Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2014,12
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Ubiquitäre Systeme (Seminar) und Mobile Computing (Proseminar) SS 2014

Mobile und Verteilte Systeme
Ubiquitous Computing

Teil XI

Herausgeber:
Martin Alexander Neumann, Anja Bachmann,
Yong Ding, Till Riedel

2014

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Karlsruhe Institute of Technology

Ubiquitäre Systeme (Seminar) und Mobile Computing (Proseminar) SS 2014

Mobile und Verteilte Systeme Ubiquitous Computing

Teil XI

Herausgeber

Martin Alexander Neumann
Anja Bachmann
Yong Ding
Till Riedel

Karlsruhe Institute of Technology (KIT)
Fakultät für Informatik
Lehrstuhl für Pervasive Computing Systems (PCS) und TECO

Interner Bericht 2014-12
ISSN 2190-4782

Vorwort

Die Seminarreihe Mobile Computing und Ubiquitäre Systeme existiert seit dem Wintersemester 2013/2014. Seit diesem Semester findet das Proseminar Mobile Computing am Lehrstuhl für Pervasive Computing System statt. Die Arbeiten des Proseminars werden seit dem mit den Arbeiten des zweiten Seminars des Lehrstuhls, dem Seminar Ubiquitäre Systeme, zusammengefasst und gemeinsam veröffentlicht.

Die Seminarreihe Ubiquitäre Systeme hat eine lange Tradition in der Forschungsgruppe TECO. Im Wintersemester 2010/2011 wurde die Gruppe Teil des Lehrstuhls für Pervasive Computing Systems. Seit dem findet das Seminar Ubiquitäre Systeme in jedem Semester statt. Ebenso wird das Proseminar Mobile Computing seit dem Wintersemester 2013/2014 in jedem Semester durchgeführt. Seit dem Wintersemester 2003/2004 werden die Seminararbeiten als KIT-Berichte veröffentlicht. Ziel der gemeinsamen Seminarreihe ist die Aufarbeitung und Diskussion aktueller Forschungsfragen in den Bereichen Mobile und Ubiquitous Computing.

Dieser Seminarband fasst die Arbeiten der Seminare des Sommersemesters 2014 zusammen. Die Themenvielfalt der hier zusammengetragenen Aufsätze umfasst Dynamische Softare Updates für Java, menschliche Aktivitätserkennung und Caching auf mobilen Geräten, Vorhersage von Zeitreihen, Datenschutz im Crowdsourcing, Aspekte des Energiehandels, ein Überblick über große Stromausfälle, User Experience von APIs, sowie Continuous Integration in der Entwicklung von grafischen Oberflächen für mobile Geräte. Wir danken den Studierenden für ihren besonderen Einsatz, sowohl während des Seminars als auch bei der Fertigstellung dieses Bandes.

Karlsruhe, den 15. September 2014

Martin Alexander Neumann
Anja Bachmann
Yong Ding
Till Riedel

Inhaltsverzeichnis

Process Mining - Determining Structure of Distributed Behaviour

Ibrahimov Kenan*

Advisor: Martin Alexander Neumann†

Karlsruhe Institute of Technology (KIT)
Pervasive Computing Systems – TECO
*uajke@student.kit.edu
†mneumann@teco.edu

Abstract. Imagine following situation: we have been given a data set, which contains records of the last n executions of a particular process and the task is to discover a model, which is best suited for the behaviour captured in given data set. It is a typical use case for Process Mining, which is supposed to solve this task. This paper examines main challenges, which might arise, while developing a Process Mining algorithm, to solve this task and highlights some possible solutions. Furthermore, the second part of this paper is dedicated to advanced Process Mining algorithms, which address some of the those challenges.

Keywords: Process Mining, event logs, Petri Nets, Process Discovery, Process Modelling

1 Introduction

This introductory section provides a brief overview of Process Mining. There are three different types of Process Mining: *Discovery*, *Conformance* and *Enhancement*. *Discovery* is used, when a model of a given process must be constructed. There exists no prior model, and the only given information source is an event log. The task of the Discovery is to extract an implicit process knowledge from the data, which is recorded during the process executions in the given event log. *Conformance* compares the existing process model with the *reality*. The goal is to find out, if the existing model validates the behaviour captured in a given event log. *Enhancement* aims to improve the existing process model based on the reality (event log). Ideally it leads to the new model, which has better *quality* with respect to the underlying process.

In this paper we focus on *Discovery* and henceforth referred to as Process Mining. Today's information systems record data during execution of a process in so called *event logs*. These data reflects the reality as opposed to the *idealized process model* which could arise if one constructs a process model based only on formalities, functional specifications of the underlying process. Thus, Process

Mining constructs more realistic models which are very close to the behaviour seen in event logs[4]. For example, in Table 1 we see the fragment of some event log, which could be one possible input for a Process Mining algorithm. Ideally the output would be the *most suitable* process model with regard to the behaviour captured in this event log. We will formalize the term *suitable* process model later in this paper. In fact, given the corresponding event log of the fragment in Table 1, the α-algorithm[1][1] returns the Petri Net (see section *Preliminaries*) in Fig. 1 [4].

The remainder of this paper is organized as follows. Chapter Two begins by laying out the *preliminaries* needed for the further chapters. The third chapter is concerned with the *challenges* existing in Process Mining. The fourth chapter presents the *advanced techniques*, which have promising results.

2 Preliminaries

Before proceeding to examine process mining, it will be necessary to introduce some techniques, which are used in the remainder of this paper.

2.1 Business process modeling

According to a definition provided by Wikipedia , "*Business process modeling* is the activity of representing processes of an enterprise, so that the current process may be analyzed and improved." [8]. Processes can be modelled via various *process modelling languages*, such as Transition Systems, Petri Nets, Business Processing Modelling Notation (BPMN), Workflow Nets, Causal Nets and so on.

If we appear to be developing a Process Mining algorithm, we will have to deal with the choice of process modelling language twice. Firstly we have to make a choice of the *representation*, which will be used during discovery process itself. This choice is very crucial. Because when we choose a particular modelling language for the representation, we limit the search space[2] only to the models in this particular process modelling language. This can bring with it several problems as well as efficiency for our algorithm[3]. We will discuss about it in paragraph *representational bias*.

Having chosen a certain modelling language for representation during discovery process, we know in which modelling language the output of our algorithm will be. Namely in that language, which we have chosen. Once we get the output of our algorithm, we may want to describe the result in other process modelling language, for example for the purpose of better visualization. We have here limited scope, as there may be some processes which can not be described in target

[1] α-algorithm is one of the first Process Mining algorithms. While we will mention this algorithm from time to time, there will be no paragraph dedicated to it. The reason for that is that this algorithm in its original form has many problems concerning challenges, which will be presented later in this paper.

[2] i.e. the set of all possible models, which come into question.

modelling language. It's important to understand the difference between *representation* used during discovery process and *visuaization* of the result. While the latter has no influence on the *correctness* of the result, the former can effect the *correctness* of the result and *efficiency* of our algorithm (see the paragraph *representational bias*).

2.2 Event logs

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	Register request	Pete	50	...
	35654424	31-12-2010:10.06	Examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	Check ticket	Mike	100	...
	35654426	06-01-2011:11.18	Decide	Sara	200	...
	35654427	07-01-2011:14.24	Reject request	Pete	200	...
2	35654483	30-12-2010:11.32	Register request	Mike	50	...
	35654485	30-12-2010:12.12	Check ticket	Mike	100	...
	35654487	30-12-2010:14.16	Examine casually	Pete	400	...
	35654488	05-01-2011:11.22	Decide	Sara	200	...
	35654489	08-01-2011:12.05	Pay compensation	Ellen	200	...

Table 1: The fragment of the event log in[4].

Case id	Trace
1	<a, b, d, e, h>
2	<a, d, c, e, g>
3	<a, c, d, e, f, b, d, e, g>
4	<a, d, b, e, h>
5	<a, c, d, e, f, d, c, e, f, c, d, e, h>
6	<a, c, d, e, g>

a = register request
b = examine thoroughly
c = examine casually
d = check ticket
e = decide
f = reinitiate request
g = pay compensation
h = reject request

Table 2: simplified event log[4].

Today's enterprise information systems record data produced during process execution in so called *event logs*. When the process is executed, a certain sequence of events (also known as a trace) occur. The sequence of events can vary from execution to execution. Each process execution (and thus also the corresponding

sequence of events) gets an unique *case id* assigned. For each event there is one entry in an event log, which is associated with the corresponding process execution through its case id. Moreover, each event is decribed by certain attributes. The list of attributes may vary between different event logs, however, it is constant within a certain event log. In Table 1 we can see that 2 process executions are captured, with the case id 1 and 2, respectively. The attributes like *Resource*, *Cost*, *Timestamp* etc. give detailed information about the events registered during process execution. The relevance of each attribute for the process discovery depends on requirements of customer on a final model and a Process Mining algorithm used. However, there are some *must-have* attributes, which have to be included in every event log. It must be clear, which *activity* is registered within a single event[3] as well as to which process execution a certain event is related. The event log in Table 1 registered the activity *Check ticket* within the event with event id 35654425, which is the third event within the process execution with case id 1. An another important information is the *order* of the events within a single process execution. In Table 1 we have timestamp for each event, which shows the completion time of that event. In Table 1 the events with the activity *Check ticket* always appear (not necessarily directly) before the events with the activity *Decide*. If this statement would appear to be true for the whole event log, a Process Mining algorithm would certainly consider it.

In terms of Process Mining we refer to the sequences of the events in a single event log as the traces. Thus, each process execution leaves in that event log one trace and each event log is the set of traces. For the compact representation of the traces in this paper we will use single-letter labels instead of activity names and leave non-obligatory attributes out, like the one shown in Table 2.

2.3 Petri Nets

Petri nets are one of the mostly used process modelling language that can describe a process not only by its sequential, deterministic behaviour, but also by considering the concurrency and non-determinism. Petri nets are bipartite, directed graphs which consist of two different type of nodes, *transitions* and *places*. Arcs interconnect a transition and a place or vice versa, but never transitions or places. Transitions (rectangular shapes) represent the activities. Places (round symbols) represent the state of the system by holding a different number of tokens (black marks). The Petri Net in Fig. 2 is in initial state, i.e. only start place contains token(s)[4]. Each transition has input and output place(s) and the state of the system is changed as token(s) move from place(s) to place(s).

Flow of tokens over a Petri Net is governed by *firing rule*. Before a transition can fire, it must be *enabled*. A transition is enabled, if each of its input places contain at least one token. When a transition fires, it removes one token from each of its input places and adds one token to each of its output places. In Fig.

[3]Each event contains exactly one activity.

[4]In this case there is only one token, however there might be any number of tokens in start place at the beginning.

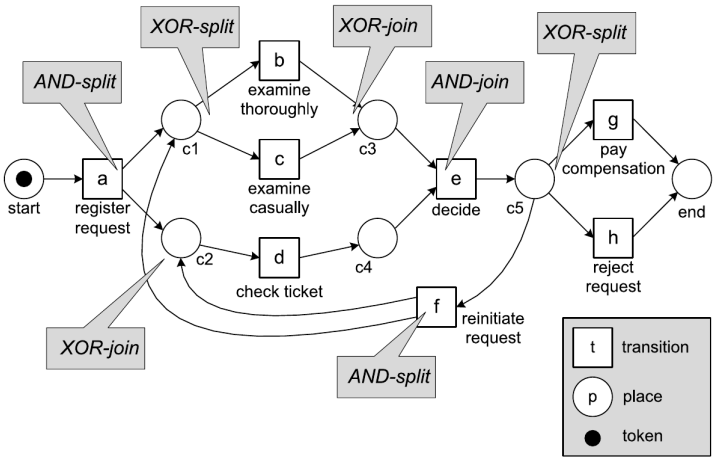


Fig. 1: A Petri net example[4]

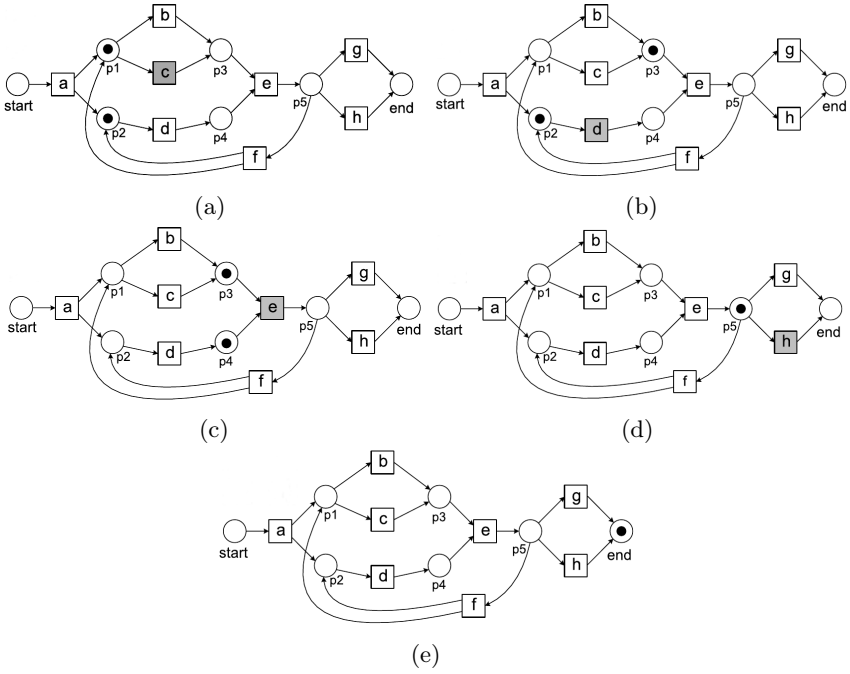


Fig. 2: Illustration of firing sequence for the trace $< a, c, d, e, h >$[4]

1 only the transition a is enabled. After a fires, the Petri Net changes its state to the new one, where only the places *p1* and *p2* hold one token, respectively (See Fig. 2 (a)). In Fig. 2 (a) *p1* is the single input place for the two transitions,

b and c. Hence both of them are enabled. As there is only one token in *p1*, only one of those transitions (either b or c) can fire, in this case c fires and we get the new state shown in Fig. 2 (b). Since here *p4* does not have any token, e is not enabled. After d fires, e is the only place, which is enabled (Fig. 2 (c)). Accordingly, it fires, which in turn enables two transitions, namely g and h (Fig. 2 (d)). And finally, h fires, which causes the Petri Net to switch to the final state (Fig. 2 (e)). Thus, we generated the trace $<a,c,d,e,h>$ with the Petri Net in Fig. 1.

3 Challenges

3.1 Representational Bias

When we develop Process Mining algorithm, we have to make a decision about the modelling language, which will be used for the representation during discovery process. Wil van der Aalst defines *representational bias* as follows: "A representational bias refers to choices that are implicitly made by selecting a particular representation"[4]. Thus, by selecting a particular modelling language for representation, we limit the search space of the algorithm to the models which can be described in this particular modelling language. In the following, we will discuss why it is very important to have *good* representational bias.

At this point one may raise the question: *Why can we not pick a modelling language, which is able to represent all possible models, so that the search space of the algorithm is not limited*? The answer is: Yes, we can pick such a modelling language (e.g. *Transition System*), but by doing so, new problems may arise.

When we do not limit the search space for the algorithm, it contains all possible models, especially *inconsistent*[5] models. Therefore, better representational bias would be the one, which allows for all possible models in search space but inconsistent models. Unfortunately, due to the fact that the inconsistencies such as *deadlocks* or *livelocks* are non-local properties, simple syntactical constraints do not work[6]. In order to avoid inconsistencies, the search space must be limited to the *sound* models. Soundness is a generic notion and it must be defined for every modelling language in its own. Unfortunately, it has its price. For the most algorithms *fulfilling* of the soundness property means time-consuming analysis or severe limitations on expresiveness of the modelling language[4].

Thus, it is crucial to address the problems related to the representational bias. This affects not only the correctness, but also the efficiency of the algorithm, as representational bias can limit the search space to the models, which are relevant for the underlying process.

3.2 Noise

Event logs can contain the data which would not correspond to the typical behaviour of the underlying process. In this context, the term *noise* refers to the

[5]For example, the models which have deadlocks, livelocks or other anomalies.

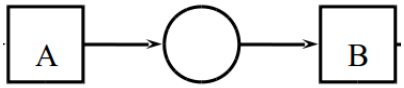


Fig. 3: First A then B

unusual or exceptional behaviour of process captured in an event log. Obviously, it would be preferable that a Process Mining algorithm is able to distinguish between typical behaviour and the noise captured in the event log and consider this fact while discovering a process model. The typical counter-measure is to include *support* and *confidence* metrics so that we can define a threshold for extracting the typical behaviour from the event logs [4]. For example we could say, our algorithm detects *event B* directly after *event A* like in Fig. 3, if and only if the *support* of it (number of times the trace <...,a,b,...> occurs in the event log divided by the number of traces) must be above 30 % and *confidence* of it (the number of times <...,a,b,...> appears in the event log divided by the frequency of a) must be above 60 % [4].

3.3 Incompleteness

One more problem related to the *data* to be analysed is *incompleteness* which is also called *insufficient logging*. The problem can arise by a lack of data in the given event log, more precisely, when a certain trace was not recorded in the given event log, but actually belongs to the underlying process. A significant challenge for a Process Mining algorithm is to find out those traces, which do not occur in the given event log, but are the part of the typical behaviour of the underlying process. Thus, when we develop a Process Mining algorithm we should not assume that the input will be complete. Indeed, the reality is that the event log will mostly contain only *fraction* of all possible traces, which could have been registered.

3.4 Quality Criteria

In this part we introduce four criteria, which can be used to evaluate the quality of the output of a Process Mining algorithm. They are *fitness*, *precision*, *simplicity* and *generalization* [4]. The reason why we include those four criteria among *challenges* is it is indeed difficult to develop a Process Mining algorithm with high quality with respect to those four criteria[4]. Thus, these are competing criteria, which means that trying to improve performance in one direction (e.g. *precision*) can lead to worsening performance in another direction (e.g. *generalisation*).

3.4.1 Fitness: Fitness describes to what extent the output of the given algorithm models the behavior seen in the input of this algorithm. The model with the high fitness is able to generate (replay) most of the typical behaviour captured in the respective event log, whereas the model with low fitness fails to do so. In general, it is desired that a Process Mining algorithm returns on average the models with high fitness. Let's take a look at the example which we presented at the beginning of this paper. As already mentioned that the model described in Fig. 1 is the output of the α-algorithm[1] with the input described in Table 2. In fact, this model has the perfect fitness, as it is able to replay each trace seen in Table 2.

3.4.2 simplicity: The term *simplicity* in the context of Process Mining is nothing more than our understanding of simplicity in general. The final process model returned by a Process Mining algorithm should be as simple as possible. Simplicity of a model can be quantified by the different metrics. In the simplest case we refer to the number of *nodes* or *arcs* of a model.

3.4.3 simplicity and fitness alone are not enough: So far we have introduced two criteria, simplicity and fitness. The example in Fig. 4 shows that those two criteria might be not sufficient to draw conclusions on the quality of some process models. The so called *Flower model* [4] is able to capture any trace consisting of the activities {a, b, ..., h}. In other words, it does not matter in which order those activities appear, the Flower model is able to replay any possible sequence of those activities. It implies that the Flower model has the perfect fitness. Moreover, this model is very simple. For example, because there are only as many nodes as necessary in this model, particularly when it comes to the number of places.

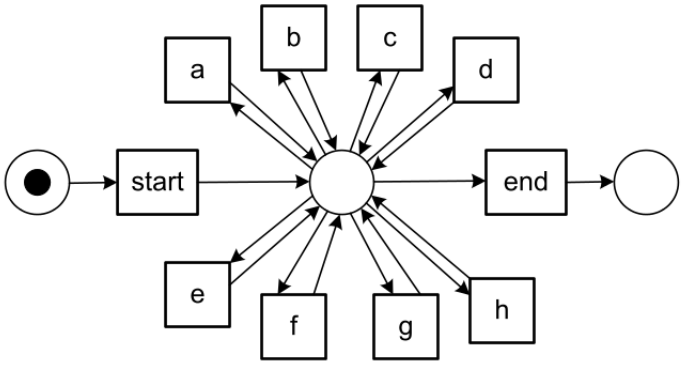


Fig. 4: The "Nonsense" of the perfect fitness + perfect simplicity, which can capture any sequence of activities {a, b, ..., h}[4].

3.4.4 precision: Despite the "perfect" fitness and simplicity, the Flower model has a poor quality. Because it contains "too much behaviour", and thus it is able to fit any process, which generates the traces with the activities {a, b, ..., h}. From this point of view the Flower model contains no specific process knowledge and is relevant for any process with the above-mentioned condition. In general, we want that a Process Mining algorithm returns such a model, which contains the specific process knowledge mostly relevant for the underlying process. Therefore, we introduce the next term, namely *precision*. Precision deals with the question how relevant is the output of a Process Mining algorithm to the underlying process. A model with the high precision (= *overfitting model*) includes only the behaviour, which is contained in the given event log or which is similar to the behaviour contained in the given event log. A model with the low precision is able to generate the behaviour, which differs much from the behaviour seen in the given event log.

3.4.5 generalisation The next important criterion is *generalisation*. If we assume that the input log of a Process Mining algorithm would be always *complete* with respect to the behaviour of the underlying process, then there would be no need to consider this criterion, and therefore, the overfitting model in Fig. 5 would have a high quality. However, as mentioned earlier the event log will mostly contain only fraction of all traces, which can be observed. From this point of view, the model which does not generalise and instead captures only the behaviour seen in the given event log can not be considered as a good model. On the other hand, the model should not overgeneralise the behaviour seen in the given event log, i.e. it should not capture the behaviour which has no similarity with the one seen in the given event log. The model with overgeneralises is also called *underfitting* model[4] (see Fig. 5).

4 Advanced Techniques

Like in the case of Machine Learning there is no *silver bullet* in Process Mining, which would be the ultimate, single solution to the most of the present challenges. The term "No Free Lunch"[9] applies also to Process Mining. This means that the techniques which work good for a certain group of processes, can show poor performance with another group of processes. Furthermore, the parameters (e.g. Noise-Thresholds) of a Process Mining technique may have to be adapted depending on the process or size of the available data set. In this section we present couple of advanced techniques, which base on different approaches.

4.1 Genetic Process Mining

Genetic Process Mining[5] (hereinafter abbreviated to "Genetic Mining") originates from the field of *computational intelligence*. Compared to the other techniques originating not from computational intelligence, Genetic Mining uses *evolutionary approach*, i.e. it does not derive the model directly from given event log,

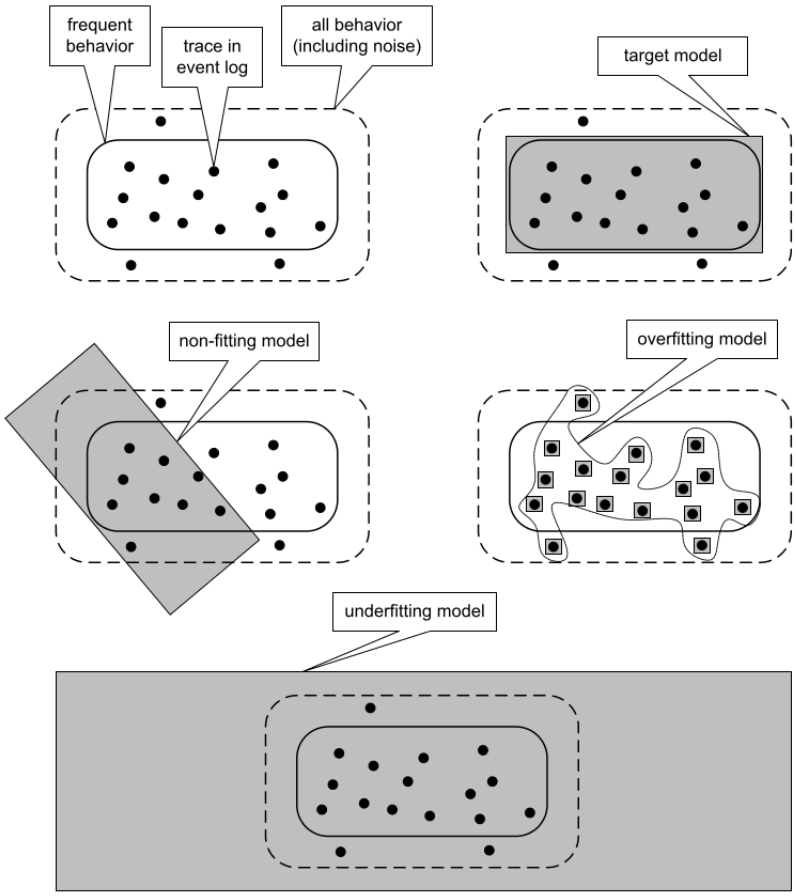


Fig. 5: Overview of the four competing criteria[4]

but rather it uses an iterative procedure in a non deterministic manner, which to some extent depends on randomization. Genetic Mining consists of four steps: a) Initialization, b) Selection, c)Reproduction and d)Termination. In the following, we describe Genetic Mining in its general form, without going into details of a certain implementation.

4.1.1 Initialization Genetic Mining always starts with initialization. During this phase, the first population of process models is created, mostly by means of a randomization. This population serves as a basis for the next generations, which are created at the end of each iteration. Size of population in this phase (as well as in other generations) can range from hundreds to thousands of models[4]. In the simplest case, each model in initial population is created randomly, by using the activities, which occur in any trace in given log and creating random models

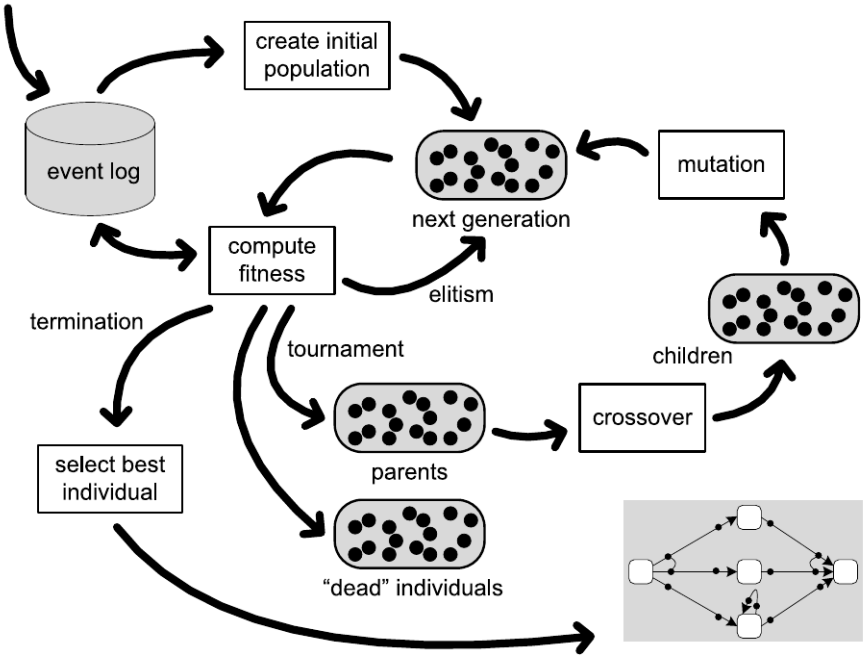


Fig. 6: Brief description of Genetic Mining[4]

consisting of those activities. Not surprisingly, the fitness of such models in initial population will be very low with respect to given event log. In [5] two different approaches are suggested, which can be used to define causal relation between each two activities, while constructing a model. The first approach considers the causality (or dependency) between two activities with the uniform distribution, the another (heuristic) approach does it by using the probability based on the information (behaviour) provided in given event log.

4.1.2 Selection At the beginning of selection phase, the quality of each model in population is determined. The quality must be related to the given event log and can be determined by different criteria, in particular, the four criteria, which are introduced in the previous section can be used. In Fig. 6 this step is described as *compute fitness*[6]. After this the models with high quality are selected and directly (without modification) moved to the next generation (*elitism* in Fig. 6). In order that a new generation can be created, as a next step "parent" models are selected (*tournament* in Fig. 6). Once again, different approaches can be used

[6]Fitness in this context means more than just the fitness criterion, which is introduced in the previous section. Here we are dealing with the term in general, which characterizes the quality of a model.

to select *"proper"*[7] parent models, e.g. Weijters et al.[5] select each parent model as follows: draw randomly 5 models from the current population and select the model with the best quality (fittest one[4]) among those models as a parent model. On the other hand, models with poor quality are no longer relevant, and therefore do not pass to the next generation, in Fig. 6 they are referred as "*dead*" individuals.

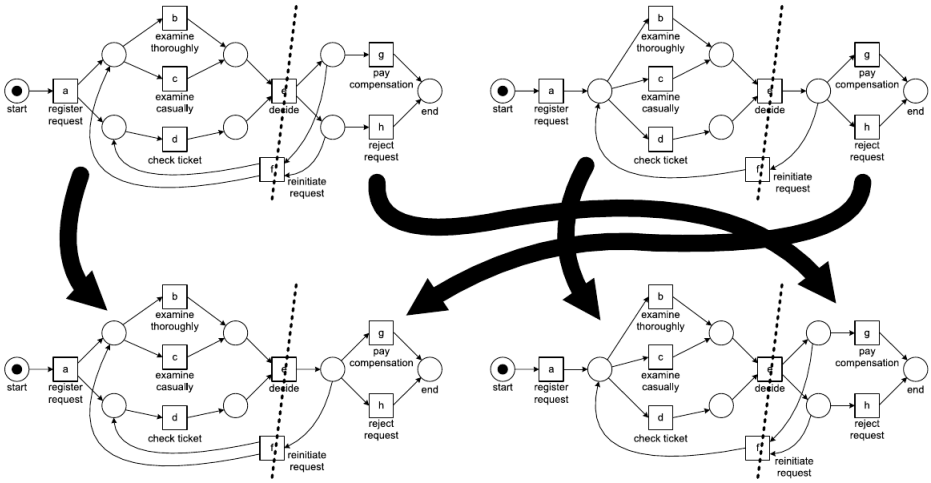


Fig. 7: Crossover of two Petri nets[4]

4.1.3 Reproduction After having selected parent models, the next step is to create new child models for the next generation, more precisely two new child models per each pair of parent models. This is done by applying *crossover* and *mutation*. Crossover takes two parent models, which contain the same activities, and chooses randomly one activity (or several activities). Next, two new child models are created by exchanging the parts of the parent models linked to this activity. For example, in Fig. 7 we have two parent Petri nets (models above), and activities d and e are chosen as crossover points. After swapping the input and output parts of those activities, we get new child Petri nets shown in Fig. 7 (models below). Those new created child models carry only genetic material of their parents. In order to add new genetic material to the new created models, mutation can be applied. The approach used during mutation depends on modelling language used for the representation of the models. The basic idea is to modify (add or remove) causal dependencies between randomly chosen activ-

[7] It is important that models having high quality do not get lost, but rather contribute to the next generation either directly or indirectly.

ities. After crossover and mutation is applied, the new generation is ready and next iteration can be started (with selection phase).

4.1.4 Termination In ideal case, Genetic Mining terminates when the desired level of quality is achieved. However, in certain situations the quality required for the termination might be too high, so that it can take to long to converge or it might be even impossible (e.g. because event log contains too few data) to reach the desired quality. The alternative break criteria to be used would be: break after at most n iterations or break if the last n generations do not yield a certain delta improvement. After termination, model with the best quality is returned as a final result.

Whereas Genetic Mining is very robust and flexible, its runtime might vary dramatically depending on input size[4].

4.2 Region-Based Mining

Region-Based (RB) Mining can be implemented with two different approaches: a) using *state-based* regions and b) using *language-based* regions. Both approaches deal with *synthesis* of Petri nets, i.e. constructing of a Petri net based on the described behaviour. The difference is that a) constructs a Petri net not directly from given log, but rather it firstly redescribes the behaviour in given log by creating appropriate transition system and then constructs a Petri net based on this transition system, whereas b) constructs a Petri net directly from given log. In this paper we cover only state-based regions. We refer the reader to [4] for more information on language-based regions.

As mentioned, state-based RB Mining is two-phase approach, i.e it consists of two phases, in the first phase a low-level model (transition system) is constructed and in the second phase based in this low-level model a high-level model (Petri net) is constructed.

There are different methods to derive transition system from given event log. In the following, we describe some of them by showing appropriate examples (see Fig. 8), where transition system is constructed for the event log

$$\mathrm{L} = [< a, b, c, d >^3, < a, c, b, d >^2, < a, e, d >][4].$$

Transition system in Fig. 8 (a) is constructed with a method, which defines the states of transition system based on the full history of activities in traces. For example, initial state is $<>$, because, all traces in L begin with empty sequence. $< a >$ is the only one state, which can be reached from initial state, by the transition *a*. If we take a look to the L, we see that all traces start with the activity *a*. The another method described in Fig. 8 (b) considers future of sequences, i.e. the opposite of the previous method. As we have three different traces in L, transition system in Fig. 8 (b) has three different initial states. The third method (Fig. 8 (c)) derives transition system based on multi-set property, which in contrast to the full history only considers the frequency of the past

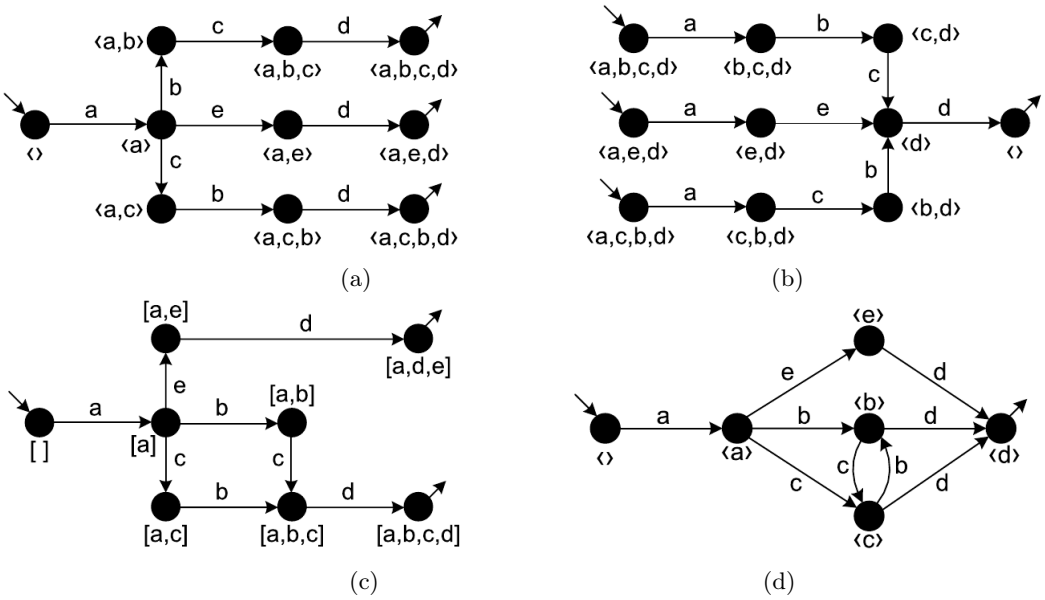


Fig. 8: Some examples of different approaches for deriving transition system from the log L = $[< a, b, c, d >^3, < a, c, b, d >^2, < a, e, d >]$[4]

activities, but not their order. For example, the both traces $< a, c, b, d >$ and $< a, b, c, d >$ end up in the final state $[a, b, c, d]$, because, if we ignore the order of the activities in those traces, then we get the same multi-set. The method used in Fig. 8 (d) considers only the last activity. This method can be easily extended to the one, which considers the last k activities in sequence. For instance, the first method described in Fig. 8 (a) would be such a method with k equal the length of the longest trace in the log. There are number of other methods, beyond the ones described here. It is important that particular attention be paid to selecting proper method, while implementing RB Mining. Because the abstraction level (e.g. coarse-grained vs. fine-grained) used during this phase has an influence on the quality of the final model, which is constructed in the next phase. If we take a look to the methods which are introduced, we see that e.g. method described in Fig. 8 (d) generalizes more than method described in Fig. 8 (a), i.e. it allows for more traces to be generated. Depending on the size (*completeness*) and the quality (*noise*) of the data in given event log, the balance between too specific and too general model can be accomplished with various methods, i.e there is no single method which always works best.

The reason why transition systems should not be final model is that the number of the states of transition system grows exponentially, when it comes to the concurrency, k parallel activities might imply in worst case 2^k states. Therefore, a high-level modelling language is needed. Thus, in the second phase a Petri net is constructed by mapping the *regions* of transition system to the

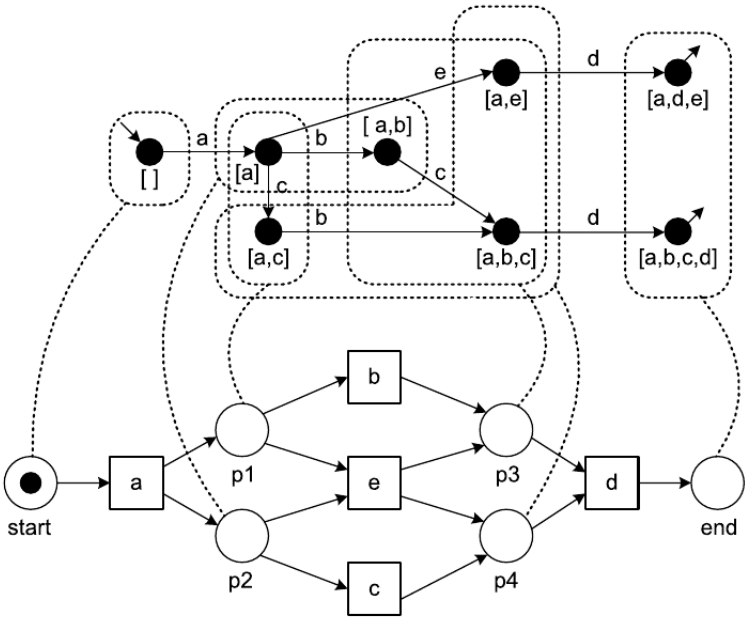


Fig. 9: Constructing of a Petri net by mapping the regions of the transition system (above) to the places of the Petri net (below)[4]

places of the Petri net. Let S be the set of all states of given transition system, the subset $R \subseteq S$ is a region, if for each activity a in transition system we observe one of the following cases: a) all transitions with the activity a *enter* R b) all transitions with the activity a *exit* R, c) all transitions with the activity a do not *cross* (i.e. neither enter, nor exit) R. For example, in Fig. 8 there are several regions (each region surrounded by dashed lines). Let's take a look at region R = $\{[a, b], [a, e], [a, b, c]\}$, all transitions with activities b and e enter R, all transitions with the activity c do not cross R and all transitions with the activity d exit R. The idea behind of the above definition is that while mapping one region to one place (of Petri net), we do not get any inconsistency regarding the transitions of the Petri net. For example, in Fig. 8 R is mapped to *p3*, and all transitions of Petri net are clearly defined with respect to *p3*.

5 Conclusion

The purpose of this paper was to describe the challenges, which must be addressed, while developing a Process Mining algorithm. Moreover, we described couple of advanced techniques, which use different approaches to deal with those challenges.

It was shown that depending on circumstances such as input data or process type the approaches to be used may vary or parameters may have to be adapted so that the best results can be achieved. In general, therefore, it seems that the underlying process or the quality of data in event log must be examined, before picking any Process Mining algorithm or adjusting parameters, in order to have an approximate idea about the challenges to be addressed in this specific case.

Furthermore, we presented four criteria, which can be used to assess the quality of the process models with respect to the underlying process and given event log. It is important to have a balance between generalization and precision, in order to avoid underfitting and overfitting models.

Whereas Genetic Mining emerged as robust technique, it has problem with the performance, which is not stable. Region-based Mining on the other hand, can internally use different approaches, depending on desired abstraction level.

As most of the current techniques suffer from many inconsistencies (i.e. they can possibly return internally inconsistent models), the issue of representational bias is an intriguing one which could be usefully explored in further research.

References

1. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. Knowledge and Data Engineering, IEEE Transactions on 16(9), 1128–1142 (2004)
2. Van der Aalst, W.M.: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
3. van der Aalst, W.M.: On the representational bias in process mining. In: Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on. pp. 2–7. IEEE (2011)
4. Aalst, W.v.d.: Process Mining : discovery, conformance and enhancement of business processes. Springer, Berlin (2011), http://digitool.hbz-nrw.de:1801/webclient/DeliveryManager?pid=4187738&custom_att_2=simple_viewer, erscheint: 16. Mai 2011
5. de Medeiros, A.K.A., Weijters, A.J., van der Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)
6. Van Der Aalst, W., Buijs, J., Van Dongen, B.: Towards improving the representational bias of process mining. In: Data-Driven Process Discovery and Analysis, pp. 39–54. Springer (2012)
7. Weijters, A., Ribeiro, J.: Flexible heuristics miner (fhm). In: Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on. pp. 310–317. IEEE (2011)
8. Wikipedia: Business process modeling — wikipedia, the free encyclopedia (2014), http://en.wikipedia.org/w/index.php?title=Business_process_modeling&oldid=605473809, [Online; accessed 27-May-2014]
9. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. Evolutionary Computation, IEEE Transactions on 1(1), 67–82 (1997)

Überblick menschlicher Aktivitätserkennung auf mobilen Geräten

Möglichkeiten, Grenzen und Herausforderungen

Florian Dittrich*

Betreuer: Anja Bachmann†

Karlsruher Institut für Technologie (KIT)
Pervasive Computing Systems – TECO
*uafac@student.kit.edu
†bachmann@teco.edu

Zusammenfassung. Mit der schnellen Entwicklung der heutigen Smartphones und deren Sensoren steigt auch die Relevanz des Forschungsgebiets der menschlichen Aktivitätserkennung. Dabei soll die Aktivität oder die Situation, in der sich der Benutzer befindet, über Sensoren am Smartphone erkannt und gegebenenfalls auch darauf reagiert werden. Dafür werden bekannte Machine Learning-Algorithmen eingesetzt. Diese Arbeit geht dabei hauptsächlich auf die aktuellen „Best Practices", die resultierenden Möglichkeiten und Anwendungsgebiete, sowie die entstehenden Herausforderungen und Grenzen näher ein.

Schlüsselwörter: menschliche Aktivitätserkennung, maschinelles Lernen, kontextsensitive Systeme

1 Einleitung

In der heutigen Zeit gelten mobile Geräte wie Smartphones als ständiger Begleiter im Alltag. Nahezu 80% der Weltbevölkerung besitzt ein Mobiltelefon und ein Drittel davon sind Smartphones [11]. Das zeigt, dass Handys in der heutigen Generation nicht mehr wegzudenken sind. Besonders die schnelle Entwicklung der Sensoren dieser Geräte bietet die Grundlage zahlreicher Anwendungsgebiete. Da schon fundierte Kenntnisse über maschinelles Lernen und Data Mining existieren, profitiert die menschliche Aktivitätserkennung von dieser, da diese Kenntnisse eingesetzt werden. Die menschliche Aktivitätserkennung kombiniert somit schon bekannte Forschungsthemen mit modernen Technologien der Hardware. Daraus ergibt sich für den Benutzer einen erheblichen Mehrwert, da die Aktivität und der Kontext des Benutzers erkannt werden und darauf reagiert werden kann. Falls ein Benutzer zum Beispiel eine SMS lesen möchte, während er gerade zu Fuß unterwegs ist, könnte das Smartphone das erkennen und die Schrift vergrößern, sodass der Benutzer die Schrift besser lesen kann. Um dies umzusetzen, musste der Benutzer früher verschiedene Sensoren an unterschiedlichen Körperpositionen anbringen, die den Benutzer möglicherweise stören. Heutzutage sind

die wichtigsten dieser Sensoren im Smartphone integriert, wodurch der Benutzer es nur noch am Körper tragen muss und somit keine Einschränkungen im Alltag erfährt.
Diese Arbeit beschäftigt sich mit dem aktuellen Stand der Technik im Bereich menschlicher Aktivitätserkennung. Dabei wird die Wahl der Sensoren, sowie Feature Extraction/Selection und auch die Wahl des Klassifikators diskutiert. Diese drei Teile sind wichtige Erfolgsfaktoren für die Klassifikation und beeinflussen die Qualität der menschlichen Aktivitätserkennung in großem Maße. Außerdem wird auf Möglichkeiten und Anwendungsgebiete eingegangen, die das Gebiet der menschlichen Aktivitätserkennung eröffnet. Diese lassen sich in Anwendungen für Endbenutzer, für Entwickler und Unternehmen, sowie für Gruppen kategorisieren. Zuletzt stehen den Möglichkeiten und Anwendungsgebieten noch die Herausforderungen und Grenzen gegenüber. Es werden hierbei Schwierigkeiten und zu beachtende Probleme beschrieben, für die es teilweise Lösungsansätze gibt, die teilweise aber noch ungelöst sind. Im Vorfeld werden allerdings zuerst Grundlagen, wie der Begriff der Klassifikation und des Features erklärt und einen Überblick über die gängigen Klassifikationsalgorithmen gegeben.
Im Folgenden wird diese Arbeit zuerst in Kapitel 2 mit den Grundlagen der menschlichen Aktivitätserkennung und des maschinellen Lernens fortgeführt. Anschließend werden die aktuellen „Best Practices“ in Kapitel 3 erläutert. Darauffolgend werden in Kapitel 4 die Chancen und Möglichkeiten, sowie Anwendungsgebiete, die das Forschungsgebiet mit sich bringt, betrachtet. Im Gegensatz dazu sind in Kapitel 5 die Herausforderungen und Grenzen zu finden. Abschließend möchte ich die Erkenntnisse dieser Arbeit in Kapitel 6 noch einmal zusammenfassen und einen Ausblick für die Zukunft geben.

2 Grundlagen

Bevor der aktuellen Stand der Technik und die charakteristischen Punkte der menschlichen Aktivitätserkennung vorgestellt werden, werden zuerst einige Grundlagen behandelt.

2.1 Klassifikation

Der wichtigste Schritt der menschlichen Aktivitätserkennung ist die Klassifikation. Hierbei wird versucht, vorhandene Daten zu Klassen zusammenzufassen, um später neue Datensätze diesen Klassen richtig zuordnen zu können.

Die Klassifikation verläuft im Allgemeinen in drei Schritten. Zuerst werden Rohdaten von möglichst vielen Testpersonen aufgenommen. Diese können von unterschiedlichen Sensoren stammen. Damit eine möglichst hohe Qualität der Klassifikation für viele Endbenutzer erreicht werden kann, sollten auch viele Trainingsdaten von vielen unterschiedlichen Testpersonen benutzt werden. Allerdings sollte auch darauf geachtet werden, dass in jeder Klasse eine ähnlich

Abb. 1. Ablauf der Klassifikation, bestehend aus den Schritten Datenakquise, Feature Selection und Klassifikation (siehe: [3]).

große Anzahl von Testdaten existiert. Ansonsten kann es ein, dass eine Klasse übertrainiert wird (*engl. overfitting*) und dadurch eher erkannt wird als andere Klassen, wodurch sich das Klassifikationsergebnis verschlechtert.
Danach wird die große Menge an Rohdaten auf eine kleinere Menge von möglichst aussagekräftigen Werten reduziert. Diese Werte bilden Eigenschaften (*engl. feature*) des Signals ab. Diesen Schritt nennt man Feature Extraction. Da manche dieser extrahierten Features redundant oder auch irrelevant für die Klassifikation sein können, versucht man diese Menge noch einmal einzugrenzen, was als Feature Selection bezeichnet wird. Diese Features werden häufig nicht über die Gesamtmenge der Daten berechnet. Stattdessen werden die Daten zuerst in Zeitfenster einer festen Länge unterteilt, die sich meist noch überlappen. Anschließend werden die gewählten Features über diese Fenster berechnet. Dieser Ansatz heißt Sliding-Window-Ansatz. Diese Bestandteile der Klassifikation werden in Abb. 1 im Punkt „Feature Selection“ zusammengefasst.
Im Allgemeinen wird überwachtes maschinelles Lernen (*engl. supervised machine learning*) verwendet, d.h. dass ein Datensatz mit seiner zugehörigen Klasse beschriftet wird und somit bekannt ist, zu welcher Klasse die aufgenommenen Daten gehören. Diese Menge von Klassen kann dann bei der Klassifikation erkannt werden.
Im letzten Schritt wird mit Hilfe des Klassifikators das gewünschte Ergebnis berechnet. Der Klassifikator ist ein Algorithmus, der ein Modell erzeugt. Dieses Modell erhält als Eingabe die Features eines zu klassifizierenden Datensatzes und gibt im Anschluss die zugehörige Klasse aus. Durch die gesammelten Trainingsdaten erstellt der Algorithmus also eine Repräsentation der Welt, in der all die Klassen erkannt werden können, die zuvor mit den Trainingsdaten erlernt wurden. Dieses Verfahren wird im Punkt „Classification“ in Abb. 1 dargestellt.
Es gibt verschiedene Arten von Klassifikatoren die in Kapitel 2.3 noch erklärt werden. Zunächst möchte ich allerdings auf den Begriff des Features näher eingehen.

2.2 Features

Ein wichtiger Begriff der Klassifikation ist der des Features. Features sind im Kontext der Klassifikation Transformationen bzw. Umwandlungen der Rohdaten. Ein einfaches Beispiel für ein Feature wäre der Durchschnitt. Die Rohdaten werden dabei auf einzelne Werte abgebildet, die möglichst eine signifikante Aus-

sage treffen. Wie gut eine Klassifikation funktioniert, hängt also in großem Maße von der Wahl der Features ab. Um sicher zu stellen, dass die richtigen Features ausgewählt wurden, existieren einige Verfahren wie die Principal Component Analysis. Dieses statistische Verfahren benutzt eine orthogonale Transformation, um von einer Menge von Features, die möglicherweise stark korreliert, auf eine Menge von unkorrelierten Features zu schließen. Zusätzlich kann es zur Reduzierung der Dimensionen genutzt werden [10]. Die Klassifikation kann allerdings auch verbessert werden indem die Abtastrate der Sensoren erhöht wird (Upsampling) oder indem Sliding-Windows ggf. mit Überlappung verwendet werden. Welche Features ein positives Resultat für die Klassifikation bedeuten, ist von Anwendungsfall zu Anwendungsfall unterschiedlich. Die geeigneten Features im Kontext der menschlichen Aktivitätserkennung werden in Kapitel 3.3 noch erläutert. Um aus diesen Features nun ein Klassifikationsmodell zu erstellen, wird ein Klassifikator benötigt. Dieser wird im Folgenden erklärt.

2.3 Klassifikator

Der Klassifikationsalgorithmus ist das Herzstück der Klassifikation und hat einen großen Einfluss auf das Ergebnis der Klassifikation. Allgemein arbeitet ein Klassifikator nach dem Prinzip des Konzeptlernens (*engl. concept learning*). Das heißt, er versucht aus Beispielen eine Unterscheidung der einzelnen Klassen zu lernen. Ein Konzept C wird also mittels einiger positiver Beispiele $x \in C$ und einiger negativer Beispiele $\bar{x} \notin C$ erlernt, sodass dieses Konzept erkannt werden kann. Damit dieses Konzept auch von anderen unterschieden werden kann, müssen gewisse Attribute definiert werden, anhand derer sich das Konzept abgrenzt. Jeder Klassifikator erhält also eine Sammlung von Beispielen und soll nun ein Modell erzeugen, welches mit zukünftigen Daten angewendet werden kann. Wie dieser Klassifikator dieses Modell erzeugt, ist von Klassifikator zu Klassifikator unterschiedlich. Ich möchte nun ein paar der wichtigsten Klassifikatoren erklären.

Naive Bayes Der Naive Bayes Klassifikator ist ein statistisches Verfahren, das auf dem Satz von Bayes beruht:

$$P(w_i|x) = \frac{P(x|w_i) \cdot P(w_i)}{P(x)}. \tag{2.3.1}$$

Hierbei wird für jede Klasse die Wahrscheinlichkeit berechnet, ob der gegebene Datensatz dazu gehört oder nicht. Das Ergebnis ist dann die Klasse mit der größten Wahrscheinlichkeit.

Entscheidungsbaum Ein Entscheidungsbaum besteht aus einem Wurzelknoten, aus inneren Knoten und aus Blattknoten. Bei der Klassifikation steigt man ausgehend von der Wurzel sukzessive ab. Jeder innere Knoten repräsentiert dabei ein Attribut anhand dessen entschieden wird, zu welcher Klasse der zu klassifizierende Datensatz gehört. Die Blätter des Baumes sind die einzelnen Klassen,

die der Klassifikator erkennen kann. In Abb. 2 ist ein Entscheidungsbaum dargestellt, der erkennt, ob ein Apfelbaum Früchte tragen kann. Entschieden wird dabei anhand der Attribute Alter, Sorte und Boden. Wenn der Algorithmus beim Traversieren des Baumes an einem Blatt angekommen ist, wird die entsprechende Beschriftung (*engl. label*) der Klasse als Ergebnis zurückgegeben.

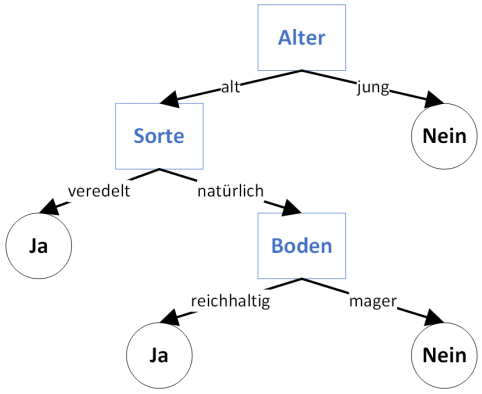


Abb. 2. Ein Entscheidungsbaum, der klassifiziert, ob ein Apfelbaum Früchte tragen kann oder nicht.

k-Nearest Neighbor Der k-Nearest Neighbor Klassifikator betrachtet die k nächsten Nachbarn des zu klassifizierenden Datenpunktes x. Um die Nähe der Nachbarn definieren zu können, muss zuerst eine Distanzfunktion gewählt werden. Üblicherweise wird hierfür die Euklidische Norm gewählt. In Abb. 3 wird x als grünes Dreieck dargestellt und $k = 7$ gesetzt. Es gibt drei Klassen, rote Punkte, schwarze Sterne und blaue Rauten. x wird als die Klasse klassifiziert, die am häufigsten in den k Nachbarn auftritt. Im Beispiel wäre x also als roter Punkt erkannt worden. Die Herausforderung ist hauptsächlich, das k richtig zu wählen. Denn bei einem kleinen k kann die Klassifikation schlechte Ergebnisse liefern, falls in den Trainingsdaten ein zu großes Rauschen herrscht. Allerdings können bei zu großem k Daten mit einem zu großen Abstand zu x berücksichtigt werden.

Support Vector Maschine Die Support Vector Maschine (SVM) versucht wie in Abb. 4 die gegebenen Trainingsdaten voneinander zu trennen. Dafür werden eine Hyperebenen mit maximalem Abstand zu den Klassen gesucht. Im Beispiel befinden sich die Daten der Einfachheit halber im zweidimensionalen Raum. Im einfachsten Fall sind die Daten ohne Transformation linear trennbar. Wie in Abb. 4(a) zu sehen ist, können die Daten dann durch eine Gerade separiert

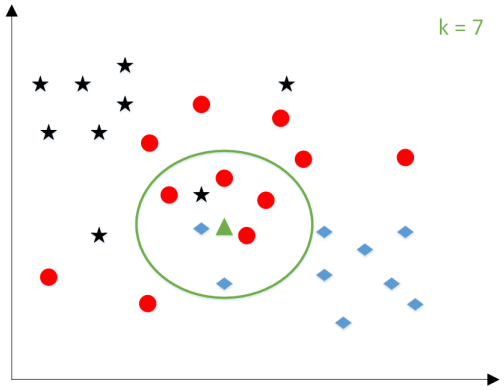
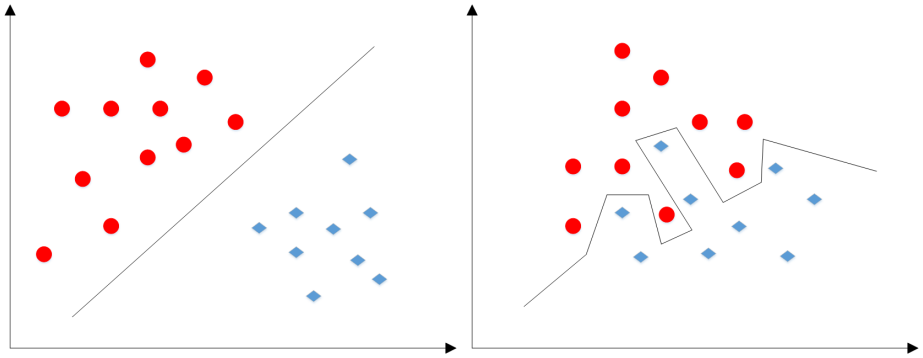


Abb. 3. k-Nearest Neighbor mit k=7 und den drei Klassen: Stern, Punkt, Raute.

werden. Da dies nicht immer der Fall ist, wie zum Beispiel in Abb. 4(b), müssen oft komplexere Trennmethoden durchgeführt werden. Der Raum wird dafür, unter Nutzung einer Kernel-Funktion, in eine höhere Dimension transformiert, sodass die Trainingsdaten durch eine (Hyper-) Ebene möglicherweise voneinander getrennt werden können. Es ist dadurch möglich, dass sich die Laufzeit stark vergrößert.



(a) Linear trennbar (b) Nicht linear trennbar

Abb. 4. Separierbarkeit bei SVM Klassifikator. In Abbildung 4(a) sind Datenpunkte zweier Klassen abgebildet, die linear voneinander trennbar sind. Im Gegensatz dazu sind die Datenpunkte in Abbildung 4(b) nicht linear voneinander trennbar. Um eine lineare Separierbarkeit zu ermöglichen, müssen sie zuerst in eine höhere Dimension transformiert werden.

Hidden Markov Modelle Ein Hidden Markov Modell besteht aus einer Menge von Zuständen und ihren Zustandsübergängen. Jeder Zustandsübergang hat eine Wahrscheinlichkeit. Es wird allerdings angenommen, dass die Markov-Eigenschaft gilt. Diese besagt, dass nur der aktuelle Zustand einen Einfluss auf den Folgezustand hat. Außerdem sind die Zustände verborgen (*engl. hidden*), das heißt, dass sie von außen nicht mess- bzw beobachtbar sind. Jeder Zustand kann allerdings sichtbare Emissionen (beobachtbare Ausgabesymbole) ausgeben. Welche Emission ausgegeben wird, hängt nur von den Wahrscheinlichkeiten ab. Ein Hidden Markov Modell ist also ein 5-Tupel $\lambda = (S, V, A, B, \pi)$ mit der Menge S der Zustände, der Menge V der Ausgabesymbole (Emissionen), den Übergangswahrscheinlichkeiten A, den Emissionswahrscheinlichkeiten B und den Initialwahrscheinlicheiten π, die besagen, welcher Zustand als Startzustand gewählt wird. Als Beispiel ist in Abb. 5 ein HMM für einen Gefangenen dargestellt, der wissen möchte, ob es draußen regnet oder sonnig ist. Er weiß, dass nach einem sonnigen Tag zu 70 % ein regnerischer Tag folgt. Umgekehrt weiß er, dass nach einem regnerischen Tag zu 50% ein sonniger kommt. Zusätzlich weiß er, dass die Schuhe der Wärter bei einem sonnigen Tag zu 40% sauber und zu 60% dreckig sind. Hingegen sind die Schuhe bei einem regnerischen Tag zu 10% sauber und zu 90% dreckig. Die verborgenen Zustände des Wetters, lassen sich also durch die sichtbaren Emissionen der Schuhe der Wärter klassifizieren.

