University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Distributed Auction System

Simon Ruesch
Rickenbach, Switzerland
Student ID: 10-708-139

Supervisor: Christos Tsiaras, Patrick Poullie
Date of Submission: March 31, 2014

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

Die Anrufzustellung in Mobilfunknetzwerken wird weithin als Monopol angesehen [3], weil nur der Mobilfunkanbieter des Anrufempfängers den Anruf zustellen kann. Das benutzerzentrierte und auktionsbasierte Abrechnungssystem (AbaCUS) wird als Lösung für dieses Monopol vorgeschlagen [29]. Anrufer können mit AbaCUS beeinflussen, welcher Mobilfunkanbieter den Anruf zustellen darf, indem sie eine gewünschte Anrufqualität fordern. Mobilfunkanbieter nehmen an einer Auktion teil, in der sie Gebote für die Anrufqualität abgeben. Der Mobilfunkanbieter, der die besten Konditionen für die gewünschte Anrufqualität anbietet, darf schlussendlich den Anruf zustellen.

Die vorliegende Bachelorarbeit basiert auf dem ursprünglichen AbaCUS-Ansatz [29] und folgt der Arbeit von [11], in der ein Mechanismus entwickelt wurde, der es erlaubt ondemand in das Netz eines anderen Mobilfunkanbieters zu wechseln. Der Mechanismus in [11] setzt die Existenz einer sogenannten Auktionsautorität (Au^2) voraus, welche als zentraler Server dafür verantwortlich ist, Gebote von Mobilfunkanbietern entgegen zu nehmen, Anrufanfragen zu behandeln und Anrufempfänger dazu aufzufordern, zum Netz eines anderen Mobilfunkanbieters zu wechseln. In dieser Arbeit wird eine prototypische Implementierung einer Au^2 beschrieben. Der Prototyp implementiert die AbaCUS-Auktionsregeln, ermöglicht es Mobilfunkanbietern und Anrufern mit der Au^2 zu kommunizieren und erlaubt es, Server in einem Auktionsraum zu verteilen, wobei jeder Server für seine eigene geographische Fläche verantwortlich ist. Nach der Beendigung der Implementierung wurde der Prototyp hinsichtlich der Fairness der Auktion und hinsichtlich der Skalierbarkeit bezüglich gleichzeitiger Anrufanfragen und Mobilfunkanbietergeboten evaluiert.

Die Evaluation hat gezeigt, dass der Prototyp der Au^2 einen fairen Auktionsmechanismus implementiert, welcher es kompetitiven Mobilfunkanbietern erlaubt, gegen weniger kompetitive Mobilfunkanbieter zu gewinnen. Zudem macht es der Auktionsmechanismus möglich, dass Mobilfunkanbieter die zu einem späteren Zeitpunkt kompetitiver werden, gegen etablierte Mobilfunkanbieter bestehen können. Die Evaluation hinsichtlich der Skalierbarkeit war nicht konklusiv, da sie aus Zeitgründen nicht in einer produktiven Umgebung durchgeführt wurde. Dennoch hat sie Hinweise darauf gegeben, dass HTTP-Nachrichten einen möglichen Flaschenhals für das System darstellen.

Für die zukünftige Forschung wird vorgeschlagen, eine Untersuchung über die beste Bietstrategie für Mobilfunkanbieter vor dem Hintergrund des AbaCUS-Auktionsmechanismus durchzuführen. Zudem wird empfohlen, die Skalierbarkeit in einer produktiven Umgebung zu evaluieren und schliesslich den entwickelten Prototyp mit der Arbeit in [11] zu integrieren.

Abstract

Call termination for mobile network users is widely considered a monopoly [3], because only the MNO to which the callee is subscribed, can terminate the call. The Auction-based Charging and User-centric System (AbaCUS) has been proposed to challenge this monopoly [29]. In AbaCUS, callers can influence which Mobile Network Operator (MNO) terminates their calls by setting a preference for a desired Quality of Service (QoS). On the other hand, MNOs compete in an auction among themselves to determine the MNO which offers the best conditions for the given QoS. That MNO is then allowed to terminate the call.

This bachelor thesis is based on the initial proposal for AbaCUS [29] and is a follow-up to [11] where an on-demand MNO switching mechanism was developed. The mechanism described in [11] depends on the existence of an Auction Authority (Au^2) which acts as a central server that allows MNOs to submit bids, allows callers to send service requests and allows callees to find out to which MNO they have to switch. This thesis describes a prototypical Au^2 implementation. Developing the prototype involved implementing the rules of the AbaCUS auction, providing a means for MNOs and callers to communicate with the Au^2 and providing a mechanism to distribute server nodes on an auction space, each covering a dedicated geographical area. Upon finishing the implementation, the solution was evaluated with regards to the fairness of the auctions and also with regards to the scalability concerning simultaneous service requests from callers and bid submission from MNOs.

The evaluation has shown that the Au^2 prototype provides a fair auction mechanism that allows competitive MNOs to win more often than less competitive ones and provides a means for MNOs that become more competitive at a later time to catch up. The scalability evaluation was not conclusive since it has not been performed in a production environment due to time constraints. Nevertheless, it identified HTTP messages as a possible bottleneck for the system.

For future work, an investigation into the best bidding strategy for MNOs in front of the background of the AbaCUS auction mechanism is proposed. Additionally, evaluating the scalability of the prototype in a production setting is advised, as well as the integration of the prototype with the work done in [11].

Acknowledgments

I would like to thank Prof. Dr. Burkhard Stiller and the Communication Systems Group for providing me with the opportunity to write my bachelor thesis on the topic of a state-of-the-art research project.

I would also like to thank Christos Tsiaras for his excellent support and supervision. Furthermore, I would like to thank Samuel Liniger for his inputs concerning the parts about his thesis. Last, but not least, I would like to thank Samuel Liniger and Bruno Ruesch for proofreading this thesis.

Contents

Chapter 1

Introduction

Call termination for mobile network users is widely considered a monopoly [3], because only the Mobile Network Operator (MNO) to which the callee is subscribed, can terminate the call. However, with the rising prevalence of smartphones [16], which have multiband capabilities and allow mobile network users to switch between different networks, the users are not physically locked into accepting only calls terminated by one exclusive MNO. In front of this background, the Auction-based Charging and User-centric System (AbaCUS) has been proposed to challenge the call termination monopoly [29]. In AbaCUS, callers can influence which MNO terminates their calls by setting a preference for a desired Quality of Service (QoS). On the other hand, MNOs compete in an auction among themselves to determine the MNO which provides the best conditions for the given QoS. That MNO is then allowed to terminate the call.

1.1 Motivation

Breaking the call termination monopoly with AbaCUS has benefits for both end-users and MNOs alike [29]. With AbaCUS, MNOs are able to collect Mobile Termination Rates (MTR) from users that aren't subscribed to them. Additionally, AbaCUS offers MNOs the possibility to provide premium services for a premium charge. Furthermore, AbaCUS gives MNOs the opportunity to make their service more attractive by lowering the MTR on-demand, *e.g.*, in case of low network load. On the opposite side, AbaCUS provides callers with a high amount of flexibility with regards to their choice of call termination fees and desired QoS. If they require high call establishment priority, they can aquire it by paying a higher MTR. They also benefit from a bigger selection of premium services from all the available MNOs.

1.2 Description of Work

This bachelor thesis is based on the initial proposal for AbaCUS [29] and is a follow-up to [11] where an on-demand MNO switching mechanism was developed. The mechanism

described in [11] depends on the existence of an Auction Authority (Au^2) which acts as a central server that allows MNOs to submit bids, allows callers to send service requests and allows callees to find out to which MNO they have to switch. This thesis describes a prototypical Au^2 implementation. Implementing the protoype involved implementing the rules of the AbaCUS auction, providing a means for MNOs and callers to communicate with the Au^2 and providing a mechanism to distribute server nodes on an auction space, each covering a dedicated geographical area. Upon finishing the implementation, the solution was evaluated with regards to the fairness of the auctions and also with regards to the scalability concerning simultaneous service requests from callers and bid submission from MNOs.

1.3 Thesis Outline

In chapter 2, related topics to this thesis are covered; especially the groundworks laid in [29] and [11] are discussed in more detail. Also, a short introduction into the technology used in the implementation is given. Chapter 3 describes the design and implementation of the prototype regarding the AbaCUS auction implementation, the AbaCUS auction space, the AbaCUS node tree, and the AbaCUS server infrastructure. Additionally, the interface for communication with the Au^2 is explained. Chapter 4 discusses the evaluation that has been conducted concerning the auction fairness and the scalability of the prototype system. Finally, in chapter 5, a conclusion on the work is drawn and possible future works are laid out.

Chapter 2

Related Work

This thesis is based on (a), the original proposal of AbaCUS [29], and (b), the on-demand MNO selection mechanism developed in [11]. In this chapter, an explanation of the most important aspects of those two works is given, as well as a short overview of the technology that was used in the implementation of the Au^2 prototype.

2.1 AbaCUS

As was mentioned before, the goal of AbaCUS [29] is to abolish the call termination monopoly held by MNOs. There are four main actors in AbaCUS, (a) the mobile caller who wants to make a call, (b) the callee who receives that call, (c) MNOs which provide the call termination service and (d) the Auction Authority (Au^2), which acts as a central server and is responsible for handling the communication between all the main actors. In AbaCUS, the caller has the possibility to choose a desired Quality of Service (QoS) for the call. The QoS consists of two aspects, namely the sound quality of the call and the call establishment priority, *i.e.*, how fast the call is transmitted to the callee. A combination of a certain degree of sound quality and a certain degree of call establishment priority is termed a Quality of Service Class (QoS-C). In AbaCUS, those QoS-C are ranked in desirability by the operator of AbaCUS and offered to the caller to choose from. On the other hand, a Mobile Termination Rate (MTR) is the rate that a MNO wants to collect by offering the call termination service. The MTR can be divided into two parts: firstly, the initial fee for call establishment, and secondly, the fee for the duration of the call. Those two components can be combined into a discrete Termination Rate Class (TeR-C). The Au^2 offers MNOs a platform to compete in auctions among themselves by submitting bids of a TeR-C for a certain QoS-C. That means that MNOs declare which TeR-C they are willing to accept for a given QoS-C. The winner of an AbaCUS auction is determined by selecting the MNO with the lowest TeR-C bid for a given QoS-C and that MNO is then allowed to terminate all the calls of that QoS-C in the respective AbaCUS auction area. The AbaCUS auction area is the geographic location where the auctions take place. It is a small part of the AbaCUS auction space which itself encompasses the area of an entire country. The AbaCUS auction space is split into smaller AbaCUS auction areas

to evenly distribute the number of service requests and in turn the network load to the Au^2 among several AbaCUS servers. Each AbaCUS server is assigned an auction area and is responsible for accepting MNO bids and call requests only from users located in that auction area. The AbaCUS server infrastructure with distributed AbaCUS servers is termed AbaCUS Node Grid. One further responsibility that an AbaCUS server has is to send a request to the callee to switch to the network of the MNO who has won the auction, such that that MNO can terminate the call.

2.2 On-demand MNO Selection Mechanism

AbaCUS depends on a mechanism for callees to switch to the mobile communication network of a different MNO on-demand. For the switching mechanism to be of any practical use, the end-users must not experience any impact on the call prodecure performance. The work in [11][27][28] provides a prototype application for such a mechanism for Android devices, called AbaCUSApp. A first draft for a message protocol between callers, callees and the Au^2 has also been developed. To test the prototype switching mechanism, a mock Au^2 has been developed which does not contain any auction logic and only ever returns one default MNO to switch to. AbaCUSApp allows callers to send switch request to the mock Au^2, which then notifies the AbaCUSApp of the callee via Google Cloud Messaging (GCM) [6] that a switch is necessary. That switch is then performed seemlessly in the background. The thesis at hand provides a more advanced Au^2 prototype which contains an auction engine and allows MNOs to submit bids. It also accepts JSON messages in the message format developed in [11] but only a dummy implementation of caller message handling was provided because further integration with the work in [11] and especially with GCM would have gone beyond the scope of the thesis at hand.

2.3 Technology Used

In this section, an overview over the main technology that was used in the development of the Au^2 is given.

