transactions performed over the exchange (physical settlement) and log incidents and the financial and operational health of the providers.
  - Rating entities hold a list with the provider reputations and ratings.

  - Rating agencies have to implement algorithms to calculate the ratings.
  - Rating agencies have to take into account the financial health of providers in order to calculate the ratings.
  - Rating agencies have to have the ability to buy resources as control samples and run benchmarks to check if the defined parameters are met.

Despite of the marketplace roles *buyer* and *seller*, the roles involved in the delivery (the settlement) of cloud resources, namely *consumer* and *provider* are described as follows:

- Consumer: Consumers are entities with the desire to get a good delivered (settled) in order to be able to use or consume it. In order to buy resources on the exchange, a consumer has to act as the marketplace role *buyer*.
- Provider: A provider is an entity that owns a good, which is delivered to a consumer through the settlement. In order to sell resources on the exchange, a provider has to act as the marketplace role *seller*.

3.2.3. Conceptual Model

Based on the generic exchange model presented in Section 3.1.3 and Figure 3.1, a conceptual model for an exchange capable of trading cloud computing commodities is presented in this section. In order to be able to show the relations and interactions between the different entities, the modelling language UML 2.0 ([52]) has been chosen. The model is developed with concepts of Model Driven Architecture (MDA) [47] to achieve a formal and abstract representation of the framework.

The main components of the core part (Figure 3.2, the four central squares) are the exchange, the clearing service, the settlement service, and the rating agency. Optional components (Figure 3.2, rectangular boxes) are brokers. The trivial case of one exchange, clearing, settlement, and rating component will be extended in the following section. Enabling more than one entity responsible for clearing, settlement, and rating is necessary to establish an open market and to avoid bottlenecks in the transactions. The security and trust between the different entities in the cloud computing commodities exchange ecosystem is ensured by introducing a public key infrastructure. The following list shows the basic functionality of the four components exchange component, clearing component, settlement component, and rating agency component.

- Exchange component
  - Publishes the list of available tradable cloud computing commodities
  - Publishes the required cloud computing commodity parameters
  - Accepts buy and sell orders for the different commodities
  - Matches buy and sell orders (a match of a buy and a sell order results in a trade)
  - Keeps track of the bid orders, ask orders, and the trades
- Clearing component
  - Carries out the financial transaction process for the trades
  - Keeps the anonymity of both buyer and seller
  - Receives money from the buyer
  - Sends money to the seller
  - Keeps a trust account for the financial transactions
- Settlement component
  - Receives encrypted credentials from the seller
  - Stores encrypted credentials in a database
  - Sends encrypted credentials to the buyer
- Rating agency component
  - Monitors occurring issues during the transaction and/or service usage taking into account the feedback of the service consumers
  - Generates periodically updated provider ratings based on cloud service quality data, consumer feedback, and financial health of the provider
  - Publishes the provider ratings

The four components interact between each other and with the three roles buyers, sellers, and brokers. First the three roles interact with the exchange by placing orders. The exchange communicates with the clearing in order to inform about successful trade. In order to initiate the flow of money and the delivery of the cloud resources, the three roles have to interact with the clearing and the settlement component. The rating agency component is constantly checking conformance to the required SLAs for each provider and publishes these information in a condensed form as ratings.

3.2.4. Description of the UML model

The UML model of the cloud computing commodities exchange infrastructure is depicted in Figure 3.3 presenting a class diagram that shows the attributes and relationships between the components and roles. In addition to the four components of the framework, namely exchange, clearing, settlement, and rating agency, the model contains consumer and provider to show their relationships to the components. As these roles may change between consumers, providers, buyers, and sellers, these roles are merged to one *market participant* in Chapter 5. To be able to outline the relationships and the information flows of the buyer/consumer and the seller/provider side, the four roles are merged into two in this section: consumer with buyer and provider with seller.

The central element of the system - the exchange component - is represented by the class *Exchange*. This class contains the list of cloud computing commodities, which can be traded on the exchange. In addition to the commodities list (*CloudCommoditiesList*), the exchange component includes the element *CloudCommodityParamList* offering the parameters (see Section 4 for cloud computing commodity parameters) of the respective cloud computing commodity. Both elements are public and can therefore be accessed by the market participants.

The exchange has to provide the functionality of an orderbook. This functionality is realised by the list for bid orders and ask orders, the list of trades *BidOrderList, AskOrderList, TradeList*, the methods for accepting an order *acceptOrder()*, and for matching bid and ask orders *matchOrders()*. If a bid and an ask order match, a trade is created and both the consumer and the provider are informed. In addition, the exchange has to provide the functionality to register and remove (deregister) commodities (*registerCommodity(), deregisterCommodity()*). Traders - the market participants - are stored in the list *TraderList*. The traders are added to the list with the *registerTrader()* method and removed from the list with the *deregisterTrader()* method.

As the trusted intermediate in the financial transaction, the clearing (see *Clearing* class in Figure 3.3) receives the payments for the trades contract from the buyer (*acceptPayment()*). The money is “stored” in an account - represented by the *ClearingAccount*. The clearing is able to inform the settlement about received money for a trade *informSettlement()* and send the money to the seller in case of a successful transaction *releasePayment()*.

The settlement represented by the *Settlement* class is responsible for the delivery of the goods. The settlement provides functionality to receive credentials from a provider and send credentials to a user (*receiveCredentials(), sendCredentials()*). After receiving credentials for a certain trade, the settlement informs the clearing about the receipt of the credentials *informClearing()*. Between receiving and sending the credentials, the settlement stores these credentials in the *CredentialList*.

For each cloud computing service offered by a provider, the rating agency computes a rating (*calculateRating*), which is stored in the *CloudServiceRatingList* and published (by either listing all ratings *listRatings()* or getting the rating for a specific service offered by a certain provider *getRating()*). The rating agency publishes the ratings and informs the providers about any changes of the rating of an offered cloud service. For the collection of technical data relevant for calculating the rating the two methods *monitorService()* and *gatherTransactionFeedback()* are available in the *Rating* class.

Consumers send buy orders for a certain cloud computing commodity they desire to use (*sendBidOrder()*). They have to be able to send money to the clearing *sendMoney()* and to receive credentials from the settlement *receiveCredentials()*.

A provider is able to publish a list of cloud services that can be offered *serviceList*. Providers send sell orders to the exchange (*sendAskOrder()*), send credentials to the settlement (*sendCredentials()*), and receive the payment for a trade from the clearing (*receiveMoney()*).

3.2.5. Information and data model

Figure 3.4 shows the communication taking place between the entities in the model.

The exchange receives orders from buyers (consumers) and sellers (providers).

An order contains the following information:

- Type of the order (buy order, sell order)
- Name of cloud computing commodity
- Amount of cloud computing commodity to buy or sell
- Desired price
- Name of the trader
- A unique order identifier (number)
- A digital signature (signed from the buyer or seller)

If the matching of two orders (bid and ask) has been successful the exchange informs both the buyer and the seller and additionally the clearing about the successful trade by sending the trade to the three entities. The matching of the orders is done by the exchange. The exchange is using order books for listing all buy and all sell orders and to apply a matching algorithm. This algorithm takes care of finding two orders with a matching quantity and price.

A trade is composed by:

- Name of the cloud computing commodity
- A unique trade identifier (number)
- The identifier of the buy order
- The identifier of the sell order
- Amount of the cloud computing commodity
- Price
- A digital signature (signed by the exchange)

The interaction between the different components and roles is described by seven use cases listed in 3.2. A mechanism to guarantee secure transactions between buyer, seller, clearing, and settlement is put in place described in detail in Section 3.2.6 together with the exchanged data.

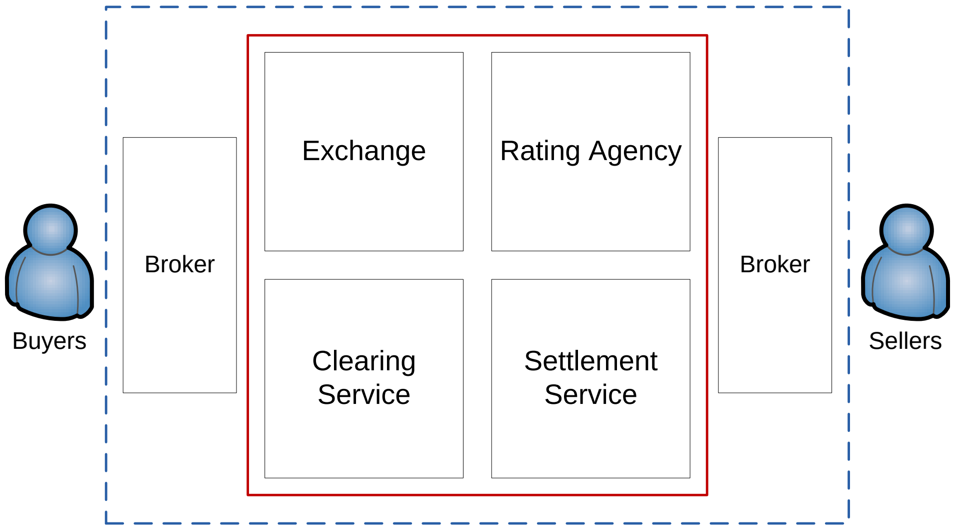


Figure 3.2.: Components of an exchange for cloud computing commodities

Table reference	**Use Case**
Table 3.3	Buyer buys a resource with the desire to use it (buyer=consumer)
Table 3.4	Buyer buys a resource with the desire to resell it or parts of it
Table 3.5	Buyer resells a resource (buyer=seller)
Table 3.6	Provider sells a resource (seller=provider)
Table 3.7	Buyer interacts with broker
Table 3.8	Seller interacts with broker
Table 3.9	Rating agency rates a provider based on monitoring data

Table 3.2.: Use cases: Interaction between the components and roles of the model

Step	**Role**	**Description**
1	Buyer	places buy order on the exchange
2	Exchange	matches order and informs about successful match (trade)
3	Buyer	initiates the secure clearing and settlement process

Table 3.3.: Use case 1: Buyer buys a resource with the desire to use it (buyer=consumer)

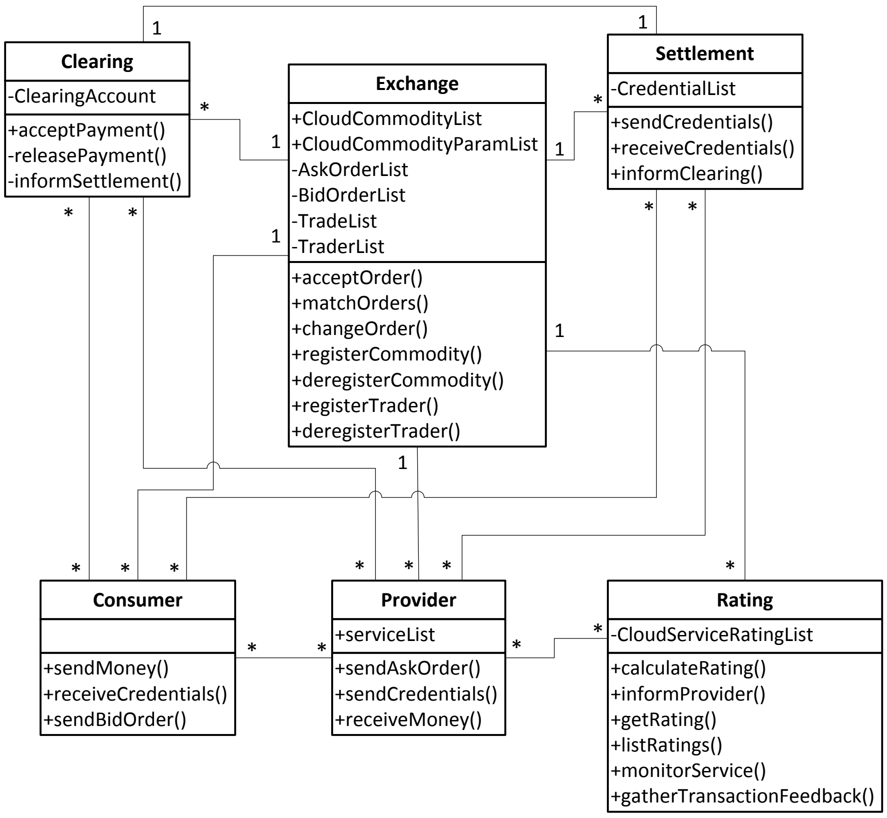


Figure 3.3.: UML diagram of the model

Step	Role	Description
1	Buyer	places buy order on the exchange
2	Exchange	matches order and informs about successful match (trade)
3 or 3	Buyer Buyer	initiates use case: Buyer resells a resource (Buyer=Seller) initiates the secure clearing and settlement process
4	Buyer	splits resource pool in two parts: resources to be used and resources to be resold
5	Buyer	informs provider about resell and the amount of resources to be resold
6	Provider	locks resources to be resold, creates credentials for these resources, and sends the credentials to the buyer
7	Buyer	initiates use case: Buyer resells a resource (buyer=seller)

Table 3.4.: Use case 2: Buyer buys a resource with the desire to resell it or parts of it

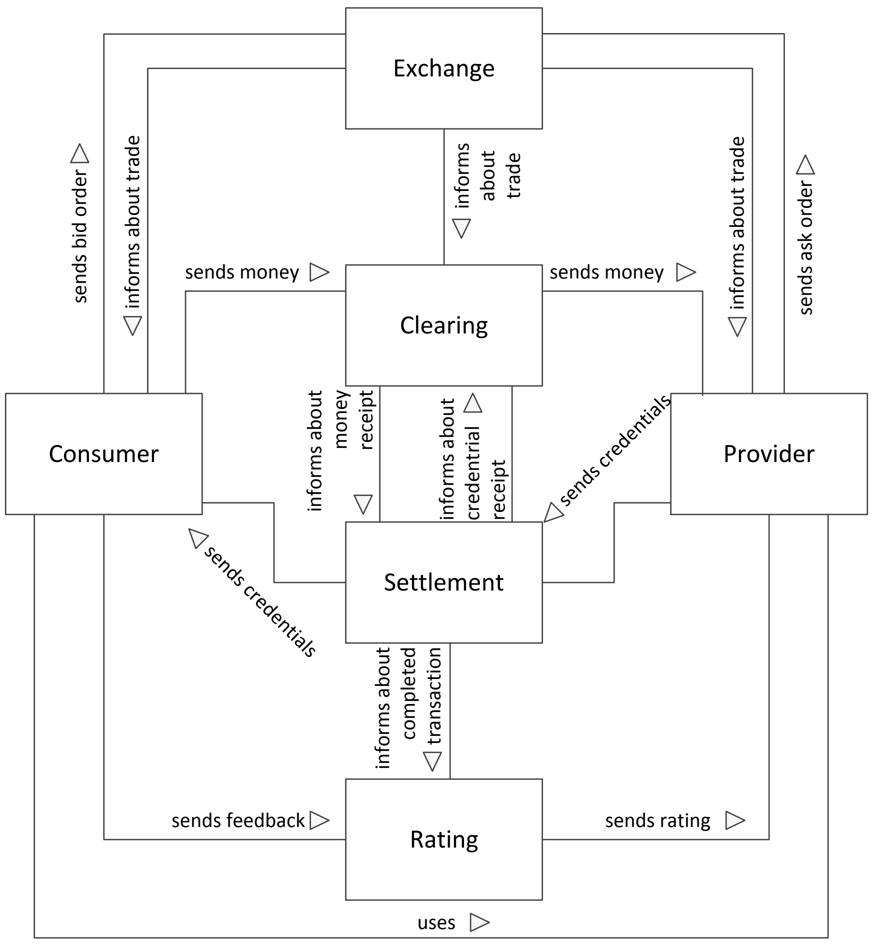


Figure 3.4.: The information model

Step	**Role**	**Description**
1	Old Buyer=Seller	places sell order on the exchange and awaits secure clearing and settlement process
2	New Buyer	initiates secure clearing and settlement process

Table 3.5.: Use case 3: Buyer resells a resource (buyer=seller)

Step	**Role**	**Description**
1	Seller	places sell order on the exchange
2	Exchange	matches order and informs about successful match (trade)
3	Seller	awaits the secure clearing and settlement process

Table 3.6.: Use case 4: Provider sells a resource (Seller=Provider)

Step	**Role**	**Description**
1	Buyer	sends a resource buy request to the broker
2	Broker	matches resource buy request with exchange traded cloud computing commodities
3	Broker	places buy order on the exchange
4	Exchange	matches order and informs about successful match (trade)
5	Broker	initiates the secure clearing and settlement process
6	Broker	sends credentials to the Buyer

Table 3.7.: Use case 5: Buyer interacts with broker

Step	**Role**	**Description**
1	Seller	sends a resource sell request to the broker (potentially including the credentials)
2	Broker	matches resource sell request with exchange traded cloud computing commodities
3	Broker	places sell order on the exchange
4	Exchange	matches order and informs about successful match (trade)
5	Broker	awaits the secure clearing and settlement process
6	Seller	transfers the money to the seller

Table 3.8.: Use case 6: Seller interacts with broker

Step	**Role**	**Description**
1	Rating	collects data by permanent monitoring and/or statistical monitoring of cloud services
2	Rating	calculates provider/cloud service ratings and publishes those ratings

Table 3.9.: Use case 7: Rating agency rates a provider based on monitoring data

3.2.6. Secure clearing and settlement

The cloud resources or services traded as cloud computing commodities need to be delivered after a trade is closed. As the resources remain in the data centre of the provider, the question is what is actually delivered in cloud computing commodities market. Despite the right to use the cloud service, there is a *good* which is delivered from seller to buyer via a settlement service in the form of credentials enabling to access the cloud resources.

The process of clearing and settlement of cloud computing commodities has to be secured in order to be able to prevent fraud on the one hand, and on the other hand keep the anonymity of the transaction partners during the clearing and settlement. In this section, a concept for securing the clearing and settlement is provided based on public key cryptosystems.

In the model proposed in this work clearing and settlement together with the exchange are the core elements of the framework for trading cloud computing commodities. A requirement for this model is that every market participant (buyer, seller), the clearing service, and the settlement service has a private and a public key. An overview of the messages which are sent between the buyer, seller, clearing service, and settlement service is provided after defining the functions sig for digitally signing, $encr$ for encrypting, and $decr$ for decrypting. t_{ID} represents the unique ID number of the trade. This ID number is created by the exchange and provided to the buyer and seller. The t_{ID} might be extended with a timestamp. sk_X is the secret key with $X \in \{B, C, S, Set\}$, where B...Buyer, C...Clearing, S...Seller, and Set...Settlement. The public key (pk_X) is defined accordingly. The final clearing step requires a confirmation message ($succ$) containing the t_{ID} and a success message including a timestamp.

- $sig(key, data)$ computes the digital signature of given data with the provided key by encrypting the hash value of data with the key
- $encr(key, data)$ encrypts the given data with the provided key with an asymmetric cryptosystem
- $decr(key, encrdata)$ decrypts the given encrypted data with the provided key with an asymmetric cryptosystem

$$a = (sig(sk_B, t_{ID}), t_{ID})$$

$$a' = (sig(sk_C, a)$$

$$b = (sig(sk_C, t_{ID}), t_{ID})$$

$$cr_1 = (sig(sk_S, cred), encr(pk_{Set}, cred), t_{ID})$$

$$setc = (sig(sk_{Set}, t_{ID}), t_{ID})$$

$$c = (sig(sk_S, t_{ID}), t_{ID})$$

$$d = (sig(sk_C, succ), succ)$$

$$cr_2 = (sig(sk_{Set}, cr), cr), cr = (sig(sk_S, cred), encr(pk_B, cred), t_{ID})$$

Figure 3.5 provides an overview of the messages, payments, and credential information sent from one entity to another. The flow of payment and credentials is structured in nine steps and must be processed in the following order to ensure a secure information exchange and a secure payment and settlement process:

1. The flow is initiated by buyer calculating a with its secret key sk_B and the identification number for the trade represented by t_{ID} and sending the payment and a “contract” in the form of a to the clearing service.
2. The clearing service checks the signature of a with the public key of the buyer pk_B and calculates a' by signing a with its secret key sk_C and computes b by signing t_{ID} with its secret key. After that, the clearing service sends a' to the buyer and b to the seller. Based on the trade ID which contains the IDs of both, the buyer’s and the seller’s offers, the seller can be determined only based on the trade ID.
3. The seller checks the signature of b with the public key of the clearing service and encrypts and signs the credentials (creating cr_1) by signing the credentials with pk_S and encrypting the credentials with the settlement service’s public key pk_{Set} before sending cr_1 to the settlement service.
4. The settlement service checks validity of the signature of the received credentials and stores the cr_1 in a database. The clearing service calculates $setc$ and sends it to the clearing service and to the seller in order to confirm the receipt of the credentials.
5. This step starts with validity checks performed by the seller and the clearing service for the received $setc$. Then the clearing service sends the payment to the seller.
6. The seller confirms the receipt of payment by sending c – which is computed with the seller’s secret key – to the clearing service.
7. The clearing service checks the validity of the signature of c and computes d. As a confirmation for buyer and seller, d is sent to both of them. For the buyer, d is necessary for

retrieving the credentials from the settlement service. This step is the final step for the clearing service.

8. For obtaining the credentials from the settlement service, the buyer sends d to the settlement service. The entity in charge for the settlement proofs the validity of d which contains the trade id t_{ID}. If the proof is successful cr_1 will be fetched from the database, the credentials decrypted with the settlement's secret key sk_{Set} and encrypted with the buyer's public key pk_B.

9. The settlement service sends cr_2 to the buyer. With this the buyer is able to decrypt the credentials with its secret key and use the containing access information to connect to the cloud service.

Trust relationships in clearing and settlement

Placing buy and sell orders on an exchange together with securing the clearing and settlement requires a trust relationship between the different entities realised by the introduction of a Public Key Infrastructure (PKI). This section provides an introduction to PKIs, shows an overview of the PKI in a cloud computing commodity exchange model, and gives an example of the realisation of this specific PKI.

Public key infrastructure

To ensure secure information and key exchange between trusted entities the framework makes use of a Public Key Infrastructure (PKI). PKIs are described in several publications (e.g. [59], [48]). A PKI is based on public key cryptosystems. Rivest, Shamir, and Adleman have proposed a well-known method enabling digital signatures and public/private key encryption in 1978 in *A method for obtaining digital signatures and public-key cryptosystems* [70]. A person or institution with the desire to get a digital certificate has to send a certificate signing request to an entity called *registration authority (RA)*. A certificate signing request contains information about the sender, the validity date, and the public part of a key pair (generated in the process of creating the request). The registration authority is responsible to check the legal status of the person or entity behind the certificate signing request. This process of checking might be a manual or personal transaction. If the check is passed the registration authority forwards the certificate signing request to the certificate authority (CA). The CA signs the request with its private key and issues the digital certificate to the originator of the request.

As the certificate is valid after signing, there is no other way than listing it in a directory called revocation list to void the certificate in case it should not be used for certain reasons (like the private key gets stolen). A revocation list implies that each client has access to the revocation list and checks if the revocation list contains the certificate. The certificate itself remains mathematically functional and valid.

Figure 3.6 shows the interaction between the entities in a PKI. The entity intending to

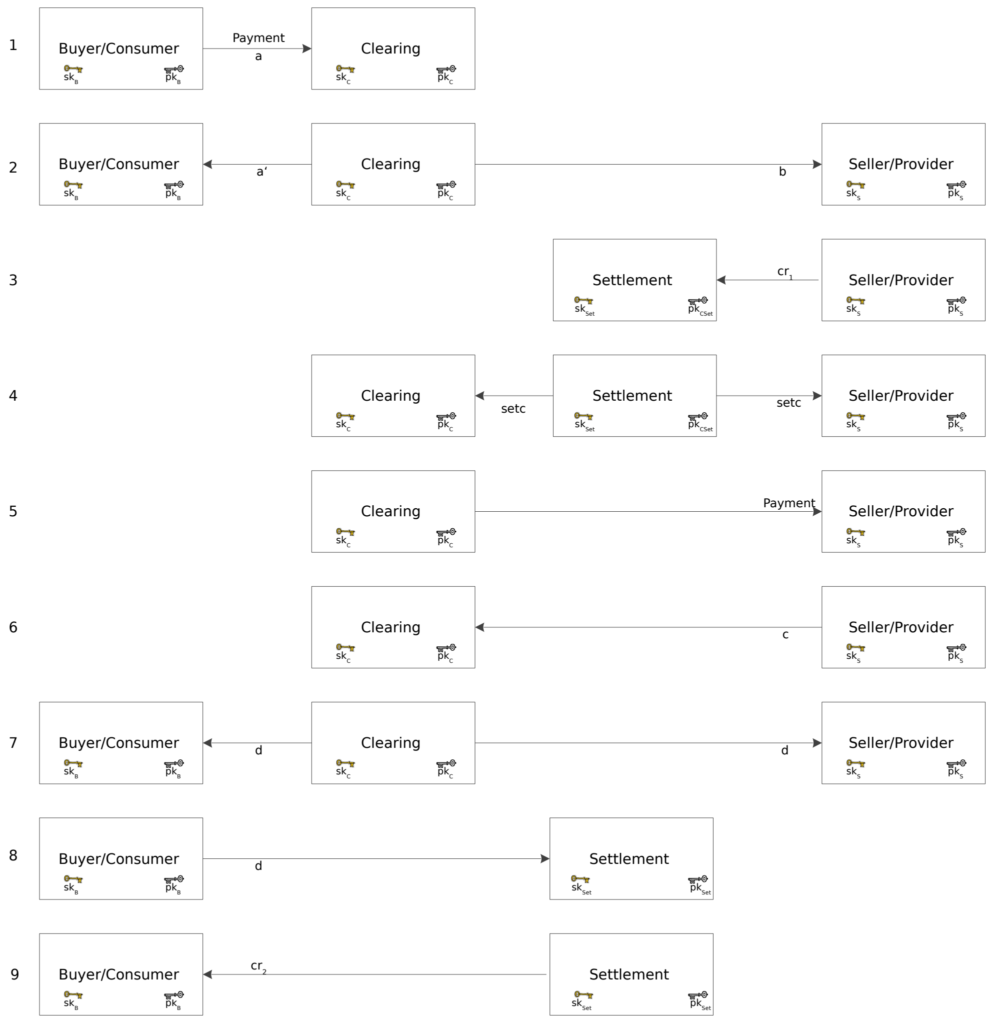


Figure 3.5.: Flow of events in a secure clearing and settlement model for cloud computing commodities

get a certificate (on the bottom) sends a certificate signing request to the registration authority (center box) together with providing an id and requested documents to prove its identity. If the registration authority approves the requesting entity, the certificate signing request is sent to the certificate authority (top left box) which signs the certificate and provides the user with the signed certificate. Additionally, the certificate authority maintains a revocation list (top right box) for certificates that have been stolen or are not allowed to be used for certain reasons. The concept of a revocation list is necessary, as certificates are mathematically valid until their expiration date.