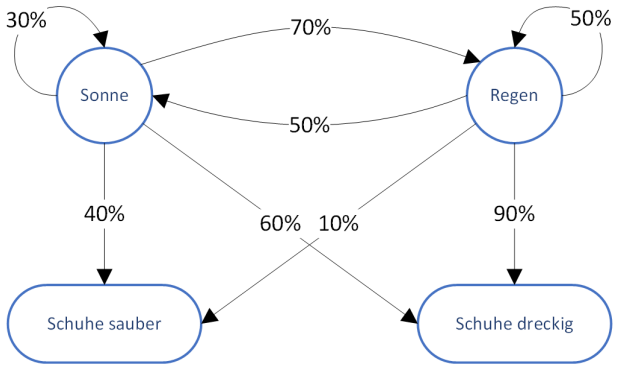


Abb. 5. HMM für Wettervorhersage eines Gefangenen. Die Kreise stellen dabei die Menge der Zustände dar. Zwischen ihnen verlaufen die Übergangswahrscheinlichkeiten. Die Ellipsen zeigen die Menge der Emissionen und die eingehenden Kanten sind die Emissionswahrscheinlichkeiten.

Es existieren noch viele andere Klassifikationsalgorithmen wie Beispielsweise künstliche Neuronale Netze, die in dieser Arbeit nicht näher erläutert werden.

2.4 Güte der Klassifikation

Die Güte der Klassifikation kann mit vielen unterschiedlichen Maßen angegeben werden. In der Literatur werden häufig F-Score, Accuracy und andere verwendet, die ich nun einführen möchte. Der Einfachheit halber wird in diesem Beispiel eine Wahrheitsmatrix mit nur zwei Klassen (positiv und negativ) betrachtet (Binäre Klassifikation). Allgemein können mit Wahrheitsmatrizen allerdings auch bei vielen Klassen die tatsächlichen Klassen mit den klassifizierten Klassen gegenübergestellt werden. Es kann so schnell abgelesen werden, welche Klassen vom Klassifikator „verwechselt“ wurden.

Tabelle 1. Wahrheitsmatrix einer binären Klassifikation für n Datenpunkte.

	positiv klassifiziert	**negativ klassifiziert**	Σ
tatsächlich positiv	True Positive (TP)	False Negative (FN)	TP+FN
tatsächlich negativ	False Positive (FP)	True Negative (TN)	FP+FN
Σ	TP + FP	FN + TN	n

Nachfolgend werden die unterschiedlichen Gütemaße vorgestellt und aufgezeigt, wie sie auf Basis der Werte aus Tabelle 1 berechnet werden.

- Die *precision* gibt an, wie viele der positiv klassifizierten Daten auch tatsächlich positiv sind.
$$precision = \frac{TP}{TP + FP} \tag{2.4.1}$$
- Der *recall* oder auch Sensitivität gibt an, wie viele der tatsächlich positiven Daten auch positiv klassifiziert wurden.
$$recall = \frac{TP}{TP + FN} \tag{2.4.2}$$
- Der $F - Score$ verbindet *precision* und *recall* mit dem gewichteten Harmonischen Mittel und ist somit ein kombiniertes Gütemaß.
$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.4.3}$$
- Die *accuracy* betrachtet das Verhältnis der klassifizierten Ergebnisse, zu allen gemessenen Werten.
$$accuracy = \frac{TP + TF}{n} \tag{2.4.4}$$
- Analog zu dem *recall* gibt die $specifity$ den Anteil der korrekt negativ klassifizierten Datensätze an allen tatsächlich negativen Datensätzen an.
$$specifity = \frac{TN}{FN + TN} \tag{2.4.5}$$

– Die *error rate* ist das Verhältnis zwischen der Summe aller falsch klassifizierten Datensätzen und der Gesamtheit aller Datensätze.

$$error\ rate = \frac{FN + FP}{n} \tag{2.4.6}$$

Üblicherweise werden alle Maße außer dem F-Score in Prozent angegeben, der F-Score hingegen als Gleitkommazahl zwischen 0 und 1.

Für eine gute Klassifikation ist immer darauf zu achten, dass die Werte TP und TN so groß wie möglich und dementsprechend die Werte FP und FN so klein wie möglich sind. Das bedeutet, dass Maße wie die Accuracy und der F-Score maximiert und Maße wie die error-rate minimiert werden müssen, um eine gute Klassifikation zu erreichen.

In der Literatur werden Testergebnisse von Experimenten auch häufig mit einer Wahrheitsmatrix (*engl. confusion matrix*) dargestellt. Wie in Abb. 6 zu sehen ist, werden die Aktivitäten „Walk, Jog, Stairs, Sit, Stand“ betrachtet. Auf der Diagonalen befinden sich die richtig klassifizierten Testergebnisse und auf der oberen- und unteren Dreiecksmatrix können wie in Tabelle 1 die Fehlerklassifikation in Form von False Negative (FN) bzw. False Postivie (FP) abgelesen werden. Für Walk gilt also $FN = 7 + 148 + 2 + 2$ und $FP = 10 + 185 + 4 + 3$.

		Predicted Class					Accur. (%)
		Walk	Jog	Stairs	Sit	Stand	
Actual Class	Walk	**1524**	7	148	2	2	90.6
	Jog	10	**1280**	31	0	0	96.9
	Stairs	185	33	**784**	4	4	77.6
	Sit	4	0	2	**272**	4	96.5
	Stand	3	1	10	0	**209**	93.7

Abb. 6. Wahrheitsmatrix für die Aktivitäten walking, jogging, walking stairs, sitting, standing zur Visualisierung einer Wahrheitsmatrix mit mehreren Klassen (siehe: [5]).

3 Best Practices

Das Forschungsgebiet des Maschinellen Lernens (*engl. machine learning*), auf das die menschlichen Aktivitätserkennung aufbaut, ist ein etabliertes Forschungsgebiet, dass schon seit Jahrzehnten existiert. Demnach lassen sich auf das viel jün-

gere Forschungsgebiet der menschlichen Aktivitätserkennung und der kontextsensitiven Systeme viele Prinzipien und Erkenntnisse übertragen. Im Folgenden möchte ich auf diese „Best Practices“ genauer eingehen.

3.1 Sensoren

Die Vielfalt an Sensoren heutiger Smartphones und anderer mobiler Geräte lässt nahezu keine Wünsche mehr offen. Anfangs wurden diese Sensoren nur zur Unterscheidung, ob das Smartphone gerade vertikal oder horizontal gehalten wird verwendet oder um beispielsweise das Lenken bei Spielen zu vereinfachen [5]. Allerdings wurde schnell erkannt, wie viel Potenzial für innovative Applikationen in diesen Sensoren steckt und die Aufmerksamkeit der Entwickler wurde größer. Aktuelle Smartphones sind mit zahlreichen Sensoren ausgestattet. Beispielsweise werden in den Spezifikationen des Google Nexus 5 folgende Sensoren genannt: „GPS, Gyroskop, Beschleunigungsmesser, Kompass, Näherungs- bzw. Umgebungslichtsensor, Barometer, Hall“ [2]. Der Kompass wird im Folgenden als Magnetometer bezeichnet. Für die menschliche Aktivitätserkennung hat sich in den letzten Jahren hauptsächlich der in Abb. 7 dargestellte 3-Achsen Beschleunigungssensor durchgesetzt. Allerdings werden auch das Gyroskop und das Magnetometer häufig verwendet. Das Gyroskop misst Winkelgeschwindigkeiten um die Koordinatenachsen, wohingegen das Magnetometer wie ein bekannter Kompass arbeitet. Damit lassen sich die Lage und Neigung des Smartphones bestimmen.

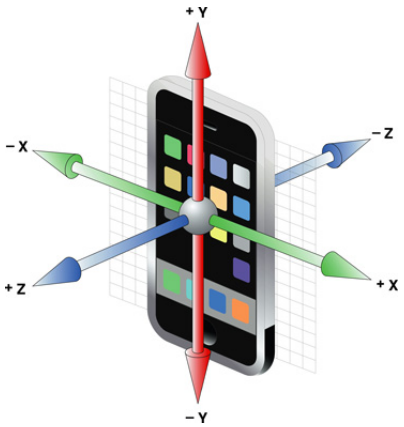


Abb. 7. Veranschaulichung des 3-Achsen Beschleunigungssensor. Er misst die Beschleunigung in den eingezeichneten Achsen. Maßeinheit ist dabei in $\frac{m}{s^2}$. (siehe: [1])

3.2 Abtastrate und Fenstergröße

Wie schon in Kapitel 2.2 angesprochen, sind auch die Abtastrate und die Fenstergröße wichtige Stellhebel der Klassifikation. Deshalb ist es wichtig, dass die Abtastfrequenz, die Fenstergröße und auch die Überlappung der Fenster genau auf den Anwendungsfall abgestimmt sind. Da dieser Bereich im Kontext der menschlichen Aktivitätserkennung sehr breit gefächert ist, stellt das eine große Herausforderung dar.
Anguita et al. [3] haben sich beispielsweise für eine Abtastfrequenz von 50 Hz und eine Fenstergröße von 2,56 Sekunden entschieden. Die Fenster überlappen sich dabei jeweils um die Hälfte. Sun et al. [13] haben versucht, anhand ihres Experiments und ihrer Trainingsdaten die beste Fenstergröße zu ermitteln. Das Ergebnis dabei waren 4 Sekunden, wobei der F-Score als Genauigkeitsmaß benutzt wurde. Als Abtastrate verwenden sie 10 Hz. Lee et al. [6] benutzen 5 sekündige Fenster und eine Abtastfrequenz von 12 Hz.
Im Experiment von Maurer et al. [9] wurde versucht die optimale Abtastrate der Sensoren zu bestimmen. Dabei haben sie auch die unterschiedlichen Trageposition am Körper in Betracht gezogen. Sie erhielten stabil gute Ergebnisse bei einer Frequenz von 15 - 20 Hz, die auch bei einer höheren Abtastrate nicht mehr signifikant verbessert wurden.

3.3 Features

Die Wahl der Features hat einen sehr großen Einfluss auf die Güte der Klassifikation. Deshalb ist es auch sehr wichtig die Features auf die Anwendungsfälle abzustimmen. Da diese im Bereich der menschlichen Aktivitätserkennung sehr breit gefächert sind, ist es entsprechend schwer sie zu bestimmen.
Falls als Sensor der Beschleunigungssensor mit mehreren Achsen gewählt wurde, werden oftmals die Features über die Werte aller Achsen berechnet. Allerdings gibt es auch Ausnahmen wie Ustev et al. [14]. Sie berechnen zuerst die Euklidische Norm über die drei Werte und berechnen danach die Features davon.

Nahezu alle Wissenschaftlichen Arbeiten benutzen zeitabhängige Features. Beispielsweise verwenden Kwapisz et al. [5] „Average, Standard Deviation, Average Absolute Difference, Average Resultant Acceleration, Time between Peaks, Binned Distribution“, wobei sie eine Genauigkeit von über 90% bei einfachen Aktivitäten wie Laufen und Joggen erreichten. Diese Features werden ausschließlich über den Zeitbereich berechnet. Bei schwieriger zu erkennenden Aktivitäten wie Treppensteigen, stagniert die Accuracy allerdings bei 78%.
Um die Accuracy noch weiter zu steigern werden häufig noch frequenzabhängige Features hinzugezogen, die in den meisten Fällen die Qualität der Klassifikation noch einmal erhöhen. Dazu muss zuerst eine Diskrete Fourier Transformation durchgeführt werden, um die Beschleunigungswerte auf ihr Frequenzspektrum abzubilden. Danach können Features im Frequenzbereich berechnet werden. Gerade bei komplexeren Bewegungen wie Treppensteigen, die eine Beschleunigung

auf mehreren Achsen aufweisen, können diese Features die Klassifikation verbessern. Gängige frequenzabhängige Features sind Zero Crossing Rate, Max. Correlation Value, Period (Index of Max. Correlation) [14] oder Correlation, FFT Energy and Frequency-Domain Entropy [13]. Im Experiment von Ustev et al. [14] wurde mit zeitabhängigen Features eine Accuracy von 83 % und mit einer erweiterten Menge von Features, die auch frequenzabhängige Features enthielt, eine Accuracy von 85% erreicht.

3.4 Klassifikator

Wie schon in Kapitel 2.3 erwähnt gibt es viele verschiedene Klassifikatoren, die sich für unterschiedliche Anwendungen besonders gut eignen. Hidden Markov Modelle, werden beispielsweise oft in der Spracherkennung eingesetzt.
Um herauszufinden, welcher Klassifikationsalgorithmus für die menschliche Aktivitätserkennung geeignet ist, haben Kwapisz et al. [5] ein Experiment durchgeführt. Dabei untersuchten sie, welcher Klassifikator bei gleichen Trainingsdaten die besten Ergebnisse liefert. Getestet wurden der Entscheidungsbaum, logistische Regression und Neuronale Netze. Bei den Aktivitäten „walking, jogging, ascending stairs, descending stairs, sitting, and standing“, die sie untersuchten, bekamen sie bei den Neuronalen Netzen die besten Ergebnisse.
Gute Ergebnisse lieferten auch die häufig eingesetzten Support Vector Machines (SVMs). Sun et al. [13] erreichten damit eine maximale Accuracy von 94,8 %, da sie zusätzlich noch die Orientierung, also die Ausrichtung in der Tasche (Abb. 8), des Smartphones berücksichtigen. Auch Shin et al. [11] erhalten mit der SVM gute Ergebnisse mit 89,6% Accuracy und einem F-Score von 0,707, allerdings war das Anwendungsgebiet hauptsächlich die Erkennung von problematischer Benutzung von Smartphones. Da die SVM zwar häufig gute Ergebnisse liefert, aber auch, durch Anwendung von Kernelfunktionen, sehr kostenintensiv ist, haben Anguita et al. [3] eine hardwarefreundliche SVM entwickelt, die trotzdem ähnlich gute Ergebnisse liefert. Ustev et al. [14] verwendeten für ihre Applikation den k-Nearest Neighbor Algorithmus, mit dem sie am Ende bei einer Accuracy von 97% erreichten. Auch sie haben die Orientierung des Smartphones berücksichtigt und konnten die Qualität der Klassifikation dadurch noch einmal verbessern.

4 Möglichkeiten und Anwendungsgebiete

Die menschliche Aktivitätserkennung bringt ein riesiges Feld an Möglichkeiten und Anwendungsgebieten mit sich. Es lassen sich enorme Vorteile für den Endbenutzer aufzählen, die ihm den Alltag erheblich erleichtern und ein großes Maß an Komfort bereitstellen. Außerdem kann menschliche Aktivitätserkennung auch für die Unternehmen in der Industrie oder auch für große Gruppen und Crowds von Nutzen sein. Im Folgenden möchte ich diese Möglichkeiten und Anwendungsgebiete, wie auch schon in der wissenschaftlichen Arbeit von Lockhart et al. [7] beschrieben, anhand dieser drei Segmente verdeutlichen.

4.1 Anwendungen für Endbenutzer

Besonders im Gesundheits- und Fitness Bereich findet die menschliche Aktivitätserkennung viele Einsatzgebiete.
Um den aktuellen Fitnessstand zu überprüfen, besteht dabei die Möglichkeit, täglich die verbrannten Kalorien zu zählen oder die gelaufenen Schritte und Stufen zu dokumentieren. Dem Benutzer können dann wöchentliche oder monatliche Fitnessreports angezeigt werden, die mit den gesetzten Zielen verglichen werden. Das kann einerseits die Motivation für körperliche und sportliche Aktivität steigern, andererseits aber auch als Erinnerung für mehr sportliche Betätigung eingesetzt werden. Der Benutzer hat dabei den Vorteil, genau ablesen zu können, auf welchem Stand er sich befindet und wie er sich im Lauf der Zeit verändert hat.
Des Weiteren können Menschen mit gesundheitlichen Beschwerden, bei denen der Arzt keine genaue Diagnose stellen kann, beobachtet und Daten gesammelt werden. Der Arzt ist dann in der Lage, die Aktivität des Patienten mit anderen zu vergleichen, um mögliche Unterschiede zu erkennen. Er hat dabei einen besseren Einblick in den Alltag des Patienten und kann Symptome besser deuten als durch einfache Gespräche. Zusätzlich stehen Messwerte zur Verfügung, die als Grundlage für weitere Untersuchungen dienen können.
Vorwiegend bei älteren und geschwächten Menschen ist es oft notwendig, betreutes Wohnen einzusetzen. Da hierfür die finanziellen Mittel allerdings häufig nicht ausreichen, könnte die menschliche Aktivitätserkennung eingesetzt werden. Die Menschen können ihr gewohntes Leben in ihrem sozialen Umfeld weiterführen und sind nicht gezwungen, sich in eine entsprechende Einrichtung zu begeben. Außerdem kann bei Unfällen wie Stürzen über das System automatisch ein Notruf gesendet werden, sodass die Sicherheit der Patienten gewährleistet ist.

Neben den Einsatzmöglichkeiten der menschlichen Aktivitätserkennung im Gesundheitssektor gibt es noch zahlreiche Anwendungen, die versuchen dem Endbenutzer den Alltag zu erleichtern. Dazu gehört zum Beispiel der Bereich der kontextsensitiven Systeme. Hier wird versucht den Benutzer in seiner aktuellen Situation und Aktivität zu unterstützen. Beispiele dafür wären einerseits, dass das Smartphone auf Stumm geschaltet wird, wenn sich der Benutzer bei der Arbeit oder in der Universität befindet. Andererseits aber auch, dass beispielsweise Schrift auf dem Display größer wird, wenn erkannt wird, dass der Benutzer in Bewegung ist. Zusätzlich gibt es die sogenannten „Selfmanaging Systems“, die nicht direkt für den Benutzer gedacht sind, sondern eher dazu dienen, Ressourcen des Smartphones zu sparen. Es wird also berücksichtigt, dass GPS nicht benötigt wird, wenn sich der Benutzer nicht mehr bewegt, oder dass Wi-Fi unterwegs ausgeschaltet werden kann. Lockhart et al. [7] geben an, dass dadurch die Akkulaufzeit um das fünffache verlängert werden kann.

4.2 Anwendungen für Unternehmen und Dritte

Unternehmen und deren Vertriebsabteilung haben den Wunsch ihre Werbung so gezielt wie möglich an den Benutzer zu übermitteln. Da liegt der Gedanke

nahe, die Situation und die Aktivität eines Menschen mit einzubeziehen. Dieses Anwendungsgebiet nennt sich „Targeted Advertising“ und beschäftigt mit dem schalten von zielgerichteter Werbung, sodass eine größtmögliche Effektivität aus der Werbung gezogen werden kann.
Auf der anderen Seite benötigen Unternehmen, insbesondere Forschungsunternehmen, häufig zuerst eine Grundlage an Daten. Es existiert also ein Markt für aufgenommene Rohdaten, die mit Aktivitäten beschriftet sind. Besonders relevant sind diese Rohdaten, wenn man sie mit Umfragen und Fragebögen kombinieren kann. Dadurch erhalten die Forschungsunternehmen große Mengen an Informationen, mit denen sie arbeiten können.
Darüber hinaus gibt es noch andere Ideen, wie beispielsweise ein Programm einer Versicherungen, in dem sich Freiwillige im Straßenverkehr durch die menschliche Aktivitätserkennung aufnehmen lassen können und ggf. bei sicherem Fahren einen Nachlass auf den Versicherungsbeitrag gewährt bekommen.

4.3 Anwendungen für Gruppen und Crowds

In Sozialen Netzwerken werden Benutzer ermutigt zu teilen, was sie gerade tun oder in welcher Situation sie sich befinden. Auch hier liegt es nahe, menschliche Aktivitätserkennung einzubeziehen. Benutzer können automatisch auf Twitter, Facebook oder anderen Plattformen ihren Status updaten lassen. Dadurch bleiben ihr Profil und damit auch ihre Freunde/Follower immer auf dem neusten Stand. Als Erweiterung kann auf den Sozialen Netzwerken, mit der eigenen Aktivität als Grundlage, nach anderen Personen im Freundeskreis gesucht werden, die das Gleiche tun. Es könnten Vorschläge gegeben werden in der Art: „Felix geht häufig in ihrer Nähe joggen.“, wodurch man sich einfacher in Interessengruppen zusammen finden kann.
Interessant ist auch, dass über Aktivitätsinformationen mit Standortdaten von genügend Personen bestimmte Regionen ausgemacht werden können, in denen eine gewisse Aktivität gut auszuüben ist. Damit können Menschen, die gerne Radfahren, bestimmte Routen in der Nähe vorgeschlagen werden, die viel gefahren werden. Dies nennt sich „Activity-based Crowdsourcing“. Diese Systeme können auch erkennen, wenn sich in einer bestimmten Region die Aktivitäten stark von den normalerweise beobachteten abweichen. Dadurch können möglicherweise Katastrophen schnell erkannt werden. Wenn man sich also eine große Liegewiese vorstellt, dann geht man in der Regel davon aus, dass die meisten Menschen ein Picknick genießen, Fußball spielen oder einfach nur faulenzen. Falls nun allerdings der Großteil der Menschen mit hoher Geschwindigkeit von der Liegewiese wegrennen, könnte man daraus schließen, dass an diesem Ort etwas passiert sein muss, was die Menschen zur Flucht verleitet.

Alle drei Segmente bieten großartige Möglichkeiten für ihre Anwender. Im Gegensatz dazu werden im Folgenden die Herausforderungen und Grenzen erläutert.

5 Herausforderungen und Grenzen

Neben den eben dargestellten Möglichkeiten gibt es Herausforderungen und auch Grenzen, sowie Probleme, die überwunden und gelöst werden müssen. Während der Klassifikation gab es häufig das Problem, dass Treppen auf- bzw. absteigen nicht richtig klassifiziert wurde. Kwapisz et al. [5] hatten dieses Problem. Sie konnten in der Wahrheitsmatrix ablesen, dass Treppen auf- und absteigen hauptsächlich untereinander verwechselt wurde. Als Lösung wurden die beiden einzelnen Aktivitäten zu „Treppensteigen“ zusammengefasst, was allerdings etwas unbefriedigend ist.
Außerdem wurde erkannt, dass die Orientierung des Smartphones in der Hosentasche eine große Auswirkung auf die Accuracy der Klassifikation hat. Es ist also wichtig, diese in der Klassifikation zu berücksichtigen. Sun et al. [13] haben sich diesem Problem angenommen und versuchten für jede Orientierung einen eigenen SVM-Klassifikator zu trainieren. Sie haben dabei die sechs verschiedenen in Abb. 8 gezeigten Orientierungen des Smartphones berücksichtigt. Damit erhielten sie eine Accuracy von 94,8%.

Abb. 8. Verschiedene Orientierungen des Smartphones, wie sie bei unterschiedlichen Benutzern auftreten können. Durch Berücksichtigung der Orientierungen verbessert sich die Klassifikation (siehe: [13]).

Ustev et al. [14] verfolgten in ihrem Experiment das gleiche Ziel, wählten aber einen anderen Ansatz. Sie haben einerseits den linearen Beschleunigungssensor benutzt, der unabhängig von der Erdbeschleunigung misst. Das heißt es wird nur die Beschleunigung auf den Koordinatenachsen gemessen ohne Berücksichtigung der Gravitation. Dadurch wurde die Qualität der Klassifikation schon verbessert. Andererseits haben sie die Werte des linearen Beschleunigungssensors noch mit den Werten des Gyroskops und des Magnetometers verbunden. Dadurch haben sie zusätzliche Lageinformationen des Smartphones erhalten und konnten damit Aussagen über dessen Orientierung treffen. Zuletzt haben sie, wie in Abb. 9 dargestellt, die Werte von Smartphone- in Weltkoordinaten umgerechnet, und schließlich eine Accuracy von 97% erreicht.
Heutzutage werden Smartphones immer leistungsfähiger und auch die Akkulaufzeit verbessert sich zunehmend. Trotzdem stellt die aktuelle Hardware im-

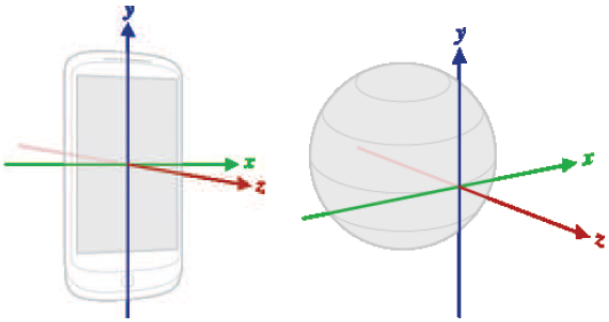


Abb. 9. Sensordaten werden von Smartphonekoordinaten in Weltkoordinaten umgerechnet, um Unabhängigkeit der Orientierung des Smartphones zu erreichen (siehe: [14]).

mer noch eine Herausforderung dar. Insbesondere der Einsatz vieler Sensoren wirkt sich auf die Akkulaufzeit und auch auf den benötigten Speicherplatz aus. Deshalb sollte die in Kapitel 3.2 angesprochene Abtastrate in dieser Hinsicht abgestimmt sein.
Lockhart et al. [8] beschreiben auch, dass auf Android-Geräten die Sensoren nur dann aktiv sein können, wenn der Prozessor aktiv ist. Damit läuft das Gerät unter voller Last und verbraucht deshalb viel Strom. Apple, Windows und Blackberry hingegen beschränken den Zugriff auf Sensoren sogar und machen es Entwicklern schwer, Anwendungen im Hintergrund laufen zu lassen [12].
Neben den eher technischen Aspekten gibt es auch Herausforderungen und Grenzen, die hauptsächlich mit den Benutzern zu tun haben. Das Hauptproblem liegt darin, dass jeder Mensch unterschiedliche Gewohnheiten und Verhaltensweisen hat. Dadurch ist es sehr schwierig, ein System zu entwickeln, welches jeden Menschen mit all seinen Eigenheiten erkennen kann. Auf der einen Seite birgt es also eine Schwierigkeit, dass unterschiedliche Benutzer die gleiche Aktivität möglicherweise unterschiedlich durchführen. Beispielsweise führt Andy Roddick eine ganz andere Bewegung beim Aufschlag aus als Rodger Federer. Beide haben allerdings soviel Erfolg damit, dass sie im Profitennis mitspielen können. Das macht es dem System schwer, diese Aktivität zuzuordnen, da möglicherweise auch in den Trainingsdaten schon eine große Variation existiert. Beispielsweise könnte es einem Aktivitätserkennungssystem sehr schwer fallen zu erkennen, ob eine Person Fußball, Basketball oder Handball spielt. In allen Sportarten muss die Person sprinten, sich freilaufen und schnelle Richtungswechsel vollziehen. Allein durch Bewegungsdaten der Smartphonesensoren stellt das eine große Herausforderung dar. Helaoui et al. [4] haben versucht, über eine Entscheidungslogik eine Ontologie aufzubauen, wodurch sie zuerst von „Atomic Gestures“ über „Manipulative Gestures“ und „Simple Activities“ auf „Complex Activities“ geschlossen haben. Dadurch müssen nur einfache Gesten erkannt werden, um am Ende komplexe Aktivität klassifizieren zu können.

Als letzte Herausforderung oder möglicherweise auch Grenze sollte noch erwähnt werden, dass menschliche Aktivitätserkennung auch für die Privatsphäre ein schwieriges Thema ist. Es werden Daten über den Alltag von Benutzern aufgezeichnet und an fremde Server übermittelt, wobei der Benutzer häufig nicht weiß, was mit den Daten passiert. Deshalb ist es für Benutzer, als auch für Entwickler wichtig darauf zu achten, dass dokumentiert wird, was mit den Daten passiert und für was sie verwendet werden. Außerdem muss heutzutage auch darauf geachtet werden, dass Daten verschlüsselt abgespeichert sind und vor potenziellen Angreifern geschützt sind.

6 Fazit

Zusammengefasst wurden in dieser Arbeit zuerst Grundlagen im maschinellen Lernen angesprochen. Dabei wurde der Begriff der Klassifikation aufgegriffen, wobei versucht wird, Datensätze zu bekannten Klassen zuzuordnen. Dabei ist es wichtig, die Rohdaten vorher in Features umzuwandeln. Ein zu klassifizierender Datenpunkt x wird dann auf ein vorher vom Klassifikationsalgorithmus erstelltes Modell angewandt. Anschließend wurden noch gängige Gütekriterien vorgestellt, wobei der F-Score und die Accuracy die am häufigsten angewandten sind.

Des Weiteren wurden die aktuellen Best Practices in den Bereichen Sensoren, Abtastrate und Fenstergröße, Features und Klassifikator angesprochen. Als Sensor wird hauptsächlich der 3-Achsen Beschleunigungssensor verwendet. Dieser ist in nahezu allen modernen Smartphones verbaut. Um die Lage des Smartphones erkennen zu können, werden häufig noch das Gyroskop und das Magnetometer hinzugezogen. Für die Zukunft gibt es allerdings viele Projekte, die noch GPS Daten und andere Sensoren hinzufügen wollen [6]. Die optimale Abtastrate wurde vom Experiment von Maurer et al. [9] auf 15-20 Hz bestimmt - allerdings ist das wie auch bei der Fenstergröße immer von den genauen Aktivitäten, die erkannt werden sollen, abhängig und kann durchaus variieren. Kwapisz et al. möchten ihre Anwendung in Zukunft um weitere Aktivitäten wie beispielsweise Radfahren und Auto fahren ergänzen [5]. Bezüglich der Features werden einerseits zeitabhängige Features wie beispielsweise der Durchschnitt oder die Standardabweichung eingesetzt. Andererseits wird aber auch eine Diskrete Fourier Transformation durchgeführt, um frequenzabhängige Features einsetzen zu können. Auch bezüglich neuer Features wollen Kwapisz et al. in Zukunft weitere Forschungen tätigen [5]. Es gibt eine große Bandbreite an Klassifikatoren, die in aktuellen Forschungsexperimenten benutzt werden. Häufig werden allerdings Support Vector Machines eingesetzt. Es muss allerdings teilweise der Trade-off zwischen Accuracy und Laufzeit getroffen werden. Zukünftig möchten viele Autoren ihre Experimente um weitere Klassifikatioren erweitern [13,6].

Anschließend wurden Möglichkeiten und Anwendungsgebiete betrachtet, die in drei Sektoren eingeteilt wurden. Im Bereich der Endbenutzer gibt es viele An-

sätze im Bereich des Fitness- und Gesundheitswesens. Allerdings gibt es auch Ideen für alltägliche Anwendungen im Bereich der kontextsensitiven Systeme. Für Unternehmen und Dritte steht hauptsächlich gezielte Werbung im Fokus. Allerdings sind forschungsorientierte Unternehmen auch auf Rohdaten angewiesen, sodass dafür ein Markt besteht. Außerdem gibt es Ansätze für Gruppen und Crowds, die hauptsächlich im Bereich „Social Networking“ und „Crowdsourcing“ angesiedelt sind.

Abschließend wurden Herausforderungen und Grenzen im Bereich der menschlichen Aktivitätserkennung vorgestellt. Dabei sind benutzerabhängige Faktoren zu beachten, z.B. dass Bewegungen von unterschiedlichen Personen unterschiedlich ausgeführt werden, sowie dass grundlegende Bewegungen in unterschiedlichen Aktivitäten auftauchen können. Zusätzlich ist die Orientierung des Smartphones zu beachten. Einige Experimente konnten ihre Ergebnisse damit signifikant verbessern [13,14], sodass auch andere Forschungsgruppen diese Herausforderung in Zukunft betrachten wollen [5]. Ustev et al. wollen zukünftig sogar noch einen Schritt weiter gehen und die Position des Smartphones erkennen. Das heißt, sie versuchen zu klassifizieren, wo der Benutzer das Smartphone trägt. Beispielsweise in der Hosentasche, im Rucksack oder in der Handtasche. Zusätzlich zu diesen benutzerspezifischen Herausforderungen sind aber auch hardwarespezifische Problemen vorhanden, da die Akkulaufzeit in modernen Smartphones noch immer sehr begrenzt ist. Insbesondere den Energieverbrauch bei mehreren Sensoren möchten Ustev et al. weiter beschränken [14]. Neben diesen Herausforderungen sollte noch beachtet werden, dass die Privatsphäre ein wichtiges Thema in der menschlichen Aktivitätserkennung ist. Deswegen möchten Kwapisz et al. zukünftig eine Anwendung entwickeln, die die Klassifikation lokal auf dem Smartphone stattfinden lässt [5].

Literatur

1. ios developer library, `https://developer.apple.com/library/ios/documentation/uikit/reference/UIAcceleration_Class/Reference/UIAcceleration.html`, zuletzt aufgerufen am 23.06.2014, 12:00.
2. Nexus 5 - google - technische daten, `http://www.google.de/nexus/5/`, zuletzt aufgerufen am 17.06.2014, 12:00.
3. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In: Ambient Assisted Living and Home Care, pp. 216–223. Springer (2012)
4. Helaoui, R., Riboni, D., Stuckenschmidt, H.: A probabilistic ontological framework for the recognition of multilevel human activities. In: Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing. pp. 345–354. ACM (2013)
5. Kwapisz, J.R., Weiss, G.M., Moore, S.A.: Activity recognition using cell phone accelerometers. SIGKDD Explor. Newsl. 12(2), 74–82 (Mar 2011), `http://doi.acm.org/10.1145/1964897.1964918`

6. Lee, Y.S., Cho, S.B.: Activity recognition using hierarchical hidden markov models on a smartphone with 3d accelerometer. In: Hybrid Artificial Intelligent Systems, pp. 460–467. Springer (2011)
7. Lockhart, J.W., Pulickal, T., Weiss, G.M.: Applications of mobile activity recognition. In: Proceedings of the 2012 ACM Conference on Ubiquitous Computing. pp. 1054–1058. UbiComp '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2370216.2370441
8. Lockhart, J.W., Weiss, G.M., Xue, J.C., Gallagher, S.T., Grosner, A.B., Pulickal, T.T.: Design considerations for the wisdm smart phone-based sensor mining architecture. In: Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data. pp. 25–33. ACM (2011)
9. Maurer, U., Smailagic, A., Siewiorek, D.P., Deisher, M.: Activity recognition and monitoring using multiple sensors on different body positions. In: Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on. pp. 4–pp. IEEE (2006)
10. Ploetz, T., Hammerla, N.Y., Olivier, P.: Feature learning for activity recognition in ubiquitous computing. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence. vol. 22, p. 1729 (2011)
11. Shin, C., Dey, A.K.: Automatically detecting problematic use of smartphones. In: Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing. pp. 335–344. ACM (2013)
12. Sieling, J., Moon, J.: Performance of smartphone on-board accelerometers for recording activity. In: OBESITY. vol. 19, pp. S123–S123. NATURE PUBLISHING GROUP 75 VARICK ST, 9TH FLR, NEW YORK, NY 10013-1917 USA (2011)
13. Sun, L., Zhang, D., Li, B., Guo, B., Li, S.: Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations. In: Yu, Z., Liscano, R., Chen, G., Zhang, D., Zhou, X. (eds.) Ubiquitous Intelligence and Computing, Lecture Notes in Computer Science, vol. 6406, pp. 548–562. Springer Berlin Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-16355-5_42
14. Ustev, Y.E., Durmaz Incel, O., Ersoy, C.: User, device and orientation independent human activity recognition on mobile phones: challenges and a proposal. In: Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication. pp. 1427–1436. ACM (2013)

Predictive Analysis on Time Series

Lucas Bechberger*
Advisor: Anja Bachmann†

Karlsruhe Institute of Technology (KIT)
Pervasive Computing Systems – TECO
*Lucas.Bechberger@student.kit.edu
†Bachmann@teco.edu

Abstract. This paper gives an overview over different methods for predicting time series. General concepts regarding time series prediction are introduced. Then, both linear methods (with focus on ARIMA) and nonlinear methods (with focus on Multi Layer Perceptrons) are presented. Approaches to combine different methods are discussed, including a hybridization of ARIMA and MLP and the more general approach of ensembles. Moreover, in addition to the widespread single-point prediction, the concept of prediction intervals is presented and practical approaches of estimating such prediction intervals (namely ensemble-based approaches and conformal predictors) are discussed.

Keywords: Predictive Analysis, Time Series, Time Series Prediction, Artificial Neural Network, ARIMA, Prediction Interval, Multi Layer Perceptron, Ensemble, Conformal Predictor

1 Introduction

Predicting the future is a task being carried out by humans every day, significantly influencing their behavior. For instance, the weather forecast tries to predict the weather conditions for the next hours and days. One might base the decision whether or not to take an umbrella or the decision whether or not to plan a picnic for the weekend on this forecast.
Hawkins even argues that predicting the future is not just an important task, but that

> "[p]rediction is not just one of the things your brain does. It is the *primary function* of the neocortex, and the foundation of intelligence." [11]

Besides this rather philosophical aspect, making accurate predictions is crucial in many applications – knowing something about the future usually is a major advantage. Wong [31] argues that every important business decision is based on some sort of forecast, and Silipo & Winters [24] note that a 1% increase in prediction accuracy can lead to operational cost savings of 10 million pounds. Therefore, accurate forecasts are of real business value.

Some forecasting examples include electricity prices, water flows, sunspot activity, tourist arriving patterns, exchange rates, stock prices or the number of airline passengers.
In all of these examples, the basic problem is to use past observations of a variable to predict its future values. For the exchange rate example, this can be formulated as "given the USD-EUR exchange rates of the last two weeks – what will be tomorrow's USD-EUR exchange rate?". If the forecast made in this scenario is sufficiently accurate, it can lead to considerable savings. For instance, an international company can optimize the scheduling of their international wire transfers based on this forecast, in order to minimize the conversion loss.
Also for the other examples, one can easily see that they use past values to predict future ones. Again, accurate predictions can be used for an optimization of some kind, may it be scheduling like in the exchange rate example or an airline's decision whether or not to buy additional planes. Usually, these optimizations will help to lower costs or to increase profits.
The examples illustrate the need for accurate predictions and their importance to business decisions. This paper will explain in more detail how the general problem of time series prediction can be approached.

The remainder of the paper is organized as follows: Section 2 contains the problem statement and gives an overview over general concepts regarding time series prediction. Section 3 presents linear methods in general and the ARIMA approach in more detail. Section 4 covers nonlinear methods and focuses on Multi Layer Perceptrons. Section 5 discusses different approaches to combine multiple methods. Section 6 motivates the use of predictive intervals and shows different methods of approximating them. Section 7 summarizes the key insights of this paper and gives an outlook on open research questions.

2 Background

There exists a wide variety of models used for predicting time series. This section's purpose is both to state the general problem of time series prediction and to give an overview of general concepts regarding time series prediction.

The mathematical elements dealt with in time series prediction are time series. A *time series* is defined as a sequence of chronologically ordered values of the same variable measured at different points in time. An example would be the temperature at a certain location, being measured every day at noon for the last 7 days, with values of 23.3 °C, 26.4 °C, 26.2 °C, 26.3 °C, 22.1 °C, 24.2 °C and 26.6 °C.
The problem of time series prediction can informally be stated as extrapolating the given time series one or multiple time steps into the future. For the given temperature example this could be stated as "given this time series of temperature data, what are the most likely temperature values to follow?"

Let Y be the time series variable to predict. Let Y_t denote the value of Y at time t. A general assumption is that the time interval between two measurements of Y is kept constant. Then, the problem of time series prediction can be defined as follows:

$$\text{Given } Y_{t-l}, ..., Y_t, \text{ predict the value of } Y_{t+k}.$$

Here, l is called the *lag* (or the *window size*) and denotes how many time steps into the past are considered, whereas k is called the *lead time* and denotes the number of time steps ahead of t to be forecast. $k = 1$ gives a *one-step ahead prediction* (or *short-term prediction*). Any greater value of k gives a so-called *multi-step ahead prediction* (or *long-term prediction*).
Although multi-step ahead prediction is a very interesting research topic, it tends to be more complicated than one-step ahead prediction. Therefore, this paper will put its focus on one-step ahead prediction.

Let $\hat{Y}_{t+k}$ be the prediction for Y_{t+k}. Then,

$$Y_{t+k} = \hat{Y}_{t+k} + \epsilon_{t+k}$$

with ϵ_{t+k} being the prediction error. Clearly, for a good prediction, the goal is to minimize ϵ_{t+k}. An alternative way of formulating the problem is the following: Find a function f such that

$$\hat{Y}_{t+k} = f(Y_{t-l}, ..., Y_t) \quad \textit{with } |Y_{t+k} - \hat{Y}_{t+k}| \textit{ being minimized.}$$

How f will look like is mainly determined by the forecasting method being used.

A *forecasting method* is a general approach to solving the time series prediction problem. Most forecasting methods include the step of fitting a *forecasting model* against the data (i.e. optimizing some parameters). In [4], Chatfield points out the distinction between "forecasting method" and "forecasting model" and argues that the terms should not be used interchangeably: There are forecasting methods that involve fitting a forecasting model (e.g. ARIMA, ANNs), but there are also forecasting methods that do not make use of a forecasting model (e.g. the naive method, where $\hat{Y}_{t+1} = Y_t$, does not contain any parameters to be optimized).

Of course, if there already is a good model of the application domain and the underlying process generating the observed time series, this model can and should be used to predict future values of the time series. Amjady [2] calls this a *white box* approach.
If, however, no such model is available (which often means that the underlying process is unknown or poorly understood), the more general forecasting methods presented in this paper can be used [33]. This is called a *black box* approach.

So far, only *univariate* time series prediction has been discussed, where all parameters of f are past values of Y. In *multivariate* time series prediction, however, f can also take other parameters as input [5]:

$$\hat{Y}_{t+k} = f(Y_{t-l}, ..., Y_t, X^{(1)}, ..., X^{(n)})$$

The temperature example from the beginning of this section can be extended to a multivariate prediction problem, if in addition to previous temperature values also the current wind strength is used for the prediction.
Multivariate time series prediction tends to be much more complex than univariate time series prediction. Moreover, it is being less applied in practice [8]. Therefore, we will focus on univariate time series prediction for the scope of this paper.

Yadav & Toshniwal [32] and Wong [31] list four major components that can be used to characterize a time series:

- **Trend Component** T: The general movement of the time series mean over a long period of time. Often, a linear trend is assumed.
- **Cyclic Component** C: The long term oscillation of the time series. May be periodic or not.
- **Seasonal Component** S: A systematic, mostly calendar-related oscillation.
- **Random Component** R: Random fluctuation in the data. Mostly assumed to be normally distributed with zero mean.

The value of the time series Y at time t can then be expressed by multiplying these components:

$$Y_t = T_t \cdot C_t \cdot S_t \cdot R_t$$

According to Wong [31], the random component R can further be divided into two parts: a pseudo-random component created by a deterministic process, and a component entirely created by stochastic noise. Furthermore, Wong argues that it is possible to model and hence predict the pseudo-random component.

If there is no trend component, a time series can be called *stationary*. More precisely, a time series is called *stationary* if its mean and variance are not changing over time. For some forecasting methods (e.g. ARIMA) it is important that the time series under analysis is made stationary before applying the method itself.

An important concept when talking about time series is *autocorrelation*. It describes the correlation between different measurements of the observed variable, i.e. the correlation of Y_t and Y_{t-m} for some $0 \leq m \leq l$. This autocorrelation is usually high for small lags (i.e. small values of m) and for lags m corresponding to the length of a cycle or season [5].
Note that the whole field of time series prediction is based on the assumption that there is some autocorrelation in the data, i.e. that future values depend on past values to at least some extent.

To evaluate the performance of a forecasting method, different accuracy measures can be used. De Gooijer & Hyndman [8] provide a thorough list of such accuracy measures, including the most commonly used mean squared error. As it is common in the field of machine learning, the accuracy measure is computed on a test set separate from the training set which is exclusively used for training the model. Usually, the test set for time series prediction tasks will be the most recent historical data available, whereas the training set will consist of all older data.

Forecating methods can be subdivided into two classes: *linear* methods and *nonlinear* methods. Linear methods assume that the forecast $\hat{Y}_{t+k}$ can be calculated as a linear combination of the lag values and other values, if applicable. These methods will be further explored in Section 3. Nonlinear methods do not assume such a linear decomposition of the time series being analyzed. They will be covered in more detail in Section 4. A detailed overview over different linear and nonlinear methods is given by De Gooijer & Hyndman [8].

3 Linear Methods

3.1 General Idea

Linear methods assume that any value of the time series can be expressed using only linear relationships. Therefore, especially all linear regression methods are linear methods. Although the theoretical assumption of linearity in the data is not always satisfied, linear methods still might yield good performance in practice.
Ruta et al. [23] name two simple linear forecasting methods: the *naive method* where

$$\hat{Y}_{t+1} = Y_t$$

and the *simple moving average* where

$$\hat{Y}_{t+1} = \frac{1}{l+1} \sum_{i=0}^{l} Y_{t-i}$$

Exponential smoothing is another simple method, given by the following equation:

$$\hat{Y}_{t+1} = \alpha \cdot Y_t + (1-\alpha) \cdot \hat{Y}_t$$

where α is the *smoothing factor* which determines the weighting of the actual value in relation to the prediction.
Note that all three of these simple methods are special cases of the more general ARIMA method [8] which will be presented in the remainder of this section.

3.2 ARIMA

"ARIMA" is an acronym for **A**uto **R**egressive **I**ntegrated **M**oving **A**verage. The ARIMA method has been the most popular linear method of the last decades and is considered to be a well established time series prediction method. Basically, an ARIMA model consists of an autoregressive and a moving average component. ARIMA models were introduced by Box & Jenkins [3]. In order to fully motivate the ARIMA formulas, we will follow their step-by-step explanation.

Let B be the *backward shift operator*:

$$\begin{aligned} B \cdot Y_t &= Y_{t-1} \\ B^2 \cdot Y_t &= Y_{t-2} \\ &\vdots \end{aligned}$$

Then the *backward difference operator* ∇ can be defined as:

$$\nabla Y_t = Y_t - Y_{t-1} = (1 - B) \cdot Y_t$$

Furthermore, let a_t be a random variable at time t which is created by a white noise process.

Assuming that the time series Y is stationary with mean μ, it can be expressed with a *linear filter* model. Values of the time series are expected to be distributed around the mean, based on the white noise process:

$$Y_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \dots$$

The assumption of stationarity of course requires $\sum_{j=0}^{\infty} |\psi_j| < \infty$. A more compact way of expressing the above formula is

$$\begin{aligned} Y_t &= \mu + \Psi(B) \cdot a_t \\ with \quad \Psi(B) &= 1 + \psi_1 B + \psi_2 B^2 + \dots \end{aligned}$$

An *Autoregressive (AR)* model tries to model the deviations from the mean, i.e.

$$\tilde{Y}_t = \mu - Y_t$$

It expresses the current deviation as linear combination of previous deviations plus the current white noise term and therefore can be written as

$$\tilde{Y}_t = \phi_1 \tilde{Y}_{t-1} + \dots + \phi_p \tilde{Y}_{t-p} + a_t$$

with p being called the *order* of the AR model. This equation can be rewritten to:

$$\begin{aligned} \Phi(B) \cdot \tilde{Y}_t &= a_t \\ with \quad \Phi(B) &= 1 - \phi_1 B - \dots - \phi_p B^p \end{aligned}$$

Also a *Moving Average (MA)* model considers the deviations from the mean. It expresses the current deviation as linear combination of the current and previous white noise terms and can be written as

$$\tilde{Y}_t = a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}$$

with q being the order of the MA model. This equation can again be rewritten into a more compact form:

$$\begin{aligned} \tilde{Y}_t &= \Theta(B) \cdot a_t \\ with \quad \Theta(B) &= 1 - \theta_1 B - \ldots - \theta_q B^q \end{aligned}$$

An *Autoregressive Moving Average (ARMA)* model is simply the combination of an AR and an MA model:

$$\tilde{Y}_t = \overbrace{\phi_1 \tilde{Y}_{t-1} + \ldots + \phi_p \tilde{Y}_{t-p}}^{Autoregressive\ Model} + \underbrace{a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}}_{Moving\ Average\ Model}$$

In a more compact form, it is also possible to write

$$\Phi(B) \cdot \tilde{Y}_t = \Theta(B) \cdot a_t \tag{1}$$

with $\Phi(B)$ and $\Theta(B)$ as defined above. For clearer distinction, p is now called *autoregressive order* whereas q is called *moving average order*. An ARMA model is usually specified as ARMA(p,q). ARMA models obviously include all AR and all MA models (for either p or q being set to zero). Note that it is possible to compute $\tilde{Y}_t$ (and therefore Y_t) by rearranging equation (1):

$$\tilde{Y}_t = \Phi(B)^{-1} \cdot \Theta(B) \cdot a_t$$

An *Autoregressive Integrated Moving Average (ARIMA)* model does not consider deviations from the mean, but the actual time series values Y_t. Moreover, the assumption of a stationary time series varying around a fixed mean is dropped. The general idea is, that even for nonsationary time series, there might still be some difference of the time series which is stationary. Box & Jenkins [3] provide a more thorough argumentation of why this assumption is feasible.
An ARIMA model is defined by the following equation:

$$\Phi(B) \cdot \nabla^d \cdot Y_t = \Theta(B) \cdot a_t \tag{2}$$

with $\Phi(B)$ and $\Theta(B)$ as defined above and d being the order of the first stationary difference of the time series.
By defining

$$w_t := \nabla^d Y_t$$

it is possible to write equation (2) like an ordinary ARMA equation:

$$\Phi(B) \cdot w_t = \Theta(B) \cdot a_t$$

Depending on the application, it is also possible to set

$$w_t := \nabla^d (Y_t - \mu)$$

The term *Integrated* in the ARIMA acronym stems from the following thought: Once the w_t has been determined, Y_t can be calculated as

$$Y_t = \nabla^{-d} w_t$$

which roughly corresponds to an integration operation on w_t. An ARIMA model is usually specified as ARIMA(p,d,q). Obviously, all ARMA models are also ARIMA models (for $d = 0$).

In the original approach described above, differencing (i.e. repeatedly using the backward difference operator ∇) is used to make a time series stationary. Makridakis & Hibon [18] provide an alternative approach: they simply try to remove the linear trend component of the time series. Their results indicate that with a short lead time the differencing approach performs better, whereas with longer lead time the removal of the linear trend outperforms the differencing approach. However, as they note, for most real world applications, the trend cannot be safely assumed to be linear, so more complex methods might be needed to remove also nonlinear (e.g. exponential) trend components.

After having defined the ARIMA model, there is still the open question how one can fit an ARIMA model to data of a time series. The general guidelines provided by Box & Jenkins [3] are now known as the *"Box-Jenkins methodology"*. Khashei & Bijari [14] describe this methodology on a high level, consisting of three main phases:

- **Model Identification**: Make the time series stationary (by identifying d) and use certain autocorrelation properties to determine the model's orders p and q.
- **Parameter Estimation**: Estimate the model parameters ϕ_i and θ_j, given some error measure to minimize.
- **Diagnostic Checking**: Check if the model assumptions about the remaining prediction errors are satisfied.

Depending on the results of the diagnostic checking phase, these steps might be repeated multiple times until a satisfactory model is selected.
The ARIMA method presented so far is univariate, but there are also multivariate generalizations [8].

The ARIMA method was explicitly created to model and analyze time series. Applying it to time series prediction is therefore very straightforward: First, the ARIMA model is fit to the data of the time series, e.g. by using the Box-Jenkins methodology. Then, the current and previous values are used as input to the formulas. Finally, the formulas are rearranged to retrieve the mathematical solution $\hat{Y}_{t+1}$ to the equations derived from the training data.

4 Nonlinear Methods

4.1 General Idea

In contrast to linear methods, nonlinear methods allow for more complex than only linear relationships within the data. Nonlinear methods include most machine learning techniques like support vector machines, hidden Markov models and artificial neural networks (ANNs). Especially ANNs are applied widely due to their inherent flexibility.

Nonlinear methods work especially well with nonlinear data (and most real-life systems are nonlinear in nature or at least contain a significant nonlinear component), but the results of applying them to linear data has yielded mixed results. Therefore, they cannot be considered a "silver bullet" and should not be blindly applied to every forecasting problem [14]. The following subsections will take a closer look at the Multi Layer Perceptron (MLP), which is the most commonly used ANN type, and at approaches based on pattern matching.

4.2 Multi Layer Perceptron

ANNs in general are neurobiologically motivated machine learning techniques which try to mimic the network of neurons found in human brains. This introduction into MLPs is necessarily brief. It is mainly based on Mitchell's work [21], which provides a very detailed description of the MLP approach.

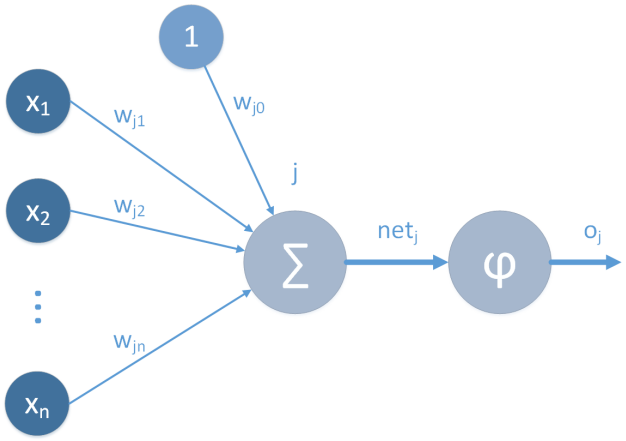


Fig. 4.2.1. Illustration of a simple perceptron with index j.

The basic unit of any ANN is an artificial neuron, which in Multi Layer Perceptrons is called a *perceptron*. Figure 4.2.1 shows such a perceptron with

index j. It takes a vector of inputs $(x_1, \ldots, x_n)$ where each input x_i has an associated weight w_{ji}. The perceptron then takes the weighted sum of the inputs $\sum_{i=1}^{n} w_{ji}x_i$ and adds so-called bias term w_{j0}. In order to write this sum nicely, often a so-called "bias unit" $x_0 = 1$ is introduced, so one can write

$$net_j = \sum_{i=0}^{n} w_{ji}x_i$$

To determine the output of a perceptron, a so-called *transfer function* $\varphi(net_j)$ is used. Popular choices are the linear transfer function

$$id(x) = x$$

and the sigmoid function

$$sig(x) = \frac{1}{1 + e^{-x}}$$

The output of a perceptron can be expressed like this:

$$o_j = \varphi(net_j) = \varphi\left(\sum_{i=0}^{n} w_{ji}x_i\right)$$

Depending on the transfer function φ, the output of a perceptron may be linear (e.g. when using the linear transfer function $id(x) = x$) or nonlinear (e.g. when using the sigmoid function).

In order to train a perceptron (i.e. determining the weights w_{ij}), an error function is needed. It quantifies the output error and will be minimized during training. The most commonly used error function in this context is defined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

where $\vec{w}$ is the weight vector of the perceptron, D is the training set and t_d and o_d are the desired and the real output for a training example d.
Minimizing this error function is done by *gradient descent*, i.e. by using the gradient

$$\nabla E(\vec{w}) = \left[\frac{\delta E}{\delta w_{j0}}, \ldots, \frac{\delta E}{\delta w_{jn}}\right]$$

The weight update Δw_{ji} for each weight is determined by

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}}$$

η is the so-called *learning rate* and determines the step size for the weight update, a negative sign is used to move the weights into the direction that decreases E.

$\frac{\delta E}{\delta w_{ji}}$ can be calculated by simply differentiating $E(\vec{w})$, which for a linear transfer function results in

$$\frac{\delta E}{\delta w_{ji}} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

where x_{id} is the value of input x_i for training example d.
Note that in order to compute the gradient, the transfer function must be differentiable. This requirement is fulfilled for both the linear transfer function and the sigmoid function.
The weights will updated according to the following rule:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

These weight updates will be computed in an iterative manner until a minimum of the error function is reached.

Gradient descent basically minimizes the error function by following its gradient (i.e. its "derivation") into a minimum. It can be shown that for a single perceptron, the error function always has only one minimum, hence gradient descent is guaranteed to always find a global minimum (as there are no local minima). Note that the learning rate η has a significant influence on the speed of convergence. In some cases, when η is chosen too large, gradient descent might not converge and end up oscillating around the minimum of E.

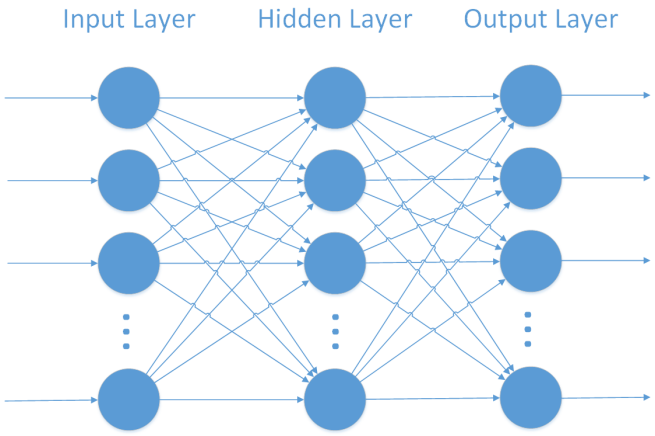


Fig. 4.2.2. An exemplary three-layer MLP with one single layer.

As a single perceptron is not very powerful – it can only model linearly separable classes, which e.g. is not sufficient for solving the XOR problem. Therefore, in practice a network of such perceptrons is used, with the perceptrons being devided into multiple layers – hence the name "Multi Layer Perceptron". See Figure 4.2.2 for an illustration.

There are three types of layers:

- **Input Layer**: First layer in the network, represents the input vector; consists of the input vector values, not of perceptrons.
- **Hidden Layer**: Layer somewhere between the first and the last layer of the network.
- **Output Layer**: Last layer in the network, the output of this layer is the overall output of the network.

Each perceptron in layer k will receive the output of each perceptron in layer $k-1$ as input and its output will be forwarded to all perceptrons in layer $k+1$. That is, the x_i of the perceptrons are actually the o_i of the perceptrons in the preceding layer. There are no other links in the network than the ones described which implies that the network is acyclic. This is the reason why networks of this type are called *feed-forward networks*.

According to Khashei & Bijari [14], MLPs are the most commonly used neural networks in time series prediction. For most applications in this area, only one hidden layer is used and the output layer contains only one perceptron. For such three-layer MLPs with a single output perceptron j one can write its output as

$$o_{MLP} = \varphi_{out}\left(\sum_{i=0}^{Q} w_{ij} \cdot \varphi_{hidden}\left(\sum_{h=0}^{P} w_{hi} x_i\right)\right)$$

with P being the number of inputs and Q being the number of perceptrons in the hidden layer. In most cases, the sigmoid function is used in the hidden layer (therefore $\varphi_{hidden} = sig$), whereas for the output layer, a linear transfer function is used (therefore $\varphi_{out} = id$). It can be easily seen that under these circumstances, due to the use of the sigmoid function, the output of the MLP is nonlinear.

Due to its increased complexity compared to a single perceptron, training an MLP is more complicated and computationally more expensive than training a single perceptron: because of the larger number of weights the hypothesis space (i.e. the space of possible weight combinations) is much larger.
The algorithm most commonly used for training MLPs is called *backpropagation* and can be thought of as a variant of gradient descent. The basic idea of backpropagation is that the error measured at the output layer is propagated backwards through all previous layers in order to update the network's weights (i.e. the weights of all perceptrons in the MLP). In the following only the big picture of backpropagation will be given.
Let δ_j be the error term associated with perceptron j. This error term will be used to compute the weight update:

$$\Delta w_{ji} = \eta \cdot \delta_j \cdot x_i$$

For the output layer neurons, the error term is calculated as

$$\delta_j = \varphi'(o_j) \cdot (t_j - o_j)$$

where φ' denotes the first derivation of φ. In this case, the weight updates are basically computed in the same way as for a single perceptron.
However, for hidden layer neurons, the error term is determined as

$$\delta_j = \varphi'(o_j) \cdot \sum_{k \in layer_{m+1}} w_{jk}\delta_k$$

where m denotes the layer of perceptron j. Hence, the sum is aggregated over all perceptrons in the subsequent MLP layer, i.e. all perceptrons that receive o_j as an input. This means, that the update depends on the weighted sum of all error terms of subsequent perceptrons. This can be intuitively understood as the error being propagated backward through the network of perceptrons. Mitchell [21] explains in greater detail the mathematical foundations of this algorithm and why it is guaranteed to converge.