2.3.1 Java Servlet

Java Servlets [10] are a component of the web tier of Java Enterprise Edition (Java EE) [33]. They provide a simple tool to develop web applications that can then be hosted on Java Servlet Containers [30]. In the prototype, the `HttpServlet` [8] class was extended and used as a basis for the main interface to the Au^2 server. The `HttpServlet` base class offers simple methods to receive HTTP [25] requests and send responses to the sender. Java Servlets are stateless; a new Java Servlet instance is constructed for each request that reaches the Servlet Container. Therefore, the internal state of a Java Servlet cannot be stored in instance variables. Instead, the `HttpSession` [9] is used to store data for one

user, whereas the `ServletContext` [22] can be used to store data that is relevant to all users of a certain Java Servlet.

2.3.2 Tomcat

Tomcat [1] is a non-commercial web container for Java Servlets developed by Apache. In the prototype implementation, it is used as a server to host the web application that represents the Au^2 prototype. More specifically, the embedded version of Tomcat was applied, which allows the Java developer to start a Tomcat instance from within a Java application. This feature of Tomcat was used, because it allows for multiple Tomcat instances to be generated which provided a means to simulate the AbaCUS node grid on a single machine.

2.3.3 Spring Framework

The Spring Framework [23] was used as a tool for dependency injection [4]. This allowed the prototype to use different class configurations for development and testing. Many components of the prototype are designed such that an alternative implementation than the one provided in the prototype can be easily injected.

2.3.4 Maven

Maven [31] is a dependency management and build automation tool developed by Apache. In the prototype, mostly the dependency management feature was used to keep track of external libraries like Tomcat and Spring. The build automation tool was not used for the final executable, because there were conflicts with embedded Tomcat which could not be resolved in a useful time.

Chapter 3

Design

In this chapter, the design of the prototype Au^2 implementation is described. However, two key elements of the AbaCUS auction are explained in more detail first: (a) the AbaCUS auction space, which models the physical location of the end-users, and (b) the AbaCUS auction that is used to select the appropriate MNO for call termination.

3.1 AbaCUS Auction Space

The AbaCUS auction space defines the geographical area where end-users place voice calls and therefore where AbaCUS auctions take place. Typically, the base auction space is equal to the area of the country where the AbaCUS system is deployed. However, the auction space can be subdivided into smaller sub auction areas, each of which contains an AbaCUS server that is responsible for the auctions that take place for that area. The size of those smaller auction areas typically corresponds to the size of an underlying Global System for Mobile Communication (GSM) cell [18]. Subdividing the auction space has multiple benefits: Firstly, it provides a means to align the size of the auction space to the actual service demands. This allows a metropolitan area, where the density of end-users is higher than in a rural area, to be covered by multiple auction areas resulting in less traffic for the servers involved. Secondly, this allows MNOs to specialize in certain auction areas which are attractive for them. While MNOs have to handle service requests in all auction areas if they are selected by the respective AbaCUS auction, they can simply provide uncompetitive bids in areas which are not lucrative for them. Without any loss of generality and for simple identification of neighbor areas, in the prototype Au^2 implementation, the base auction space is modelled as a square. The implementation allows the base auction space to be split on demand into four quarters or two halves, split either vertically or horizontally. Each of those resulting sub auction areas can itself be split in any of the three aforementioned ways. However, to keep the system simple, the sub auction spaces that resulted from a split in half can only be divided such that the resulting auction spaces are squares. Figure 3.1 shows the possible split options for an auction space, while Figure 3.2 shows a possible auction space after a few splits. In the AbaCUS system, the notion of neighborhood is important for the tie-breaking between

MNOs in an auction. A neighboring AbaCUS auction area is defined as an auction area that shares either an edge or a corner with the auction area of interest. The number of neighboring auction areas can vary widely. For example, an AbaCUS auction area may have zero neighbors. This is the case when the base auction space is not split. Or the auction area may have up to an infinite number of neighboring areas, if all the surrounding areas have been split many times.

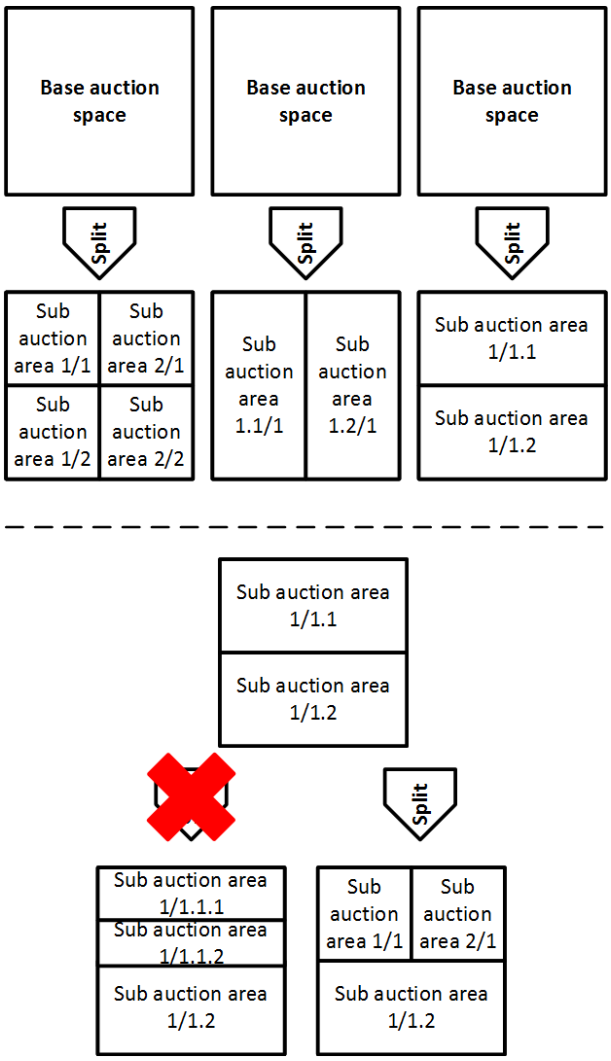

Figure 3.1: The possible split options for AbaCUS auction areas

3.2 AbaCUS Auction

The AbaCUS auction is performed to determine the MNO which is allowed to terminate the calls for a certain QoS-C. These auctions are held in a reverse Vickrey's auction scheme as described in [29]. In each bidding round of a reverse Vickrey's auction, each auction participant submits a hidden bid to the Au^2. The winner of the auction is the participant with the lowest bid, however in a Vickrey's auction, the winner is only accountable for paying the second lowest bid. It has been shown that the Vickrey's auction incentivizes

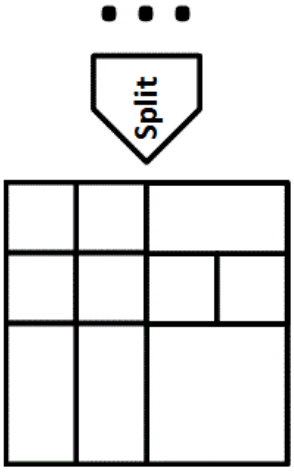

Figure 3.2: A protoypical AbaCUS auction space after a number of splits

the participants to bid at their real valuation of the good [2]. In the following section, the AbaCUS auction is explained.

The AbaCUS auction extends the Vickrey's auction with additional rules concerning the form that a bid takes and the validity of a bid. First of all, MNOs compete in auctions over a certain predefined number of QoS-Cs and may submit a bid containing a TeR-C value for each of the QoS-Cs. The TeR-C value signifys the MTR that the MNO collects for the specified QoS. In the standard configuration of the prototype Au^2 implementation, both QoS and MTR have five classes with values ranging from 0 to 4, although different numbers of classes can easily be applied. Additionally, the number of classes for QoS and MTR do not have to be equal. In an example case, having more TeR-Cs than QoS-Cs enables MNOs to make more fine grained bids, which allows the bids to be closer to the real valuation and allows for more varied bidding strategies. On the other hand, having more QoS-Cs than TeR-Cs allows end-users to specify their preferences more clearly.

There are three key rules concerning the validity of a bid. Firstly, in each auction round, a winner will be determined for each QoS-C. This means that each MNO has to submit a TeR-C for each QoS-C, or the MNO cannot participate in the entire AbaCUS auction. Secondly, the TeR-C value for a certain QoS-C must be higher than or equal to the value for the QoS-C below it. For example, if the TeR-C value for the first QoS-C is 2, then the value for the second QoS-C must be 2 or higher. Figure 3.3 shows an example of a valid and an invalid bid. The second bid is invalid because the TeR-C for the third QoS-C should have been at least 2 because the TeR-C for the second QoS-C was 2 as well. Thirdly, there are two additional rules concerning the MTR that the winning MNO collects at the end. Firstly, when a caller requests the service of the Au^2, the caller has to provide the desired QoS-C and also a TeR-C threshold. The TeR-C threshold is the maximum TeR-C value that the caller accepts to pay for the call. For example, this threshold could be TeR-C 3. If the winning TeR-C for the desired QoS-C is 3 or lower, the winning MNO is selected to terminate the call. If it is 4 or higher, the caller rejects the offer and uses the original MNO without the callee switching to a different MNO. This encourages MNOs to submit competitive bids to collect the MTR. Secondly, MNOs are only held accountable for providing the termination service for the second lowest TeR-C,

with one caveat. If the second lowest TeR-C is more than one TeR-C higher than the winning value, then the winning value is chosen. This is a measure to discourage MNOs from submitting fraudulent bids. Without this rule, losing MNOs can submit bad bids which cause the second best bid to be above the caller's threshold, leading to the rejection of the winning MNO.

Valid bid	
QoS-C	TeR-C
0	1
1	1
2	2
3	3
4	4

Invalid bid	
QoS-C	TeR-C
0	1
1	2
2	1!
3	3
4	4

Figure 3.3: Example of a valid and an invalid bid

3.2.1 Tie-breaking

Since the MNOs bid in discrete and limited values of TeR-C, ties are likely to happen. The following tie-breaking mechanism was chosen to allow for diverse bidding behavior by the MNOs, meaning that MNOs will not be dominated by MNOs which ran a more competitive strategy at an earlier time [29]. If there is a tie between two or more MNOs for a certain QoS-C, the winner will be determined in three steps. Firstly, if any of the tied MNOs has won more than a predefined maximum amount of the last auctions for that QoS-C, that MNO will be taken out of contention. Secondly, the winner of the most auctions in the neighborhood of the respective AbaCUS node for the QoS-C will be determined. If one MNO is the sole winner of most auctions for the QoS-C in the neighborhood, that MNO will be chosen as the winner of the tie. If multiple MNOs have the same amount of wins in the neighborhood, a random MNO among them will be chosen as winner. This tie-breaking scheme was chosen to incentivize MNOs to be competitive over the entire auction space [29].

3.3 AbaCUS Architecture

Figure 3.4 shows the main components of AbaCUS. The Au^2 node grid is in the center of the entire system and supervises all interaction between the actors in the system. Callers are end-users which want to make a call over AbaCUS, callees are the ones that receive the call and MNOs submit bids to the the Au^2 node grid for the right to terminate calls. A Web-GUI is used to monitor the current state of the the Au^2 node grid. Google Cloud Messaging (GCM) is a service that was used by [11] to send messages from the Au^2 node grid to the callers and callees. The Au^2 node grid is a collection of servers where each is responsible for handling a specific part of the AbaCUS auction space. The Au^2 node grid itself consists of the AbaCUS node tree and the AbaCUS auction engine, which are further explained in the following sections. This thesis provides a prototypical implementation for

the Au² node grid, the caller, the MNO and the Web-GUI. A prototype for the callee has not been developed in this thesis, because anything more than a simple message receiver would have required the integration with the work done in [11] which would have gone beyond the scope of this thesis. However, this integration is an interesting topic for future research.