A PKI for a cloud computing commodities exchange model

This section shows a PKI for the model of a cloud computing commodities exchange and a brief description of the entities in this PKI.

- Exchange: The exchange acts as the central trust entity – the certificate authority. All the participants including buyers, sellers and entities offering clearing, settlement, and brokering services have to initially trust the exchange. The exchange contracts one or more entities providing registration authority capabilities. The exchange takes care of a revocation list containing certificates with a valid expiration date but revoked by the exchange or the participant.
- Registration Authority: Before traders (buyers, sellers, brokers) and participants offering clearing and settlement services are approved to interact with the exchange or other participants and the certificate authority (the exchange) signs their certificate signing request, the registration authority proofs their identity and legal status, and checks if they meet the requirements to trade or offer clearing or settlement services.
- Clearing and Settlement: Clearing and settlement require a signed certificate from the CA in order to guarantee a trust relationship and to perform secure key interchange. They are linked in the secured process of closing a trade. Entities offering clearing services may sign certificates (and therefore express their cooperation, trust, and direct interaction) with certain settlement service providers and vice versa.
- Trader: Traders (buyers, sellers, brokers) need to obtain a signature for a certificate signing request before they can start to place their orders on the exchange. The traders trust each other transitively via the exchange (the certificate authority) – each one of the traders trusts the exchange and the exchange trusts it's registered traders.
- Rating agency: A rating agency requires a valid certificate signed by the exchange in order to offer rating services in the cloud computing commodity trading environment. In Section 4.5, a method based on a combination of a public key infrastructure and mechanisms of the web of trust integrating consumer ratings in the centralised rating approach is described.

In Figure 3.7 the PKI for an framework enabling the trading of cloud computing commodities is shown.

3.2.7. Matching the model and the requirements

Table 3.10 lists the identified requirements for a marketplace for cloud computing resources and shows that they are fulfilled by the model presented in this work.

Functional Requirements	
Requirement	This Work
1. The system has to be able to regulate supply and demand.	✓
2. The system has to define a method to limit the access to a group of (verified) users.	✓
3. The system needs to accept requests in a pre-defined form from consumers and providers.	✓
4. The system has to provide information about available resources or classes of resources.	✓
5. The system has to be able to bring together a certain number of consumers with a certain number of providers.	✓
6. Computing services have to be classified or grouped based on quantity, quality, and performance parameters.	✓
7. The system has to provide the capability to compare service offers from different providers.	✓
8. A method to provide information about the quality and reliability of computing services and their providers has to be established (rating mechanisms).	✓
9. The model needs to take into account quantitative, qualitative, and reliability parameters to price computing services.	✓
10. A mechanism for the pricing of cloud computing resources based on the current supply and demand have to be provided.	✓

Non-Functional Requirements	
Requirement	This Work
1. Standards have to be used to define the service classes.	✓
2. Legal constraints (for data privacy) have to be met by the infrastructure and computing service classes.	✓
3. The system should be designed in a way to avoid performance bottlenecks.	✓
4. Users should be able to identify matching computing services without complex brokering.	✓
5. Rating mechanisms should be designed in a not-only centralised approach.	✓
6. A decoupling of trading from the resource delivery.	✓
7. A mechanism to guarantee prices for both providers and consumers of resources for future usage.	✓

Table 3.10.: Matching of requirements and the presented approach for exchange-based trading of cloud computing commodities

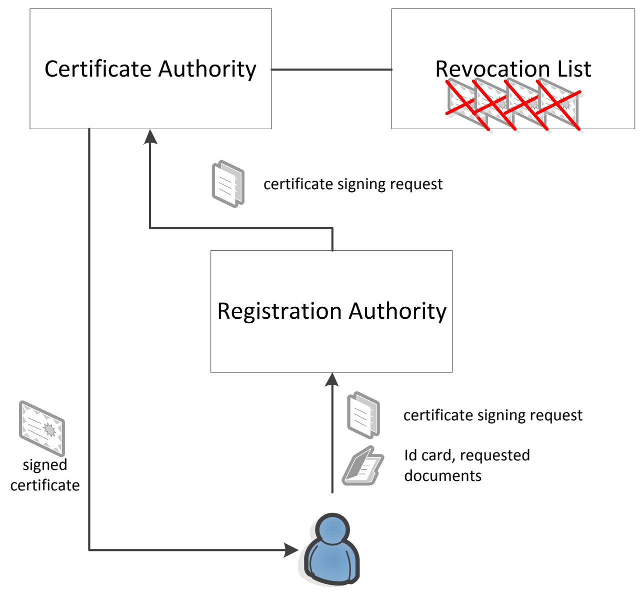

Certificate Authority
Revocation List
certificate signing request
Registration Authority
certificate signing request
Id card, requested documents
signed certificate

Figure 3.6.: Public key infrastructure

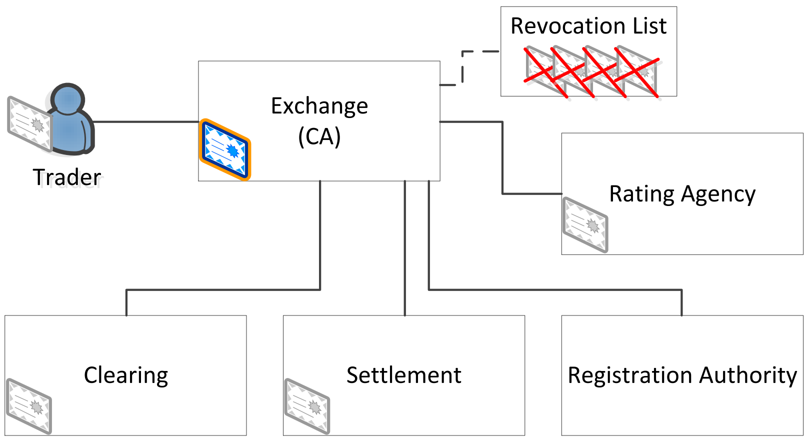


Figure 3.7.: PKI for a cloud computing commodities exchange

Chapter 4

Cloud Computing Commodities

The basis for exchanges as outlined in the previous chapter are standardised goods or commodities. This chapter introduces the concept of cloud computing commodities and the requirements and parameters for their definition. Cloud computing commodities describe a classification of cloud services to enable the creation of tradable goods and achieving comparability. An important input for the comparability and the pricing of a cloud computing commodity is the reputation or the rating of the provider offering the cloud service. Therefore a method for rating cloud providers is presented as a combination of a centralised rating agency and consumer feedback.

4.1. Definition and overview

Cloud computing commodities describe a classification of cloud services. For every cloud computing commodity, requirements are defined together with providing information about the performance of certain applications when executed on the cloud computing commodity resources. Cloud computing commodities are *not* a mechanism to compare all cloud services but enable the comparison of cloud services belonging to a certain class.

A cloud computing commodity is described by a set of parameters – *cloud computing commodity parameters*. The different nature of the existing cloud services implies the creation of a meta-model for the parameters as the those can range from countable objects, to sets of objects or even a description of a certain application functionality.

This work does not provide a list of possible parameters that would be necessary to describe all existing cloud services because it could only be a snapshot of the parameters of today's cloud services and would be incomplete as soon as new services that introduce new parameters are developed. The presented approach is open and enables the description of current and future cloud services.

4.2. Parameters describing cloud computing commodities

For describing a cloud computing commodity, a parameter set is necessary. The elements of this set – the parameters – can be split in two groups.

- **Cloud service parameters** describe the cloud service belonging to the cloud computing commodity class and provide information about its functionality and way of access.
- **General parameters** describe non-functional properties of a cloud service. They provide information about additional requirements for cloud computing commodities including information about service levels, location and legal constraints.

A parameter P is defined as the tuple:

$$P = (name, value, threshold, f_g(value, threshold), f_e(value, threshold), f_l(value, threshold))$$

This tuple is the basis for the standardisation and the comparability.

- $name$ The identifier of parameter - a string value.
- $value$ An object containing the value of the parameter for a certain cloud computing commodity. The $value$ can range from integer numbers to strings (in terms of application functionality descriptions) or sets of values or other objects.
- $threshold$ The $threshold$ is of the same type as $value$ and denotes the “minimum” value (in the sense of the previously defined operators), which has to be achieved for a certain cloud computing commodity.
- f_g This function defines an algorithm to evaluate $value > threshold$ to **true** or **false**.
- f_e This function defines an algorithm to evaluate $value = threshold$ to **true** or **false**.
- f_l This function defines an algorithm to evaluate $value < threshold$ to **true** or **false**.

In the following, an exemplary list of important cloud computing commodity parameters are given. This list represents parameters that are able to describe a cloud computing commodity based on existing cloud services. It is not intended as a complete list because cloud computing is evolving fast and new cloud services are developed constantly. Instead, it should serve as an example for the general parameter model.

- Cloud service parameters
  - Cloud resource information parameters
  - Cloud resource access parameters
  - Data transfer parameters
- General parameters
  - Service Level Agreement (SLA) parameters
  - Benchmark parameter sets
  - Legal and audit parameters
  - Locality parameters

Cloud service parameters describe the nature of a cloud service and information the consumer needs to access the service and other functionality, whereas *general parameters* provide information of non-functional properties of a cloud service. In the following, the cloud service parameters and the general parameters are described in more detail:

Cloud resource information parameters provide information about the nature of a cloud service, its basic properties, and the application(s) or the type of application(s) this commodity is intended to serve.

An example set of cloud resource information parameters for Infrastructure-as-a-Service (IaaS) offers could include the following parameters:

- CPU
- RAM
- Virtual machine (VM) image format
- VM storage
- Speed of the interconnect between virtual machines
- Speed of the up/downlink per customer
- Type of application

For Platform-as-a-Service (PaaS) this information could include the supported programming languages or available libraries for compiling, running, and executing applications.

The resource parameters for Software-as-a-Service (SaaS) are weaker. In general, cloud services of the SaaS layer offer a certain functionality or application. The resource parameters represent a description of this functionality. An example of such a description could be the conversion of pictures in certain input formats to a certain output format. Every service meeting the description and being provided in a way to meet the remaining cloud computing commodity parameters can be offered as the corresponding cloud computing commodity.

Cloud resource access parameters describe the way of accessing cloud resources. This way is defined in the cloud computing commodity as the resource access parameters. In general, resource access parameters provide a description of the API (Application Programming Interface) of a cloud service or its management interface. For designing the interfaces the use of open standards should be considered to open the market to a broader base of consumers and providers and avoid the pushing of de-facto standards or proprietary interfaces. A more detailed view on standards is provided in this work in Section 4.3.1.

Data transfer parameters provide information about the network characteristics between the corresponding entities, data amounts, information about who is responsible for the potential data transfer, and information who pays for the potential data transfer. With the introduction of standardised terms in this work (Section 4.3.3), this information can be represented in a unique and standardised way. Data transfer parameters are important for the transfer of large amounts of data when switching providers.

SLA parameters consist of Key Performance Indicators (KPIs) and corresponding thresholds. SLA parameters may also include information about penalties in the case of non-delivery.

Benchmark parameters consist of two parts. The first one is the unique description of the benchmarks and/or the set of benchmarks $((b_1, b_2, \ldots, b_n), n \in \mathbb{N})$ used for measuring the performance of the offered cloud service and a unique description of the prerequisites of the benchmarking process. The second part provides a threshold $((tr_1, tr_2, \ldots, tr_n), n \in \mathbb{N})$ representing the minimum result which has to be achieved running the benchmark or the set benchmarks (the benchmark result has to be *better or equal* than the corresponding threshold). $((br_1, br_2, \ldots, br_n), n \in \mathbb{N})$ represents the benchmark result achieved by the benchmarks $(b_1, b_2, \ldots, b_n)$.

For each of the benchmarks the *better or equal* relation $\succeq$ has to be defined as $\leq$ or $\geq$. For the benchmark results of a cloud service which is offered as a certain cloud computing commodity the following has to hold:

$$\{\forall i \in \mathbb{N} \wedge i \leq n : br_i \succeq tr_i\}$$

Potential consumers of the cloud computing commodity get information about the performance of certain application types running on the cloud resources offered as a certain cloud computing commodity with the performance numbers of the benchmarks and the threshold values for these benchmarks. An overview of benchmarks for quantifying cloud computing commodities is given in Section 4.3.2.

Legal and audit parameters represent requirements for certain audits (the data centre has to have certain audit certificates) or legal constraints (e.g. data privacy).

Locality parameters Locality parameters provide information about the location of the data centre (and the data). These parameters may be linked to the legal parameters in order to meet certain requirements for data locality (e.g. banking data, personalised health records are not allowed to be stored outside the country of origin). Despite the legal reason, locality refers to latency and cost of data transfer.

The definition of a set of parameters for cloud computing commodities enables the comparability of the offered services. A provider can only offer services as a cloud computing commodity if the service parameters comply with the cloud computing commodity parameters.

Cloud services represent non-storable goods relying also on time as dimension. (This is a similarity to electricity markets.) Therefore, the start and end time of the service usage is part of the cloud computing commodity traded on an exchange. In the case of hourly auctions, a day would be divided into 24 different buckets the cloud computing commodity. This would imply 24 commodities per commodity and day.

Advance reservation of cloud resources (as mentioned in [76]) and the possibility to guarantee certain costs can be realised with the help of derivatives, namely futures and options as described in Chapter 3.

4.3. Standardisation of cloud services – Introducing new cloud computing commodities

Goods have to be commoditised before they can be traded. In the case of cloud computing commodities, these commoditisation has to be done by the exchange. Standardised buckets of goods have to be defined together with specifying a way to measure amount and quality of the good. The cloud computing commodities listed on an exchange have to be described by a set of parameters in a way to satisfy both, providers and consumers before they can be traded. A way to specify cloud computing commodities is to collect requirements of the consumer and the providers side represented by a set of potential cloud computing commodity parameters in order to gain a high market liquidity. After this collection is done, the "greatest common denominator" has to be found.

Ways to find such "greatest common denominators" are consensus processes. Previous work on multi criteria decision making (e.g. [78]) can act as a basis for this specification.

The exchange has to ensure an open and neutral process of specifying cloud computing commodities in order not to favour certain consumers, providers or groups of them. Non-neutrally specified cloud computing commodities would result in a market that could be manipulated easily by certain participants.

In this section, existing standards in the field of cloud computing that can be used in order to define a cloud computing commodity are presented together with ways to quantify performance and standardise the data transfer between consumers and providers or between providers.

4.3.1. Standards for cloud computing commodities

With standardised cloud computing commodities, consumers are able to compare different offers and know in advance what they can expect from a service. Currently, in cloud computing, consumers have very little information about the performance of the cloud service when running certain applications before they use the cloud service. By providing the type of application, the cloud computing commodity tells the consumers for which applications the offered cloud computing services are optimised.

Looking at the previously identified set of parameters, for some elements of this set, standards could help the faster and easier adoption. The element where standards are necessary the most are resource access parameters. Consumers who want to use services offered as a cloud computing commodity usually interact with this cloud service over a certain interface. They integrate the way of accessing a resource over this interface in their client software and production environments. The usage of standards for the resource access parameters is crucial for enabling a fast provider switching (in terms of the service interface). Consumers don't have to change their client software. They can access different provider offers by just changing the credentials.

Another element are the SLA parameters. Consumers or buyers of a cloud computing commodity might not use the service they get but resell the service. They intend to offer the service with additional functionality and a certain quality of service. If the SLA parameters are provided in a standard form, resellers don't have to adapt the SLA parameters for their service offers when changing the cloud provider.

Benchmarks are a unique way of measuring the performance of resources or services but they are not considered as a standard in this work. The benchmarks used to measure the performance of cloud services offered as a cloud computing commodity are defined in the cloud computing commodity itself. All services have to meet the same performance minimums if offered as a certain cloud computing commodity. More details on benchmarking is provided in the following section (Section 4.3.2).

Various Standards Developing Organisations (SDOs) have created standards which are relevant for cloud computing commodities. Five examples are given in the following list.

- OCCI (Open Cloud Computing Interface, OGF - Open Grid Forum) [37], [38], [39]
- CDMI (Cloud Data Management Interface, SNIA - Storage Networking Industry Association) [3]
- CIMI (Cloud Infrastructure Management Interface, DMTF - Distributed Management Task Force) [4]
- OVF (Open Virtualisation Format, DMTF - Distributed Management Task Force) [2]
- Cloud SLA Application Note, Version 1.2, (TMForum - TeleManagement Forum) [6]

For data transfer in cloud environments there is no standard. A proposal to handle the data transfer in a standardised way is provided in this work in Section 4.3.3. The terms used are a form of standardising the exchange/transfer of large amounts of data. These terms should primarily give a standardised way to clarify the responsibilities of the different parties (at least a sender and a receiver) involved in a data transfer transaction.

4.3.2. Quantifying performance

A core criteria for the quality of a cloud service is performance. The desire to measure the performance of cloud services implies the classification of those services inside the three layers IaaS, PaaS, and SaaS. Cloud computing commodities provide a reasonable and easy way to group cloud services in corresponding classes.

For measuring the performance of IT resources, benchmarks have been developed. With benchmarks, the peak performance of various parameters of a system can be determined. In the process of the definition of cloud computing commodities, application characteristics are crucial criteria. Benchmarks are broadly used to determine the performance of a system. In the case of cloud computing commodities, the threshold value of a benchmark has to be reached at any utilisation level and at any time. There is no "benchmark-scenario" like a completely empty data centre to run the benchmark.

Cloud services are based on virtualisation of the underlying hardware resources, which means the hardware resources can be utilised in a more optimised way but also lead into the problem of non predictable workloads in different virtual machines on the same hardware. In designing benchmarks, virtualisation has to be taken into account as well as overbooking rates of the underlying hardware. With overbooking, more resources than physically available are sold virtually. The overbooking rate is usually determined by the average utilisation of the systems (similar to airline booking systems [76]). There is no problem until the current utilisation reaches the maximum the hardware can be utilised.

As consumers have their specific applications, workloads, and jobs in their minds when it comes to the selection of a cloud service, the cloud computing commodity has to provide information about the performance of certain applications relevant for a broad spectrum of consumers. These consumers are interested to get their computations done as quickly as possible. In their work *Evaluation and performance of computers: application benchmarks: the key to meaningful computer evaluations* ([54]), Joslin and Hitti talk about the importance of the information on the performance of a system regarding certain applications. In this work, the question pointed out in the paper, how long it takes to compute a certain (the consumer's) workload, is not the primary target of cloud computing commodities. The core is to provide the consumer with information on the performance of a generic workload represented by an application benchmark or a set of benchmarks executed on a cloud service offered under the umbrella of a certain cloud computing commodity.

A set of benchmarks, which has been developed in order to gain significant results according to certain applications or workloads, is pooled in the TPC benchmarks published by the Transaction Processing Performance Council (TPC) [10][1]. These benchmarks allow the measurement of the performance of a predefined application/software set-up. Table 4.1 provides an overview of the existing TPC benchmarks (TPC-C, TPC-DS, TPC-E, TPC-H, TPC-VMS, TPC-Pricing, TPC-Energy) based on the information provided by the TPC. The obsolete benchmarks TPC-A, TPC-B, TPC-D, TPC-R, TPC-W, and TPC-App are not considered in the table. In general, the TPC benchmarks are used to measure the performance of a certain hardware infrastructure. They have not been intended to provide performance information about cloud infrastructures.

In [20] the drawbacks of the TPC benchmarks in cloud computing environments are described. Additionally, the paper identifies requirements for cloud benchmarks, which are not covered by TPC. A new idea for cloud benchmarks is briefly described in the following. The authors suggest the that a cloud benchmark should be based on the TPC-W benchmark (transactional web e-Commerce benchmark) with certain extensions. The idea is to have a holistic benchmark instead of running a set of micro benchmarks. Their condition for a cloud benchmark is the ability of a dynamic system to adapt to changing load according the costs of scalability. This benchmark idea aims to measure the (hypothetically) infinite scalability of cloud computing applications. Multi-location data replication has to be considered as well when designing a cloud benchmark. The authors propose a three layer consistency model of guaranteed performance. In their model cloud services can be offered as:

[1]Transaction Processing Performance Council – http://www.tpc.org

Name	Description
TPC-C	On-line transaction processing benchmark. TPC-C measures the performance of an order-entry environment with a simulation of a computing environment consumers execute database transactions.
TPC-DS	Decision support benchmark. TPC-DS provides performance information of decision support systems that have to examine large data sets and experience high CPU load and IO.
TPC-E	On-line transaction processing benchmark with respect to brokering firm workloads. TPC-E is able to simulate workloads of brokerage firms measuring transactions related to trades and account information on the one side and to exchange systems (financial) on the other side.
TPC-H	Ad-hoc decision support benchmark. TPC-H is able to measure ad-hoc queries and concurrent data-modifications. The *TPC-H Composite Query per-Hour Performance Metric (QphH@Size)* is the metric of TPC-H. It can be extended with price resulting in $/qphH@Size.
TPC-VMS	Virtual Measurement Single System Specification. TPC-VMS provides information of the performance of transactions on virtualised databases. Three database workloads each represented by one of the benchmarks TPC-C, TPC-E, TPC-H, or TPC-DS are run on one server. The minimum performance of the three workloads is the TPC-VMS result.
TPC-Pricing	Consistent pricing information for all TPC benchmarks. TPC-Pricing provides a single pricing specification holding for all pricing results of TPC benchmarks.
TPC-Energy	Energy metrics for TPC benchmarks. Extension of existing TPC benchmarks with energy metrics.

Table 4.1.: TPC benchmarks

- low performance: Only basic performance guarantees are given (e.g. overbooking could cause performance degradations)
- medium performance: A mixture of basic performance guarantees in combination with high performance guarantees for certain defined attributes is provided.
- High performance: All transactions meet guaranteed high performance criteria.

Another aspect is the fault tolerance of a cloud system together with self-healing capabilities. An approach of fault tolerance and self healing mechanisms in cloud environments has been presented in *An Automated Approach for Fault Recovery Planning in Science Clouds* [32].

Ideas how benchmarking can support measuring cloud services of the three layers of cloud computing – IaaS, PaaS, and SaaS – are provided in the following.

Before measuring the performance of IaaS, application classes have to be defined. An application class is an abstraction of applications with similar characteristics and requirements. Examples of such application classes would be a class for numerical applications with low diskIO or a class of applications that need high bandwidth. For each of these classes, a set of benchmarks (containing at least one benchmark) has to be defined with corresponding threshold values and the definition of the *better or equal* operators for each of the benchmarks. Figure 4.1 shows that consumers use the services offered as the cloud computing commodities with the application class fitting the application they intend to run.

For PaaS, applications have to be assigned to classes for the performance measurements with a benchmark or set of benchmarks available for each of the classes.

SaaS is a special case in terms of application benchmarks as each of the SaaS services represents a certain application or functionality. There are no application classes because there is only one application provided by cloud service offered as SaaS. The performance measurement is in this case given by the application itself. The application is benchmarked by providing a sample workload which has to be completed by the cloud service to test. If the result of this SaaS benchmark is better than the threshold value given in the cloud computing commodity parameters the cloud service can be offered as the desired cloud computing commodity.

The upcoming section deals with the standardisation of the data transfer either between the consumer and the provider or between two provides. The following standardised data transfer concept has been developed based on an approach observed in the commercial delivery of goods.