As the backpropagation algorithm seems to be biologically not plausible, Aitkenhead et al. [1] propose an alternative way of training neural networks. Their "Local Interaction Method" refrains from propagating the global error from the output layer back through the complete network. Instead, the weight updates are determined only based on local criteria, i.e. only depending on a perceptron's immediate neighborhood. They show in two small experiments, that their approach can perform better than the traditional backpropagation approach. However, backpropagation is still the most widely applied training algorithm for MLPs and can be thought of as an accepted standard.

MLPs are a very flexible machine learning technique because they do not make any assumptions about the function to be approximated. However, this adaptability does not come without any cost:
First of all, the overall network structure (the number of layers to use and the number of perceptrons in each of these layers) usually must be determined before training the network via backpropagation. Although there exist some approaches to automatically determine these parameters (e.g. the use of genetic algorithms as in the work of Flores et al. [10]), in practice this is mostly done by expert judgement or trial and error.
Moreover, it is often necessary to train the network for a long time until good performance is reached. However, if a network is trained too long or if it contains too many perceptrons, it will start to *overfit* the data. This means, that there will be a very low error rate on the training data, but large errors for previously unseen data, i.e. the network does not generalize. Overfitting is a well known issue in machine learning and there exist techniques to deal with it (Mitchell [21] suggests e.g. the use of validation sets in addition to training and test sets).
Another problem is the existence of local minima: because backpropagation can be viewed as variant of gradient descent, it will find a minimum, but the error surface of an MLP usually contains multiple local minima. Both the learning rate η and the initial values of the weights can influence whether the algorithm will find a local or a global minimum. There are also some advanced techniques to prevent backpropagation to get stuck in local minima in order to find a global

minimum (e.g. adding a momentum term to the weight update). In most real-world applications, however, local minima are not a big concern.

ANNs in general and therefore also MLPs were not explicitly created to model, analyze or predict time series. In fact, the original intentions of use were classification and function approximation.
However, time series prediction can be seen as a special case of the more general function approximation problem. Therefore, the MLP can be trained to approximate the function

$$f(Y_t, \cdots, Y_{t-l}) := Y_{t+1}$$

This can be done by creating a training set from the time series data:

$$(x_1, \ldots, x_n) := (Y_t, \ldots, Y_{t-l}) \quad \textit{and}$$
$$t_{net} := Y_{t+1}$$

Here, t_{net} denotes the desired output of the network.
Moreover, since ANNs do not make any assumptions about the input data or about the function to approximate, one can simply add additional variables as input to make the prediction multivariate.

There are, of course, also other types of ANNs that can be used for time series prediction. So-called *Time Delayed Neural Networks (TDNNs)* are capable of recognizing input patterns independent of their position in time and were originally used for phoneme recognition [30]. They are interesting in the context of time series analysis because of their ability to robustly handle inputs shifted in time. This property is called *time invariance*.
The class of *Recurrent Neural Networks* (the counterpart of feed forward networks) describes neural networks that have feedback connections, e.g. the output of a neuron being fed back as one of its inputs for the next time step. Due to this property, RNNs are capable of maintaining some inner state through time. This can be very valuable when dealing with time-dependent data like time series.
Kim and Shin [15] present the application of both TDNNs and RNNs to time series prediction. Although these types of ANNs might be better suited for dealing with time series, the MLP can still be considered to be the "standard ANN" in time series prediction. Many papers in this area even use the general term ANN to refer to an MLP.

4.3 Pattern-Matching approaches

A group of nonlinear approaches different from ANNs is based on pattern matching on sequences of labels. Figure 4.3.1 illustrates the general architecture of these approaches: In a first step, the input is transformed from a sequence of real values $Y_t, \ldots, Y_{t-l}$ into a sequence of labels $L_t, \ldots, L_{t-l}$. Then, this sequence of labels is matched to the knowledge base (consisting of other label-sequences

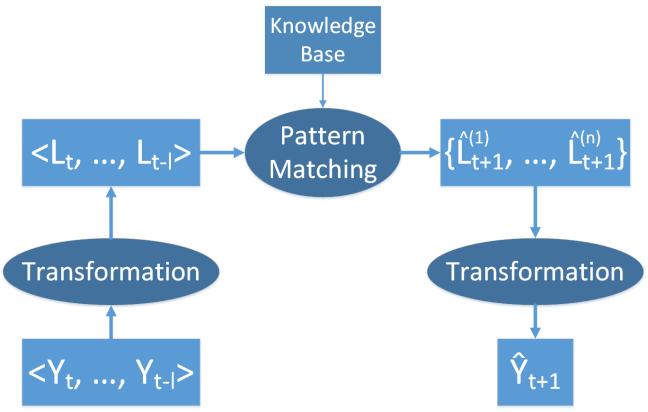


Fig. 4.3.1. Overview over the general architecture of pattern-matching approaches.

or consisting of a set of rules) in order to map it to one or more output labels $\hat{L}_{t+1}^{(1)}, \ldots, \hat{L}_{t+1}^{(n)}$. These output labels are then in a third step converted back into a real-valued prediction $\hat{Y}_{t+1}$.

Martinez et al. [19] use an approach called "Pattern Sequence-based Forecasting" on an energy consumption data set. This data set contains the energy consumption per hour. In a first step, they divide their data set into days and cluster these days based on their energy consumption curve using the k-means algorithm. For each day, the according cluster ID is stored as label.
When predicting the energy consumption curve for the next 24 hours, the sequence of the last l labels is taken into account. The history of label sequences is searched for the current sequence and for each match, the day immediately following this match is recorded. The prediction is then generated by averaging over the energy consumption rate of these days. If the sequence of the last l labels has never occurred in the history, the sequence of the last $l-1$ days is used and so on, until a match is found. Note that only exact matches are taken into account.

Also approaches using fuzzy sets fit into the category of pattern-matching approaches. Song & Chissom [25, 26] propose so-called *fuzzy time series* in this context. The training data input (consisting of real numbers) is first mapped to fuzzy sets, then fuzzy relationships between sequences of fuzzy sets are learned. These relationships have the form of "IF ... THEN ..." rules which are much easier to interpret by a human than e.g. the weights of an MLP.
In the forecasting step, the input sequence is mapped to a sequence of fuzzy set memberships. The learned fuzzy rules are applied and the result (also expressed with fuzzy set memberships) is "defuzzified" into a real-valued prediction. This "defuzzification" can be done e.g. by computing a weighted sum of the set means

with the set memberships being used as weights. Due to the inherent fuzziness of this whole process, there is no exact matching required. Hence, this approach seems to be more flexible than the one of Martinez et al.
Li et al. [16] apply the idea of fuzzy time series to the prediction of electricity prices. However, in general this approach is not yet used widely in practice.

5 Combination of Methods

5.1 General Idea

Virtually every individual forecasting method has its advantages, but also its limitations. Although the combination of different methods does not necessarily lead to better performance, it reduces the risk of using a completely inappropriate method and therefore the risk of total failure [12, 14].

Khashei & Bijari [14] distinguish hybrid architectures in two dimensions: They are either *homogeneous* (e.g. a set of differently configured MLPs) or *heterogeneous* (e.g. a combination of an ARIMA and a MLP). Moreover, one can distinguish *competitive* and *cooperative* architectures: in a competitive architecture, the "best" method will alone determine the output. In a cooperative architecture, the overall prediction is computed by combining the individual methods' forecasts. Mostly, the different methods model different aspects of the time series and are used sequentially.

In the following, two approaches towards hybrid architectures are presented in more detail.

5.2 Cooperative Combination of ARIMA and MLP

As described in Sections 3 and 4, ARIMA is a linear method, whereas MLP is a nonlinear method. Although they are quite different in nature, Zhang [33] also notes some similarities, e.g. both methods include a rich variety of models, need a large training set and are prone to overfitting.

Both Zhang [33] and Khashei & Bijari [14] use a cooperative approach of combining the ARIMA and MLP methods by dividing the time series Y_t into a linear part L_t (modeled by the ARIMA) and a nonlinear part N_t (modeled by the MLP). Both only consider univariate one-step ahead forecasts.

Zhang [33] assumes that the linear and the nonlinear components are additive, therefore

$$Y_t = L_t + N_t$$

His hybrid architecture (see Figure 5.2.1) is constructed as follows:
First, the ARIMA is used to fit the linear component L_t yielding predictions

$$\hat{L}_{t+1} = Y_{t+1} + e_{t+1}$$

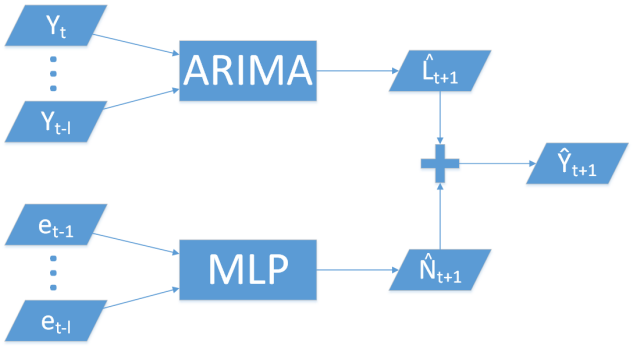


Fig. 5.2.1. Visualization of Zhang's [33] hybrid model for time series prediction which uses an additive combination of ARIMA and MLP predictions.

of the linear component. The residuals e_{t+1} only contain nonlinear relationships, which are modeled using an MLP with a single hidden layer, such that

$$e_{t+1} = f(e_{t-1}, ..., e_{t-l}) + \epsilon_{t+1}$$

with f being the function realized by the MLP and ϵ_{t+1} being the remaining error term. The forecast of the MLP is called $\hat{N}_{t+1}$. The overall forecast of the model is determined by adding the two components:

$$\hat{Y}_{t+1} = \hat{L}_{t+1} + \hat{N}_{t+1}$$

As Zhang showed by carrying out an experiment with three different data sets, this hybrid approach was able to outperform both individual ARIMA and ANN models.

Khashei & Bijari [14], however, argue that one can not safely assume the additive decomposition of a time series into a linear and a nonlinear part. Their approach is depicted in Figure 5.2.2. They describe the time series as general function of a linear and a nonlinear component:

$$Y_t = f(L_t, N_t)$$

Again, an ARIMA is used to fit the linear component. The nonlinear component is split into two subcomponents $N_t^{(1)}$ (based on the residuals) and $N_t^{(2)}$ (based on the w_j used in the ARIMA model) which are both approximated by an MLP:

$$\hat{N}_{t+1}^{(1)} = f^{(1)}(e_t, \ldots, e_{t-n}) \quad and$$
$$\hat{N}_{t+1}^{(2)} = f^{(2)}(w_t, \ldots, w_{t-m})$$

with $f^{(1)}$, $f^{(2)}$ being the nonlinear functions represented by the respective MLP. The overall prediction can then be represented as

$$\hat{Y}_{t+1} = f(\hat{N}_{t+1}^{(1)}, \hat{L}_{t+1}, \hat{N}_{t+1}^{(2)})$$

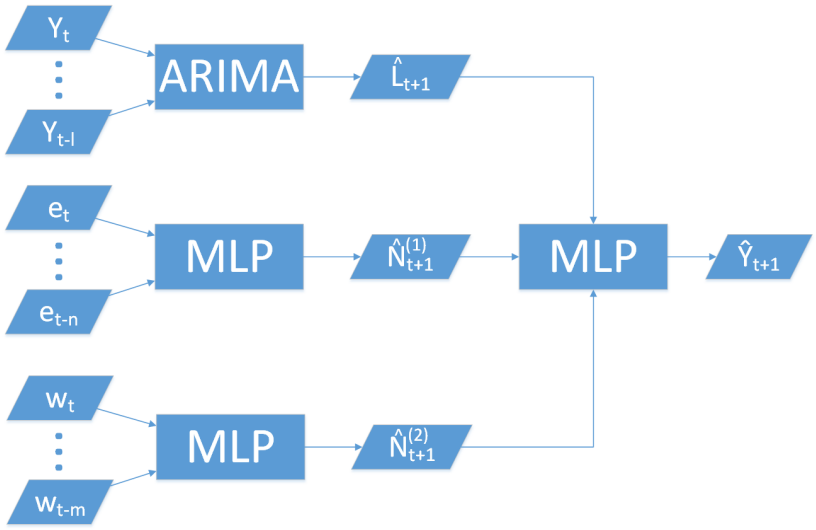


Fig. 5.2.2. Visualization of the hybrid model by Khashei & Bijari [14] which uses multiple MLPs in combination with an ARIMA model for time series prediction.

with f being the overall function represented by the resulting MLP.
Although three different MLPs are used, they can easily be combined into a single MLP receiving inputs $e_t, \ldots, e_{t-n}, w_t, \ldots, w_{t-m}, \hat{L}_{t+1}$.
When comparing the performance of their hybrid method to individual ANN and ARIMA models and to the hybrid method of Zhang, they arrived at the following results: The individual ANN tended to be slightly better than the individual ARIMA. Moreover, both hybrid methods outperformed the individual methods. These two results support the ones of Zhang mentioned aboved. Khashei & Bijari also found that their model yielded higher performance than Zhang's model. They conclude that their approach of combining linear and nonlinear forecast with an arbitrary function is more appropriate than Zhang's approach of simply adding them up.

5.3 Ensemble-based approaches

Another approach of combining different methods is based on so-called *ensembles*. An ensemble is a group of individual forecasting methods that use the same functional approach but different models (i.e. different parameter sets). The key idea is that the methods within an ensemble should be diverse in order to get the most performance gain. This model diversification can be reached by training the models differently.
Ruta et al. [23] suggest the use of different subsets of the training data, different features, different noise levels and a different initialization of the model parameters. They also list different approaches to combine the individual forecasts,

e.g. weighted average with weights determined by the prior accuracy of the individual forecasting models. For their practical evaluation, they use a two-stage approach with k groups: within each group, a single best predictor is selected ("winner takes it all") and then in a second step, the forecasts of the groups are combined by computing a simple average.
The work of Ruta et al. does not only cover ensembles, but proposes a complete architecture for time series prediction which cannot be treated in more detail here.

The approach of Martínez-Rego et al. [20] also makes use of ensembles, although in a somewhat different setting. For their "Distributed Commitees of Local Experts", they use clustering algorithms to create clusters in the input space. Each cluster is represented by one representative data point, called a *cluster node*. Then, for each cluster, a MLP is trained on the data belonging to this cluster and on the data belonging to neighboring clusters. After this, ensembles (called "committees" in their paper) are created: For each cluster, the corresponding ensemble contains all predictors that have been trained on the data points belonging to this cluster. That means, the ensemble of a cluster contains the MLP of that cluster itself and the MLPs of the neighboring clusters.
In the prediction step, the input is mapped to a cluster by picking the cluster node being closest to the input in the input space. The ensemble of this cluster will then determine the overall prediction. Martínez-Rego et al. use another MLP for the combination of the predictions made by the predictors within the ensemble. They call this a *trainable fusion-rule* in contrast to *fixed fusion-rules* like the average operation used by Ruta et al.

The approach taken by Silipo and Winters [24] can also be seen as a ensemble-based approach in the wider sense as it is also based on cluster-wise prediction. They predict the total electricity usage of about 6000 private and commercial consumers by first clustering them, then fitting a model for each of the clusters, and finally, calculating the overall prediction by summing up the individual predictions for all clusters.

6 Predictive Intervals

6.1 General Idea

Every prediction about the future necessarily contains come uncertainty. This uncertainty can have different sources [13]:

- **Input Uncertainty**: Errors in measuring and sampling the data.
- **Parametric Uncertainty**: Inability to identify the best parameter set when fitting the model.
- **Model Structure Uncertainty**: Information loss due to the conversion of real world problems into abstract mathematical formulas.

Of course, also the random component R of the time series (mentioned in Section 2) contributes to the overall uncertainty.

For multi-step ahead predictions the uncertainty is in general larger than for one-step ahead predictions: the larger the lead time, the greater the uncertainty [5]. Therefore, the forecasting accuracy will decrease with increasing lead time [31]. There are different reasons for this effect:
Li et al. [17] point out that historical data used for training the forecasting models are often incomplete, noisy and ambiguous and do not necessarily contain the necessary patterns for long-term forecasts.
As Sovilj et al. [29] add, long term prediction requires a long history of observations for training purposes. This results in the widely known "curse of dimensionality", which states that the amount of training data needed grows exponentially with the number of input dimensions being analyzed. In this context, the number of input dimensions corresponds to the length of the observation history.
Moreover, for multivariate time series, the prediction $\hat{Y}_{t+k}$ might depend on some $X^{(i)}_{t+j}$ $(1 \leq j \leq k)$. This introduces the additional uncertainty attached to forecasting $\hat{X}^{(i)}_{t+j}$ [23].
Another reason for this increased uncertainty is based on the approach used for making multi-step ahead predictions. In general, one can distinguish two approaches to multi-step ahead prediction [17, 23, 27]:
On the one hand, there is the *recursive* approach: the one-step ahead prediction is recursively applied with predictions $\hat{Y}_{t+1}, ...\hat{Y}_{t+k-1}$ being fed back into the prediction method:

$$\begin{aligned}\hat{Y}_{t+1} &= f(Y_t, Y_{t-1}, \ldots, Y_{t-l}) \\ \hat{Y}_{t+2} &= f(\hat{Y}_{t+1}, Y_t, \ldots, Y_{t-l+1}) \\ &\vdots\end{aligned}$$

This approach is very straightforward to implement, but has one major disadvantage: feeding back the predictions as inputs leads to an accumulation of prediction errors and therefore decreases the prediction accuracy [27, 28].
On the other hand, there is the *direct* approach, where k different models are trained for a k-step ahead prediction:

$$\begin{aligned}\hat{Y}_{t+1} &= f_1(Y_t, Y_{t-1}, \ldots, Y_{t-l}) \\ \hat{Y}_{t+2} &= f_2(Y_t, Y_{t-1}, \ldots, Y_{t-l}) \\ &\vdots\end{aligned}$$

This approach is computationally more expensive (because multiple models need to be fit), but does not suffer from accumulated errors like the recursive approach. However, there is still some problem with this approach: the autocorrelation of the time series for large lag values will probably be rather low, which means that historical data is not suitable for explaining data points far in the future.

By combining both approaches into the *DirRec* approach, Sorjamaa & Lendasse [28] try to overcome the weaknesses of the single approaches.

Most forecasting methods do not explicitly take into account the uncertainty attached to their forecast. They usually output exactly one value: their forecast for the future value of the time series. However, this gives no information about the confidence with which this forecast is made. To better assess this prediction confidence, so-called *predictive intervals* (or *prediction intervals*) can be used.
A predictive interval is an interval that will contain the true value of Y_{t+k} with a specified probability p (most commonly $p = 0.95$). Therefore, the output of a forecasting method using predictive intervals is no longer a single value $\hat{Y}_{t+k}$, but rather an interval $\Gamma_{t+k} = [LOW_{t+k}, HIGH_{t+k}]$. De Gooijer & Hyndman [8] argue that the term *prediction interval* should be used for predictions, whereas the term *confidence interval* should be reserved for model parameters to avoid confusion.

For evaluating predictive intervals, two measures are usually used, which are both computed on the test data set [13]: the probability of coverage POC which represents the percentage of test data points lying in the predictive interval, and the average width AW of the predictive interval. The general goal is to maximize POC while minimizing AW. As Dashevskiy & Luo [6] point out, a narrow AW indicates an efficient prediction interval, whereas a large POC close to the desired value of p indicates that the prediction interval is valid.

If some sort of Bayesian method is used, calculating a predictive interval is relatively straightforward as probabilities and distributions are already used throughout the method. The more interesting questions is how to determine predictive intervals for other methods which usually would only make a single-point forecast.
Chatfield [4] provides a thorough introduction into the mathematical problem of constructing prediction intervals and gives an overview of different approaches and common pitfalls. Here, we will focus on two simple techniques to construct approximations of predictive intervals: ensembles and conformal predictors. Although they might be mathematically inappropriate, they are are fairly easy to implement and in practice yield satisfactory results.

6.2 Ensembles

The general idea of ensembles has already been introduced in Section 5. Ensembles not only can be used for achieving a more robust prediction accuracy, but they also can be used to estimate predictive intervals. This can be achieved by calculating mean and variance of the predictions made by the ensemble members.
Kasiviswanathan et al. [13] used this approach to approximate predictive intervals for rainfall runoff data: in their experiment, they trained an MLP on the available data, and then created an ensemble by making copies of the trained

MLP. For each of these copies, they perturbed some randomly selected weights to diversify the ensemble.
The overall prediction of the ensemble was computed as simple arithmetic mean and the variance of the predictions made by the individual MLPs was used to compute the width of the predictive interval. For multi-step ahead forecasts, the direct approach was used. Figure 6.2.1 shows an illustration of their results, where the red bands indicate the predictive interval and the black dots denote the observed values. It nicely illustrates that for larger lead times the uncertainty tends to grow larger. This can be seen by the tendency that the predictive intervals grow larger and contain less of the observed data points as the lead time increases.

6.3 Conformal Predictors

Conformal predictors are a general approach of approximating a confidence interval. They can be used on top of virtually any classification or prediction algorithm (called the *underlying algorithm*). Dashevskiy & Luo [6, 7] provide an introduction into conformal predictors in general and their application to time series prediction. This section is largely based on their work.

Conformal predictors are a machine learning technique that makes predictions based on how well a new example will fit into the set of known training examples. A so-called *nonconformity measure* α is used to represent the dissimilarity between examples. Each example z_i consists of an object x_i and a label y_i, therefore

$$z_i = (x_i, y_i)$$

The *p-value* for a new example $z_n = (x_n, y)$ is defined as:

$$p(y) = \frac{\#\{i = 1, \ldots, n : \alpha_i(y) \geq \alpha_n(y)\}}{n} \tag{3}$$

where $\alpha_i(y)$ denotes the nonconformity score of object x_i if the value y is assigned to the new object x_n.
A large value of $p(y)$ indicates that the new example x_n with label y is very typical for the training set. This is because equation (3) computes the fraction of elements in the training set that have a greater dissimilarity to the training set than the new example. If this fraction is large, obviously the new example fits in very well. Therefore, the general goal is to find the label y with the largest p-value. Assuming that the order of the examples is irrelevant, it is even possible to prove that for any given error probability ϵ one can construct a prediction interval that will contain the actual observation with a probability of $1 - \epsilon$:

$$\Gamma = \{y : p(y) \geq \epsilon\} \tag{4}$$

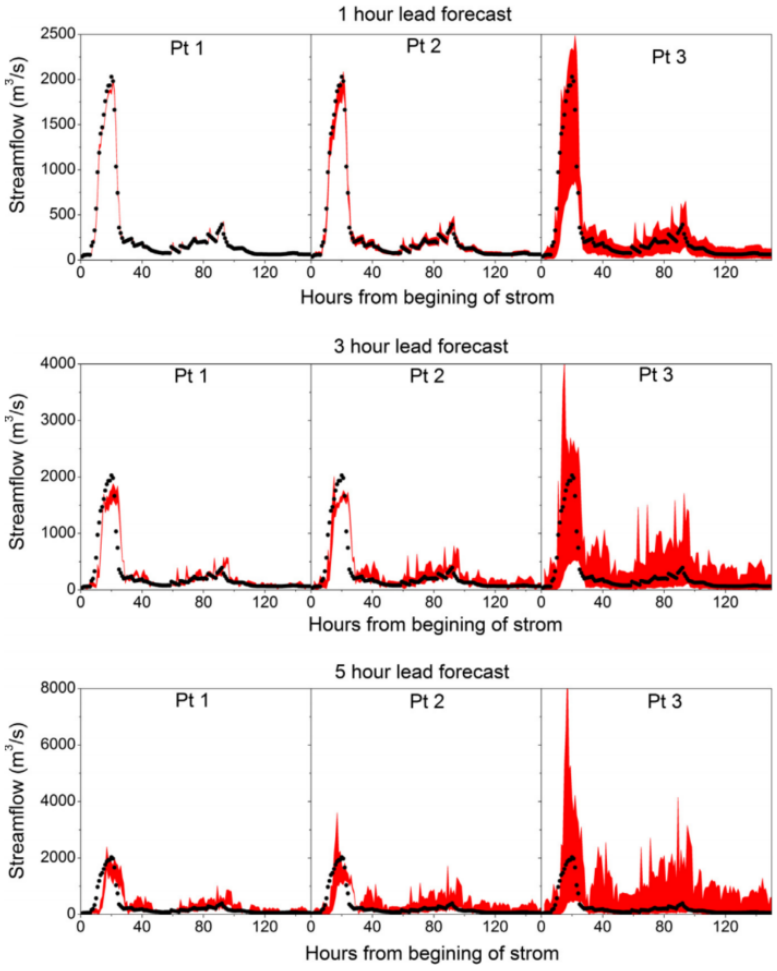


Fig. 6.2.1. Diagram by Kasiviswanathan et al. [13], showing the resulting predictive intervals for different examples and different lead times.

Due to the use of labels, conformal predictors are rather a classifier approach than a regression approach. However, it is possible to define the z_i in a way that conformal predictors can be also used for time series prediction:

$$z_i = (x_i, y_i) := ((Y_{t-l}, \ldots, Y_t), Y_{t+1})$$

Let $\hat{Y}_{t+1}$ be the prediction of the underlying algorithm. Let the error measure for the new object be defined as

$$\alpha_n = |\hat{Y}_{t+1} - Y_{t+1}|$$

Then, for an error probability ϵ, construct the predictive interval as

$$\Gamma_{t+1} = \left[\hat{Y}_{t+1} - r_{max}, \hat{Y}_{t+1} + r_{max}\right] \tag{5}$$

with some r_{max}. The true value Y_{t+1} lies in this interval if and only if:

$$\alpha_n = |\hat{Y}_{t+1} - Y_{t+1}| \leq r_{max} \tag{6}$$

Therefore, r_{max} gives an upper bound on α_n. For given error probability ϵ, equation (4) shall hold true. With equations (5) and (6), r_{max} can then be calculated like this:

$$r_{max} = max_{r \in \mathbb{R}} \left\{ \frac{\#\{i = 1, \ldots, n : \alpha_i(y) \geq r\}}{n} \geq \epsilon \right\}$$

By computing r_{max} in this way, the predictive interval Γ_{t+1} can be calculated based on the prediction $\hat{Y}_{t+1}$ of the underlying algorithm.
The assumption of the order of examples being irrelevant clearly does not hold for time series data, therefore no provable guarantees about the predictive interval can be made anymore. Despite this theoretical lack of valididy, Dashevskiy & Luo report good practical results for applying conformal predictors to time series.

Papadopoulos & Haralambous [22] propose a variant of conformal prediction they call "Inductive Conformal Prediction". It yields improved performance when used with ANNs as underlying algorithm. Figure 6.3.1 shows their results for a sunspot activity data set.

7 Conclusion

This paper gave an introduction into the topic of time series prediction which is a relevant problem in many application areas.

Due to space limitations, this paper can only serve as a brief overview over the topic of time series prediction. With ARIMA and MLP, the two most popular methods were introduced, but there are many other approaches that could not be discussed in this paper, e.g. support vector machines (SVMs) or classification and regression trees (CARTs). De Gooijer & Hyndman [8] provide a very thorough overview over a variety of approaches.

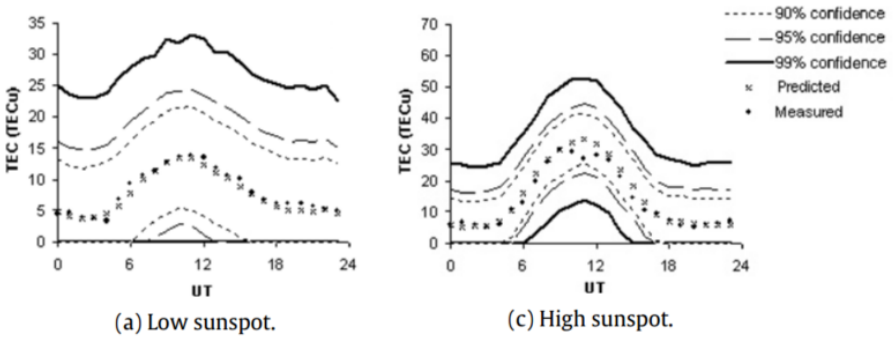


Fig. 6.3.1. Results from the paper of Papadopoulos & Haralambous [22], showing the predictive intervals for two different sunspot activity time series.

In recent years, the use of MLPs has become more and more popular in the field of time series prediction. But although there are other neural networks like TDNNs and RNNs that might be better suited to deal with time series data, they are not yet widely applied in practice.

All predictive methods are based on the assumption that the future is somehow correlated to the past. While this may be true most of the time, there are unforeseeable events like natural disasters, which in general cannot be predicted and therefore cannot be taken into account. It is not quite clear how this limitation can be overcome. Change point analysis [9] tries to detect points in the time series when the underlying process changes (i.e. another model needs to be used) and might be one possible approach to overcome this limitation.

Although most prediction methods work quite well in practice, there is currently no approach that performs equally well for all kinds of input data. As every approach has its individual strengths and weaknesses, the combination of different approaches and the use of predictive intervals instead of single-point forecasts seem to be reasonable steps towards better forecasts. There is still a lot of work to be done until the problem of time series prediction can be considered to be solved.

The original intention of this paper was to analyze the application of time series prediction to activity recognition on mobile devices. However, it turned out that there seems to be no research on the question how time series prediction can be used in this context. Therefore, this seems to be a worthwhile research direction for future work.

References

1. Aitkenhead, M., McDonald, A., Dawson, J., Couper, G., Smart, R., Billett, M., Hope, D., Palmer, S.: A novel method for training neural networks for time-series prediction in environmental systems. Ecological Modelling 162(1), 87–95 (2003)
2. Amjady, N.: Day-ahead price forecasting of electricity markets by a new fuzzy neural network. Power Systems, IEEE Transactions on 21(2), 887–896 (2006)
3. Box, G., Jenkins, G.M., Reinsel, G.: Time series analysis: Forecasting & control (1994)
4. Chatfield, C.: Calculating interval forecasts. Journal of Business & Economic Statistics 11(2), 121–135 (1993)
5. Chatfield, C.: Time-series forecasting. Significance 2(3), 131–133 (2005)
6. Dashevskiy, M., Luo, Z.: Time series prediction with performance guarantee. Communications, IET 5(8), 1044–1051 (2011)
7. Dashevskiy, M., Luo, Z.: Network traffic demand prediction with confidence. In: GLOBECOM. pp. 1453–1457 (2008)
8. De Gooijer, J.G., Hyndman, R.J.: 25 years of time series forecasting. International journal of forecasting 22(3), 443–473 (2006)
9. Denison, D.G., Mallick, B.K., Smith, A.F.: A bayesian cart algorithm. Biometrika 85(2), 363–377 (1998)
10. Flores, J.J., Graff, M., Rodriguez, H.: Evolutive design of arma and ann models for time series forecasting. Renewable Energy 44, 225–230 (2012)
11. Hawkins, J., Blakeslee, S.: On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines. Henry Holt & Company, New York, NY (2004)
12. Hibon, M., Evgeniou, T.: To combine or not to combine: selecting among forecasts and their combinations. International Journal of Forecasting 21(1), 15–24 (2005)
13. Kasiviswanathan, K., Cibin, R., Sudheer, K., Chaubey, I.: Constructing prediction interval for artificial neural network rainfall runoff models based on ensemble simulations. Journal of Hydrology 499, 275–288 (2013)
14. Khashei, M., Bijari, M.: A novel hybridization of artificial neural networks and arima models for time series forecasting. Applied Soft Computing 11(2), 2664–2675 (2011)
15. Kim, H.j., Shin, K.s.: A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets. Applied Soft Computing 7(2), 569–576 (2007)
16. Li, G., Liu, C.C., Mattson, C., Lawarree, J.: Day-ahead electricity price forecasting in a grid environment. Power Systems, IEEE Transactions on 22(1), 266–274 (2007)
17. Li, S.T., Kuo, S.C., Cheng, Y.C., Chen, C.C.: Deterministic vector long-term forecasting for fuzzy time series. Fuzzy Sets and Systems 161(13), 1852–1870 (2010)
18. Makridakis, S., Hibon, M.: Arma models and the box–jenkins methodology. Journal of Forecasting 16(3), 147–163 (1997)
19. Martinez Alvarez, F., Troncoso, A., Riquelme, J.C., Aguilar Ruiz, J.S.: Energy time series forecasting based on pattern sequence similarity. Knowledge and Data Engineering, IEEE Transactions on 23(8), 1230–1243 (2011)
20. Martínez-Rego, D., Fontenla-Romero, O., Alonso-Betanzos, A.: Efficiency of local models ensembles for time series prediction. Expert Systems with Applications 38(6), 6884–6894 (2011)
21. Mitchell, T.M.: Machine learning. 1997. Burr Ridge, IL: McGraw Hill 45 (1997)

22. Papadopoulos, H., Haralambous, H.: Reliable prediction intervals with regression neural networks. Neural Networks 24(8), 842–851 (2011)
23. Ruta, D., Gabrys, B., Lemke, C.: A generic multilevel architecture for time series prediction. Knowledge and Data Engineering, IEEE Transactions on 23(3), 350–359 (2011)
24. Silipo, R., Winters, P.: Big data, smart energy, and predictive analysis (2013)
25. Song, Q., Chissom, B.S.: Forecasting enrollments with fuzzy time series—part i. Fuzzy sets and systems 54(1), 1–9 (1993)
26. Song, Q., Chissom, B.S.: Fuzzy time series and its models. Fuzzy sets and systems 54(3), 269–277 (1993)
27. Sorjamaa, A., Hao, J., Reyhani, N., Ji, Y., Lendasse, A.: Methodology for long-term prediction of time series. Neurocomputing 70(16), 2861–2869 (2007)
28. Sorjamaa, A., Lendasse, A.: Time series prediction using dirrec strategy (2006)
29. Sovilj, D., Sorjamaa, A., Yu, Q., Miche, Y., Séverin, E.: Opelm and opknn in long-term prediction of time series using projected input data. Neurocomputing 73(10), 1976–1986 (2010)
30. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.J.: Phoneme recognition using time-delay neural networks. Acoustics, Speech and Signal Processing, IEEE Transactions on 37(3), 328–339 (1989)
31. Wong, F.S.: Time series forecasting using backpropagation neural networks. Neurocomputing 2(4), 147–159 (1991)
32. Yadav, V., Toshniwal, D.: Graph based framework for time series prediction. Trends in Information Management 7(2) (2011)
33. Zhang, G.P.: Time series forecasting using a hybrid arima and neural network model. Neurocomputing 50, 159–175 (2003)

Predictive Pre-Caching von Daten aus Cloud-Diensten zur Verwendung auf mobilen Geräten

Christoph Michel*

Betreuer: Anja Bachmann†

Karlsruher Institut für Technologie (KIT)
Pervasive Computing Systems – TECO
*uaevw@student.kit.edu
†bachmann@teco.edu

Zusammenfassung. Mit der zunehmenden Verbreitung von Smartphones und Internet-Diensten spielt der mobile Internetzugang eine immer wichtigere Rolle. Ursache hierfür ist, dass die Funktionalität von Applikationen zunehmend auf Daten basiert, die via Internet ausgetauscht werden. Problematisch ist hierbei, dass nicht zu jeder Zeit ein lückenloses und schnelles mobiles Internet gewährleistet werden kann. Daher kann die Funktionalität von Applikationen, die auf das Internet angewiesen sind, stark eingeschränkt werden. In diesem Paper wird zunächst dieses grundlegende Problem der eingeschränkten Funktionalität etwas genauer beleuchtet. Zusätzlich wird die Limitierung der Hardware von Smartphones im Hinblick auf Predictive Pre-Caching betrachtet. Darüber hinaus werden die Ziele, die man auf dem Gebiet des Predictive Pre-Caching verfolgt, in einem Überblick dargestellt. Im Anschluss werden Fragestellungen behandelt, die im Kontext von Predictive Pre-Caching aufkommen. Abschließend werden diverse aktuelle Arbeiten kurz vorgestellt und ein Überblick über diese geliefert.

Schlüsselwörter: Predictive Pre-Caching, Mobile Computing, Cloud-Services

1 Motivation

Die Benutzung von mobilem Internet nimmt immer weiter zu. Allein im Jahr 2013 stieg die Zahl der mobilen Internetnutzer in Deutschland um 43% [2]. Mit diesem Aufschwung wachsen sowohl die Zahl der Anwendungszwecke, als auch die Anforderungen an die Geräte. Da mobile Geräte im Vergleich zu stationären Geräten meistens weniger Rechenleistung und Speicher besitzen [7], stoßen Entwickler immer wieder an die Hardwaregrenzen der mobilen Geräte. Dabei spielt neben begrenzter Akku-Kapazität, Prozessorleistung und Speicher vor allem die Netzanbindung eine immer wichtigere Rolle. Denn das Zusammenspiel zwischen rechenstarken Cloud-Diensten und den dazugehörigen Interfaces auf mobilen Geräten stellt ein wichtiges Prinzip dar [4]. Dieses Prinzip führt dazu, dass die Funktionalität von Programmen, die auf mobilen Endgeräten zum Einsatz kommen,

zunehmend auf Daten beruhen, die via Internet ausgetauscht werden. Da ohne Verbindung keine Daten ausgetauscht werden können bzw. mit einer schlechten Verbindung nur ein langsamer Datenaustausch stattfinden kann, steht und fällt die Funktionalität und das Nutzungserlebnis der Applikation jeweils mit der Verfügbarkeit bzw. Geschwindigkeit des Internets. Bei drahtlosen Verbindungen kann eine hohe Mobilität der Nutzer zu Einschränkungen der Übertragungsleistung bis hin zum Verlust der Verbindung führen [3]. Dadurch kann also kein lückenlos schnelles mobiles Internet gewährleistet werden. Es wurden verschiedene Techniken entwickelt, um diesem Problem entgegenzuwirken und selbst bei schwankender Netzwerkverbindung eine möglichst störungsfreie Benutzung von Applikationen, die auf das Internet angewiesen sind, zu ermöglichen (siehe Kapitel 5 „Aktuelle Arbeiten“). Die grundlegende Idee der Arbeiten ist im Wesentlichen, Vorhersagen über die Verfügbarkeit der Verbindung zu treffen und/oder über die Daten, die in naher Zukunft benötigt werden. Diese Arbeiten stellen dabei jeweils verschiedene Ansätze dar, wie man das Problem der nicht uneingeschränkten Internetverbindung angehen kann.

In Kapitel 2 wird versucht, einen Überblick über die Problematik der eingeschränkten Internetverbindung zu schaffen. Anschließend werden die Ziele, die sich aus der Problemlage motivieren lassen, in Kapitel 3 aufgeführt und erläutert. Bei dem Versuch, die Ziele zu erreichen, ergeben sich weitere Fragestellungen, die in Kapitel 4 angesprochen werden. Im Anschluss daran werden in Kapitel 5 verschiedene Arbeiten aufgeführt, die sich der Probleme annehmen. Diese versuchen, die vorher genannten Ziele bestmöglich zu erreichen. Um einen Überblick über die Arbeiten zu erhalten, befindet sich am Ende des Kapitels eine Tabelle, die einen kompakten Überblick ermöglichen soll. Das Fazit in Kapitel 6 stellt dann den abschließenden Teil dieser Arbeit dar.

2 Hintergrund

Der folgende Abschnitt befasst sich zunächst mit Grundlagen. Es werden technische Rahmenbedingungen aufgezeigt, die zum Einen zu der Problemstellung führen, mit der sich Predictive Pre-Caching befasst, und zum Anderen auch die Hürden und Grenzen für die Lösungsansätze darstellen.

2.1 Problemstellung

Zunächst wird erläutert, weshalb eine Internetverbindung für eine Applikation notwendig sein kann und weshalb es zu Problemen führt, wenn diese nicht uneingeschränkt verfügbar ist.

Notwendigkeit einer Verbindung Die Funktionalität vieler Apps hängt von Daten ab, die via Internet bezogen werden. Zum Beispiel benötigt eine App, die stets möglichst aktuelle Nachrichten anbieten möchte, eine Internetverbindung, um die jeweils aktuellsten Nachrichtenartikel abrufen zu können. Ein anderes Beispiel sind Apps, die via Internet Zugriff auf multimediale Inhalte wie z.B.

Musik, Podcasts oder Videos ermöglichen. Anbieter wären hierfür beispielsweise Spotify für Musik oder YouTube für Videos. Diese bieten jeweils Applikationen für verschiedene mobile Plattformen an, um den Zugriff auf die jeweils angebotenen Inhalte möglichst komfortabel zu gestalten.

Wenn man internetgebundene Apps nun unterwegs verwenden möchte, ist man, falls kein Predictive Pre-Caching oder Ähnliches verwendet wird, stark von der Verfügbarkeit und Qualität mobiler Netze abhängig. Denn ohne Verbindung kann man weder aktuelle Nachrichten, noch sonstige Daten aus dem Internet abrufen. Aber auch eine schlechte/schwache Verbindung kann eine Einschränkung für die Funktionalität darstellen. Man kann zwar Daten abrufen, allerdings nur mit einer begrenzten Geschwindigkeit. Bezogen auf das oben genannte Beispiel der Nachrichten-App würde das bedeuten, dass man lange auf die Darstellung der Nachrichten warten müsste. Bei multimedialen Streaming-Anwendungen würde eine eingeschränkte Verbindung die Wiedergabe immer wieder unterbrechen. Dies wird dadurch verursacht, dass die Datenrate der Inhalte höher sein kann, als die, die das Netz in dem Moment leisten kann. Störungen dieser Art möchte man vermeiden.

Resultierende Probleme Diese Notwendigkeit, eine (ausreichend leistungsfähige) Internetverbindung zu haben, wird im Alltag immer wieder zum Problem. Da die Mobilität in drahtlosen Netzen zu einer Beeinträchtigung der Verbindungsleistung führt [3], kann man als Entwickler einer Applikation nicht mit einer ständig verfügbaren, uneingeschränkt leistungsstarken Internetanbindung rechnen. Daher gilt es, Wege zu finden, trotz der eingeschränkten Konnektivität, möglichst viel Funktionalität durch Applikationen, die auf das Internet angewiesen sind, zu bieten.

Eine mögliche Lösung für diese Probleme ist das Prinzip des Predictive Pre-Caching. Wie der Name schon vermuten lässt, besteht das Prinzip daraus, vorherzusagen (*„predictive“*), wann und/oder welche Daten benötigt werden, und diese dann vorabzuladen (*„pre-caching“*) und für die Benutzung durch Applikationen zu speichern bzw. bereit zu halten . Um dies zu ermöglichen, gilt es (je nach Anwendungsfall), diverse Probleme zu lösen und sich Fragen zu stellen, die das System letztendlich charakterisieren (siehe Abschnitt 4 „Fragestellungen“).

2.2 Limitierte Hardware

Ein Problem, dem man sich gegenüberstellen muss, ist die limitierte Hardware auf mobilen Endgeräten. Trotz der technischen Weiterentwicklung vieler Hardwarekomponenten unterliegen Smartphones dennoch maßgeblichen Beschränkungen wie begrenzte Rechen- und Akkuleistung [12]. Die Mobilität, die man von Smartphones abverlangt, führt also zu diversen Einschränkungen, was Hard- und Software betrifft.

Da die Kapazität eines Akkus beschränkt ist, gilt es grundsätzlich, mit den vorhanden Ressourcen möglichst sparsam umzugehen, um eine möglichst lange Laufzeit des Gerätes ohne zwischenzeitlichen Ladevorgang zu erreichen. Was

die begrenzte Hardware auf Smartphones darüber hinaus für Auswirkungen hat, wird im Folgenden kurz besprochen. Die genannten Aspekte werden dabei hauptsächlich im Hinblick auf Predictive Pre-Caching betrachtet.

Rechenoperationen Im Kontext von Predictive Pre-Caching bedeutet das, dass der Rechenaufwand für Vorhersagen möglichst gering gehalten werden sollte. Denn jeder Abruf von Sensordaten und jede Rechenoperation, die durchgeführt werden, benötigen Energie, die nur begrenzt zur Verfügung steht. Ein weiteres Argument für sparsame und effiziente Algorithmen ist die Gegebenheit, dass im Regelfall die Leistung des Prozessors und des Arbeitsspeichers auf mobilen Geräten ohnehin geringer ist, als beispielsweise bei Desktop-Rechnern.

Traffic Auch der Datenaustausch über mobile Netze stellt eine Belastung für den Akku dar. Zusätzlich kommt hierbei noch hinzu, dass der Verbrauch von Datenvolumen oftmals durch Mobilfunktarife begrenzt ist. Ein Grund für den Einsatz von Volumengrenzen ist das Ziel, unnötig hohe Belastung der Netzkapazität zu verringern [6]. Somit gilt auch hier, dass eine sparsame Nutzung der verfügbaren Ressourcen unabdingbar ist und somit eine wichtige Rolle spielt.

Ein weiteres Problem ist, dass manche Applikationen nur arbeiten, wenn beispielsweise WLAN verwendet werden kann. Ein Beispiel hierfür ist der Download von Prime Instant Video-Titeln. In der Hilfe findet man folgende Information: *„Stellen Sie eine WLAN-Verbindung her. Prime Instant Video-Titel können nicht über einen mobilen Datendienst wie 3G oder 4G heruntergeladen werden.“*[1] Ist eine solche Verbindung nicht immer verfügbar, kann es von Interesse sein, die Nichtverfügbarkeit vorherzusagen, um während der Verfügbarkeit die Daten zu buffern.

Speicher Da es im Kontext von Pre-Caching schwer ist, herauszufinden, welche Daten in Zukunft genau benötigt werden [8] (falls dies nicht unmittelbar aus dem Kontext der App hervorgeht), kann man diesbezüglich nicht immer hundertprozentig korrekte Vorhersagen treffen. Allerdings ist es auch nicht sonderlich praktikabel, „auf gut Glück“ viele Daten, die eventuell benötigt werden könnten, vorab zu laden, da nur begrenzt viel Speicherplatz auf mobilen Geräten zur Verfügung steht.

Stattdessen ist das angestrebte Ziel, möglichst präzise Vorhersagen über die in naher Zukunft benötigten Daten zu treffen, um ungenutzten Speicher und überflüssigen Aufwand zu minimieren.

Konsequenz Ein optimales Pre-Caching sähe also vermutlich so aus, dass generell nur vorab geladen wird, wenn es notwendig ist. Befindet man sich in einer Situation, in der Daten vorab geladen werden müssen, sollten nur exakt die Daten geladen werden, die benötigten werden. Außerdem müsste das Herunterladen dann passieren, wenn die Verbindung am schnellsten bzw. am kostengünstigsten ist. Der dadurch entstandene Rechenaufwand sollte minimal sein.

Es gilt also, Ideen und Algorithmen zu finden, die in den genannten Kategorieren jeweils möglichst sparsam, aber dennoch effizient und präzise genug sind, um den Anforderungen gerecht zu werden.

3 Ziele

Im Bereich des Predictive Pre-Caching gibt es verschiedene Ziele, die man erreichen möchte. Welche es im Einzelnen gibt und welche Vorteile man sich von ihnen erhofft, wird im Folgenden dargestellt. Grundsätzlich gilt hierbei, dass das Erreichen der jeweiligen Ziele vom Bedarf der Anwendung abhängt. Dementsprechend variiert auch die Wichtigkeit der jeweiligen Ziele von Fall zu Fall.

3.1 Offline-Zeit überbrücken

Das Vorhaben, Zeit zu überbrücken, in der keine Internetverbindung besteht, ist ein zentrales Ziel verschiedenster Predictive Pre-Caching-Techniken. Ziel dabei kann beispielsweise sein, Funktionalität von Applikationen zu erhalten [9] oder möglichst lückenlose Musikwiedergabe zu ermöglichen [4].

Ein Teilziel der Überbrückung von Offline-Zeiten ist die Vorhersage dieser Zeiträume. Um auf einen Netzwerkausfall mit entsprechenden Maßnahmen reagieren zu können, muss dieser rechtzeitig vorhergesagt werden. Denn sobald keine Internetverbindung mehr verfügbar ist, ist es nicht mehr möglich, Daten für den zu überbrückenden Zeitraum herunterzuladen [4]. Daher spielt die Genauigkeit der Vorhersagemodelle eine zentrale Rolle für den Erfolg des Pre-Cachings.

3.2 Latenz verringern

Eine weitere Absicht, die man verfolgt, ist die Reduzierung der Latenz. Als Benutzer möchte man, wenn man mobil mit dem Smartphone surft, eine möglichst geringe Verzögerung beim Seitenaufbau erleben. Denn jede Verzögerung stellt eine ungewollte Wartezeit für den Nutzer dar, was sich wiederum negativ auf das Nutzungserlebnis auswirkt. Denn gerade wenn man unterwegs schnell zwischendurch Informationen abrufen möchte, sind Wartezeiten besonders hinderlich.

3.3 Datenvolumen sparen

Für Smartphone-Nutzer kann es von Interesse sein, Datenvolumen, das über das Mobilfunknetz anfällt, zu minimieren. Mögliche Gründe könnten sein:

- Das Datenvolumen ist vom Netzanbieter (monatlich) begrenzt
- Keine oder nur langsame Verbindung verfügbar
- Man möchte bereits laufende Downloads möglichst wenig einschränken bzw. verlangsamen
- Der aktuell verwendete Mobilfunktarif bietet keinen oder teuren Internetzugang

Möchte man nun das anfallende mobile Datenvolumen minimieren, aber dennoch das Internet auf dem Smartphone nutzen, bietet sich die Möglichkeit an, für das Pre-Caching auf eine WLAN-Verbindung zurückzugreifen.

3.4 Benötigte Daten vorhersagen

Wenn die zu cachenden Daten sich nicht direkt aus dem Kontext der Applikation ergeben, sondern mehr Flexibilität und Individualität gefordert ist, kommen Systeme zum Einsatz, die versuchen, möglichst präzise Vorhersagen über die in (naher) Zukunft benötigten Daten zu machen.

Guo et al. [5] ziehen sogar den Schluss, dass *„[. . .] der Erfolg von Web Prefetching hauptsächlich von der Genauigkeit der Web-Page-Vorhersage abhängt.“* (*„Consequently, the success of Web prefetching relies mainly on the Web page prediction accuracy.“* [5] S. 32). Dies begründen sie damit, dass das prefetchen von ungenutzten Daten eine ineffiziente Nutzung der Ressourcen darstellt, und somit die Zugriffszeit auf die tatsächlich geforderten Webseiten, und damit auch die gesamte Latenz, erhöht [5]. Dies lässt sich im Wesentlichen auch auf Pre-Caching übertragen. Auch hier stellt das Herunterladen und Cachen von unbenutzten Daten eine ineffiziente Nutzung von Ressourcen dar. Da hierbei auch zusätzlich der Cache unnötig belastet und somit verlangsamt wird, gilt es umso mehr, falsche Vorhersagen möglichst zu vermeiden.

3.5 Nutzererlebnis verbessern

Ein Teil der genannten Ziele, wie beispielsweise das Überbrücken von Offline-Zeiten oder die Verringerung von Latenz, tragen direkt zum Nutzungskomfort einer App bei. Somit kann Predictive Pre-Caching auch zu einem besseren Nutzererlebnis führen.

Je nach Anwendungskontext, Nutzerverhalten und Erwartungshaltung des Nutzers tragen manche der genannten Punkte mehr oder weniger zu einem guten Nutzungserlebnis bei. Um eine als möglichst gut empfundene App anbieten zu können, gilt es also, die Ziele entsprechend abzuwägen und an den Bedarf der Nutzer anzupassen.

4 Fragestellungen

Bei der Erstellung eines Predictive Pre-Caching Systems gilt es, diverse Fragen zu beantworten, die bestimmen, welche Funktionalität letztendlich benötigt wird. Wie solche Leitfragen aussehen können und vor allem welche Folgen sie mit sich ziehen, wird im Folgenden (beispielhaft) dargestellt.

Sollte man verschiedene Daten(typen) unterschiedlich handhaben? Wenn Daten möglichst aktuell sein müssen, wie beispielsweise Verspätungsinformationen von Verkehrsverbindungen, verlieren diese mit der Zeit sehr schnell an Relevanz und eignen sich daher kaum zum Pre-Cachen über längere Zeiträume.

Hieraus ergibt sich die Frage, wie lange genau die Daten aktuell bzw. für den Nutzer von Relevanz sind. Je nach zeitlicher Dauer der Relevanz ergeben sich die Optionen, die man wählen kann, um Daten vorab zu laden. Sind die Daten über längere Zeit für den Nutzer interessant, wie z.B. Musik oder Videos, könnte man diese frühzeitig per WLAN-Verbindung herunterladen. Müssen die Daten jedoch relativ aktuell sein, ist man oftmals situationsbedingt an das Mobilfunknetz gebunden und unterliegt somit wieder den entsprechenden Einschränkungen.

Es wird deutlich, dass mit den Daten auch direkt die Rahmenbedingungen für die Systeme festgelegt werden und man somit individuelle, auf die jeweiligen Probleme angepasste Lösungen benötigt.

Welche Daten werden genau benötigt? Je nach Anwendungsfall ist es unterschiedlich schwierig, vorherzusagen, welche Daten genau vorab geladen werden sollen. Für lückenlose Medienwiedergabe könnte man beispielsweise einfach nur die aktuelle Datei oder auch zusätzlich n folgenden Dateien der aktuellen Playlist vorab laden. Möchte man jedoch flexiblere und vor allem individuellere Daten cachen, gilt es eine Informationsquelle zu finden, aufgrund der man Vorhersagen über Inhalte treffen kann. Ein Beispiel hierfür ist das Benutzen von Stichwörtern in Kalendereinträgen [8]. Diese Stichwörter werden verwendet, um individuell auf den Benutzer abgestimmte Inhalte zu pre-cachen.

Soll flexibel auf einen Ausfall reagiert werden? Wenn flexibel auf eine Änderung der Netzanbindung reagiert werden soll, sollte diese möglichst verlässlich, präzise und frühzeitig vorhergesagt werden, damit man rechtzeitig darauf reagieren kann. Je früher die Vorhersage stattfindet, desto eher kann man beispielsweise die Vorzüge einer WLAN-Verbindung nutzen und ist nicht nur auf das Mobilfunknetz angewiesen. Wenn die Vorhersage genauer ist, müssen weniger unbenötigte Daten heruntergeladen und gecached werden. Somit spart man den Verlust von Rechenzeit, Datenvolumen, Akku und Speicher.

Ist das System effizient genug? Um eine App letztendlich in der Praxis einsetzen zu können, muss man sich der Frage stellen, wie tauglich sie im Alltag ist. Ein ganz entscheidender Faktor hierbei ist, wie sparsam bzw. effizient sie ist. Wenn beispielsweise Berechnungen zu kostspielig sind und dadurch der Ressourcenbedarf zu hoch wird, wird die App in der Praxis kaum zum Einsatz kommen. Welche Rolle der sparsame Umgang mit Hardwareressourcen spielt, wurde bereits in Abschnitt 2.2 dargestellt.

5 Aktuelle Arbeiten

In diesem Kapitel werden einige Beispiele aufgeführt, die sich mit dem Thema Predictive Pre-Caching auf verschiedene Arten befassen. Es werden jeweils die grundlegende Idee bzw. Funktionsweise geschildert, um einen Überblick über die verschiedenen Arbeiten zu bekommen.

5.1 Offline-Zeitraum vorhersagen

Im Jahr 2014 haben Gordon et al. [4] gezeigt, dass es es möglich ist, mit einem auf Aktivitätserkennung basierenden System korrekte Vorhersagen über die Verfügbarkeit von Netzen zu treffen. Dieses System kann maschinell lernen, welche Verhaltensmuster zu Veränderungen der Verfügbarkeit von Netzen führt. Dies geschieht, indem quantifiziertes Verhalten auf Muster hin untersucht wird, die zu Verbindungsänderungen führen. Taucht nun im Alltag ein solches, vorher als verbindungsändernd erkanntes Verhaltensmuster auf, kann eine Vorhersage über die Verbindungsqualität, die in naher Zukunft zu erwarten ist, getroffen werden.

Eine Besonderheit des Ansatzes ist, dass das System in Situationen funktionieren kann, in denen Vorhersagen scheitern, die auf Zeit- oder Ortsdaten basieren. Somit könnte diese Vorhersage-Methode eine sinnvolle Ergänzung für zeit- oder ortsbasierte Vorhersagemodelle darstellen (die Kombination dieser Ansätze wurde in [4] nicht untersucht).

5.2 Offline-Zeit überbrücken

Kouici et al. [9] haben ein service-orientiertes cache-management-System für CORBA *(Common Object Request Broker Architecture)* Komponenten vorgestellt, das es ermöglichen soll, Funktionalität trotz Verbindungsverlust zu erhalten. Dazu werden Daten, oder je nach Rahmenbedingungen und Bedarf auch Code, des Services auf dem Endgerät gecached. Diese Daten liegen dann in Form einer Proxy-Komponente, die in der Arbeit als *„discomponent“* bezeichnet wird, lokal auf dem Endgerät vor. Hierzu müssen Entscheidungen getroffen werden, wie z.B. welche Daten welche Priorität haben oder ob Komponenten überhaupt notwendigerweise während eines Verbindungsverlustes auf dem Endgerät vorliegen müssen. Als Entscheidungsgrundlage hierfür dienen Metadaten, die die Komponenten beschreiben. Außerdem wird ein Ansatz vorgestellt, wie man sowohl Abhängigkeiten innerhalb von Services als auch serviceübergreifende Abhängigkeiten als Graph darstellen kann. Dieser Graph kann dann wiederum als Informationsquelle für die Cache-Mechanismen dienen, welche Daten für welche Funktionen benötigt werden. Bei dieser Arbeit ist die DOMINT-Plattform zuständig für die Synchronisation der Daten, nachdem wieder eine Verbindung zum Nutzer besteht. Dies ist nötig, da die Interaktion des Nutzers während der Phase ohne Netzanbindung Daten verändert haben könnte.

5.3 Latenz verringern

Zu dem Thema der Latenzverringerung werden im Folgenden zwei verschiedene Arbeiten vorgestellt. Ein Ansatz, der Proxies als Cache verwendet [10], und ein Ansatz, der die Benutzung von Applikationen vorhersagt, um bereits vor der eigentlichen Benutzung Inhalte vorab laden zu können [11].

Proxies als Cache Um die Latenz auf mobilen Endgeräten zu verringern, haben Li et al. [10] einen Caching-Mechanismus vorgestellt. Dieser verwendet Proxies als Cache, die sich in der Nähe des Endgerätes befinden. Wenn Verbindungen bzw. Datenanfragen nun von dem Proxy anstelle des eigentlichen Servers verarbeitet werden, verringert sich die Latenz sowohl aufgrund der entstandenen Lokalität der Daten als auch dadurch, dass Anfragen, die die selben Daten fordern, bereits auf dem Proxy gecached sein können. Dabei entsteht die Herausforderung, das Cachingverhalten der Proxies unter Berücksichtigung der Bewegung des Endgerätes so zu optimieren, das insgesamt eine Leistungssteigerung möglich wird. Um eine möglichst gute Performance zu erreichen, basiert das Verhalten der Proxies (also Entscheidungen über das Handhaben der Daten) in der vorliegenden Arbeit auf einem heuristische Verfahren, dass wiederum durch die gesammelten statistischen Daten beeinflusst wird.