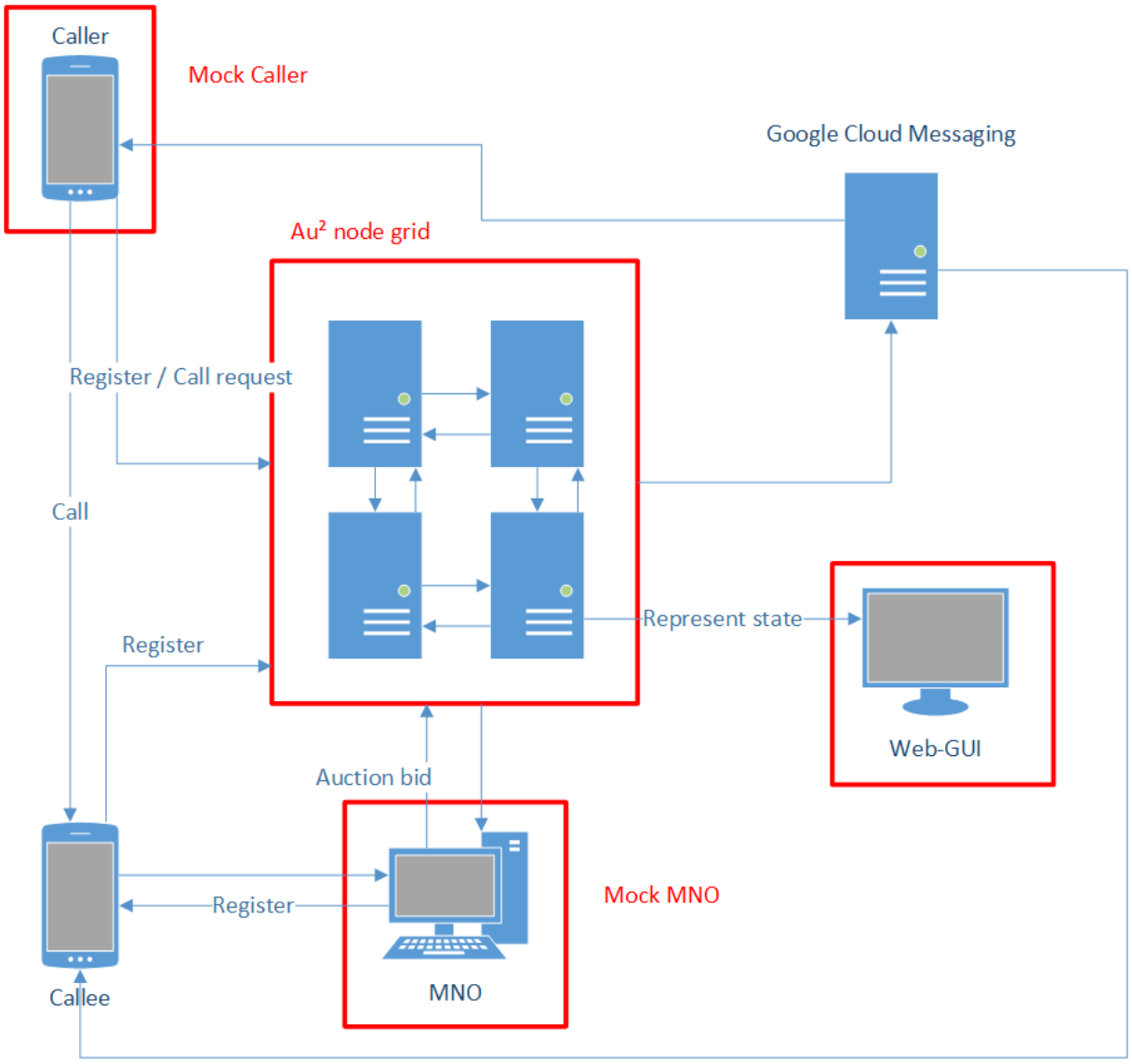

Figure 3.4: High level view of the AbaCUS architecture

3.4 AbaCUS Node Tree

To implement the AbaCUS prototype, an appropriate data structure to represent the AbaCUS auction space had to be selected. The data structure had to be efficient with regards to splitting an auction space in a manner described above, finding auction areas by various criteria, finding neighbors of an auction area and communication between the servers for each space. Neither literary research for a data structure that allows the search

for neighbors in an arbitrary grid (*e.g.*, a 5x5 cell grid) nor inventing such a data structure have been a success. Allowing the split methods described above leads to a very complex and unpredictable neighborhood situation after only a few splits. Basic principles like finding the distance between centers of two areas or finding edges between two areas cannot capture this situation fully. Also, they are computationally complex, because to find neighbors, each node would have to test all the other nodes for the aforementioned properties. Those problems have led to the restriction of the requirements on the grid by only allowing square grids with a multiple of 2 for the rows and columns. Researching existing literature has led to the discovery of the quadtree which has been determined an appropriate solution.

3.4.1 Quadtree

The quadtree is a tree structure which allows nodes to be split into four children [17]. The main advantages of the quadtree are that splitting tree nodes is effortless, searching nodes is quick and a neighbor finding algorithm has already been publicized [21]. One disadvantage about the quadtree is that it only reasonably supports splits of tree nodes into four quarters. A tree based data structure which allows splits of nodes into less than four nodes would clearly be a better solution to capture the split methods described above with nodes split only in half. An alternative version of a quadtree which allows nodes to be split into less than four nodes has been implemented and experimented with, but no neighbor finding algorithms for this case were found in the literature. Devising a new neighbor finding algorithm for this case has not been a success, either, so this approach was abandoned. Therefore, the a slightly modified version of the quadtree termed AbaCUS node tree is used in the prototype to model the auction space of AbaCUS.

Properties of the AbaCUS Node Tree

Nodes in the AbaCUS node tree represent servers sitting in auction areas of the AbaCUS auction space, which means that each node is responsible for handling service requests and MNO bids that happen in that area. For that reason, an auction area is assigned to them when they are generated. Additionally, each node can be uniquelly identified by an id. The following list gives an overview over the most important attributes needed for tree traversal algorithms that a node in the AbaCUS node tree stores:

- `id`: the unique identifier of a node.
- `auctionArea`: The auction area for which the node is responsible for.
- `parent`: the node that was split to generate this node.
- `northWestChild`: the child node that is responsible for the north west quadrant of the auction area of this node.
- `northEastChild`: the child node that is responsible for the north east quadrant of the auction area of this node.

- `southWestChild`: the child node that is responsible for the south west quadrant of the auction area of this node.
- `southEastChild`: the child node that is responsible for the south east quadrant of the auction area of this node.
- `isSplit`: clarifies if this node has been split, *i.e.*, has child nodes, or not.

For networking purposes, the node also stores a URL which is later used to reach the server that is represented by that node.

As mentioned above, a quadtree can only ever be split into exactly four child nodes. This also applies to the AbaCUS node tree. When a node is split, its children are each assigned a quarter of the auction area of the original node. For example, a parent node might be responsible for an auction area with $x \in \{0, 100\}$ and $y \in \{0, 100\}$, where x is the horizontal coordinate and y is the vertical coordinate. As a consequence, the `northWestChild` is responsible for the auction area $x \in \{0, 50\}$ and $y \in \{0, 50\}$. Figure 3.5 shows an example of a split auction space with information about the area for which each node is responsible and the corresponding node tree. For simplicity's sake, the coordinate system of the AbaCUS node tree of the prototype implementation uses integer numbers between 0 and a customizable maximum value. Of course, in a real world scenario, a geopositional coordinate system like WGS84 [32] would be used. Another facet of the quadtree is that the tree does not necessarily have to be balanced, so the depths of the leaf nodes are not necessarily equal to each other. Three main algorithms for tree traversal have been used in the prototype, (a) finding nodes by their id, (b) finding nodes by their auction area, and (c) finding neighbors of a node.

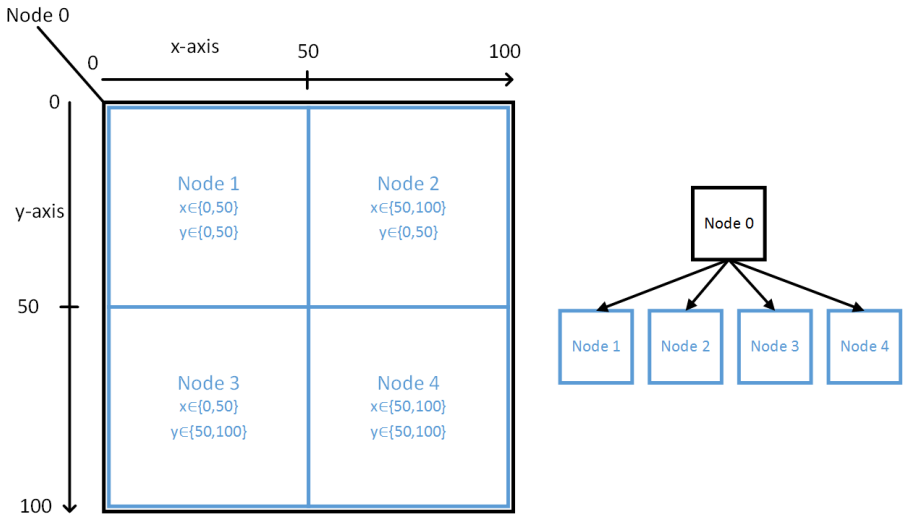

Figure 3.5: Example auction space with corresponding node tree

Finding Nodes by Their Id

The algorithm used for finding nodes by their id is recursive and starts at the root node. A node for which this algorithm is applied first checks if the node itself is the node with

the correct id. If it is, search is finished and the correct node is returned. If not, the node applies the algorithm on its children. If it either has no children or the children find out that they are not the correct nodes, the original node returns `null`. The complexity of this algorithm is $O(n)$ where n is the number of nodes. This is because in the worst case, each node has to be checked if it has the correct id.

Finding Nodes by Their Auction Area

This algorithm is also implemented in a recursive manner and starts from the root node. The root node first checks which of the auction areas of its child nodes covers the auction area that was supplied as a parameter to the algorithm. If none of them do, `null` is returned. If one of them does, that node itself checks its children for appropriate nodes, and so on. The complexity of this algorithm is $O(h)$ where h is the height of the tree, because at each level of the tree, only one node can have an auction area that covers the auction area from the parameter, so at most one node per level continues the algorithm.

Finding Neighbors of Nodes

The neighbor finding algorithms used for the prototype is formulated in [21]. Two different algorithms are applied, one to find neighbors that are diagonal to the node and one to find neighbors that are either horizontal or vertical neighbors to the original node. For the explanation of the algorithms, the term child type has to be introduced. The child type of a node is defined by the quadrant of the parent node that that node covers. For example, the child type of node x is north west, if it is the `northWestChild` of its parent node. The algorithms always start at the node for which neighbors are to be found. If, say, the eastern neighbor of that node should be found, first, the child type of the original node is determined. If the child type is adjacent to the direction in question (in the example, north east and south east are adjacent to the eastern direction), the search continues with the parent of the node. This process is repeated, until either the root node is reached through a child with an adjacent child type, which means that the original node is at the border of the entire auction area and has no neighbor in that direction. Second possibility is that a parent node is found that was reached by a child node with a child type that is not adjacent to the direction in question. When such a node x is found, the path taken to reach that node, expressed in child types, is mirrored by the direction in question (*e.g.*, north east mirrored by east becomes north west) and traversed in reverse order starting from node x. The node reached in this manner is the neighbor in the direction in question for the original node. This process is shown in Figure 3.6.

Finding a neighbor in a diagonal direction is more complicated. If for example the south eastern neighbor of node x shall be found, the tree has to be ascended starting from node x until a parent node p is reached from a child node with a child type b that is different from the child type of the desired neighbor c, *i.e.*, any child type different from south east. Then, it is determined which direction d both b and c are adjacent to, *i.e.*, north east and south east are both adjacent to east, so d would be east. After that, the neighbor in direction d of node p is searched. If that node q is found, the steps that were taken to

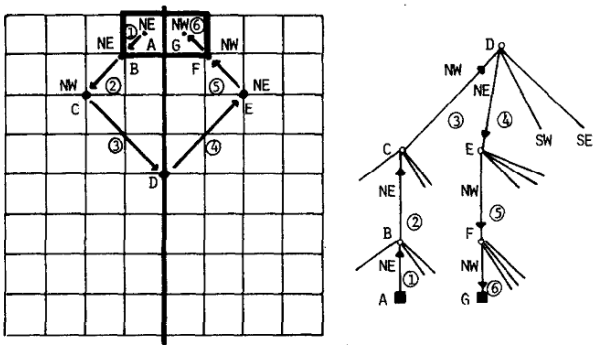

Figure 3.6: Process of finding the eastern neighbor of node A [21]

reach p are rotated by 180 degrees (*e.g.*, south east becomes north west), and traversed in backwards order from node q on until the desired neighbor is found. The neighbor finding for directional neighbors is demonstrated in Figure 3.7. Both neighbor finding algorithms find neighbors of the same size (*i.e.*, they have the same depth) or bigger. Small variations can be applied to those algorithms to also find neighbors that are smaller. Both neighbor finding algorithms have a complexity of $O(2h)$, because in the worst case, the algorithms have to find a neighbor for a node with depth equals to h which means ascending the tree to the root node and back down to a neighbor of the same size. Some functions of the AbaCUS node tree require that all neighbors of a node are found. To do this, the aforementioned algorithms were simply applied in every cardinal direction and for all diagonals.