4.3.3. Data transfer

Motivated by the *provider switching scenario* (Section 2), a concept for standardised cloud data transfer based on a model used in commercial trading is presented in this section. Standardised terms for data transfer enable this clarification together with the definition of responsibilities. A consumer with the desire to move from one cloud provider to another cloud provider offering a service of the same cloud computing commodity type has to take care of the movement of virtual machines and data. It might not even be possible to transfer the data directly from the old provider to the new one.

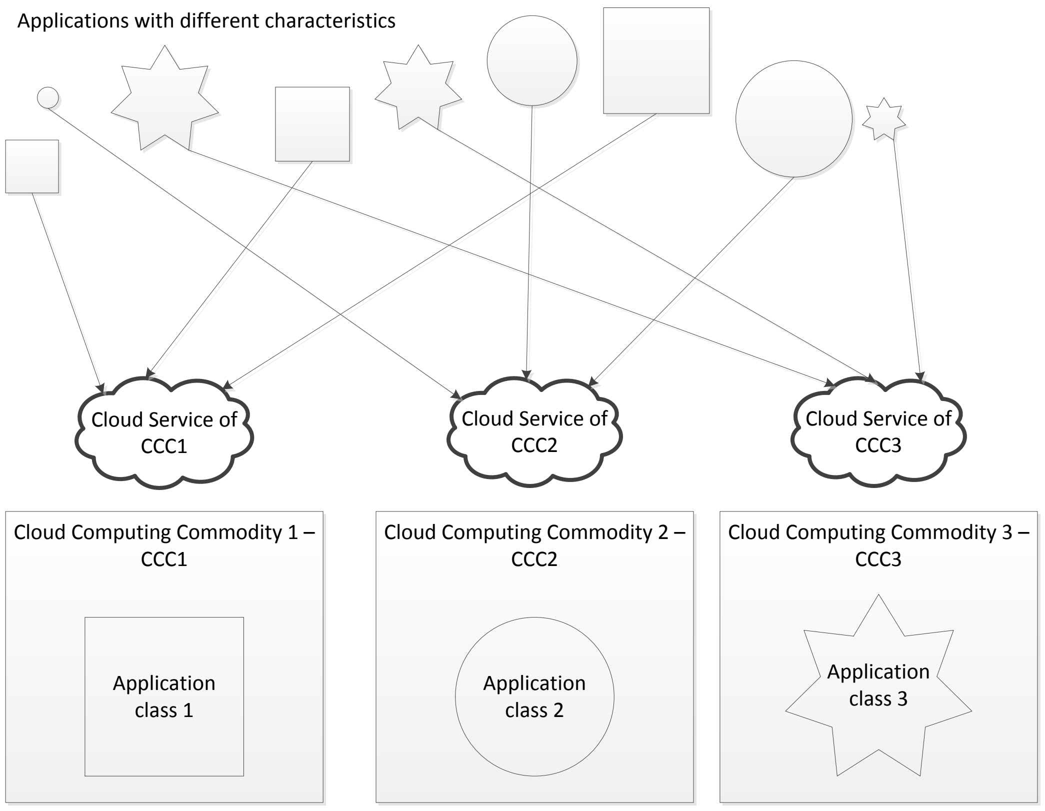

Applications with different characteristics
Cloud Service of CCC1
Cloud Service of CCC2
Cloud Service of CCC3
Cloud Computing Commodity 1 – CCC1
Application class 1
Cloud Computing Commodity 2 – CCC2
Application class 2
Cloud Computing Commodity 3 – CCC3
Application class 3

Figure 4.1.: Application classes

The delivery of (traditional) commodities is defined by a set of standardised terms created by the *International Chamber of Commerce* – the *Incoterms*. The current edition of the Incoterms is the Incoterms 2010 after several updates of the initial version of 1936. The Incoterms are used to define who is paying what during the process of the delivery of a good including transport, insurance and taxes. These terms are depicted in Table 4.2 where last four terms refer to the transport of goods on water (sea and inland).

Cloud services run in provider's data centres. The delivery of a cloud computing service is not the usual delivery of a good which is shipped from an origin to a destination or electricity which is available through a power grid and can be consumed at the location where it is needed. This is not a problem until large amounts of data are involved. Cloud computing services used to apply certain computation on certain data, require the data available in their data centre. The data has to be transferred from its origin (which might be another cloud provider) to the data centre fulfilling the contract. Since the transfer of large amounts of data is costly (e.g. leased lines) and consumes significant amounts of time, it is a crucial part of every contract and also an important part of a cloud computing commodity since otherwise the comparability of different offers can not be guaranteed.

Commoditising cloud computing services is one part of the delivery of the service but there is still the lack of a standardised way for the transfer of the data in the sense of a commercial delivery. The parts taken into account are data preprocessing, encryption, direct transport, transport over a storage centre or physical shipping, decryption, postprocessing, and insurance. With this approach the questions

- "Who is responsible for what part of the data transfer?"
- "Who is taking which risk?"
- "Who pays for what?"

are solved based on an international standard (Incoterms, [64] for commodity delivery implemented successfully on a global level. A proposition of Incoterms applied to data stored and/or processed in cloud computing environments is provided in this section.

In Table 4.2 the Incoterms in the edition of 2010 are listed together with the expanded abbreviations and the descriptions of how the terms are applied to cloud data transfers. With this, the transfer of data is defined for direct transfer or for transfer over a storage centre between origin and destination. With this model also insurances for the data transfer (e.g. for the case of data transfer not completed at a certain time) can be realised.

To each cloud computing commodity with a certain amount of storage a type (term) of data transfer has to be assigned. This type defines the way the data transfer of an amount of data equivalent to the amount of storage available in the cloud computing commodity or defined in the cloud computing commodity. In terms of traditional commodities one of the generally used terms is *FOB – Free On Board* (see [46]).

Table 4.3 shows the responsibilities of providers and consumers for each of the terms. It also provides an overview of who is covering which costs. The columns "Transport to Storage

Centre", "Transport cost", and "Transport from Storage Centre" refer to a transfer via a storage centre between origin and destination and the column "Direct Transport" refers to a transfer from origin to destination on a direct route (without "buffering"). The two different ways of transport exclude each other.

The party responsible for the data transfer can choose different ways for transferring the data from one data centre to another as long as it complies with the agreed constraints like maximum transfer time. In some cases the transfer of the data by writing the data on tapes or harddisks and shipping them by mail (physical) may be faster and cheaper. A combination of two ways of transfer if possible in the case of the terms CFR and CIF where data is sent to a storage centre. Storage centres might offer services for getting data electronically and prepare the data for a physical shipping to the provider or another data centre which takes care of reading the data from the storage medium and the sending to the receiver.

Term	Meaning and Description of Cloud Data Term
EXW	Ex Works – The data is available at the consumer's site.
FCA	Free Carrier – The data is preprocessed but not encrypted and ready to be transferred to the provider.
CPT	Carriage Paid To – The data is ready to be transferred to the provider (preprocessed and encrypted) on a direct route. The consumer pays for the data transfer.
CIP	Carriage & Insurance Paid To – The data is ready to be transferred to the provider (preprocessed and encrypted). Moreover, insurance is paid (e.g. insurance for not timely availability of the data)
DAT	Delivered at Terminal – Not used for cloud data.
DAP	Delivered At Place – Not used for cloud data.
DDP	Delivered Duty Paid – The consumer pays for the complete data transfer and is responsible for preprocessing, postprocessing and data encryption.
FAS	Free Alongside Ship – Not used for cloud data.
FOB	Free On Board – The data is ready to be transferred to the provider (preprocessed and encrypted) on a direct route. The provider pays for the data transfer.
CFR	Cost and Freight – The data is ready to be picked up by the provider (preprocessed and encrypted) at an agreed storage data centre. The consumer pays for the storage and the data transfer from the data centre providing the storage.
CIF	Cost, Insurance & Freight – The data is ready to be picked up by the provider (preprocessed and encrypted) at a agreed storage data centre. The consumer pays for the storage and the data transfer from the data centre providing the storage and insurance. (e.g. insurance for not timely availability of the data)

Table 4.2.: Incoterms 2010 applied to cloud data

Term	Data preprocessing	Encryption	Transport to Storage Centre	Transport cost	Transport from Storage Centre	Direct Transport	Decryption	Data postprocessing	Insurance
EXW	Provider	Provider	Provider	Provider	Provider	Provider	Provider	Provider	Consumer
FCA	Consumer	Provider	Provider	Provider	Provider	Provider	Provider	Provider	Consumer
FOB	Consumer	Consumer	-	-	-	Provider	Consumer	Provider	Consumer
CFR	Consumer	Consumer	Consumer	Provider	Provider	-	Consumer	Provider	Consumer
CIF	Consumer	Consumer	Consumer	Provider	Provider	-	Consumer	Provider	Provider
CPT	Consumer	Consumer	-	-	-	Consumer	Consumer	Consumer	Consumer
CIP	Consumer	Consumer	Consumer	Consumer	Consumer	Consumer	Consumer	Consumer	Provider
DDP	Consumer	Consumer	Consumer	Consumer	Consumer	Consumer	Consumer	Consumer	Consumer

Table 4.3.: Cost coverage and responsibility for data transfer

4.4. Rating service providers

When cloud service providers offer a service as a certain cloud computing commodity meeting the definition and all the parameters described in Section 4.1 and 4.2, no statement can be made about the reputation and reliability of the providers. Ratings are crucial not only for consumers before they acquire a cloud computing commodity on the exchange. Ratings are a very important instrument in order to be able to create insurances or derivative financial instruments based on a certain underlying, as they provide information about the risk a cloud provider not being able to deliver the resource acquired on the exchange. The cloud computing commodity exposes SLA parameters but no information about the providers conformity with these parameters in the past. With this rating method buyers and/or consumers of a service are able to take into account the providers' reliability together with the financial health when acquiring a cloud computing commodity over the exchange.

The rating scheme depicted in 4.6 is used by the *cloud provider rating agencies* in a modified version. In the approach presented in this work, providers are rated based on the monitoring information regarding the compliance with the required SLAs in a certain period of time or for a given number of transactions in the past. The proposed rating scheme for cloud providers and/or their services is inherited from the S&P rating scheme. The ratings scale ranges from *AAA* to *D* with the possibility to extend the rating with a plus or minus sign ($+$ for *AAA* to *D* and $-$ for *AAA* to *C*) representing a positive or negative outlook. If a provider is offering more than one services the rating is done for each of the services.

Two major criteria have been identified which are important for the rating of providers and their offered services:

1. Rating the service quality
2. Rating the financial health of the provider

These criteria cover on the one hand the cloud service itself including performance and SLA parameters and on the other hand criteria not related to the cloud service but to the provider.

4.4.1. Rating the service quality

Linked to settlement services, rating agencies monitor the fulfilment of the contracts – the actual delivery of the services. A rating agency collects data about the SLA parameters for a certain cloud computing commodity after its delivery.

Monitoring cloud computing commodities' SLA parameters

For monitoring the SLA parameters of a certain cloud computing commodity, two ways are possible. The service and/or underlying resources can be monitored permanently (*permanent monitoring*) or can be monitored periodically at random times (*statistical monitoring*). In the second case, the monitoring data has to be statistically evaluated in order to achieve appropriate results. There are drawbacks of both, the permanent and the statistical monitoring. An example for such a drawback for the permanent monitoring is the higher network traffic caused by the monitoring data and for the statistical monitoring the potential inaccuracy of the statistically interpolated monitoring data.

- **Permanent monitoring** Permanent monitoring can be realised with monitoring agents for hardware and software components reporting their status. The drawback of permanent monitoring is the huge effort to set up the monitoring system, to change the monitoring system, and the communication overhead. In terms of IaaS, monitoring agents could also report from inside a consumer's virtual machine but as consumers normally have root access to their virtual machines, monitoring data could be manipulated on purpose.
- **Statistical monitoring** In the case of statistical monitoring, the monitoring data is collected during a defined *rating period* for a certain period of time, a number of transactions and/or a number of control samples within a specific time frame or out of a number of transactions.

The data collected by the monitoring has to be processed and transformed into a value representing the rating. In the upcoming part, a rating algorithm is proposed taking into account the current and historical monitoring data.

Rating algorithm

This section presents a basic rating algorithm, which enables to computation of a rating value with the monitoring data as an input. Based on this data collected by either the permanent or the statistical monitoring, a value representing the rating is obtained in the end of each rating period and assigned to this rating period. The values and their meaning is shown in table 4.4. The algorithm takes into account the current situation and historical rating data and is depicted as Algorithm 4.1. In this rating mechanism, three cases for a rating period are defined:

- SLA parameter violated
- SLA parameter met
- SLA parameter met with higher quality than required

Introducing the third case (higher availability than required) enable a stricter limitation of up-ratings. The cases can be reduced to the first two and fed into an adapted algorithm.

For the computation of a possible change of the current provider rating, the five previous rating periods are taken into account. A *rating cycle* represents the calculated values for five consecutive rating periods. The vector rp containing the values of the five rating periods is the input for the computation of the rating change (rch) (Algorithm 4.1). The output of this algorithm is the value for the rating change rch. The possible values for the rating change rch are given in Table 4.5.

As stated in Section 4.7 the numbers representing the ratings are natural numbers from 0 (D) to 9 (AAA). A positive outlook is represented by adding 0.3 to the rating, a negative outlook by decreasing the rating by 0.3. The upcoming rating $r + 1$ is computed by adding $rrch$ (see Table 4.5) to the current rating rounded to a natural number. Algorithm 4.1 is calculating rch based on the following rules:

- All five values of rp are -1: $rch = -2$ (down-rating)
- All five values of rp are 1: $rch = 2$ (up-rating)
- Four values of rp are -1 and the remaining value is 0: $rch = -2$ (down-rating)
- Four values of rp are 1 and the remaining value is 0: $rch = 2$ (up-rating)
- Three values of rp are -1 and the remaining two values are 0: $rch = -1$ (negative outlook)
- Three values of rp are 1 and the remaining two values are 0: $rch = 1$ (positive outlook)
- Two values of rp are -1 and the remaining three values are 0: $rch = -1$ (negative outlook)
- Two values of rp are 1 and the remaining three values are 0: $rch = 1$ (positive outlook)
- In any other case: $rch = 0$ (rating stays)

4.4.2. Rating the financial health of a provider

Despite the technical parameters of the service that can be measured, also information about the financial situation of a provider is of interest. A provider with excellent service performance and no SLA violations could be almost insolvent. In this case, a rating of the delivery of the resources only would not provide sufficient information about the potential risk of non-delivery. For the financial health of a provider, the well established rating system from the financial services sector can be adopted. This work is referring to the Standard & Poor's rating scheme depicted in Table 4.6. To give an impression of the criteria used for ratings in the financial world, the table shows the definitions for rating credit obligations. The table is taken from the document published by Standard & Poor's: *Understanding Standard & Poor's Rating Definitions* [36]. When rating cloud providers, rating agencies check and constantly monitor the financial situation of the providers and take into account major incidents in the service delivery like outages for a certain period of time. In the case of a change in the financial situation of the provider, the rating agency will adjust the rating of the provider accordingly.

The current rating of a provider or a service offered by a provider makes a statement about a provider's conformity to the cloud computing commodity SLA parameters during the

Algorithm 4.1 Computation of the rating change value r

```
Input: $rp = (r_1, r_2, r_3, r_4, r_5), r_i \in (-1, 0, 1), i \in (1, 2, \ldots, 5)$
Output: $rch$
  $rated \leftarrow false$
  if $r_1 = r_2 = r_3 = r_4 = r_5$ then
      $rch \leftarrow 2r_3$
      $rated \leftarrow true$
  else
      if $rated = false$ then
          if $(((r_1 = r_2) \wedge (r_2 = r_3) \wedge (r_3 = r_4) \wedge (r_5 = 0)) \vee ((r_2 = r_3) \wedge (r_3 = r_4) \wedge (r_4 = r_5) \wedge (r_1 = 0)))$ then
              $rch \leftarrow 2r_3$
              $rated \leftarrow true$
          else
              if $(((r_1 = r_2) \wedge (r_2 = r_3) \wedge (r_3 = r_4)) \vee ((r_2 = r_3) \wedge (r_3 = r_4) \wedge (r_4 = r_5)))$ then
                  $rch \leftarrow r_3$
                  $rated \leftarrow true$
              end if
          end if
          if $rated = false$ then
              for $j = 0 \rightarrow 3$ do
                  if $((rp[j] = rp[j+1]) \wedge (rp[j+1] = rp[j+2]) \wedge (rated = false))$ then
                      if $((rp[((j+3) \mod 5)] = 0) \wedge (rp[((j+4) \mod 5)] = 0))$ then
                          $rch \leftarrow rp[j]$
                          $rated \leftarrow true$
                      else
                          $rch \leftarrow 0$
                          $rated \leftarrow true$
                      end if
                  end if
                  $i \leftarrow i + 1$
              end for
              if $rated = false$ then
                  for $j = 0 \rightarrow 4$ do
                      if $((rp[j] = rp[j+1])(rp[j] \neq 0))$ then
                          $rch \leftarrow r + rp[j]$
                      end if
                  end for
              end if
          end if
      end if
  end if
```

previous five rating periods. If a provider faces a major incident, rating agencies react instantly after the incident is recognized and adjust the rating. If necessary, the rating is changed more than one step on the scale. In contrast to the Standard & Poor's rating scheme, the proposed rating scheme for cloud providers includes additionally CC+, CC-, C+, C-, and D+.

A cloud computing commodity listed on an exchange does not provide any information about the potential risk of failure of the service or the provider. Therefore, the concept of subcommodities is introduced in the next section.

4.4.3. Subcommodities

Based on the ratings presented in the previous section, a subcommodity is a commodity offered by a provider with a certain rating. As an example, an IaaS commodity named *IAAS1* would then be traded as the subcommodities *IAAS1-AAA, IAAS1-AA, ..., IAAS1-C, IAAS1-D*. For IAAS1-AAA, *IAAS1* is the commodity name and *AAA* represents the rating extension. A provider can offer a service as different commodities with rating extensions equal to the provider's rating for this service or lower. The commodity itself is treated like an index and represents an indicator for the overall supply and demand of the subcommodities of a cloud computing commodity. The theoretical upper bound of the price for the subcommodities except *C-AAA* (where C represents the commodity name) is the price of the subcommodity with the nearest higher rating. The theoretical lower bound for the subcommodities except *C-D* is represented by the subcommodity with the nearest lower rating. These boundaries are implied by the market mechanism itself since no buyer would pay more for a subcommodity with a lower rating if a subcommodity with a higher rating extension can be obtained for a lower price. Providers, on the other hand, would not get a higher price for a subcommodity with a worse rating if subcommodities with a higher rating have a lower price.

With detailed information about the risk of buying cloud computing commodities over an exchange, insurance mechanisms can be introduced.

4.4.4. Insurance mechanisms

Insurance mechanisms represent a way to insure the provider against non-payment or the consumer against non-delivery of a resource or service. In case of transactions made over a clearing service requiring margin payments, the clearing service or clearing house requests a margin payment before the transaction and acts like an insurance if the provider is not able to deliver the service or the consumer is not able to pay. Insurances need information about the risk of a cloud provider to fail. This information is provided by rating agencies.

The problem is that the risk on the side of the consumer might be much higher than on the side of the provider. The provider's loss is the usage fee whereas the consumer's loss might be greater if parts of the consumer's business are dependent on the service which cannot be delivered.

There are several options for creating an insurance contract for the consumer business risk involving the three roles consumer, provider, insurance. A set of important ones is listed in the following:

- Insurance contract between consumer and insurance (w/o fixed sum excess of provider)
- Insurance contract between consumer, insurance and provider (fixed sum excess of provider)
- Insurance contract between consumer, insurance and provider (fixed sum excess of consumer and provider)

An important aspect for insurance providers is information about risk. This work is able to provide this information by introducing rating agencies making use of rating mechanisms that take into account information about the financial health of a provider and information about the reliability of a cloud service including historical data.

Insurance mechanisms are also required for the transmission of data (large amounts of data). The terms introduced in Section 4.3.3 clarify the responsibilities in the data transfer process when using a cloud service acquired with a transaction over the cloud exchange. Insurance is needed for delayed data transmissions. In some cases, cloud computing resources cannot be used until a certain data set is available on the resources site. This implies that the cloud service reserved is idling until the data is available and therefore generates a loss on the side of the consumer. If the cloud computing commodity includes a term which assigns the responsibility of data transfer to the providers, they have to take the risk and are in charge to compensate the loss of the consumers. The providers might want to use an insurance service to insure the data transfer against possible delays, transmission errors or other issues.

Value	Meaning
-1	Rating period does not meet the required parameters
0	Rating period meets the required parameters
1	Rating period meets the required parameters with a better result than required

Table 4.4.: Description of the values quantifying a rating period

Value	Meaning	$rrch$
-2	Down-rating	-1.0
-1	Negative outlook	-0.3
0	Rating stays	0.0
1	Positive outlook	0.3
2	Up-rating	1.0

Table 4.5.: Description of the rating change values

Rating	Description
AAA	An obligation rated 'AAA' has the highest rating assigned by Standard & Poor's. The obligor's capacity to meet its financial commitment on the obligation is extremely strong.
AA	An obligation rated 'AA' differs from the highest-rated obligations only to a small degree. The obligor's capacity to meet its financial commitment on the obligation is very strong.
A	An obligation rated 'A' is somewhat more susceptible to the adverse effects of changes in circumstances and economic conditions than obligations in higher-rated categories. However, the obligor's capacity to meet its financial commitment on the obligation is still strong.
BBB	An obligation rated 'BBB' exhibits adequate protection parameters. However, adverse economic conditions or changing circumstances are more likely to lead to a weakened capacity of the obligor to meet its financial commitment on the obligation.
BB	An obligation rated 'BB' is less vulnerable to nonpayment than other speculative issues. However, it faces major ongoing uncertainties or exposure to adverse business, financial, or economic conditions, which could lead to the obligor's inadequate capacity to meet its financial commitment on the obligation.
B	An obligation rated 'B' is more vulnerable to non-payment than obligations rated 'BB', but the obligor currently has the capacity to meet its financial commitment on the obligation. Adverse business, financial, or economic conditions will likely impair the obligor's capacity or willingness to meet its financial commitment on the obligation.
CCC	An obligation rated 'CCC' is currently vulnerable to non-payment, and is dependent upon favourable business, financial, and economic conditions for the obligor to meet its financial commitment on the obligation. In the event of adverse business, financial, or economic conditions, the obligor is not likely to have the capacity to meet its financial commitment on the obligation.
CC	An obligation rated 'CC' is currently highly vulnerable to non-payment.
C	A 'C' rating is assigned to obligations that are currently highly vulnerable to non-payment, obligations that have payment arrearages allowed by the terms of the documents, or obligations of an issuer that is the subject of a bankruptcy petition or similar action which have not experienced a payment default. Among others, the 'C' rating may be assigned to subordinated debt, preferred stock or other obligations on which cash payments have been suspended in accordance with the instrument's terms or when preferred stock is the subject of a distressed exchange offer, whereby some or all of the issue is either repurchased for an amount of cash or replaced by other instruments having a total value that is less than par.
D	An obligation rated 'D' is in payment default. The 'D' rating category is used when payments on an obligation are not made on the date due even if the applicable grace period has not expired, unless Standard & Poor's believes that such payments will be made during such grace period. The 'D' rating also will be used upon the filing of a bankruptcy petition or the taking of similar action if payments on an obligation are jeopardized. An obligation's rating is lowered to 'D' upon completion of a distressed exchange offer, whereby some or all of the issue is either repurchased for an amount of cash or replaced by other instruments having a total value that is less than par.

Table 4.6.: Standard & Poor's rating definitions (Source: [36])

Rating	Number	Rating	Number	Rating	Number	Rating	Number
AAA+	9.3	BBB+	6.3	CCC+	3.3	D+	0.3
AAA	9	BBB	6	CCC	3	D	0
AAA-	8.7	BBB-	5.7	CCC-	2.7		
AA+	8.3	BB+	5.3	CC+	2.3		
AA	8	BB	5	CC	2		
AA-	7.7	BB-	4.7	CC-	1.7		
A+	7.3	B+	4.3	C+	1.3		
A	7	B	4	C	1		
A-	6.7	B-	3.7	C-	0.7		

Table 4.7.: Proposed rating scheme for cloud providers

4.4.5. Disadvantages of rating agencies

Rating agencies are a reasonable way to provide information about the reliability and health of providers and their offered cloud services. But what if rating agencies fail in their ratings? As seen in the financial crisis starting in 2007 rating agencies can trigger knock-out effects after publishing certain ratings. Rating agencies can make mistakes in calculating their ratings. They could be dependent on certain institutions and therefore might turn a blind eye on certain issues, which would otherwise relate in a worse rating for those institutions. Rating agencies are subject of several publications that also try to (partially) tackle the issues mentioned. In a "bad-guy scenario" a provider could even pay a rating agency for publishing certain ratings for itself or opponents.

Problems with the methodologies used by rating agencies for rating complex financial instruments and the regulation of rating agencies are discussed in [65] and [50].

The algorithm presented in this work intends to offer a transparent way of a constant rating of cloud providers based on the conformity to the SLA parameters. The question arising is: *Who rates the rating agencies?* In the following section, an alternative or complementary model to rating agencies, for gathering quality information as feedback from the resource users, is shown based on web of trust mechanisms.