Application prediction Parate et al. [11] versuchen vom Benutzer wahrgenommene Ladezeiten zu minimieren, indem sie vorhersagen, welche Applikation ein Nutzer wann verwenden wird. Diese Vorhersage passt sich an den Nutzer an, indem die Nutzung der Apps analysiert wird. Somit kann man, bereits bevor der Nutzer die App verwenden möchte, Inhalte der App herunterladen. Wenn der Nutzer dann die Applikation startet, ist im Optimalfall der Inhalt bereits lokal gecached. Somit kann der Inhalt direkt präsentiert werden, was dazu führt, dass die Dauer des Herunterladens nicht zur Wartezeit für den Nutzer wird. Die Funktion beruht im Wesentlichen auf zwei Modellen. Das erste Modell wird verwendet, um die Verwendung von Apps vorherzusagen. Hierfür orientieren sich die Autoren an einem Algorithmus namens *Prediction by Partial Match (PPM)*. Dieser Algorithmus dient eigentlich der Komprimierung von Texten. Um damit Vorhersagen über die Verwendung von Applikationen treffen zu können, passen die Autoren den Algorithmus entsprechend an. Daraus entsteht das adaptiv lernende Vorhersagemodell *App Prediction by Partial Match (APPM)*, das vorhersagen kann, welche Applikation wahrscheinlich als nächstes verwendet wird. Das zweite Modell *Time Till Usage (TTU)* ist ein zeitliches Modell. Ziel dieses Modells ist es, die Wahrscheinlichkeit zu bestimmen, zu welchem Zeitpunkt eine gegebene Applikation verwendet werden wird. Die Kombination dieser beiden Modelle ermöglicht dann sowohl die Vorhersage, welche App als nächstes verwendet werden wird, als auch wann dies voraussichtlich der Fall sein wird.

5.4 Benötigte Daten vorhersagen

Die Arbeit von Komninos und Dunlop [8] beschreibt einen pre-caching Agenten, der basierend auf Kalender-Einträgen und den darin enthaltenen Stichwörtern web-queries formuliert und somit Daten passend zu dem Kalendereintrag bzw. den Stichwörtern vorab lädt. In dem durchgeführten Experiment haben Benutzer knapp 30% der angebotenen Dokumente tatsächlich geöffnet. Der Autor schließt daraus, dass man also durchaus Vorhersagen basierend auf Kalenderdaten treffen kann, die für den Nutzer von Relevanz sein können. Um das Ergebnis weiter

zu verbessern, schlagen die Autoren vor, das System mithilfe zusätzlicher Informationsquellen wie Logs von Instant Messengern oder E-Mails zu erweitern. Vor allem in der Analyse von E-Mails sieht der Autor eine sehr wertvolle Informationsquelle, da mithilfe dieser Kommunikationsmittel oftmals Meetings und Aufgaben geplant werden und dementsprechend relevante Informationen beinhalten. Die Analyse von Informationsquellen dieser Art würde jedoch den Einsatz von Textverarbeitungstools erfordern. Dies würde wiederum zu höherem Rechenaufwand führen.

5.5 Überblick

Ein Überblick über die angesprochenen Arbeiten findet sich in Tabelle 1 auf Seite 12. Die Tabelle beinhaltet verschiedene Merkmale der jeweiligen Arbeiten. Um einen kompakten Überblick zu ermöglichen, sind die Merkmale jeweils nur kurz beschrieben bzw. beinhalten sie einen Verweis auf die jeweilige Seite der Arbeit.

6 Fazit

Die Tatsache, dass der Bedarf an mobilem Internet nicht immer vollkommen gedeckt werden kann, führt zu diversen Problemen. Beispielsweise möchte man Offline-Zeiträume überbrücken können, indem man die Funktionalität einer App erhält. Hier gilt es, sowohl den Netzausfall, als auch die Daten für die Überbrückung rechtzeitig zu bestimmen. Die Schwierigkeit, genaue Vorhersagen zu treffen, variiert dabei mit den Anforderungen bzw. Rahmenbedingungen der Applikation. Ein weiteres Ziel kann es sein, die Latenz von Anwendungen zu verringern. Auch hier müssen Vorhersagen getroffen werden, um ungewollten Wartezeiten rechtzeitig zu entgegnen.

Zusätzlich erschwert wird die Erfüllung der vorgestellten Ziele durch die Einschränkungen mobiler Geräte wie z.B. begrenzte Rechenleistung, Traffic, Speicher und Akkuleistung. Aufgrund dieser Gegebenheiten sind intelligente Ansätze gefragt, die möglichst effizient sind.

Die besprochenen Arbeiten haben gezeigt, dass es möglich ist, diese Probleme auf vielfältige und kreative Art zu lösen und dass es durchaus machbar ist, korrekte Vorhersagen zu treffen. Durch die schnelle Weiterentwicklung mobiler Hardware öffnen sich außerdem weiterhin immer neue Spielräume für die Verbesserung der bestehenden Ansätze.

Obwohl die Arbeiten sehr ausgereift und praxistauglich sind, ist Predictive Pre-Caching bisher kaum oder nur ansatzweise im Einsatz. Ein möglicher Grund könnte sein, dass das Verhalten eines Menschen in vielen Fällen (noch) nicht genau genug vorhergesagt werden kann, und somit eine zu hohe Ungenauigkeit in der Praxis erreicht wird. Ein weiterer Faktor, der die Anwendung von Predictive Pre-Caching-Systemen erschwert, könnte die geringe Akzeptanz durch Nutzer sein. Um eine möglichst individuelle Vorhersage treffen zu können,

sind entsprechend persönliche Daten nötig. Das Sammeln und Auswerten solcher personenbezogener Daten erfordert vom Nutzer Vertrauen in die Anbieter einer App. Wenn ein Nutzer dieses notwendige Vertrauen jedoch nicht aufbringen kann oder will, wird er die auch App nicht verwenden. Da viele Mobilfunktarife mit einer Volumenbegrenzung belegt sind, werden vermutlich bisher generell weniger Cloud-Dienste mobil verwendet. Daher ist die Nachfrage nach Predictive Pre-Caching-Systemen auch noch entsprechend gering.

Aufgrund der Zunahme von Online-Angeboten kann man jedoch davon ausgehen, dass Predictive Pre-Caching in Zukunft zunehmend wichtiger sein wird. Somit wird auch die Weiterentwicklung bzw. allgemein das Angebot von Pre-Caching Techniken auch weiterhin eine wichtige Rolle spielen.

Arbeit	**Ziel(e)**	**Nutzereinfluss**	**Grenzen**	**Performance**	**Anmerkungen**
Gordon et al. [4]	Verbindungsausfall vorhersagen und überbrücken, indem eine Unterbrechung des Musikstreams verhindert wird [4]	Konditionieren des Systems möglich	Kann Aktivitäten erkennen aber nur bedingt örtliche Unterschiede [4] S. 155	Offline-Zeiträume wurden zu 100% vorher erkannt; Die Vorhersagen passierten im Schnitt ca. 8,5 Minuten vor dem Eintreffen des Ereignisses [4] S. 156	Kann funktionieren, wenn GPS- oder Uhrzeitbasierte Modelle scheitern [4] S. 155
Kouici et al. [9]	Funktionalität von Komponentenbasierten services trotz Verbindungsverlust erhalten [9]	Kein Nutzereinfluss	*„[. . .] our work currently assumes that the cache size is large enough to deploy developer-necessary services. We are currently investigating this limitation.“* [9] S. 1338	Das Extrahieren der Metadaten aus dem Graph erzielte sehr gute Ergebnisse (siehe [9] S. 1331ff)	Es muss auf die Konsistenz der Daten geachtet werden [9] S. 1324
Li et al. [10]	Nahegelegene Proxies als Cache verwenden, um Latenz zu verringern [10]	Kein Nutzereinfluss	Keine Angabe	*„ [. . .] $O(Dln^2D)$, which is acceptable for most (if not all) mobile learning applications“* [10] S. 580. (D = Tiefe des Suchbaumes [10] S. 576)	Heuristische Suche durch statistische Daten beeinflusst [10] S. 577
Komninos und Dunlop [8]	Daten basierend auf Stichwörtern in Kalendereinträgen cachen [8]	Eintragen der Kalenderdaten erfolgt durch Nutzer	*„[. . .] such a system in its own right would not be able to completely satisfy all of a user's internet content needs or desires [. . .]“* [8] S. 510	Cache hit-rate von knapp 30% [8] S. 510	Könnte durch zusätzliche Informationsquellen (E-Mail, SMS, . . .) verbessert werden [8] S. 510
Parate et al. [11]	Nutzung von Applikationen vorhersagen; Prefetch während des Entsperrens des Bildschirms [11]	Nutzer kann Apps auswählen, die geprefetched werden sollen [11] S. 281 *(Anmerkung: Beim Testen der App konnte diese Option nicht gefunden werden)*	Die Implementierung an Einschränkungen durch Android gebunden [11] S. 280	Top-5 Vorhersagegenauigkeit APPM: ca. 82%. Median der Frische: 2,7 Minuten [11] S. 281 ff	Im Google Play Store als Applikation namens *AppKicker*[a] erhältlich [11] S. 282

Tabelle 1. Überblick über die angesprochenen Arbeiten.

[a] `https://play.google.com/store/apps/details?id=org.appkicker.app`

Literatur

1. Hilfe und Kundenservice: Download von Prime Instant Video-Titeln, http://www.amazon.de/gp/help/customer/display.html?nodeId=201460820, (aufgerufen am 07.08.2014, 15:57 Uhr)
2. Statistisches Bundesamt (Hrsg.): Pressemitteilung Nr. 089 vom 11.03.2014. Zahl der mobilen Internetnutzer im Jahr 2013 um 43 % gestiegen (2014), https://www.destatis.de/DE/PresseService/Presse/Pressemitteilungen/2014/03/PD14_089_63931.html, (aufgerufen am 07.08.2014, 15:59 Uhr)
3. Forman, G., Zahorjan, J.: The challenges of mobile computing. Computer 27(4), 38–47 (April 1994)
4. Gordon, D., Frauen, S., Beigl, M.: Reconciling cloud and mobile computing using activity-based predictive caching. In: Memmi, G., Blanke, U. (eds.) Mobile Computing, Applications, and Services, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 130, pp. 140–157. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-05452-0_11
5. Guo, Y., Ramamohanarao, K., Park, L.: Web access latency reduction using crf-based predictive caching. In: Liu, W., Luo, X., Wang, F., Lei, J. (eds.) Web Information Systems and Mining, Lecture Notes in Computer Science, vol. 5854, pp. 31–44. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-05250-7_4
6. Harno, J.: Impact of 3g and beyond technology development and pricing on mobile data service provisioning, usage and diffusion. Telematics and Informatics 27(3), 269 – 282 (2010), http://www.sciencedirect.com/science/article/pii/S0736585309000781
7. Hassan, M., Zhao, W., Yang, J.: Provisioning web services from resource constrained mobile devices. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. pp. 490–497 (July 2010)
8. Komninos, A., Dunlop, M.: A calendar based internet content pre-caching agent for small computing devices. Personal and Ubiquitous Computing 12(7), 495–512 (2008), http://dx.doi.org/10.1007/s00779-007-0153-4
9. Kouici, N., Conan, D., Bernard, G.: Caching components for disconnection management in mobile environments. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science, vol. 3291, pp. 1322–1339. Springer Berlin Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-30469-2_32
10. Li, Q., Zhao, J., Zhu, X.: Mobile learning support with statistical inference-based cache management. In: Leung, H., Li, F., Lau, R., Li, Q. (eds.) Advances in Web Based Learning – ICWL 2007, Lecture Notes in Computer Science, vol. 4823, pp. 566–583. Springer Berlin Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-78139-4_50
11. Parate, A., Böhmer, M., Chu, D., Ganesan, D., Marlin, B.M.: Practical prediction and prefetch for faster access to applications on mobile phones. In: Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing. pp. 275–284. UbiComp '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2493432.2493490
12. Qi, H., Gani, A.: Research on mobile cloud computing: Review, trend, and perspectives. CoRR abs/1206.1118 (2012)

Energiehandel mit Fokus auf Erneuerbaren Energien mit Schwerpunkt Marktmodellierung und Simulationstools

Kai Braun*

Betreuer: Yong Ding†

Karlsruher Institut für Technologie (KIT)
Pervasive Computing Systems – TECO
*uldlg@student.kit.edu
†ding@teco.edu

Zusammenfassung. Ziel dieser Arbeit ist es, einen Überblick über die Mechanismen hinter den heutigen Stromgroßhandelsbörsen zu geben, diese zu modellieren und verschiedene Simulationstools vorzustellen. Dabei wird auf die Bildung des Marktpreises am Day-Ahead-Spotmarkt der EPEX eingegangen. Ein besonderer Fokus liegt auf der Integration der Erneuerbaren Energien und deren Auswirkungen auf den Marktpreis und das Übertragungsnetz. Behandelt wird unter anderem der Merit-Order-Effekt der Erneuerbaren, das Nash-Gleichgewicht im Energiehandel, das Simulationstool AMES test bed und der Einsatz von Multiagentensystemen am Beispiel des PowerMatchers.

Schlüsselwörter:
Energiehandel
Spotmarkt
EPEX
Merit-Order
Nash-Gleichgewicht
AMES test bed
PowerMatcher

1 Einführung

1.1 Motivation

Durch den massiven Ausbau an Erneuerbaren Energien in Deutschland in den letzten Jahren kam es an den Stromhandelsbörsen manchmal zu einem erstaunlichen Phänomen: Der Preis für Elektrizität fiel unter die Grenze 0€. Wie kann es zu einer solchen Anomalie kommen? Dieses Phänomen weckte mein Interesse an den Mechanismen hinter den Stromhandelsbörsen und deren Funktionsweisen, denen in dieser Arbeit auf den Grund gegangen wird.

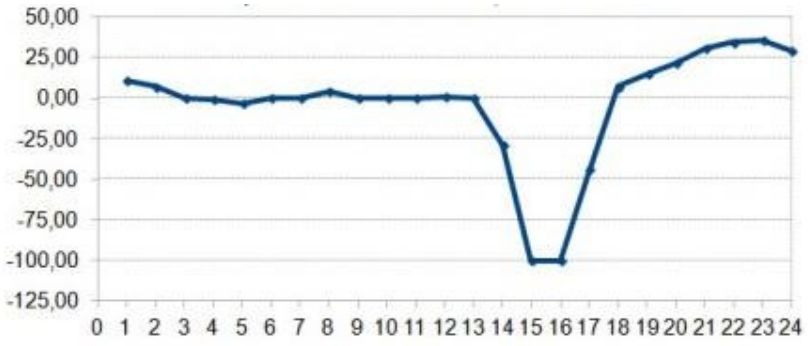


Abb. 1. "Strompreis in Euro pro Megawattstunde für die 24 Stundenprodukte am 16.06.2013, Quelle EEX.de" [21]

1.2 Einschränkungen

Da im allgemeinen Energiehandel neben Elektrizität auch Energieträger wie zum Beispiel Öl, Gas oder auch Holz berücksichtigt werden, der Fokus dieser Arbeit jedoch auf der Modellierung und Simulation von Stromhandelsbörsen liegt, gelten folgende Einschränkungen: Die Arbeit bezieht sich nur auf den Energieträger Elektrizität, da dieser besondere Eigenschaften, wie zum Beispiel die schwierige Speicherung, besitzt. Weiter bezieht sich diese Arbeit nur auf den Raum Deutschland ohne Import/Export, da dieser durch das Erneuerbare Energien Gesetz (EEG) und den massiven Ausbau an Kraftwerkskapazitäten im Bereich der Erneuerbaren Energien besonders interessant ist. Der Schwerpunkt liegt auf dem day-ahead Handel, da der intraday Handel und die Energiemärkte zur Vorhaltung von Regelleistung stark von zufällig auftretenden Ereignissen (z.B. Kraftwerksausfällen) abhängen und sich deshalb nur schwer adäquat simulieren lassen.

2 Der deutsche Markt für Elektrizität

2.1 Marktliberalisierung im Bereich Elektrizitätsmärkte

Bis zum Jahr 1998 war der Energiemarkt geprägt durch wenige Energieversorgungsunternehmen, die in ihren zugewiesenen Regionen für die Stromerzeugung, den Stromtransport und die Stromverteilung zuständig waren. Ihnen wurde in diesen Bereichen ein natürliches Monopol zugestanden. Durch Unterzeichnung des Energiewirtschaftsgesetzes im Jahr 1998 wurden die Energieversorgungsunternehmen gezwungen, auch anderen Wettbewerbern diskriminierungsfrei Zugang zur Netzinfrastruktur zu gewähren. Im Zuge dieser Marktliberalisierung mussten die Energieversorgungsunternehmen den Sektor Energietransport buchhalterisch in Tochterunternehmen auslagern oder diesen verkaufen (unbundling) [1] . Die resultierenden Unternehmen, die sogenannten Übertragungsnetzbetreiber, besitzen auch heute noch das Monopol auf Elektrizitätstransport, müssen jedoch jedem Erzeuger und Verteiler Zugang zu ihrem Netz gewähren. Die Übertragungsnetzbetreiber werden durch die Bundesnetzagentur insbesondere auf Diskriminierungsfreiheit und Rechtfertigung der Netzentgelte kontrolliert. [2] Nur durch diese Marktliberalisierung wurde der Energiehandel, wie er in der heutigen Form stattfindet, ermöglicht. Es entstand ein Wettbewerb im Bereich der Energieerzeugung und Verteilung. Außerdem wurden neue Handelsmechanismen und Marktplätze benötigt.

2.2 European Power Exchange (EPEX)

Nach der Liberalisierung des Elektrizitätsmarktes konkurrierten immer mehr Unternehmen in der Stromerzeugung miteinander, und es entstanden Börsen, an denen der erzeugte Strom gehandelt werden konnte. Die europäische Stromgroßhandelsbörse EPEX (European Power Exchange) hat ihren Sitz in Paris und handelt Strom in Deutschland, Frankreich, Österreich und der Schweiz. Zumeist wird die Elektrizität an der EPEX kurzfristig für den folgenden Tag gehandelt. Neben dem day-ahead Geschäft bietet die EPEX auch intraday und Termingeschäfte. Siehe Kapitel 2.4 [Formen des Marktes]. Im Jahr 2013 wurden am day-ahead Markt ca. 250 TWh Strom für Deutschland und Österreich gehandelt.[3]

2.3 European Energie Exchange (EEX)

Die European Power Exchange EEX in Leipzig übergab 2009 den kurzfristigen Strom-Spotmarkt an die EPEX in Paris.[5]. Die EEX handelt im langfristigen Termingeschäft sowohl Features als auch Optionen für den deutschen Energiemarkt. Siehe Kapitel 2.4 [Formen des Marktes] vgl. [6]

2.4 Formen des Marktes

Terminmarkt

Im Terminmarkt werden langfristige Geschäfte gehandelt. Die Zeiträume liegen in der Regel bis zu mehreren Jahren in der Zukunft. Stromerzeuger können hier Strom verkaufen, der in der Zukunft erst erzeugt und abgenommen wird. Dies dient zur strategischen Kraftwerksplanung, bei der große fixe Investitionen für den Bau neuer Kraftwerke berücksichtigt werden können.[7] Man unterscheidet zwei Arten von Termingeschäften: Bei den sogenannten Features ist der Käufer vertraglich verpflichtet in einem festgelegten Zeitraum eine festgelegte Energiemenge zu dem vereinbarten Preis abzunehmen.[8] Die sogenannten Optionen bieten der Käufer das Recht, eine festgelegte Energiemenge in einem festgelegtem Zeitraum zu dem vereinbarten Preis abzunehmen, er ist jedoch nicht verpflichtet dies zu tun. Man unterscheidet Verkaufs- und Kaufoptionen (Put bzw. Call).[9]

Spotmarkt day-ahead

Im kurzfristigen day-ahead Spotmarkt wird der Strom für den nächsten Tag gehandelt. Der Strom wird dabei stundenweise angeboten. Es ist möglich für jede Stunde des folgenden Tages Strom zu erwerben oder zu verkaufen. Erzeuger und Abnehmer sind somit zu einer Schätzung der morgen erzeugten/verbrauchten Energiemenge gezwungen. Des Weiteren wird der Tag in den Baseload und den Peakload unterteilt. Der Baseload ist der Preis für den Zeitraum von 24h des Folgetages. Der Peakload ist der Preis von 8-20 Uhr. Der Preis für jede Stunde des Folgetages wird für Deutschland immer um 12 Uhr mittags ermittelt. Dies erfolgt über eine blinde Auktion, bei der die Angebots- und Nachfragefunktion ermittelt wird. Deren Schnittpunkt bildet den Strom-Einheitspreis für jede Stunde des Folgetages. Siehe dazu Kapitel 3.5 [Preisbildung am Spotmarkt]. Im Jahr 2013 wurden 324 TWh Strom am day-ahead Spotmarkt der EPEX gehandelt, ungefähr 250 TWh davon in Deutschland. [3]

Spotmarkt intraday

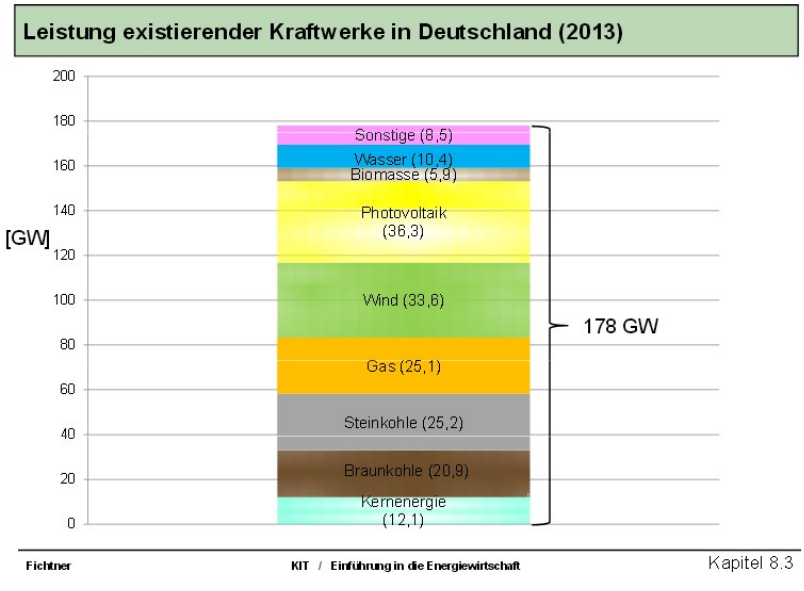
Weiter gibt es die Möglichkeit in Deutschland bis zu 45 Minuten vor Lieferung der Energie diese zu handeln. Beim sogenannten Intraday-Handel wird die Elektrizität sowohl viertelstündlich als auch für eine Stunde zum Handel angeboten.[3] Im Jahr 2013 wurden insgesamt 23 TWh Strom an der EPEX gehandelt, ein Großteil davon in Deutschland. [3]
Der intraday Handel bietet Energieerzeugern und Verbrauchern die Möglichkeit sehr kurzfristig auf unvorhergesehene Ereignisse, wie zum Beispiel Kraftwerksausfälle oder einen plötzlichen Wetterumschwung, zu reagieren.

3 Anforderungen und Herausforderungen der Marktmodellierung

Um den Markt für Elektrizität adäquat modellieren und simulieren zu können, muss man sich zuerst einen Überblick über die Mechanismen und Besonderheiten dieses speziellen Marktes verschaffen. Besondere Anforderungen an den Markt gelten besonders für das Übertragungsnetz - Kapitel 3.3. Weitere Besonderheiten des Elektrizitätsmarktes sind in Kapitel 3.4 ausgeführt. Herausforderungen ergeben sich durch die Integration von Erneuerbaren Energien, insbesondere von Photovoltaik- und Windanlagen, die mittlerweile in Deutschland große Kapazitäten ausmachen. Diese wirken sich auch auf den Großhandelspreis aus - Kapitel 3.6.

3.1 Situation Stromerzeugung in Deutschland

Das Diagramm Abb. 2 zeigt die im Jahr 2013 installierte Kraftwerkskapazitäten in Deutschland. Es sind insgesamt 178 GW Leistung installiert, davon ca 39% Photovoltaik und Wind. Der theoretische maximale Energieertrag von



[10]

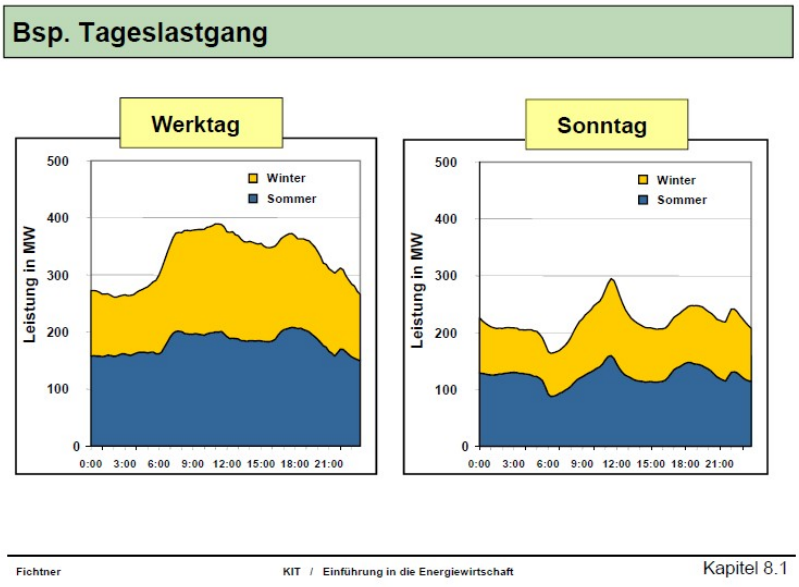
Abb. 2. Leistung existierender Kraftwerke in Deutschland im Jahr 2013 [10]

Photovoltaik- und Windkraftanlagen wird nur unter optimalen Umweltbedingungen erreicht, die jedoch in Deutschland nur relativ selten herrschen. Durch

diese Umweltabhängigkeit kommen die genannten Kraftwerkstypen nur auf wenige Volllaststunden pro Jahr. Siehe Kapitel 3.6 [Besonderheiten durch Integration Erneuerbarer Energien].
Trotzdem kam es 2012 am Tag der Höchstlast (Dezember) nicht zu einem Versorgungsengpass. Die noch vorhandenen Leistungsreserven (nach Abzug von bereits vorgehaltener Reserveleistung für Kraftwerksausfälle und Revisionen) betrugen im Jahr 2012 ca. 11 GW.

3.2 Situation der Stromnachfrage in Deutschland

Die Nachfrage ist zu verschiedenen Zeitpunkten sehr unterschiedlich. Beispielsweise wird im Winter gemittelt mehr Leistung aufgenommen als im Sommer. Man unterscheidet nicht nur die Jahreszeit, sondern sogar Werk- und Feiertag sowie Tag und Nacht. Innerhalb eines Tages schwankt die Nachfrage zusätzlich. Besonders fällt auf, dass die Nachfrage im Winter (gelb) gemittelt größer ist als



[10]

Abb. 3. Der Lastgang (Elektrizitätsnachfrage) eines exemplarischen Werk- und Sonntags im Sommer und Winter [10].

im Sommer. Außerdem sind die zwei typischen Nachfrage-Peaks zur Mittags- und Abendzeit an diesen Tagen zu erkennen.

3.3 Physikalische und technische Einschränkungen des Stromnetzes

Eine Besonderheit des Elektrizitätshandels ist die Leitungsgebundenheit des Gutes. Elektrizität kann nur über ein existierendes Leitungsnetz verteilt werden. Diese Leitungen sind außerdem in ihrer Kapazität begrenzt.
Elektrizität kann nur an bestimmten Netzzugangspunkten ins Übertragungsnetz eingespeist bzw. daraus entnommen werden, da sie erst auf das entsprechende Spannungsniveau gebracht werden muss. Es existieren mehrere Spannungsebenen um die elektrische Energie unterschiedlich lange Strecken ohne große Verluste transportieren zu können.
Um das Netz stabil zu halten, sprich Angebot und Nachfrage auszutarieren, existieren Märkte, auf denen Energieerzeuger oder verbrauchsintensive Abnehmer sehr kurzfristig Regelleistung anbieten. Sie können vom Übertragungsnetzbetreiber innerhalb weniger Minuten ans Netz gebracht oder vom Netz genommen werden um eine stabile Netzfrequenz von 50 Hz beizubehalten.

3.4 Besonderheiten des Elektrizitätshandels

Im Elektrizitätshandel gelten besondere Bedingungen, die am Markt berücksichtigt werden müssen.
Zuerst lässt sich Elektrizität nur sehr eingeschränkt speichern. Bisher vorhandene Speichermöglichkeiten existieren in Deutschland nur in Form von wenigen Pumpspeicherkraftwerken. Diese können in der Marktmodellierung aufgrund ihrer vergleichsweise geringen Speicherkapazität vernachlässigt werden.
Zweitens ist die Elektrizität leitungsgebunden. Sie kann nur an bestimmten Zugangs- und Entnahmestellen in das Netz eingespeist werden. Siehe Kapitel 3.3 [Physikalische und technische Einschränkungen des Stromnetzes].
Um die Versorgungssicherheit zu gewährleisten, muss die bereitgestellte Energiemenge immer der Nachfrage entsprechen. Des Weiteren ist nach *§ 12 Energiewirtschaftsgesetz* gesetzlich festgelegt, dass die Versorgung mit Elektrizität ständig vorhanden ist.
Bei Strom handelt es sich um ein sogenanntes homogenes Gut. Das heißt, dass Strom immer die gleiche Qualität hat, egal welches Kraftwerk oder an welchem Ort er produziert wurde.
Die Elektrizitätsnachfrage ist geprägt durch charakteristische Lastganglinien. Diese sind abhängig von Tageszeit, Wochentag und Jahreszeit. Siehe Kapitel 3.2 [Energienachfrage].
Zuletzt ist der Elektrizitätstransport aufgrund eines sehr hohen Fixkostenanteils für den Aufbau eines Höchstspannungsnetzes noch heute ein Monopol. Siehe Kapitel 2.1 [Marktliberalisierung].

3.5 Preisbildung am Spotmarkt

Die Preisbildung erfolgt durch eine blinde Auktion. Die Energieerzeuger melden die erzeugbare Energiemenge und den entsprechenden Preis der zentralen Börse. An der Börse werden die Kraftwerke aufsteigend nach dem Preis sortiert bis die

Nachfrage gedeckt ist. Das letzte und damit teuerste Kraftwerk, das benötigt wird, bestimmt den Marktpreis. Diesen ermittelten Marktpreis erhält jeder Erzeuger. Er heißt deshalb auch Einheitspreis. [10]
Durch den sehr kurzfristigen Handel an den Strombörsen berücksichtigt der rationale Stromerzeuger bei der Ermittlung seiner Kosten nur die sogenannten variablen Kosten. Diese enthalten im Wesentlichen Brennstoffkosten, Kosten für das Anfahren von Kraftwerken und Kosten für die CO_2 Zertifikate.[11] Folglich sind sie besonders vom Kraftwerkstyp und dessen Wirkungsgrad abhängig. Diese Kosten will der Erzeuger auf jeden Fall decken und bietet deshalb den erzeugten Strom zu den berechneten Grenzkosten an der Börse an. Der dadurch erwirtschaftete Erlös sollte am Ende einer gewissen Zeitperiode die Fixkosten eines Kraftwerkes decken um Gewinn zu erwirtschaften.
Das untere Schaubild 4 zeigt auf der linken Seite verschiedene Kraftwerke aufsteigend sortiert nach ihren Grenzkosten. Diese Funktion wird Merit-Order genannt und wird für jedes handelbare Zeitintervall neu gebildet. Die Nachfragefunktion ist sehr steil, da die Stromnachfrage sehr unelastisch ist. Elektrische Energie wird schlichtweg benötigt, der Preis ist oft nicht wichtig. Das Angebot sind die Kraftwerke aufsteigend sortiert nach ihren Grenzkosten. Da Kraftwerkstypen unterschiedliche Grenzkosten besitzen, entsteht die "Treppenbildung". Das letzte Kraftwerk, das benötigt wird um die Nachfrage zu decken (am Schnittpunkt der Angebots- und Nachfragefunktion), bestimmt den für alle gültigen Marktpreis.

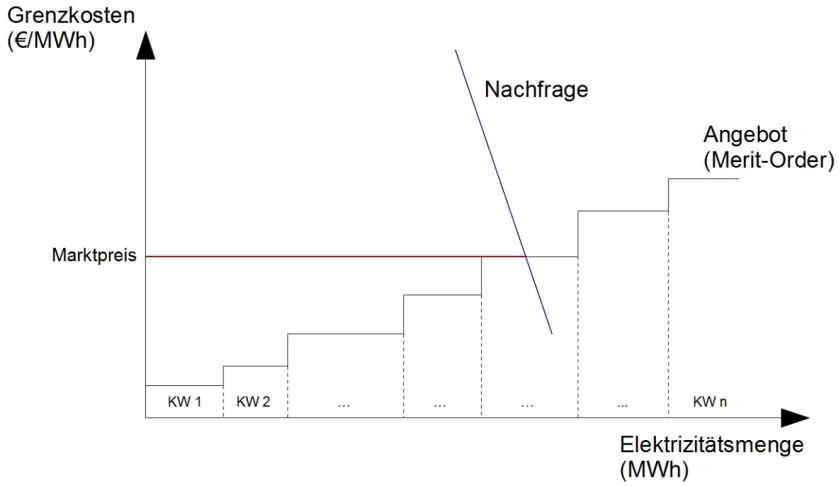


Abb. 4. Das Diagramm veranschaulicht die Marktpreisbildung im Elektrizitätshandel einer Stunde.

3.6 Besonderheiten durch Integration Erneuerbarer Energien

Einspeisemengen und Zeiträume von Photovoltaik- und Windkraftanlagen

Das Angebot an Strom aus Photovoltaik und die Strom-Nachfrage divergieren. Während Photovoltaik-Anlagen besonders in den Mittagsstunden im Sommer viel Strom erzeugen, wird in Deutschland besonders in den Abendstunden des Winters Elektrizität nachgefragt. In diesem Zeitraum ist die von PV-Anlagen erzeugte Energiemenge zu gering um die Elektrizitätsversorgung zu gewährleisten, während im Sommer nur der Nachfrage-Peak zu Mittagszeiten gedeckt wird. Der Energieüberschuss führt dann zu einem Preisverfall an der Börse.[11] Seite 5, Zeile 12.
Windkraft bietet eine saisonal kontinuierlichere Menge an erzeugtem Strom. [13] Seite 29. Bei einer typischen Windkraftanlage wurde im Jahr 2006 von ca. 2200 Volllaststunden ausgegangen, während man bei Photovoltaik-Anlagen nur ca. 1000 Volllaststunden rechnet. Je nach Standort sind bei Windkraftanlagen jedoch bis zu 4500 Volllaststunden möglich.[13] Seite 23.

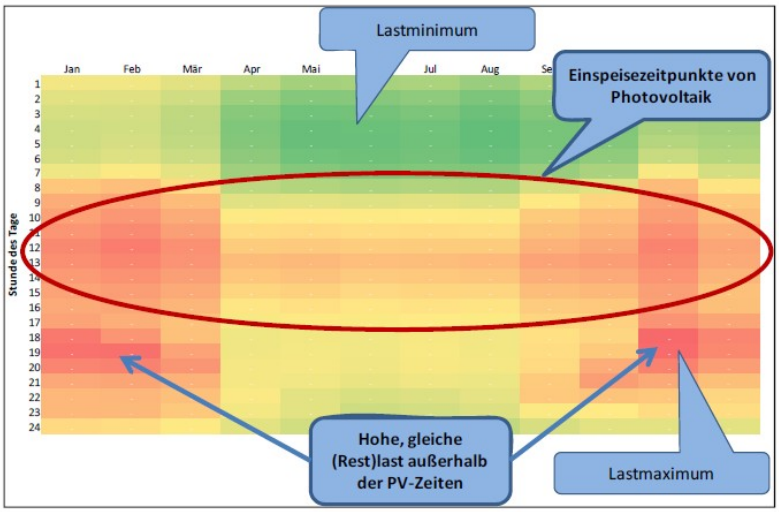


Abbildung 2: Lastgebirge für Deutschland (durchschnittliche Nachfrage (GW) in verschiedenen Stunden des Jahres)

[11]

Abb. 5. Das Diagramm zeigt die Lastverteilung (Nachfrage) für das Jahr 2012 in Deutschland zu verschiedenen Stunden des Tages. Je kräftiger die Rotmarkierung, desto größer die Nachfrage. Die rote Ellipse repräsentiert den durchschnittlichen Einspeisezeitraum von PV-Anlagen. Zu erkennen sind die Nachfrage-Peaks besonders zur Mittagszeit des Jahres, sowie den Abendstunden im Winter [11].

Bevorzugte Einspeisung von Strom aus Erneuerbaren Energien durch EEG und Auswirkungen auf den Börsenpreis

Betrachtet man nun die beiden am weitesten verbreiteten Erneuerbaren Kraftwerkstypen Wind und Photovoltaik, gilt es Verschiedenes zu berücksichtigen:
Zuerst sind in Deutschland im Jahr 2013 fast 70 GW an Kraftwerkskapazitäten von Wind- und Photovoltaik-Anlagen installiert. Dies entspricht 39% der gesamten in Deutschland installierten Kraftwerkskapazitäten, Tendenz weiterhin steigend.
Des Weiteren besitzen beide Kraftwerkstypen variable Stromerzeugungskosten von beinahe Null. Sobald die Anlagen am Netz angeschlossen sind, produzieren sie entweder Strom oder eben nicht. Es wird kein zusätzlicher Brennstoff benötigt, und es entstehen keine Kosten für CO_2-Zertifikate.
Weiterhin ist der Netzbetreiber nach dem EEG verpflichtet jede erzeugte Kilowattstunde aus Erneuerbaren Energien zu einem festgelegten Preis abzunehmen. [14]
Der Netzbetreiber muss nun abgenommenen Strom weiter handeln. Da er ihn auf jeden Fall verkaufen will, bietet er die Elektrizität zu Grenzkosten von Null an der Strombörse an. Wie in Abbildung 6 abgebildet, ist Folgendes zu beobachten: Die von Windkraft- und Photovoltaik-Anlagen zu Grenzkosten von Null erzeugte elektrische Energie wird auf jeden Fall zur Deckung der Nachfrage genutzt. Durch dieses zusätzliche Angebot verschiebt sich die Angebotsfunktion nach rechts. Diese schneidet nun die Nachfragefunktion später. Je mehr Strom aus Erneuerbaren Energien erzeugt und gehandelt wird, desto stärker fällt der Strompreis. Dies wird als Merit-Order-Effekt der Erneuerbaren bezeichnet.[10]

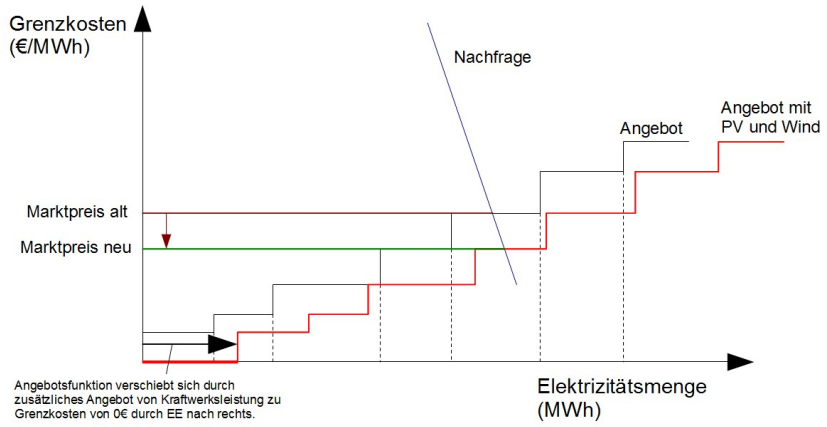


Abb. 6. Der Merit-Order Effekt der Erneuerbaren. Die schwarze Kurve zeigt die Grenzkosten der konventionellen Kraftwerke, die rote enthält zusätzlich PV- und Windkraftanlagen. Die Verschiebung sorgt für einen späteren Schnittpunkt mit der Nachfragefunktion und führt damit zu einer Preisverringerung.

4 Marktmodellierung

4.1 Ziele des Marktes und Nebenbedingungen

Es existieren verschiedene Methoden der Marktmodellierung. Die beiden hier vorgestellten Methoden erfüllen das Ziel, den Marktpreis zu, wobei das Angebot und die Nachfrage gegeben sind. Als Nebenbedingung gilt, dass das Angebot ständig der Nachfrage entspricht. Dies folgt aus den Besonderheiten des Übertragungsnetzes, das keine (ausreichenden) Speichermöglichkeiten für überschüssige Elektrizität besitzt (Kapitel 3.3).

4.2 Modellierung der Nachfrage

Die Elektrizitätsnachfrage ist sehr unelastisch. Darunter versteht man, dass die nachgefragte Energiemenge nur sehr schwach auf Preisänderungen reagiert. Dies liegt daran, dass Strom ständig benötigt wird, er jedoch kaum gespeichert oder substituiert werden kann. Daher geht man von einer sehr steilen linearen Nachfragefunktion an einem Handelszeitpunkt aus.

4.3 Modellierung der Merit-Order Funktion

Die Merit-Order Funktion lässt sich durch ein Optimierungsproblem modellieren. Man minimiert die erzeugte Energiemenge q des Kraftwerks i multipliziert mit dessen Erzeugungskosten zum Zeitpunkt t. Als Nebenbedingung gilt, dass das Angebot der Nachfrage entsprechen muss. Die Funktion c gibt die Kosten für das Kraftwerk i zum Zeitpunkt t mit den Brennstoffkosten P_t. N ist die Menge aller Kraftwerke.

$\min_{q_{it}} \sum_{t=1}^{24} \sum_{i=1}^{N} q_{it} * c_i(P_t, t)$ [15]

Unter der Nebenbedingung:

$\sum_{i=1}^{N} q_{it} = D_t,\ t = 1, ..., 24$ [15]

D_t ist die Nachfrage zum Zeitpunkt t.

Der Marktpreis wird wie im Kapitel Preisbildung beschrieben durch das teuerste Kraftwerk bestimmt, das benötigt wird um die Nachfrage zu decken. Dieser Preis S_t wird durch folgende Formel ermittelt:

$S_t = \max c_i(P_t, t) | q_{it} > 0, t = 1, ..., 24$ [15]

4.4 Nash-Gleichgewicht

Nash-Gleichwicht mit reinen Strategien in der Marktwirtschaft

Das Nash-Gleichgewicht stammt ursprünglich aus der mathematischen Spieltheorie, es lässt sich jedoch auch auf andere Gebiete wie zum Beispiel den Marktwettbewerb anwenden.
In der Spieltheorie existieren mehrere Spieler, die unterschiedliche Strategien wählen können. In den Wirtschaftswissenschaften sind die Spieler Unternehmen, und die Strategien sind deren jeweiligen Produktionsmengen eines homogenen Gutes. Ein Spieler besitzt außerdem eine sogenannte Auszahlungsfunktion (payoff function), die jedoch von den Strategien aller Spieler abhängt. Diese ist in unserem Fall die Gewinnfunktion eines Spielers/Unternehmens, abhängig von den Strategien/Produktionsmengen aller anderen Unternehmen.

Es seien n Spieler und die Menge aller Strategien $\sum$ der Spieler gegeben.
Die Menge der Strategien ist das kartesische Produkt der Strategien der Spieler $1...n$: $\sum = \sum_i \times ... \times \sum_n$
Die Strategie $S_i \in \sum_i$ sei die Strategie des Spielers i.
Die Auszahlungsfunktion g_i des Spielers i sei $g_i(S_1, ..., S_n)$

Das Nash-Gleichgewicht mit reinen Strategien ist dadurch gekennzeichnet, dass die Spieler die Strategien der anderen kennen. Sie können gezielt auf eventuelle Strategieänderungen reagieren und werden immer die für die eigenen Ziele beste Strategie wählen. [16]

Das Nash-Gleichgewicht ist eine Menge an Strategien $S^* = (S_1^*, ..., S_n^*) \in \sum$. bei dem es sich für keinen Spieler i mehr lohnt, seine Strategie zu ändern, da seine Auszahlungsfunktion bereits maximiert ist. Es ist jedoch nicht garantiert, dass bei den reinen Strategien ein Nash-Gleichgewicht existiert. Formal:

$\forall S_i \in \sum_i : g_i(S_1^*, ..., S_i^*, ..., S_n^*) \geq g_i(S_1^*, ..., S_i, ..., S_n^*)$ [16]

Voraussetzungen für die Anwendbarkeit des Nash-Gleichgewichts mit reinen Strategien auf die Marktwirtschaft

Um das Nash-Gleichgewicht mit reinen Strategien auf die Marktwirtschaft anwenden zu können, müssen verschiedene Voraussetzungen erfüllt sein:

- Das gehandelte Gut ist homogen.
- Es sind mehrere Unternehmen (Spieler) beteiligt. Es handelt sich um ein Oligopol.
- Der Marktpreis ist abhängig von der Menge aller produzierten Güter.
- Käufer ziehen das günstigere Produkt vor und sind nicht an Unternehmen gebunden.
- Unternehmen können beliebige Mengen produzieren.
- Spieler und Käufer sind sofort über Preisänderungen informiert.

Beispiel für ein Nash-Gleichgewicht mit reinen Strategien bezüglich Energiehandel

Zusätzlich zu den oben genannten Voraussetzungen gelten im Folgenden weitere Einschränkungen, wie sie kurzfristigen Spotmarkt für Elektrizität herrschen:

- Die produzierten Mengen werden zeitgleich bestimmt.
- Der Marktpreis wird durch einen Dritten mithilfe des Market-Clearing Algorithmus festgelegt.
- In diesem Beispiel zwei Unternehmen (Spieler): i = 1,2
- Die Strategien (Produktionsmengen) x_i des Spielers i sind größer 0.
- Die Nachfrage sei linear: $f(x_1 + x_2) = A - (x_1 + x_2)$
- Die Kosten des Spielers i seien konstante Grenzkosten, keine Fixkosten: $K_i(x_i) = c_i * x_i$
- Die Auszahlungsfunktion sei $\pi_i(x_1, x_2) = x_1 * f(x_1, x_2) - K_i(x_i)$ [17]

Die Unternehmen wollen ihre Auszahlungsfunktion maximieren. Leite π_i nach x_i ab und setze die Ergebnisgleichung gleich 0. Zu prüfen ist noch, ob es sich tatsächlich um einen Hochpunkt handelt.

$\frac{\delta \pi_i(x_1,x_2)}{\delta x_i} = f(x_1 + x_2) + x_i f'(x_1 + x_2) - K_i'(x_i)$ [17]

Das Problem dabei ist, dass die entstehende Funktion, welche die gewinnmaximale Menge des Unternehmens angibt, von der Mengenwahl des Konkurrenten abhängt. Mit den in unserem Fall gewählten Kosten- und Nachfragefunktionen ergibt sich für die Ableitung:

$\frac{\delta \pi_1(x_1,x_2)}{\delta_{x1}} = A - x_1 - x_2 - x_1 - c_1 = 0$
$\Leftrightarrow A - x_2 - c_1 = 2x_1$
$\Rightarrow x_1 = \frac{A - c_1 - x_2}{2} =: R_1(x_2)$

Analog folgt für $i = 2$

$R_2(x_1) := \frac{A - c_2 - x_1}{2}$

Diese Funktion wird Reaktionsfunktion genannt, da sie stets die optimale Reaktion auf die Strategie des Anderen beschreibt. In einem Gleichgewicht darf kein Spieler einen Anreiz haben, seine Strategie zu ändern. In unserem Fall gibt die Funktion an, wie viel Elektrizität es produzieren sollte, gegeben die produzierte Menge des Konkurrenten. Da beide Unternehmen versuchen optimal auf den Anderen zu reagieren, liegt das Nash-Gleichgewicht gerade im Schnittpunkt der beiden Reaktionsfunktionen.
Um das Nash-Gleichgewicht (x_1^*, x_2^*) zu bestimmen löst man nun die beiden Reaktionsfunktionen zum Beispiel durch Einsetzen in die andere nach x_1^* und x_2^* auf.

Für das Nash-Gleichgewicht (x_1^*, x_2^*) gilt:
$x_1^* = R_1(x_2^*) = \frac{A-c_1-x_2^*}{2}$
$x_2^* = R_2(x_1^*) = \frac{A-c_2-x_1^*}{2}$

Löse $x_1^* = R_1(x_2^*)$ nach x_2^* auf:
$x_2^* = A - c_1 - 2x_1^*$

Setze $R_1(x_2^*)$ ein und löse nach x_1^* auf:
$A - c_1 - 2x_1^* = R_2(x_1^*) = \frac{A-c_2-x_1^*}{2}$
$\Rightarrow x_1^* = \frac{A-2c_1+c_2}{3}$

Setze x_1^* in $x_2^* = A - c_1 - 2x_1^*$ ein:
$x_2^* = A - c_1 - 2x_1^* = A - c_1 - 2(\frac{A-2c_1+c_2}{3})$
$= \frac{A-2c_2+c_1}{3}$

Interpretation: Bei den Produktionsmengen x_1^* für Unternehmen 1 und x_2^* für Unternehmen 2 befinden sich beide in einem Gleichgewicht und reagieren optimal aufeinander. Kein Unternehmen hat einen Anreiz seine Strategie zu ändern um einen einseitigen Vorteil zu gewinnen.

5 Modellierungstools

Der reale Energiehandel ist aufgrund der im Kapitel Besonderheiten im Energiehandel näher ausgeführten Eigenschaften komplexer als in den vorgestellten Modellen angenommen und nur schwer mit traditionellen analytischen und statistischen Methoden zu modellieren. [18]
Um den Markt dennoch abbilden zu können, verwendet die Simulationstools keinen zentralen Ansatz, sondern modellieren diesen mit Hilfe von vielen beteiligten Akteuren.

5.1 Agent-based Modeling of Electricity Systems (AMES)

Das erste vorgestellte Simulationstool kombiniert viele Akteure mit maschinellem Lernen, um ökonomisches Handeln zu simulieren. *"Das AMES test bed erlaubt eine systematische Untersuchung des strategischen Handelsverhaltens im Energiegroßhandel mit Berücksichtigung realistisch dargestellter Übertragungsnetze."* Frei übersetzt aus [18].

Systementwurf

AMES test bed simuliert den Energiegroßhandel mithilfe dreier Arten von Marktteilnehmern:

- Der **Independent System Operator (ISO)** betreibt den Großhandel. Sein Ziel ist die Maximierung des Netto-Gesamtüberschusses aller Beteiligten unter Berücksichtigung der Netztopologie.
- Die **Load-Serving Entities (LESs)** sind Großabnehmer, die Energie an bestimmten Entnahmestellen im Netz für ihre Kunden abnehmen. Sein Ziel ist die sichere Bereitstellung der nachgefragten Energie.
- Die **Generation Companies (GenCos)** produzieren die Energie und speisen diese an bestimmten Einspeisepunkten in das Netz ein. Ihr Ziel ist die Maximierung ihrer Nettoerlöse.

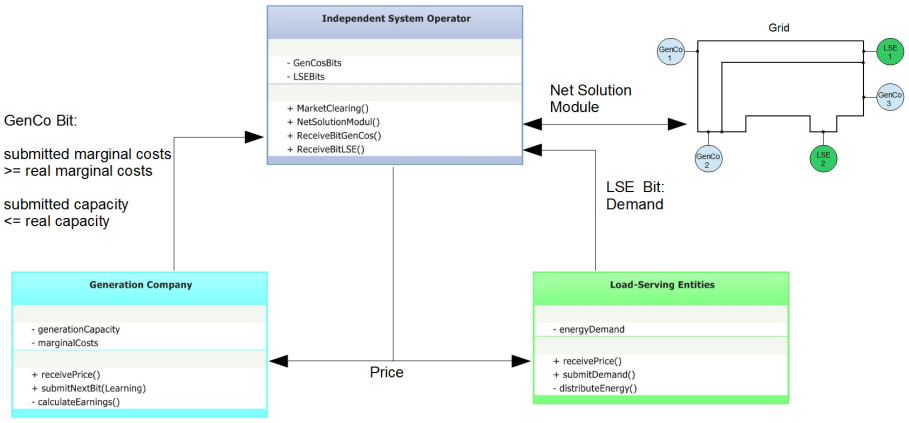


Abb. 7. Ein vereinfachtes Schema des AMES test bed Systementwurfs.

Der zentrale Akteur ist der ISO. Er verteilt den per MarketClearing() unter Berücksichtigung des NetSolutionModule() ermittelten Marktpreis an die anderen Akteure durch Aufrufen der receivePrice() Methode.
Das Übertragungsnetz (Grid) bietet kapazitätsbeschränkte Übertragungsmöglichkeiten für die elektrische Energie. Algorithmisch werden GenCos und LSEs durch Quellen und Senken in einem Flussnetzwerk modelliert werden. Das Flussnetzwerk entspricht dabei dem Übertragungsnetz.
Die GenCos kennen als einzige ihre Grenzkosten (marginalCosts) und Versorgungskapazität (generationCapacity). Mit der Methode calculateEarnings() berechnen sie ihren Erlös. Um diesen zu maximieren, können sie höhere Grenzkosten oder eine geringere Versorgungskapazität an den ISO melden. Die Auswirkungen werden durch einen Lernalgorithmus gelernt submitNextBit(Learning).
Die LSEs kennen die Nachfrage und melden diese mit submitBit() an den ISO. Mit dem erhaltenem Preis berechnen sie die abgenommene Energiemenge und verteilen diese distributeEnergy().

Funktionalität

Es wird der day-ahead Handel simuliert. Zu Beginn des Tages D melden die LSEs ihre benötigte Energie für Tag D+1. Diese Meldung besteht aus einer auf jeden Fall abgenommenen, als auch einer preis-sensitiven Abnahmefunktion für jede Stunde des nächsten Tages. Außerdem melden die GenCos für jede Stunde des Folgetages ihre Angebotsfunktion. Diese ist eine **selbst bestimmte** lineare Grenzkostenfunktion über ein **selbst bestimmtes** Kapazitätsfenster.
Die GenCos sind durch diese Selbstbestimmung in der Lage, eine höhere Grenzkostenfunktion oder eine geringe Produktionskapazität an den ISO zu melden, um ihren Erlös zu erhöhen.
Das Besondere ist, dass GenCos in der Simulation mit einem Lernalgorithmus

ausgestattet sind, der sie in die Lage versetzt, die für ihr Ziel optimale Grenzkostenfunktion/Produktionskapazität zu erlernen. Dieses Lernen erfolgt immer am Ende des Tages D unter Berücksichtigung des aktuellen Preises.
Der ISO errechnet und publiziert den stündlichen Beitrag der GenCos zur Deckung des Energiebedarfes. Er berücksichtigt dabei die Kapazitätsbeschränkungen und Topologie des Übertragungsnetzes.
Mit dem Preis berechnen die LSEs die bei ihnen abgenommene Energiemenge und verteilen diese an ihre weiteren Kunden.

Softwaretechnischer Aufbau

AMES test bed wurde modular entworfen um für zukünftige Weiterentwicklungen gerüstet zu sein. Im Diagramm 8 ist die Anordnung der Module schematisch dargestellt.
Zu sehen sind unter anderem: das Learning Module (JReLM), der DC - Optimal Power Flow Solution Module (DC-OPFJ), das Übertragungsnetz, die verschiedenen Agenten und das Graphical User Interface (GUI).

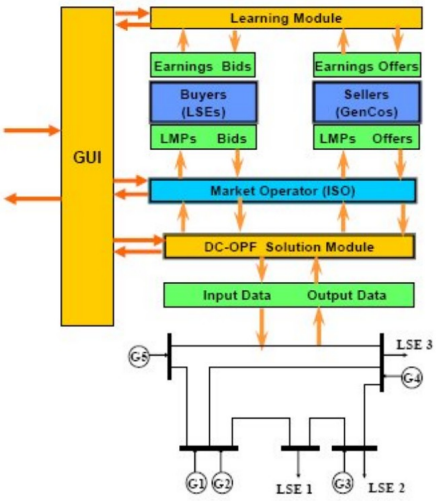


Abb. 8. Die Anordnung einiger softwaretechnischen Module des AMES test bed. [22]

Grenzen der Simulation

- Es werden weder Wetter noch zufällige Ereignisse wie Netz- oder Kraftwerksausfälle behandelt.
- Es treten weder neue Teilnehmer in den Markt ein noch verlassen welche den Markt.

5.2 Multiagentensysteme

Ein weiterer Ansatz zur Simulation des Energiemarktes bedient sich der Multi-Agent Systems (MAS). Unter diesen Systemen versteht man einen Zusammenschluss vieler Akteure (agents), von denen jeder seine eigenen Zielsetzungen und Spezialwissen besitzt. Das durch die Interaktionen entstehende Gesamtsystem lässt sich mithilfe dieses Ansatzes ohne zentrales Wissen der Systemdynamik simulieren.
Im Systementwurf werden Marktteilnehmer als Agenten mit individuellen Zielen repräsentiert. Die Agenten können sowohl mit anderen Agenten kommunizieren, als auch Änderungen ihrer Umwelt wahrnehmen und darauf reagieren. Sie schöpfen ihre Möglichkeiten zum Erreichen ihres Zieles vollständig aus.

PowerMatcher

Der PowerMatcher ist ein Multiagentensystem, das auf dem Prinzip des Marktgleichgewichts basiert. Der Auktionator bestimmt dabei einen Marktpreis, bei dem Angebot und Nachfrage im Gleichgewicht sind. Für Details siehe Kapitel 3.5 [Preisbildung am Spotmarkt] und 4.4 [Nash-Gleichgewicht].

Systementwurf und Funktionalität

Der PowerMatcher ist in einem Baum aus vier verschiedenen Arten von Agenten aufgebaut:

- **Local Device Agent:** Dieser Agent repräsentiert einen Energieerzeuger oder Verbraucher. Dieser meldet seine verbindliche Grenzkosten- oder Bedarfsfunktion an den Auktionator oder den nächsthöheren Concentrator. Er bestimmt weiter ökonomisch, anhand des vom Auktionator bestimmten Marktpreises und seiner Gebote, die Menge seiner angebotenen bzw. verbrauchten elektrischen Energie.
- **Auctioneer Agent:** Er sammelt alle Gebote von Angebot und Nachfrage und bestimmt den Marktpreis durch das Market-Clearing Verfahren. Den ermittelten Preis gibt er an alle anderen Agenten weiter.
- **Concentrate Agent:** Er fasst ein Teil des Netzes zusammen, indem er alle Gebote des Teilnetzes aggregiert und an den Auktionator weiterleitet. Umgekehrt gibt er die aktuellen Preise des Auktionators an alle Agenten seines Teilnetzes weiter. Aus der Sicht der unterliegenden Schicht verhält sich der Concetrator als Auktionator, aus der höheren Schicht als Device Agent.
- **Objective Agent:** Er kann einem Teilnetz ein anderes (von außen vorgegebenes) Ziel geben. Normalerweise ist das Ziel aller Agenten, den Markt auszubalancieren. Der Objective Agent erlaubt es zum Beispiel, dass "Komponenten eines virtuellen Kraftwerks entsprechend externer Produktionsvorgaben ausgeregelt werden."[19] Seite 18.

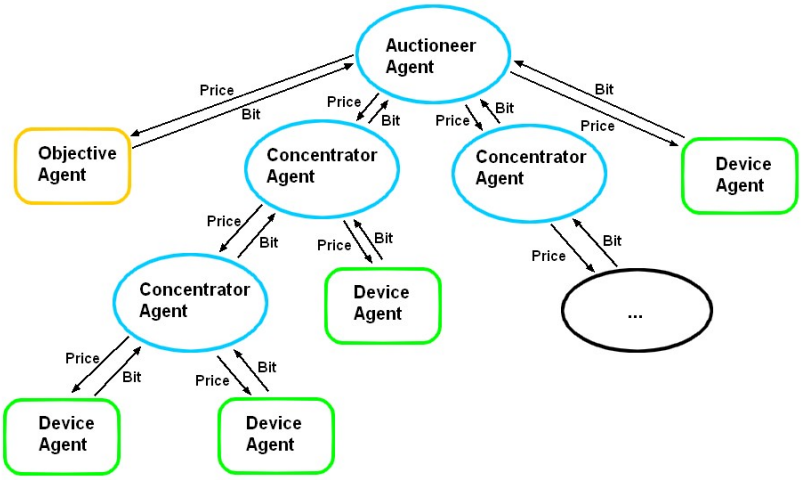


Abb. 9. Der Systementwurf des PowerMatchers. Die verschiedenen Agenten werden in einer Baumstruktur angeordnet.