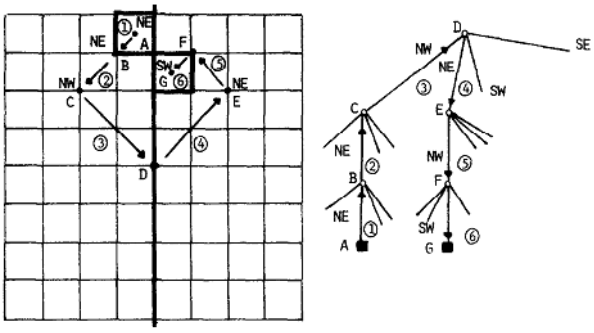

Figure 3.7: Process of finding the south eastern neighbor of node A [21]

3.4.2 Service Nodes and Non-service Nodes

Since nodes in the AbaCUS node tree represent servers which are responsible for handling service requests from callers and conduct auctions between MNOs, they also manage a so-called AbaCUS Auction Engine object. What that object exactly is will be explained in the following section, however, it is important to make a distinction here. There are some nodes in the node tree that hold an AbaCUS auction engine object which are called service nodes, and some that do not which are called non-service nodes. The lifecycle of a node in the AbaCUS node tree is as follows: When a node is first generated, it inherits a copy of the AbaCUS auction engine from its parent, and is intially a service node. If at

some point, the operator of the AbaCUS node grid decides to split that node, it becomes a non-service node and passes its auction engine on to its children. This is done because the child nodes of that node cover the entire auction area of the recently split node. Allowing that parent node to also accept bids and handle caller requests would be redundant and misleading. However, non-service nodes still have the responsibility to forward messages to the correct child node. Those messages can happen if for example a caller was not active for a long time and has not yet received an update on the split. In that case, the newly split node is mistaken for the correct node to handle the caller's request. The split node still accepts the request, forwards it, and informs the caller that a new node is now responsible.

3.4.3 Paired Nodes

In section 3.1 the different split methods were explained, among them the possibility to split a node into either vertical or horizontal halves. However, as it was mentioned before in this section, nodes in a quadtree can only split into four quarters. Because the possibility to split nodes into halves was deemed important, as this provides the operator of the AbaCUS auction space more flexibility in auction area planning, a solution had to be implemented. Although a node is still split into four quarters, the four nodes are assigned as two pairs. Each of those pairs only holds one instance of an AbaCUS auction engine object. Therefore, when a request arrives at either node in a pair, the same auction engine is used to handle that request. This is demonstrated in Figure 3.8. Finding neighbors of paired nodes is done by simply finding all neighbors for each of the nodes, uniting the resulting neighbor lists and removing possible duplicates.

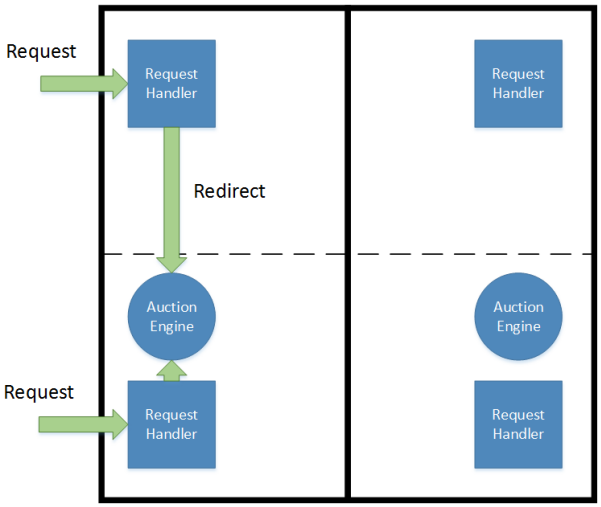

Figure 3.8: Request handling with paired nodes

3.5 AbaCUS Auction Engine

The AbaCUS auction engine is the central module that is responsible for all service requests and all auctions between MNOs inside an AbaCUS auction area. To explain the

workings of the AbaCUS auction engine, a few utility classes that it uses as well as the most important fields of the AbaCUS Auction Engine are introduced. Subsequently, the major components of the AbaCUS Auction Engine are explained.

3.5.1 Utility Classes and Important Fields of the AbaCUS Auction Engine

AbacusServiceClass

The `AbacusServiceClass` is an interface to encapsulate a certain value for a QoS-C or a TeR-C. It also contains information about the maximum numbers of QoS-Cs and TeR-Cs of the implementation in question. In the prototype, this interface is implemented by `StandardQualityOfServiceClass` and `StandardTerminationRateClass`, each of which sets the maximum number of classes to five, internally represented as integers reaching from 0 to 4. For example, a `StandardQualityOfServiceClass` object can be instantiated with the value 3. This represents the fourth QoS-C for purposes of bid submission. The interface is easily implemented to provide the possibility of different numbers of classes. The prototype also contains QoS-C and TeR-C implementations with 3 classes, but those have not been tested thoroughly.

MnoData

A `MnoData` object uniquely identifies a MNO. It contains the name and the id of a given MNO, which are checked to assign bids to the correct MNO.

Bid

A `Bid` object is submitted by MNOs, if they want to take part in an AbaCUS auction. It contains the `MnoData` of the submitting MNO, one `AbacusServiceClass` for QoS-C and TeR-C respectively to identify which numbers of classes were used and a map with a TeR-C for each of the possible QoS-C. This map represents the TeR-C that the MNO is willing to offer for each QoS-C, if the MNO wins the auction.

WinnerRecord

The `WinnerRecord` stores the history of a certain number of AbaCUS auctions that were held in the AbaCUS auction engine. It stores a predefined number of auction rounds, each with the winning MNOs together with the respective winning TeR-C for each QoS-C. Whenever an auction round concludes, the record of this auction is added to the `WinnerRecord` as the most recent entry. At the same time, the oldest entry of the `WinnerRecord` is removed. The `WinnerRecord` is not used to keep a historical record of all auctions that took place, but rather to provide a means of tie-breaking between tied

MNOs, as was described in section 3.2.1. The standard implementation of the `WinnerRecord` used in the prototype offers a history depth of two, which means data of the last two auction rounds is stored in the `WinnerRecord`. But `WinnerRecord` implementations with higher or lower history depth can easily be integrated. Figure 3.9 shows an example of a possible `WinnerRecord`.

WinnerRecord					
Winners of last Auction			Winners of second-to-last Auction		
QoS-C	TeR-C	MNO	QoS-C	TeR-C	MNO
0	1	MNO_1	0	0	MNO_2
1	1	MNO_2	1	1	MNO_2
2	2	MNO_3	2	1	MNO_1
3	3	MNO_3	3	3	MNO_3
4	4	MNO_1	4	4	MNO_1

Figure 3.9: Example of a `WinnerRecord` with history depth 2

mostRecentBids

This field stores the most recent `Bid` objects that each MNO has submitted to the auction engine. However, `mostRecentBids` only stores the last bid that was submitted per MNO, so the previous entry for that MNO is overwritten. The bids saved in `mostRecentBids` are later used to determine a winner for each QoS-C.

abacusNode

This field represents the AbaCUS node tree which contains the AbaCUS auction engine. The auction engine has to store a reference to its own AbaCUS node tree to retrieve the winner records of its neighbors. Those winner record are later used for tie-breaking between the MNOs.

winnerRecord

The AbaCUS auction engine stores its own `WinnerRecord` in an instance field, firstly, for tie-breaking inside the auction area where the auction engine resides. Secondly, the `winnerRecord` is shared with neighbor nodes to tiebreak auctions that happened there.

3.5.2 Components of the AbaCUS Auction Engine

BidValidator

The `BidValidator` is responsible for checking if a submitted `Bid` object is valid according to the AbaCUS auction bid validity rules formulated in section 3.2. First, it is checked if the bid employs the correct `AbacusServiceClass` for both QoS and MTR. Second, it is

verified that the `Bid` has a TeR-C assigned to each QoS-C. The reverse is not necessarily true, not every possible TeR-C has to be used in the bid. Finally, it is checked if each TeR-C for a given QoS-C is at least equal to or higher than the TeR-C for the QoS-C below.

WinnerDeterminator

As the name implies, the `WinnerDeterminator` is used to appoint a winning MNO for each QoS-C. To do this, it uses the field `mostRecentBid` to calculate the lowest TeR-C for each QoS-C among the bids submitted by the MNOs. If two or more MNOs submitted the same TeR-C for a given QoS-C, the `TieBreaker` is applied.

TieBreaker

The `TieBreaker` receives a list of tied MNOs as well as the QoS-C for which the MNOs are tied. This information combined with the `WinnerRecords` of neighbors is used to determine a winner according to the tie-breaking rules describe in section 3.2.1.

Each of the components and utility classes mentioned above is used by the auction engine through their respective interfaces, which means that alternative implementations can be easily injected with Spring as long as the implementation complies with the interface provided.

3.5.3 AbaCUS Auction Procedure

The AbaCUS auction procedure starts whenever a bid from a MNO reaches the AbaCUS node. The AbaCUS node then invokes the method `acceptBid()` of the auction engine with the bid as the argument. Inside the `acceptBid()` methode, the bid is first validated by the `BidValidator`. If it is valid, the bid is entered into the field `mostRecentBid` for the respective MNO and that field is used by the `WinnerDeterminator` to calculate the winner for each QoS-C. If necessary, the `TieBreaker` is also applied. When a unique winner for each QoS-C is found, the auction is finished and `winnerRecord` is updated. The MNO that started the auction procedure also receives a reply from the AbaCUS node with the new winners after the auction if the bid was valid. If it was not valid, the MNO is informed that the bid was not accepted. The mechanism, that the auction starts whenever a bid reaches the AbaCUS node, was implemented in the prototype, because it is simple and allows for quick testing. However, other auction mechanisms can be applied with little effort. One such mechanism could be that the auction engine waits until all MNOs have submitted a bid before the auction engine starts the auction. Another mechanism could be that the auction engine waits for a certain time period before starting a new auction round and considers the most recent bids that were submitted by each MNO respectively.

3.6 AbaCUS Server Infrastructure

In the Au^2 prototype, the server infrastructure is implemented using Java Servlets and with embedded Tomcat as the Web Server/Servlet Container. The reason for this choice was that Java Serlvets provide a simple way to implement both the web application as well as the Web-GUI, whereas embedded Tomcat is a light-weight option to run multiple server instances on a single personal computer. A further reason for choosing this technology is that the work done in [11] already used Java Serlvets and Tomcat, so a few parts of that implementation could be reused in the cases where a combination of the Au^2 prototype with the application from [11] was desired. In the Au^2 prototype, all messages between servers, MNOs and end-users are sent via HTTP. Each server in the Au^2 prototype is realized as its own embedded Tomcat instance running the so-called `AbacusServlet` as its main interface.

3.6.1 AbacusServlet

The `AbacusServlet` extends the default `HttpServlet` implementation provided by Java EE. Since a new Java Servlet instance is generated each time a request with the appropriate URL reaches the web server, the persistent state of the AbaCUS node tree, which also contains the AbaCUS auction engine, is stored in the `ServletContext`. This `ServletContext` is shared among all Java Servlet instances of a certain type, *i.e.*, the `AbacusServlet` in the case of the Au^2 prototype, that run on the same Tomcat instance. Additionally, the `ServletContext` is also the same for every user, be it MNO or caller, that sends a request to that particular Tomcat instance, as opposed to the `HttpSession`. The `AbacusServlet` is the main access point for all requests from MNOs, callers, and also for messages between AbaCUS nodes. In the following section, the bootstrapping process of the AbaCUS node grid is shown. Subsequently, the implemented messages in the prototype are explained.