4.5. Ratings by consumers

An additional instrument for rating the entities in a framework for the exchange-based trading of cloud computing commodities is trust. With the introduction of a PKI in section 3.2.6, a hierarchical trust infrastructure has been introduced. Each entity trusts the other entities transitively over the exchange – the certificate authority. With an extension of the hierarchical infrastructure with Web Of Trust (WOT) mechanisms, the ratings calculated by rating agencies could be correlated or even extended by the trust to providers expressed by the consumers. This sections introduces a concept for ratings generated from user feedback.

4.5.1. Trust based on key legitimacy

The trust model that forms the basis of the approach presented in this section is the PGP (Pretty Good Privacy) trust model ([11]). This trust model distinguishes three trust levels:

- undefined – no information about the validity of the public key
- marginal – not sure if the public key (certificate) can be trusted
- complete – the public key is valid – the key or certificate can be trusted

For the trust expressed between the entities in a cloud computing commodity market, the two levels *marginal* and *complete* are relevant. *Undefined* is the initial trust status after the issuing of the certificate by the CA. Participants of the web of trust infrastructure can provide the level of their trust for a certificate of another participant and sign this certificate with their own key.

As described in [53], the GnuPG [2] and the PGP trust model make use of the general formula to compute the key legitimacy L.

$$L = \frac{c}{C} + \frac{m}{M}$$

where $c \ldots$ represents the number of complete trust expressions and $m \ldots$ representing the number of marginal trust expressions. C and M is the minimal number of complete C and marginal M trust ratings needed.

For the key legitimacy of a key with the number of marginal and complete trust ratings (m, c) the following holds:

- for $L = 0 \ldots$ the key is not valid
- for $0 < L < 1 \ldots$ the key is marginally valid and
- for $L \geq 1 \ldots$ the key is valid

The parameters for C and M are specified for the WOT infrastructure. Table 4.8 shows the values for C and M used by PGP and GnuPG.

4.5.2. An extended key legitimacy model

There are disadvantages of the PGP/GnuPG trust model described in [53]. These problems can also be seen in P2P infrastructures [79]. In a hybrid model, where a public key infrastructure is combined with web of trust mechanisms, these disadvantages can be avoided to a certain degree. As every participant has to be registered and its status proven by the registration authority, the number of "bad" participants is limited. This section shows a possible extension to the centralised rating approach by a participant centric – and therefore distributed – approach.

The consumer trust model for a cloud computing commodity trading environment is based on the previously presented key legitimacy model. In this case, a participant can express the trust for another participant and its reliability (from a business perspective) by rating it with the three levels:

- no trust: if the participant does not trust another participant('s reliability)
- marginal trust: if a participant is not sure about the trust of the party to rate
- complete trust: if a participant trusts another participant and is convinced about its reliability

[2]GnuPG (GNU Privacy Guard) is a free implementation of PGP using algorithms under no patent.

The *unknown* level is used as initial trust level for participants with new certificates. In this approach, an additional trust level referred to as *no trust* is introduced for a participant who wants to express problems with the reliability of another participant. The rating is expressed by providing a trust level and signing it together with the participant's certificate. The participant attaches the signature to its certificate.

The key legitimacy formula is extended to a *participant legitimacy formula* to include the *no trust* information.

$$P_L = \frac{c}{C} + \frac{m}{M} - \frac{u}{U}$$

with u representing the number or ratings expressing the *no trust* status. $U = C$ holds to treat complete and no trust ratings equitable. The computation of the values for C, M, and U is described in the following paragraphs.

Certificates issued by a CA are limited by an expiration date. This is set equally for all participants. In this approach, every certificate (past, current, future) is issued for the same amount of time. This implies a reset not only of the revocation list but also of the certificate trust values to the unknown status. This is an advantage for new participants as they are able to enter an infrastructure with the same chances to collect reputation as participants with a longer history in this environment. Another advantage is that participants cannot rest on their previously earned laurels.

The duration between issue date and expiration date of the certificates mark a *participant rating cycle*. The computation of the *participant legitimacy formula* requires the values for C, M, and U. These values depend on the number of trades a participant has performed in the latest *participant rating cycle*. The values for C, M(, and U) are managed by the CA or an entity the CA has entrusted to do. For this service the name *Legitimacy Authority (LA)* is used.

An example for computing the values with the assumption the ratings follow a normal distribution could be

$$C = U = \frac{n_{t_i}}{8}$$

$$M = \frac{n_{t_i}}{4}$$

with n_{t_i} representing the number of trades of participant i. Taking into account the number of trades per participant is important as the feedback of users with a lower trading volume should be weighted equally as the feedback of users with higher trading volume. Setting a constant value for all participants would discriminate smaller participants. Any other computation rule is possible as long as the basis includes the number of trades.

In contrast to the key legitimacy, the following holds for the participant legitimacy as the value for the participant legitimacy is not bounded below by 0:

- for $P_L \leq 0$ ... the key is not valid
- for $0 < P_L < 1$ ... the key is marginally valid and

- for $P_L \geq 1 \ldots$ the key is valid

In the case somebody would like to compute the participant legitimacy of another participant a request is sent to the Legitimacy Authority (LA). The LA requests the certificate and computes the participant legitimacy with the help of the two (three) values managed by the LA. Participants of the cloud computing commodity trading environment cannot only rate providers. The term participant used in this subsection holds for all members including providers, consumers, entities offering clearing and settlement services.

The exchange and other participants can set actions for certain values of P_L of the infrastructure members. The reasons for a low P_L value are investigated and in the case of unacceptable behaviour or violations of the rules the participant is banned from the cloud computing commodity trading environment. Also rating agencies can include the participant legitimacy in the rating.

A direct exclusion e.g. of a provider with a bad trust rating (low P_L value) by a buyer is not possible because this would interfere with the exchange model and the anonymity of the market participants.

Trust Value	PGP	GnuPG
C	1	1
M	2	3

Table 4.8.: Values for C M in the GnuPG and PGP trust model

Chapter 5

Implementation of the Framework for Exchange-Based Trading of Cloud Computing Commodities

In this chapter, the implementation of a framework for cloud computing commodities is shown based on the model presented in Chapter 3.2.3. Simulations based on the implementation have been performed with an environment containing exchange, clearing, settlement, rating agency, and market participants (buyers and sellers). The results of the simulation are presented showing diagrams of provider ratings and resulting graphs of the cloud computing commodity spot prices.

5.1. Implementation of the core elements

The implementation of the core parts of the model (see Section 3.2) is presented. The design and implementation details of the following classes are shown.

- exchange
- market participants
- clearing
- settlement
- rating agency

In this implementation the basic features of an exchange are taken from the *webcurvesim* project [28], which enables the simulation of a stock exchange. It has been chosen as it is available as open source and makes use of a very modular design. The implementation of the additional functionality, which is required by the model presented in this work, is shown as follows.

5.1.1. Exchange

The core part of the trading is represented by the exchange – the marketplace for the commodities. The exchange class is depicted in the UML diagram (Figure C.2. Compared to the original exchange class of the webcurvesim project (Figure C.3), the following functions have been added in order to fulfil the requirements for the trading of cloud computing commodities.

- `setListingChanges` to set the listeners to enable the tracking of new orders and trades
- regAtListingChanges to register the exchange `Listingchanges` class
- `register` informs the exchange about a new trader and add the new trader to the list of registered traders. This method takes an instance of the class `MarketParticipant` as input.
- `deregister` removes a certain trader from the list of registered traders. Input is an instance of `MarketParticipant`
- `notifyTraders` to inform a trader about a successful order (increase the number of trades)
- `getTraders` get a list of the registered traders
- `getTrades` returns a list of successful trades
- Getters and Setters for clearing and settlement, and commodities (to set or return the commodities listed on the exchange)

The clearing and settlement is represented in the implementation by the class `Clearing` (Figure C.4) and by the class `Settlement` (Figure C.5).

5.1.2. The market participants

Both, buyers and sellers of resources are represented by the `MarketParticipant` class (Appendix, Figure C.1). The following functions are provided by the `MarketParticipant` class:

- `increaseNumTrades` if an order of this trader is successful this method increases the variable holding the number of trades

- `loseMoney` if a trade is cleared the trader (buyer) has to pay the money stated on the trade. The money on the trader's account decreases by the total transaction amount.
- `earnMoney` a trader (seller) gets the money stated on a trader during the process of clearing. The money on the trader's account increases by the total transaction amount.
- Getters and Setters for the number of trades of the participant, the money, and the parameters necessary for the secure clearing and settlement, namely $a, a', b, c, d, cr_1, cr_2,$
- `initKeys` to initially create the private and public key of the participant for the simulation
- `checkSignature` gets an instance of the `SignedMessage` class as an input and checks if the containing signature is valid. This method returns `true` or `false`
- `genSignature` generates an instance of the `SignedMessage` class with a given String as an input
- `checkSuccess` checks if the signature contained in a `SuccessMessage` instance is valid (return value `boolean`)
- `checkCredentials` checks the validity of a given instance of the `Credentials2` class (in case the trader is a buyer)
- `genCredentials` generates an instance of the `Credentials1` class with given credentials and a trade ID (in case the trader is a seller)
- `getPublicKey` returns the public key of the trader

The following methods for the basic functionality enabling the starting and stopping of the automated trading of a marketparticipant and getter and setter functions are used from the *webcurve* implementation.

- `run`
- `start` start the trader thread of this instance of the `MarketParticipant` class
- `pause` pause the trader thread of this instance of the `MarketParticipant` class
- Functions to set the initial values for the trading of an instance of a `MarketParticipant`
  - `getStock` returns the name of the item (stock, commodity) the `MarketParticipant` is trading at the moment

- `setStock` sets the name of the item the trader is trading
- `getParticipantName` returns the string identifier of an instance of the `MarketParticipant` class
- The methods (getters and setters) `getBasePrice`, `setBasePrice`, `getPriceVariant`, `setPriceVariant`, `getSd`, `setSd`, `getSdFactor`, `setSdFactor`, `getTradingLength`, `setTradingLength`, `getTradingMinInterval`, `setTradingMinInterval`, `getTradingMaxInterval`, `setTradingMaxInterval`, `getMinQuantity`, `setMinQuantity`, `getMaxQuantity`, `setMaxQuantity`, `getPriceStep`, and `setPriceStep` are used to get and set the basic value parameters for the prices and quantities the simulated trader should operate together with parameters to vary the basic value parameters.

5.1.3. Datastructure for traders

The datastructure `KeyMp` is holding an instance of the class `MarketParticipant` and a corresponding key value (`String`). It implements two functions for retrieving these elements.

- `getKey` returns the key value
- `getMP` returns an instance of the class `MarketParticipant`

5.1.4. Clearing

The `Clearing` class provides the functionality to keep a basic account for the money exchanged between buyer and seller over the clearing (functions removeMoney, addMoney). During the creation of an instance of the `Clearing` class a keypair is computed. Together with this keypair and the remaining functions of the `Clearing` class the security of the clearing and settlement process (Section 3.2.6) is ensured. The following functions necessary for the secure clearing and settlement are provided by the `Clearing` class:

- Getter and Setter functions for a, a', b, c, d
- `checkSignature` to validate the digital signature included in an instance of the `SignedMessage` class. As an input this method requires an instance of the class `SignedMessage`.
- `genSignature` to create an instance of the `SignedMessage` class including the tradeID and its digital signature. `genSignature` gets the tradeID of type `String` as input.

- `genSuccess` to create an instance of the `SuccessMessage` class including a digital signature of the success message. This method requires the tradeID and a success message – both of type `String`.
- `getPublicKey` to retrieve the public key of the clearing

5.1.5. Settlement

The `Settlement` class is responsible for the encrypted exchange of the credentials needed to access the cloud resource traded between seller and buyer. The requirement to decrypt, encrypt and store the credentials is met by this class utilising public key encryption and a vector for storing the encrypted credentials.

- Getters and Setters for $c, d, cr1, cr2$
- `storeCredentials` to store the (with the settlement's public key) encrypted credentials in the format given by the class `Credentials1` (described in the following) and store it in suitable data structure (here: java Vector)
- `genCredentials` to decrypt the stored credentials received from the seller with the settlements private key and create an instance of `Credentials2` encrypted with the buyers public key
- `checkSignature` to check the validity of the signature contained in an instance of the `SignedMessage` class
- `genSignature` to generate an instance of the `SignedMessage` class containing a digital signature
- `getPublicKey` to retrieve the settlement's public key

The UML sequence diagram for the secure clearing and settlement is depicted in Figure C.13.

5.2. Secure information interchange

The secure clearing and settlement process presented in Section 3.2.6 requires the ability to exchange data in a secure way. Public key cryptosystems [70] are used for the implementation as they allow the secure key exchange. For the interchange of information and ensuring the security of the messages sent between the buyer, the seller, the clearing, and the settlement, additional classes are introduced for the secure information interchange and the PKI. The classes necessary for this functionality are described in detail below.

- Signed message
- Success message
- Credentials
- Encryption
- Datastructure for traders

5.2.1. Signed message

Instances of the class `SignedMessage` (for the UML diagram see Figure C.7) are used for sending requests or confirmation messages. For sending a, a', b, c, or $setc$ between the participants in the secure clearing and settlement process, instances of the class `SignedMessage` are generated.

- `getPk` returns the public key of the signer/sender of the message. This key has to be validated by either comparing it with the sender's public key or checking the key's fingerprint.
- `setPk` sets the signer's public key
- `getSignature` returns the digital signature of the given string (tradeID)
- `getTradeid` returns the tradeID

5.2.2. Success message

The `SuccessMessage` class is used to represent the message sent by the clearing to the buyer and the seller in order to inform them about the successful execution of the transaction. The success message is further used by the buyer to send a request for the delivery of the credentials to the settlement.

- `getSignature` returns the digital signature of a message composed by the tradeID and the success message `String`
- `getStringdata` returns the success message `String`
- `gettradeid` returns the tradeID
- Getter and setter method for the public key of the signer of the success message

5.2.3. Credentials

For the secure interchange of the credentials over the settlement as intermediate entity, two datastructures have been defined represented by the classes `Credentials1` and `Credentials2`. `Credentials1` is responsible for holding the credentials (and their signature) encrypted with the public key of the settlement. The encrypted credentials as an instance of `Credentials1` are stored by the settlement until requested by the respective buyer. After a request, the credentials are decrypted by the settlement with its private key and an instance of `Credentials2` is generated containing the credentials encrypted with the buyer's public key.

The classes `Credentials1` and `Credentials2` contain methods for retrieving the tradeID, the signature of the tradeID and the credentials and a function `getEncrypted_credentials` which is responsible to return the encrypted credentials in order to decrypt it. The UML diagrams of the classes are shown in Figure C.8 and C.9.

5.2.4. Encryption

For the encryption and decryption of the credentials, the class `EncryptDecrypt` has been implemented (UML depicted in Figure C.10).

- `encrypt` encrypts a given `String` with a given public key and returns the encrypted message
- `decrypt` decrypts a given encrypted message with a given private key and returns the plain text message

5.2.5. Rating agency

For the simulation of a rating agency for cloud providers, the approach presented in Section 4.4 has been implemented. As input values for the rating agency, historical data (job waiting time from a grid infrastructure) and randomly generated data has been used.

As a first proof of concept, the algorithm to compute the rating changes has been implemented in Mathematica. The input for the simulation consists of waiting times in the case of real input data. The data consists of job waiting times with one rating period was assumed to be 2000 jobs. In the simulation, the first step is to calculate the rating period values $(-1, 0, 1)$ from the given waiting times (see Appendix B.3.1). This results in a vector containing the rating period values for the entire simulation. Rating period values represent information about the conformity to a certain SLA in a rating period (-1 not met, 0 met, 1 better) .The values act as the basis for the computation of the rating change values (Algorithm 4.1). Rating change values describe the change of the rating in every step. The algorithm has been implemented together with the calculation of the current provider ratings (source code in Appendix B.3.1).

The rating simulation has also been implemented in Java to enable the integration in the trading simulation framework written in Java. In the Appendix selected parts of the Java source code of the rating simulation can be found in Section B.3.2.

The implementation of the rating agency takes into account a basic service level agreement parameter based on waiting times. In Figure C.6 the UML class diagram of the implemented `WaitSLA` class can be seen.

As the ratings are simulated with existing values the constructor of the class gets a filename as an input value together with the SLA parameters and the initial rating at the time of creation of the class instance. In the simulation, the ratings have been calculated in advance based on grid monitoring data (job waiting times).

The class `WaitSLA` offers the following functions:

- `getRatingChanges` returns a vector with the integer values containing information about the rating periods; one of the values $(-1, 0, 1)$ for each rating period
- `getRatings` returns a vector containing one of the double values $(-2, -1, 0, 1, 2)$ for each rating period
- `getRPTrans` returns the number of transactions equal for every rating period
- `quantifyRPsSeq` computes the rating vector
- `ratingchangeSeq` computes the rating change vector
- `saveToFile` saves the ratings to a file

Chapter 6

Simulation and Evaluation

This chapter starts with a description of the simulation environment. The two core parts of this environment are on the one hand the trading, clearing, and settlement simulation and on the other hand the simulation of the rating agency. After presenting the results of the rating agency simulation, the results of the simulation of the complete framework are presented together with their evaluation.

6.1. Simulation environment

Based on the implementation described previously, the cloud computing commodities exchange has been simulated with a setup of one cloud computing commodity and its subcommodities. The marketparticipants are represented by 20 providers with their rating based on real data from job execution times from the EGI grid infrastructure, and 500 traders who act as buyers or sellers randomly in order to gain a more liquid market.

In order to be able to simulate the trading of subcommodities (Section 4.4.3), provider ratings have to be simulated. The simulation of the provider ratings has been implemented in Mathematica and Java as stated in Section 5.2.5 (see Appendix for selected parts of the source code: Section B.3.1 for the Mathematica source and Section B.3.2 for the source code in Java). Following the description given in Section 4.4, the provider ratings have been computed based on both, randomly generated and real data. For the random data a normal distribution has been chosen with $(\mu, \sigma) = (0.11, 0.5)$. The mean has been chosen > 0 because it is assumed that providers try to improve their rating in order to be able to offer the better and more lucrative subcommodities. In the case of the random data, 170 rating period values have been calculated. The numbers have been rounded to their nearest natural number. In addition, all values ≥ 1.5 have been set to 1, and all values ≤ -1.5 have been set to -1. The distribution of the three values can be seen in Figure 6.1. It can be observed, that the value 0 (SLA met) is the highest followed by 1 (better) and -1 (SLA not met). The meaning of the three values is described in Table 4.4. In both, the random and the real simulation the initial rating was set to 5 which equals *BB*. This initial rating has been chosen as the number of rating values represented by natural numbers divided by two in order to start from a value between the best and the worst rating to be able to observe movements of the ratings in both directions, up and down.

After applying Algorithm 4.1 which computes the potential changes in provider ratings, the resulting distribution is depicted in Figure 6.2. The value 0 representing a stay of the rating is significantly higher in the case of the normal distributed input data. Caused by the mean ≥ 0, the number of positive rating change values is higher than the negative ones. The description of of the different resulting values is given in Table 4.5.

In the case of randomly generated rating period data, the rating graph over the 170 rating periods shows an expected picture with the majority of rating change values holding at 0 representing no change. The rating graph corresponds to the distribution of the rating change values shown in Figure 6.2. Most of the time, the rating stays and the graph shows an overall positive trend. The graph is shown in Figure 6.3.

The real data used for this simulation is representing the waiting times for running jobs on the EGI (European Grid Initiative - [5]) infrastructure. The data consists of 300000 data sets from late 2009 and has been obtained from grid observatory ([7]).

The fictive cloud provider in this simulation offers a PaaS to run jobs. This service is provided with the the following service level agreement KPIs. The numbers have been chosen after analysing and normalising the data set and calculating values that deliver significant results.

- The average waiting time for a job in one rating period has to be below 10080s
- For 15% of the jobs in a rating period of 2000 jobs the waiting time is allowed to be up to 14400s

For the quantification of a rating period consisting of 2000 measured waiting times, the rules given in Table 6.1 have been applied.

The resulting distribution for the waiting time of the considered 300000 job requests is depicted in Figure 6.4.

The distribution of the resulting rating change values after the execution of Algorithm 4.1 is shown in Figure 6.5. It can be observed that the number of positive rating periods is higher than the number of both, the rating periods meeting the required parameters and the rating periods with a better result than demanded whereas the number of up-ratings is higher than the number of stay-ratings. A spike in the values changing the provider rating is noticeable in the negative outlook represented by -1.

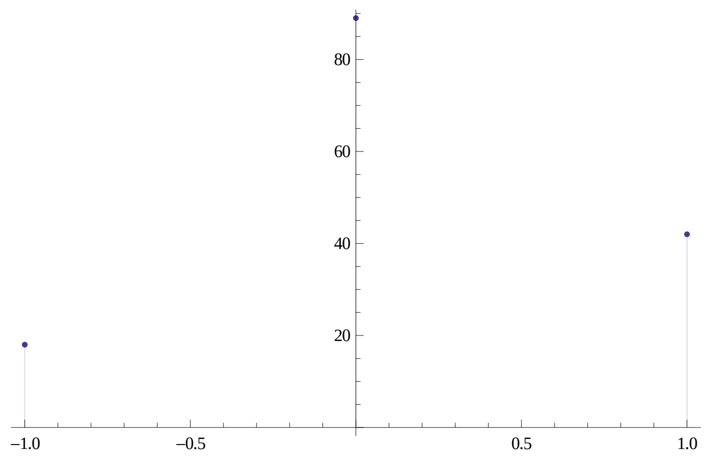


Figure 6.1.: Distribution of the values $(-1, 0, 1)$ (random data)

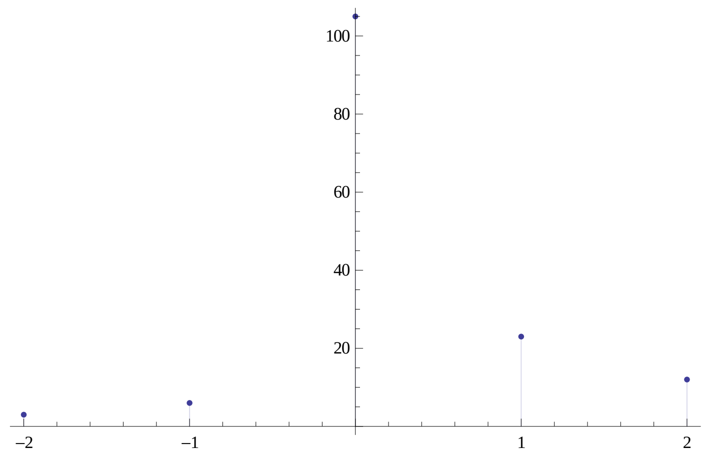


Figure 6.2.: Distribution of the values $(-2, -1, 0, 1, 2)$ for random input data

Value	Rules
-1	The average waiting time to run a job is > 10080 or more than 15% of the waiting times are > 14400
0	The average waiting time in the rating period is between 5143s and 10080s
1	The average waiting time in the rating period is better than the required 5143s

Table 6.1.: Description of the rules for obtaining the quantifying values of a rating period

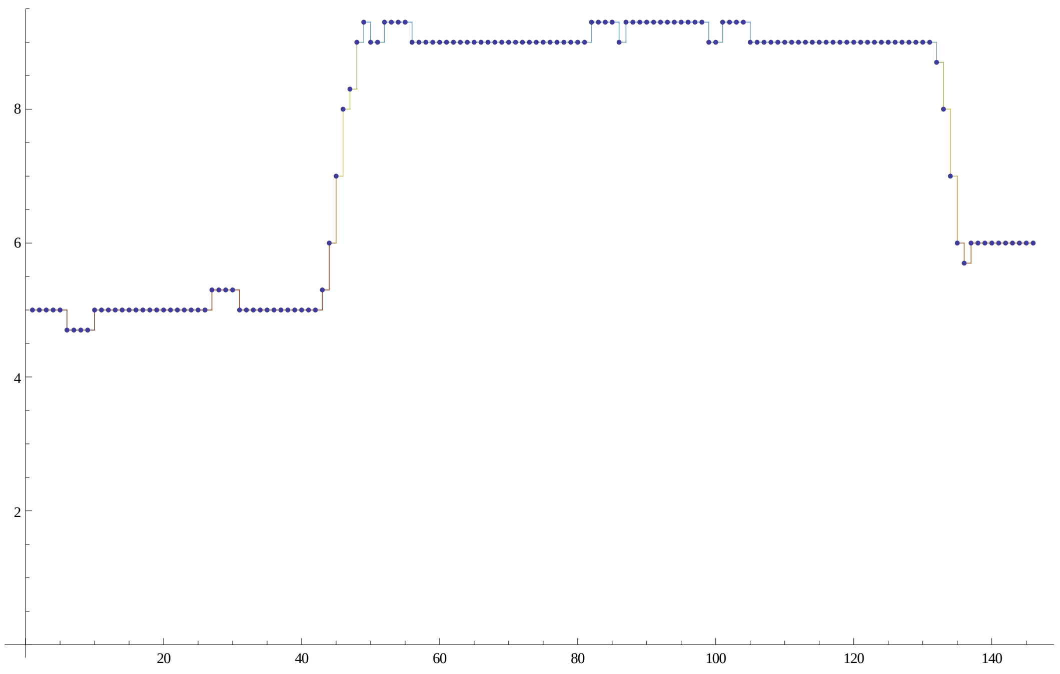


Figure 6.3.: Graph showing the provider ratings of a single provider based on random input data

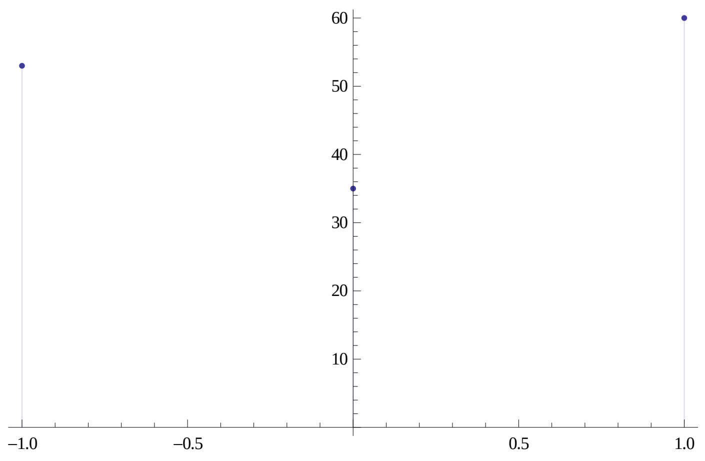


Figure 6.4.: Distribution of the values $(-1, 0, 1)$ for real input data

Starting with an initial rating of *BB* represented by 5, the graph reflecting the provider ratings during the 170 rating periods is presented as Figure 6.6. It can be seen that the number of changes in the rating is much higher than in the random case.