Die Wurzel ist der Auctioneer Agent, die Blätter sind die Device Agents. Die Knoten des Baumes sind die Concentrate Agents, die Gebote zusammengefasst nach oben und Preisänderungen nach unten weiterleiten.
Die Bestimmung des Marktpreises erfolgt anhand des *CoTree*-Algorithmus. Die Berechnungskomplexität liegt durch diesen Algorithmus in $O(\log(n))$, wobei n die Anzahl der Device Agents ist.[20]

Funktionalität

Jeder Device Agent meldet seine spezifische Nachfragefunktion oder seine Grenzkosten an der Auktionator (oder Concetrator). Dieser fasst alle Gebote zusammen und bestimmt den Marktpreis. Siehe Kapitel 3.5 [Preisbidung am Spotmarkt]. Der berechnete Marktpreis wird an alle Device Agents kommuniziert. Diese bestimmen anhand des erhaltenen Preises ihre zu produzierende/verbrauchende Energiemenge.

Grenzen der Simulation

- Eigenschaften des Übertragungsnetzes werden nicht berücksichtigt.
- Local Device Agents haben keine Möglichkeit ihre produzierte/verbrauchte Energiemenge dem Preis anzupassen, da ihre Gebote im Voraus festgelegt sind und sie nicht lernfähig sind.

Erweiterungen des PowerMatchers

Es existieren mehrere Ansätze zur Erweiterung des PowerMatcher Konzepts um die die Simulation zu verbessern.

1. PowerAce (KIT)
   Beim PowerAce werden zusätzlich die Märkte für Regelleistung, CO_2 Zertifikatshandel und langfristige Termingeschäfte simuliert. Weiterführende Informationen sind unter [23] zu finden.
2. DEZENT (TU Dortmund)
   Das Simulationstool DEZENT erweitert die Concentrator Agents um zusätzlich Funktionalität. Es ist ihnen möglich, für ihr Subnetz einen Marktpreis zu bestimmen. Sie geben nur bisher unvermitteltes Angebot und unvermittelte Nachfrage an den jeweils höheren Agenten weiter. Sie können so als Markt eines (regionalen) Spannungsniveaus interpretiert werden. Weiterführende Informationen sind unter [19] zu finden.

5.3 Vergleich der vorgestellten Modellierungstools

Die folgende Tabelle listet einige Vor- und Nachteile der vorgestellten Modellierungstools auf und vergleicht diese. Verglichen wird dabei die Flexibilität der einzelnen Agenten. Damit ist gemeint, wie groß deren Handlungsraum ist und ob sie selbständig auf Änderungen ihrer Umgebung reagieren können. Dargestellt sind auch die Flexibilität des Systementwurfs, das heißt, ob zur Laufzeit die Anzahl an Agenten geändert werden kann, zudem die Berücksichtigung des Netzes, die Reaktionsfähigkeit und die Skalierbarkeit.

	AMES test bed	PowerMatcher
Flexibilität während der Simulation	- Netz nicht veränderbar - kein Hinzufügen/Entfernen von Agenten	+ Hinzufügen/Entfernen von Agenten ständig möglich + Objective Agents erlauben Einfluss von außen
Berücksichtigung der Netzeigenschaften	+ Netz wird simuliert + Leitungskapazitäten werden berücksichtigt	- Leitungskapazitäten werden nicht berücksichtigt - Vorhaltung von Regelleistung nicht berücksichtigt
Skalierbarkeit	- Flussproblem nur durch ISO lösbar	+ Rechenlast wird auf Concentrator Agents verteilt
Simulation	- Evaluation von Handelsstrategien an amerikanischen Märkten	+ simuliert den deutschen Energiemarkt

6 Fazit

6.1 Diskussion der bisherigen Ansätze

In den derzeitigen Modellen des day-ahead Elektrizitätsmarktes ist es für den Energieerzeuger profitabler die konventionellen Kraftwerke unter Volllast zu betreiben, da sie so die geringsten Grenzkosten aufweisen. Ein rein mit Volllast betriebenen Kraftwerken gespeistes Versorgungsnetz ist nur schwer regelbar. Kurzzeitige Nachfrageschwankungen oder ein Kraftwerksausfall sind nur langsam zu handhaben und verursachen zusätzliche Kosten. Dies wird vor dem Hintergrund der massiven Kraftwerkskapazität von Wind- und Photovoiltaikanlagen mit den variablen Einspeisemengen/Zeiträumen und geringen Grenzkosten in Deutschland noch deutlicher. Der Ausgleich von Angebot und Nachfrage ist von zentraler Bedeutung im Energiehandel und sollte von den Simulationstools berücksichtigt werden.
Das Simulationstool AMES wurde besonders für den amerikanischen Markt entwickelt. Da dort die Preisbildung nicht wie am deutschen Markt erfolgt, eignet es sich nur bedingt für eine akkurate Simulation in Deutschland.
Der PowerMatcher simuliert die Preisbildung so, wie sie an der EPEX abläuft, sollte jedoch mit weiterer Funktionalität ausgestattet werden um mit dem oben genannten Problem besser umgehen zu können. Ansätze zur Erweiterung sind im Kapitel 5.2 angedeutet.

6.2 Das Phänomen des negativen Strompreises

Um diese Arbeit abzuschließen und auf die Motivation zurückzukommen, wird im Folgenden das Entstehen eines negativen Strompreises erläutert.
Der negative Strompreis kommt durch ein besonders großes Angebot und eine zu niedrige Nachfrage zustande. Konventionelle Grundlastkraftwerke, wie zum Beispiel ein Atomkraftwerk, können nur sehr langsam geregelt werden, und es entstehen zusätzliche Lastwechselkosten für den Kraftwerksbetreiber. Erneuerbare Energien werden wann immer möglich Energie erzeugen, da nach dem EEG **jede** erzeugte Kilowattstunde **fix** vergütet wird. An einem windigen Sommertag akkumuliert sich daher die Energie aus Grundlastkraftwerken und Erneuerbaren Energien. Durch dieses große Angebot fällt der Strompreis immer weiter, da neue Verbraucher gefunden werden müssen um das Gleichgewicht zu erhalten. Sonst droht ein Netzversagen. Die Vergangenheit zeigt, dass der Preis dabei bis ins Negative fallen kann. Aus wirtschaftlicher Sicht wird der negative Preis akzeptiert, da die Verluste durch den Verkauf der Energie von den Energieerzeugern geringer eingeschätzt werden als das Herunter- und Wiederanfahren eines Kraftwerks mit den entsprechenden Lastwechselkosten.

Literatur

1. Gesetz zur Neuregelung des Energiewirtschaftsrechts §9, http://www.iwr.de/re/eu/recht/ewg.html (03.07.2014)
2. EnGw, Zuständigkeit Bundesnetzagentur, http://www.gesetze-im-internet.de/enwg_2005/__54.html (03.07.2014)
3. EEX Trading Brochure, http://www.eex.com/blob/68250/27b48c17c6925d18d84f5607d9a51d30/e-eex-unternehmen-februar-2014-pdf-data.pdf (12.06.2014)
4. EPEX SPOT Trading Brochure, https://www.eex.com/en/products/power/power-spot-market (03.07.2014)
5. Wikipedia EEX, http://de.wikipedia.org/w/index.php?title=European_Energy_Exchange&oldid=132669754 (14.09.2014)
6. EEX Features/Options, https://www.eex.com/en/market-data/power/derivatives-market (03.07.2014)
7. Wikipedia Terminmarkt, http://de.wikipedia.org/w/index.php?title=Terminmarkt&oldid=125813719 (03.07.2014)
8. Wikipedia Future, http://de.wikipedia.org/w/index.php?title=Future&oldid=126324938 (03.07.2014)
9. Wikipedia Option, http://de.wikipedia.org/w/index.php?title=Option_%28Wirtschaft%29&oldid=130857218 (03.07.2014)
10. Vorlesung Einführung in die Energiewirtschaft, Fichtner, KIT, SS2014
11. Bode Zeitgespräch, http://www.ecco-hamburg.com/uploads/media/Bode_2010_EE_Strommarkt_Zeitgespraech_WD_Server_01.pdf (12.06.2014)
12. Marktdaten der EPEX, http://www.epexspot.com/de/marktdaten/auktionshandel/chart/auction-chart/2013-05-19/DE (03.07.2014)
13. Vorlesung Renewable Energies and Economics, McKenna, WS 2013
14. EEG-Aktuell, http://www.eeg-aktuell.de/das-eeg/ (03.07.2014)
15. Stochastic modeling of the spot price of electricity incorporating commodities and renewables as exogenous factors, Dissertation Jan Müller, http://dokumentix.ub.uni-siegen.de/opus/volltexte/2014/796/pdf/mueller_jan.pdf (03.07.2014)
16. Wikipedia Nash Gleichgewicht, http://de.wikipedia.org/w/index.php?title=Nash-Gleichgewicht&oldid=129798107 (03.07.2014)
17. Mengenwettbewerb und Kapazitätsschranken bei Preiswettbewerb, http://www.compecon.vwl.uni-muenchen.de/dateien/wettbewerb_08_09/vl_4_mengenwettbew.pdf (03.07.2014)
18. Development of Open Source Software for Power Market Research: The AMES Test Bed, http://www2.econ.iastate.edu/tesfatsi/OSS_AMES.2009.pdf (03.07.2014)

19. Open-Source-Modellierung und auktionsorientierte Regulierung dezentraler Energienetze, Diplomarbeit Olaf Struß, http://www.informatik.uni-bremen.de/~ostruss/da/da-struss.pdf (03.07.2014)
20. Market-based Control in Decentralized Electrical Power Systems, http://users.ecs.soton.ac.uk/acr/ates2010/Paper_09.pdf (03.07.2014)
21. negative Strompreise, http://www.iass-potsdam.de/de/forschungscluster/globaler-gesellschaftsvertrag-fur-nachhaltigkeit-gcs/news/strom-umsonst-negative (04.09.2014)
22. AMES modularer Aufbau, http://www2.econ.iastate.edu/tesfatsi/images/AMESfigure.jpg (09.09.2014)
23. Weiterführende Informationen zu PowerAce, http://www.gsdp.eu/uploads/tx_conturttnews/Philipp_Ringler_-_PowerACE_-_Agent-Based_Simulation_of_Electricity_Markets.pdf (10.09.2014)

Secure Multiparty Computation - eine Lösung für Datenschutzaspekte im Crowdsourcing?

Clemens Wallrath*

Betreuer: Anja Bachmann†

Karlsruher Institut für Technologie (KIT)
Pervasive Computing Systems – TECO
*uagzs@student.kit.edu
†bachmann@teco.edu

Zusammenfassung. Ziel dieser Arbeit ist es, herauszufinden, ob sich die Datenschutzprobleme, die sich beim Crowdsourcen von Sensordaten mittels Smartphones und ähnlichen Geräten ergeben, mit den kryptographischen Protokollen der *Secure Multiparty Computation* (MPC) lösen lassen. Solche Protokolle erlauben es, verteilt Berechnungen durchzuführen, ohne dass die Teilnehmer ihre Eingabedaten irgendjemandem offenlegen müssen.
Es werden mehrere Modelle für solche Berechnungen vorgestellt und verglichen. Dann wird gezeigt, dass MPC in der Tat den Datenschutz im Crowdsourcing verbessern kann und welches der vorgestellten Protokolle auf dieses konkrete Problem anwendbar ist. Weiterhin wird ein Ansatz für die Implementation eines solchen Systems für das verteilte Data Mining gegeben.

Schlüsselwörter: Secure Multiparty Computation, verteiltes Data Mining, Smartphones, Datenschutz

1 Einführung

Moderne Smartphones bieten aufgrund ihrer verschiedenen integrierten Sensoren und ihrer weiten Verbreitung eine gute Möglichkeit, durch Crowdsourcing in großem Umfang Daten über die Umgebung zu sammeln. So könnte man z.B. großflächig die Güte des Mobilfunknetzes untersuchen.

Für ein solches Experiment fallen zwangsläufig datenschutzkritische Datensätze mit z.B. der Position der Teilnehmer an. Um den Datenschutz zu gewährleisten und die Akzeptanz solcher Datensammlung zu steigern, muss dringend darauf geachtet werden, dass die Daten anonym gespeichert und verarbeitet werden und es keine Möglichkeit für Missbrauch gibt. Wenn man die Daten durch einen Dritten anonymisieren lässt, muss man sich allerdings auf deren Vertrauenswürdigkeit verlassen, da sie die nicht anonymisierten Daten mitlesen könnte. Außerdem lassen sich solche vertrauenswürdige Dritte meistens gut bezahlen.

Eine mögliche Lösung dieses Problems könnte mit Hilfe von *Secure Multiparty Computation* (MPC, auch SMC) umgesetzt werden. MPC bietet die Möglichkeit,

Daten verteilt zu verarbeiten, ohne dass einzelne Teilnehmer mehr als ihre eigene Eingabe und die endgültige Ausgabe der Berechnung erfahren. Dadurch werden viele Datenschutzprobleme, wie die Einigung auf Richtlinien zur Speicherung persönlicher Daten, direkt vermieden, indem diese Daten gar nicht erst anfallen. Statt die Daten nur zu sammeln und selbst das Ergebnis zu berechnen, führt man also die ganze Berechnung verteilt durch und sammelt nur das Ergebnis ein. Man betreibt *verteiltes Data Mining*.

In diesem Paper werden verschiedene MPC-Protokolle vorgestellt, die für das datenschutzfreundliche Sammeln und Auswerten von Daten in beispielsweise aus Smartphones bestehenden, verteilten Sensornetzen interessant sein könnten. Es wird dann evaluiert, ob und welche dieser Protokolle das Datenschutzproblem lösen können.

Im Folgenden werden nach einer Erklärung der wichtigsten Begriffe in Kapitel 2 zunächst drei verschiedene Varianten der MPC vorgestellt, die sich insbesondere darin unterscheiden, von wem die eigentliche Berechnung durchgeführt wird. In Kapitel 3 wird zunächst der klassische Ansatz erläutert, in dem alle Teilnehmer die ganze Zeit an der Berechnung beteiligt sind. Anschließend wird in Kapitel 4 eine modifizierte Version vorgestellt, in der die Datensätze erst verschlüsselt gesammelt werden und die tatsächliche sichere Berechnung später von wenigen Servern umgesetzt wird. Als letzte Variante folgt in Kapitel 5 ein Ansatz, wie man sichere Berechnungen in der Cloud oder auf anderer, nicht vollständig vertrauenswürdiger Hardware durchführen kann.

Anhand ihrer Eignung für verteiltes Data Mining auf Smartphones werden sie dann in Kapitel 6 bewertet. Dabei wird besonders die Sicherheit gegen Angreifer und die Umsetzbarkeit beleuchtet, sowie grob eine mögliche konkrete Implementation geschildert. In Kapitel 7 folgt ein Fazit, in dem die Frage nach der Lösung des Datenschutzproblems beantwortet wird.

2 Begriffserklärung

Nachfolgend werden einige grundlegende Begriffe und Verfahren erklärt, die später in den vorgestellten Protokollen Anwendung finden.

2.1 Crowdsourcing

Der Begriff *Crowdsourcing* ist ein Neologismus, zusammengesetzt aus den englischen Worten *crowd* (=Menschenmenge) und *outsourcing* (=Auslagerung). Als erster benutzte ihn der Journalist Jeff Howe in einem Artikel auf wired.com [13]. Auf seiner Website gibt er folgende Definition [12]:

> Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call.

Crowdsourcing bedeutet also, eine Aufgabe, wie in unserem Beispiel das Sammeln von Daten, mit vielen freiwilligen Helfern zu erledigen, statt sie eigenen

Mitarbeitern oder einem externen Dienstleister zuzuweisen. Dadurch spart man unter Umständen Geld, oder ermöglicht großflächiges Erheben von Daten überhaupt erst.

2.2 Secure Multiparty Computation

Unter Secure Multiparty Computation versteht man nach Bogetoft et al. das Problem, dass n Parteien $P_1, ..., P_n$ mit den privaten Eingabewerten $x_1, ...x_n$ die Funktion $f(x_1, ..., x_n) = (y_1, ..., y_n)$ berechnen wollen, ohne dass ein Teilnehmer P_i mehr als seinen Eingabewert x_i und das Ergebnis y_i erfährt, auch wenn ein Teil der Teilnehmer bösartig ist [5]. Oft wählt man $y_1 = y_2 = ... = y_n$, d.h. es wird ein einzelnes gemeinsames Ergebnis berechnet, nicht verschiedene Ergebnisse für jeden Teilnehmer [7].

Allerdings ist alles, was Partei P_i allein aus x_i und y_i berechnen kann, als bekannt zu betrachten und darf im Laufe des Protokolls benutzt werden. So ist es wenig sinnvoll, mit MPC den Mittelwert oder die Summe aus nur zwei Eingaben zu berechnen, da beide Teilnehmer aus ihrer Eingabe und dem Ergebnis die Eingabe des jeweils anderen berechnen können. Ein solches Protokoll wäre nach Definition sicher, verrät aber dennoch eventuell mehr als erwünscht.

2.3 Angreifermodelle und Sicherheit

Im Folgenden werden einige Aspekte vorgestellt, die man bei der Konzeption eines MPC-Protokolls beachten muss und anhand derer man ein solches im Hinblick auf Sicherheit und Effizienz bewerten kann.

Die vorgestellten Begrifflichkeiten bieten eine Basis für die Vorstellung und Bewertung der einzelnen Protokolle in den späteren Kapiteln.

Referenz Für eine Bewertung braucht man zuerst eine Referenz. Dies ist eine rein theoretische, sicherste Möglichkeit, gemeinsam eine Funktion zu berechnen.

Dazu geht man von einer vollkommen vertrauenswürdigen Partei aus, die die Eingabewerte aller Teilnehmer einsammelt, die Funktion berechnet und die Ergebnisse bekannt macht. Man findet keine Variante, die sicherer ist, da sie nur von der Ehrlichkeit der Teilnehmerdaten (die man immer nur auf Plausibilität prüfen kann) und der Sicherheit des Kommunikationskanals (den man immer benötigt) abhängt. Auch benötigt diese Variante nur ein Minimum an Kommunikation, nämlich pro Teilnehmer eine Übertragung an den vertrauenswürdigen Server, und benötigt keine Berechnungen außerhalb der eigentlichen Funktion. Dadurch ist sie auch die effizienteste Lösung.

Anhand dieser Referenz kann man nun alle anderen Protokolle bewerten. Ziel ist es, sich dieser Lösung möglichst stark anzunähern.

Wie viele Parteien sind korrumpiert? Unter der Annahme, dass mehr als die Hälfte der teilnehmenden Parteien potenziell unter der Kontrolle eines

Angreifers stehen, gibt es keine Möglichkeit, ein bedingungslos sicheres Protokoll für die MPC zu konstruieren [8].

Benötigt man Sicherheit auch dann, wenn nur noch ein ehrlicher Teilnehmer im Spiel ist, muss man sich auf kryptographische Annahmen verlassen [8], d.h. davon ausgehen, dass der Angreifer nur begrenzte Rechenleistung hat. Ein solches verstärktes Protokoll ist auch deutlich komplexer und mit erheblichem Mehraufwand verbunden, kann dafür aber - unter der Bedingung, dass die Annahmen zutreffen - auch bei nur einem nicht korrumpierten Teilnehmer sicherstellen, dass der Angreifer dessen Eingabedaten nicht mitlesen kann und die Berechnung notfalls abgebrochen wird anstatt falsche Ergebnisse zuzulassen.

Angreifermodelle Es werden drei Arten von Angreifern unterschieden: *aktive*, *passive* und *versteckte*.

Ein *aktiver Angreifer* wird auf allen möglichen Wegen versuchen, weitere Informationen zu gewinnen, auch wenn er dafür vom Protokoll abweichen muss und sich so eventuell als Angreifer zu erkennen gibt.

Ein *passiver, aber neugieriger Angreifer* wird sich zwar an das Protokoll halten, aber versuchen, zusätzliche Geheimnisse zu erfahren.

Ein Protokoll, dass sicher gegen passive Angreifer ist, kann immer zu einem, dass sicher gegen aktive Angreifer ist, erweitert werden. Dabei geht allerdings Effizienz verloren, da jade Aktion eines Einzelnen von mehreren Teilnehmern überprüft werden muss [16].

Ein *versteckter (engl. „covert“) Angreifer* wird zwar nicht unbedingt strikt dem Protokoll folgen, versucht aber dabei besonders, nicht aufzufallen.

Je nach Sensibilität der Daten kann es ausreichend sein, nur von einem passiven oder versteckten Angreifer auszugehen. Dadurch lässt sich der Rechen- und Kommunikationsaufwand des Protokolls weiter verringern.

Angriffszeitpunkt Normalerweise betrachtet man *statische Angreifer*, d.h. Angreifer, die im Voraus einen bestimmten, während des Durchlaufs des Protokolls dann unveränderlichen Teil der Teilnehmer unter ihre Kontrolle gebracht haben. Je nachdem, wie kritisch die Anwendung ist, sollte man aber auch sogenannte *adaptive Angreifer* betrachten, die noch während des Durchlaufs entscheiden können, welche Teilnehmer sie unter ihre Kontrolle bringen [6].

Sicherer Kanal Die meisten MPC-Protokolle setzen einen vollkommen sicheren Kanal zwischen den Parteien voraus. In der Realität wird hier z.B. eine bestehende (oder neu aufgebaute) Public-Key-Infrastruktur verwendet. Eine solche Implementierung bietet bei unsauberer Umsetzung im schlimmsten Fall die Möglichkeit eines Man-in-the-Middle Angriffs und damit eine Möglichkeit das Protokoll zu stören. Bei der Implementierung eines MPC-Protokolls ist es wichtig, den sicheren Kanal ordentlich umzusetzen.

2.4 Secret Sharing

Die im späteren Verlauf der Arbeit vorgestellten Protokolle für die MPC machen meist Gebrauch von *Secret Sharing*. Dies soll im Folgenden kurz erklärt werden. Die Langform der Erklärung findet sich bei Frikken [10].

Die Aufgabe von *Secret Sharing* Methoden ist es, einen Datensatz s (das Geheimnis) so an mehrere Parteien zu verteilen, dass es nur durch Kombination einer Mindestanzahl t von insgesamt n Anteilen (sogenannten „Shares“) wieder rekonstruiert werden kann. Beim *Secure Secret Sharing* dürfen zusätzlich Besitzer von weniger als t Shares nicht mehr über das Geheimnis wissen, als jemand, der überhaupt keine Shares besitzt.

Für $t = n$ lässt sich das folgende einfache Verfahren nutzen: Man gibt jeder bis auf einer Partei eine Zufallszahl p_i mit der gleichen Anzahl an Bits, wie das Geheimnis. Die letzte Partei bekommt dann $p_n = s \oplus p_1 \oplus p_2 \oplus ... \oplus p_{n-1}$. Wobei $\oplus$ bitweises XOR darstellt. Das Geheimnis ergibt sich dann als $p_1 \oplus p_2 \oplus ... \oplus p_n$, wenn alle Parteien ihre Shares zusammenlegen.

Stellt man das Geheimnis als Summe von n Zufallszahlen modulo einer Zahl N dar, ist es sogar möglich, einige arithmetische Operationen wie z.B. Addition durchzuführen, ohne dass die Besitzer der einzelnen Shares kommunizieren müssen.

Für $t \leq n$ kann man z.B. *Shamir's Scheme* verwenden. Dazu wählt man ein Polynom p vom Grad $t - 1$ mit $p(x) = a_0 + a_1x + a_2x^2 + ... + a_{t-1}x^{t-1}$, so dass $a_0 = s$. Nun kann man n unterschiedliche Punkte auf p wählen und an die Teilnehmer verteilen. Jede Kombination von t verschiedenen Punkten legt ein Polymon vom Grad $t - 1$ eindeutig fest, so dass sich aus t Shares das Polynom p und dann das Geheimnis als $p(0)$ berechnen lassen.

Auch hier lassen sich allein auf den Shares einige Berechnungen durchführen: so lässt sich z.B. die Addition einer Konstanten durch Addition auf alle einzelnen Shares (=Punkte auf dem Polynom, d.h. insgesamt Verschiebung des Polynoms in y-Richtung) realisieren.

3 Variante I: Vollständig dezentral

Nachdem die grundlegenden Begriffe und Verfahren eingeführt sind, folgen nun drei Familien von MPC-Protokollen. Zuerst werden in diesem Kapitel Protokolle vorgestellt, die keine zentralen Server benötigen. Anschließend folgt im nächsten Kapitel eine Variante, die aus mehreren Servern einen vertrauenswürdigen Dritten konstruiert. Zuletzt werden Protokolle zum Rechnen auf nicht vertrauenswürdiger Hardware vorgestellt.

Variante I fasst viele Protokolle zusammen, die darauf basieren, dass alle Teilnehmer die ganze Zeit an der Berechnung beteiligt sind, dafür allerdings ohne zentrale Infrastruktur auskommen. Diese dezentrale Struktur wird in Abbildung 1 verdeutlicht.

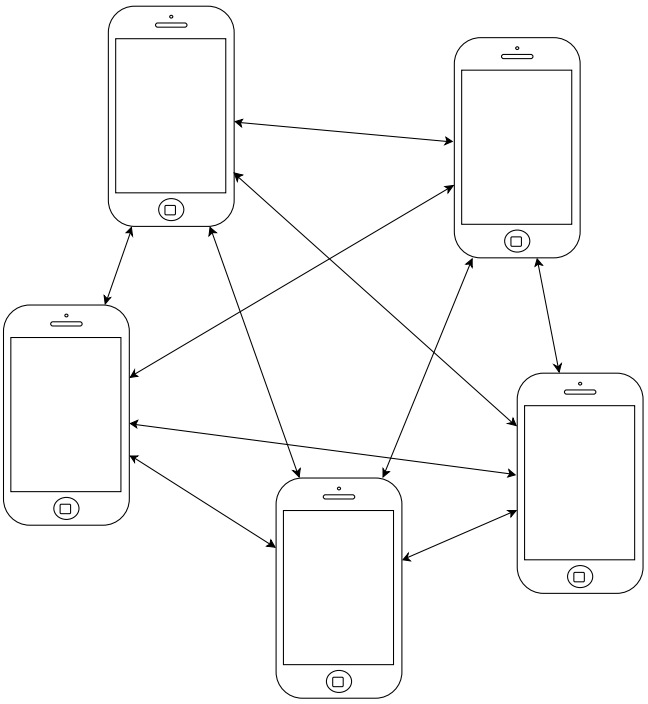


Abb. 1. Visualisierung der Struktur von Variante I. Es wird kein zentraler Server benötigt, da die Teilnehmer die ganze Zeit direkt miteinander kommunizieren.

3.1 Grundlegende Funktionsweise

Garbled Circuits Den meisten Protokollen für MPC liegen *Yao's garbled circuits* zugrunde. Ursprünglich für MPC mit nur zwei Parteien entwickelt, ist hier die Idee, die zu berechnende Funktion in eine logische Schaltung zu überführen und alle Ein- und Ausgaben zu verschlüsseln (engl. *garbled* = *durcheinandergewürfelt*), so dass eine andere Partei sie berechnen kann, ohne alle Eingaben kennen zu müssen [14].

Die beiden Teilnehmer seien P_1 und P_2. P_1 generiert aus der Funktion eine Schaltung und legt zufällige Zahlen als Belegung für logisch 1 und 0 für jeden Draht der Schaltung fest. Alle Gatter werden ebenfalls so angepasst, dass sie die Zufallszahlen an den Eingängen auf die entsprechenden Zufallszahlen an den Ausgängen überführen.

Die so verschlüsselte Schaltung schickt P_1 nun zusammen mit seinen ebenfalls verschlüsselten Eingabewerten an P_2, welcher einen *1-2 Oblivious Transfer* nutzt, um von P_1 die Verschlüsselung für seine Eingaben zu bekommen. Ein *Oblivious Transfer* ist eine Technik, mit der in diesem Fall P_2 einen von zwei Werten von P_1 bekommen kann, ohne dass P_1 erfährt, welcher gewählt wurde. So bekommt

P_2 die jeweilige Kodierung für seine Eingabedrähte, ohne die für den negierten Wert zu erfahren und ohne dass P_1 die unverschlüsselten Eingaben sieht.

P_2 kann jetzt die Schaltung mit den Eingabewerten beider Teilnehmer auswerten und dann zusammen mit P_1 die Ausgabe entschlüsseln.

Bei zwei Teilnehmern kann allerdings nie sichergestellt werden, dass nicht einer der beiden das Protokoll abbricht, sobald er das Ergebnis erfahren hat, und damit dafür sorgt, dass es der andere nicht auch erfährt. Außerdem muss sich im bisher vorgestellten Protokoll P_2 darauf verlassen, dass die Schaltung auch wirklich die gewünschte Funktion berechnet.

Um dieses Problem zu lösen, erweitert man das Protokoll so, dass P_1 mehrere Schaltungen für die Funktion erstellt und P_2 sich die Verschlüsselung der Drähte für beliebig viele dieser Schaltungen offenlegen lassen kann. P_2 kann dann diese Schaltungen auf ihre Richtigkeit prüfen. Die offengelegten Schaltungen werden verworfen und nicht weiter verwendet.

Wie man auf Basis dieses Protokolls nun eines für mehr als zwei Teilnehmer konstruiert, wird im nächsten Abschnitt erläutert.

GMW Das erste generische Protokoll für MPC mit beliebig vielen Teilnehmern stammt von Goldreich, Micali und Widgerson [11]. Die grundlegende Struktur dieses und aller darauf aufbauenden Protokolle ist im Folgenden erklärt.

In einer ersten Phase verteilen alle Parteien ihre Eingabewerte über Secret Sharing an alle anderen Teilnehmer. Je kleiner man t (die benötigte Anzahl Shares zum Rekonstruieren des Geheimnisses) wählt, desto robuster ist das Protokoll gegen ausfallende Teilnehmer, sobald alle Eingabewerte verteilt sind. Ein kleineres t bedeutet aber auch, dass weniger Teilnehmer korrumpiert sein müssen, um die privaten Eingaben mitzulesen. Die zu berechnende Funktion wird wieder als Schaltung dargestellt. Während der nächsten Phase werden in Runden für jedes Gatter im Schaltkreis die Ausgaben so berechnet, dass jeder Teilnehmer nur einen Share des (Zwischen-) Ergebnisses bekommt. In der letzten Phase wird dann aus den Shares das endgültige Ergebnis zusammengesetzt.

Dieses vergleichsweise einfache Protokoll unterliegt allerdings einigen Einschränkungen. Ein Problem ist, dass, auch wenn man nur eine konstante Anzahl Runden pro Gatter braucht, man immernoch insgesamt mindestens linear in der Tiefe der Schaltung ist. Die Schaltungen werden schnell sehr groß, alleine dadurch, dass oft mit 64 Bit langen Zahlen gerechnet wird. Dadurch sind für ernsthaften Einsatz anwendungsspezifische oder zumindest schnellere generische Protokolle - wie das im nächsten Abschnitt erläuterte SPDZ - nötig [9].

3.2 SPDZ

Effizientere Protokolle Als ein Beispiel eines effizienteren generischen Protokolls seien hier die Ideen hinter *SPDZ* genannt. Neben den generischen gibt es auch viele spezialisierte Protokolle für z.B. Skalarprodukt, Mengen-Operationen und Suche in verteilten Datenbanken. Einige davon werden von Laud et al. genauer vorgestellt [14].

SPDZ (vorgestellt in [7] und weiter verbessert in [8]) ist ein generisches Protokoll für MPC mit beliebig vielen Teilnehmern, das auch mit potenziell korrumpierter Mehrheit noch sicher funktioniert. Es unterteilt sich in eine Offline- und eine Online-Phase, wie im Folgenden vorgestellt.

Offline-Phase Unter der Annahme eines sicheren *Dealers* entwickelt man zuerst ein generisches MPC-Protokoll im *Preprocessing Model*. Dieses Modell lässt Vorberechnungen wie das Generieren von Zufallswerten in einer *Offline-Phase* zu, in der die Eingabewerte und eventuell sogar die zu berechnende Funktion noch nicht bekannt sind. Der Dealer wird in dieser Phase durch ein Public-Key basiertes Protokoll implementiert und verteilt die vorberechneten Daten.

Wichtigstes Produkt der Offline-Phase sind sogenannte Multiplikations-Tripel. Sie bestehen aus Zufallswerten a, b und c, so dass $a \cdot b = c$. Diese Zufallszahlen werden via Secret Sharing an die Teilnehmer verteilt und ermöglichen später die Multiplikation von verteilten Werten.

Online-Phase Durch das Verschieben der teuren Public-Key-Kommunikation in den Vorberechnungsschritt kann das folgende *Online-Protokoll* deutlich effizienter konstruiert werden. Die Online-Phase ist zudem garantiert sicher gegen einen aktiven Angreifer mit mehr als der Hälfte korrumpierten Teilnehmern. Zusammen mit einer Offline-Phase mit aktiver Sicherheit lässt sich so ein Protokoll mit vollständig aktiver Sicherheit konstruieren.

Die funktionsunabhängigen Vorberechnungen aus der Offline-Phase können zudem für mehr als eine Funktion benutzt werden, so dass in der Online-Phase z.B. abhängig von einzelnen Ergebnissen weitere Berechnungen durchgeführt werden können.

Im nächsten Kapitel wird nun eine Möglichkeit vorgestellt, wie man MPC durchführen kann, ohne dass ständig alle Teilnehmer online sein müssen.

4 Variante II: 2 bis 3 zentrale Server

4.1 Grundlegende Funktionsweise

Das Problem Bei gemeinsamen Berechnungen in der Größenordnung von hunderten oder sogar einigen tausend Teilnehmern trifft man sehr schnell auf mehrere Probleme: Zum einen steigt der Kommunikationsaufwand und der Aufwand des Public-Key Systems, insbesondere muss man aber auch einen Zeitpunkt finden, zu dem alle Teilnehmer gleichzeitig online sind, um die Berechnung durchzuführen. Die bisher vorgestellten Protokolle verkraften zwar das Ausscheiden einzelner Teilnehmer, allerdings erst nachdem in der Sharing-Phase die Eingabedaten verteilt wurden.

Im Folgenden wird ein Protokoll vorgestellt, dass dieses Problem lösen kann, dafür aber wieder einen gewissen Teil Vertrauen seitens der Teilnehmer benötigt.

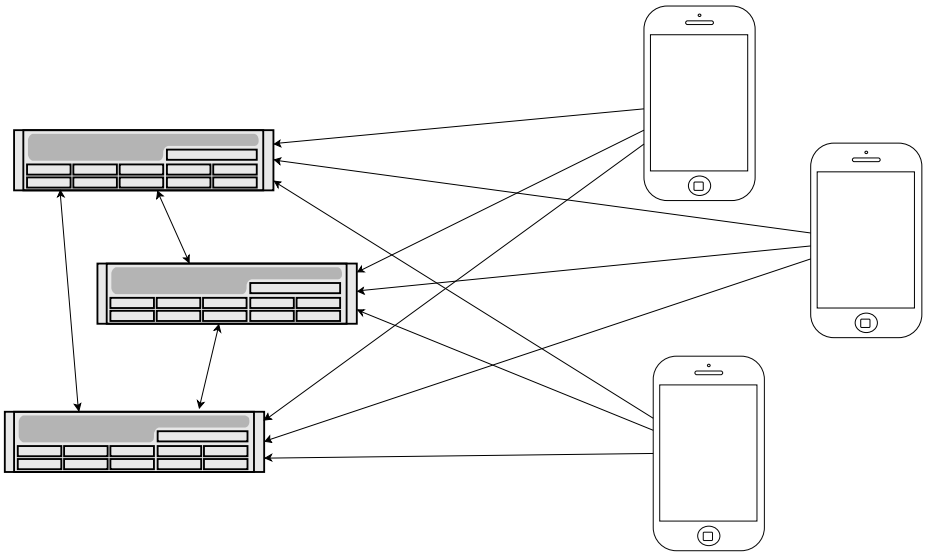


Abb. 2. Bei Variante II verteilen die Teilnehmer ihre Daten via Secret Sharing auf die Server. Diese berechnen dann zusammen das Ergebnis.

Das Protokoll Eine Lösung für die oben genannten Probleme ist wie bei Damgård et al. erläutert [5]: Anstatt alle Teilnehmer in die Berechnung mit einzubeziehen, nutzt man ein MPC-Protokoll, um eine Art vertrauenswürdige Drittpartei zu konstruieren. Diese besteht aus zwei bis drei voneinander unabhängigen Parteien. Es sind auch mehr Parteien denkbar, für MPC mit zwei oder drei Teilnehmern existieren allerdings effizientere Protokolle.

Alle Teilnehmer übermitteln ihre Eingabedaten, indem sie diese jeweils mit Secret Sharing auf die Parteien der zentralen Berechnungseinheit verteilen. Diese ist allergings nicht wirklich „zentral“, zumindest nicht räumlich und organisatorisch. Niemand darf Zugriff auf mehr als einen dieser Server besitzen. Unter der Annahme, dass diese Parteien wirklich kein Interesse haben, sich zu verbünden, um an die Daten zu kommen, sind diese also für niemanden lesbar. Gilt diese Annahme nicht, kann das Protokoll keinen Schutz der Daten garantieren. Die Auswahl der zentralen Parteien ist also kritisch.

Die Eingabedaten können über einen längeren Zeitraum gesammelt werden, da immer nur das sendende Endgerät und die zentralen Server online sein müssen. Sobald alle (bzw. genug) Teilnehmer ihre Daten übermittelt haben, kann die Berechnung inklusive der Entschlüsselung des Ergebnisses mit einem MPC-Protokoll, ähnlich wie in Variante I, durchgeführt werden. Dies geschieht allein zwischen den zentralen Parteien und benötigt keinen Eingriff der anderen Teilnehmer mehr. Abbildung 2 zeigt Aufbau und Datenfluss in dieser Variante.

Bewertung Dieses Protokoll löst auch noch ein weiteres Problem: Dadurch, dass die Daten „zentral“ entgegengenommen werden, müssen die Teilnehmer nicht mehr alle vorher bekannt sein. Die Public Key Infrastruktur ist deutlich einfacher aufgebaut, da nur öffentliche Schlüssel für die zentralen Server zur Verfügung gestellt werden müssen.

Kritisch ist dafür die Wahl der zentralen Parteien, da das Protokoll - wie weiter oben erläutert - nur sicher ist, wenn diese nicht zusammenarbeiten. Auch ist die Akzeptanz des Verfahrens seitens der Teilnehmer natürlich von deren Vertrauen in die Unabhängigkeit abhängig [5].

Ein ähnliches Protokoll wurde schon in der Praxis erfolgreich eingesetzt. Der dortige Einsatz von MPC wird im Folgenden wiedergegeben.

4.2 Anwendung - Danish Sugar Beet Auction

Der in [5] vorgestellte Einsatz dieser Technik ist besonders interessant, da es der erste große Einsatz von MPC in der Praxis ist. Trotz des großen theoretischen Interesses an Multiparty Computation sind bisher nur wenige wirklich praktische Einsätze dokumentiert, in denen mit einer großen Anzahl Teilnehmer gearbeitet wird.

Das Szenario Ausgangssituation war eine anstehende Auktion um die Produktionsrechte[1] für Zuckerrüben in Dänemark. Die teilnehmenden Bauern geben bei einer solchen Auktion an, zu welchem Preis sie wie viele dieser Rechte kaufen oder verkaufen würden. Solche Angaben verraten den finanziellen Zustand der Bauern, was besonders kritisch ist, da Danisco, einziger Käufer der Zuckerrüben und Auftraggeber der Auktion, auch Kredite an Bauern vergibt.

Ziel war es also, den *Market Clearing Price*, also den Preis, zu dem An- und Verkäufe aufgehen, zu finden, ohne dass die Gebote zu anderen Preisen offengelegt werden müssen [5].

Umsetzung Man entschied sich für die Umsetzung des oben geschilderten Protokolls mit drei zentralen Parteien: Danisco, die Vereinigung der Zuckerrübenbauern DKS, und die beteiligte Forschergruppe. Die Teilnehmer konnten ihre Gebote über ein Java-Applet eintragen, das dann daraus ein verschlüsseltes Secret Sharing berechnete und auf einen Webserver, der nur für das Sammeln der Daten zuständig war, hochlud. Nach Ablauf der Teilnahmefrist wurden diese dann auf die drei Server verteilt.

Gebote wurden als eine Liste von Anzahlen zu den ungefähr 4000 möglichen Preisen dargestellt. Für Verkaufsgebote sollte diese Liste absteigend, für Einkaufsgebote aufsteigend sein[2]. Das Protokoll war nur auf passive, statische Angreifer ausgelegt, da dies als ausreichend empfunden wurde, und basierte hauptsächlich auf einem spezialisierten Protokoll für binäre Suche.

[1] Handelbare Verträge über die Abnahme der Erzeugnisse
[2] Im Protokoll wurde dies später sichergestellt

Für die endgültige Berechnung des Preises aus ungefähr 1200 Geboten rechneten die drei Server[3] ungefähr eine halbe Stunde [5].

Fazit Die generelle Akzeptanz dieser Umsetzung war sehr gut. In einer Umfrage gaben mehr als 80% der Bauern an, dass ihnen der so garantierte Datenschutz sehr wichtig ist. Auch war diese Lösung deutlich günstiger, also wenn man ein Auktionshaus als vertrauenswürdigen Dritten bezahlt hätte [5]. Für weitere solcher Auktionen müsste man nur kleine Änderungen am bestehenden System vornehmen.

Kapitel 5 erläutert nachfolgend noch einen anderen Ansatz, die Teilnehmer zu entlasten und die Berechnung auszulagern. Dazu wird von Cloud-Servern auf verschlüsselten Daten gerechnet.

5 Variante III: MPC in der Cloud

Manchmal kann es interessant sein, eine Berechnung mit Eingabedaten von einer oder mehreren Parteien auf nicht vollständig vertrauenswürdiger Hardware durchzuführen. Ein solcher Fall wäre z.B. eine Berechnung in der Cloud. Auch für diese Anwendung gibt es ein MPC-Protokoll. Es basiert auf *homomorphen Verschlüsselungsschemata*, die im Folgenden genauer erklärt werden.

5.1 Homomorphe Verschlüsselungsschemata

Homomorphe Verschlüsselung (*Fully Homomorphic Encryption, FHE*) erlaubt, einige Berechnungen direkt auf verschlüsselten Daten durchzuführen. Grundvoraussetzungen sind nach [10]:

1. basiert auf einem *Public-Key System*
2. *semantisch sicher*, d.h. die Wahrscheinlichkeit, einen Text seiner Verschlüsselung zuzuordnen, darf nur vernachlässigbar größer als die Ratewahrscheinlichkeit sein.
3. $E(x + y) = E(x) * E(y)$ mit $E(x)$, $E(y)$ Verschlüsselung von x bzw. y (Addition)
4. $E(x \cdot c) = E(x)^c$ für beliebige Konstante c (Multiplikation)
5. $E(x) \cdot E(0)$ ergibt eine weitere Verschlüsselung für x (Wiederverschlüsselung)

Durch passende Kodierung der Eingabewerte lassen sich daraus unter anderem auch Polynom- und Mengenoperationen ableiten.

FHE-basierte Protokolle haben zwar einen sehr geringen Kommunikations- und damit Bandbreitenbedarf, sind dafür aber sonst so langsam, dass sie für die bisher vorgestellten Protokolle nicht konkurrenzfähig wären. Allerdings verwendet z.B. SPDZ im Vorberechnungsschritt FHE[8]. Auch andere spezialisierte Protokolle, wie sie z.B. in [10] gezeigt werden, nutzen FHE. Für das im folgenden Absatz geschilderte Protokoll gibt es bisher keine andere Möglichkeit, daher ist Geschwindigkeit dort kein Ausschlusskriterium.

[3] Drei Laptops mit je 2.2GHz Intel Centrino Dual Core, 4GB RAM

5.2 Umsetzung

Der grundlegende Ablauf eines solchen Protokolls ist in Abbildung 3 dargestellt. Es funktioniert wie folgt: Anfangs verschlüsseln alle Teilnehmer ihre Eingabedaten und laden sie auf den nicht vertrauenswürdigen Server bzw. in die Cloud. Dort können nun eine oder mehrere Funktionen gewählt werden, die auf einer beliebigen Teilmenge der verschlüsselten Daten berechnet werden sollen. Zum Auswerten der Funktion werden die Teilnehmer nicht benötigt [15].

Das Ergebnis ist wiederum verschlüsselt. Erst, wenn es zur Verfügung steht, müssen die Teilnehmer wieder aktiv werden und es zusammen entschlüsseln. Es ist zu beachten, dass die Verschlüsselung zwar die Daten vor eventuell neugierigen Cloudanbietern schützt, alleine aber keinesfalls die Korrektheit der Berechnung garantiert [15].

5.3 Multikey Homomorphic Encryption

Ein Problem von FHE, wie sie hier vorgestellt wurde, ist, dass sie nur auf Daten rechnen kann, die mit dem gleichen Schlüssel verschlüsselt wurden. Das bedeutet, dass alle Teilnehmer zuerst mit einem klassischen MPC-Protokoll alle ihre Daten mit einem gemeinsamen Schlüssel verschlüsseln müssen, bevor sie hochgeladen werden. Dazu müssen alle Teilnehmer einmal gleichzeitig online sein. Das nachträgliche Hinzufügen von Datensätzen ist nicht bzw. nur sehr umständlich umsetzbar [15].

López-Alt et al. heben diese Beschreibung auf, indem ein homomorphes Verschlüsselungsschema konstruiert wird, das auf mit verschiedenen Schlüsseln verschlüsselten Daten rechnen kann. Das erlaubt es nun, dass jeder Teilnehmer seine Eingabedaten einfach mit seinem eigenen Schlüssel verschlüsselt und hochlädt, ohne mit den anderen Teilnehmern interagieren zu müssen. Erst wenn sein Datensatz in einer Berechnung benutzt wurde, muss er zusammen mit allen anderen Teilnehmern dieser speziellen Berechnung das Ergebnis entschlüsseln. Es ist jetzt einfach, neue Teilnehmer hinzuzufügen, da sie ihre Daten in den Pool hinzufügen können, ohne die anderen Teilnehmer überhaupt zu kennen [15].

6 Diskussion

Im Folgenden wird nun diskutiert, welches der vorgestellten Protokolle für den in dieser Arbeit gegebenen Anwendungsfall, also Crowdsourcing mithilfe von Smartphones, am einfachsten umsetzbar ist und welcher Anspruch an die Sicherheit bei der konkreten Implementierung gestellt werden muss.

6.1 Umsetzbarkeit

Ein großes Problem beim Crowdsourcing mit Smartphones ist, dass man nicht davon ausgehen kann, dass alle Teilnehmer gleichzeitig online sind. Auch kann es passieren, dass Teilnehmer ihre Teilnahme beenden, bevor alle Berechnungen zu Ende sind.

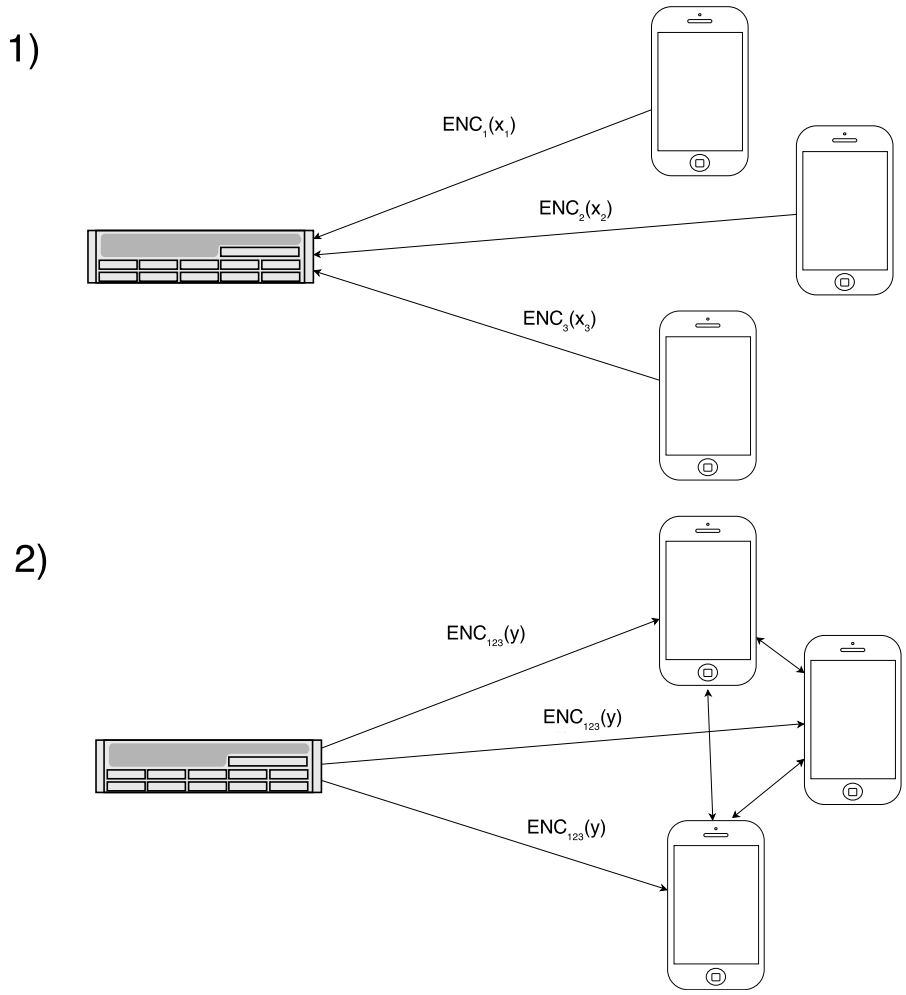


Abb. 3. Visualisierung der Variante III. 1) Die Teilnehmer schicken ihre Daten an sich selbst verschlüsselt an den Server. Der Server berechnet direkt auf den verschlüsselten Daten das Ergebnis, das wieder verschlüsselt ist. 2) Die Teilnehmer können dann zusammen das Ergebnis entschlüsseln.

Diese Einschränkungen machen die Variante I nicht praktikabel. Hier müssen alle Teilnehmer bis zum Ende der Berechnung zur Verfügung stehen. Variante III benötigt nach Abschluss jeder Berechnung auch alle Teilnehmer, um das Ergebnis zu entschlüsseln, daher ist sie mit unzuverlässigen Teilnehmern auch nicht durchführbar.

Um unabhängig von einzelnen Teilnehmern zu sein, muss man die Daten auf dauerhaft laufenden Servern sammeln und speichern. Variante II ermöglicht es, die Daten so auf ein paar wenigen Servern zu speichern, dass keiner allein die einzelnen Datensätze lesen kann. Der Sammelvorgang kann über einen längeren Zeitraum stattfinden und benötigt immer nur die Kommunikation zwischen mobilem Gerät und den Servern. Die Berechnung kann nach Abschluss der Sammelphase allein von den zentralen Servern durchgeführt werden. Ein weiterer Vorteil ist damit auch, dass auf den Mobilgeräten nur wenig Rechenzeit und damit Akkuladung gebraucht wird.

Generell hat Variante II bei der Nutzung spezifischer Protokolle auch den geringsten Mehraufwand an Rechenzeit der drei Verfahren und benötigt vergleichsweise wenig Kommunikation.

Bisher wurde hier allerdings die Sicherheit gegen Angreifer in der Diskussion vernachlässigt. Es folgt nun eine Betrachtung des Sicherheitsanspruchs mit Fokus auf Variante II.

6.2 Sicherheitsanspruch

Die in Kapitel 2.2 vorgestellten Sicherheitskriterien sollen hier auf die konkrete Fragestellung bezogen werden.

Prinzipiell wäre es möglich ein Protokoll zu verwenden, das Sicherheit gegen einen aktiven, adaptiven Angreifer mit Kontrolle über die Mehrheit der Teilnehmer bietet. Ein solches Protokoll hätte allerdings einen deutlich höheren Aufwand an Kommunikation und Rechenzeit, was, besonders im Hinblick auf die eher schwächeren mobilen Endgeräte und deren begrenzte Akkulaufzeit, nicht erwünscht ist. Da durch die Einschränkung des Sicherheitsanspruchs das Protokoll deutlich vereinfacht werden kann, lohnt es sich zu prüfen, wie viel Sicherheit man wirklich fordern muss.

Zuerst: Wie interessant wären die verarbeiteten Daten für einen Angreifer? Wie wahrscheinlich ist es also, dass dieses Protokoll aktiv angegriffen wird? Die interessantesten Informationen sind wahrscheinlich die Positionsdaten. Kreditkartendaten oder ähnlich Wertvolles sind ohnehin nicht Bestandteil des Protokolls. Um allerdings Bewegungsprofile einer bestimmten Person zu bekommen, muss ein Angreifer nur einzelne Smartphones kompromittieren.

Aufgrund der Festlegung auf Variante II, wie in Kapitel 6.1 motiviert, kann man diese jetzt auch als Basis für die Sicherheitsbetrachtung nehmen: Die Teilnehmer liefern nur ihre Daten ab und sind dann nicht mehr Teil des Protokolls zur eigentlichen Berechnung des Ergebnisses. Stellt man sicher, dass die eingereichten Daten alle wohlgeformt sind, können beliebig viele Teilnehmer befallen sein, ohne einem Angreifer einen Vorteil zu bieten. Um immer wohlgeformte Eingabedaten zu bekommen, kann man die von Bogetoft et al. vorgestellte Methode

nutzen, die hier aber nicht weiter erläutert wird, da sie nicht direkt in den Rahmen dieser Arbeit fällt [5]. Ein Angreifer kann natürlich trotzdem alle Daten der Teilnehmer lesen, die unter seiner Kontrolle sind.

Für die zentralen Server reicht es vorauszusetzen, dass die Mehrheit ehrlich ist. Bei der Auswahl der beteiligten Parteien muss deren Vertrauenswürdigkeit sowieso geprüft werden. Man kann diese kritischen Server zusätzlich vom Internet isolieren, indem man die an die einzelnen Server verschlüsselten Shares zuerst mithilfe eines zusätzlichen Webservers sammelt [5].

Insgesamt genügt es also, einen passiven, statischen Angreifer anzunehmen, der beliebig viele der Teilnehmer, aber nur eine Minderheit der zentralen Server, unter seiner Kontrolle haben darf.

6.3 Die konkrete Implementierung

Bei der Umsetzung kann man sich stark am Beispiel aus Kapitel 4.2 bzw. [5] orientieren, d.h. drei zentrale Server und ein Webserver, der die verschlüsselten Shares einsammelt. An Stelle eines Java-Applets würde man hier auf Seite der Teilnehmer Apps für die entsprechenden Zielbetriebssysteme anbieten.

Die Auswahl der berechnenden Parteien ist abhängig vom konkreten Fall. Denkbar wären aber z.B. neben der Forschungseinrichtung noch ein Unternehmen mit Interesse am Ergebnis der Untersuchung und eine unabhängige Organisation, die im Interesse der Teilnehmer arbeitet und die das Ergebnis eventuell auch interessiert.

Ein wesentlicher Unterschied zum gegebenen Beispiel besteht allerdings darin, dass beim verteilten Sammeln von Daten nicht unbedingt bei jedem Teilnehmer gleich viele Daten anfallen. Da das bestehende Protokoll aber eine feste Länge der Datensätze voraussetzt, empfiehlt es sich hier mehrere Einreichungen pro Teilnehmer zu erlauben[4] oder das Protokoll anders anzupassen.

Die Datensätze können dann durchgängig gesammelt werden und entweder zu einem bestimmten Datum oder aber auch regelmäßig, also z.B. täglich oder monatlich, ausgewertet werden.

7 Fazit

Von den vorgestellten Verfahren ist, wie im letzten Kapitel erörtert, nur Variante II für Crowdsourcing mit mobilen Geräten wirklich praktikabel, da so jedes Gerät nur einmal kurz online sein muss, um seine Eingabedaten auf die zentralen Server zu verteilen.

Wichtig ist hier allerdings die Wahl der Standpunkte für die zentralen Server. Stehen sie z.B. alle in der gleichen Hochschule oder sogar dem gleichen Institut, gibt es keinen nennenswerten Gewinn an Datenschutz, da so eventuell sogar eine Person alleine Zugriff auf alle Daten hat. Die Betreiber dürfen kein Interesse daran haben, ihren Teil der Daten mit den anderen zu teilen, um die ursprünglichen Datensätze zu rekonstruieren und müssen sich ehrlich verhalten und die

[4] aus Effizienzgründen mit jeweils einigen Datensätzen auf einmal

Berechnung zu Ende führen wollen. In [5] wurden für das Durchführen einer Auktion z.B. das Auktionshaus, der Interessenverband der Teilnehmer und das Forscherteam als Betreiber gewählt.

Unter der Annahme, dass die Teilnehmer sinnvolle Werte liefern (diese Annahme muss man im „perfekten“ Modell mit einer zentralen vertrauenswürdigen Partei auch treffen) kann so das korrekte Ergebnis berechnet werden, ohne dass ein Teilnehmer oder die Personen hinter der Untersuchung die Werte anderer Teilnehmer erfahren kann. Das ist aus Sicht des Datenschutzes optimal, da nirgendwo Daten anfallen, die einer Person zuzuordnen sind. Man nennt dies auch Datensparsamkeit.

Schlussendlich kann man also sagen, dass Multiparty Computation, besonders, wie hier in Variante II vorgestellt, eine datenschutzfreundliche, kostengünstige Möglichkeit bietet, Crowdsourcing mithilfe von Smartphones oder ähnlichen mobilen Geräten zu betreiben.

Als nächsten Schritt gilt es jetzt, bestehende Frameworks zu testen und herauszufinden, welche das hier gewünschte Modell unterstützen. Besonders interessant wirkt hier auf den ersten Blick *Sharemind* [2], das ein System wie in Variante II mit drei zentralen Servern implementiert. Eine kurze Übersicht über die Implementierungen findet sich in Tabelle 1.

	Sharemind	Tasty	VIFF	Fairplay
Typ	Variante II	Two-Party	Multiparty (Variante I)	Multiparty (Variante I)
Teilnehmer	3 Server, 1+ Clients	2	3+	2+
Lizenz	frei für Forschungszwecke, Sourcecode auf Anfrage	GPL v3	LGPL v3	offen, unspezifiziert

Tabelle 1. Übersicht über bestehende Implementierungen [1,2,3,4]

Literatur

1. The Fairplay project. `http://www.cs.huji.ac.il/project/Fairplay/`, zuletzt abgerufen 06.07.2014
2. Sharemind. `http://research.cyber.ee/sharemind/`, zuletzt abgerufen 06.07.2014
3. The TASTY project. `https://code.google.com/p/tastyproject/`, zuletzt abgerufen 06.07.2014
4. VIFF, the Virtual Ideal Functionality Framework. `http://viff.dk/`, zuletzt abgerufen 06.07.2014
5. Bogetoft, P., Christensen, D., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J., Nielsen, J., Nielsen, K., Pagter, J., Schwartzbach, M., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol.