3.6.2 Bootstrapping the AbaCUS Node Grid

To bootstrap the node grid in the prototype, first a root node has to be created. To do this, a Tomcat instance is started and the `AbacusServlet` is mapped to a predefined URL. The URL mapping is done after a certain pattern: It contains the root URL of the server plus a predefined Servlet name plus the node id of that particular node. This is done to uniquely identify the nodes that run on a Tomcat instance. Then, a new AbaCUS node tree is created together with an AbaCUS auction engine. The URL generated before and the auction engine are encapsulated into the node tree, which is then sent to the root node via a HTTP request. The approach of sending the node tree via a HTTP request was chosen because Tomcat does not allow the initialization of a `ServletContext` with a Java object, as far as is known. Also, as will be be shown later, the method of sending a node tree via HTTP is used while generating child nodes and to update all nodes in the grid. In a real world scenario, those nodes are not stationed at the same location as the root node either, so the updates and bootstrap messages have to be sent over a

network somehow. Appendix A shows a prototypical implementation of such a bootstrap process. The root node is now bootstrapped and can be used by MNOs to send bids to and for callers to send service requests. However, because the goal of AbaCUS is to have a distributed server landscape with many servers, each responsible for its own auction area, the root node can also be split. In the prototype, the `AbacusSplitRequestSender` can be used to send a so-called split request to the root node, containing the node id of the node that is to be split. In the prototype, the split process is done as follows:

1. The root node searches its node tree for the node to split.
2. The root node splits that node inside the data structure. The node tree itself is only responsible for keeping track of the layout of the node grid. Splitting a node inside the node tree does not itself start a new server.
3. The root node then starts a new Tomcat instance for each of the newly created child nodes.
4. The root node creates four copies of its node tree and of its auction engine and sends one of each to each of the child nodes.
5. Finally, the root node sends an update message to each node in the node grid, to inform each node about the new layout of the node grid.

The update message contains the updated node tree which the nodes then insert into their `ServletContext`. Every node in the node grid is always up-to-date on the layout of the entire node grid. This allows nodes to independently search for neighbor nodes or nodes where they have to forwards requests, without consulting the root node. A prototypical implementation of such a node splitting procedure can be found in Appendix B. As a special case, AbaCUS nodes can also be split into halves. To do this, only one Tomcat instance is started for each pair. Although both nodes in a pair handle requests equally, an individual node in a pair can be reached by sending a request to the unique URL of the node.

3.6.3 Messages in the Au^2 Prototype

The messages between MNOs, end-users and servers in AbaCUS are sent via HTTP. Sending a HTTP GET to a Tomcat instance leads to the Web-GUI, which will be explained at a later point. The HTTP POST message is used to interact with the server. In the prototype, the intent of a message is encoded in the Content-Type [24] field of the HTTP header. The server provides a well-known interface, which includes `String` constants for all the possible message types. For example, if a MNO wants to submit a bid to the server, the MNO applies the `String` constant `ACCEPT_BID` to the Content-Type of the request. Any additional required information in the request is written to the body either as a `String` or via an `ObjectStream` that serializes the object. In the example of a bid request, the MNO has to use an `ObjectStream` to write a `Bid` object to the body of the request. To allow integration with the work in [11], the server also accepts messages

encoded with JSON, if the Content-Type of the message is set to `application/json`. In JSON messages, the message intent is encoded in a key-value pair. The response from the server also contains an appropriate Content-Type and, depending on the message, a body that can be read with an `ObjectStream`. The approach of encoding the intent in the Content-Type was chosen because firstly, it was easy to implement, and secondly, because all the actors in AbaCUS are well-defined and the AbaCUS server can enforce a practical interface. It is not desirable that any random application can interact with AbaCUS. Of course, if in the future the AbaCUS message protocol changes, a different method to serialize the objects, *e.g.*, JSON, and a different way to encode the intent can be chosen. The possible messages for each user of AbaCUS, categorized by their `String` constant for the intent, and the possible response from the server, are described in the next section.

3.6.4 Messages from MNOs

The messages sent by MNOs to the AbaCUS server largely concern getting information about the available service nodes, as well as sending AbaCUS auction bids.

- `ACCEPT_GET_SERVICE_NODES`: This message is used by MNOs to find out which service nodes exist in the node grid, *i.e.*, which nodes actually are responsible for conducting auctions. The response from the server contains a list of `AbacusNodeData`, which contain the id, the URL and the auction area of the node. It is advised that this message is sent before each bid, to be informed in case a split has happened in the mean time.
- `ACCEPT_BID`: This message is used by MNOs to send bid requests to the server. The body of this message must contain a `Bid` object. The server responds with either a `BID_SUCCESS` message that contains the new winners after the auction for each QoS-C in its body, a `BID_INVALID` message if the `BidValidator` has decided that the bid was invalid in one way or another, or a `BID_INVALID_NODE`. This message is sent if the queried server is not a service node. The `BID_INVALID_NODE` message also contains a list of the `AbacusNodeData` of the children of the queried node, if they are service nodes.
- `ACCEPT_GET_CURRENT_WINNERS`: This message allows MNOs to query nodes for the current winners in each QoS-C. This information can also be gained by visiting the Web-GUI, but this message offers a programmatical way to find out.

3.6.5 Messages from Callers

The AbaCUS server accepts messages from callers that are compliant with the message format described in [11]. Although the server can receive the four message types `register`, `requestService`, `update` and `okay`, the messages are only accepted and no complex message handling is done. This is the case because anything more than a dummy implementation would have required the complete integration with the work in [11], which

would have gone beyond the scope of the thesis. The only type of message handling that is done by the server for all messages is to check the location of the caller. If the caller is not in the auction area of the server that has received the message, the server will respond with the URL of the correct server. For the `requestService` message, a mechanism to calculate the correct MNO based on the provided QoS-C and the TeR-C threshold in the message is implemented. But since the information about the correct MNO would have to be sent to the callee via GCM in an integrated system, that information is returned to the caller instead for testing purposes.

3.6.6 Messages between Nodes and Split Messages

Messages between AbaCUS nodes largely concern the exchange of information that is needed to conduct the auctions. Additionally, the dedicated application `AbacusSplitRequestSender`, that is responsible for the organization of the AbaCUS node grid can send split requests to the root node.

- `ACCEPT_GET_NEIGHBOR_WINNER_RECORD`: This message is sent by a node to another node to aquire the current `WinnerRecord`. As was mentioned, the winner records of neighbors are used to tiebreak during an auction.
- `ACCEPT_GET_WINNER_FOR_QOS`: This message is used when a service request from a caller reaches a node that is not responsible for the caller. To save the caller time, the incorrect node queries the correct node itself for the correct MNO to terminate the call. As was explained above, the correct MNO is only returned to the caller, as opposed to the callee.
- `ACCEPT_SPLIT_BY_ID`: This message is sent by the `AbacusSplitRequestSender` to the root node to initiate the split of a node into four child nodes. The id of the node to be split is contained in the body of the message.
- `ACCEPT_SPLIT_HORIZONTALLY_BY_ID`: This message is sent by the `AbacusSplitRequestSender` to the root node to initiate the split of a node into two horizontal pairs of child nodes. The id of the node to be split is contained in the body of the message.
- `ACCEPT_SPLIT_VERTICALLY_BY_ID`: This message is sent by the `AbacusSplitRequestSender` to the root node to initiate the split of a node into two vertical pairs of child nodes. The id of the node to be split is contained in the body of the message.

3.6.7 Web-GUI

To visualize the data about an AbaCUS node, a prototype Web-GUI was implemented. It consists of a HTML web page and can be accessed by sending a HTTP GET request to the respective node. Another possibility to view the Web-GUI is to access the URL of the AbaCUS node through a web browser. The Web-GUI is only a rough draft and

does not provide any interactive elements. The development of a better-looking Web-GUI that includes AJAX [5] elements for real-time updates is advised for a future work. The prototype Web-GUI provides the following information about a node:

- the node id
- the node URL
- the node position, *i.e.*, the path that leads from the root node to the node at hand, expressed in child types
- the auction area, that the node covers
- a graphical representation of the entire auction space with the node at hand colored red and the neighbor nodes colored blue
- a list of neighbor nodes with a hyperlink to their respective Web-GUI
- the information if the node has been split
- a list of direct children of the node with a hyperlink to their respective Web-GUI
- a list of all children (including children of children) of the node with a hyperlink to their respective Web-GUI
- the id of the auction engine of the node
- the current winners plus the `WinnerRecord` of the node
- the `WinnerRecords` of all neighbor nodes

Figure 3.10 shows an example for the graphical representation of the auction space, whereas Figure 3.11 shows an example for the local `WinnerRecord` as well as the `WinnerRecords` of two neighbors.

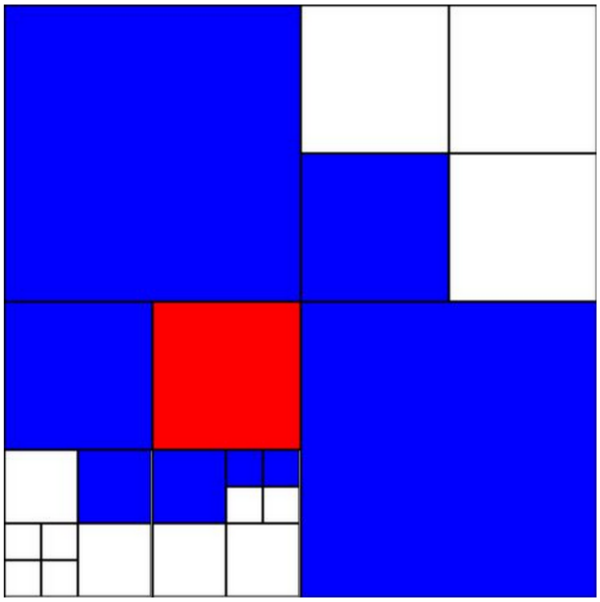

Figure 3.10: Example for a graphical representation of the auction space

Winner Record

Current Winners		
QoS-C	MNO	TeR-C
0	Name:sunrise,Id:1	0
1	Name:sunrise,Id:1	0
2	Name:orange,Id:2	2
3	Name:orange,Id:2	3
4	Name:swisscom,Id:0	4

Table 1		
QoS-C	MNO	TeR-C
0	Name:sunrise,Id:1	0
1	Name:sunrise,Id:1	0
2	Name:orange,Id:2	2
3	Name:orange,Id:2	3
4	Name:swisscom,Id:0	4

Table 2		
QoS-C	MNO	TeR-C
0	Name:sunrise,Id:1	0
1	Name:sunrise,Id:1	0
2	Name:orange,Id:2	2
3	Name:orange,Id:2	3
4	Name:swisscom,Id:0	4

neighbor-winners:

node-id=15

Table 1		
QoS-C	MNO	TeR-C
0	Name:orange,Id:2	1
1	Name:swisscom,Id:0	2
2	Name:sunrise,Id:1	2
3	Name:sunrise,Id:1	2
4	Name:sunrise,Id:1	3

Table 2		
QoS-C	MNO	TeR-C
0	Name:orange,Id:2	1
1	Name:swisscom,Id:0	2
2	Name:orange,Id:2	3
3	Name:orange,Id:2	3
4	Name:orange,Id:2	4

node-id=17

Table 1		
QoS-C	MNO	TeR-C
0	Name:orange,Id:2	1
1	Name:orange,Id:2	1
2	Name:sunrise,Id:1	2
3	Name:sunrise,Id:1	2
4	Name:sunrise,Id:1	2

Table 2		
QoS-C	MNO	TeR-C
0	Name:orange,Id:2	1
1	Name:orange,Id:2	1
2	Name:sunrise,Id:1	2
3	Name:sunrise,Id:1	2
4	Name:sunrise,Id:1	2

Figure 3.11: Example for a graphical representation of the local `WinnerRecord` and the `WinnerRecords` of two neighbors

Chapter 4

Evaluation

To evaluate the developed Au^2 prototype, various tests have been carried out. It is important to know if the winning MNO is determined in a fair manner by the AbaCUS auction engine, *i.e.*, if the MNO who runs the most competitive strategy also wins most often. Furthermore, a test regarding the scalability with regards to the number of simultaneous service request was conducted. A third test regarding the security of the system against the insertion of unauthorized bids was proposed but in the end not carried out, because due to lack of time no security-related issues were considered during the development of the Au^2 prototype. All tests were done on a Hewlett-Packard EliteBook 8540w with an Intel Core i7 1.73 GHz CPU with 8 cores, and 8 Gigabytes of RAM. An evaluation of the prototype system in the CSG Computer Lab was planned but eventually also postponed due to time constraints. The following sections describe the aforementioned tests in more detail regarding (a) the test configuration, (b) the results of of the tests and (c) an interpretation of those results.