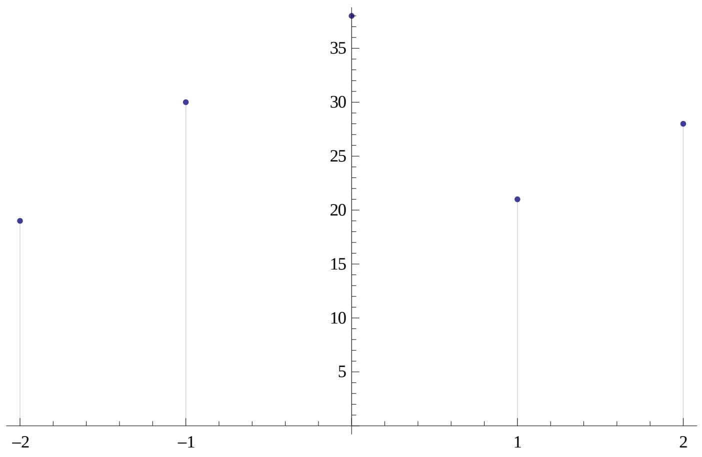


Figure 6.5.: Distribution of the values $(-2, -1, 0, 1, 2)$ for real input data

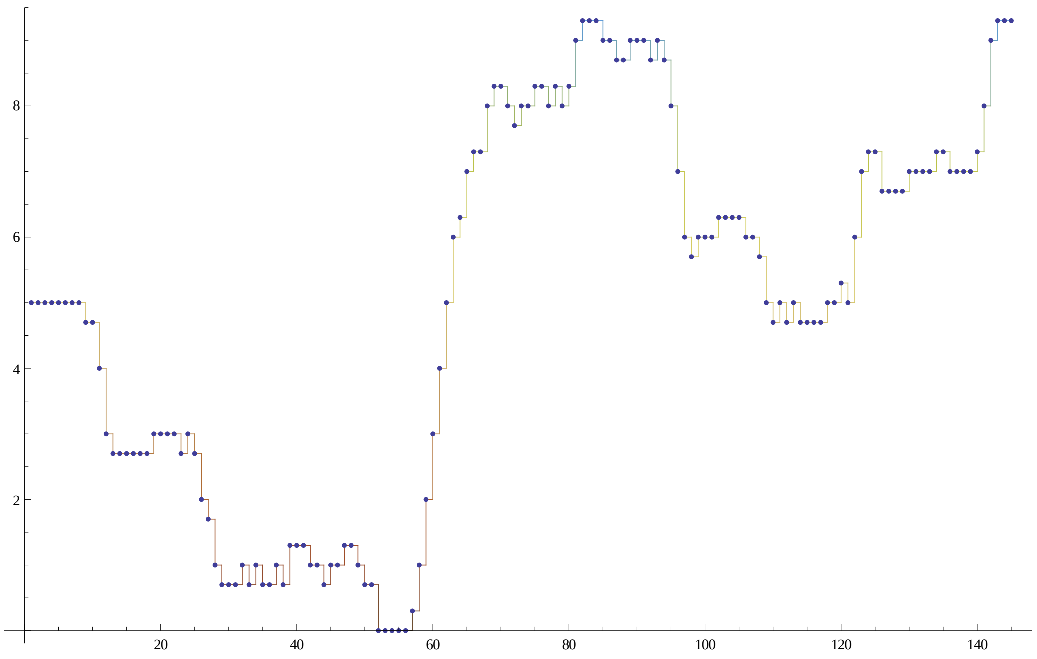


Figure 6.6.: Graph showing the provider ratings of a single provider based on real input data

6.2. Simulation results and discussion

For the simulation, a cloud computing commodity *C1* has been chosen. This commodity is available for trading on the exchange with all of its subcommodities from *AAA* to *D*. The initial prices for the subcommodities (t_0) in the simulation are depicted in Table 6.2.

Simulation setup: In the following, the simulation setup is described. The number of traders, traded commodities, and the numbers relevant for the simulation are described.

- For 15 traders, real data according the example presented for the rating in the previous section has been used. For each of the 15 traders, 300000 data sets with job waiting times act as the basis for their rating. In total, 4500000 data sets have been used for the simulation.
- The other traders are committed to a certain subcommodity, which they constantly trade. The trading agents have a minimum intelligence, which allows them to decide not to buy a subcommodity with a lower rating for a higher price. If the price of two subcommodities is close, the traders place their orders with a predefined difference from the expected intersection of the two price curves or the traders switch to the other cloud computing commodity (from the lower to the higher rated subcommodity). In this simulation, traders do not have a limit of the price they are willing to pay.
- The time steps in the simulations are one second. Every second, the price of all subcommodities is determined and saved to a file. Every four seconds, traders are randomly stopped or started if stopped previously. This is done to artificially generate small spikes in the charts and a change in supply and demand. One day in the simulation lasts 25 seconds. All orders placed by the market participants are limit orders valid for one day. If an order cannot be matched until the market close, the order is deleted from the order book.
- Each trader is equipped with a capital of 1000000 of a not specified currency.

Six simulations have been selected to be shown in this section. The selected simulations represent different market situations that are reflected in the resulting charts.

Simulation 1: This simulation covers the period of 6 simulated days of trading (150 seconds) with a limited number of traders (15 traders) and no starting and stopping of traders. In Figure 6.7 it can be seen that there is almost no change in the price of the subcommodities. This is the result of a low liquidity and a very well balanced supply and demand in this case.

Simulation 2: In this simulation, 24.5 simulated days (612.5 seconds) of trading with 520 traders and periodic starting and stopping of traders is shown. The supply and demand is random

Price	Subcommodity
24.00	C1-AAA
22.70	C1-AA
21.40	C1-A
20.10	C1-BBB
18.80	C1-BB
17.50	C1-B
16.20	C1-CCC
14.40	C1-CC
13.60	C1-C
12.30	C1-D

Table 6.2.: Subcommodity prices

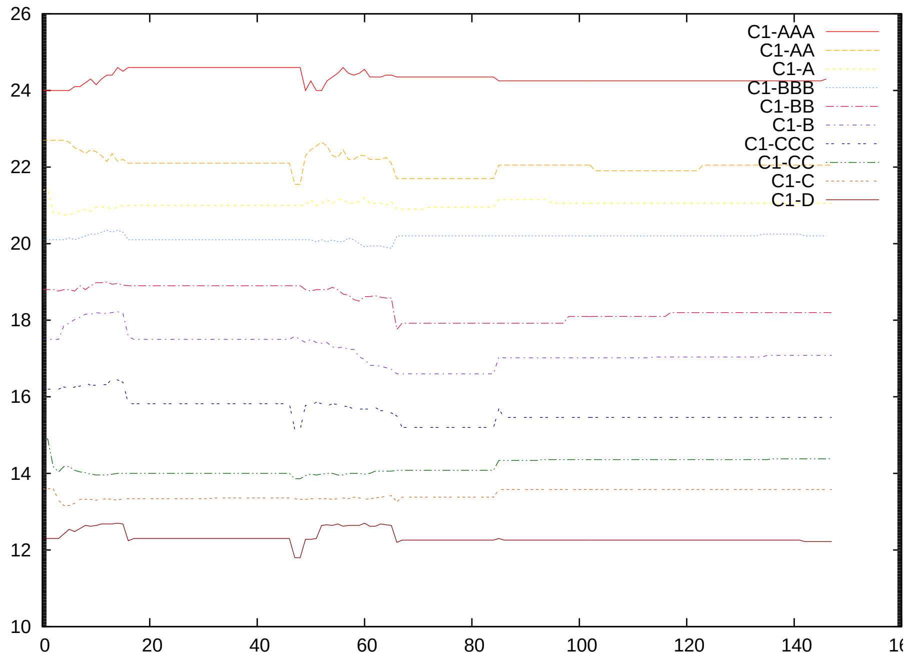


Figure 6.7.: Chart of the price of the subcommodities of C1 - Simulation 1

but balanced. In this setup, the change of the price of the subcommodities is mostly caused by the randomly starting and stopping of the traders (Figure 6.8). The demand of the highest rated subcommodity is slightly higher than the demand for the remaining subcommodities, resulting in a higher price for the highest one.

Simulation 3: The third simulation shows 12 days of simulated trading (300 seconds) with 520 traders and periodic starting and stopping of traders. The demand for the subcommodity C1-AAA has been increased, which results in a continuously increasing price (Figure 6.9).

Simulation 4: This simulation shows 40 days of simulated trading (1000 seconds). 1515 simulated traders are involved, which are started and stopped periodically. The subcommodities C1-AAA to C1-BBB have been traded with balanced supply and demand, whereas the offer is higher than the demand for the lower rated subcommodities (C1-BB to C1-D, in credit risk rating the speculative, non investmentgrade products have such ratings). Figure 6.10 depicts the chart of this simulation.

Simulation 5: Simulation 5 shows again 40 days of simulated trading (1000 seconds) with 1515 traders and a periodically starting and stopping of these traders. The offer for all of the subcommodities has been chosen slightly higher in this simulation, which results in decreasing prices (Figure 6.11).

Simulation 6: Simulation 6 has been run for 40 days of simulated trading (1000 seconds) and 1515 traders, which can be started and stopped periodically. The demand of the C1-AAA and C1-AA subcommodities has been increased in contrast to the remaining subcommodities. It can be observed that the price of the two highest rated subcommodities increases whereas the price of the other subcommodities goes down (Figure 6.12).

The simulations run show the possibility of regulating supply and demand of cloud computing commodities by dynamic prices determined by an exchange-based framework. In the case of a balanced supply and demand (Simulation 1 (Figure 6.7) and 2 (Figure 6.7)), it can

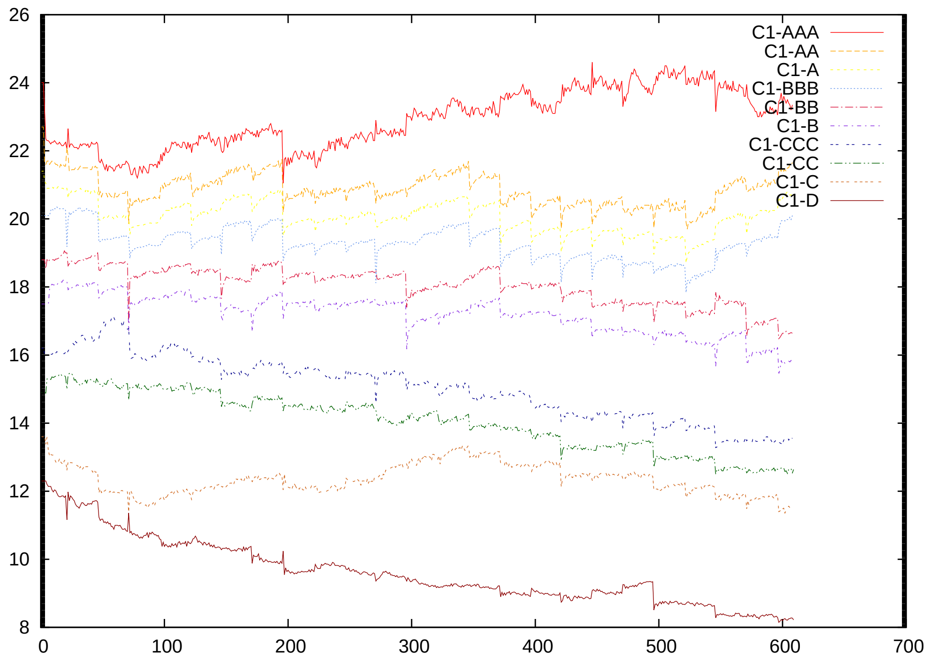


Figure 6.8.: Chart of the price of the subcommodities of C1 - Simulation 2

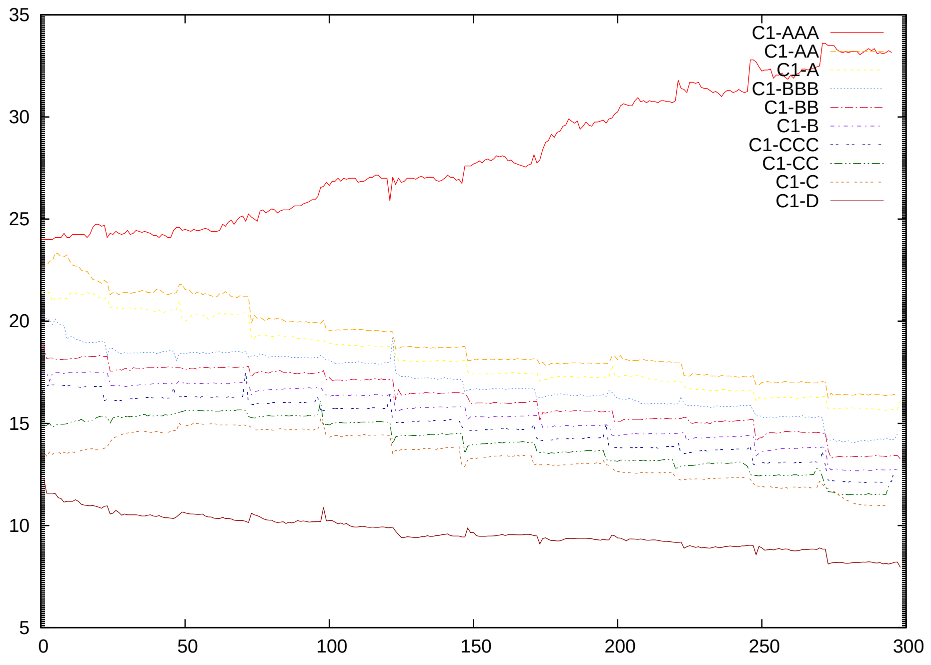


Figure 6.9.: Chart of the price of the subcommodities of C1 - Simulation 3

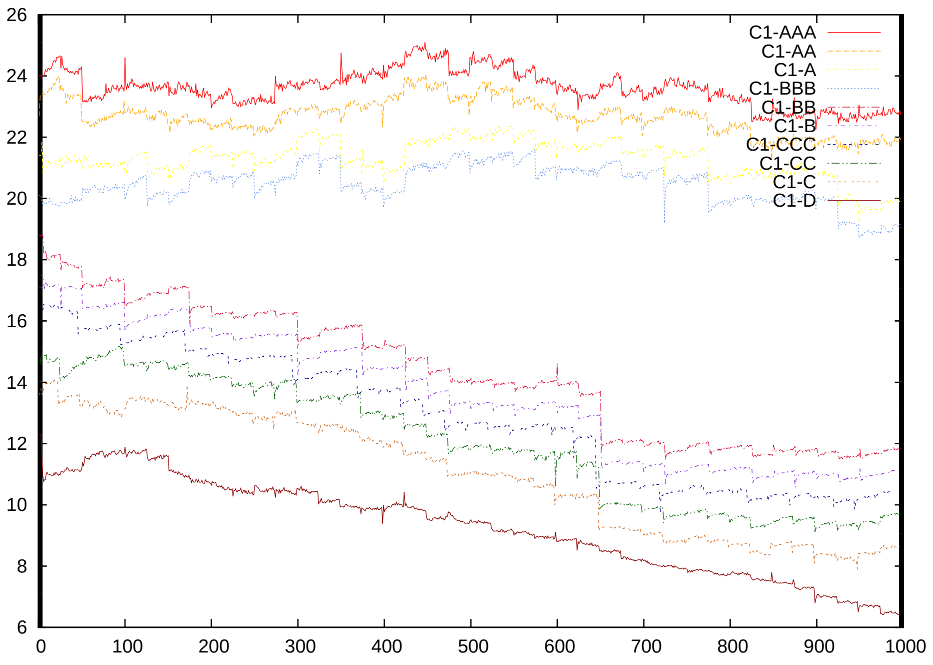


Figure 6.10.: Chart of the price of the subcommodities of C1 - Simulation 4

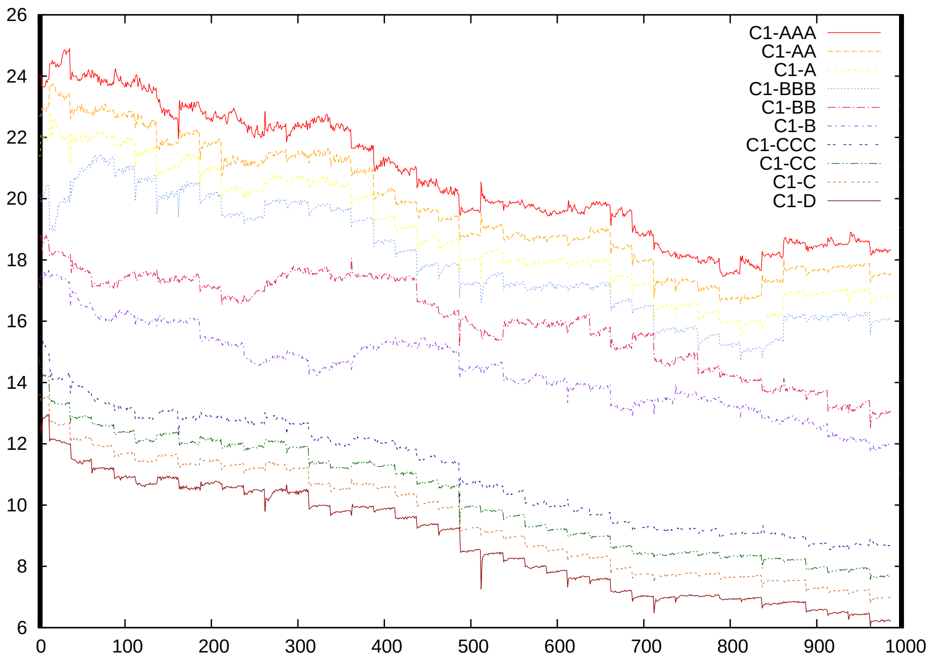


Figure 6.11.: Chart of the price of the subcommodities of C1 - Simulation 5

be observed that prices stay in a certain range not varying too much. The graphs for the price of the subcommodities stay constant within a certain range. In Simulation 2, an additional factor of randomness is introduced, as traders of a certain subcommodity are started and stopped periodically. If the supply and demand of certain subcommodities is artificially increased or decreased, the result is higher fluctuations in the prices. Simulation 3 with continuously increasing demand of one subcommodity shows an increasing price for this subcommodity over time, whereas the price for the other subcommodities stays as the supply and demand for these is balanced. The chart shows the price of subcommodity C1-AAA increasing constantly. The other subcommodities stay similar to the previous simulations. Simulation 4 marks a balanced supply and demand for the first four subcommodities, whereas the demand of the remaining subcommodities is decreased. The chart shows relatively constant prices for the four highest rated subcommodities and decreasing prices for the lower rated ones. Simulation 5 shows a higher offer in each of the subcommodities resulting in decreasing prices for all the subcommodities. The chart shows decreasing prices for all the subcommodities. In Simulation 6, the demand of the two highest rated subcommodities has been significantly increased, whereas the demand for the other subcommodities has been slightly decreased. In this simulation, the buyers of the two highest rated subcommodities would be willing to pay a price, which more than doubles over the simulation time. The chart shows the two subcommodity prices for C1-AAA and C1-AA increasing continuously, whereas the prices for the other subcommodities decrease.

The results show that the approach for exchange-based trading of cloud computing commodities is able to regulate supply and demand as the prices react very quickly on a changing market situation. Applied to the real world, higher prices would lead to a decreasing demand because every buyer has a maximum price for a certain cloud computing commodity. If the price decreases to a certain level, providers will offer less resources, as their revenue decreases accordingly. In both cases, the prices would start to move in the opposite direction. Simulation 1 shows that a certain liquidity (number of marketparticipants and therefore orders and trades) is necessary to enable an exchange for cloud computing commodities. The framework for exchange-based trading of cloud computing commodities is able to regulate supply and demand of standardised and comparable cloud computing commodities in the case of a liquid market.

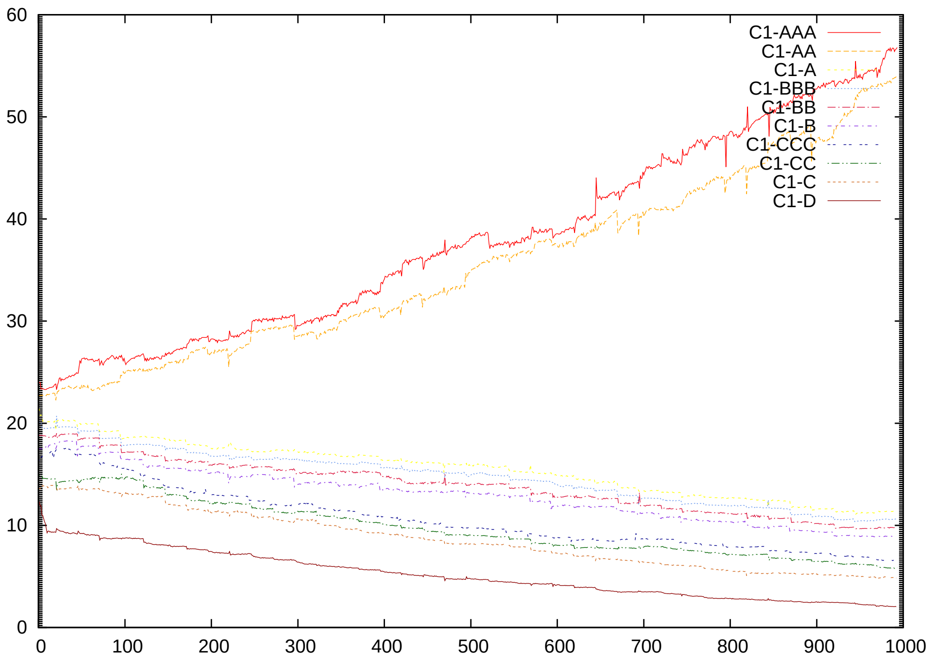


Figure 6.12.: Chart of the price of the subcommodities of C1 - Simulation 6

Chapter 7

Conclusions and Outlook

This chapter provides an overview of the achievements of the thesis and highlights additional topics in different research fields, which need to be investigated by future work.

7.1. Summary of achievements

An approach for using exchange mechanisms to regulate the supply and demand of cloud computing resources is presented. The increasing use of cloud computing implies a higher demand in the future. The strategy proposed in this work is the standardisation of goods – in this case cloud computing resources – and an introduction of a marketplace for these standardised goods and their trading on the newly created cloud computing commodities exchange. The proposed approach has never been applied to cloud resources before.

To be able to trade cloud computing resources, they have to be commoditised. The commoditisation also enables the comparison of different cloud service offers. For achieving this goal, cloud computing commodities representing classes of cloud computing services have to be introduced and defined. Cloud computing commodities represent the tradable goods and are described by a set of parameters including resource or service parameters, performance, service level, and additional parameters (e.g. location and legal constraints). Within a cloud computing commodity, the values of the parameters have to meet certain criteria. The cloud services are comparable within a class of cloud services. Cloud computing commodities not only allow transparent and comparable offers. With those, the establishment of a contract between provider and consumer includes service level parameters with the implicitly agreed cloud computing commodity parameters by placing an order.

A certain level of quality can be ensured together with providing the market participants information about the potential risk of failure of a cloud service acquired on the exchange. The framework provides the capability to rate providers based on their conformity to SLA parameters and their financial health. With the introduction of subcommodities representing a certain cloud computing commodity in combination with a rating, the value of a cloud computing commodity can be defined.

Based on the information about the risk, insurance companies are able to introduce insurance products. In addition, derivative financial instruments can be built on the underlyings.