5628, pp. 325–343. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-03549-4_20
6. Cramer, R., Damgård, I.: Multiparty computation, an introduction. In: Contemporary Cryptology, pp. 41–87. Advanced Courses in Mathematics - CRM Barcelona, Birkhäuser Basel (2005), http://dx.doi.org/10.1007/3-7643-7394-6_2
7. Damgard, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535 (2011), http://eprint.iacr.org/
8. Damgard, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. Cryptology ePrint Archive, Report 2012/642 (2012), http://eprint.iacr.org/
9. Damgård, I., Nielsen, J.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) Advances in Cryptology - CRYPTO 2007, Lecture Notes in Computer Science, vol. 4622, pp. 572–590. Springer Berlin Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74143-5_32
10. Frikken, K.B.: Algorithms and theory of computation handbook. chap. Secure Multiparty Computation, pp. 14–14. Chapman & Hall/CRC (2010), http://dl.acm.org/citation.cfm?id=1882723.1882737
11. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. pp. 218–229. STOC '87, ACM, New York, NY, USA (1987), http://doi.acm.org/10.1145/28395.28420
12. Howe, J.: Jeff Howes Website, http://crowdsourcing.typepad.com, zuletzt abgerufen 30.08.2014
13. Howe, J.: The Rise of Crowdsourcing, http://archive.wired.com/wired/archive/14.06/crowds.html, zuletzt abgerufen 30.08.2014
14. Laud, P., Talviste, R.: Review of the state of the art in secure multiparty computation. Online unter http://usable-security.eu/news/article-2 (2012)
15. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing. pp. 1219–1234. STOC '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2213977.2214086
16. Orlandi, C.: Is multiparty computation any good in practice? In: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. pp. 5848–5851 (May 2011)

Konzeptionelle Fragen und Implementierungsprobleme aktueller DSU-Systeme für Java

Marcel Kost*

Betreuer: Martin Alexander Neumann†

Karlsruher Institut für Technologie (KIT)
Pervasive Computing Systems – TECO
*marcel.kost@student.kit.edu
†mneumann@teco.edu

Zusammenfassung. Wir haben uns aktuelle DSU-Systeme für Java angesehen und versucht herauszufinden, welche konzeptionellen Fragen beim Entwurf entstanden sind und wie man sich jeweils entschieden hat. Dazu beleuchten wir u.a. die Frage, ob die laufende Software vom Update wissen muss oder nicht, und ob damit die DSU-Fähigkeit besser in der Software oder in der VM aufgehoben ist. Außerdem haben wir geschaut, auf welche Probleme die Entwickler bei der Implementierung gestoßen sind, wie schwerwiegend diese Probleme waren und wie viel Aufwand notwendig war, diese zu lösen oder zu umgehen.

Schlüsselwörter: Dynamische Software Updates, Live Updates, Java

1 Einleitung

Software ändert sich ständig. Deshalb werden laufend Softwareupdates und Bugfixes veröffentlicht, die neben Fehlerbehebungen und neuen Features auch immer wieder sicherheitskritische Änderungen beinhalten.

Doch gerade auf Systemen, die eine hohe Verfügbarkeit aufweisen müssen, werden Updates immer erst sehr spät oder im schlimmsten Fall überhaupt nicht aufgespielt. Dazu zählen Systeme wie etwa Mail- und Webserver, alles was mit dem Stichwort Cloud zu tun hat, bis hin zu Telefonanlagen. Selbst Mobiltelefone sind heutzutage Computer, die rund um die Uhr erreichbar sind. Dabei sind all diese Systeme gerade durch ihre dauernde Erreichbarkeit besonders gefährdet gegenüber Angriffen von außen.

Das Problem ist, dass Softwareupdates normalerweise nicht im laufenden Betrieb eingespielt werden können, und daher die Software gestoppt und neu gestartet werden muss. Neben einem oft hohen Aufwand bedeutet dies für Systeme, die eine hohe Verfügbarkeit gewährleisten müssen, vor allem eine lange Downtime, in welcher der Service nicht zur Verfügung steht.

Das gleiche Problem tritt auch bei der Softwareentwicklung auf: Beim Debuggen werden ständig kleine Teile einer Software verändert; zum Testen muss

aber jedes Mal die komplette Software neu gestartet werden. Zwar geht es hier nicht um Verfügbarkeit oder Aufwand, dennoch ist dies ein störender Zeitfaktor in der Entwicklung.

1.1 Die Idee dahinter

Die Idee von Dynamischen Software Updates (DSU) ist nun, genau dies zu ermöglichen: „Softwareupdates im laufenden Betrieb einspielen zu können, ohne die Software stoppen und neu starten zu müssen.“ [8]

Das bedeutet, dass der Zustand einer Software im Speicher komplett erhalten bleiben kann, solange dieser nicht selbst von dem Update betroffen ist. Dadurch kann das Update nach außen hin komplett unbemerkt ablaufen, da I/O-relevante Zustände beibehalten werden können. Außerdem können alle Bestandteile der Software aktualisiert werden, wodurch Updates architekturunabhängig werden.

Die allgemeine Beschreibung stammt von E. Miedes und F. Muñoz-Escoí und beschreibt gut das Hauptziel aller DSU-Systeme. Zugleich bemängeln sie aber auch, dass bisher keine eindeutige Definition für DSU herausgearbeitet wurde. In anderen Zusammenhängen spricht man auch von Live Updates [2], Hot Updates oder Code Evolution [14] - das oben genannte Ziel ist aber immer gleich.

Die ersten Gedanken zu DSU stammen bereits aus dem Jahre 1976 [8] und das erste DSU-System PODUS für C [11] wurde 1991 veröffentlicht. [12] Im letzten Jahrzehnt entstanden dann weitere DSU-Systeme wie Gutpa's [5], PoLUS [1] und UpStare [7], welche hauptsächlich für C geschrieben waren und eine ganze Reihe verschiedener Techniken umsetzten. Einer der bekanntesten Vertreter der letzten Jahre ist Kitsune [6], ebenfalls für C, dessen Code sogar als OpenSource zur Verfügung steht.

Trotz inzwischen weit fortgeschrittener Forschung werden DSU aber kaum eingesetzt. Obwohl schon einige DSU-Systeme als freie Software verfügbar sind, hat die freie Software-Community bis jetzt nur in einzelnen Fällen eines der Systeme oder Ansätze daraus übernommen. [12] Und das, obwohl z.B. die Mehrzahl der Webserver weltweit, welche gute Kandidaten für DSU wären, OpenSource-Software einsetzen. Ein Beispiel für DSU im praktischen Einsatz ist Ksplice von Oracle [9], welches wichtige Updates des Linux-Kernels zur Laufzeit einspielen kann.

1.2 DSU für Java

In den letzten Jahren hat sich die Entwicklung von DSU-Systemen mehr und mehr auf Managed Code Programmiersprachen wie z.B. Java konzentriert, die erst in Bytecode kompiliert und dann mittels einer Virtuellen Maschine (VM) ausgeführt werden. Durch die zusätzliche Schicht zwischen Bytecode und Prozessor kann besser in den Ablauf der Software eingegriffen werden, als es bei Programmiersprachen wie C möglich wäre, deren Code direkt auf dem Prozessor ausgeführt wird. Für die Analyse haben wir uns fünf aktuelle DSU-Systeme für Java herausgesucht:

Wir haben uns mit Rubah (2014) [10] beschäftigt, dessen Entwickler bereits das DSU-System Kitsune [6] für C entwickelt haben. Rubah ist das einzige System, das keine Änderungen an der VM vornimmt.

Unser zweiter Kandidat ist DCEVM (2010) [14], welcher im Gegensatz zu den anderen DSU-Systemen für den Einsatz beim Debuggen gedacht ist, und nicht den Anspruch erhebt Hochverfügbarkeitssysteme sicher aktualisieren zu können.

Wir haben uns JVolve (2009) [13] angesehen, welches auf der Jikes Forschungs-VM aufbaut und sich an den Garbage Collector hängt, um Updates effizient durchzuführen.

Das DSU-System Javelus (2012) [4] nutzt ähnliche Ansätze wie JVolve und führt zusätzlich einen trägen (lazy) Algorithmus ein, der das System nach und nach aktualisiert.

Zu guter Letzt warfen wir noch einen Blick auf Gosh! (2013, früher Javelon) [3], dessen Team von zeroturnaround.com, den Entwicklern des momentan einzigen kommerziell verfügbaren DSU-System JRebel [15], aufgekauft wurde. Sie versprechen eine sehr umfassende Updatefähigkeit, ohne dabei auf Performance zu verzichten.

Anhand dieser fünf DSU-Systeme haben wir beobachtet, was für konzeptionelle Fragen bei der Entwicklung der Systeme aufgetaucht sind und auf was für Probleme die Entwickler am Schluss bei der Implementierung gestoßen sind.

2 Konzeptionelle Fragen

Vor dem Erstellen eines DSU-Systems muss man sich gezwungenermaßen mit ein paar Fragen auseinandersetzen. Dabei geht es zum einen um die Herangehensweise, ob z.B. die virtuelle Maschine Teil des DSU-Systems ist oder ob das System auch auf einer Standard-VM funktionieren soll. Zum anderen geht es um die Ziele, für die man das DSU-System zu optimieren versucht. Dazu zählt auch das primäre Einsatzgebiet. Meist muss man mit einer solchen Entscheidung aber auch Abstriche in anderen Bereichen machen.

Wir haben versucht herauszufinden, vor welche konzeptionellen Entscheidungen die Entwickler der DSU-Systeme gestellt wurden, wie sie sich entschieden haben und wie sich diese Entscheidungen gegenseitig beeinflussten.

2.1 Updatefähigkeit

Die Updatefähigkeit (Update Level) besagt, was für Änderungen ein Update enthalten darf, d.h. welche Programmierkonstrukte während der Ausführung der Software tatsächlich verändert werden können. Sie sagt damit etwas über die Mächtigkeit eines DSU-Systems aus.

Bei den betrachteten DSU-Systemen wird oft von „beliebigen Änderungen“ [14] [4] gesprochen, bei Rubah ist auch von „Release-Level Updates“ die Rede [10]. All das sind aber sehr schwammige Begriffe. Mehrere Autoren haben bereits

versucht die Beschaffenheiten eines Updates zu klassifizieren. Ein paar Herangehensweisen wollen wir hervorheben.

C. Giuffrida und A. Tanenbaum [2] stellten fest, dass Änderungen abstrakt betrachtet entweder den Code oder die Daten einer strukturellen Einheit, also eines Moduls oder einer Klasse, betreffen können. Sie teilten ein Update daraufhin in fünf Kategorien auf: Ein Update betrifft entweder eine strukturelle Einheit selbst, ein Protokoll zur Kommunikation zwischen den strukturellen Einheiten, globale Daten (z.B. error-codes), globale Algorithmen (von denen die strukturellen Einheiten abhängig sind) oder die gespeicherten Daten (z.B. auf der Festplatte). Ihre Auflistung enthält auch Änderungen an Ressourcen, d.h. Betriebssystem und Hardware. Da wir uns hier auf Softwareupdates beschränken, gehen wir darauf nicht näher ein.

Etwas konkreter hat sich das Team um T. Würthinger [14] mit der Frage um die Klassifizierung eines Updates auseinandergesetzt. Sie haben sich überlegt, welche Änderungen in einem Java-Programm vorkommen können, und wie sich diese hierarchisch anordnen lassen. (Abb. 1) Der einfachste Fall ist das Austauschen von Methodenrümpfen, d.h. der Implementierung einer einzelnen Methode. Dabei wird lediglich Bytecode ausgetauscht, was zu keiner Inkonsistenz gegenüber anderen Methoden oder Typen führen kann. Dies wird seit 2002 unter dem Namen Hot Swap auch von der Java HotSpot™ VM direkt unterstützt. Alle weiteren Änderungen, die das Hinzufügen oder Löschen von Methoden, Variablen oder Oberklassen (Supertypes) betreffen, werden aufgeteilt in binär kompatible und binär inkompatible Änderungen. Das Hinzufügen von neuen Methoden, Variablen oder Oberklassen fällt unter die binär kompatiblen Änderungen, da Bytecode aus der alten Version dadurch nicht beeinflusst wird. Das Löschen dieser ist jedoch eine binär inkompatible Änderung (in der Abbildung dunkelgrau hinterlegt), da Bytecode aus der alten Version nun unter Umständen nicht mehr gültig ist, da er nicht mehr existente Referenzen enthält. Das bedeutet, dass bei einem Update sichergestellt werden muss, dass mit dem Einspielen der Änderungen kein alter Bytecode mehr ausgeführt wird. Zusammengefasst stellen wir fest, dass im allgemeinen das Hinzufügen von Klassen, Methoden und Variablen weniger Probleme bereitet als das Löschen.

Die Entwickler von Gosh! [3] haben in einer Tabelle mögliche Änderungen an einem Java-Programm aufgetragen und die Updatefähigkeiten ihres eigenen Systems mit denen der DSU-Systeme JRebel [15] und DCEVM [14] verglichen. (Abb. 2) Anhand dieser Tabelle kann man sehen, welche Arten von Problemen durch Änderungen bei DSU auftreten können. Das sind zum einen statische Variablen, die in den Klassendefinitionen gespeichert werden, und zum anderen ist es das Verschieben von Variablen und Klassen an eine andere Stelle der Hierarchie. Dabei soll der Inhalt gleich bleiben, d.h. zusätzlich zum Löschen und Hinzufügen muss der Zustand vom alten zum neuen Objekt überführt werden.

Anhand dieser Transformation, die nötig ist um dem Zustand eines alten Objekts in das neue zu überführen, kann man die Komplexität eines Updates

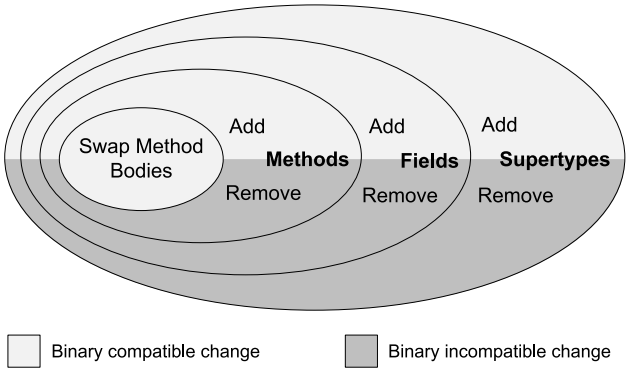


Abb. 1. Ebenen der Code Evolution nach DCEVM [14]

messen. Während Javelus [4], JVolve [13] und Rubah [10] Transformationen der Daten vom alten zum neuen Objekt vornehmen, die der Benutzer vor dem Update anpassen kann, so verzichten die Entwickler von DCEVM [14] absichtlich auf diese Möglichkeit. Sie begründen ihre Wahl damit, dass ihr System für den Einsatz beim Debuggen gedacht ist, und daher die Transformation der einfachen Bedienbarkeit weichen muss. Änderungen lassen sich dafür auch mit einem einzigen Klick in das laufende System einspielen.

Findet diese Transformation zwischen mehreren Objekten statt, also nicht nur von der einen alten zur neuen Version des Objekts, so sprechen die Entwickler von Javelus von „komplexen Transformationen“ (Complex Conversions) [4]. Solche Transformationen können bis dato von keinem DSU-System vorgenommen werden. Die Entwickler sind sich einig, dass sich DSU ab einer bestimmten Komplexität der Updates nicht mehr lohnt. C. Giuffrida und A. Tanenbaum [2] beschreiben sogar ein paar Fälle, in denen DSU auf Grund von gewissen Abhängigkeiten grundsätzlich nicht möglich sind. Für die meisten Fälle reichen die Möglichkeiten der gezeigten DSU-Systeme jedoch völlig aus.

Bei DSU-Systemen, die auf Software im produktiven Bereich abzielen, wird die Updatefähigkeit gerne durch das Aufspielen regulärer Updates auf bekannte OpenSource-Software gezeigt. Die Entwickler von Rubah [10] haben z.B mehrere reguläre Updates in die OpenSource-Datenbanken H2 und Voldemort eingespielt. Mit JVolve [13] wurde der Webserver Jetty, der JavaEmailServer und der CrossFPT Server aktualisiert. Hierfür mussten zwar sowohl die Software als auch die Updates angepasst werden, aber es zeigt, dass DSU mit vertretbarem Aufwand schon im produktiven Bereich eingesetzt werden könnte.

2.2 Änderungen an der VM

In Managed Languages wie Java wird das Programm zunächst in Bytecode kompiliert, welcher dann von einer Virtuellen Maschine ausgeführt wird. Durch diese zusätzliche Schicht kann in den Ablauf des Programms eingegriffen werden wodurch es nahe liegt, ein DSU-System mit Hilfe der VM zu realisieren.

Code change	*Gosh!*	*JRebel*	*DCEVM*
Changes to method bodies	✓	✓	✓
Adding/removing fields	✓	✓	✓
Adding/removing methods	✓	✓	✓
Adding/removing constructors	✓	✓	✓
Adding/removing classes[i]	✓	✓	⚠
Replace superclass	✓	✗	✓
Adding/removing implemented interfaces	✓	✗	✓
Automatic new instance field initialization (developer-defined default value)[ii]	⚠	✗	✗
Automatic new static field initialization (developer-defined default value)[iii, iv]	⚠	⚠	✗
Move field to super class (preserving the state)[iv]	✓	✗	⚠
Move field to sub class (preserving the state)[iv]	✓	✗	⚠
Changing static field value[iii]	✗	⚠	✗
Changing primitive static final field value[v]	✓	⚠	⚠
Adding/removing enum values[vi]	✓	⚠	✗

✓ Supported ✗ Not supported ⚠ Issues

Abb. 2. Vergleich der Updatefähigkeit nach Gosh! [3]

Diesen Ansatz haben auch die meisten Entwickler gewählt. Das DSU-System JVolve [13] wurde in die Jikes RVM, eine Java-in-Java Forschungs-VM, integriert. Für Javelus [4] und DCEVM [14] dient die professionell eingesetzte Java HotSpot™ VM als Grundlage. Beide benutzen die bereitgestellte HotSwap-Funktion, mit welcher sich Methodenrümpfe austauschen lassen. Die Entwickler von DCEVM betonen, dass die Änderungen an der VM so gering wie möglich gehalten wurden. Auch die Entwickler von Javelus sind der Meinung, dass viele der Ideen, die sie in ihrem DSU-System umgesetzt haben, auf andere VM übertragen werden könnten.

Der Ansatz des DSU-Systems Rubah [10] unterscheidet sich grundlegend von den anderen Ansätzen. Rubah wurde mittels Bibliotheken und der verfügbaren HotSwap-Funktion entworfen und lässt sich damit auf jeder Standard-JVM einsetzen, die diese Funktion anbietet. Die einzige Änderung ist ein von Rubah angepasster Classloader. Da dem System damit die Sicht von außen fehlt, können viele Dinge nicht so umgesetzt werden, wie es im Zusammenspiel mit einer VM möglich wäre. Dennoch kann Rubah trotz der fehlenden Möglichkeiten mit den VM-orientierten Ansätzen mithalten. Da man für alle anderen Ansätze eine

veränderte virtuelle Maschine braucht, wird Rubah somit zu einer einfach zu implementierenden Variante.

Gosh! [3] nutzt zum Laden der Klassen ebenfalls einen eigenen Classloader, die DSU-Systeme DCEVM und Javelus kommen dagegen ohne aus. Die Entwickler begründen ihre Entscheidung für eine Anpassung der VM aber gegen einen eigenen Classloader damit, dass Software im professionellen Bereich häufig eigene angepasste Classloader nutzt. Das Austauschen der VM ist somit einfacher und sicherer, als den Classloader der Software anzupassen. Beide Ansätze sind begründet und zielen auf verschiedenen Einsatzgebiete ab.

2.3 Die Software als Teils des DSU-Systems

Ähnlich zur letzten Frage ist die Entscheidung, ob das Update der zu aktualisierenden Software bewusst ist oder nicht. Bei einem DSU-System ohne Anpassungen der VM, wie Rubah [10] es ist, muss die Software gezwungenermaßen Teil des DSU-Systems sein, d.h. implizit auch vom Update und dessen Zeitpunkt wissen. Andererseits weiß das DSU-System auch vom Zustand der Software und kann so gegebenenfalls bessere Entscheidungen z.B. zum Update-Zeitpunkt treffen. Muss die Software allerdings nichts vom DSU-System und dem Update wissen, so kann jedes beliebige Java-Programm im laufenden Betrieb aktualisiert werden. Mehraufwand, der entsteht um das System DSU-fähig zu machen, fällt damit weg.

Die meisten Ansätze der DSU-Systeme, die wir uns angesehen haben, sehen die DSU-Funktionalität ausschließlich in der VM vor, wodurch die zu aktualisierende Software nicht für DSU ausgelegt sein muss. Nach der Meinung von C. Giuffrida und A. Tanenbaum [2] allerdings sollte die Software DSU aktiv unterstützen, was vor allem bedeutet, dass sie davon wissen muss. Außerdem sollte ein Update neben den bloßen Änderungen auch weitere Informationen über den Zusammenhang enthalten, sodass die Software selbst entscheiden kann, wann ein guter und sicherer Zeitpunkt zum Einspielen wäre. Sie bemängeln, dass dadurch, dass die Verantwortung für das Update allein beim DSU-System liegt, ein sehr hohes Fehlerrisiko in Kauf genommen wird, obwohl der Entwickler, der ein Update entwickelt hat, genau weiß, durch welche Änderungen Fehler auftreten könnten und worauf beim Einspielen geachtet werden sollte.

2.4 Schnell oder träge

In der Regel werden dynamische Updates schnell (eager) eingespielt, indem das System angehalten wird, um die veränderten Klassen auszutauschen und die Instantiierungen auf dem Heap zu aktualisieren. Allerdings kann dies bei vielen Klassen bis zu ein paar Sekunden dauern, in denen das System de facto nicht ausgeführt wird. Für manche Anwendungen ist selbst diese kurze Unterbrechung zu lange.

Um die Unterbrechungszeit (disruption time) zu reduzieren, verwenden einige DSU-Systeme einen trägen Ansatz (lazy). Dabei werden die einzelnen Klassen erst dann aktualisiert, wenn sie das nächste Mal benötigt werden. Während

der Unterbrechung müssen somit nur gewisse Mechanismen eingebaut werden, welche das Update einer Klasse veranlassen, sobald diese aufgerufen wird.

Leider verursachen die benötigten Mechanismen einen Mehraufwand (Overhead), welcher sich durch das laufende System mitzieht, bis alle Klassen aktualisiert wurden. [4]. Das System befindet sich nach dem Anstoßen des Updates eine gewisse Zeit in einem Mischzustand zwischen der alten und der neuen Version. Dies kann unter Umständen sehr lange dauern, da z.B. Klassen, die Absturzberichte erstellen, im Idealfall nie aufgerufen werden. Dann kann es vorkommen, dass ein zweites Update eingespielt wird, während noch nicht einmal alle Klassen des ersten Updates aktualisiert wurden. Durch die Implementierung muss also sichergestellt werden, dass die Versionen auf keinen Fall vermischt werden und dadurch veralteter Code ausgeführt wird.

Sowohl Rubah [10] als auch Javelus [4] verwenden einen trägen Ansatz. Bei Javelus wird nach einem Update jedes veraltete Objekt als ungültig markiert. Diese Markierung ist für die laufende Software nicht sichtbar. Zur Laufzeit wird dann überprüft, ob eine ungültige Referenz verwendet wird und bei Bedarf werden die Änderungen ausgeführt. Da Rubah ohne die VM auskommt, kann die Information nicht vor der Software versteckt werden. Stattdessen wird die Software vor dem ersten Start präpariert, sodass jede Klasse durch ein gesetztes Flag zu einem Proxy von sich selbst werden kann und jeden Aufruf an die Rubah-API weiterleitet. Wird die API über einen Proxy aufgerufen, so wird das betroffene Objekt aktualisiert und alle referenzierten Klassen ebenfalls zu Proxies gemacht. So wird eine Klasse erst dann zum Proxy, wenn sie von einer gerade aktualisierten Klasse referenziert wird.

Der träge Ansatz verkürzt zwar deutlich die Unterbrechungszeit, doch bringt er viel Komplexität ins Spiel. Ein schnelles Update ist effizienter [4] und kann einfach auf Fehler und Korrektheit überprüft werden, während Fehler beim trägen Ansatz erst sehr spät auftreten können. Die Entwickler von Rubah [10] haben für ihr DSU-System nicht nur einen trägen, sondern auch einen schnellen Algorithmus implementiert, der parallel mit mehreren Threads arbeitet. Sie zeigen damit, dass auch in dem schnellen Ansatz noch Potential vorhanden ist.

2.5 Update-Zeitpunkt

Die Frage, ob das Update schnell oder träge durchgeführt wird, geht fast fließend in die Frage nach dem richtigen Zeitpunkt über. Damit ein Update sicher eingespielt oder initiiert werden kann, müssten alle Threads zu einem sicheren Zeitpunkt (safe update point) angehalten werden. Allerdings soll bei einem dringenden Update nicht lange gewartet werden.

Für den sicheren Zeitpunkt gibt es verschiedene Definitionen. Oft wird als Voraussetzung ein ruhiger Zustand (quiescent state) genannt. Je nach Autor und DSU-System bedeutet dies, dass entweder keine Funktion, die aktualisiert werden soll, momentan ausgeführt wird, oder generell keine Funktion auf dem Stack liegt. [8] Weiß die Software von dem Update, so kann sie einen geschickten Zeitpunkt bestimmen oder zumindest mögliche Zeitpunkte vorschlagen, in denen ein Update möglich ist. Der Entwickler muss sich daher gute Stellen im Code

aussuchen, zu dessen Ausführung der Zeitpunkt sicher ist, oder einen Algorithmus entwerfen, der zur Laufzeit dynamisch einen guten Zeitpunkt auswählt. Als guten Zeitpunkt empfehlen die Entwickler von Rubah [10] den Anfang langer Loops. Im anderen Fall liegt die Entscheidung für den richtigen Zeitpunkt allein bei der VM. Da sie nicht aktiv in den Ablauf eingreifen kann, kann sie nur entscheiden, ob der aktuelle Zustand ausreichend ist, ober ob noch gewartet werden soll. Dabei läuft sie Gefahr einen solchen Zustand nie zu erreichen.

Nach Spezifikation von Oracle/Sun kann jede Java VM eine laufende Software anhalten, indem sie jeden Thread an einem safe-point stoppt. In einem safe-point sind alle Objekte auf dem Heap konsistent und vollständig. Die Java VM garantiert, dass alle Threads in endlicher Zeit einen safe-point erreichen können. [14] Dies wird normalerweise genutzt, um den Garbage Collector anzustoßen. Es bietet sich an, safe-points als Grundlage für einen guten Update-Zeitpunkt zu nutzen. [2] Ist die zu aktualisierende Komponente zu so einem Zeitpunkt komplett unangetastet, d.h. sie wird weder ausgeführt, noch wird ein andere Komponente ausgeführt, die irgendwann einen Zugriff verursachen könnte, so kann sie direkt ausgetauscht werden. Ist dies nicht der Fall sind komplexe Transformationen im gesamten System notwendig, damit die Komponente direkt weiter verwendet werden kann. Während bei DCEVM [14] jeder safe-point für ein Update in Frage kommt, so werden bei JVolve [13] noch weitere Bedingungen an die Methoden auf dem Stack gestellt. Die Entwickler von Javelus [4] haben das Konzept von JVolve übernommen.

2.6 Notwendiger Programmieraufwand

Um die von uns betrachteten DSU-System nutzen zu können, müssen sowohl vorab an der Software, also auch bei jedem Update, eventuell Anpassungen vorgenommen werden. Die Entwickler von Gosh! [3] schreiben über dem notwendigen Programmieraufwand folgendes: „Wir nennen ein DSU-System endbenutzertransparent, wenn es keinen Eingriff eines Benutzers während des Updates braucht, und dementsprechend nennen wir es entwicklertransparent, wenn vom Entwickler keine besonderen Vorkehrungen getroffen werden müssen.“ [3] Sie sind auch der Meinung, das die Benutzer- und Entwicklertransparenz eine entscheidende Rolle darin spielt, wie erfolgreich das DSU-System sein wird.

Bis auf Rubah [10], welches VM-unabhängig ist, sind alle von uns betrachteten DSU-Systeme entwicklertransparent, d.h. die Software benötigt vorab keine Anpassungen. Um eine Software mit Rubah zu starten, müssen die sicheren Zeitpunkte (update points) festgelegt werden, sowie Anpassungen bei der Initialisierung und bei der Fehlerbehandlung gemacht werden. Bei den anderen DSU-Systemen wird der Zeitpunkt des Updates ausschließlich durch die VM bestimmt. Die Entwickler von Rubah bewerben ihr DSU-System damit, dass nur wenige Änderungen in der Software erforderlich sind. Sie schreiben, dass aus eigener Erfahrung die Anzahl der zu ändernden Zeilen in der Größenordnung um die 100 liegt.

E. Smith, M. Hicks und J. Foster [12] beklagen, dass die zur Messung der Bedienungsfreundlichkeit oft genutzte Anzahl der verändernden Codezeilen eine

sehr schlechte Messgröße ist. Zwar sind die Anpassungen meist nicht groß, doch ist dafür tiefgründiges Wissen sowohl über DSU, als auch über die Software nötig. Des Weiteren kritisieren sie, dass viele Entwickler ihre DSU-Systeme mit ihrer eigenen Software testen, anstatt vergleichbare Software zu nehmen.

Komplett benutzertransparent ist bei uns nur das DSU-System DECVM [14], bei welchem ein Update mit einem einzigen Klick direkt aus der Entwicklungsumgebung eingespielt werden kann. Allerdings ist es auch für schnelles Debuggen während der Entwicklung ausgelegt und verzichtet komplett auf Transformationen und Sicherheitsmechanismen. Ein Vergleich fällt daher schwer. Bei den anderen DSU-Systemen läuft das Einspielen eines Updates ähnlich ab: Alle liefern einen Quellcode-Analysierer (sourcecode analyzer) mit, welcher aus dem Quellcode der neuen und der alten Version eine Änderungsliste erstellt, die an die laufende Software bzw. die VM gegeben wird. Die Transformationen der alten auf die neuen Klassen sind einfache Java-Klassen, die der Entwickler zuvor anpassen kann.

2.7 Update-Sicherheit

Gerade wenn große Systeme mit hoher Verfügbarkeit aktualisiert werden muss sichergestellt werden, dass während des Updates keine Probleme auftreten. Diese könnten zum Absturz des Systems führen, wodurch der komplette Zustand, der im Speicher liegt, verloren geht. Die laufende Software sollte sich nach der Aktualisierung exakt so verhalten, wie nach einem Neustart der neuen Version. [13] Gerade wenn auf träge Algorithmen zur Aktualisierung gesetzt wird kann es vorkommen, dass Fehler erst lange Zeit nach dem eigentlichen Update auftreten und sich dann fortpflanzen.

JVolve [13] behauptet von sich besonders sicher zu sein, was sich hauptsächlich auf die Typsicherheit bezieht. Da Rubah [10] keinen Zugriff auf die VM hat, werden viele Änderungen im Code der Software vorgenommen, die durch Eigenschaften der Programmiersprache Java notwendig werden. Dadurch entsteht ein hohes Fehlerpotential, welches auch zur Verwirrung bei der Fehlersuche führt.

Da es für das Debuggen bei der Entwicklung gedacht ist, setzt DCEVM [14] keinerlei Sicherheitsmechanismen ein. Updates zu ungünstigen Zeitpunkten oder in ungünstigen Konstellationen bringen das System zum Absturz. Aber selbst bei Gosh! [3], das sich mit dem kommerziellen JRebel [15] vergleicht, können Updates unter Umständen Probleme oder sogar Abstürze verursachen. Die Phänomene, die bei verschiedenen Aktualisierungen auftreten können, haben die Entwickler von Gosh! in einer Tabelle gegliedert. (Abb. 3)

Manche DSU-Systeme bieten ein Rollback an [8], mit dem von einer neueren Version zurück zur alten gewechselt werden kann. Von den hier betrachteten DSU-Systemen wird dies jedoch nicht unterstützt.

3 Implementierungsprobleme

Nachdem die konzeptionellen Fragen entschieden sind, kommt die Implementierung des DSU-Systems. Dass es hierbei öfters zu unvorhergesehenen Problemen

ID	Code change description	Possible run-time phenomenon
7	Class removed	Phantom objects
8	Class renamed	Phantom objects / Lost State
16	Modifier abstract added to class	Phantom objects
6	Class added	Absent state
22	Super class of class changed	Absent state
68/ 71	Instance/static field added to class	Absent state
21	Modifier static removed from inner class	Absent state
70/ 73	Instance/static field type changed in class	Lost state
65	Static initialization impl. changed in class	Oblivious update
30	Constructor impl. changed in class	Oblivious update
114	Static field value changed	Broken assumption
38/ 44	Instance/static method impl. changed (e.g., conditional statement, method split / merged)	Broken assumption / Transient inconsistency

Abb. 3. Mögliche Laufzeitphänomene nach Gosh! [3]

kommt liegt teilweise an den konzeptionellen Entscheidungen, oft aber auch an den Eigenschaften von Java.

Die Entwickler der DSU-Systeme schreiben unterschiedlich viel über die Probleme, auf die sie bei der Entwicklung gestoßen sind. Von manchen DSU-Systemen erfahren wir daher sehr wenig über Implementierung. Das ist schade, da oft erst die Details interessant sind und zum Vergleich taugen. Die Entwickler von Rubah [10] beschreiben sehr ausführlich, welche Probleme sie bei der Implementierung hatten, und wie sie diese zu lösen versuchten. Durch die Umsetzung komplett ohne Änderungen an der VM wurden sie allerdings auch mit mehr Problemen konfrontiert, da ihnen nur die Sicht eines ausgeführten Java-Programms auf die VM bleibt.

Wir möchten im Folgenden ein paar Probleme aufzeigen, die entweder bei mehreren DSU-Systemen aufgetaucht sind, oder die spezielle Maßnahmen erfordert haben, und schauen uns an, ob sie gelöst wurden und wenn ja wie.

3.1 Umsetzung schneller und träger Algorithmen

Um Updates durchzuführen und sicherzustellen, dass alles aktualisiert wurde, ist ein atomarer Zugriff auf alle zu aktualisierenden Instanzen nötig. (atomicity)

Dieser Zugriff wird umso teurer, desto höher die Ebene ist, auf der er stattfinden muss. [2] Bei einem System müssen daher alle Threads gleichzeitig gestoppt werden, was eine lange Unterbrechungszeit mit sich zieht. Muss sogar ein Protokoll aktualisiert werden, mit dem mehrere Systeme kommunizieren, so muss der atomare Zugriff auf allen Systemen gleichzeitig stattfinden. Bei schnellen Updates dauert dieser atomare Zugriff besonders lange.

Spätestens wenn mehrere verteilte Systeme aktualisiert werden müssen, bietet sich ein träger Algorithmus an, wie Rubah [10] und Javelus [4] ihn nutzen. Dieser ist darauf ausgelegt, dass nicht alle Klassen sofort wieder gebraucht werden. Die für einen trägen Algorithmus benötigten Mechanismen, die die Updates der einzelnen Instanzen anstoßen, brauchen jedoch Speicher und Rechenzeit. Zwar braucht dieser Ansatz ebenfalls einen atomaren Zugriff, doch der Aufwand währenddessen wird geringer. In ungünstigen Konstellationen können träge Algorithmen zum Speicherüberlauf (stack overflow) oder sogar zum Live Lock führen. [10] Die Entwickler sind sich dem Problem bewusst, haben jedoch keinen Schutz implementiert, der dies verhindern kann.

3.2 Gleiche Klasse andere Version

Ein DSU-System muss nach einem Update garantieren können, dass kein alter Code mehr ausgeführt wird. Dies lässt sich am einfachsten erreichen, indem man dafür sorgt, dass nach einem Update kein alter Code mehr im System vorhanden ist. Bei den verwendeten Ansätzen ist dies jedoch praktisch nicht möglich. Um Daten von einer alten in eine neue Klasse transferieren zu können, müssen beide Versionen im System geladen sein. Java verbietet jedoch mehrere Klassen mit dem gleichen Namen im gleichen Classloader.

Um dennoch mehrere Versionen der gleichen Klasse im gleichen Classloader halten zu können, benennt Rubah [10] alle verwendeten Klassen um und fügt die jeweilige Version dem Klassennamen hinzu. Die Klasse java.lang.Thread wird gegen eine eigene Klasse ausgetauscht, mit der beim Erstellen eines neuen Threads automatisch Rubah's eigener Classloader initiiert wird. Damit die Aufrufe der Reflection-API (z.B. class.forName()) weiterhin funktionieren, überschreibt Rubah ebenfalls sowohl vor dem ersten Start als auch bei jedem Update alle Aufrufe und ersetzt sie mit einem Aufruf der Rubah-API, welcher die Klasse mit der aktuellen Version zurückgibt.

Die Umbenennung führt auch dazu, dass Klassen, die sich bei einem Update nicht geändert haben, plötzlich Referenzen mit falschem Typ enthalten, da die neue Version der Klasse einen anderen Namen hat. Um nicht alle Klassen bei einem Update aktualisieren zu müssen, haben sich die Entwickler von Rubah [10] zu einer radikalen Variante entschieden: Alle Variablen und Argumente in Methoden, die auf aktualisierbare Klassen zeigen, werden vorab zum Typ java.lang.Object geändert und bei jeder Verwendung ein Cast eingefügt. Damit sind Klassen unabhängig von Änderungen in referenzierten Klassen. Allerdings bringt diese Methode einen gewissen Mehraufwand bei der Ausführung (Overhead) mit sich. Die Entwickler sind jedoch der Meinung, dass der JIT-Compiler die Regelmäßigkeiten erkennt und dementsprechend optimiert.

Das DSU-System JVolve [13] umgeht dieses Problem, indem es mit Hilfe der VM nur die alten Klassen umbenennt und nur die neusten Klassen im System ihren richtigen Namen tragen. Gosh! [3] dagegen nutzt einen neuen Classloader für jede Version. In jedem Classloader gilt ein eigener Namensraum, wodurch gleiche Klassen unterschiedlicher Version existieren können. Jedoch erschwert das die Transformation und den Zugriff untereinander erheblich und ist nur mit zusätzlichen Klassen möglich, die eine andere Version einer Klasse im gleichen Namensraum repräsentieren.

3.3 Private und konstante Variablen

Um die Daten einer alten Klasse auf die neue zu transferieren, bieten die meisten DSU-Systeme Transferfunktionen an. Diese sind in Java geschriebene Klassen, die die Daten aus dem alten Objekt auslesen und in die Variablen des neuen Objekts schreiben. Wie es die Java-Spezifikation vorsieht, kann allerdings nicht auf die privaten Variablen eines Objekts zugegriffen werden. Genauso wenig können Werte in konstante Variablen (final) geschrieben werden.

Um die Transferfunktion dennoch in Java implementieren zu können, werden die Transformationsklassen bei JVolve [13] mit einem separaten Compiler übersetzt, welcher so modifiziert wurde, dass die Sichtbarkeits- und Schreibschutzmechanismen ignoriert werden. Dadurch entsteht ungültiger Bytecode, der von Bytecode-Verifizierern erkannt werden würde. JVolve wurde in der Jikes RVM, einer Forschungs-VM, umgesetzt, welche keine Bytecode-Verifizierung verwendet. In klassischen VM wäre dies allerdings nicht möglich.

3.4 Optimierungen der VM

Aktuelle VMs sind sehr mächtig und beschränken sich nicht mehr nur auf das Ausführen von Bytecode, sondern Optimieren ihn und kompilieren oft benötigte Abschnitte automatisch zur Laufzeit. Des Weiteren werden verschiedenen Caches eingesetzt, um den Zugriff auf Code und Daten zu beschleunigen.

Leider sind die verfügbaren VMs noch nicht für DSU und die damit verbundenen Eigenschaften ausgelegt. So rechnet eine normale VM nicht damit, dass sich der Code zur Laufzeit ändern kann. Selbst wenn eine DSU-Unterstützung nachträglich integriert wird, so wie es die meisten DSU-Systeme machen, sind die restlichen Komponenten ohne das Konzept von DSU entworfen worden.

Daher müssen viele Optimierungsmöglichkeiten der VM bei einem Update zurückgesetzt oder komplett deaktiviert werden, damit keine alte Funktionalität übrig bleibt. So muss auch bei DCEVM [14] der JIT-Compiler deoptimiert werden werden, damit keine falschen Annahmen bei der Ausführung gemacht werden. Aber gerade diese Optimierungen machen Java im praktischen Einsatz so performant, gerade im professionellen Bereich.

Die Entwickler von Javelus [4] kritisieren auch die oft genutzte Herangehensweise, den Updateprozess an den Garbage Collector zu binden, da immer öfter parallele und inkrementelle Garbage Collectoren genutzt werden. DSU müssen

auch in der Optimierung unterstützt werden, anstatt diese einfach zu deaktivieren.

4 Zusammenfassung

Java eignet sich durch die zusätzliche Schicht zwischen Bytecode und Prozessor sehr gut für DSU. Da Java eine Vielzahl an Programmierkonstrukten unterstützt, hat man dementsprechend auch mit mehr Fällen, wie der Typisierung oder der Klassenhierarchie, zu kämpfen. Während es z.B. bei einem in C geschriebenen Programm reicht Methoden im Speicher zu überschreiben, so müssen für Java weitaus mehr Funktionen zur Verfügung gestellt werden.

Uns gefällt der Ansatz von Rubah [10], ein DSU-System völlig ohne Änderungen an der VM zu bauen, das auf jeder beliebigen VM läuft. Allerdings überzeugt uns die Argumentation der Entwickler von Gosh! [3], dass Software im professionellen Bereich häufig eigene angepasste Classloader nutzt, und das Verwenden einer angepassten VM in diesen Fällen einfacher ist. Bevor DSU auch im privaten Bereich eingesetzt werden wird, wie z.B. für das bequeme Aktualisieren des Browsers ohne ihn neu starten zu müssen, wird sich eher die VM als die Software selbst an DSU anpassen. Vielleicht finden ja neben der HotSwap-Funktion auch weitere nützliche DSU-Funktionen Einzug in kommende Versionen der Spezifikation der Java Virtual Machine (JVM).

Die Entwickler von Gosh! [3] haben festgestellt, dass die Bedienbarkeit eines DSU-Systems hinsichtlich der notwendigen Anpassungen an der Software entscheidend ist für dessen Erfolg. Dies widerspricht sich jedoch mit der Meinung von C. Giuffrida und A. Tanenbaum [2], dass eine Software DSU aktiv unterstützen sollte und damit Teil des DSU-Prozesses wird, um DSU sinnvoller und damit performanter anzuwenden.

In der Entwicklung werden DSU bereits für schnelleres Debuggen eingesetzt, dort existieren sogar schon kommerzielle Lösungen. Dieser Einsatz in der Entwicklung muss jedoch klar abgetrennt werden vom produktiven Einsatz, wo Systeme Wochen, Monate oder sogar Jahre lang mit Updates versorgt werden sollen, ohne sie neu starten zu müssen.

Damit sich DSU dort durchsetzen können, muss ein viel größerer Wert auf die Sicherheit und Stabilität der Updates gelegt werden. Gerade bei Systemen mit hoher Verfügbarkeit, wo sich DSU anbieten würde, ist das ein KO-Kriterium. Es müssen Möglichkeiten bestehen ein Update zu verifizieren und eventuell mit einem Rollback rückgängig machen zu können. Dies muss auch funktionieren, wenn schon während dem Update Probleme auftreten.

Das wichtigste ist jedoch, dass erste OpenSource-Projekte anfangen sich für DSU zu interessieren und beginnen, Ansätze aus den Arbeiten zu übernehmen und weiterzuentwickeln.

Literatur

1. Chen, H., Yu, J., Chen, R., Zang, B., Yew, P.C.: Polus: A powerful live updating system. In: Proceedings of the 29th international conference on Software Engineering. pp. 271–281. IEEE Computer Society (2007)
2. Giuffrida, C., Tanenbaum, A.S.: A taxonomy of live updates. In: Proc. of the 16th ASCI Conf (2010)
3. Gregersen, A.R., Rasmussen, M., Jørgensen, B.N.: Dynamic software updating with gosh! (2014)
4. Gu, T., Cao, C., Xu, C., Ma, X., Zhang, L., Lu, J.: Javelus: A low disruptive approach to dynamic software updates. In: APSEC. pp. 527–536 (2012)
5. Gupta, D., Jalote, P.: On-line software version change using state transfer between processes. Software: Practice and Experience 23(9), 949–964 (1993)
6. Hayden, C.M., Smith, E.K., Denchev, M., Hicks, M., Foster, J.S.: Kitsune: Efficient, general-purpose dynamic software updating for c. In: ACM SIGPLAN Notices. vol. 47, pp. 249–264. ACM (2012)
7. Makris, K., Bazzi, R.A.: Immediate multi-threaded dynamic software updates using stack reconstruction. In: USENIX Annual Technical Conference. vol. 2009 (2009)
8. Miedes, E., Muñoz-Escoí: A survey about dynamic software updating. Tech. rep., Tech. Rep. ITI-SIDI-2012/003, Instituto Universitario Mixto Tecnológico de Informática, Universitat Politècnica de València (2012)
9. Oracle: Never reboot linux for linux security updates | ksplice, `http://www.ksplice.com/`
10. Pina, L., Veiga, L., Hicks, M.: Rubah: DSU for java on a stock JVM. In: Proceedings of the ACM Conference on Object-Oriented Programming Languages, Systems, and Applications (OOPSLA) (Oct 2014), conditionally accepted
11. Segal, M., Frieder, O.: On-the-fly program modification: systems for dynamic updating. Software, IEEE 10(2), 53–65 (March 1993)
12. Smith, E.K., Hicks, M., Foster, J.S.: Towards standardized benchmarks for dynamic software updating systems. In: Hot Topics in Software Upgrades (HotSWUp), 2012 Fourth Workshop on. pp. 11–15. IEEE (2012)
13. Subramanian, S., Hicks, M., McKinley, K.S.: Dynamic software updates: a vm-centric approach. ACM Sigplan Notices 44(6), 1–12 (2009)
14. Würthinger, T., Wimmer, C., Stadler, L.: Dynamic code evolution for java. In: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java. pp. 10–19. ACM (2010)
15. ZeroTurnaround: Jrebel java plugin, `http://zeroturnaround.com/software/jrebel/`

Approaches for Evaluation of User-Experience of APIs

Vincent-Johannes Schnitzbauer*
Advisor: Dr. Till Riedel†

Karlsruhe Institute of Technology (KIT)
Pervasive Computing Systems – TECO
*uagmv@student.kit.edu
†ridel@teco.edu

Abstract. Various methods to evaluate what makes Application Programming Interfaces difficult to use and how to improve usability are presented. The cognitive dimension framework, with the programmer seen as human, provides breeding ground for the tested real-world approaches, presented for evaluation of user-experience. Different methods are categorized and compared against automatic evaluation tools. Advice when to use which method concerning effectiveness and cost is given.

Keywords: Software Engineering, Human-Computer Interaction, Application Program Interface (API), Evaluation, Usability, Cognitive Dimensions, Documentation

1 Motivation

To make digital code efficiently and dependably reusable, it became common to develop Application Programming Interfaces (APIs). This pattern makes it possible to reuse existing solutions in a new context without modifying or understanding their inner functionality. Therefore API designers are required to create an easy, learnable and extendable framework which meets this challenge. Additionally, it has to be hard to misuse as well as being appropriate for the intended user group [2]. However there is great evidence that many APIs are difficult to use [2,5,15,16]. It's intuitively understandable that the quality of an API, used for development as part of a finished product, has an influence on it: Poor APIs directly increase development cost [11].
To improve the user-experience in the field, an API has to be improved to be attractive for actual and potential user groups. API designers hence have an eligible right to search for methods for evaluating their work. This work aims to describe and compare functioning real-world methods, in order to evaluate user-experience of APIs. It's the overall goal to categorize them and help deciding, when to use which method for evaluation. Important criteria are, amongst others, the cost of execution and the adequacy at the current API lifecycle stage.

The question of usability is easy to answer in the end-customer segment. Clear demands are given and if the product does not meet the expectations it can be relatively easy identified why. Missing functionality can be reasonably identified due to the obviousness and knowledge of the processes and procedures (e.g. text-processing system or spreadsheet program). By contrast, APIs in the past and present are to a large extend business to business solutions, used by experts. In the last decade, the API area has been changing and the semi-professionals and non-specialist user groups have grown and are of growing importance.

Pervasive computing, for example, addresses a wide range of users with different knowledge, contexts and expectations, depending on the API used in the product. To create APIs which solve problems in the way users expect it, API designers have to face those challenges.

The following section will introduce the theoretical background to the currently used API user-experience evaluation methods.

2 Background

Current API evaluation methods emerged from two different branches. The first to mention is the *cognitive dimensions branch* emerging from usability analysis of visual programming environments. The chronology of the publications shows when advances in research were made and how they influenced following studies. The work of Clark in 2004 [4] connected the cognitive dimensions brought up by Green in 1996 [8] with the API evaluation topic. Following the citations in other studies his initiative had a ground breaking effect on the research in this field.Those methods are often informal and simple in usage.
The second branch is the code metrics branch, which connects software cost estimation models like COCOMO with the codebase by which the API is used. Automatic evaluation methods are here more practically usable than in the first branch. The third section will address a variety of methods, describing, linking and comparing them. Section four gives an usage advice for the presented methods, followed by the fifth and final section, which discusses the subject and gives a brief outlook.

2.1 Cognitive Dimensions in API Evaluation

The user is the center of the evaluation, and it is seen as an advantage to design the API around him. API users are now seen as humans [1, Arnold]. The *cognitive dimensions framework* provides structure for discourse and practice of API user-experience evaluation. Considering Clarke' modifications [4], the framework comprises twelve parts. He modified the original cognitive dimensions framework [8] to use it in context of API user-experience evaluation. As mentioned above Greens framework was build to evaluate the visual environments of computer interfaces. The following list gives a brief presentation of all twelve dimensions:

1. **Abstraction Level:** What are the minimum and maximum levels of Abstraction the API offers, and which segment is appropriate for the targeted user?
2. **Learning Style:** Which learning methods are provided by the API and which ones are useable by the targeted user?
3. **Working Framework:** What size does a working fraction (code snippet) have in which the API is used? Related to the *Work-Step Unit*, Clarke introduced this dimension to describe the size of the context around the code which has to be maintained and kept track of, to work efficiently as developer.
4. **Work-Step Unit:** Clarke introduced this dimension to describe the amount of work a developer has to complete before the code chunk can be referenced elsewhere. The Work-Step achieves a particular step of a working solution,
5. **Progressive Evaluation:** This dimension measures how the design of the API supports evaluation of partially finished tasks. It is essential to provide extensive progressive evaluation to semi-professionals and non-specialists: In contrast to experts they reley on evaluating small chunks of their work to create a solution [8, S.30].
6. **Premature Commitment:** To what extend is the user forced to make decisions before all relevant information for this decision is available?
7. **Penetrability:** How much does the API support notion of it's inner functionality and components (limited access)? Clarke newly introduced this dimension, formerly not available in Greens & Petres framework. The other dimensions mentioned until now are part of the original framework.
8. **API Elaboration:** To what extend does the API needs to be modified to be used by another target developer? Having the growing semi-professionals user group in mind, this section is closely related to the *Abstraction Level*.
9. **API Viscosity:** How much effort does a single change cost?
10. **Consistency:** How much conceptual clarity does the user-experience? After learning a part of the API, how much knowledge can be reused in other API parts?
11. **Role Expressiveness:** How obvious is the relationship between the components and the API on the whole?
12. **Domain Correspondence:** How do API components map to the domain? Do users abuse the API to achieve special goals by using components of the API originally not intended to be used in the domain?

The cognitive dimensions help discussing API user-experience evaluation results and avoid confusion by dividing experience into dimensions. They make it easier to address specific problems and prevent duplication by defining a evaluation vocabulary.

3 Methods

User centric methods put the user into the center of evaluation. They aim to record and monitor the behavior of different personas [10]. Important to mention

is that those personas capture different work styles and not the experience or skills. This section aims to present a traditional lab study at first, followed by multiple team and cognitive dimension based evaluation methods. Automatic tools and API independent design pattern evaluation will be addressed, too.

3.1 Lab study - Think aloud evaluation

The method [13] McLellan, Roesler and Tempest developed at Iowa State University views the programmer as customer in mind and suggests API designers to watch the users, as it would be done in a common end-user selling environment with customers. Their methodology offers a way to evaluate an API at a development stage, where it is still possible to alter important parts for the better.

For the purpose of analysis they started out with *Field Oberservations*. They monitored 20 users for 20-60 min over several days at three different working sites. The rules for selection of the working sides are not described. It can be assumed that it's sufficient to cover the complete range of different domains and divisions in the company. The observations were audio-taped and the amount of gathered information was increased by interviewing the user during the observation.

As next step they organized a *think aloud protocol study* with 4 expert users as test subjects. A typical job scenario was created, covering up to 75% of the API functions. The scenario was implemented linear. Linear means without multiple functions or classes, in order to be easier to understand. Each subject was placed alone in front of a computer and was asked to read and understand the scenario. During the procedure the subject should think aloud and ask questions. The user navigation through the scenario, as well as his/her comments and questions, were recorded. A session had 45min. After the session, the subject answered a user satisfaction questionnaire based directly on Chin's work [3].

This method shows that evaluating APIs with examples can provide helpful designers with helpful information about the usability of an API and its documentation at an early stage of the creation for designers. The feedback helps to verify design decisions and highlights potential improvements. It is generated by real members of the potential user group and therefore valuable and authentic. A disadvantage of this method is that it is is time-, cost- and work-intensive.

The implemented job scenarios can be modified according to the user feedback and used for adding code samples to the API documentation. This work shows exemplary that providing and maintaining example code for APIs is of great importance for usability.

3.2 API Usability Peer Reviews

With API Peer Reviews Farooq and Zirkler [7] describe an API user-experience evaluation method, which overcomes the scalability limitations of e.g. a classical lab study. First, the required personas for this method are defined. Then a brief test record is given and followed by a rating and comparison of this method.

Personas: API Peer Reviews require four personas.

- *Feature owner:* The API author who created the specific API feature, that are under evaluation.
- *Feature area manager:* API author who has in depth knowledge about the API part the feature resides in.
- *Usability Engineer:* The person who evaluates the usability of the API.
- *Reviewers:* The people who provide feedback to the API in form of a peer review. Another API design team would be predestined. They are usually *Feature Owners*, *Feature area Managers* and *Usability Engineers* who are not familiar with the API which shall be evaluated.

Procedure: After those personas are defined, the following procedure is executed to do the peer reviews:

1. In a one-hour meeting, the *Usability Engineer* and the *Feature Owner* discuss these questions:
   - What are the key questions (1-2) to be answered by the evaluation results?
   - Which code sample should be used to review the API feature?
   - Which reviewers should be associated with which sample?
   - When and where are reviewers recruited?
   - When and were are the samples reviewed?
2. The review (1.5 h) is done in a room with all personas present. The *Usability Engineer* points out the goals (5 min) and the *Feature Owner* explains the background of the feature (10-15 min). The remaining time is used to review the feature. In the main part, which is a group-based review process, the *Usability Engineer* moderates the discussion and the *Feature Owner* provides background knowledge.
3. The three roles who are familiar with the API, record the results as bugs in usability again in a one-hour meeting. The effort to categorize those bugs in terms of the cognitive dimensions made by Farooq and Zirkler will not be addressed here.

An advantage of this evaluation method is its redundancy of proceeding, compared to agile software development methods (e.g. SCRUM). Participating personas therefore are more familiar with the API evaluation methods and productivity rises.

Farooq and Zirkler emphasize the low costs and time requirements for this highly scalable and communicative method. Their experience at Microsoft showed that this method is highly beneficial for API usability.
In contrast to other methods, the knowledge about best-practices in API design is spreading in a social collaboration. API Peer Reviews are a way to generate information for guidelines and best practices in order to design usable APIs within an ecosphere.

3.3 Design Pattern evaluation

Evaluation of the user-experience of a specific API mainly improves the evaluated API. Thinking in bigger terms it is even possible to evaluate design choices in general, without having a specific API in mind. Design choices are standardized in design patterns. Given a specific pattern a name and defining it prevents it from being duplicated with another name, but same meaning. Design patterns can be evaluated on a basis of user-experience in order to be compared against an alternative pattern choice for a common situation.
What we know about API design pattern evaluation is largely based upon empirical studies of Stylos and Ellis which investigated how *Requiring Parameters in Objects' Constructors* [15], the *factory pattern in APIs* [6] and method placement [16] influence the usability of an API.
It is not the goal of this section to describe the properties and mechanics that that make an API design "good" and usable, but giving and outline of that topic.

The factory pattern is a creational pattern that allows object creation of an unknown type, which has methods solving problems. Definite subclasses overwrite those methods in a further differentiated case. Comparison of the differences and similarities of the three studies [15,6,16] distills their common characteristics. Independently from the given factory pattern example, following steps have to be taken to evaluate design patterns. After giving a general manual for proceedings the factory pattern example will conclude the section.

1. Recruiting different user personas. The most systematic approach was used in [15] and fragmented the participants in:
   - *Systematic users:* Utilizing the top-down approach, they build understanding for the system as a whole before they focus on details. They don't trust APIs and want a deeper understanding of what is going on, detached from the aim of building a working solution. Stylos recommends searching for users of this defensive persona in hardware near environments.
   - *Pragmatic users:* They are willing to use high-level tools (e.g. Microsoft Visual Studio GUI Builder) to get a working solution in little time. They only revert to the method systematic users apply in case the high level approach fails. Users of this persona utilize SDK driven languages like Java and the Microsoft .NET hemisphere.

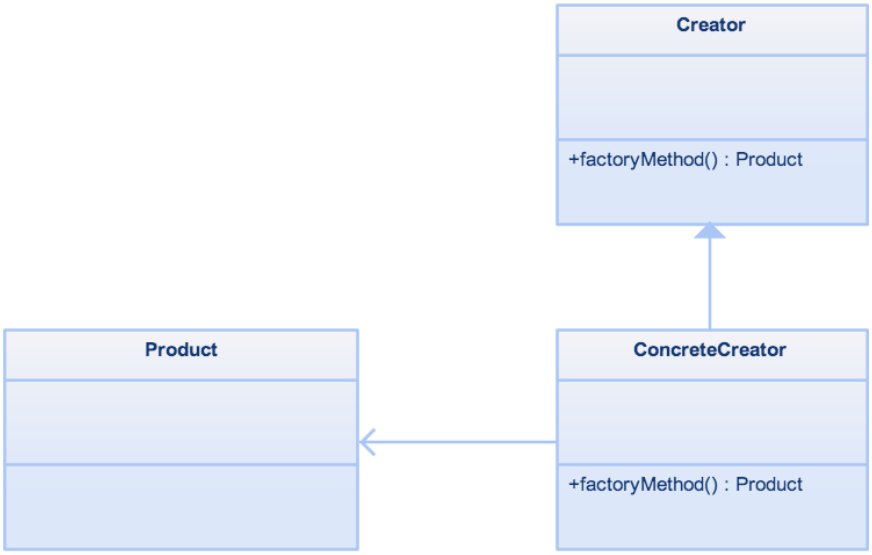


Fig. 1. Minimal factory pattern class UML diagram

  - *Opportunistic users:* Users of this persona don't care about concrete implementations or low-level details. They tend to us all high-level tools and APIs without understanding more than necessary to get a solution.
2. Creation of tasks which include usage of the design patterns that are selected for evaluation.
3. Execution of a think-aloud evaluation as explained in the section above. The time for the tasks and the interaction with the expert can be adjusted and it differs within the three studies.

Taking the factory pattern versus constructor pattern exemplary: API designers profit from usage of the factory design pattern because the factory can be altered or switched without the need to change the applications using that specific part of the API.

The results of the studies however showed clearly that the constructor pattern is more intuitive and leads to better understanding of the API functionality than the usage of the factory pattern does.

3.4 (Automated) Documentation Improvement

Until now this work mainly featured user centered API evaluation approaches. The documentation improvement is the first of several more code centric methods.