4.1 Fairness Evaluation

The evaluation of the fairness of the prototype Au^2 implementation consisted of a variety of self-contained subtests. Because in most countries around the world the largest part of the market share of mobile telecommunication is split between three MNOs [12][13][14][15] or their subsidiaries, a number of three prototypical MNOs, each running in a seperate thread, was used in the tests as well. For each test, 10 test runs were carried out with each MNO submitting a total of a thousand valid random bids. The bids were created by generating a random number between 0 inclusive and the number of TeR-C (in this test's case 5) exclusive for each QoS-C. However, the random number generation mechanism considered the AbaCUS validity rules such that each TeR-C for a QoS-C is at least equal or higher than the TeR-C of the previous QoS-C to guarantee the validity of the bid. The tests used the default implementation of the `Random` class [19] provided by Java as Random Number Generators (RNGs), because it was deemed sufficient for the purpose of the test [7]. To initialize a RNG a so-called seed [20] is used, which in most cases is a large number. The RNG then uses the seed to generate its random numbers. Two

different methods to initialize the RNGs have been used during the test. Firstly, all MNOs used RNGs initialized with the same seed, which leads them to generate the exact same numbers in the same order. For example, calling `nextInt()` on the object of the `Random` class of MNO_1 results in the same number as calling `nextInt()` on the objects of the `Random` class of MNO_2 and MNO_3. Secondly, each RNG was initialized with a random seed, which means there was no correlation between the numbers generated by the three RNGs. The random seed case was used to simulate a more realistic bidding behaviour because in a real world scenario, MNOs are unlikely to submit the exact same bids as their competitors. In a first step, the prototype MNOs only submitted a bid to a single node with no neighboring nodes using either of the two aforementiond initial seeds. In later steps, additional factors influencing the winner determination have been added. The first of those factors was adding a handicap for one MNO which makes the bids of that MNO worse. The second factor was adding neighbor nodes for the node where bids are submitted. Those neighbor nodes contained winner records favoring certain MNOs which leads to a different tie-breaking behaviour of the AbaCUS auction.

4.1.1 Same Seed

In this test, each of the MNOs submitted exactly the same bids in the same order. It is expected that a lot of draws happen in this test, but the tie-breaking algorithm should give each MNO an equal chance of winning in the long run. Table 4.1 shows the respective win ratio of the three MNOs. Each test run consist of 1000 bids per MNO. In a fair system where each MNO submits the same bids as the competitors, all of the MNOs should win exactly the same amount. The evaluation in Table 4.1 gives evidence that this is in fact the case for the prototype Au[2].

	MNO_1	**MNO_2**	**MNO_3**
Test run 1	0.3123	0.3451	0.3427
Test run 2	0.3224	0.3222	0.3555
Test run 3	0.3301	0.3529	0.3170
Test run 4	0.3682	0.3079	0.3240
Test run 5	0.3454	0.3231	0.3315
Test run 6	0.3567	0.3053	0.3380
Test run 7	0.3086	0.3101	0.3812
Test run 8	0.3518	0.3354	0.3128
Test run 9	0.3246	0.3658	0.3097
Test run 10	0.2935	0.3997	0.3069
Average	0.3313	0.3367	0.3319

Table 4.1: Average win ratio of three MNOs using the same seed, 1000 bids per test run and MNO

4.1.2 Different Seed

In this test, even though the RNGs do not produce the same bids, they should generate equally good bids on average. This should be reflected in an equally distributed win ratio between the three MNOs. The test results confirm this claim, as Table 4.2 shows.

	MNO_1	**MNO_2**	**MNO_3**
Test run 1	0.3150	0.3438	0.3411
Test run 2	0.3104	0.3662	0.3234
Test run 3	0.3501	0.3198	0.3302
Test run 4	0.3350	0.3249	0.3401
Test run 5	0.3245	0.3139	0.3616
Test run 6	0.3345	0.3322	0.3333
Test run 7	0.3488	0.3401	0.3111
Test run 8	0.3222	0.3337	0.3441
Test run 9	0.3340	0.3247	0.3412
Test run 10	0.3260	0.3387	0.3353
Average	0.3300	0.3338	0.3362

Table 4.2: Average win ratio of three MNOs using a different seed, 1000 bids per test run and MNO

4.1.3 Same Seed, Handicapped MNO_1

In this test, a handicap for MNO_1 was introduced. A handicap means in this context that a certain number was added to the random result of the RNG, which means that the bid of MNO_1 for each QoS-C is always worse by the height of the handicap compared to the other two MNOs. Table 4.3 shows an example of the handicap being applied. The RNGs of all three MNOs initially generate the same bid. However, for MNO_1 a handicap of 1 is assigned in the example, *i.e.*, 1 is added to each TeR-C. To keep all bids valid, if the initial TeR-C plus the handicap results in a TeR-C that is higher than the maximum TeR-C, the maximum TeR-C is used instead. This test should simulate that MNO_1 is less competitive than the other MNOs and therefore, MNO_1 should also win less often if the auction is fair. It is apparent that MNO_1 can only win in case of a draw (*e.g.*, QoS-C 4 in the example). This still happens approximately 15 percent of the time as table 4.4 shows. This test demonstrates clearly that submitting better bids leads to more wins.

An evaluation for handicaps higher than 1 was also carried out. Table 4.5 shows the average of 10 test runs for each of the handicaps. From these results, it is apparent that the win ratio of MNO_1 is lower the higher the handicap is. It also shows that there is little difference in win ratio between handicaps 3 and 4. Leaving MNO_1 out of the picture, tables 4.4 and 4.5 demonstrate that the other two MNOs win approximately the same amount of times.

QoS-C	$MNO_{2,3}$ TeR-C	MNO_1 TeR-C
0	0	$0 + 1 = 1$
1	0	$0 + 1 = 1$
2	2	$2 + 1 = 3$
3	3	$3 + 1 = 4$
4	4	$4 + 1 = 4$

Table 4.3: Example bid using the same seed for each MNO, handicap 1 applied for MNO_1

	MNO_1	MNO_2	MNO_3
Test run 1	0.1429	0.4340	0.4230
Test run 2	0.1524	0.4309	0.4170
Test run 3	0.1675	0.4175	0.4150
Test run 4	0.1585	0.4457	0.3958
Test run 5	0.1726	0.4223	0.4051
Test run 6	0.1379	0.4795	0.3826
Test run 7	0.1558	0.4300	0.4143
Test run 8	0.1501	0.4182	0.4316
Test run 9	0.1303	0.3999	0.4699
Test run 10	0.1530	0.3417	0.5052
Average	0.1521	0.4220	0.4259

Table 4.4: Average win ratio of three MNOs using the same seed, 1000 bids per test run and MNO, handicap 1 applied to MNO_1

4.1.4 Different Seed, Handicapped MNO_1

A similar evaluation as the one above but with different seeds for the RNGs has also been conducted. This is a more realistic situation since MNOs are unlikely to bid exactly the same. This situation allows the handicapped MNO to win even if there is no draw, because the RNG of that MNO can generate TeR-Cs that are lower than those of the other MNOs. As it turns out, this does not increase the win ratio of the handicapped MNO; the results show that the win ratio of MNO_1 is even worse than the win ratio in the case of the same seed. One possible explanation for this is that due to the large number of bids, the bids of MNO_1 are worse on average than the bids of the other two MNOs. The evaluation shows that the potential of wins in cases other than draws cannot compensate for overall worse bids. Table 4.6 shows an example of a random bid for each MNO where MNO_1 has the best TeR-C for one QoS-C despite the handicap. The TeR-Cs for QoS-Cs 3 and 4 are 4 because adding 1 would generate a TeR-C higher than the maximum. Table 4.7 shows the average wins over the different handicaps.

4.1.5 Handicapped MNO_1, Neighborhood Favorable for MNO_2

Here, the influence of adding a neighborhood for tie-breaking has been analyzed. To simulate this situation, the root node has been split into two halves. Then, a mock

	MNO$_1$	**MNO$_2$**	**MNO$_3$**
Handicap 2	0.1202	0.4379	0.4419
Handicap 3	0.1072	0.4430	0.4498
Handicap 4	0.1064	0.4366	0.4569

Table 4.5: Average win ratio of three MNOs using the same seed, 1000 bids per test run and MNO, handicap 2–4 applied to MNO_1

QoS-C	**MNO$_1$ TeR-C**	**MNO$_2$ TeR-C**	**MNO$_3$ TeR-C**
0	$0 + 1 = 1$	2	2
1	$1 + 1 = 2$	2	3
2	$2 + 1 = 3$	2	3
3	$4 + 1 = 4$	3	3
4	$4 + 1 = 4$	3	4

Table 4.6: Example random bids of three MNOs using a different seed, handicap 1 applied to the bid of MNO_1.

winner record where MNO_2 has won for all QoS-Cs has been injected into one of the sub nodes. The MNOs however submitted their bids to the other sub node. This situation leads to MNO_2 always winning in case of a draw, and should analyze the influence of an unfavorable neighborhood on the win ratio of MNO_1. As expected, the win ratio for MNO_1 is much lower in this case, already dropping to approximately 10 percent with handicap 1 (shown in Table 4.8) compared to the test without a neighborhood as a factor (shown in Table 4.7).

While the win ratio of MNO_1 for handicap 0 is much better in the case of a random seed, the difference in win ratio is not significant for handicap 1 – 4. The most likely reason for this divergence is that with a random seed, there is a lower chance for draws, which MNO_2 wins with a high probability. One would expect that in the case of the same seed that MNO_2 wins all the auctions. However, in this case, the rule that a MNO cannot win more than a certain amount in a row is demonstrated, allowing MNO_1 and MNO_3 to win approximately 21 percent of the time. This means it is possible for MNOs to win a certain amount of times even in an unfavorable neighborhood, although it is clearly an advantage to have a good neighborhood.

The win ratio of MNO_3 shows an expected pattern. In the case of the same seed being used, MNO_3 will always lose to MNO_2 in case of a draw. This is why the win ratio is higher on average in the case of a different seed.

4.1.6 Handicapped MNO$_1$, Neighborhood Favorable for MNO$_2$ & MNO$_3$

A scenario where the neighborhood is favorable for both MNO_2 and MNO_3, while MNO_1 still has a handicap, was created in this test to analyze what impact on the tie-breaking such a situation has. Similarly to the test case above, the root node was split. However,

	MNO_1	**MNO_2**	**MNO_3**
Handicap 1	0.1480	0.4272	0.4247
Handicap 2	0.1113	0.4450	0.4436
Handicap 3	0.1068	0.4420	0.4513
Handicap 4	0.1015	0.4423	0.4562

Table 4.7: Average win ratio of three MNOs using a different seed, 1000 bids per test run and MNO, handicap 1–4 applied to MNO_1

	MNO_1	**MNO_2**	**MNO_3**
Handicap 0	0.1944	0.5838	0.2218
Handicap 1	0.0998	0.6283	0.2719
Handicap 2	0.0885	0.6232	0.2883
Handicap 3	0.0892	0.6478	0.2629
Handicap 4	0.0721	0.6194	0.3085

Table 4.8: Average win ratio of three MNOs using the same seed, 1000 bids per test run and MNO, handicap 0–4 applied to MNO_1, favorable neighborhood for MNO_2

additionally to injecting a favorable winner record for MNO_2, a favorable winner record for MNO_3 was inserted into a third node. The three MNOs still submitted their bids to a node which does not have a winner record initially.