With the present framework it is possible to regulate supply and demand within the cloud market to allow more flexible trading mechanisms for cloud computing commodities and to provide the functionality identified in Chapter 1.

7.2. Future work

This work provides a basis for future work in computer science, financial mathematics, economics, and law.

In computer science, standardised ways of managing and accessing cloud services, benchmarking, and monitoring are topics which need further investigation to meet the requirements of the framework for the exchange-based trading of cloud computing commodities. The first standards for cloud infrastructure management have been published, but there is a lot of research still required to establish standards for managing IaaS, PaaS, and SaaS together with interface standards for all cloud layers.

Additionally, there are no benchmarks, which are able to deliver precise results for virtualised infrastructures or cloud platforms. SaaS needs specialised benchmarks to be developed.

Monitoring of cloud services is another area for future research as comparable monitoring data has to be collected from different cloud providers and analysed for potential SLA violations.

In the area of financial mathematics, new financial products can be designed based on existing ones inspired by electricity markets and commodities and more complex ones to realise insurance mechanisms that could be introduced as tradable financial instruments.

In economics, the influence of tradable cloud computing commodities to existing businesses, markets, and the change of a provider driven market to an supply and demand based one could be examined. In a multi-national cloud exchange setup, various different legislations concerning topics from data privacy to billing and contract establishment laws apply. The harmonisation of these will be a big challenge that have to be mastered in the future.

Further economic aspects

The framework for exchange based trading of cloud computing commodities presented in this work can act as the basis for a variety of financial instruments. This work does not introduce derivatives on the underlying cloud computing commodities or other complex financial instruments, but it establishes a basis for their creation and to open the reader's mind to further economic considerations and work. The thesis would also like to act as a foundation for a broad field of economic models and business models for cloud computing.

- Indexes

  For commodities, indexes have been created (e.g. GSCI (Goldman Sachs Commodity Index) [1] or Thomson Reuters/CRB (Commodity Research Bureau) Index) [2]. Indexes are a measure for the commodity prices and act as a basis for various different investment strategies.

[1]GSCI (Goldman Sachs Commodity Index), http://www.goldmansachs.com/what-we-do/securities/products-and-business-groups/products/gsci/

[2]Thomson Reuters/CRB (Commodity Research Bureau) Index, http://thomsonreuters.com/commodity-indices/

In the case of a cloud computing commodities exchange, indices can be built upon a certain group of commodities. An example could be to establish an index for IaaS commodities (with indexes for storage or indexes for compute commodities) or an index for certain PaaS commodities. Indexes in a cloud computing commodities market would be a good barometer of the current situation and the price trends for these commodities in the future. Cloud computing commodity indices might be able to act as an indicator for certain good or bad situations in different business sectors.

- Carbon certificates and baskets

  A large proportion of electricity is generated by burning fossil combustible material (gas, oil, coal) inducing a correlation between electricity prices and the price of gas, oil, coal, but also carbon certificates and weather. The burning of these materials generates a vast amount of carbon dioxide. For this, the providers of the electricity have to hold an equivalent in carbon certificates. If providers want to extend their power generation capacity they have to acquire more carbon certificates with an effect on the price (unless they work with carbon certificate hedging strategies). In terms of weather, the electricity price normally goes up when the temperatures go down because a lot of heating is done with electrical energy. There is also a correlation between the weather and the generation of energy taking into account hydro-electrical power stations. When the amount of water in rivers decrease caused by aridities the price for electricity rises.

  Speaking of cloud computing commodities which are completely dependent on the availability of electrical power, the price for cloud computing commodities is directly proportional to the price of electricity.

  For commodities of all kinds, different strategies to decrease the risk exist based on futures, options, and combinations of different kinds of futures and options ([41]). Adapted electricity hedging strategies can be developed accordingly and applied to cloud computing commodities.

- Insurances

  Insurance mechanisms are inevitable for cloud computing commodities. Especially, the loss of profit in case of resource failure, which is independent from the actual cost of the commodity (not independent from the rating of the subcommodity), enables the creation of a variety of insurance mechanisms and *insurance-like* financial instruments like CDSs (Credit Default Swaps) that compensate such losses. Insurance products represented e.g. by CDS like instruments could be realised as tradable products. Potential losses in the case of a cloud service failing in its operation could be insured by a buyer acquiring a certain amount of such insurance products.

  Insurances against non-delivery or non-payment are not necessary for future contracts because these financial instruments are standardized, regulated, and the trading is done on an exchange. The financial settlement of futures is performed over a clearinghouse.

Clearinghouses request margin payments upfront and take the counterparty risk. Future holders are requested to adjust their margins in the case of changes in the price (margin call if the provided margin is less than the margin which is deposited at the clearing house). In the case of a non-delivery or non-payment the clearinghouse covers the loss. In the financial sense clearinghouses have to be strong positioned.

Concluding remarks

As discussed in this chapter, exchange based marketplaces enable the creation of complex financial instruments. Looking back to former centuries marketplaces for commodities, stocks, and respective derivative financial products have encountered problems. As briefly mentioned in Chapter 3, a very prominent example happened in the Netherlands during the so called "Dutch Golden Age". The Netherlands are famous for their tulips not just today. They have been famous in former days as well, especially among emperors and wealthy people all over Europe. In the Netherlands tulip bulbs and their derivatives could be traded on an exchange. The trading of tulip bulb contracts has been a high-flying business. The price for certain types of bulbs was skyrocketing but the boost was an unhealthy, even noxious increase. In the year 1637 the price for certain types of tulip bulbs reached a level of more than the value of houses in the centre of Amsterdam when the bubble burst. Various people, not only traders and salesmen, lost their entire property. In literature (e.g. [44]), this development is reported as *tulip mania* or *tulip boom*. The most expensive tulip bulbs sold before the crash have been the bulbs of the Semper Augustus (Figure 7.1), which should be kept in mind when trading cloud computing commodities and their derivatives.

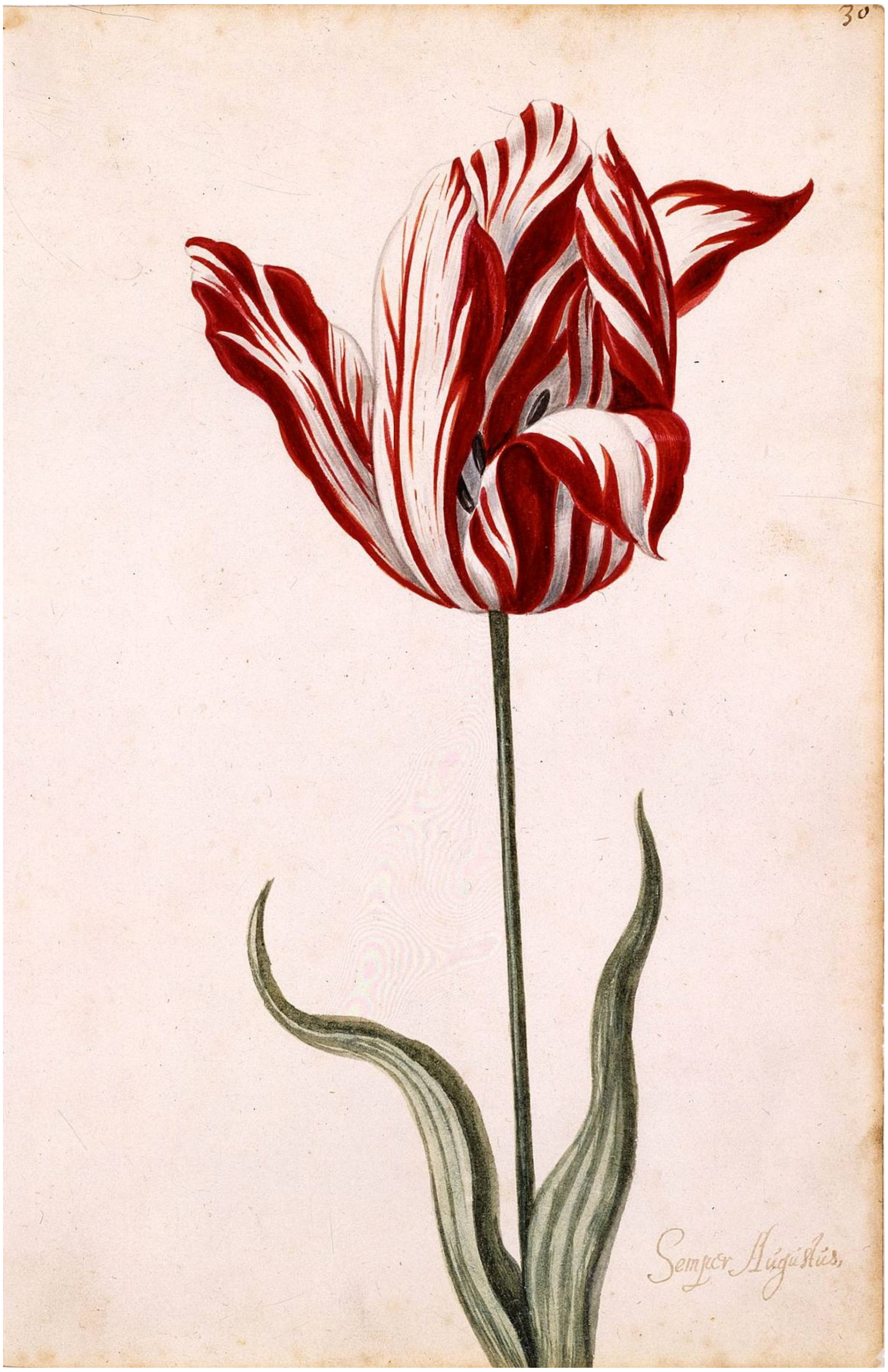


Figure 7.1.: Semper Augustus, picture source: Wikimedia Commons

Appendix A

The NIST Definition of Cloud Computing [60]

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

- Essential Characteristics:
  - On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
  - Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
  - Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data centre). Examples of resources include storage, processing, memory, and network bandwidth.
  - Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with de-

mand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability [1] at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both, the provider and consumer of the utilized service.

• Service Models:

- Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure [2]. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user- specific application configuration settings.
- Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming

  languages, libraries, services, and tools supported by the provider.[3] The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.
- Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

[1]Typically this is done on a pay-per-use or charge-per-use basis.

[2]A cloud infrastructure is the collection of hardware and software that enables the five essential characteristics of cloud computing. The cloud infrastructure can be viewed as containing both, a physical layer and an abstraction layer. The physical layer consists of the hardware resources that are necessary to support the cloud services being provided, and typically includes server, storage and network components. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential cloud characteristics. Conceptually the abstraction layer sits above the physical layer.

[3]This capability does not necessarily preclude the use of compatible programming languages, libraries, services, and tools from other sources.

- Deployment Models:

  - Private cloud. The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

  - Community cloud. The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

  - Public cloud. The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

  - Hybrid cloud. The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Appendix B

Source Code Excerpts of the Framework

In Appendix B, selected pieces of the source code of the implementation of the framework are shown as parts of the proof-of-concept implementation used for the simulations (see Chapter 5).

The simulation of the cloud computing commodities exchange consists of various classes, which are described in Chapter 5. As an extract of the entire source code, code snippets of the two classes (*Marketparticipant* and *TradingClearingSettlementSimulation*) have been choosen to be shown in the following.

B.1. Marketparticipant source code

The *run()* function, which is executed when a marketparticipant thread is started, is shown below. In this function, a marketparticipant determines in which subcommodity the orders can be placed, creates and places orders based on input parameters and random numbers.

B.1.1. Class MarketParticipant

```
//...
  public void run() {

    String stocksave = stock;

    // OrderBook book = exchange.getBook(stock);
    while (true) {
      try {
        synchronized (this) {
          while (paused)
            wait();
        }
      } catch (InterruptedException e1) {
        e1.printStackTrace();
      }
      if (this.money > 0) {
        // generating random order
        Random ran = new Random();
        BaseOrder.SIDE side = BaseOrder.SIDE.BID;
        ran.nextBoolean();

        if (ratings.size() > 0) {
          long ratingp = getNumtrades() / rptrans;
          int intrating;
          int index = (int) ratingp;
          if ((long) index != ratingp) {
            System.out.println("error casting " + ratingp
                + " to integer");

          }

          double currentrating = ratings.get(index);

          intrating = (int) Math.round(currentrating);

        switch (intrating) {

        case 9:
          stock = (stock + "-AAA");
          break;

        case 8:
          stock = (stock + "-AA");
          break;

        case 7:
          stock = (stock + "-A");
          break;

        case 6:
          stock = (stock + "-BBB");
          break;

        case 5:
          stock = (stock + "-BB");
          break;

        case 4:
          stock = (stock + "-B");
          break;

        case 3:
          stock = (stock + "-CCC");
          break;

        case 2:
          stock = (stock + "-CC");
          break;

        case 1:
          stock = (stock + "-C");
          break;

        case 0:
          stock = (stock + "-D");
          break;

        default:
          System.out.println(stock
```

```java
            + "Error determining subcommodity");
        break;

      }

    }

    OrderBook book = exchange.getBook(stock);

    if (role == 0) {
      side = BaseOrder.SIDE.ASK;
    } else {

      if (role == 1) {
        side = BaseOrder.SIDE.BID;
      } else {
        if (ran.nextBoolean()) {
          side = BaseOrder.SIDE.ASK;
        }
      }
    }
    Double sign = ran.nextBoolean() ? 1.0 : -1.0;

    if (ran.nextBoolean()) {
      if ((exchange.getBook(stock).getTrades().size() - 1) > 1) {
        basePrice = exchange
            .getBook(stock)
            .getTrades()
            .get(exchange.getBook(stock).getTrades().size() - 1)
            .getPrice();
      }
    }

    Double price = basePrice
        * (1 + sign * ran.nextInt(priceVariant) / 10000.0);
    price = priceStep.getRoundedPrice(exchange.getName(), stock,
        price);

    if (side == BaseOrder.SIDE.BID) {
      Double bestAsk = book.getBestAsk();
```

```
          price = (null != bestAsk && price > bestAsk) ?
              bestAsk
                : price;
        } else {
          Double bestBid = book.getBestBid();
          price = (null != bestBid && price < bestBid) ?
              bestBid
                : price;
        }

        int quantity = (ran.nextInt(maxQuantity - minQuantity)
            + minQuantity)
              / minQuantity * minQuantity;

        exchange.enterOrder(stock, Order.TYPE.LIMIT, side,
            quantity,
              price, participantname, "");
        stock = stocksave;
        try {
          sleep(tradingMinInterval
              + ran.nextInt(tradingMaxInterval
                  - tradingMinInterval));
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }else{
        System.out.println(this.getName()+" out of money.");
      }
    }

  }

//...
```

B.2. Simulation framework source code

The *TradingClearingSettlementSimulation* class is the core of the simulation putting together all the pieces necessary from trading to clearing, settlement, and rating. In the following source code part of the *TradingClearingSettlementSimulation* class the functions are shown, which are run periodically responsible for executing the market close operations, the trader starting, and the secure clearing and settlement operations.

B.2.1. Class TradingClearingSettlementSimulation

```
//...
  private static TransactionSimulator transactionsim;

  private static Vector<String> listings = new Vector<String>()
      ;
  private static Vector<KeyMp> traders = new Vector<KeyMp>();

  private static int delay = 9000; // i.e. 5000 delay for 5 sec
      .
  private static int period = 25000; // i.e. 1000 repeat every
      sec.
  private static int firsttickdelay = 10000;
  private static int tick = 1000; // i.e. 1000 repeat every sec
      .
  private static Timer timer = new Timer();

  private static Timer ticker = new Timer();
  private int maxticks = 61;
  // private int maxticks = 3600;
  private int numticks = 0;

  private boolean marketcloselock = false;

  private static Timer pausetraders = new Timer();
  private int steps = 11000;

//...

  class RemindTask extends TimerTask {
    public void run() {
      int i;
      synchronized (this) {
        try {
```

```
        semaphore.take();
      } catch (InterruptedException e1) {
        System.out.println(e1);
        e1.printStackTrace();
      }
      try {
        // if (!marketcloselock) {

        MarketParticipant mp;

        marketcloselock = true;
        listings = exchange.getListings();
        traders = exchange.getTraders();

        for (i = 0; i < traders.size(); i++) {
          traders.get(i).getMP().pause();
          // System.out.println("Trader paused " +
          // traders.get(i).getMP().getParticipantName());

        }
        System.out
            .println("++++++++++++++++++++++++++++ Market
               Closed ++++++++++++++++++++++++++++++++++");

        System.out.println("Clearing & Settlement");

        System.out.println("Initializing transaction
            simulator");

        System.out.println("Settle trades");
        transactionsim.settletrades();
        System.out.println("Done");

        for (i = 0; i < listings.size(); i++) {
          exchange.getBook(listings.get(i))
              .deleteExpiredLimitOrders();
          exchange.getBook(listings.get(i)).clearTrades();
          // System.out.println(listings.get(i));

        }

        System.out
            .println("++++++++++++++++++++++++++++ Market
               Opened ++++++++++++++++++++++++++++++++++");

        for (i = 0; i < traders.size(); i++) {
          traders.get(i).getMP().start();
        }
      } finally {
        try {
          semaphore.release();
        } catch (InterruptedException e) {
          System.out.println(e);
          e.printStackTrace();
        }
      }
    }
  }
}

class LogTask extends TimerTask {
  public void run() {
    synchronized (this) {
        for(int i=0;i<100000;i++){
          System.out.print(numticks+" ");
          if(i%50==0){
            System.out.println();
          }
        }

        System.out
            .println("Timer Tick" + numticks + " " + maxticks);
        if (numticks < maxticks) {
          numticks++;

          Vector<String> strlistings = exchange.getListings();
          Object[] bookd;

          for (int i = 0; i < strlistings.size(); i++) {
            bookd = exlisting.getBookData(exchange
                .getBook(strlistings.get(i)));
            for (int j = 0; j < bookd.length; j++) {
              out.print(bookd[j].toString() + ";");
            }
            out.println();
          }

          System.out.println("numticks " + numticks);
        } else {
          out.close();

          for (int i = 0; i < participants.size(); i++) {

            System.out.println("Trader money "
                + participants.get(i).getParticipantName()
                + ": " + participants.get(i).getMoney());

          }
          System.exit(0);
        }
    }
  }
}

class PauseTask extends TimerTask {
  public void run() {
    int rand;
    Random ran = new Random();
    System.out
        .println("Random trader pausing"
            + participants.size());
    synchronized (this) {
      try {
        semaphore.take();
      } catch (InterruptedException e1) {
        System.out.println(e1);
        e1.printStackTrace();
      }
      try {
        for (int i = 0; i < participants.size(); i++) {

          rand = ran.nextInt(5);

          if (rand == 1) {
            participants.get(i).pause();
          } else {
            participants.get(i).start();
          }
        }
      } finally {
```

```
                try {
                    semaphore.release();
                } catch (InterruptedException e) {
                    System.out.println(e);
                    e.printStackTrace();
                }

            }
        }
    }

}

//...
```

B.3. Rating simulation source code

The rating agency naturally represents an independent entity. Initially, the rating simulation has been implemented in Mathematica. In Chapter 4 4.4 the methods are described the following code is based on.

B.3.1. Mathematica source code

Calculation of rating period values

The values representing waiting times in this simulation can be either created randomly or read from a file. These values are then checked if they are between certain borders, below or above. Accordingly, the values -1, 0 or 1 are assigned and placed in a list.

```
wait = ReadList["~/Documents/ratingsim/wait.txt", Number];

work = {};

tmp = {};

i = 1;

wrkcount = 0;

limit = 9000;

max = 2000;

nlimit = 2000 * 0.15;

While[i < Length[wait],


tmp = Append[tmp, wait[[i]]];

If[Mod[i, max] == 0,

wrktmp = Mean[tmp];

ll = Length[Select[tmp, # > 2 * limit&]];

If[ll > nlimit, wrktmp = (limit * 1.4 + 1000), ];

work = Append[work, wrktmp]; tmp = {}, ];

i++;

];

Print[i];
```

```
realin = {};
For[i = 1, i < Length[work], i++,
If[(work[[i]]>=(limit/1.4))&&(work[[i]]<=(limit * 1.4)),
realin = Append[realin, 0],
If[work[[i]] > (limit * 1.4), realin = Append[realin, −1],
If[work[[i]] < (limit/1.4), realin = Append[realin, 1], ]]];
];
Save["~realin.txt", realin]
```

Calculation of rating change values

The rating change values represent the direction of the rating in the next timestep. -2 stands for down-rating, 2 for up-rating, 0 for stay, -1 for negative outlook, and 1 for positive outlook. The following source code is an implementation of Algorithm 4.1.

```
(*ratinginput = RandomInteger[{−1, 1}, 50]; *)
(*ratinginput = Round[RandomVariate[NormalDistribution[−0.11, 1/2], 5000]];
For[i = 1, i < Length[ratinginput], i++,
If[ratinginput[[i]] > 1, ratinginput[[i]] = 1, ];
If[ratinginput[[i]] < −1, ratinginput[[i]] = −1, ]
]; *)
ratinginput = ReadList["~realin.txt", Number];
rated = False;
ratingchangelist = List[0, 0, 0, 0]; srl = SparseArray[ratingchangelist];
For[i = 1, i < Length[ratinginput] − 3, i++,
rated = False;
fff = ratinginput[[i;;i + 4]];
ratingchange = 0;
If[(fff[[1]]==fff[[2]]&&fff[[2]]==fff[[3]]&&fff[[3]]==fff[[4]]&&
fff[[4]]==fff[[5]]), ratingchange = fff[[3]] * 2; rated = True, ];
If[rated == False,
If[((fff[[1]]==fff[[2]]&&fff[[2]]==fff[[3]]&&fff[[3]]==fff[[4]])||
(fff[[2]]==fff[[3]]&&fff[[3]]==fff[[4]]&&fff[[4]]==fff[[5]])),
If[MemberQ[fff, 0] == True, ratingchange = fff[[3]] * 2; rated = True,
ratingchange = fff[[3]]; rated = True]], ];
If[rated == False,
For[j = 1, j < Length[fff] - 1, j++,
If[fff[[j]] == fff[[j + 1]]&&fff[[j + 1]] == fff[[j + 2]],
If[MemberQ[fff, fff[[j]] * -1] == False, ratingchange = fff[[j]]; rated = True,
ratingchange = 0; rated = True], ]], ];
If[rated == False,
For[j = 1, j < Length[fff], j++,
If[fff[[j]] == fff[[j + 1]]&&fff[[j]] ≠ 0,
ratingchange = ratingchange + fff[[j]], ]]; ];
srl = Append[srl, ratingchange];
ratingchangelist = Normal[srl];

initialrating = 5;
rating = initialrating;
ratinglist = List[initialrating];
sr = SparseArray[ratinglist];
For[i = 4, i < Length[ratingchangelist], i++,

Switch[ratingchangelist[[i]], -2, rating–, -1, rating-=0.3, 1, rating+=0.3,
2, rating++];
If[rating < 0, rating = 0, ];
```

```
If[rating > 9.3, rating = 9.3, ];
(*Print[rating]; *)
sr = Append[sr, rating];
rating = Round[rating]];


ratinglist = Normal[sr];
ListPlot[ratinglist, Joined → True, InterpolationOrder → 0, Mesh → Full,
ColorFunction → "SouthwestColors"]
```

B.3.2. Java source code

For the integration of the rating simulation implemented in Mathematica into the exchange simulation, the source code has been ported in Java. A class WaitSLA has been created for handling SLAs based on waiting times of certain services. The following source code snippets from the class *WaitSLA* show the two functions responsible for calculating the rating values.