The aim of good API design is to create intuitive frameworks that can be understood easily. In fact, many of the already existing APIs are hard to use; the evidence is overwhelming [2,5,15,16]. Many of those existing APIs are public APIs. Almost every popular web company published several ones to enable third-party-developers to integrate their services. Those APIs cannot be changed for obvious reasons of backward compatibility. They are widely integrated in applications and a change would break the compatibility with all clients – quoting Joshua Bloch form Google: *"Public APIs are forever - one chance to get it right"* [2]. To evaluate those APIs on the basis of user-experience it's apparent that the API documentation can be improved.

Stylos and Faulring Yang [17] developed a prototype documentation system which integrates the API usage of other users into the documentation. This makes solving popular tasks easier. Besides they use Google to acquire additional information relevant to the viewed documentation entry. One could of course also use use SVN repositorys or source-code-specific search engines. However, none of those alternatives to Google are able to provide that extensive and representative code chunks. Their system, called *Jadeite*, provides an hierarchical browsing interface which is identical to the structure featured by most APIs. In contrast to other atomatic documentation systems e.g. *Jungloids* [12] all information are intended to be rather an example than an abstract description and rather popularly used than complete.

The integration of usage data into the documentation was achieved through following two features:

Function placeholders reference: Usually the API documentation lists the functions and classes of the API, following the inherent hierarchical structure of the API. But in case that users expect to find a specific function on a place in the API where it dosen't reside, workflow is slowed down remarkably and usability deteriotates. On the basis of usage information the documentation is extended in those cases with placeholders for the expected functionality. *Placeholder references* link to the expected function in another part of the API.
The pool of placeholders is altered and extended collaboratively by the members of the target user group and synchronized within the user group.

Font sizing: This method increases the font size of popular parts and functions of the API. Letting the font size reflect the popularity of an API object saves time for the user. It can be seen intuitively which objects are used seldom and which are popular. This speeds up finding solutions for common tasks. One could critically argue that this makes finding solutions for special requirements more difficult, because the user overestimates the popularity of his problem and therefore searches in popular classes.
To calculate the font-size of the objects, a frequency analysis is created. Hits of the specific object on Google are taken as reference value for popularity.

Depending on those values API popularity is mapped to a font size. For very popular APIs e.g. Java API, a logarithmic weighting is practically applicable.

3.5 Technical writers

Technical writers are professionals who produce documentation for technical products, in this case for APIs. They are trained to have the knowledge context of the target user group in mind while writing. This makes them helpful for evaluating user-experience in interaction with an API design team.
The documentation of an API can be written at any time in the API life-cycle, but the documentation usability and quality is influenced by the stage at which it is created [18].

- *Design stage:* The technical writer can help reflecting design choice from the perspective of a user. This is the best stage to consult or integrate a technical writer in the process of API creation, thinking in perspective of usability.
- *Implementation stage:* Inconsistent design choices often make it complicated to write an understandable documentation. This problem could have been avoided by integrating the technical writer earlier to the benefit from "the users advocate" to reveal and resolve design problems.
- *After release:* In the worst case, the technical writer may have serious problems to use his skills in favor of a better API usability.

No matter in which stage, the application of the following method by a technical writer helps to improve API usability. Regarding the evaluation method: The documentation is considered as important, but "learning by doing" is essential for users. The API itself is the first "documentation" the user is faced with. Watson [18] states that naming choices of features and API parts can be improved with the advice of a technical writer.
Categorizing this study ([18]) in terms of the cognitive dimensions, only *Consistency* applies to it.

The biggest effort to analyze the API for consistency is the data preparation. The whole API source code has to be converted to a list-format containing all interfaces, methods, structures and enumerators. Watson used over 900 additional parameters (e.g. *Method Name, Normalized Method name, Parameter Name, Parameter Data Type*) to analyze consistency. As data-structure he chose an XML based content management system to reduce the overhead.
After data preparation, the results are imported into Microsoft Excel and with filtering and sorting, the consistency analysis is done. Following inspection parameters are described by Watson [18].

- Parameter Names and Data Types
- Parameter Name Matches Data Type
- Interface and Method Name Analysis
- ...

The inconsistencies that are found are reported to the design team. If the inconsistency is intentional, this can be emphasized in the documentation for better understanding out of the user perspective, otherwise the inconsistency is resolved which in turn improves the API usability.

3.6 Automatic tools using metrics and visualizations

The investigation of several code centric methods, which use complex metrics, visualizations or other automatic approaches show:
Many of the papers presenting automatic API evaluation methods feature a tool which is build by the authors helping to reveal or avoid bad API design habits. Neither screenshots nor statistics of tool application on a specific source code give reasonable approaches to evaluate API user-experience. Nevertheless it's of importance to mention that automatic tools exist. The following list gives examples to get a brief overview over some of the popular tools, if needed.

Mica - A Web-Search Tool for Finding API Components and Examples: Mica (Making Interfaces Clear and Accessible) is a prototype to help searching the web to learn how to use APIs. It displays API methods in filtered and ordered manner, based on the frequency of the usage and correlation with other search queries.

Jadeite - Improving API Documentation Using Usage Information: Jadeite is a documentation system making it easier for users to learn a new API. Further information on the method can be obtained in the paragraph 3.4.

Apatite - Associative Perusal of APIs That Identifies Targets Easily: This is a tool to learn and explore the Java 6 API. It helps to find methods, classes and structures as well as it aggregates information extracted from the documentation.

Metrix - Automatic calculation of Bandi's metrics: Empirical studies that examine how design pattern choices influence usability and maintainability (e.g. [5]) are very powerful but as expensive as a traditional user-experience lab study [14]. In contrast to this effort, the work of de Souza [**?**, 20]trives to give immediate feedback to users and creators. This feedback informs the user about the complexity of the API that is currently attached to the tool. Aiming to assist the user in understanding the API, their tool provides visualization of API complexity.
To categorize this work in terms of the cognitive dimensions, an additional *Learning Style* is provided and the *Penetrability* is increased.

4 Usage Advice

Many APIs have the potential to be improved concerning their usability. When this is recognized, next steps can be taken. Most methods presented here gather data from users. Doing this in a early stage of the creation can help to resolve many problems before the API is released.

Design choices which make a good architecture often result in bad usability. Knowing this can help to evaluate the design as well as to find a good compromise between clean architecture and usability.
To evaluate the user-experience of already released APIs, methods of documentation improvement exist. Furthermore, users can be assisted with better IDEs. Stylos's work [5,15,16,17,6] tries to remove the boundaries between API documentations and IDEs. Including programming samples in the documentation is expected to be highly beneficial for the learning curve of the user and it's worth adding it, even if the API is released and in a later part of its life cycle. Until now programming tools and API documentation don't integrate usage information to present popular and important parts in prominent form. The tools presented above are prototypes delivering this functionality. Taken together, these studies suggest that usability profits from enriching the documentation.
Farooq and Zirkler state that formal, empirical usability evaluation methods overlook many problems [7]. Lab studies based on the cognitive dimension framework overcome those shortcomings, but are expensive and don't scale well [17]. Table 1 gives overview of the presented methods gives qualitative advice on usage based on the recommendations and indications that the authors grant. The categories are valued from - - - (worst) to +++ (best). To compare two methods, a property has to be chosen. Than, in its line, the methods can be compared against each other.

	LS	PR	DPE	ADI	TW
Before implementation	- - -	+	+ + +	- - -	+ + +
During implementation	+ + +	+ + +	-	+	+ +
After publishing	+ +	- -	- - -	+ + +	-
Human ressources	- - -	+ +	- -	+ + +	+ +
Time consumption	- -	+ +	- - -	+ +	+ +
Scalability	- - -	+ +		+ + +	+
Feedback relevance	+ + +	+ +	- -	-	+
Transferability of results	- -	+ +	+ + +	- - -	- -

Table 1. Qualitative method comparison of relevant usage parameters

LS = Lab studies
PR = Peer Reviews
DPE = Design pattern evaluation
ADI = (Automated) Documentation improvement
TW = Technical writers

5 Discussion and outlook

It is possible to evaluate the API user-experience of an abstract design pattern like it was done with the Factory Pattern [5]. This shows that API user-experience evaluation is not only an engineering skill which API designers have to learn to create good APIs. It's an open field for research on complex metrics and informal guidelines.

After the API is released, its documentation can be improved by adding code examples. The aim of those samples is to give solutions to common and popular tasks in the domain of the API. The manual way is adding editorial code chunks which are searched and selected by an API expert. The automatic way is only possible if the API comes with an IDE that the target users have installed. For this purpose an automated system has to be created which integrates with the documentation system of the IDE. Now it could fetch information from the internet, or retrieve editorial content which is created by the author of the API.
It can be imagined to enrich API documentation not only by manually created placeholders. Placeholders could be created automatically by matching API calls from big semantic distance in the API, but small distance in relevant code chunks by an algorithm.

Based on the results of the usability evaluation on design patterns [6], their evaluation in general is a valid method to improve API design. Unfortunately there is a lack of publications in this field of research. A lot of important design pattern choices are not covered by professional usability evaluation. The explanation that companies rarely finance such research on their own comes up easily. No specific API benefits directly from such research without further investment. Therefore, it's not a lucrative way to improve an existing API, rather something that should be done before implementing the API. Greenberg and Buxton came to the conclusion, that API Peer Reviews and API lab tests should be used both to evaluate API user-experience at different points in a API life-cycle [9]. A reasonable approach to tackle this issue could be the usage of Peer Reviews at earlier stages to get an external opinion on design choices. Lab studies with real users are more useful in late stages.
There are many best-practice and guideline type papers as well as literature which addresses a wide range of issues with formal and informal API evaluation methods. Those where not addressed here. On the other hand the current literature on API user-experience evaluation has a big shortcoming: There is no widespread quantitative evaluation of API usability with general purpose metrics.

API user-experience evaluation is as a task of growing importance. Users as well as creators are still in demand of general purpose tools and methods to improve usability. The field of API user-experience evaluation is of growing significance and the cited work shows, that research in this field is rewarding and important.

References

1. Arnold, K.: Programmers are people, too. Queue 3(5), 54–59 (Jun 2005), http://doi.acm.org/10.1145/1071713.1071731
2. Bloch, J.: How to design a good api and why it matters. In: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications. pp. 506–507. OOPSLA '06, ACM, New York, NY, USA (2006), http://doi.acm.org/10.1145/1176617.1176622
3. Chin, J.P., Diehl, V.A., Norman, K.L.: Development of an instrument measuring user satisfaction of the human-computer interface. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 213–218. CHI '88, ACM, New York, NY, USA (1988), http://doi.acm.org/10.1145/57167.57203
4. Clarke, S., Becker, C.: Using the cognitive dimensions framework to evaluate the usability of a class library. In: In Proceedings of the 15h Workshop of the Psychology of Programming Interest Group (PPIG 2003 (2003)
5. Ellis, B., Stylos, J., Myers, B.: The factory pattern in api design: A usability evaluation (2007)
6. Ellis, B., Stylos, J., Myers, B.: The factory pattern in api design: A usability evaluation. In: Proceedings of the 29th International Conference on Software Engineering. pp. 302–312. ICSE '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/ICSE.2007.85
7. Farooq, U., Welicki, L., Zirkler, D.: Api usability peer reviews: A method for evaluating the usability of application programming interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2327–2336. CHI '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1753326.1753677
8. Green, T.R.G., Petre, M.: Usability analysis of visual programming environments: a 'cognitive dimensions' framework. JOURNAL OF VISUAL LANGUAGES AND COMPUTING 7, 131–174 (1996)
9. Greenberg, S., Buxton, B.: Usability evaluation considered harmful (some of the time). In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 111–120. CHI '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1357054.1357074
10. Grudin, J., Pruitt, J.: Participatory design and product development: An infrastructure for engagement. In: Proc. PDC 2002. pp. 144–161 (2002)
11. Henning, M.: Api design matters. Queue 5(4), 24–36 (May 2007), http://doi.acm.org/10.1145/1255421.1255422
12. Mandelin, D., Xu, L., Bodík, R., Kimelman, D.: Jungloid mining: Helping to navigate the api jungle. SIGPLAN Not. 40(6), 48–61 (Jun 2005), http://doi.acm.org/10.1145/1064978.1065018
13. McLellan, S.G., Roesler, A.W., Tempest, J.T., Spinuzzi, C.I.: Building more usable apis. IEEE Softw. 15(3), 78–86 (May 1998), http://dx.doi.org/10.1109/52.676963
14. de Souza, C., Bentolila, D.: Automatic evaluation of api usability using complexity metrics and visualizations. In: Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on. pp. 299–302 (May 2009)
15. Stylos, J., Clarke, S.: Usability implications of requiring parameters in objects' constructors. In: Proceedings of the 29th International Conference on Software Engineering. pp. 529–539. ICSE '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/ICSE.2007.92

16. Stylos, J., Myers, B.A.: The implications of method placement on api learnability. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 105–112. SIGSOFT '08/FSE-16, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1453101.1453117
17. Stylos, J., Myers, B.A., Yang, Z.: Jadeite: Improving api documentation using usage information. In: CHI '09 Extended Abstracts on Human Factors in Computing Systems. pp. 4429–4434. CHI EA '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1520340.1520678
18. Watson, R.: Improving software api usability through text analysis: A case study. In: Professional Communication Conference, 2009. IPCC 2009. IEEE International. pp. 1–7 (July 2009)

Continuous Integration for mobile devices

Enes Erdogan*

Advisor: Dr. Till Riedel†

Karlsruhe Institute of Technology (KIT)
Pervasive Computing Systems – TECO
*uhdgl@student.kit.edu
†riedel@teco.edu

Abstract. In today's world, mobile applications and devices are blooming. Testing and development of mobile software are increasing to demand for higher quality, scalability and stability, with a decreasing request for costs and time. Different approaches have been made to overcome several problems caused by the vast amount of combinations between devices and operating systems and the rapid growth of industry. This paper presents and discusses different, classified issues and solutions provided by the use of continuous integration.

Keywords: Continuous Integration, automated testing, mobile devices

1 Introduction

The main problem considering the development of mobile applications is the huge amount of variations between hardware and software and the difficulties of being able to find reliable tests which reflect the reality outcome by comparison to the exptected.

Of course, developers want the results of their work to be of high-quality, well tested, bugless, platform independent and easy to maintain. On the other hand, the costs should be minimized; the more time a project needs, the more money has to be paid at the end.

The obstacles caused by the problems listed above make the demands of perfection seemingly impossible. Testing is nearly impossible, because the incredible amount of testers required to accomplish a reliable report would be immense; the overhead caused by version upgrades, bugfixing and maintaining is multiplied by the activity of testing. In addition to this, each bugfix and version upgrade may imply another bug. Features that used to work now return an unexpected behavior, and these regressions have to be found, fixed (which, again, can lead to another regression) and tested again. Thus, other solutions are required.

Therefore, this paper discusses different approaches that were made to overcome various difficulties. First of all, in **section 2**, a small motivation of why the use of continuous integration is propably a huge step towards solving many of the problems that are being classified and discussed in **section 3**. **Section 4** presents a few tools which use different methods of bridging the difficulties in

software engineering for mobile devices, and finally, in **section 5**, a conclusion will summarize the results.

2 Motivation

To give a small peek into what the solutions regarding the problems of mobile devices are, the quote below will expose the main practice:

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible." - Martin Fowler [10]

This quote precisely sums up the technique which is used by many approaches that were made to overcome most of the issues regarding development of mobile applications. Since continuous integration (CI) has alot of benefits which ideally fill the gap of disadvantages, it provides a huge simplification to the wall of problems.

Any and all code deployment and integration issues are fixed much earlier in the process of development as well as the creation of mobile applications become repeatable and reliable. This increases the motivation of team members, reducing frustration and raising productivity by seeing how the application is progressing day to day.[14]

The use of CI reduces software flaws simply by immediately testing the code after compiling. This implies predictible progress because bugfixing and re-testing is continuously performed, making quick advancement to a standard experience. By removing bugs much earlier in the process of development compared to the "classical" way (planning, implementing, debugging), the whole project becomes a lot cheaper and less time is being needed to accomplish a task. This decreases the amount of debugging drastically and gives the team the ability to react much quicker to changes in the underlying hardware, which is the main challenge regarding mobile devices. The concerns can be seperated and easily pinpointed, and thus quickly removed, without taking them to further steps of development. Yet, integration and unit tests can be run on target hardware, cross-compiled, run on the development machine or also run in a simulator, which offer a huge chance to test the code on different platforms and circumstances, eliminating bugs because of hardware and software, or the combination of both.

Due to the predictable progress, reliability and stability of the code, priorities, budgets, timelines and features can be easily adjusted as needed.[11]

As it is not hard to see, using CI brings along alot of benefits. But before any of them can be applied, it is required to see the exact problems developers of mobile applications are encountered with in the process of development. This

paper classifies various problems which have been restraining mobile software from reaching the maximum of quality and flawlessness.

3 Problem classification

In this section, different classes of problems regarding the development of mobile applications will be presented. Many different problems can occur during the progress, and the assurance of quality is hard to achieve because testing the applications in today's amount of different devices seems pretty hard.

3.1 Platform Dependency

The biggest problem by far is the vast amount of hardware/software combinations. Many different behaviors occur between the same software version on only a different device.

For example, there is the battery issue on Android 4.4.2. Many customers have been complaining about a really high battery usage (40-%-80%) after Google has released a new version for the camera Software [9]. But not all users have issued the same problem. Especially users of the mobile phone Google Nexus 5 and Samsung Note 3 have reported the bug, and after confirmation, users were told to ask the manufacturer of their own device. How comes that especially these gadgets were suffering from the software upgrade way more than others? Let's take a deeper look inside the Android architecture:

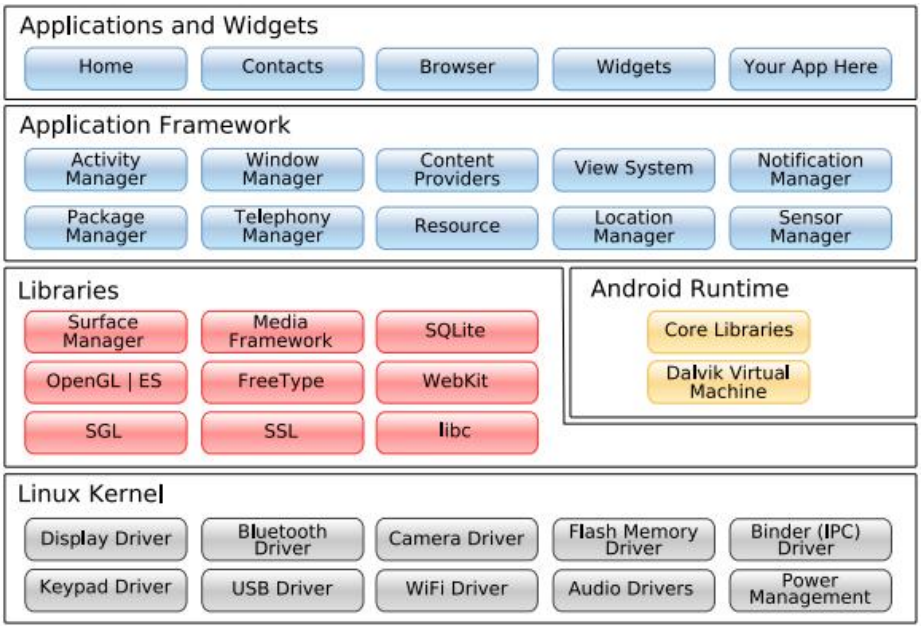


Fig. 1. The Android Software Architecture [1]

The architecture shows the different layers of the OS. The kernel is next to the hardware, the libraries next, and it goes all up to the UI. Many manufacturers modify the original version of Android and adapt it to each device. These modifications are called Stock-Firmware. Anything inside the architecture can be changed, be it the libraries or the kernel itself, just how the manufacturer desires. Matt Porter criticizes Android being a "screwed, hard-coded, non-portable abomination"[15]. These different versions of firmware provide different possibilities for unexptected behaviour to appear. If a newer version of Android is finally released, the new features are available for every device, including, for example, the camera upgrade mentioned above. These features may not be suitable with the already existing firmware on the current devices, leading to unexpected or bad behaviour.

This is just a small example of how bugs can appear. Now, it can be guessed that the upgrade was not tested properly in terms of long time behavior and platform dependency. But apart from this example, many more errors can occur between different software versions; there are enough bugs existing on the same device using different versions of firmware (which on the other hand work perfectly well on other devices).

So there are alot of possibilites of combinations between hardware and software to contain errors. Solely Android supports more than 500 different devices[2], many different Android versions and different mods. Additionally, there are even more devices and operating systems next to Android, e.g. Apple with their IPhone, and Microsoft supporting their Windows Phones, and many more. The amount of different platforms and software is immense, and you can only guess what inconstistency is existing.

Regarding other problems and problem classes, keep in mind that everything occurs in addition to the problem of the platform dependency. Any fix and upgrade needs to be seperately done on each version and on each device.

3.2 Regressions

As a code grows and new features are added step by step, programmers are cautious not to implement any buggy and not-working methods. New features should, in the best case, immediately work flawlessly, without causing any trouble or the need of extra bugfixing. However, no matter how carefully the new code is being implemented, the risk of an already existing code not working as expected after modifying always persists. Chang et al. refer to these regressions as follows:

"When a new feature is implemented, in addition to verifying whether the implementation is correct, it is equally important to ensure that it does not break any existing feature that used to be working. This practice is often known as regression testing in software engineering" - **Tsung-Hsiang Chang, Tom Yeh, Robert C. Miller**[7]

The difficulty programmers are confronted with when dealing with regressions is the fact that they are, compared to other kinds of bugs, hard to find. The

consequences of adding something new are not always obvious, and depending on the frequency of commits and changes, the amount of potential regressions grow.[8]

Since the use of CI exceeds common programming techniques considering changes to the code and improving quality by continuously working on it, there is a high chance of unexpected alteration of the functions which used to work and can now be very hard to reproduce and troubleshoot. A way of automated testing is needed.

Luckily, testing and CI are strongly involved within each other. After each commit and integration, the new build is being tested automatedly. This increases the amount of tests proportionally to the amount of potential regressions, and finally reduces the risk of breaking something functional, since a good, automated test is supposed to test all functionality, including the influence of the alternation on the already existing code.[13]

The advantage of using CI compared to regular programming (= different phases of programming: planning, designing, implementing, debugging, testing..) is minimizing the gap between the original version and altered version. This also minimizes the amount of tests required to scan for regressions, compared to regular programming: After the implementing phase, programmers stand in front of a huge amount of untested, propably malworking code which needs to be debugged step by step. The phase of bugfixing takes a lot of time, and does not guarantee that the new changes which are meant to improve the quality do the exact opposite of what they were supposed to: Some fixes are more complex and need to "optimize" already existing code, influencing the functionality of an already working segment and propably creating new bugs, which again have to be removed by testing and more bugfixing, and so on. Newer features could have an unexpectedly high amount of impact on the application. Additionally, the time for testing and debugging tends to be insufficient and therefore, regressions can increase the final cost and completion time of a project.

This paper discusses different approaches towards the solution of regressions on mobile devices, on which, again, the high amount of combinations between hardware and software offers even more chances for hard-to-test bugs to appear.

3.3 GUI

A special problem regarding automated testing is the difficulty of testing patterns on the top level of programming: The user interface. Testers of non-GUI applications have the ability to automate their tests by writing a short unit test which is supposed to test functionality and filter unexpected output. Chang et al. use an example like follows:

For instance, a method called `addOne()` needs to be tested. This can easily be done by writing a small script which invokes this method and is combined with an assertion, i.e.

```
assert(addOne(3) == 4);.
```

This short test now checks if the function correctly returns what is expected; in this case, the return value should be "4", and if it's not correct, an error is to be reported. The advantage of this of testing is the possibility for the tester to re-run the code automatically as often as it is desired. This reduces the effort for the tester, compared to the common way of GUI testing: How is a code supposed to test something, which does not provide internal functionality, but a visual change to the outside? Since testing the same way like presented above does not work, the authors give another example on how tests on the user interface are supposed to work: To give a small example, let's say the interface of a music player needs to be working fine and therefore has to be tested. After clicking the "play" button, it should become a "pause" button, just like this:

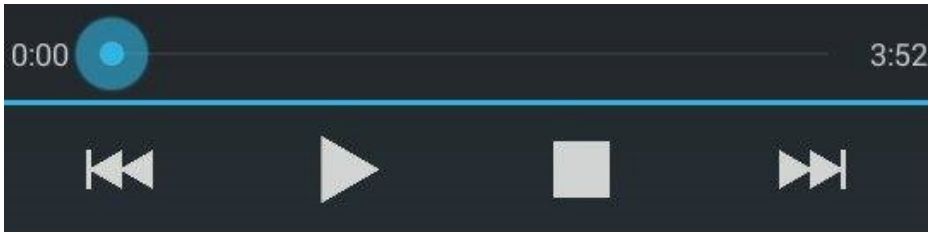


Fig. 2. Music player before pressing the "play" button.

After klicking the button, the music player should look like this:

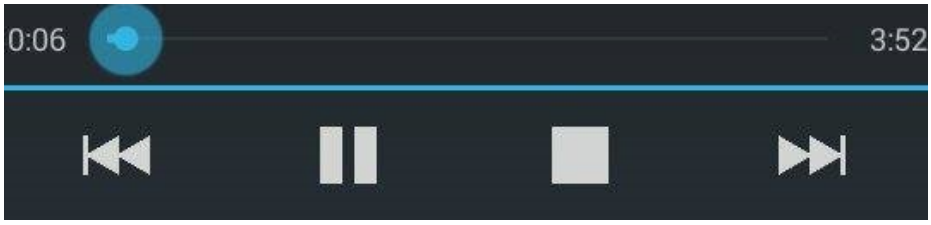


Fig. 3. Music player after pressing the "play" button.

In the common way, first, a quality assurance tester has to *look* for the play button, manually *click* the button, and finally *see* if it successfully changed. This procedure has to be repeated again, every time the code is built, to prevent regressions.[7]

The importance of GUI testing for mobile devices is again motivated by the high variety of different hard- and software, and therefore needs special treatment.

4 Tools

Considering the problems connected to mobile computing, several tools and programs were modified or created just to bridge the difficulties of device fragmentation and testability. This paper will give a small overview on some of the tools and programs using the practice of Continuous Integration and which are considered helpful.

First of all, this paper will present the *Mobile-D approach*, which has the aim of finishing a project as soon as possible after it was started, which means it intends to deliver a fully functional mobile application in a very short time.

In addition to the Mobile-D approach, research about an adaption of *Jenkins*, which is a well-known tool for Continuous Integration, has been made and its advantages will be presented in this paper. The tool is called *JenkinsMobi*, one of many plugins available for Jenkins. For mobile development purposes, this plugin provides an API for mobile devices.

The last tool presented by this paper is the *Xamarin Test Cloud*, which, combined with another tool called *Calabash*, offers a solution to the majority of problems related to mobile development and testing.

4.1 The Mobile-D Approach

The Mobile-D Approach has been developped to overcome the challenges involved in mobile development. It is optimized for a team of less than ten developers, which has the target of delivering a perfectly functioning mobile application in a small timeframe, for example, in a span less than ten weeks. In detail, this works as follows:

A project is divided into five iterations: Set-Up, core, core2, stabilize, and wrap-up. Each of these phases, again, is divided into different types of devlopment days: Planning Day, Working Day, and Release Day. With different teams working on different parts of the project, an integration day is also needed. These different phases represent nine, mostly well-known agile practices:

1. Phasing and Pacing
2. Architecture Line
3. Mobile Test-Driven Development
4. Continuous Integration
5. Pair Programming
6. Metrics
7. Agile Software Process Improvement
8. Off-Site Customer
9. User-Centered Focus

The development is required to be testing oriented to ensure flawless operation of this method on multiple mobile platforms and increase software quality. Mobile-D involves automating unit tests and acceptance tests with the customer by writing test cases before the actual implementation.

The Mobile-D Method was tested and further developed by the Technical Research Centre of Finland. Increased progress visibility, early identification and solving of technical problems, efficient information sharing, high process-practice coherence, low defect density and a constant development rhythm could be observed.[6]

Since the observations are representative for the advantages of using Continuous Integration, the Mobile-D approach is a good demonstration of how CI can enhance the development of mobile applications. In a short time, a fully functional application is delivered. Because the application is on its last version, any bugs and other problematics should have been removed on various different platforms until the end of the timespan; therefore, the Mobile-D offers a fine solution to the problematics of mobile computing.

4.2 JenkinsMobi

For the use of Continuous Integration, the mentioning of one tool is unavoidable: Jenkins.

Jenkins is an open source CI tool which provides CI services for software development. It is a server-based system supporting Revision Control, e.g. GitHub, Subversion, CVS etc. The application monitors execution of repeated jobs and automates builds continuously by building in different situations.

For example, a simple commit in a version control system or the completion of other builds can trigger another automated build, running and testing all the changes that have been made from the last test. This offers a continuous way of rebuilding and testing the code, making it easy for the developer to detect and fix any misbehaviour he came upon.

The reason why Jenkins could also be a helpful tool regarding mobile computing is the fact that there are many plugins available. Various extensions and add-ons help Jenkins become a very powerful tool. Since there are over 300 plugins available, there happen to be one available for mobile devices, called *JenkinsMobi*.

JenkinsMobi is a plugin for Jenkins, making it available for mobile devices (such as Android, iOS). According to the official blog, it serves 4 goals:

Goal #1: Improved stability

The application is supposed to be slim and easily adaptable and portable to other mobile OSes, increasing the platform independency of the actual device. Further, a targeted, small amount of code should reduce the use of valuable memory and battery life and also reduce the amount of bugs.

Goal #2: Ease of Use

JenkinsMobi is not a complicated tool to use. Entering one URL and the user's credentials is sufficient to access all services from the server.

Goal #3: Performance

Content is cashed locally on the device, allowing the user to browse even when offline. Upon going online, the differences to the new builds are being downloaded.

Goal #4: API and Plugin Integration

Since Jenkins is already open to extension and plugins, also JenkinsMobi can profit from features and add-ons using Plugin APIs. For example, multiple authentications are being supported (i.e. GitHub authentication).

JenkinsMobi offers support for the use of CI during development, helping the developer to automate and test their builds, recording and publishing the results automatedly. This can be a great help in continuous debugging and increasing quality on mobile devices.

However, the setup and practice in a wide-spread term (like on many different devices) can be time-consuming and is not the most comfortable way of testing cross-platform, since the application has to be downloaded on each device, set-up and test results have to be viewed manually. [4]

4.3 Xamarin Test Cloud

Testing software simultanieously on different devices can be a hard task to accomplish. Since each device may react differently on software, and the result may differ depending on the OS version etc., cross-platform testing is required to guarantee a high quality result. The Xamarin Test Cloud is a tool specially designed to overcome the problems of widespread hardware-software combinations. Basically, the idea of the Xamarin Test Cloud is to test mobile software directly on the target devices. This means that the cloud provides a variety of hundreds of different, physical devices on which the software is going to be tested on. The developer can decide on which physical devices exactly his application is going to be tested. This saves the need for the developer to simulate different devices of look for a possibility of testing on other physical devices. The idea of the Xamarin Test Cloud works as follows:

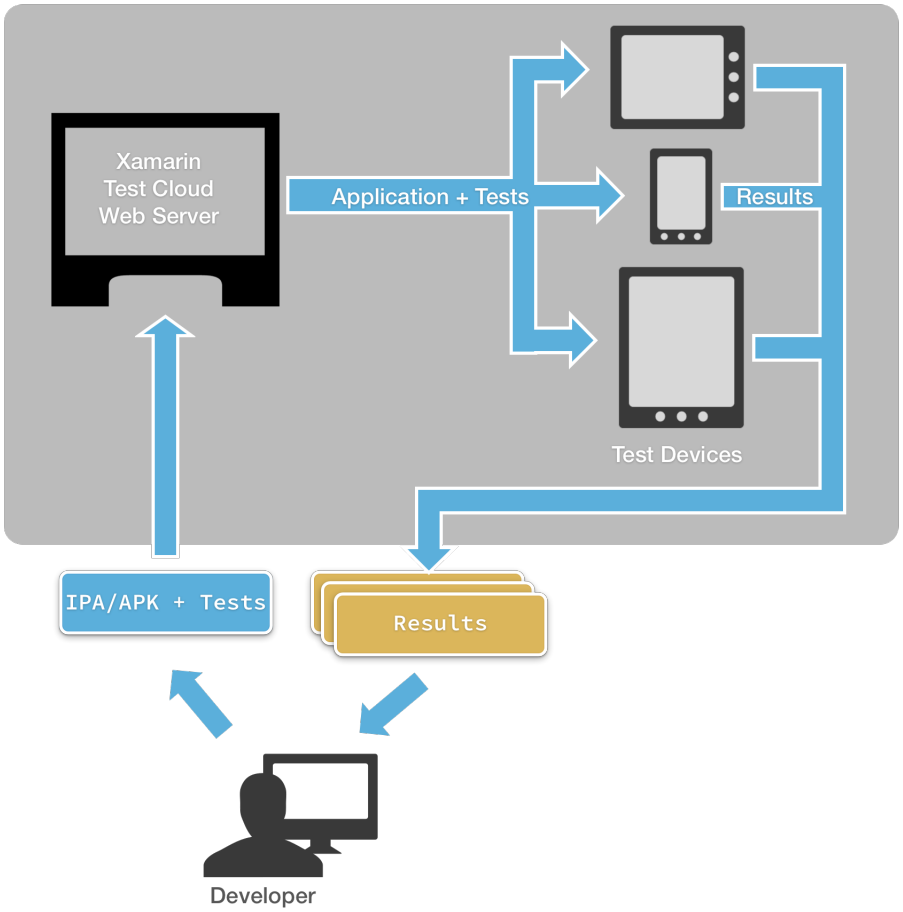


Fig. 4. Img. 4.3.1: Workflow of the Xamarin Test Cloud[5]

The Image illustrates how testing on the cloud works. First of all, the developer uploads his application and test-cases into the Xamarin Test Cloud. The application and tests then are being downloaded to different physical devices which are running inside the labs of Xamarin, running, testing and logging the application. The test results finally are sent back to the developer, so that he may see and react to the outcome, start bugfixing, etc. However, this practice still may require more work and time than other Continuous Integration practices. Because of that, Xamarin provides another way of using the test-cloud, combining it with CI and automated testing. This also allows the use of Calabash, a tool which gives the possibility to interact directly with the UI of the application, making it possible to create reliable UI tests.

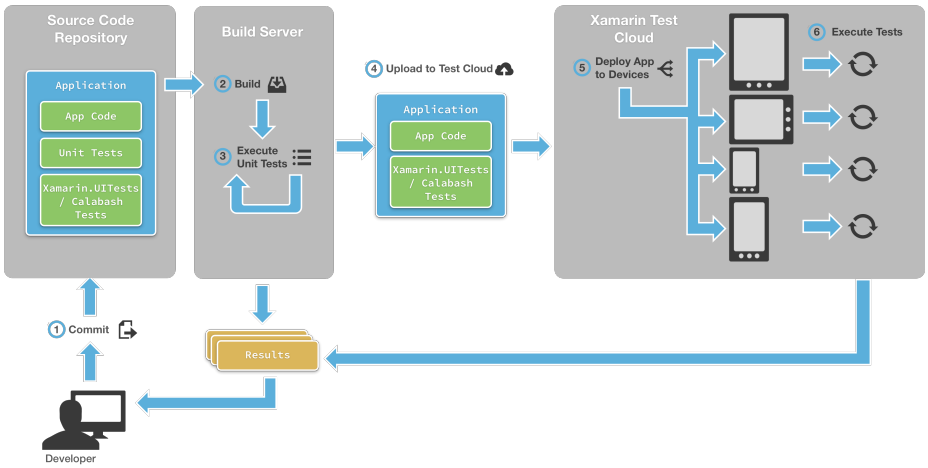


Fig. 5. Workflow of the Xamarin Test Cloud (automated)[5]

The user may decide whether to invoke the testing manually by uploading the application and tests to the Xamarin Test Cloud, or integrating it as a part of CI workflow. As image 4.3.2 shows, a commit of the user is enough to trigger the build server, execute unit tests, upload to the Xamarin Test Cloud, and repeat the procedure described above. This way of Continuous Integration offers a solution to permanently testing an application simultaneously on different devices, making it almost impossible for bugs and regressions caused by platform dependancy to appear as result. The use of Calabash additionally enhances the way of UI testing, offering ways to interact with the UI of the application, for example by simulating touch gestures, swipe gestures or rotating the device.[3] This does not work by image recognition, but by automatically navigating through the application, visiting every screen and pressing every button and interacting with every UI element, identifying and accessing every element by ObjectIds. With this method, even changes to layout etc. will not affect the correctness of the test results.[12]

5 Conclusion

Continuous Integration offers a solution for automated testing and increasing the quality of mobile applications. The problems developers of mobile applications are confronted with are mainly caused by the high fragmentation of mobile devices, which means a high variety of different operating systems running on different physical devices. However, this is not the only kind of problem mobile developers have to deal with. Regressions may be a very big challenge, since detecting the errors is difficult; and while these bugs develop with the time the project grows, a proper reaction to the regression can be too late and in the end a time consuming task. Another problem is the difficulty to test the User Interface

of hundreds of different mobile devices, since it is hard to replace QA-testers by automedly testing code. These problems, however, are not unbridgable. Different approaches and tools have been proved useful, specializing in the development of mobile applications. In this paper, three approaches and tools have been presented, and even though it is hard to have a direct comparison between the results, each of the tools and approaches have their own way of solving most of the problems regarding mobile computing. The Mobile-D approach focuses on starting and finishing a mobile application in a very short time, causing the developers to have an overview over anything done in the time period. The project is to be delivered fully functional, which requires the developers to fix any bugs immediatley and run tests all the time to ensure product quality. This approach leads to a solution to any kind of problem, since in the time period, the project has to be finished perfectly. Another way of overcoming the problems with mobile computing is using CI tools like JenkinsMobi, which is a plugin for Jenkins, making the use of CI possible on mobile devices. The last tool discussed in this paper is the Xamarin Test Cloud, a possibility to test a developer's application on hundreds of different physical devices automatedly, by uploading the application into the cloud, where it is downloaded on other devices provided by Xamarin, ran and tested, and test results sent back to the developer, who then can react to the results.

Continuous Integration is a practice to improve software quality by testing in a permanent way. The use of CI on mobile devices is a nice solution to the problems caused by the vast amount of different devices, since testing is the way of creating a flawless application.

Bibliography

[1] http://kebomix.wordpress.com/.

[2] http://nomtek.com/continuous-integration-ci-for-mobile-devices/.

[3] *Calabash*. http://calaba.sh.

[4] *Jenkins ci*. http://jenkins-ci.org/.

[5] *Xamarin - introduction to test cloud*. http://developer.xamarin.com/guides/testcloud/introduction-to-test-cloud/.

[6] P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M. Korkala, J. Koskela, P. Kyllönen, and O. Salo, *Mobile d: An agile approach for mobile application development*, Vancouver, British Columbia, Canada, October 2011.

[7] T.-H. Chang, T. Yeh, and R. C. Miller, *Gui testing using computer vision*, Atlanta, GA, USA, April 2010.

[8] N. daVeiga, *Change code without fear*. http://www.drdobbs.com/tools/change-code-without-fear/206105233.

[9] A. Floemer, *Android 4.4.2 kamera bug: Google arbeitet an bugfix, empfielt skype testweise zu deinstallieren*, March 2014.

[10] M. Fowler, *Continuous integration*, May 2006.

[11] M. Karlesky, G. Williams, W. Bereza, and M. Flechter, *Mocking the embedded world: Test-driven development,continuous integration, and design patterns*, Embedded Systems Conference Silicon Valley, San Jose, California, April 2007.

[12] F. Lardinois, *Xamarin launches test cloud automated mobile ui testing platform, acquires mobile test company lesspainful*, (2013).

[13] Z. Moćkun, *Automated tests in continuous integration environment*, August 2011. http://www.cognifide.com/blogs/quality-assurance/automated-tests-continous-integration-environment-1/.

[14] G. Nolan, O. Cinarm, and D. Truxall, *Agile android*, (2013). http://link.springer.com/chapter/10.1007/978-1-4302-5858-2_4.

[15] M. Porter, *Android mythbusters*, (2009). http://laforge.gnumonks.org/weblog/2009/11/04/#20091104-android_mythbusters.

The Power and Overhead of Java Dynamic Software Updating Systems

Tobias Schwarz*

Advisor: Martin Alexander Neumann†

Karlsruhe Institute of Technology (KIT)
Pervasive Computing Systems – TECO
*tobias.schwarz@student.kit.edu
†mneumann@teco.edu

Abstract. Dynamic Software Updating systems are interesting area. The paper starts off with the problems of DSU systems in general. Then it compares four different Java DSU systems: DCE VM, JRebel, Gosh! and Rubah. At the comparison we looking especially on two properties: functionality (ability to change the code) and overhead (how much runtime the functionality costs).

Keywords: Dynamic Software Updating, Java, virtual machine, class hierarchy, runtime evolution, DCE VM, JRebel, Gosh!, Rubah

1 Introduction

In this seminar paper we will look at so called Dynamic Software Updating (DSU) systems. Central to the entire discipline of DSU is: the concept of updating a running program without it terminating. Dynamic Updates have the advantage that you can update a program with the same state. However, for example security reasons you need to update them. You don't need to reload the program data. This helps high availability systems a lot, because you don't want to shutdown them even a minute. Another advantage is possibility to modify a program independent form the software architecture. End user has the gain of quicker update times and less effort. In this work we look exclusive at DSUs that are implemented in the Java Programming Language.

Recent developments of new Java DSU systems (such as Gosh![3,4,6,8,7] and Rubah[18,19]) have heightened the need for a survey paper. These state of the art systems have not yet been compared to each other in academic publications. In comparing the Dynamic Code Evolution Virtual Machine (DCE VM) and JRebel, the developers of Gosh noted "Gosh! is currently the most comprehensive DSU system available" [6]. Rubah is a system which uses the theory behind Kitsune. The developers of Rubah wrote in their paper, that it is "the first full-featured, portable DSU for Java that enjoys good performance and is not difficult to use" [19]. They compared their system with many other Java DSU systems, however not Gosh! This survey paper aims to fill that gap of knowledge

by comparing the newest, state of the art Java DSU systems' Gosh! and Rubah. JRebel [26,12] and DCE VM [23,24] are the other two programs that we look at in this paper. Rubahl is the only available DSU system, which doesn't modify the VM. On the contrary the DCE VM, JRebel and Gosh! modify the VM. We want both types to see on which level (VM or Application) a DSU system should work for better results.

The first section of this paper will examine the challenges of DSU systems. In the next we look at them in detail and also see how they handle the challenges. After looking at the details of the different DSU systems, we compare them. For the comparison we use properties: functionality and overhead. Not all types of code changes are allowed by using the different systems. Functionality describes their ability to change the code. However, the overhead describe how much runtime the functionality costs. A program has normally an overhead if it runs with a DSU system than without one.

2 The Problems of Dynamic Software Updating Systems

There are different kind of problems. First we have architectural problems like increase of overhead, break time, not supported updates, unportable VM, difficulty to integrate already existing software and difficulty to write an update. Secondly we have software run time problems as Phantom Objects, Absent State, Lost State, Oblivious Update, Broken Assumption and Transient Inconsistency.[8,6]

Now back to the architectural problems. A program that works not as efficiently after update than before has an overhead. The break time means the time during an update, where the program needs a break. Not supported updates are changes of code that aren't possible at runtime. They differ between the various DSU systems. However, we discuss that later in section 3.6. The unportable is really a disadvantage, because Java has the aim to be portable. However, the VM is the base of Java. It could be very difficult or very easy to integrate a DSU system in an already existing software depending on the software. There is also the question how difficult is it to write updates for software with a DSU system.

There are different kinds of runtime issues. Phantom objects are objects from a class, which was deleted in by updating. It can cause strange behavior to have such objects in the application state. Other possibilities to cause phantom objects are class renaming and adding an abstract modifier to a class. Absent state is caused by class adding, super class changing, instance/static fields adding and removing of the static modifier. It means that there isn't a state available after finishing the update. The next problem is Lost State. "Lost State happens when an updated class makes binary incompatible changes to the type of a member field"[6]. There are two causes: Changing of Instance/static field type in classes and renaming of classes. Oblivious updates are new updates that aren't complete, because a different behavior than a cold start. It is possible to create issue of oblivious updates by changing the implemented static initializer of a class or by changing the implemented constructor. Broken Assumption is created by changing a static field value and by changing implemented instance/static

methods. The relationship between the program state and the program logic is broken. For example you increment a parameter and if the parameter reaches a constant something happens. However, by setting the value of the constant under incremented parameter you get the broken assumption. Transient Inconsistency is a state you never reach in valid run time. This issue is caused by implemented instance/static methods changing.[6]

3 Previous approaches

3.1 Selection of the approaches

Java and C are the most used programming languages in DSU research and development. We chose Java because of; the advantage of portability, the higher level of programming (more complex programs) and many newly published papers.

The paper, "Dynamic Software Updating with Gosh!' provides a major influence on this report, because developers of Gosh! (Called Javeleon before acquisition by ZeroTurnAround[25]) compare their system with DCE VM and JRebel. We want to verify their results and extend the comparison by adding Rubah. Also, we get a better estimation about the usefulness of these four DSU systems. At the same time, we also examine the overhead each program creates by using the performance tests of the different developers. Rubah and Gosh! were chosen owing primarily to the recently published paper mentioning claims of their superiority by their authors, as mentioned in the introduction.

After having given the background of our chosen inquiry, we categorize DSU systems into two sections. Each concerns the manner by which DSU systems execute their ability to change code. The first category concerns DSU systems which optimize the Virtual Machine of Java. Those that change Java code directly can be placed in a second category.

3.2 DCE VM

After explaining the previously mentioned approaches, we now look at the motivation for developing the DCE VM. There is only limited possibility to change the code dynamically in Java Standard. The Java HotSpot^{TM} VM[9] furthermore only allows developers to change method bodies. However, there is strong desire among developers for this feature to be advanced in the future, by offering more customizability. In the ranking of desired features for Java, developers voted this item to the top five (at the time developers published DCE VM)[16] The DCE VM is more flexible than the Java HotSwap, because it supports both adding and removing methods, fields and super types. [23]

Conflicts by accessing deleted members However, there are difficulties with removing these instances of code. Problems are found in old references to methods or fields that were deleted. If you change the code after a time when

referenced methods are available, the Java thread will revert to old code. A similar problem is that of dynamic calls with a deleted superclass. That can cause the termination of the VM. However, the developer has solutions for each problem with their newest version, as illustrated in the following. [23]

Approaches for deleted methods and fields For the first, the software engineer has the possibility to use four different approaches; and can decide it for every class through a byte code attribute.

A: Wait for a time when a safe update is possible

B: Update once available

C: Get access to old methods along with static fields

D: Ensure that old code references only to old code and new code references only to new code

However, the developers aim to eventually implement so called "forward points that are hints which instructions of the old method match instructions of the new method"[24]. With this implementation, they think that should solve the problem and also increase the update time.[24]

Solution for dynamic calls For the second problem (changing a superclass) developers now have a solution to avoid termination of the VM. The VM performs two different verifications. First, the VM checks if a narrowed type – a type affected by superclass changing – is on the stack. Second, the VM checks byte code, to determine whether it considers the new class hierarchy. [24]

DCE VM Implementation For the implementation, developers only need to modify; the garbage collector, the system dictionary and the class metadata of the VM. These changes do not influence normal program execution. However, the VM will be modified, that's why the DCE VM belongs to the first category.[24]

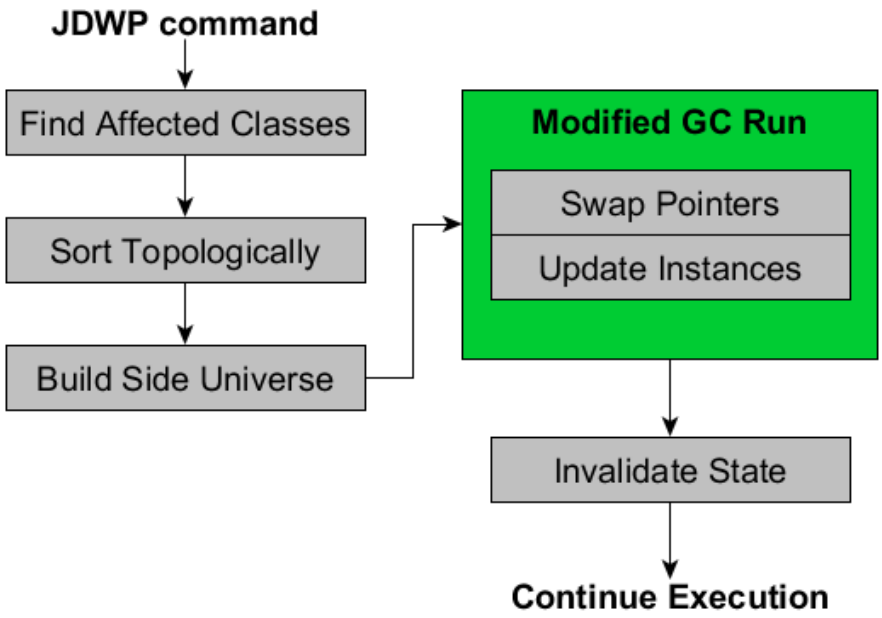


Fig. 1. Steps performed by the code evolution algorithm[24]

The Figure 1 shows steps performed by the DCE VM during code evolution. The code change preparation can run simultaneously to the working program, until we reach the 'Modified GC Run'. The modified Garbage Collector needs to cease the working program with the assistance of a safe point mechanism used by the standard GC for suspending all threads. Before the first step 'Find Affected Classes' a Java Debug Wire Protocol (JDWP) [15] command is called.[24]

Find Affected Classes is important, because it is faster to redefine only the new parts than all the code. This is achieved through the fact that Java is an object-oriented programming language with subclasses and subinterfaces affected by modified classes and interfaces respectively. New fields must be added to all subclasses. New or deleted methods in a class also change the virtual method tables of subclasses and subinterfaces.[24]

Sort Topologically is used for sorting the classes "based on their subtype relationships"[24]. The problem that occurs is that the information about the relationships becomes available only after loading the classes. The developers parse a part of the class files to get the needed information.[24]

Build Side Universe: In this step the new classes coexist with the old classes. This prevents the problem of cyclic dependencies, that is the problem that a

class needs another class and vice versa for updating. The VM only copies the static field values instead of using the static class initializer.[24]

Modified GC Run - Swap Pointers: The Heap is traversed for updating every instances of a class. Every object that was redefined is collected in a list. Every object that have increased size (e.g. new attribute) will be saved for next step.[24]

Modified GC Run - Update Instances : First all fields of the old instances which exist in new instances are copied. The fields which only exist in the new instances are set to 0, null or false depending on the type. However modified GC algorithm is triggered by changing, heap increase or decrease as need for the size of the objects. Also the algorithm overwrites the old instances on the heap.[24]

Invalidate State In this stage different subsystem need modification. They must handle issue of broken assumption by the original unsupported dynamic updating. For example "that field offsets never change".[24]

DCE VM Runtime performance The authors performed a number of benchmarks to show the DCE VM performance. In general, the modified DCE VM runs at about the same efficiency as the original Java VM. In the first benchmark, the program was run twenty five times. In the eighth and sixteenth run, they started a dynamic update. The two updates they performed, created about three percent overhead at the peak. The VM needs to recompile updated methods thereby making the runs last longer at the beginning. After that the overhead vanishes.[24]

3.3 JRebel

Opposite to DCE VM, JRebel modifies the Java VM only slightly. Therefore, it belongs more to the second category. JRebel extends the –javaagent, a plugin of the Java VM. JRebel has great flexibility and can be used with "multiple JVMs and JVM-based languages like Scala and Groovy"[26]. As the DCE VM, it is possible to add and change methods, classes and fields. However, adding and changing of super classes and implemented interfaces is not supported. The developers had in mind, to increase the functionality. The Java HotSwap works on the VM to increase support more than changing method bodies. This way, there arises a need to modify the internal memory structure of a running program. To get extended code changing support, the DCE VM developers modify the garbage collector as it is responsible for relocating objects. Java supports multiple hardware platforms and operating systems. By changing VM we lose the portability, which could save developers a huge amount of work. Another problem is the existence of four different Garbage Collectors available in the VM, with only two equipped to handle multi-threading support. That is why JRebel takes another road to reach the goal, as follows.[26,12]

JRebel Implementation: "JRebel makes use of two remarkable features of the JVM abstract byte code and class loaders"[12]. As can be seen in Figure 2, JRebel works as an layer between the Byte Code and Java Virtual Machine layers. JRebel utilizes the class loaders as notifies, informing whether a class was loaded or not. Then, it translates code on-the-fly to integrate code changes. It is not possible to reload a class for changing after it is loaded. To solve the this problem the class is rewritten at the start to a frontend class and an interchangeable backend class as shown in Figure 3 is used. To update a class the backend class is changed and Reflection API uses the frontend class without any issues. [12]

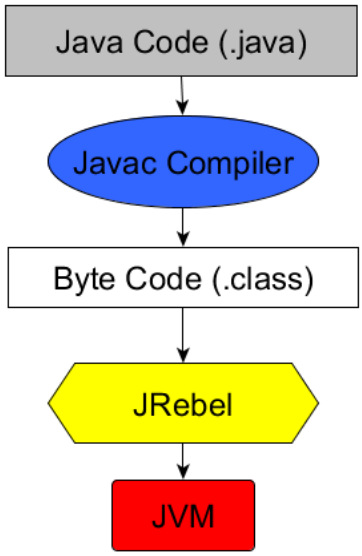


Fig. 2. Codechange hierarchy of JRebel

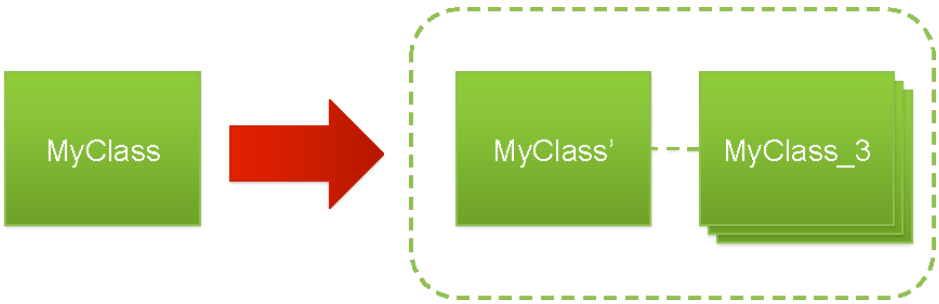


Fig. 3. Front- and Backendclass[12]

JRebel's Performance: The developers used Chameneos, a Concurrency Game for Java, Ada and Others[13] for the benchmark. They figured out, that if the update change all classes the overhead will be below 60%. However normally the updates will only change a few, so the Overhead will be much lower than 60%. The developers say that the overhead during development, the aim of JRebel, is about 10 - 15%.[26,12]

3.4 Gosh!

Gosh! (previously Javeleon) works on application-level same as JRebel so it belongs to the second category. The developers want to create a DSU system that would have more options to change the code. Like stated at section 1 they claim to have the most functionality. Gosh! offers at least more functionality than the DCE VM and JRebel as discussed later in section 3.6.[5,6]

Gosh! Implementation: Gosh!'s architecture consists of two abstraction layers called Gosh! Runtime and Gosh! Agent. Figure 4 displays the outline of the structure. Gosh! uses custom class loaders to create different namespaces to have versioned classes. That cause a problem called version barrier. "The version barrier prohibits one component from passing an instance to another component if each component contains that class type"[21]. However there is already a solution "called sister namespaces, to address this problem"[21]. Passing instances of objects to another component between two sister namespaces of a namespace. Now follows a breakdown into the two layers explaining their usage.[6]

Gosh! Agent: The Agent consist of four subcomponents. The class-loading plug-in is needed to integrate Gosh! into an integrated development environment (IDE), however at the moment only NetBeans[1] is supported. It is used to update configurations files and resources of the IDE.

Then there is the bytecode-transformer, which is used to control the classes. The classes are divided in two categories: system and non-system classes. The latter have normal a higher overhead, because you can update them dynamic. However, the system classes are only aware of the updates of their subclasses. We decrease the overhead trough the assumption to have no need to update system classes.

The two last subcomponents bootstrap-class-transformer and sub- process-spawner are responsible for creating a new sub process with modified bootstrap-classes. The bootstrap-class is the classloader for the core libraries. It also aims to give the user transparent view on it.[6]

Gosh! Runtime: For the runtime mechanism three components needed. The mechanism is used for the dynamic updating, so the references are handled right. The state transfer from old class and object versions to the new are done by the runtime mechanism. It is lazy state transfer, so the state transfer starts once the new version is requested. This is to prevent the block that user received by transfer everything at once.[6]

The In-Place Proxification "uses in-place-generated code to provide the level of indirection to a newer version within the code of the present one"[5]. This used for the so called lazy algorithm. Incoming request on an object need to be executed locally or forwarded to the actual object with the help of the Java Reflection API[17].There are issues with forwarding if types-changed, because the Java Reflection API is not able to handle it. That is why the second component the Correspondence mapping is needed. It delegates the request to recent version of the class.[5]

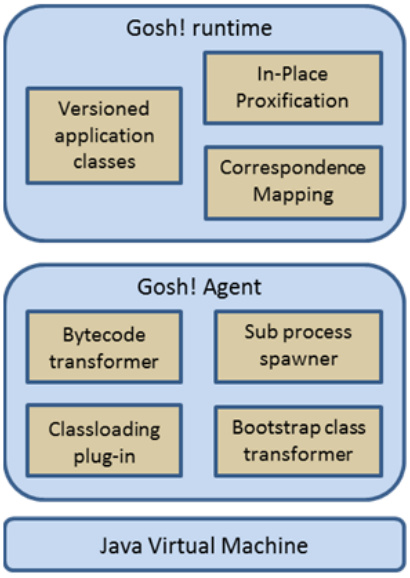


Fig. 4. Architectural overview of Gosh![6]

Gosh! Runtime performance The developers of Gosh! decided to benchmark their DSU system against JRebel, because both systems belong to the second category. To compare the result there is a need for a baseline, so they decided to benchmark the standard VM, too. In the most benchmarks Gosh! is better than JRebel. The rest of the time they are about equal. The Standard VM performs at maximum about 20% better than Gosh!, However, sometimes they are equal. The maximum and average overhead of JRebel should be higher than Gosh!. Thats why the average Overhead of Gosh! probably is about 10% They comparison of all in this paper covered approaches is in section 3.6.[6]

3.5 Rubah

Rubah is the newest Java DSU system, because Gosh! is only advancement of Javeleon. Rubah took his idea from a DSU system programed in C called

Kitsune[11,22]. New this approach that it solve the problem to create a DSU system without modifiying the VM. Rubah also want to reduce the high steady overhead as known from DCE VM, JRebel and Gosh!. THis overhead is mainly so high, because of the usage of proxies that forward calls. Rubah use two different algorithms to reduce pause time between an update. [18,19]

Rubah Implementation: There are six steps involved in updating program with Rubah as shown in Figure 5. The Analyzer extracts meta-data and the list with all updateable classes and save it in v0.desc. After the first step the descriptor(v0.desc) and application's classes sent to the driver. That rewrites the byte code to get the possibility to update the loaded class in the future. The rewriting is needed for the state transformation. In the next step the new Code and old descriptor are sent to Analyzer. As result we got the UpdateClass.java (saves how the objects need to change), Skeleton.jar and a new descriptor. At the fourth step javac compiles the UpdateClass.java and the Skeleton.jar to the UpdateClass.class. In next update is deployed by the updater.[18,19]

The updater triggers the driver, who deploys the update in three steps (quiescence, state transformation, control flow migration). Quiescence is the step where the program waits that the thread reach an update point and halt the program. In the state transformation "the driver initiates (and may complete) the modification of object instances whose class has changed (according to the update class)"[19]. The driver doesn't complete the In the last step called control flow migration all threads are restarted where they stopped. For next updates there is only the need to repeat the steps three to six. The steps one and two are only used the first time.[19]

For only changing method bodies in a class they use the support of the Java HotSwap. However for more they need to rewrite the bytecode. The developers used an already existing ASM byte code rewriting tool[2]. The bytecode rewriting consists of three parts. First all classes get renamed from 'class name' to 'class name'__'version number'. To prevent problems with the Reflection API by changed class names, they rewrite all invocation of Reflection API to use Rubah's API instead. The second part is to rewrite all updateable fields and method parameters with java.lang.Object. It prevents the need to update classes that only refer to changed classes. It is important to update all objects of the updated classes and the overwritten hashCode methods need the same semantic, because that the Collections work properly. The steady overhead is primarily created through the type erasure and the hashCode issue.[19]

For a shorter break time between updates, there is the lazy proxy approach. At the moment a proxy is called, the update starts and the proxy gets deleted. So that steady overhead stays low.[19]

The developers only tested Rubah on the Java HotSwap, however they didn't modify the VM so Rubah should be portable. They used some 'unsafe operations' to build Rubah. The developers think that Rubah is portable to IBM's Jikes and OpenJDK.[19]

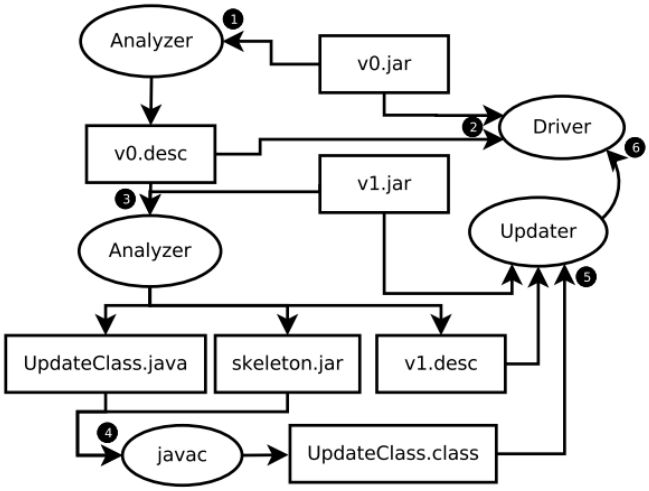


Fig. 5. "Deploying a program, and preparing and installing an update for it, using Rubah. Square boxes represent artifacts: Compiled code (jar/class), source code (java), or update descriptors (desc). Round boxes represent tools: Rubah's driver, analyzer, and updater, and the unmodified Java compiler (javac)"[19].