While it is still reasonably likely for MNO_1 to win in the case of handicap 0, winning becomes increasingly unlikely for all other cases. Interesting about this evaluation is how MNO_1 can win at all when the seeds are the same (although MNO_1 only does so in 10 percent of the auctions), since even if one of the other MNOs has won the predefined maximum times in a row, the other has not by default. One possible explanation for this is that the threads on which the simulated MNOs run, cannot submit their bids at exactly the same time. It is expected that MNO_1 wins if the RNG generates a better TeR-C than the current winning one for a certain QoS-C and can submit that bid faster than the other MNOs. Another unexpected facet in this case is that MNO_1 still wins 21 percent of the time when the seed is different and a handicap of 1 is applied. It appears that draws do not happen as often as to bring down the win ratio of MNO_1 in an unfavorable neighborhood. In fact, Table 4.11 shows that draws are approximately 25 percent more likely to happen in the case of the same seed being used. This explains the difference in win ratios shown in Table 4.10 and Table 4.12.

4.1.7 Handicapped MNO_1, Neighborhood Favorable for MNO_1

This test was conducted to find out if having a favorable neighborhood can compensate for having bad bids. The results of this test were surprising, allowing MNO_1 to still win more often than the other MNOs even with handicap 4 in the case of the same seed. One possible explanation for this could be the fact that MNO_1 always wins the draws in QoS-Cs where the initial bid generated a TeR-C of 4, which happens regularly with only five different TeR-C.

	$\mathbf{MNO_1}$	$\mathbf{MNO_2}$	$\mathbf{MNO_3}$
Handicap 0	0.2643	0.4634	0.2723
Handicap 1	0.1069	0.5614	0.3317
Handicap 2	0.0739	0.5726	0.3535
Handicap 3	0.0629	0.5754	0.3617
Handicap 4	0.0630	0.5755	0.3615

Table 4.9: Average win ratio of three MNOs using a different seed, 1000 bids per test run and MNO, handicap 0–4 applied to MNO_1, favorable neighborhood for MNO_2

	$\mathbf{MNO_1}$	$\mathbf{MNO_2}$	$\mathbf{MNO_3}$
Handicap 0	0.1122	0.4387	0.4491
Handicap 1	0.0294	0.4841	0.4865
Handicap 2	0.0043	0.4939	0.5018
Handicap 3	0.0017	0.5000	0.4982
Handicap 4	0.000	0.4943	0.5057

Table 4.10: Average win ratio of three MNOs using the same seed, 1000 bids per test run and MNO, handicap 0–4 applied to MNO_1, favorable neighborhood for MNO_2 & MNO_3

As expected, in the case of the random seed, the win ratio of MNO_1 is lower than in the case of the same seed being used. Here, MNO_1 has the disadvantage that draws are less likely to happen, however, MNO_1 gains the possibility to generate winning TeR-Cs for certain QoS-Cs. Nevertheless, a winning percentage of 28 percent with handicap 4 was unexpected. It appears that it is possible to compensate for bad bids by having a favorable neighborhood.

4.1.8 Conclusions from the Fairness Evaluation

Many interesting conclusions can be drawn from the evaluation of the fairness, one of which is that it is certainly possible to make up for bad bids by having a favorable neighborhood, as shown in the last test case. On the same token, the results also show that when a MNO has an unfavorable neighborhood, bad bids have a worse effect on the win ratio. This is most evident in the test case where the seed was different and the neighborhood favorable for both MNO_2 and MNO_3. While MNO_1 still reached a relatively high win percentage of 21 percent with handicap 0, it dropped to 4 percent already with handicap 1. This situation can be considered the worst case scenario for MNO_1, because it is highly unlikely that both competitors are equally dominating in the neighborhood of the node in question. That would require both of them to have won exactly the same amount of time in the entire neighborhood and over all QoS-C. The fact that bad bids combined with an unfavorable neighborhood lead to a low win percentage is evidence for the fairness of the auction, because MNOs in such a situation are not allowed to win a many auctions for the auction mechanism to be fair.

One of the requirements of the AbaCUS auction was that there is the possibility for MNOs to catch up who suddenly become more competitive at a certain point in time.

	Same Seed	**Different Seed**
Handicap 0	0.8007	0.4697
Handicap 1	0.7260	0.5146
Handicap 2	0.7553	0.5185
Handicap 3	0.7449	0.5092
Handicap 4	0.7958	0.5077

Table 4.11: Draw probability of the auctions for handicap 0 – 4

	MNO_1	**MNO_2**	**MNO_3**
Handicap 0	0.2127	0.3937	0.3936
Handicap 1	0.0444	0.4770	0.4786
Handicap 2	0.0112	0.4958	0.4929
Handicap 3	0.0026	0.4969	0.5004
Handicap 4	0.000	0.5010	0.4990

Table 4.12: Average win ratio of three MNOs using a different seed, 1000 bids per test run and MNO, handicap 0–4 applied to MNO_1, favorable neighborhood for MNO_2 & MNO_3

According to the evaluation, this requirement has been fulfilled. Furthermore, it appears that concentrating on a few valuable nodes and therefore submitting better bids only to them is not a viable strategy. This was most apparent in the case where MNO_1 had a favorable neighborhood but a handicap of 4. Even with a different seed, MNO_1 still won 28 percent of the auctions. The best strategy is possibly to provide better than average bids all over the auction space rather than providing the best bids for a select few nodes. Proving this hypothesis and further investigation into the ideal bidding strategy for MNOs provide interesting tasks for future research.

From the end-users' point of view, the findings are positive, because MNOs are encouraged to provide good QoS for a low MTR in the entire auction space if they want to maximize their profits, which also means cheaper average MTR and better average QoS for the end-users.

4.2 Evaluation of Scalability

Two tests were conducted to find a measure of the scalability of the implemented prototype system. Firstly, a test with three MNOs sending bids as fast as they can for 10 minutes was carried out. Secondly, multiple mock callers were implemented which sent simultaneous service requests. Due to time constraints, the tests were not conducted in a realistic production environment. Instead, they were run on the aforementioned personal computer, which can be considered a best case scenario offering minimal network delay and packet loss. A consequence of this is that especially the numbers for HTTP requests should be taken lightly. However, the test was conducted to see if even under those conditions possible bottlenecks can be identified.

	MNO_1	**MNO_2**	**MNO_3**
Handicap 0	0.5591	0.2009	0.2400
Handicap 1	0.3714	0.3112	0.3175
Handicap 2	0.3533	0.3186	0.3282
Handicap 3	0.3446	0.3240	0.3318
Handicap 4	0.3483	0.3291	0.3226

Table 4.13: Average win ratio of three MNOs using the same seed, 1000 bids per test run and MNO, handicap 0–4 applied to MNO_1, favorable neighborhood for MNO_1

	MNO_1	**MNO_2**	**MNO_3**
Handicap 0	0.4562	0.2695	0.2742
Handicap 1	0.3299	0.3344	0.3357
Handicap 2	0.2943	0.3519	0.3538
Handicap 3	0.2880	0.3590	0.3530
Handicap 4	0.2799	0.3592	0.3608

Table 4.14: Average win ratio of three MNOs using a different seed, 1000 bids per test run and MNO, handicap 0–4 applied to MNO_1, favorable neighborhood for MNO_1

4.2.1 Simultaneous Bids from MNOs

As described above, three mock MNOs were run in their separate threads and instructed to send bids as fast as possible to a random node for 10 minutes. This number of minutes was chosen after initial test runs showed that there was no significant change of results after this time. During this test, the average time to send a bid and receive a response from the server was measured. On the server side, the time it takes the server to handle the MNO request, as well as the time it takes to request and receive the winner records of the neighbors, was measured. The difference between those two values can be attributed to the time it takes to calculate the winner for each QoS-C. The aforementioned measurements were conducted using five different server node configurations, once for 10 nodes, once for 20 and so on increasing in steps of ten until 50.

No. nodes	**Avg. time/request [ms]**	**Avg. time to fetch neighbor winner records [ms]**	**Avg. time to calculate winners [ms]**
10	6.5649	6.2291	0.3357
20	8.1903	7.8286	0.3617
30	8.8059	8.4263	0.3796
40	10.7912	10.3394	0.4518
50	11.3366	10.8900	0.4466

Table 4.15: Server side measurement of average time taken for handling a request

The evaluation shows that it takes the nodes longer to handle the request the more nodes there are in the AbaCUS system. It also shows that a large part of the time it takes to handle the request is spent getting the winner records from neighboring nodes. This is

done by sending HTTP requests to each of the neighbors and in turn receiving a reply from them. The time to calculate the winners is still below 0.5 ms with 50 nodes in the system, which is less than around 4 percent of the total time. It only takes between 1.4 and 1.7 milliseconds for the communication between the MNO and the server which is also done via HTTP. The time it took for the HTTP requests to transmit is on the low end, because both the nodes and the MNOs were running on the same machine using a local network. It most certainly take longer for those messages to be sent around in a real world scenario, because MNOs and nodes are further apart and it takes longer for messages to reach the destination. Therefore, a possible bottleneck for the system can be identified in the HTTP requests, which encompass 96 percent of the time taken for handling a request even on a local machine with ideal network conditions. An important aspect that has to be mentioned is, that it is very unlikely for MNOs to send bids as fast as they can in a real world scenario, because they are restricted by their network infrastructure and the cost involved and cannot react to changing bids instantly. The test below shows a more realistic situation with simultaneous service requests by callers.

No. nodes	**Avg. time/request [ms]**
10	7.9624
20	9.5725
30	10.1465
40	12.4387
50	13.0561

Table 4.16: Client side measurement of average time taken for sending a request and receiving a reply

4.2.2 Simultaneous Service Requests by Callers

Since callers and callees outnumber MNOs by orders of magnitude in the real world, a stress test of the system was carried out with mock callers as opposed to with MNOs. Caller requests have different requirements than MNO requests; the most work-intensive request a caller can send is a service request. Handling such a request first requires the AbaCUS node to check if the caller is in the auction space of that specific node. If it is, the node finds the correct MNO to contact, looking at the provided QoS-C and TeR-C in the request. Since no connectivity between MNOs and callers or callees are implemented in the prototype Au^2 implementation, for test purposes, the node just sends a reply with the correct MNO to the original caller. If the node that was contacted is not the correct node for the location of the caller, the node searches the AbaCUS node tree for the correct node and sends a request to the correct node, querying for the correct MNO for the request by the original caller. The AbaCUS node which was first contacted by the caller then replies with the correct MNO and also the contact information of the correct node for the location of the caller, so the caller can contact that node directly the next time. To simulate a scenario of heavy strain on the personal computer, a node network of 50 nodes was chosen again. The mock callers each ran in their own threads, sending a service requests to the root node. Each caller was generated with a random position inside the AbaCUS auction space. Since the root node was split, it had to forward all the requests

to the correct node. The average time it takes for each caller to send a request and receive a reply has been tested for 100, 1000 and 10000 and 100000 callers.

No. callers	**Avg. time/request [ms]**
100	105.02
1000	480.36
10000	511.56
100000	920.00

Table 4.17: Client-side evaluation of time taken for a caller request in milliseconds

The results show that it takes the server longer to handle a request the more simultaneous requests reach the server. However, because the test was conducted on a single notebook with 8 cores running both server nodes and clients, the resulting numbers have to be taken lightly. In future research, an investigation into the scalability using dedicated commercial servers is required, such that meaningful estimates for the scalability can be made.

Chapter 5

Summary, Conclusions and Future Work

This thesis presents a prototypical Au^2 implementation, which provides means for mock MNOs to submit bids and for mock callers to send service requests. The implementation has been thoroughly evaluated in terms of fairness of the auction mechanism. It has been shown that the goal of providing a fair auction mechanism has been accomplished. The evaluation has also confirmed that the requirement, that there is a catch-up mechanism for MNOs, and that the auction mechanism gives an incentive for MNOs to be competitive over the entire auction space, has been fulfilled. Against this background, an investigation into the best bidding strategy for MNOs provides an interesting topic for future research.

A second evaluation regarding the scalability has also been done, although not in a production environment but rather on a personal computer. The evaluation of the scalability has identified HTTP messages as a possible bottleneck. To confirm this suspicion, the prototype has to be evaluated with regards to the scalability in a distributed network with dedicated servers.