B.3.3. Class WaitSLA

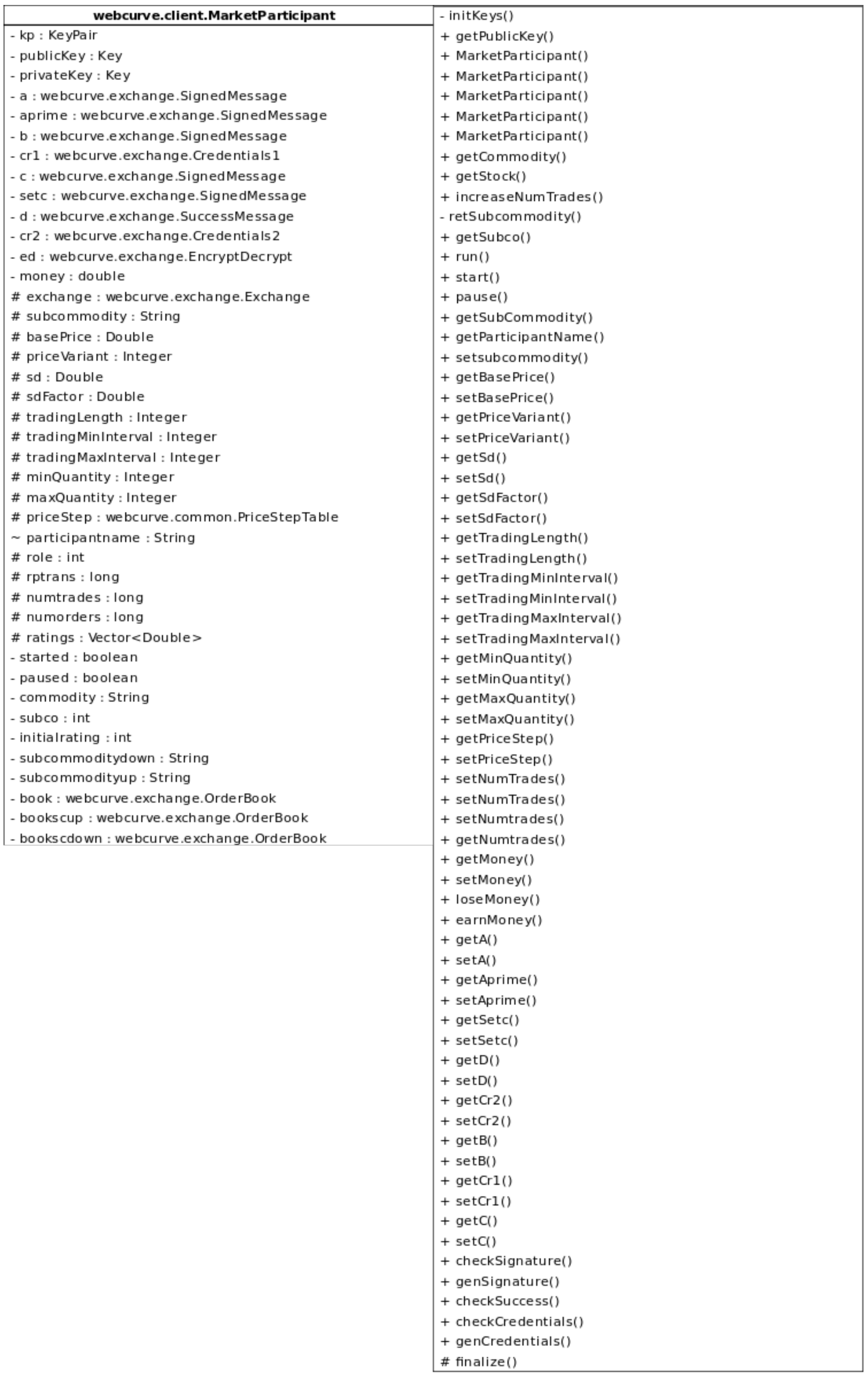
```


  public int quantifyRPsSeq (){

    int i=0;
    int nv=0;
    double wrktmp=0.0;
    double dtmp;
    double sum=0.0;
    int counter=0;
    double d1,d2,d3;
    Double dd1, dd2, dd3;

    File infile = new File(filename);
    this.rpquan=new Vector<Integer >();
    Scanner scanner;
    Vector tmp;
    Vector work;

    if(!(infile.exists())) {
      return (-1);
    }
    else{

      try{
        scanner = new Scanner(infile);
      }
      catch (FileNotFoundException ex) ;
        return (-1);
      }

      while(scanner.hasNextDouble()){
        counter++;
        waittimes.add(scanner.nextDouble());
      }

      tmp=new Vector();
      work=new Vector();
      while(i < waittimes.size()){

        tmp.add(waittimes.get(i));
        if (((i%(rptrans))==0) && (i!=0)){
          nv=0;
          sum=0;
          for(int j=0;j<tmp.size();j++){
            dtmp=((Double)tmp.get(j)).doubleValue();
            sum=sum+dtmp;

            dd2 = new Double(limit*2);
            dd3 = ((Double)tmp.get(j)).doubleValue();
            if(dd3.compareTo(dd2)>0){

              nv++;
            }
          }
          dd2 = new Double(nv);
          dd3 = new Double(rptrans*violationlimit);

          if(dd2.compareTo(dd3)>0){
            wrktmp=limit*epsilonp+1000;
          }
          else{
            wrktmp=sum/tmp.size();
          }

          work.add(wrktmp);
          tmp = new Vector();
          wrktmp=0;
        }

        i++;

      }


      d2=limit/epsilonp;
      d3=limit*epsilonp;

      dd2 = new Double(d2);
      dd3 = new Double(d3);
      for(i=0;i<work.size();i++){
        d1=((Double)work.get(i)).doubleValue();
        dd1 = new Double(d1);
        if((dd1.compareTo(dd2)>=0) && (dd1.compareTo(dd3)<=0)){
          this.rpquan.add(new Integer(0));
        }
        else{
          if(dd1.compareTo(dd3)>0){
            this.rpquan.add(new Integer(-1));
          }
          else{
            if(dd1.compareTo(dd2)<0){
              this.rpquan.add(new Integer(1));
            }
          }
        }
      }
      return (0);
    }
  }


  public int ratingchangeSeq (){

    int ratingchange=0;
    int r1,r2,r3,r4,r5;
     int[] rn;
     double rating;
     rn=new int[5];
     boolean rated=false;

     Double dd1, dd2;

     ratingchanges.add(new Integer(0));
     ratingchanges.add(new Integer(0));
     ratingchanges.add(new Integer(0));
       ratingchanges.add(new Integer(0));

    for(int i=0;i<rpquan.size()-4;i++){

      ratingchange=0;

      rn[0]=r1=((Integer)rpquan.get(i)).intValue();
```

```
        rn[1]=r2=((Integer)rpquan.get(i+1)).intValue();
          rn[2]=r3=((Integer)rpquan.get(i+2)).intValue();
        rn[3]=r4=((Integer)rpquan.get(i+3)).intValue();
        rn[4]=r5=((Integer)rpquan.get(i+4)).intValue();

        if((r1==r2) && (r2==r3) && (r3==r4) && (r4==r5)){
          ratingchange=r3*2;
          rated=true;
        }
        else{
          if(!rated){
            if(((r1==r2) && (r2==r3) && (r3==r4) && (r5==0)) ||
                ((r2==r3) && (r3==r4) && (r4==r5) && (r1==0))){
              ratingchange=r3*2;
              rated=true;
            }
            else{
              if(((r1==r2) && (r2==r3) && (r3==r4)) || ((r2==r3)
                  && (r3==r4) && (r4==r5))){
                ratingchange=r3;
                rated=true;
              }
            }
          }
          if(!rated){
            for(int j=0;j<rn.length-2;j++){
              if((rn[j]==rn[j+1]) && (rn[j+1]==rn[j+2]) && (!
                  rated)){
                if((rn[((j+3)%5)]==0) && (rn[((j+4)%5)]==0)){
                  ratingchange=rn[j];
                  rated=true;
                }
                else{
                  ratingchange=0;
                  rated=true;
                }
              }
            }
          }
          if(!rated){
            for(int j=0;j<rn.length-1;j++){
                if((rn[j]==rn[j+1]) && (rn[j]!=0)){
                ratingchange=ratingchange+rn[j];


                }
              }
            }
          }
            ratingchanges.add(new Integer(ratingchange));
            rated=false;
          }
      rating=initialrating;

      ratings.add(new Double(initialrating));

      for(int i=3;i<ratingchanges.size();i++){

        if(((Integer)ratingchanges.get(i)).intValue()==-2){
          rating=rating-1.0;
        }

        if(((Integer)ratingchanges.get(i)).intValue()==-1){
          rating=rating-0.3;
        }

        if(((Integer)ratingchanges.get(i)).intValue()==1){
          rating=rating+0.3;
        }

        if(((Integer)ratingchanges.get(i)).intValue()==2){
          rating=rating+1.0;
        }

        if(((Integer)ratingchanges.get(i)).intValue()==0){
          rating=rating-0.0;
        }

        dd1=new Double(rating);
        dd2=new Double(0.0);

        if(dd1.compareTo(dd2)<0){
          rating=dd2;
        }

        dd2=new Double(9.3);

        if(dd1.compareTo(dd2)>0){
            rating=dd2;
          }
          ratings.add(new Double(rating));
          rating=Math.round(rating);
        }
        return(0);
      }
```

Appendix C

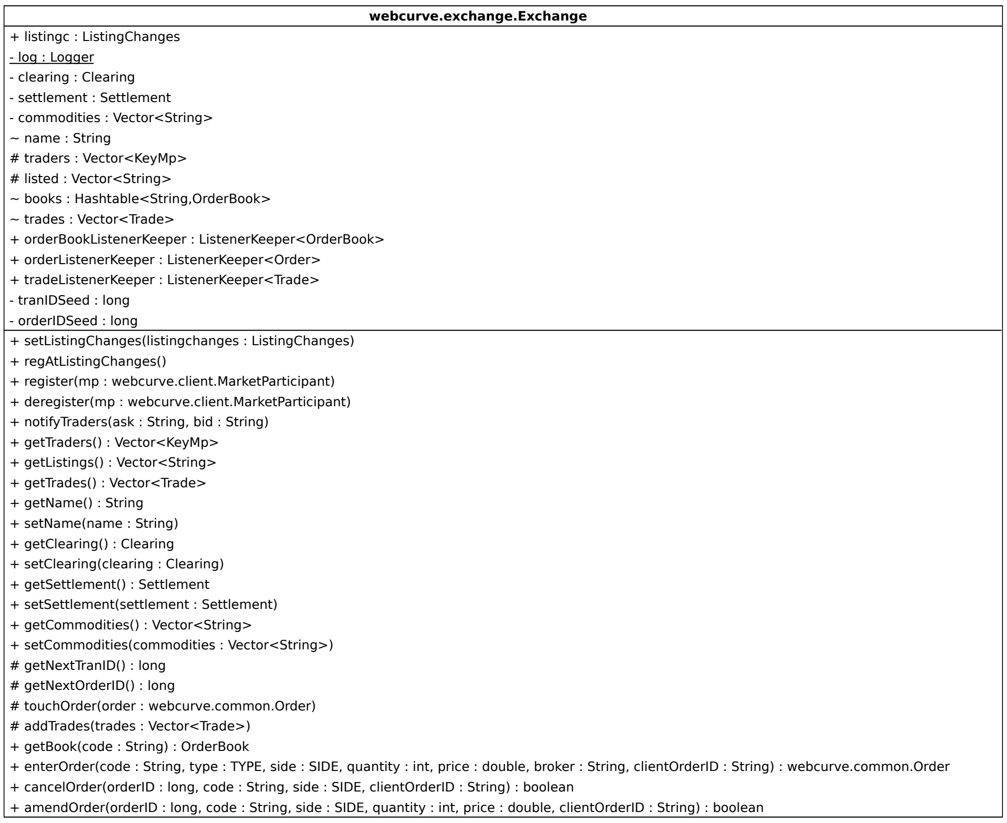
UML Diagrams

This appendix shows the UML diagrams of the classes, the UML sequence diagram of the secure clearing and settlement process, and the UML class diagram of the entire framework.



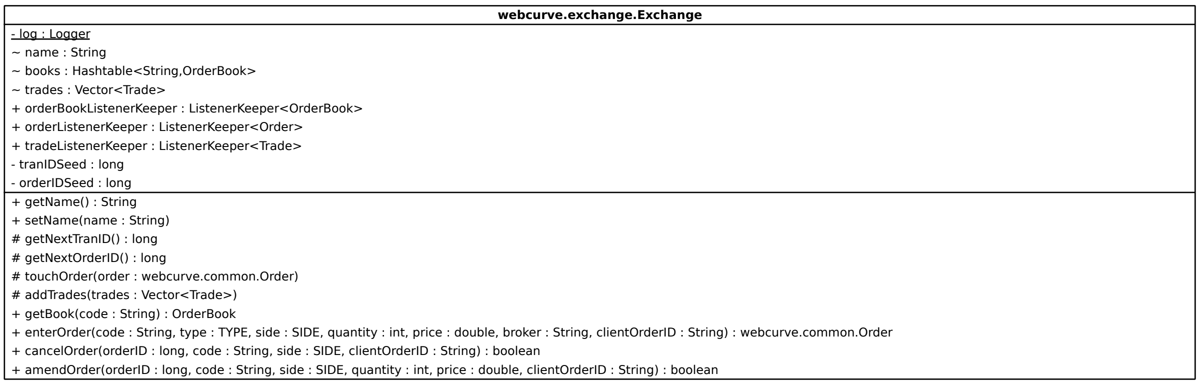
```
webcurve.client.MarketParticipant
- kp : KeyPair
- publicKey : Key
- privateKey : Key
- a : webcurve.exchange.SignedMessage
- aprime : webcurve.exchange.SignedMessage
- b : webcurve.exchange.SignedMessage
- cr1 : webcurve.exchange.Credentials1
- c : webcurve.exchange.SignedMessage
- setc : webcurve.exchange.SignedMessage
- d : webcurve.exchange.SuccessMessage
- cr2 : webcurve.exchange.Credentials2
- ed : webcurve.exchange.EncryptDecrypt
- money : double
# exchange : webcurve.exchange.Exchange
# subcommodity : String
# basePrice : Double
# priceVariant : Integer
# sd : Double
# sdFactor : Double
# tradingLength : Integer
# tradingMinInterval : Integer
# tradingMaxInterval : Integer
# minQuantity : Integer
# maxQuantity : Integer
# priceStep : webcurve.common.PriceStepTable
~ participantname : String
# role : int
# rptrans : long
# numtrades : long
# numorders : long
# ratings : Vector<Double>
- started : boolean
- paused : boolean
- commodity : String
- subco : int
- initialrating : int
- subcommoditydown : String
- subcommodityup : String
- book : webcurve.exchange.OrderBook
- bookscup : webcurve.exchange.OrderBook
- bookscdown : webcurve.exchange.OrderBook

- initKeys()
+ getPublicKey()
+ MarketParticipant()
+ MarketParticipant()
+ MarketParticipant()
+ MarketParticipant()
+ MarketParticipant()
+ getCommodity()
+ getStock()
+ increaseNumTrades()
- retSubcommodity()
+ getSubco()
+ run()
+ start()
+ pause()
+ getSubCommodity()
+ getParticipantName()
+ setsubcommodity()
+ getBasePrice()
+ setBasePrice()
+ getPriceVariant()
+ setPriceVariant()
+ getSd()
+ setSd()
+ getSdFactor()
+ setSdFactor()
+ getTradingLength()
+ setTradingLength()
+ getTradingMinInterval()
+ setTradingMinInterval()
+ getTradingMaxInterval()
+ setTradingMaxInterval()
+ getMinQuantity()
+ setMinQuantity()
+ getMaxQuantity()
+ setMaxQuantity()
+ getPriceStep()
+ setPriceStep()
+ setNumTrades()
+ setNumTrades()
+ setNumtrades()
+ getNumtrades()
+ getMoney()
+ setMoney()
+ loseMoney()
+ earnMoney()
+ getA()
+ setA()
+ getAprime()
+ setAprime()
+ getSetc()
+ setSetc()
+ getD()
+ setD()
+ getCr2()
+ setCr2()
+ getB()
+ setB()
+ getCr1()
+ setCr1()
+ getC()
+ setC()
+ checkSignature()
+ genSignature()
+ checkSuccess()
+ checkCredentials()
+ genCredentials()
# finalize()
```

Figure C.1.: UML diagram of the MarketParticipant class



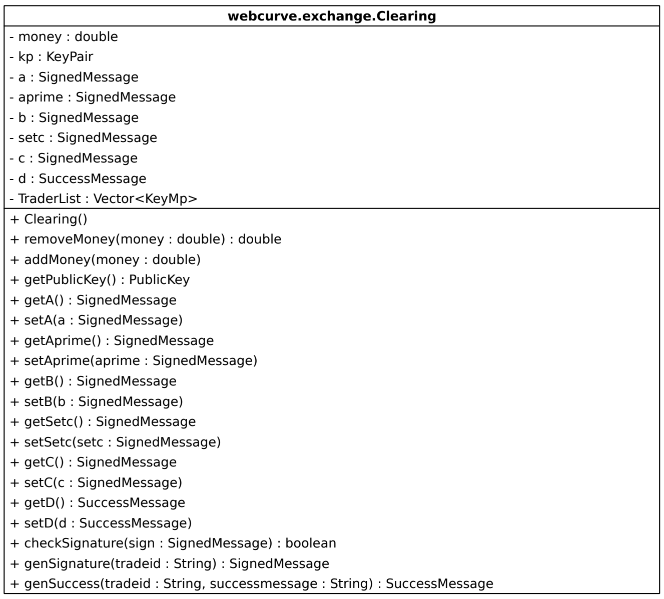
```
webcurve.exchange.Exchange
+ listingc : ListingChanges
- log : Logger
- clearing : Clearing
- settlement : Settlement
- commodities : Vector<String>
~ name : String
# traders : Vector<KeyMp>
# listed : Vector<String>
~ books : Hashtable<String,OrderBook>
~ trades : Vector<Trade>
+ orderBookListenerKeeper : ListenerKeeper<OrderBook>
+ orderListenerKeeper : ListenerKeeper<Order>
+ tradeListenerKeeper : ListenerKeeper<Trade>
- tranIDSeed : long
- orderIDSeed : long
+ setListingChanges(listingchanges : ListingChanges)
+ regAtListingChanges()
+ register(mp : webcurve.client.MarketParticipant)
+ deregister(mp : webcurve.client.MarketParticipant)
+ notifyTraders(ask : String, bid : String)
+ getTraders() : Vector<KeyMp>
+ getListings() : Vector<String>
+ getTrades() : Vector<Trade>
+ getName() : String
+ setName(name : String)
+ getClearing() : Clearing
+ setClearing(clearing : Clearing)
+ getSettlement() : Settlement
+ setSettlement(settlement : Settlement)
+ getCommodities() : Vector<String>
+ setCommodities(commodities : Vector<String>)
# getNextTranID() : long
# getNextOrderID() : long
# touchOrder(order : webcurve.common.Order)
# addTrades(trades : Vector<Trade>)
+ getBook(code : String) : OrderBook
+ enterOrder(code : String, type : TYPE, side : SIDE, quantity : int, price : double, broker : String, clientOrderID : String) : webcurve.common.Order
+ cancelOrder(orderID : long, code : String, side : SIDE, clientOrderID : String) : boolean
+ amendOrder(orderID : long, code : String, side : SIDE, quantity : int, price : double, clientOrderID : String) : boolean
```

Figure C.2.: UML diagram of the Exchange class



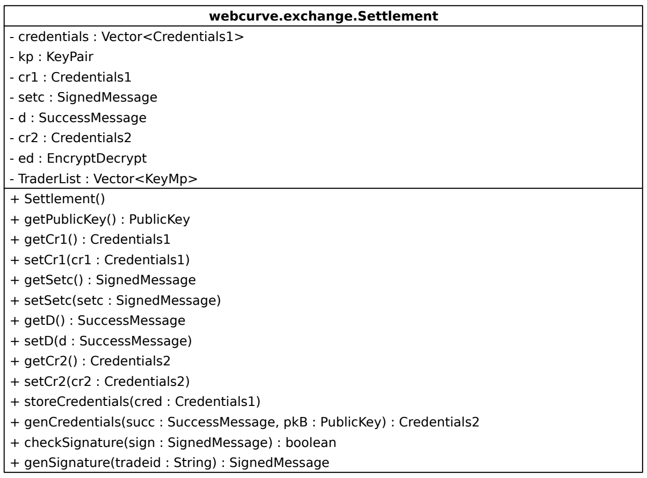
```
webcurve.exchange.Exchange
- log : Logger
~ name : String
~ books : Hashtable<String,OrderBook>
~ trades : Vector<Trade>
+ orderBookListenerKeeper : ListenerKeeper<OrderBook>
+ orderListenerKeeper : ListenerKeeper<Order>
+ tradeListenerKeeper : ListenerKeeper<Trade>
- tranIDSeed : long
- orderIDSeed : long
+ getName() : String
+ setName(name : String)
# getNextTranID() : long
# getNextOrderID() : long
# touchOrder(order : webcurve.common.Order)
# addTrades(trades : Vector<Trade>)
+ getBook(code : String) : OrderBook
+ enterOrder(code : String, type : TYPE, side : SIDE, quantity : int, price : double, broker : String, clientOrderID : String) : webcurve.common.Order
+ cancelOrder(orderID : long, code : String, side : SIDE, clientOrderID : String) : boolean
+ amendOrder(orderID : long, code : String, side : SIDE, quantity : int, price : double, clientOrderID : String) : boolean
```

Figure C.3.: UML diagram of the original webcurvesim Exchange class



```
webcurve.exchange.Clearing
- money : double
- kp : KeyPair
- a : SignedMessage
- aprime : SignedMessage
- b : SignedMessage
- setc : SignedMessage
- c : SignedMessage
- d : SuccessMessage
- TraderList : Vector<KeyMp>
+ Clearing()
+ removeMoney(money : double) : double
+ addMoney(money : double)
+ getPublicKey() : PublicKey
+ getA() : SignedMessage
+ setA(a : SignedMessage)
+ getAprime() : SignedMessage
+ setAprime(aprime : SignedMessage)
+ getB() : SignedMessage
+ setB(b : SignedMessage)
+ getSetc() : SignedMessage
+ setSetc(setc : SignedMessage)
+ getC() : SignedMessage
+ setC(c : SignedMessage)
+ getD() : SuccessMessage
+ setD(d : SuccessMessage)
+ checkSignature(sign : SignedMessage) : boolean
+ genSignature(tradeid : String) : SignedMessage
+ genSuccess(tradeid : String, successmessage : String) : SuccessMessage
```

Figure C.4.: UML diagram of the Clearing class



```
webcurve.exchange.Settlement
- credentials : Vector<Credentials1>
- kp : KeyPair
- cr1 : Credentials1
- setc : SignedMessage
- d : SuccessMessage
- cr2 : Credentials2
- ed : EncryptDecrypt
- TraderList : Vector<KeyMp>
+ Settlement()
+ getPublicKey() : PublicKey
+ getCr1() : Credentials1
+ setCr1(cr1 : Credentials1)
+ getSetc() : SignedMessage
+ setSetc(setc : SignedMessage)
+ getD() : SuccessMessage
+ setD(d : SuccessMessage)
+ getCr2() : Credentials2
+ setCr2(cr2 : Credentials2)
+ storeCredentials(cred : Credentials1)
+ genCredentials(succ : SuccessMessage, pkB : PublicKey) : Credentials2
+ checkSignature(sign : SignedMessage) : boolean
+ genSignature(tradeid : String) : SignedMessage
```

Figure C.5.: UML diagram of the Settlement class

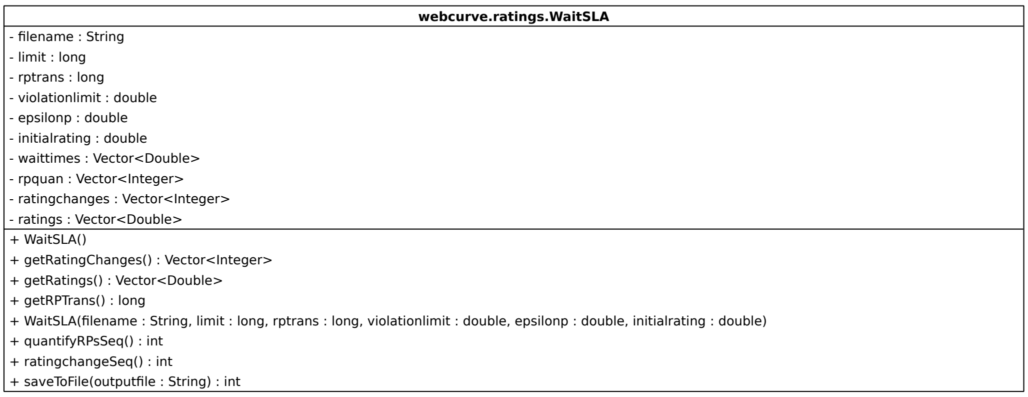

webcurve.ratings.WaitSLA
- filename : String
- limit : long
- rptrans : long
- violationlimit : double
- epsilonp : double
- initialrating : double
- waittimes : Vector<Double>
- rpquan : Vector<Integer>
- ratingchanges : Vector<Integer>
- ratings : Vector<Double>
+ WaitSLA()
+ getRatingChanges() : Vector<Integer>
+ getRatings() : Vector<Double>
+ getRPTrans() : long
+ WaitSLA(filename : String, limit : long, rptrans : long, violationlimit : double, epsilonp : double, initialrating : double)
+ quantifyRPsSeq() : int
+ ratingchangeSeq() : int
+ saveToFile(outputfile : String) : int

Figure C.6.: UML diagram of the WaitSLA class

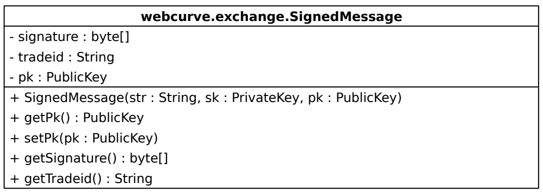


Figure C.7.: UML diagram of the SignedMessage class

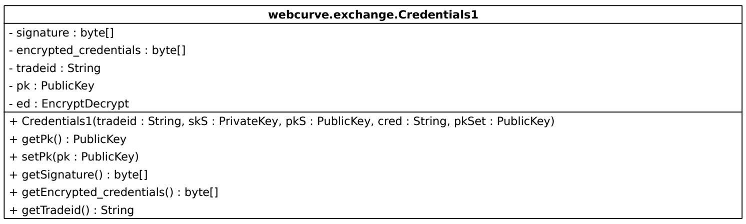

webcurve.exchange.Credentials1
- signature : byte[]
- encrypted_credentials : byte[]
- tradeid : String
- pk : PublicKey
- ed : EncryptDecrypt
+ Credentials1(tradeid : String, skS : PrivateKey, pkS : PublicKey, cred : String, pkSet : PublicKey)
+ getPk() : PublicKey
+ setPk(pk : PublicKey)
+ getSignature() : byte[]
+ getEncrypted_credentials() : byte[]
+ getTradeid() : String

Figure C.8.: UML diagram of the Credentials1 class

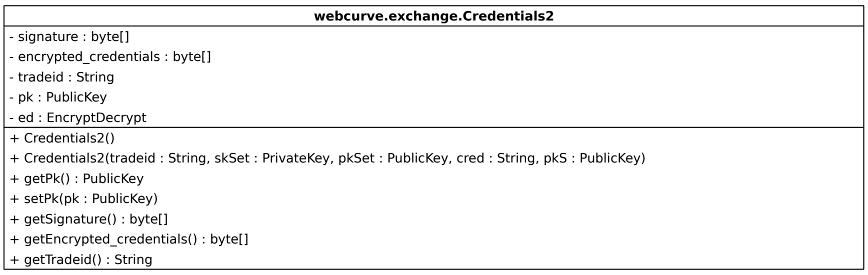

webcurve.exchange.Credentials2
- signature : byte[]
- encrypted_credentials : byte[]
- tradeid : String
- pk : PublicKey
- ed : EncryptDecrypt
+ Credentials2()
+ Credentials2(tradeid : String, skSet : PrivateKey, pkSet : PublicKey, cred : String, pkS : PublicKey)
+ getPk() : PublicKey
+ setPk(pk : PublicKey)
+ getSignature() : byte[]
+ getEncrypted_credentials() : byte[]
+ getTradeid() : String

Figure C.9.: UML diagram of the Credentials2 class

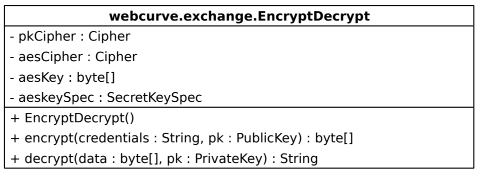

webcurve.exchange.EncryptDecrypt
- pkCipher : Cipher
- aesCipher : Cipher
- aesKey : byte[]
- aeskeySpec : SecretKeySpec
+ EncryptDecrypt()
+ encrypt(credentials : String, pk : PublicKey) : byte[]
+ decrypt(data : byte[], pk : PrivateKey) : String

Figure C.10.: UML diagram of the EncryptDecrypt class

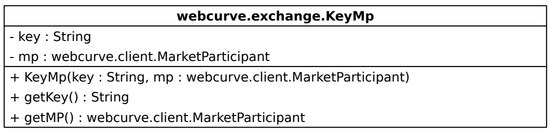

webcurve.exchange.KeyMp
- key : String
- mp : webcurve.client.MarketParticipant
+ KeyMp(key : String, mp : webcurve.client.MarketParticipant)
+ getKey() : String
+ getMP() : webcurve.client.MarketParticipant

Figure C.11.: UML diagram of the KeyMp class

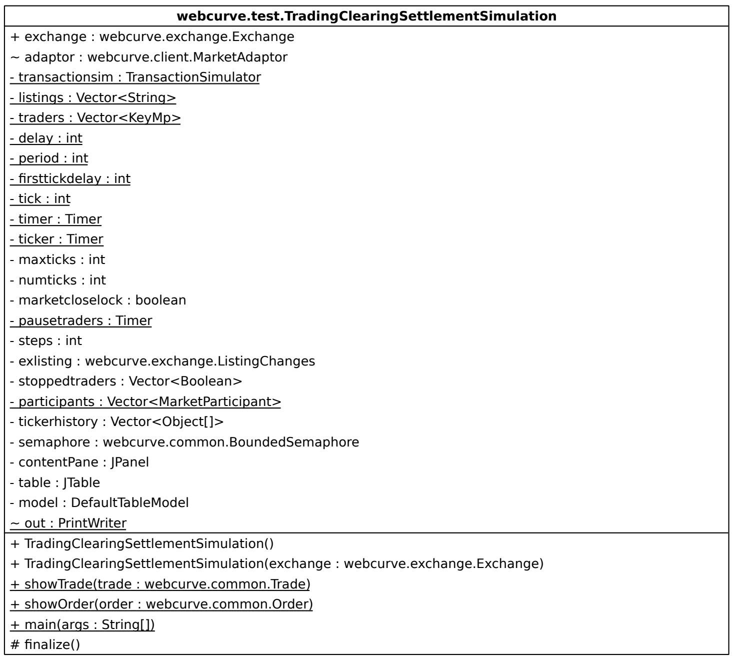

webcurve.test.TradingClearingSettlementSimulation
+ exchange : webcurve.exchange.Exchange
~ adaptor : webcurve.client.MarketAdaptor
- transactionsim : TransactionSimulator
- listings : Vector<String>
- traders : Vector<KeyMp>
- delay : int
- period : int
- firsttickdelay : int
- tick : int
- timer : Timer
- ticker : Timer
- maxticks : int
- numticks : int
- marketcloselock : boolean
- pausetraders : Timer
- steps : int
- exlisting : webcurve.exchange.ListingChanges
- stoppedtraders : Vector<Boolean>
- participants : Vector<MarketParticipant>
- tickerhistory : Vector<Object[]>
- semaphore : webcurve.common.BoundedSemaphore
- contentPane : JPanel
- table : JTable
- model : DefaultTableModel
~ out : PrintWriter
+ TradingClearingSettlementSimulation()
+ TradingClearingSettlementSimulation(exchange : webcurve.exchange.Exchange)
+ showTrade(trade : webcurve.common.Trade)
+ showOrder(order : webcurve.common.Order)
+ main(args : String[])
# finalize()

Figure C.12.: UML diagram of the TradingClearingSettlementSimulator class

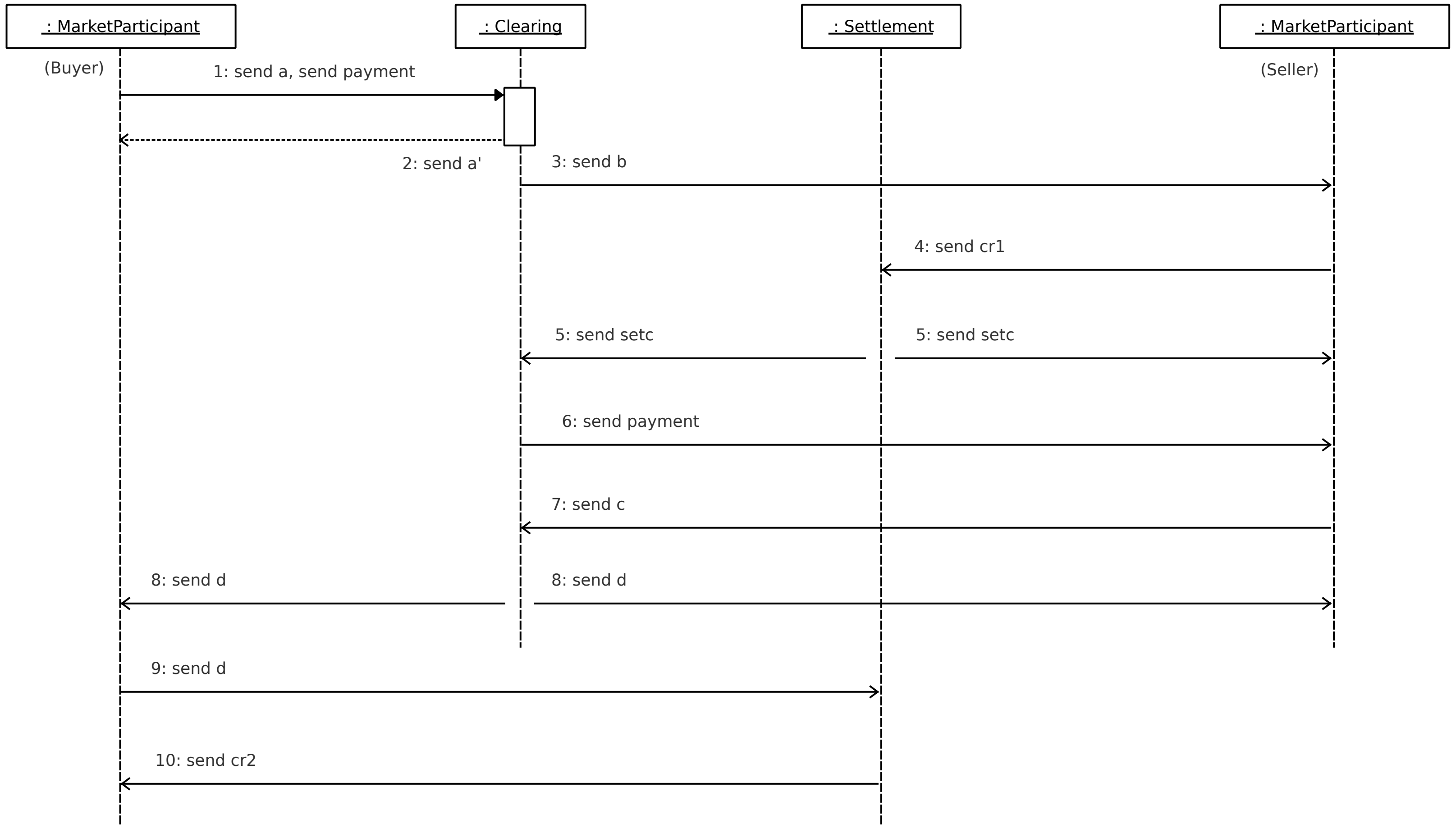


Figure C.13.: UML sequence diagram of the clearing and settlement process

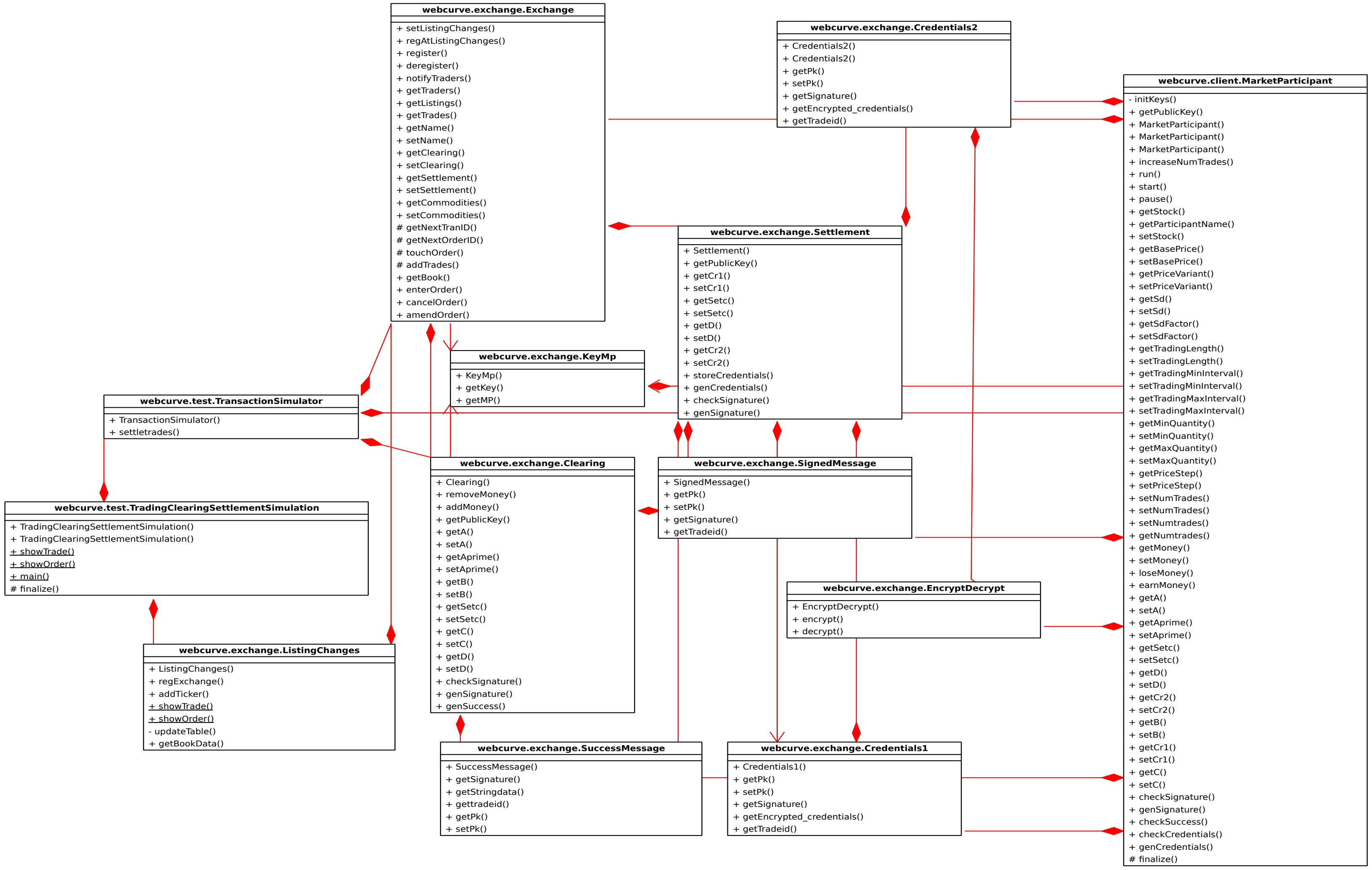


Figure C.14.: Meta UML class diagram

Acknowledgment

I would like to thank all who believed in this idea, first of all Prof. Dieter Kranzlmüller for supporting this idea from the inital thoughts in 2008, Prof. Manish Parashar for his input, the fruitful discussions and the hospitality at the Rutgers University in NJ.

I thank Shawn Findlan for working together realising the patent and trusting in the commercial value of the idea. I also express my thanks to Paul Strong for sharing his thoughts beeing a strong believer since the beginning.

I would like to thank the members of the Munich Network Management Team (MNM-Team) who helped with discussions and valuable comments.

I express my gratitude to my family and friends for their mental support during the creation of this work.

Bibliography

[1] Emerging Technologies Hype Cycle 2009. http://my.gartner.com/it/content/1101800/1101817/august12_hype_cycle_final_jfenn.pdf, 2009.

[2] DSP0243, Open Virtualization Format Specification, Version 1.1.0, 2010.

[3] Cloud Data Management Interface (CDMI), 2012.

[4] Cloud Infrastructure Management Interface (CIMI) Model and REST Interface over HTTP Specification, 2012.

[5] European Grid Infrastructure. http://www.egi.eu/, 2012.

[6] GB963, Cloud SLA Application Note, Version 1.2, 2012.

[7] Grid Observatory. http://www.grid-observatory.org/, 2012.

[8] Hype Cycle for Cloud Computing, 2012. http://www.gartner.com/id=2102116, 2012.

[9] IT Infrastructure Library, Core Cabinet Office Material. http://http://www.

`itil-officialsite.com/`, 2012.

[10] TPC-Transaction Processing Performance Council. `http://www.tpc.org/`, 2013.

[11] A. Abdul-Rahman. The PGP Trust Model. In *EDI-Forum: the Journal of Electronic Commerce*, volume 10, pages 27–31, 1997.

[12] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212. IEEE, 2012.

[13] J. Altmann, C. Courcoubetis, J. Darlington, and J. Cohen. GridEcon–The Economic-Enhanced Next-Generation Internet. *Grid Economics and Business Models*, pages 188–193, 2007.

[14] J. Altmann, C. Courcoubetis, G. Stamoulis, M. Dramitinos, T. Rayna, M. Risch, and C. Bannink. GridEcon: A Market Place for Computing Resources. *Grid Economics and Business Models*, pages 185–196, 2008.

[15] M. Armbrust, A. Fox, R. Griffith, A. D Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[16] P. Banerjee, R. Friedrich, C. Bash, P. Goldsack, B. A Huberman, J. Manley, C. Patel, P. Ranganathan, and A. Veitch. Everything as a Service: Powering the new Information Economy. *Computer*, 44(3):36–43, 2011.

[17] M. Becker. *Stromhandel an der Börse*. VDM Verlag, 2008.

[18] D. Bellenger, J. Bertram, A. Budina, A. Koschel, B. Pfänder, C. Serowy, I. Astrova, S.G.

Grivas, and M. Schaaf. Scaling in Cloud Environments. *Recent Researches in Computer Science*, 2011.

[19] O.A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 304–311. IEEE, 2011.

[20] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing. How is the Weather tomorrow?: Towards a Benchmark for the Cloud. In *Proceedings of the Second International Workshop on Testing Database Systems*, page 9. ACM, 2009.

[21] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *The journal of political economy*, pages 637–654, 1973.

[22] I. Breskovic, J. Altmann, and I. Brandic. Creating Standardized Products for Electronic Markets. *Future Generation Computer Systems*, 2012.

[23] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Proceedings of the IEEE*, 93(3):698–714, 2005.

[24] R. Buyya, S. Pandey, and C. Vecchiola. Cloudbus Toolkit for Market-Oriented Cloud Computing. *Cloud Computing*, pages 24–44, 2009.

[25] R. Buyya, S. Pandey, and C. Vecchiola. Market-Oriented Cloud Computing and the Cloudbus Toolkit. 2010.

[26] R. Buyya and S. Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. In *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, pages 19–66. IEEE, 2004.

[27] R. Buyya, C.S. Yeo, and S. Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. IEEE, 2008.

[28] D. Chen. Webcurvesim, Google Code. `http://code.google.com/p/webcurvesim/`, 2011.

[29] C. Courcoubetis, M. Dramitinos, T. Rayna, S. Soursos, and G. Stamoulis. Market Mechanisms for Trading Grid Resources. *Grid Economics and Business Models*, pages 58–72, 2008.

[30] C. Courcoubetis, S. Soursos, and R. Weber. Pricing Resources on Demand. In *Bandwidth on Demand, 2006 1st IEEE International Workshop on*, pages 12–18. IEEE, 2006.

[31] M. Crouhy, D. Galai, and R. Mark. Prototype Risk Rating System. *Journal of Banking & Finance*, 25(1):47–95, 2001.

[32] V. Danciu, D. Kranzlmüller, F. Liu, J. Watzl, M. Ahrens, and P. Kerestey. An Automated Approach for Fault Recovery Planning in Science Clouds, 2010.

[33] V. Danciu, D. Kranzlmüller, M. Schiffers, J. Watzl, and N. gentschen Felde. Der Cloud-Broker: dynamische Orchestrierung von Cloud-Diensten zu Smart Mobile Apps. In *Smart Mobile Apps*. Springer, Berlin, Heidelberg, 2011.

[34] E. Doering. *Handbuch der Münz- Wechsel- Mass- und Gewichtskunde oder Erklärung der Wechsel- Geld- und Staatspapiere-Kurszettel, der Wechsel-Usancen, Masse und Gewichte aller Länder und Handelsplätze, nebst der allgemeinen deutschen Wechselordnung*. Verlag

von J. Hölscher, 1854.

[35] I. Drago, M. Mellia, M.M. Munafò, A. Sperotto, R. Sadre, and A. Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. 2012.

[36] M. Adelson et al. Understanding Standard & Poor's Rating Definitions, 2009.

[37] T. Metsch et al. Open Cloud Computing Interface - Core (GFD-P-R.183), 2011.

[38] T. Metsch et al. Open Cloud Computing Interface - Infrastructure (GFD-P-R.184), 2011.

[39] T. Metsch et al. Open Cloud Computing Interface - RESTful HTTP Rendering (GFD-P-R.185), 2011.

[40] The London Stock Exchange. Rules of the London Stock Exchange, 2012.

[41] A. Eydeland and K. Wolyniec. *Energy and Power Risk Management: New Developments in Modeling, Pricing and Hedging*. Wiley, 2002.

[42] S.P. Findlan and J.R. Watzl. c^2EX Compute Commodities Exchange, US Patent 13/864,880, 2013.

[43] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a new Computing Infrastructure*. Morgan Kaufmann, 2003.

[44] P.M. Garber. Tulipmania. *Journal of Political Economy*, pages 535–560, 1989.

[45] C. Garner. *A Trader's First Book on Commodities*. Financial Times Prentice Hall, ISBN 978-0137015450, 2010.

[46] H. Geman. *Commodities and Commodity Derivatives: Modelling and Pricing for Agriculturals, Metals and Energy*. Wiley, 2005.

[47] V. Gruhn, D. Pieper, and C. Röttgers. *MDA: Effektives Software-Engineering Mit UML2*

und Eclipse. Springer Verlag, 2006.

[48] P. Hammer, S. Nagl, and J. Appoldt. Public Key Infrastructure. 2000.

[49] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999.

[50] C. Hill. Regulating the Rating Agencies. In *American Law & Economics Association Annual Meetings*, page 1. bepress, 2004.

[51] S. Hughes and R. Trope. Red skies in the morning-professional ethics at the dawn of cloud computing. *38 William Mitchell Law Review 111 (2011)*, 2011.

[52] M. Jeckle, C. Rupp, J. Hahn, B. Zengler, and S. Queins. *UML 2 Glasklar*, volume 1. Hanser, 2004.

[53] J. Jonczy, M. Wüthrich, and R. Haenni. A Probabilistic Trust Model for GnuPG. In *23C3, 23rd Chaos Communication Congress, Berlin, Germany*, pages 61–66, 2006.

[54] E.O. Joslin and RF Chairman-Hitti. Evaluation and Peformance of Computers: Application Benchmarks: The Key to Meaningful Computer Evaluations. In *Proceedings of the 1965 20th national conference*, pages 27–37. ACM, 1965.

[55] R. Kamat and S.S. Oren. Two-Settlement Systems for Electricity Markets under Network Uncertainty and Market Power. *Journal of Regulatory Economics*, 25(1):5–37, 2004.

[56] P.E. Kloeden, E. Platen, and H. Schurz. Stochastic Differential Equations. *Numerical Solution of SDE Through Computer Experiments*, pages 63–90, 1994.

[57] A. Linden and J. Fenn. Understanding Gartner's Hype Cycles, 2003.

[58] T. Lindinger. *Optimierung des Wirkungsgrades virtueller Infrastrukturen*. PhD thesis, Dissertation, Ludwig–Maximilians–Universität München, 2010.

[59] U. Maurer. Modelling a Public-Key Infrastructure. In *Computer Security-ESORICS 96*, pages 325–350. Springer, 1996.

[60] P. Mell and T. Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 53(6):50, 2009.

[61] R. Michie. *The Global Securities Market: A history*. Oxford University Press, 2007.

[62] G. Münzl, B. Przywara, M. Reti, J. Schäfer, K. Sondermann, M. Weber, and A. Wilker. Cloud Computing-Evolution in der Technik, Revolution im Business. *Berlin: BITKOM*, 2009.

[63] J. Nielsen. Nielsen's Law of Internet Bandwidth (April 1988). `http://www.useit.com/alertbox/980405.html`, 2010.

[64] International Chamber of Commerce. Incoterms ® 2010, 2010.

[65] F. Partnoy. How and why Credit Rating Agencies are not like Other Gatekeepers. *Financial Gatekeepers: Can they protect investors*, 59:88, 2006.

[66] S. Rajan and A. Jairath. Cloud Computing: The Fifth Generation of Computing. In *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*, pages 665–667. IEEE, 2011.

[67] O. Regev and N. Nisan. The POPCORN Market. Online Markets for Computational Resources. *Decision Support Systems*, 28(1):177–189, 2000.

[68] S. Ried, H. Kisker, and P. Matzke. The evolution of cloud computing markets. *Evolution*,

2010.

[69] M. Risch, I. Brandic, and J. Altmann. Using SLA Mapping to Increase Market Liquidity. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 238–247. Springer, 2010.

[70] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[71] R. Rose. Survey of system virtualization techniques. *Collections*, 2004.

[72] J.W. Ross and G. Westerman. Preparing for Utility Computing: The Role of IT Architecture and Relationship Management. *IBM systems journal*, 43(1):5–19, 2004.

[73] W.F. Treacy and M. Carey. Credit Risk Rating Systems at Large US Banks. *Journal of Banking & Finance*, 24(1):167–201, 2000.

[74] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[75] L. Wang, J. Tao, M. Kunze, A.C. Castellanos, D. Kramer, and W. Karl. Scientific Cloud Computing: Early Definition and Experience. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 825–830. IEEE, 2008.

[76] J. Watzl, N. gentschen Felde, and D. Kranzlmüller. Analyzing the Applicability of Airline Booking Systems for Cloud Computing Offerings. *Data Driven E-Science: Use Cases and Successful Applications of Distributed Computing Infrastructures (ISGC 2010)*, page 331,

2011.

[77] E. Weber. A Short History of Derivative Security Markets. 2008.

[78] M. Zeleny. *Multiple Criteria Decision Making*, volume 25. McGraw-Hill New York, 1982.

[79] Z. Zhang, X.M. Wang, and Y.X. Wang. A P2P Global Trust Model Based on Recommendation. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 7, pages 3975–3980. IEEE, 2005.