There are certain aspects of the prototype that can be improved. First of all, no security mechanisms were implemented in the prototype. For the prototype to be used in a practical environment, end-to-end encryption of messages as well as an authentification and authorization process have to be implemented. Secondly, AbaCUS nodes only store the bare minimum of data that is required for the moment-to-moment operation. Adding a persistence layer and the corresponding database system would allow the AbaCUS nodes to store historical data of the auctions as well as the service requests. This would also allow a statistical analysis of the auction mechanism and the service handling. Examination of those statistics would provide a new view into AbaCUS and could discover new aspects that have not been considered before. A persistence layer with a corresponding database system also allows the server to restart where it left off in case of a crash or voluntary shutdown. Thirdly, the implemented Web-GUI is a first draft and only shows the most important information about a AbaCUS node and its neighbors. Furthermore, a complete integration with the work done in [11] should be achieved in a next step. This would show if the two implementations are compatible or if there are any unforeseen problems.

All in all, this thesis has shown that AbaCUS is a viable proposal to solve the monopoly problem in mobile call termination. However, further research into this topic is certainly required, with the goal that AbaCUS can one day see the light of day in a real world application.

Bibliography

[1] Apache Tomcat, http://tomcat.apache.org/, Visited in March 2014.

[2] Laurence M. Ausubel, Paul Milgrom, "The lovely but lonely Vickrey auction", Combinatorial auctions, pp. 17–40, 2006.

[3] Danish competition and consumer authority, "Chapter 5. Call termination", http://www.kfst.dk/en/service-menu/publications/publication-file/publikationer-2004/telecompetition-towards-a-single-nordic-market-for-tele-communication-services/chapter-5-call-termination-pock-ets-of-monopoly-power/, Visited in March 2014.

[4] Martin Fowler, "Inversion of Control Containers and the Dependency Injection pattern", http://martinfowler.com/articles/injection.html, Visited in March 2014.

[5] Jesse James Garrett, "Ajax: A New Approach to Web Applications", http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/, Visited in March 2014.

[6] Google Cloud Messaging for Android, http://developer.android.com/google/gcm/index.html, Visited in March 2014.

[7] How good is java.util.Random?, http://stackoverflow.com/questions/453479/how-good-is-java-util-random, Visited in March 2014.

[8] HttpServlet, http://docs.oracle.com/javaee/1.3/api/javax/servlet/http/HttpServlet.html, Visited in March 2014.

[9] HttpSession, http://docs.oracle.com/javaee/1.4/api/javax/servlet/http/HttpSession.html, Visited in March 2014.

[10] Java Servlet Technology Overview, http://www.oracle.com/technetwork/java/overview-137084.html, Visited in March 2014.

[11] Samuel Liniger, "Implementation of an automatic, on-demand Mobile Network Operator (MNO) selection mechanism on Android devices", Bachelor thesis, Communication Systems Group (CSG), Computer Science Department (IFI), University of Zurich (UZH), Zurich, Switzerland, August 2013.

[12] List of mobile network operators of Europe, http://en.wikipedia.org/w/index.php?title=List_of_mobile_network_operators_of_Europe, Visited in March 2014.

[13] List of mobile network operators of the Americas, http://en.wikipedia.org/w/index.php?title=List_of_mobile_network_operators_of_the_Americas, Visited in March 2014.

[14] List of mobile network operators of the Asia Pacific region, http://en.wikipedia.org/w/index.php?title=List_of_mobile_network_operators_of_the_Asia_Pacific_region, Visited in March 2014.

[15] List of mobile network operators of the Middle East and Africa, http://en.wikipedia.org/w/index.php?title=List_of_mobile_network_operators_of_the_Middle_East_and_Africa, Visited in March 2014.

[16] Multiplying Mobile: How Multicultural Consumers Are Leading Smartphone Adoption, http://www.nielsen.com/us/en/newswire/2014/multiplying-mobile-how-multicultural-consumers-are-leading-smartphone-adoption.html, Visited in March 2014.

[17] Quadtree, http://en.wikipedia.org/w/index.php?title=Quadtree, Visited in March 2014.

[18] Moe Rahnema, "Overview of the GSM system and protocol architecture", Communications Magazine, IEEE, vol. 31, no. 4, pp. 92–100, 1993.

[19] Random, http://docs.oracle.com/javase/7/docs/api/java/util/Random.html, Visited in March 2014.

[20] Random seed, http://en.wikipedia.org/w/index.php?title=Random_seed, Visited in March 2014.

[21] Hanan Samet, "Neighbor finding techniques for images represented by quadtrees", Computer Graphics and Image Processing, vol. 18, no. 1, pp. 37–57, 1982.

[22] ServletContext, http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html, Visited in March 2014.

[23] Spring Framework, http://projects.spring.io/spring-framework/, Visited in March 2014.

[24] The Content-Type Header Field, http://www.w3.org/Protocols/rfc1341/4_Content-Type.html, Visited in March 2014.

[25] The Original HTTP as defined in 1991, http://www.w3.org/Protocols/HTTP/AsImplemented.html, Visited in March 2014.

[26] Tomcat, http://tomcat.apache.org/tomcat-7.0-doc/api/org/apache/catalina/startup/Tomcat.html, Visited in March 2014.

[27] Christos Tsiaras, Samuel Liniger, Burkhard Stiller, "An Automatic and On-demand MNO Selection Mechanism", IEEE/IFIP Network Operations and Management Symposium (NOMS 2014), Management in a Software Defined World, Krakow, Polland, May 2014.

[28] Christos Tsiaras, Samuel Liniger, Burkhard Stiller, "Automatic and On-demand Mobile Network Operator (MNO) Selection Mechanism Demonstration", IEEE/IFIP Network Operations and Management Symposium (NOMS 2014), Management in a Software Defined World, Krakow, Polland, May 2014.

[29] Christos Tsiaras, Burkhard Stiller, "Challenging the Monopoly of Mobile Termination Charges with an Auction-Based Charging and User-Centric System (AbaCUS)", in 2013 Conference on Networked Systems (NetSys), Stuttgart, Germany, March 2013, pp. 110–117.

[30] What is a Servlet Container?, `http://java.dzone.com/articles/what-servlet-container`, Visited in March 2014.

[31] What is Maven?, `http://maven.apache.org/what-is-maven.html`, Visited in March 2014.

[32] World Geodetic System, `http://en.wikipedia.org/w/index.php?title=World_Geodetic_System`, Visited in March 2014.

[33] Your First Cup, `http://docs.oracle.com/javaee/6/firstcup/doc/gcrky.html`, Visited in March 2014.

Abbreviations

AbaCUS	Auction-based Charging and User-centric System
Au^2	Auction Authority
CPU	Central Processing Unit
GCM	Google Cloud Messaging
CSG	Communication Systems Group
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
MNO	Mobile Network Operator
MTR	Mobile Termination Rate
QoS	Quality of Service
QoS-C	Quality of Service Class
RAM	Random-access Memory
RNG	Random Number Generator
TeR-C	Termination Rate Class
URL	Uniform Resource Locator

Glossary

Communication Systems Group The thesis at hand was written under the supervision of the CSG research group of the Computer Science Department (IFI) of the University of Zurich.

Depth of a tree node The lenght of the path from the root node to the tree node.

Height of a tree node The length of the longest path from the tree node to a leaf node.

Leaf node of a tree A tree node without child nodes.

Mobile Termination Rate The monetary value that a Mobile Network Operators collects for terminating a call. Consists of an initial call establishment fee and a fee depending on the duration of the call.

Quality of Service A measure for sound quality and call establishment priority.

Root node of a tree The topmost node of a tree.

List of Figures

List of Tables

Appendix A

Bootstrapping the AbaCUS Node Grid

```
public void bootstrap() {
    AbacusNode rootNode = new AbacusNode(null); // null == no parent
    AuctionEngine auctionEngine = new StandardAuctionEngine(); // or use
                                                               // Spring

    rootNode.setAuctionEngine(auctionEngine);
    auctionEngine.setNode(rootNode);
    String rootUrl = AbacusConnectionUtils.startTomcat(0);
    rootNode.setUrl(rootUrl); // http://127.0.0.1:9090/an0

    AbacusConnectionUtils.sendNodeTree(rootNode, rootUrl);
}

public static String startTomcat(int servletNumber) { // traditionally 0
                                                      // for the root
    Tomcat tomcat = new Tomcat();
    tomcat.setPort(BASE_PORT + servletNumber); // BASE_PORT is 9090 in the
                                               // prototype
    String webappDirLocation = "";
    Host host = tomcat.getHost();
    Context rootCtx = tomcat.addContext(host, "", new File(
           webappDirLocation).getAbsolutePath());
    AbacusServlet abacusServlet = new AbacusServlet();
    String servletMappingName = "AbacusServlet";
    Tomcat.addServlet(rootCtx, servletMappingName, abacusServlet);

    String servletUrl = BASE_SERVLET_NAME + String.valueOf(servletNumber);
    // BASE_SERVLET_NAME is "an" (short for AbacusNode) in the prototype

    rootCtx.addServletMapping(servletUrl, servletMappingName);

    try {
        tomcat.start();
```

```
        } catch (LifecycleException e) {
            // ... error handling ...
        }

        String url = BASE_HOST_URL + String.valueOf(BASE_PORT + servletNumber)
                + servletUrl;
        // BASE_SERVLET_URL is "http://127.0.0.1:" in the prototype

        return url;
    }
```

Appendix B

Splitting an AbaCUS Server Node

```
public void splitAbacusNodeById(int nodeId) {
    AbacusNode node = findNode(nodeId);
    if (node != null) { // node can be null if it wasn't found
        node.splitAbacusNode();
        updateChildren();
    }
}

public void splitAbacusNode() {
    if (!isSplit()) {
        setSplit(true);

        setNorthEastChild(createNewChild());
        setNorthWestChild(createNewChild());
        setSouthEastChild(createNewChild());
        setSouthWestChild(createNewChild());
    }
}

public AbacusNode createNewChild() {
    AbacusNode node = new AbacusNode(this);

    AuctionEngine auctionEngineCloned = SerializationUtils
            .clone(getAuctionEngine());

    node.setAuctionEngine(auctionEngineCloned);
    auctionEngineCloned.setNode(node);
    String nodeUrl = AbacusConnectionUtils.startTomcat(node.getId());
    node.setUrl(nodeUrl);

    return node;
}
```

```
public void updateChildren() {
    List<AbacusNode> children = getAllChildren(new ArrayList<AbacusNode>());
    for (AbacusNode child : children) {
        AbacusConnectionUtils.sendNodeTree(child, child.getUrl());
    }
}
```

Appendix C

Installation Guidelines

Import the folder webapp in the folder thesis-das-abacus on the enclosed CD into the IDE of choice (recommended: Eclipse). Run Maven to download the necessary external libraries.

Alternatively, import any of the three jar archives `thesis-das-abacus-bootstrap.jar`, `thesis-das-abacus-split.jar` or `thesis-das-abacus-mno.jar`. They already contain the necessary external libraries.

Demo

The folder thesis-das-abacus contains three runnable jar archives. Run them from that folder in the following order to get a quick demonstration of the Au^2 prototype:

1. Run `java -jar thesis-das-abacus-bootstrap.jar` on the command line to bootstrap the Au^2 root node.
2. Run `java -jar thesis-das-abacus-split.jar` on the command line to split the Au^2 root node a random number between 0 and 10 times. This can be repeated as many times as desired.
3. Run `java -jar thesis-das-abacus-mno.jar` on the command line to start three mock MNOs that send random bids to a random node in the Au^2 prototype node grid.
4. To view the Web-GUI of the root node, visit `http://127.0.0.1:9090/an0`. From there, the Web-GUIs of all other nodes can be accessed through hyperlinks.

Appendix D

Contents of the CD

The enclosed CD contains the following items:

- **thesis-das-abacus:** Folder that contains the source code of the Au^2 prototype application including all tests. The folder also contains three jar archives for demoing purposes.
- **thesis-latex:** Folder that contains the latex source files and images of the thesis.
- **thesis-relatedworks:** Folder containing the related works papers referenced in the thesis.
- **thesis.pdf:** Printable pdf file of the thesis.
- **final-presentation.ppt:** The final presentation of the thesis.
- **Zusfsg.txt:** The abstract in German.
- **Abstract.txt:** The abstract in English.