Rubah Runtime performance : The benchmarks show that the steady overhead of Rubah is between 5% and 9%. The break time until the update is finished by parallel algorithm is very high for bigger heaps, however the lazy algorithm isn't influenced that much by the heap. The developers want to improve the performance of the parallel algorithm and also decrease the steady overhead.[19]

3.6 Comparison Power and Overhead

If we look at the performance, the steady overhead of DCE VM is the smallest. Short time after an update the overhead disappears. However, the DCE VM is not portable, because they modified Virtual Machine. Second smallest overhead with 5%-9% steady overhead is owned by Rubah. The big advantage of Rubah is also that the developers think they can port it to other VM without changing the code. The next one is Gosh! that runs similar fast with a steady overhead about 10%. There are little modifications on the VM to get the possibility to update. The last is JRebel with normal steady overhead of 10%-15% and at peak 60%. As Gosh! the VM is also slightly modified.

After looking at all the four DSU machines, the best would be Rubah, because of the small steady overhead and portability. Gosh! and JRebel are eventually also portable with a few modifications, however JRebel only aims to be used at Development. This result is only based on the performance and portability. The result should be proofed by benchmarking all systems together. We need to look on the power of the possible code changes to get a better picture.

Table 1. Supported Operations, table is based on the paper 'Run-time Phenomena in Dynamic Software Updating'[8] and the symbols are from clker.com[14,20,10]

Supported Operation	HotSpot VM	DCE VM	JRebel	Gosh!	Rubah
Swap method body	✓	✓	✓	✓	✓
Adding/removing fields Move	✗	✓	✓	✓	✓
Move field to super/sub class (state preserving)	✗	?	✗	✓	✓
Changing static field value	✗	✗	?	✗	?
Changing static final field value	✗	✓	✓	✓	✓
Automatic initialization of new fields	✗	✗	✗	?	?
Adding/removing methods	✗	✓	✓	✓	✓
Adding/removing constructors	✗	✓	✓	✓	✓
Adding/removing classes	✗	?	✓	✓	✓
Changing interfaces	✗	✓	✓	✓	?
Adding/removing enum values	✗	✗	✓	✓	?
Replace superclass	✗	?	✗	✓	?
Adding/removing implemented interface	✗	✓	✗	✓	?
Reloading of resources	✗	✗	?	?	?

✓ Supported ✗ Not Supported ? Issues

Not supported are all code changes that you can't use. The functionality marked with issue are all functions that work only partly or in case of Rubah work most likely. We assume the functionality of Rubah, because it was not possible to test it. Some of the issue are explained in the earlier part of this work.

4 Conclusion

The different DSU systems have a broad enough range of code changes to use them in project. However, there are only a few program that use or try to implement some DSU systems. For better comparison there should be more benchmarks or least one that test all systems equally. Only the developer. As written earlier there are more properties that you can compare for example difficult of writing updates. In future works it is useful to look at other properties.

References

1. NetBeans IDE Website (2014), `https://netbeans.org/`

2. Bruneton, E., Lenglet, R., Coupaye, T.: ASM: a code manipulation tool to implement adaptable systems. Adaptable and extensible ... (2002), http://geoe.mu.edu.tr/sharedoc/asm2-2.2.3/asm-eng.pdf
3. Gregersen, A.R.: Implications of modular systems on dynamic updating. Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering - CBSE '11 p. 169 (2011), http://portal.acm.org/citation.cfm?doid=2000229.2000254
4. Gregersen, A.R., Jorgensen, B.N., Hadaytullah, H., Koskimies, K.: Javeleon: An Integrated Platform for Dynamic Software Updating and Its Application in Self-* Systems. In: Engineering and Technology (S-CET), 2012 Spring Congress on. pp. 1–9. IEEE (2012)
5. Gregersen, A.R., Jorgensen, B.N.: Dynamic update of Java applications — balancing change flexibility vs programming transparency pp. 81–112 (2009)
6. Gregersen, A.R., Rasmussen, M., Jorgensen, B.N.: Dynamic Software Updating with Gosh! (2013)
7. Gregersen, A.R., Simon, D., Park, M., Jorgensen, B.N.: Towards a Dynamic-Update-Enabled JVM (2009)
8. Gregersen, A., Jorgensen, B.: Run-time phenomena in dynamic software updating: causes and effects. ... Workshop on Principles of Software ... pp. 6–15 (2011), http://dl.acm.org/citation.cfm?id=2024448
9. Group, O.H.: OpenJDK HotSpot group - Website, http://openjdk.java.net/groups/hotspot/
10. Hahn, G.: Red Cross x clip art (2007), http://www.clker.com/clipart-3584.html
11. Hayden, C., Smith, E., Denchev, M.: Kitsune: Efficient, general-purpose dynamic software updating for C. ACM SIGPLAN ... (2012), http://dl.acm.org/citation.cfm?id=2384635
12. Kabanov, J.: JRebel Tool Demo. Electronic Notes in Theoretical Computer Science 264(4), 51–57 (Feb 2011), http://linkinghub.elsevier.com/retrieve/pii/S1571066111000429
13. Kaiser, C., Pradat-Peyre, J.F.: Chameneos, a concurrency game for Java, Ada and others. In: Int. Conf. ACS/IEEE AICCSA. vol. 114 (2003)
14. Meliss: Green Tick clip art (2007), http://www.clker.com/clipart-12246.html
15. Oracle Corporation: Java Debug Wire Protocol (JDWP) (2011)
16. Oracle Corporation: Top 25 RFEs (Requests for Enhancements) (2011), http://bugs.sun.com/top25_rfes.do
17. Oracle Corporation: Java reflection specification (2013), http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/package-summary.html
18. Pina, L., Hicks, M.: Rubah : Efficient , General-purpose Dynamic Software Updating for Java. In: Proceedings of the Workshop on Hot Topics in Software Upgrades (HotSWUp). pp. 1–6 (2013)
19. Pina, L., Veiga, L., Hicks, M.: Rubah: DSU for Java on a stock JVM
20. Rodee, M.: Question Mark clip art (2007), http://www.clker.com/clipart-10842.html
21. Sato, Y., Chiba, S.: Loosely-Separated " Sister " Namespaces in Java pp. 49–70 (2005)
22. Smith, E.K., Hicks, M., Foster, J.S.: Towards standardized benchmarks for Dynamic Software Updating systems. 2012 4th International Workshop on Hot Topics in Software Upgrades (HotSWUp) pp. 11–15 (Jun 2012), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6226609

23. Würthinger, T., Wimmer, C., Stadler, L.: Dynamic code evolution for Java. In: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java. pp. 10–19. ACM (2010)
24. Würthinger, T., Wimmer, C., Stadler, L.: Unrestricted and safe dynamic code evolution for Java. Science of Computer Programming 78(5), 481–498 (2013)
25. ZeroTurnAround: JRebel - ZeroTurnAround Webpage, `http://zeroturnaround.com/software/jrebel/`
26. ZeroTurnAround: JRebel: WHAT DEVELOPERS WANT (2012)

Parallelen und Unterschiede vergangener kritischer Stromausfälle

Maximilian Kremer*

Betreuer: Yong Ding†

Karlsruher Institut für Technologie (KIT)
Pervasive Computing Systems – TECO
*uagrl@student.kit.edu
†ding@teco.edu

Zusammenfassung. Unsere heutige Gesellschaft ist mehr von Strom abhängig denn je. Durch das Wachstum unserer Gesellschaft und speziell das Wachstum der Entwicklungsländer nimmt der Stromverbrauch zu.[1] Das Stromnetz wird stärker belastet und fragiler. Betrachtet man die zehn größten Stromausfälle, die sich bisher ereigneten, haben sich alleine sieben davon in diesem Jahrtausend ereignet. Erst 2012 kam es zum weltweit größten Stromausfall aller Zeiten. In Indien waren 600 Millionen Menschen stundenlang ohne Strom. [2] Aufgrund dieser Entwicklung hat die Stromausfallprävention einen immer höheren Stellenwert. Im folgenden werden zwei große Stromausfälle analysiert und einige Präventionsmöglichkeiten vorgestellt und erläutert.

Schlüsselwörter: Stromausfall, Blackout, Prävention

1 Einleitung

Zu Stromausfällen kann es jederzeit kommen. So gut wie jeder war bereits einmal davon betroffen und hat einen solchen miterlebt.

In dieser Arbeit werden zwei große Stromausfälle von 2003 in Italien und Nordamerika untersucht. Bei beiden waren in etwa 55 Millionen Menschen bis zu mehreren Stunden ohne Strom. Bei der Untersuchung werden Ursachen für die Stromausfälle herausgearbeitet. Anschließend wird erörtert wie die Stromausfälle hätten verhindert werden können. Im zweiten Teil der Arbeit werden verschiedene Präventionsmöglichkeiten, um Stromausfällen vorzubeugen, untersucht und vorgestellt. Dabei wird es sich um einen groben Überblick verschiedener Maßnahmen handeln. Zunächst werden aber ein paar Grundlagen vermittelt, welche benötigt werden, um den Ablauf der Stromausfälle verstehen zu können.

2 Grundlagen

Nachfolgend wird häufiger von Hochspannungsleitungen, die in Folge von zu hoch gewachsener Bäume ausgefallen sind, die Rede sein. Dieses Ereignis passiert nicht zufällig bei den meisten Stromausfällen, sondern lässt sich physikalisch

erklären. Die Leitungen bestehen aus Metall, meist Kupfer und Aluminium mit einem Stahlkern. Metall dehnt sich bei erhöhten Temperaturen aus. Der Stromfluss erzeugt aufgrund des Widerstandes Wärme. Ein höherer Stromfluss durch eine Leitung, erzeugt mehr Wärme und das Metall dehnt sich aus. Das heißt, dass die Leitungen länger werden, also tiefer hängen und sich somit unter den Leitungen befindlichen Bäumen oder anderen Objekten nähern. Sollte nun die Distanz zwischen einem Objekt und der Leitung einen gewissen Wert unterschreiten, kommt es zu einem Lichtbogen. Dieser Lichtbogen löst einen Kurzschluss aus. Damit keine Folgeschäden entstehen werden diese Kurzschlüsse von Sicherheitssystemen erkannt und die Leitung vom Netz genommen. Dadurch werden stärkere Schäden an Leitungen vermieden und es wird etwaigen kostspieligen Reparaturen vorgebeugt. Um solche Vorfälle zu vermeiden, werden regelmäßig die Hochspannungsleitungen abgefahren und Bäume gekürzt. Dies findet meist in einem Zyklus von fünf Jahren statt. [3]

Um die Ausbreitung von Stromausfällen nachvollziehen zu können, ist es wichtig zu wissen, wie eine Kaskade funktioniert und warum sie auftritt. Eine Kaskade tritt auf, wenn es zu einem sequenziellen Ausfall mehrerer Leitungen und Generatoren kommt. Ausgangspunkt ist meist ein Ausfall einer Leitung. Dieser Ausfall löst Strom- und Spannungsschwankungen auf anderen Leitungen aus. Sicherheitssysteme erkennen diese Schwankungen als Fehler und schalten die Leitungen ab. Ähnliches gilt für Generatoren, die aus Sicherheitsgründen abgeschaltet werden. Durch das Abschalten von Leitungen und Generatoren breitet sich der Stromausfall weiter aus.
Bei Ausfall einer Stromleitung kann es durch den zusätzlichen Stromfluss auf anderen parallel geleiteten Leitungen zu einer höheren Wärmeentstehung kommen. Hier kann eine Kaskade wie bereits beschrieben mittels Lichtbögen stattfinden. [4]

Die N-1 Regel ist ein Grundsatz eines jeden Stromanbieters und besteht aus zwei Schritten:
1. Der Verlust einer Verbindungs- oder Stromgenerierungseinheit darf die Stabilität des verbundenen Stromnetzes nicht gefährden. Mit anderen Worten: *„sind für eine Aufgabe n Objekte zuständig oder verfügbar, so kann bei Einhaltung der (n–1)-Regel beim Ausfall eines Objekts der Betrieb oder Funktionstüchtigkeit durch die anderen n-1 Objekte sicher gewährleistet werden.“* [5]
2. Nach Wahrung der Stabilität müssen, falls nötig, alle Maßnahmen ergriffen werden, um das System wieder in den N-1 Secure State zu bringen. Dieses Prinzip wird weltweit angewandt, um Kaskaden vorzubeugen. [6]

3 Untersuchung vergangener überregionaler Stromausfälle

Im Folgenden wird auf zwei große Stromausfälle aus dem Jahre 2003 eingegangen. Diese Stromausfälle wurden gewählt, da sich beide in diesem Jahrtausend ereigneten und auf unterschiedliche Art und Weise zum Ausfall führten. Auch zeigen sich hier die Unterschiede zwischen dem nordamerikanischen und dem europäischen Stromnetz.

3.1 Stromausfall in Nordamerika am 14. August 2003

Ursachen Der Stromausfall in Nordamerika wurde durch ein Zusammenspiel mehrerer Umstände ausgelöst. Das Midwest Independent System Operator (heute Midcontinent Independent System Operator, oder kurz MISO)[1] überwacht das Hochspannungs-Netz in Nordost USA und Teilen von Kanada. Dafür wird ein sogenannter Zustandsanzeiger (state estimator)[2] verwendet.
Die Rohdaten, die der Zustandsanzeiger liefert, werden an Software Tools, wie das real time contingency analysis (RTCA), weitergegeben. Dieses Tool simuliert bei jedem Aufruf verschiedene Zustände und Ausfälle, um die Zuverlässigkeit des Stromnetzes zu testen. Mit dem Zustandsanzeiger und dem Software-Tool können die System Operatoren jede Möglichkeit betrachten und so bestimmen, ob die möglichen zukünftigen Zustände immer noch dem N-1 Secure State entsprechen. [7] Sollten die Daten, die der Zustandsanzeiger empfängt, zu alt oder falsch sein, wird ein Fehler produziert. Ein Trigger lässt den Zustandsanzeiger alle fünf Minuten und das RTCA etwas seltener laufen.
An jenem 14. August um 12:15 EDT[3] lieferte der Zustandsanzeiger einen Fehler. Dieser Fehler resultierte aus dem Ausfall Cinergy's Bloomington-Denois Creek 230-kV Leitung. Der Ausfall der Hochspannungsleitung wurde nicht aktualisiert und der Zustandsanzeiger arbeitete mit Daten, die ergaben, dass die Hochspannungsleitung immer noch verfügbar sei. Dieser Fehler wurde von einem Mitarbeiter bemerkt. Um das Systemabbild wieder auf den aktuellen Stand zu bringen, schaltete der Mitarbeiter den Trigger, welcher dafür sorgt, dass Zustandsanzeiger und RTCA regelmäßig laufen, aus. Nach der erfolgreichen Fehlerbehebung wurde vergessen, den Trigger wieder zu aktivieren. Dies geschah gegen 13:00 EDT. Um 14:40 EDT wurde bemerkt, dass das System nicht lief und man schaltete es wieder ein.
Als um 14:40 EDT das System wieder online ging, war in der Zwischenzeit die Stuart-Atlanta Leitung ausgefallen. Der Ausfall wurde, wie schon bei der Cinergy's Bloomington-Denois Creek 230-kV Leitung, nicht bemerkt und das System arbeitete erneut mit falschen Daten. Dementsprechend lieferte das System einen Fehler. Die Überprüfung des Netzes ergab fälschlicher Weise, dass die

[1] Ist ein Dienstleistungsunternehmen, welches das Stromnetz Mittleren Westen und in Manitoba, Kanada operativ betreibt
[2] „Computer software that takes redundant measurements of quantities related to system state as input and provides an estimate of the system state"[7]
[3] Eastern Daylight Time

Stuart-Atlanta Leitung nicht ausgefallen war. Es wurde also mit falschen Daten weitergearbeitet. Um 15:29 EDT rief ein MISO Operator PJM[4] an und erhielt die Information, dass die Hochspannungsleitung ausgefallen ist. Durch die neuen richtigen Informationen war es möglich das System um 16:04 EDT wieder zuverlässig zum Laufen zu bringen.
Zusammenfassend lässt sich sagen, dass das MISO System zwischen 12:15 EDT und 16:04 EDT nicht aktiv war. Durch ein korrekt funktionierendes System hätte der Stromausfall verhindert werden können. [8]

Parallel zu den obigen Ereignissen fiel um 13:31 EDT die Eastlake Unit 5[5] aus. FirstEnergy[6] führte trotz des Ausfalls, aus unbekannten Gründen, keine Analysen durch, um zu testen, ob weitere Ausfälle den N-1 Secure-State des Stromnetzes gefährden würden.
Erschwerend kam hinzu, dass das Alarmsystem von FirstEnergy um 14:14 EDT ausfiel. Dieses System gibt automatisch bei ungewöhnlichen Ereignissen einen Alarm raus. Durch den Ausfall arbeiteten die Operatoren von FirstEnergy mit einem komplexen Stromversorgungssystem, ohne adäquate Indikatoren, wann Schlüsselelemente des Systems die Grenzen von Safe-Operations erreichten und übertraten. [9] Der Grund für den Ausfall war, dass das System bei Verarbeitung eines Alarms in eine Endlosschleife gelangte. Weitere eingehende Daten wurden in eine Warteschlange gestellt. Da es aber kein Meldesystem für den Ausfall des Alarmsystems gab, wurde nicht bemerkt, dass das Alarmsystem nicht mehr lief. Durch den Ausfall des Alarmsystems waren die FE Area Operatoren nicht in der Lage zu bemerken, dass elementare Teile des Stromnetzes ausgefallen waren. Nach dem Ausfall der Harding-Chamberlin Leitung um 15:05 EDT und durch den Ausfall der Eastlake Unit 5 war die Stromversorgung so geschwächt, dass das regionale Stromnetz seinen N-1 Secure-State verlor. [10]

Ausbreitung Durch die oben beschriebenen Ereignisse kam es zu großen Ladungsverschiebungen. Daraus resultierte, dass zwischen 15:05:41 EDT und 15:41 :35 EDT drei weitere große Leitungen ausfielen. Diese Ausfälle geschahen nicht durch Zufall, sondern entstanden durch Lichtbögen. Dies führte zu einer Kaskade, die das gesamte Stromnetz im Raume Cleveland betraf, welches als Folge der Überlastung zusammenbrach und der Strom in der gesamten Region ausfiel. Der entscheidende Ausfall, der zur Kaskade des Blackouts über Großteile Nordamerikas führte, war der Ausfall der Sammis-Star 345-kV Leitung um 16:05:57 EDT. Innerhalb von sieben Minuten verbreitete sich der Stromausfall von Cleveland und Umgebung über den Nordosten der USA und Süden Kanadas, sodass um 16:13 EDT mehrere Zehnmillionen Menschen keinen Strom hatten. [11]
Die Ausbreitung des Stromausfalls wurde aufgrund der zunehmenden Entfernung vom Ausgangspunkt, ähnlich eines Erdbebens und dessen Distanz zum

[4] Ist ein Dienstleistungsunternehmen, welches das Stromnetz in Nordost-USA operativ betreibt und mit MISO zusammenarbeitet
[5] Kernkraftwerkblock in Ohio
[6] kurz FE, ist ein US-Amerikanisches Energieversorgungsunternehmen, welches in den Bundesstaaten Ohio, Pennsylvania und New Jersey tätig ist

Epizentrum, eingedämmt. Besser ausgebaute und geschützte Regionen konnten die Strom- und Spannungsschwankungen ausgleichen. Einerseits durch Stromleitungen, die mehr Spannung tragen konnten, andererseits durch automatische Präventionsmechanismen, wie Load Shedding. [12]

Analyse Das Untersuchungsteam fand heraus, dass der Stromausfall mit Load Shedding hätte verhindert werden können. Genauer der Verlust der Sammis-Star hätte verhindert werden können. Wäre die Sammis-Star nicht ausgefallen, hätte es keine Kaskade über Nordamerika gegeben und es wäre bei einem kleinen lokalen Stromausfall geblieben.[11] Die ganze Region um Cleveland und Akron hatte zu diesem Zeitpunkt 2003 kein automatisches Load Shedding. Durch die Missachtung der kritischen Situation wurde auch kein manuelles Load Shedding durchgeführt.

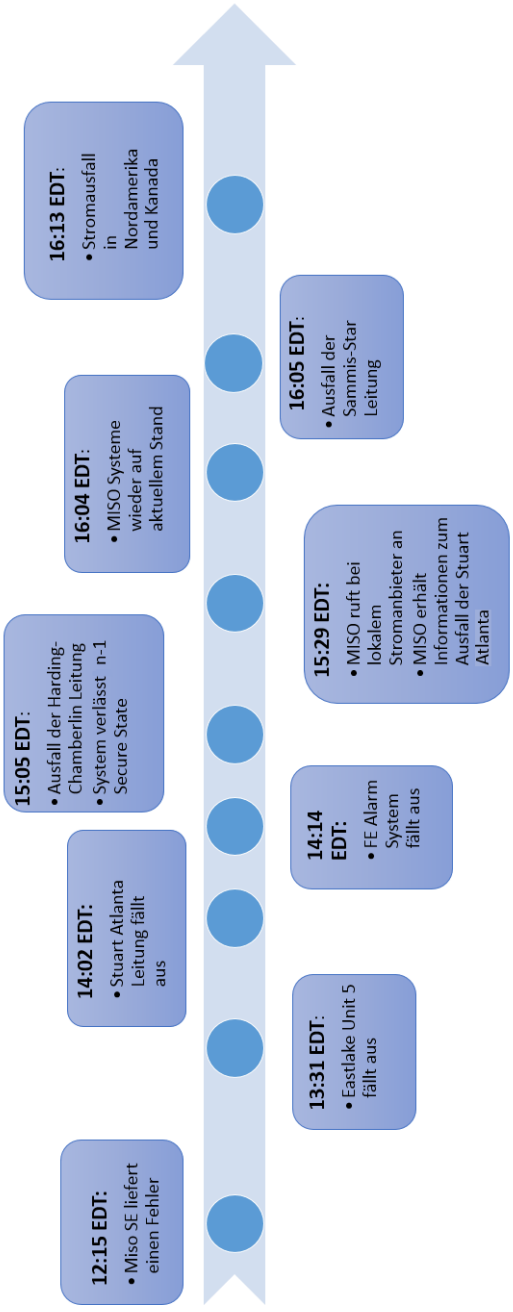


Abb. 1. Die Grafik zeigt eine zeitliche Darstellung der wichtigsten Ereignisse, die zu dem Stromausfall in Nordamerika führten [13]

3.2 Stromausfall vom 28.09.2003 in Italien/Schweiz

Ursachen Italien importierte im Jahre 2003 24 Prozent des eigenen Strombedarfs aus Frankreich und der Schweiz. Zum Zeitpunkt des Stromausfalls importierte Italien mehr als das vereinbarte Volumen. Durch die zusätzlichen Importe kam es auf den Leitungen zwischen Schweiz und Italien zu einer höheren Belastung als geplant.
Um 3:01 CET[7] kam es zum Ausfall der Lukmanier Leitung aufgrund des Kontakts mit einem Baum. Der Ausfall der Lukmanier Leitung führte zu Überladungen auf anderen Leitungen, unter anderem auf der San Bernardino Leitung. Versuche der automatischen Wiedereinschaltung waren erfolglos. Die Leitung wurde von seinem eigenen Sicherheitssystem abgekoppelt. Auch Versuche der Operatoren, die Leitung wieder ans Netz anzuschließen, misslangen, aufgrund des hohen Phasenwinkels von 42 Grad, welcher durch den kontinuierlich hohen Stromfluss nach Italien verursacht wurde.[14] Ein Wiederverbinden wird bei hohem Unterschied des Phasenwinkels verhindert, um Generatoren vor Schaden zu schützen. [14]
Um 3:10 CET nahm die ETRANS[8] Kontakt zur GRTN[9] auf und bat die Importe auf die ursprünglich vereinbarten 6.400 MW zu reduzieren. Dieses Vorgehen sollte dafür sorgen, dass die San Bernardino Leitung nicht mehr einer Überladung von 110 Prozent ausgesetzt ist. Denn es war bekannt, dass die San Bernardino Leitung der Überlastung nur eine gewisse Zeit Stand halten konnte. Die GRTN ging diesem Wunsch nach. Allerdings wurde das erforderte Level von 100 Prozent Auslastung erst um 3:21 CET erreicht und dieses auch nicht lange gehalten. Denn kurz nach dem Erreichen wurde das Importvolumen wieder gesteigert.
Analysen ergaben, dass die Leitung die Überladung circa 15 Minuten schadenfrei überstehen konnte. Die gesamte Prozedur der Verständigung und der Anpassung der Spannung dauerte aber 24 Minuten. Daraus resultierend kam es um 3:25 CET zum Ausfall der San Bernardino Leitung aufgrund eines Lichtbogens mit einem Baum.

Ausbreitung Um 03:25:24 CET gab es keine Verbindung mehr zwischen dem Stromnetz von Italien und dem Netz der UCTE[10]. Durch das Verlieren der Synchronizität zwischen dem italienischen Netz und dem Netz UCTE, wurden Sicherheitsmechanismen aktiviert, die die Leitungen verbindenden Netze, kappten. Nach der Isolation des italienischen Netzes entstand ein Defizit von 6 646 MW. Durch das Defizit wurde ein Frequenz-Transient ausgelöst, der sämtliche Sicherheitssysteme aktivierte.[15]

[7] Central European Time
[8] ETRANS AG ist die schweizerische Koordinationsstelle für das Höchstspannungsnetz in der Schweiz
[9] Gestore dei Servizi Energetici GSE S.p.A, italienischer Stromanbieter
[10] Die Union for the Co-ordination of Transmission of Electricity war für die Koordinierung und Erweiterung des europäischen Netzverbundes zuständig. Seit 01.07.2009 hat die ENTSO-E, European Network of Transmission System Operators for Electricity, die Aufgaben der UCTE übernommen.

Innerhalb von 12 Sekunden fielen sämtliche Leitungen aus, die Italien mit Rest-Europa verbanden und das italienische Stromnetz isolierten. Um 3:27 CET kam es aufgrund des großen Energiemangels zu einem Stromausfall in ganz Italien.

Analyse Als Reaktion auf den Stromausfall wurden vielerlei Maßnahmen getätigt. Die Net Transfer Capacity nach Italien wurde verringert. Ein Memorandum of Understanding zwischen ETRANS und GRTN wurde unterzeichnet, mit dem Ziel die Zusammenarbeit zu verbessern. Des Weiteren wurde die Phasenwinkelkalkulation in die Contingency Analysis eingebunden. Es werden nun Echtzeitinformationen über die Regionen auf die Schweiz und Italien gemeinsam den größten Einfluss haben, zwischen der Schweiz und Italien ausgetauscht. Außerdem wurden verstärkt zeitsynchronisierte Zeigermessgeräte installiert, um mehr Daten über den Strom zu haben.[16]

Schlussendlich lässt sich sagen, dass der Auslöser des Stromausfalls nicht auf italienischem Boden lag. Der Ausgangspunkt war der Ausfall der Lukmanier Leitung in der Schweiz. Die Kommunikationsprobleme zwischen Italien und der Schweiz führten dazu, dass der Stromausfall nicht schnell behoben werden konnte. Italien muss sich vorwerfen lassen, dass das Stromnetz nicht in der Lage war den Frequenzverlust und die damit verbundene Kaskade zu verhindern.

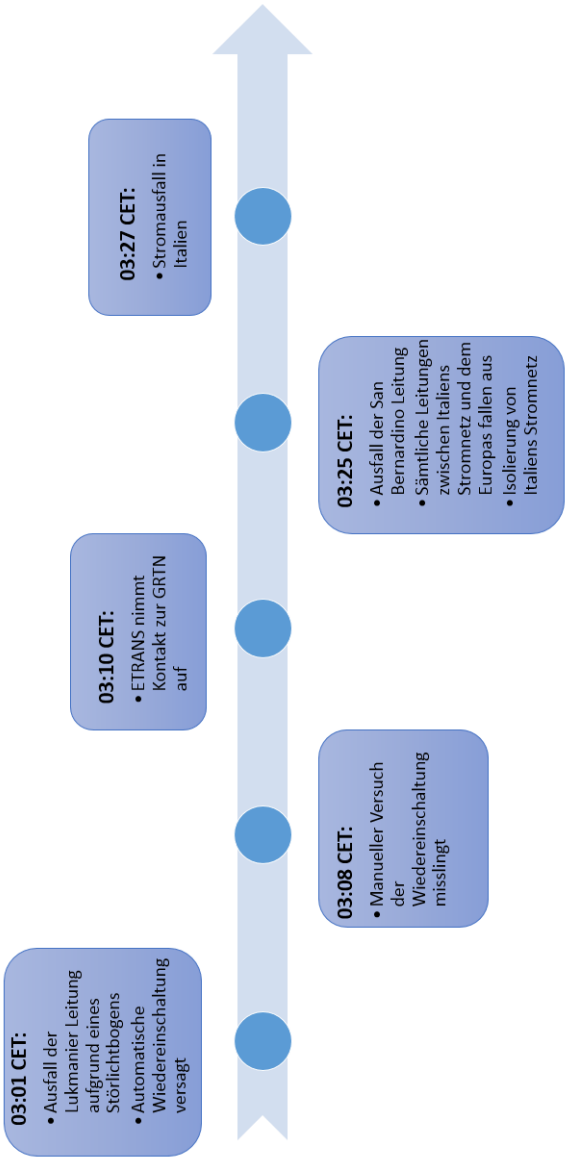


Abb. 2. Die Grafik zeigt eine zeitliche Darstellung der wichtigsten Ereignisse, die zu dem Stromausfall in Italien führten [13]

3.3 Vergleich der beiden Stromausfälle

Die beiden Stromausfälle unterscheiden sich durch den unterschiedlichen Ausbau der Präventionsmaßnahmen. Während in den USA das fehlende Load-Shedding und das Vernachlässigen des regelmäßigen Abfahrens der Stromleitungen als Hauptgründe zu sehen sind, geschah der Ausfall in Italien in erster Linie durch schlechte Kommunikation zwischen der Schweiz und Italien. Denn in Europa werden gewisse Standards durch die ENTSO-E (vormals UCTE) vorgeschrieben, welche alle Übertragungsnetzbetreiber erfüllen müssen. In Nordamerika gab es ebenfalls Vorgaben durch eine zentrale Organisation. Allerdings waren die Vorgaben der NERC [11] nur Empfehlungen. Dies hat sich in den vergangenen Jahren, auch als Reaktion auf den Stromausfall 2003, geändert. 2005 bekam die FERC[12] durch den Energy Policy Act 2005 die Möglichkeit die NERC zum nationalen Electric Reliability Organization zu ernennen.[17] Durch diese Stellung konnte die NERC ihre Empfehlungen zu Standards umformulieren.

[11] North American Electric Reliability Corporationst *ïst eine nordamerikanische Non-Profit-Organisation [...] welche für die Koordinierung der elektrischen Stromnetze im nordamerikanischen Raum zuständig ist"*[18]

[12] Federal Energy Regulatory Commission ist eine Unabhängige Behörden der Vereinigten Staaten

4 Prävention

Stromausfallprävention ist ein sehr weitreichendes Gebiet. Es befasst sich mit dem Aufbau des Netzes, der Belastung des Netzes, der Menge des produzierten Stroms und vielem mehr. Im Folgenden wird es eine Aufzählung verschiedener Präventionsmöglichkeiten mit kleineren Erklärungen geben.

Im Stromnetz treten Stromausfälle meist an den Flaschenhälsen auf. Während an anderen Stellen bei Ausfall einer Leitung der Strom auf andere Leitungen übertragen wird und diese auch ohne die zusätzliche Belastung nicht 100 Prozent Auslastung erreichen, kann dies beim Flaschenhals sehr schnell auftreten. Hier wird der Strom auf wenige Leitungen verteilt und diese sind dann besonders ausgelastet. Ein Ausfall ist hier besonders kritisch und so müssen besondere Maßnahmen ergriffen werden. Die folgenden Präventionsmaßnahmen sind strukturell gegliedert.

4.1 Stromnetzaufbaubetreffende Präventionen

Stromausfallprävention beginnt bereits beim Bau des Netzes. Hier gibt es verschiedene Methoden, um die Belastung des Stromnetzes einzudämmen.

Hochspannungs-Gleichstrom-Übertragung (HGÜ) HGÜ ist ein Verfahren, welches Gleichstromverbindungen nutzt. Dies wird aus mehreren Gründen im Stromnetz verwendet, zum einen ist es bei längeren Leitungen kostengünstiger Gleichstromleitungen zu verwenden. Es besteht aber auch ein Präventionsnutzen. Denn Gleichstromleitungen müssen nicht synchronisiert werden. Es kann also eine Leitung zwischen zwei Regionen bestehen, ohne dass die beiden Regionen synchronisiert sind. Der Vorteil, der sich daraus ziehen lässt, ist, dass sich Kaskaden häufig aufgrund von Asynchronität verbreiten. Diese Verbreitung wird durch die Gleichstromleitungen verhindert. Gleichstromleitungen können nicht überladen. Diese Eigenschaft wird genutzt um Teile eines Systems zu isolieren. Die Gleichstromleitungen werden als eine Art 'Firewall' schützend um ein Teilsystem gebaut. So kann die Überladung sich nicht auf den Teil übertragen.[19]

Second Line Design of System Das Second Line Design ist eine Designentscheidung. Es werden zwischen verschiedenen Generatoren oder Teilsystemen statt einer Leitung zwei gebaut. Das führt dazu, dass die Spannung sich verteilt und so Überlastungen vorbeugt.[20]

4.2 Stromnetzüberwachung

Um auf etwaige kritische Situationen reagieren zu können, ist es notwendig das Stromnetz zu überwachen. Dies ist ein sehr wichtiger Aspekt der Prävention.

Fault Diagnosis and Restoration Control Program or FDARC Das FDARC ist ein Programm, in C++ geschrieben und in China entwickelt, welches die Operatoren in kritischen Situationen unterstützen soll. Dabei soll das Programm erst aktiviert werden, wenn sämtliche automatischen Systeme gescheitert sind, und der Operator manuell einschreiten muss.
Dabei passt sich das System an die Art des Problems an und bietet entsprechende Informationen an. Hier handelt es sich also nicht um eine aktive Prävention, sondern um eine Unterstützung von Operatoren, die wiederum die eigentliche Prävention durchführen. Das Programm wurde 1997 entwickelt.[21]

Intelligente Alarmsysteme Ein Alarmsystem gibt einen Alarm als Warnung heraus. Dies geschieht, sobald sich auffällige oder unübliche Werte ergeben. Dieses System ist ebenfalls eine Hilfestellung für Operatoren. Wie weitreichend ein Versagen solch eines Systems sein kann, war 2003 bei dem Stromausfall in den USA ersichtlich. [22]

4.3 Automatische Präventionen

Verschiedene Mechanismen reagieren auf Veränderungen im Stromnetz. Dies verläuft meist automatisch mittels der Informationen, die mit den Überwachungssystemen gesammelt werden können. Die Mechanismen können aber auch manuell von den Operatoren ausgelöst werden.

Flexible-AC-Transmission-System, FACTS Das Flexible-AC-Transmission-System ist ein System mit dem der Stromfluss beeinflusst und gesteuert werden kann. Hier handelt es sich um eine sehr frühe Art der Prävention. Mittels FACTS wird versucht Flaschenhälse oder kritische Situationen, mittels Umverteilung von Strom, gar nicht erst entstehen zu lassen. Denn da Stromausfälle häufig aufgrund von Überlastung auftreten, kann durch Beeinflussung des Stromflusses ein Ausfall von Leitungen und Generatoren verhindert werden. [23]

Automatische Wiedereinschaltung (AWE) Die Automatische Wiedereinschaltung ist eine Funktion, die bei Ausfällen von Hochspannungsleitungen verwendet wird. Wie im Grundlagenkapitel beschrieben wird bei einem Transienten die Leitung aus Sicherheitsgründen vom Netz genommen. Nun kann es vorkommen, dass der Transient durch ein temporäres Problem ausgelöst wird, wie zum Beispiel einen Ast, der auf die Leitung gefallen ist. Wenn nun die Leitung vom Netz genommen wurde, wird die AWE aktiv. Sie versucht nach einer kurzen Zeitspanne, ca. 300ms, die Leitung wieder an das Netz anzuschließen. Durch die direkte Wiedereinschaltung kann Umverteilung des Stroms verhindert und einer Überlastung auf anderen Stromleitungen vorgebeugt werden. [24]

Load Shedding *„Der Lastabwurf bezeichnet das kurzzeitige Ausschalten von Maschinen, Klimaanlagen, Motoren, Öfen, Kompressoren, Pumpen oder der Beleuchtung. Ziel ist es, Ausfälle auf der Erzeugungsseite [...] auszugleichen.“* [25]

Das bedeutet, dass wenn es zu einem Ausfall eines Generators kommt, kommt es zwangsläufig zu einem Defizit an Strom. Dieses Defizit wird nun versucht per Abschalten von Verbrauchern auszugleichen. Dabei werden die Verbraucher üblicher Weise in Gruppen eingeteilt. Bei einem Spannungsverlust wird zuerst die erste Gruppe ausgeschaltet. Sollte dies keinen Ausgleich herstellen wird die zweite Gruppe ausgeschaltet. Dies wird fortgeführt bis das Defizit ausgeglichen ist. Auf diese Weise soll die Anzahl der auszuschaltenden Verbraucher reduziert werden. Der Lastabwurf ist dabei die letzte Instanz der Prävention, ein Notschalter, der nur als letztmöglicher Ausweg gezogen wird. Load Shedding wird heutzutage automatisch gesteuert. Dabei werden Überwachungssysteme wie das SCADA verwendet. [26]

Load Shedding gibt es auch für Frequenzen. Das Under Frequenxy Load Shedding (oder kurz UFLS) versucht auf gleiche Weise einen Frequenzabfall auszugleichen. Allerdings ist der Anwendungsfall hier etwas anders. Das UFLS wird eingesetzt um die Balance zwischen Stromgenerierung und Stromladung wiederherzustellen. Dies kann nur bei isolierten Stromnetzen angewandt werden. Die Frequenz wird dabei der Frequenz der Generatoren angepasst.[27] UFLS funktioniert nicht, wenn es innerhalb des Stromnetzes Instabilität undSspannungseinbrüche gibt. [16]

Preventive Short-Term Sectioning Beim Preventive Short-Term Sectioning wird das Stromnetz bewusst in Inseln unterteilt. (Abb.3, I) Die Idee dahinter ist, dass man die Überladung, die sich ausbreitet, isoliert. Dies geschieht über das gleichzeitige Abschalten von Leitungen, welche sich nahe der Quelle befinden. Durch das Abschalten der Leitungen und das Isolieren der Überladung entstehen zwei Teilgebiete, eins mit Stromdefizit und Frequenzabfall und eins mit Stromüberschuss und einer Frequenzsteigerung. Im Teilgebiet mit Defizit wird durch fast-acting AUFLSI[13] (Abb.3, II) und danach slow-acting AUFLS2[14] (Abb.3, III) die Frequenz wieder auf das normale Level gehoben. Durch sogenannte turbine governors (zu deutsch Turbinen-Regler) wird der Überschuss wieder reguliert. (Abb.3, IV) Alternativ kann auch forced regulations (gezwungene Regulierung) eingesetzt werden. Diese arbeitet wesentlich schneller. Nachdem die beiden Teilgebiete wieder angepasst wurden, werden sie zusammengeführt (Abb.3, V) und der alte Zustand wird hergestellt. (Abb.3, VI) [28]

[13] fast automatic under-frequency load shedding
[14] slow-acting under-frequency load shedding automatics

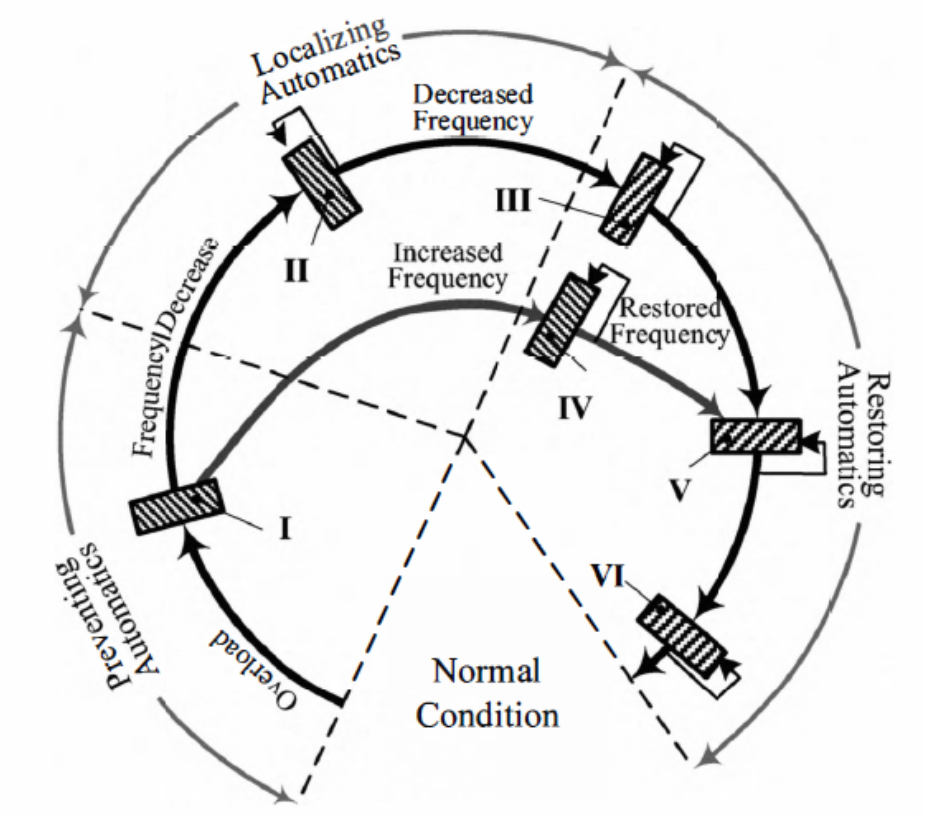


Abb. 3. Diese Grafik stellt das Preventive Short-Term Sectioning dar [28]

4.4 Kombination von Überwachung und automatischer Prävention

Es gibt Systeme, welche die bereits vorgestellten Präventionsmöglichkeiten in sich vereinen und sowohl Überwachung als auch das Ausführen der eigentlichen Prävention automatisch durchführen.

N-1 Regel An dieser Stelle sei noch einmal die N-1 Regel erwähnt, welche ebenfalls, wie bereits oben beschrieben, einen Präventionsgedanken verfolgt.

On-Line Dynamic Security Assets (DSA) Das DSA ist in verschiedene Phasen unterteilt. Es beginnt mit dem Measurement (Vgl. Abb.4). Um DSA auszuführen, muss als erstes eine Momentaufnahme des Systems gemacht werden. Diese wird heutzutage mit einem SCADA, dem Supervisory Control And Data Acquisition System, gemacht.
Danach folgt das Modelling (Vgl. Abb.4). Damit die Momentaufnahme weiter genutzt werden kann, muss diese vorbereitet werden. Der EMS State-Estimator baut ein Grundmodell, welches verwendet werden kann. Da das SCADA sich aber auf die zu untersuchende Region beschränkt und die Nachbarregionen nicht betrachtet werden, muss das System noch mit eben diesen Informationen gefüttert werden. Des Weiteren werden sogenannte auxiliary data hinzugefügt. Hier handelt es sich um systeminterne, dynamische Daten, wie zum Beispiel Generatorenzustände. Diese Daten werden nun zu einem großen Systemmodell zusammengefügt und weitergegeben.

Danach werden in der Computation-Phase (Vgl. Abb.4) anhand dieses Modells Analysen durchgeführt, um die Sicherheit des Systems zu überprüfen. Ist die Sicherheit in Gefahr, werden Maßnahmen, wie generator rescheduling oder load shedding, direkt vom System durchgeführt (Control-Phase (Vgl. Abb.4)). Des Weiteren werden die Daten visualisiert und den Operatoren zur Verfügung gestellt (Reporting and Visualization (Vgl. Abb.4)). Das gesamte Verfahren wird in regelmäßigen Abständen, meist zwischen 10 und 30 Minuten, durchgeführt.

Damit das DSA zuverlässig funktionieren kann, braucht es weitere wichtige Tools, welche folgende Sicherheiten gewährleisten müssen.
Das Voltage Security Assessment (VSA) muss in der Lage sein aktuelle Spannungsverläufe darzustellen und messen zu können. Es muss weiterhin Berechnungen durchführen können um sicher zu stellen, dass das System sich nicht in einem kritischen Zustand befindet. Bei einem kritischen Zustand muss das VSA in der Lage sein, etwaige Aktionen durchzuführen, um das System wieder in einen Secure State zu bringen.
Das Transient Security Assessment (TSA) muss sehr schnell seine Analysen durchführen können. Um dies zu gewährleisten wurden drei Methoden entwickelt, die schnelles Reagieren ermöglichen.
1. Mit sogenanntem Contingency Screening lassen sich große Mengen an möglichen Systemzuständen filtern und nur kritische Zustände betrachten.
2. Die üblichen Analysen werden mit Indizes versehen, welche Stabilität oder Instabilität eines Systems bereits vor dem Ende der Simulation zeigen können.

3. Durch die Verwendung von Mehrkernprozessoren können die Berechnungen beschleunigt werden.
Ein weiterer wichtiger Punkt von TSA's ist, dass dem Operator ermöglicht wird zukünftige Zustände des Systems zu betrachten.
Das Small Signal Stability Assessment (SSA) hat die Aufgabe zu verhindern, dass kleine Störungen im Stromnetz zu großen Störungen werden. Dabei berechnet es den Worst-Case bei einer Störung und versucht diesen zu verhindern. [29]

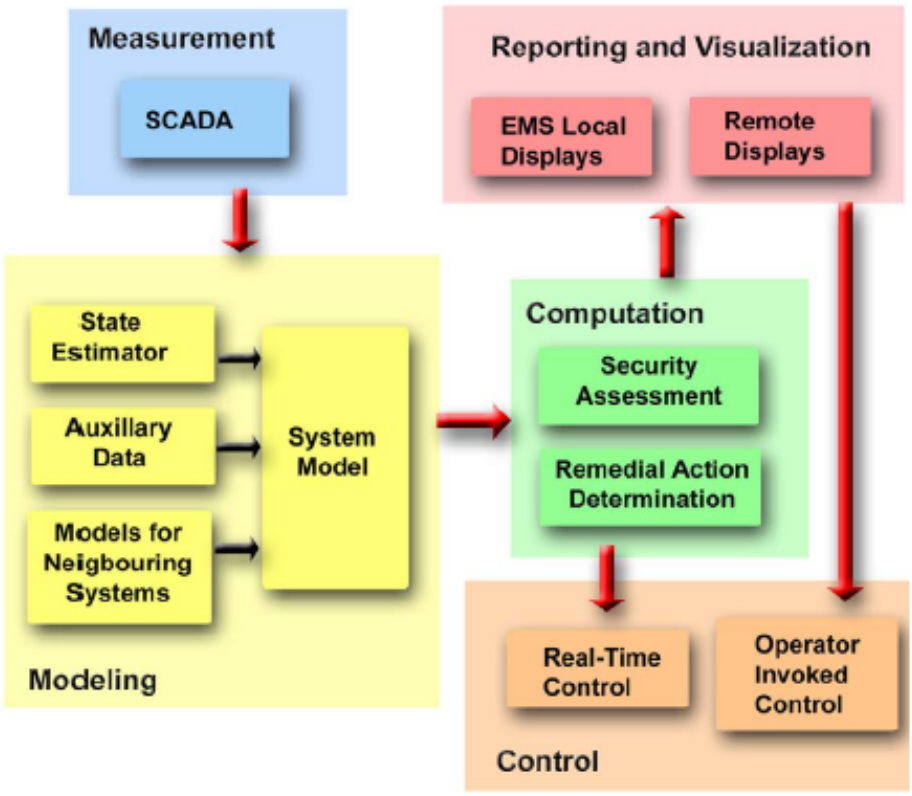


Abb. 4. Diese Grafik stellt das On-Line Dynamic Security Assets grafisch dar [29]

Dynamic Protection Security Assets (DPSA) Eine Erweiterung des DSA ist das DPSA. Hier wird zusätzlich zum eigentlichen DSA System ein Protection System integriert. Diese Protection Systeme haben die Möglichkeit, die Netzwerk

Topologie während der Simulation des Netzes zu ändern und dadurch Kaskadensimulation zu ermöglichen. [30]

5 Konklusion

Die Vielfältigkeit der verschiedenen Präventionsmaßnahmen hat in den letzten Jahren zugenommen. Dies wird sich auch in der Zukunft in diese Richtung weiter entwickeln, da wir von einem stetig ansteigenden Energieverbrauch ausgehen müssen [1]. Sowohl die Überwachung des Stromnetzes als auch aktive Reaktionen auf Ausfälle bestimmter Bestandteile des Stromnetzes sind unverzichtbar. Präventionen wie das Load Shedding oder das Preventive Short-Term Sectioning sind von immenser Wichtigkeit. Mit beiden kann man schnell auf Ausfälle reagieren und Verbreitungen einschränken. Allerdings führen beide zu temporären Ausfällen im Stromnetz, daher werden sie erst im absoluten Notfall genutzt. Wichtiger ist es bereits vorher, mittels Überwachungsmaßnahmen, kritische Situationen zu erkennen und diese zu vermeiden. Komplexe Systeme wie On-Line Dynamic Security Assets oder einfache Alarmsysteme sind ebenfalls von großer Notwendigkeit bei der Prävention von Stromausfällen. Dass das manuelle Eingreifen durch den Menschen weiterhin nötig ist, hat man beim Stromausfall in Italien gesehen. Auch wenn hier die Kommunikation nicht schnell genug war, so wäre durch schnelleres Reagieren der Operatoren der Stromausfall vermeidbar gewesen.
Stromausfallprävention wird weiter einen zentralen Punkt in dem Betreiben von Stromnetzen einnehmen. Man muss sich lediglich vor Augen führen, welche Vorgänge heute ohne Strom nicht mehr möglich wären. Bei der Arbeit, bei der Fortbewegung, bei der Freizeitgestaltung, beinahe in allen Lebenslagen ist Strom ein entscheidender Faktor unseres Lebens. Unsere Abhängigkeit ist so stark gewachsen, dass ein stundenlanger Stromausfall katastrophale Folgen haben kann. Aus diesem Grund wird Stromausfallprävention stets unverzichtbar bleiben und weiter entwickelt werden.

Literatur

1. http://www.eia.gov/pressroom/releases/press395.cfm (10.09.2014)
2. http://en.wikipedia.org/wiki/List_of_major_power_outages#Largest (10.09.2014)
3. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 59
4. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 73
5. http://de.wikipedia.org/wiki/N-1-Regel (10.09.2014)
6. Final Report of the Investigation Committee on the 28 September 2003 Blackout in Italy der UTCE, Seite 57
7. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 219
8. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 48-49
9. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 53
10. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 49-50
11. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 70
12. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 75-77
13. Eigene Grafik
14. Final Report of the Investigation Committee on the 28 September 2003 Blackout in Italy der UTCE, Seite 28
15. Final Report of the Investigation Committee on the 28 September 2003 Blackout in Italy der UTCE, Seite 38
16. Final Report of the Investigation Committee on the 28 September 2003 Blackout in Italy der UTCE, Seite 7
17. http://www.gpo.gov/fdsys/pkg/PLAW-109publ58/pdf/PLAW-109publ58.pdf 119 STAT. 942 ff. (11.09.2014)
18. http://de.wikipedia.org/wiki/North_American_Electric_Reliability_Corporation (11.09.2014)
19. Assessment of HVDC Grid Segmentation for Reducing the Risk of Cascading Outages and Blackouts von Omid Alizadeh Mousavi, Lazar Bizumic, Rachid Cherkaoui
20. New Techniques for the Prevention of Power System Collapse von F. A. Shaikh, Ramanshu Jain , Mukesh Kotnala, Nickey Agarwal
21. IMPLEMENTATION OF AN ON-LINE INTELLIGENT SYSTEM TO PREVENT A SYSTEM BLACKOUT von Fu Shuti, Chen Jingcheng, Chen Kaiyong, Zeng Zhaoqi, Hu Xiang

22. Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations der U.S.-Canada Power System Outage Task Force, Seite 53
23. FACTS and HVDC Technologies for the Development of Future Power Systems von X.-P. Zhang, L.Yao, B. Chong, C. Sasse, and K.R. Godfrey
24. Elektroenergiesysteme. Erzeugung, Transport, Übertragung und Verteilung elektrischer Energie von Adolf J. Schwaab erschienen im Springer Verlag im Jahre 2009.
25. http://www.siemens.com/innovation/apps/pof_microsite/_pof-spring-2013/_html_de/lastabwurf-fuers-stromnetz.html (10.09.2014)
26. New Techniques for the Prevention of Power System Collapse von F. A. Shaikh, Ramanshu Jain , Mukesh Kotnala, Nickey Agarwal
27. http://www.nerc.com/pa/RAPA/ri/Pages/UnderFrequencyLoadShedding.aspx (10.09.2014)
28. NEW CONCEPT AND SOLUTIONS FOR PREVENTION OF POWER SYSTEM BLACKOUTS von J. Barkanst, D. Zalostiba
29. New Tools for Blackout Prevention von Kip Morison, Lei Wang, Hamid Hamadani
30. Dynamic Protection Security Assessment, a Technique for Blackout Prevention von Christian Romeis, Johann Jaeger

Authenticated Data Structures - A Solution for Data Protection Aspects in Crowdsourcing?

Juliane Köckert*

Advisor: Anja Bachmann†

Karlsruhe Institute of Technology (KIT)
Pervasive Computing Systems – TECO
*juliane.koeckert@student.kit.edu
†bachmann@teco.edu

Abstract. This paper examines different types of authenticated data structures like the hash tree scheme, skip lists and commutative hashing as well as the RSA accumulator on their ability to solve data protection issues in crowdsourcing. The different types will each be explained in their general method and an implementation of them as authenticated data structure will be introduced. After a comparison of these types, a conclusion on how suitable they are for the problem statement will be made.

Keywords: Authenticated Data Structures, Crowdsourcing, Data Protection

1 Introduction

Nowadays, many companies rely on crowdsourcing to do research. Whether it is simply marketing (which is kind of been done by every enterprise) or rather scientific will not be distinguished in this paper. One of the first examples for crowdsourcing is the Oxford Dictionary. In the late-nineteenth century, Professor James Murray asked thousands of people to send him their definitions of certain English words. The process endured over centuries but at the end he gathered a great collection of words and adjoining definitions to put into the dictionary [2]. Back then, no one was thinking about data protection, because it was not an issue at that time. That has changed and to do his collection Murray would not have gotten paper slips on which the definitions were written. It is more likely that he would have received emails, put up an online platform or developed an application. Maybe the users of this application would not want that he will get any information about themselves through this application. But how could this problem been solved? One way for crowdsourcing could be that there is a trusted middleman to whom the users send their data. Then the middleman clears the data of all personal information that could lead to the identity of the user. After this has been done this middleman transfers the data to Murray who would put the information in his dictionary.

It can be seen that this is not the best way to transfer data, because the user still needs to trust someone they probably do not really know. Thus, the aim of this paper is to have a closer look on so-called authenticated data structures in order to find out if they could be a way of solving such data protection issues. To introduce the topic, there will be an introduction into crowdsourcing and authenticated data structures in general in Section 2. It will be followed by taking a closer look at some (hash trees, skip lists and commutative hashing as well as the RSA accumulator) types of authenticated data structures in Section 3 and a discussion on how suitable for the given problem of this paper they are in Section 4. Section 5 will conclude the whole matter.

2 Background

Generally speaking, *crowdsourcing* is a concept of doing research respectively gather a great variety of information about a certain topic. Instead of doing this by assigning it to the employees in the research department of a company, it is 'outsourced'. This means the task of gathering the information is taken to the 'crowd', thus the word 'crowdsourcing' was created. The 'crowd' can be understood as any not too precisely defined large group of people that is asked to provide information on the given topic [6].

Figure 1 shows the development of crowdsourcing and that it becomes more and more important and therefore, to protect the data of the users, that are willing to take part in crowdsourcing, plays an increasingly significant role.

Now a more today example for crowdsourcing will be given. There is an organisation in Germany called *NABU*. Every year they have a project called "Stunde der Gartenvögel" (hour of the garden birds) [8]. Aim of this project is to count all the garden birds that are seen within an hour in a chosen area and to report the results to the NABU. The participants of this project can do so via mail, via an online platform, via telephone or through a smart phone application. [9] The application is what will be the interesting part for this paper. The participant who uses the application may want to provide the information about the birds, but probably does not want the NABU to be able to reconnect it with them as a person. The purpose of this paper will be to examine if and how authenticated data structures can solve such a problem.

Now that it is clear what crowdsourcing means, it will be taken a closer look on authenticated data structures in general. Briefly speaking there are three parties involved, e.g. a *responder* which is not trustworthy, a *source* which is reliable and a *user*. In addition there is a "structured collection S of objects" [11] which defines a set of *query operations* and optional *update operations*. Each of the three parties performs a different task in the process [11]:

- The *source* contains all information about S and its original version. It produces *structure authentication information* consisting of a statement about the latest version of S which is signed and has a timestamp.
- The *responder* manages a copy of S. It gets all required information from S like the latest update along with the matching *structure authentication*

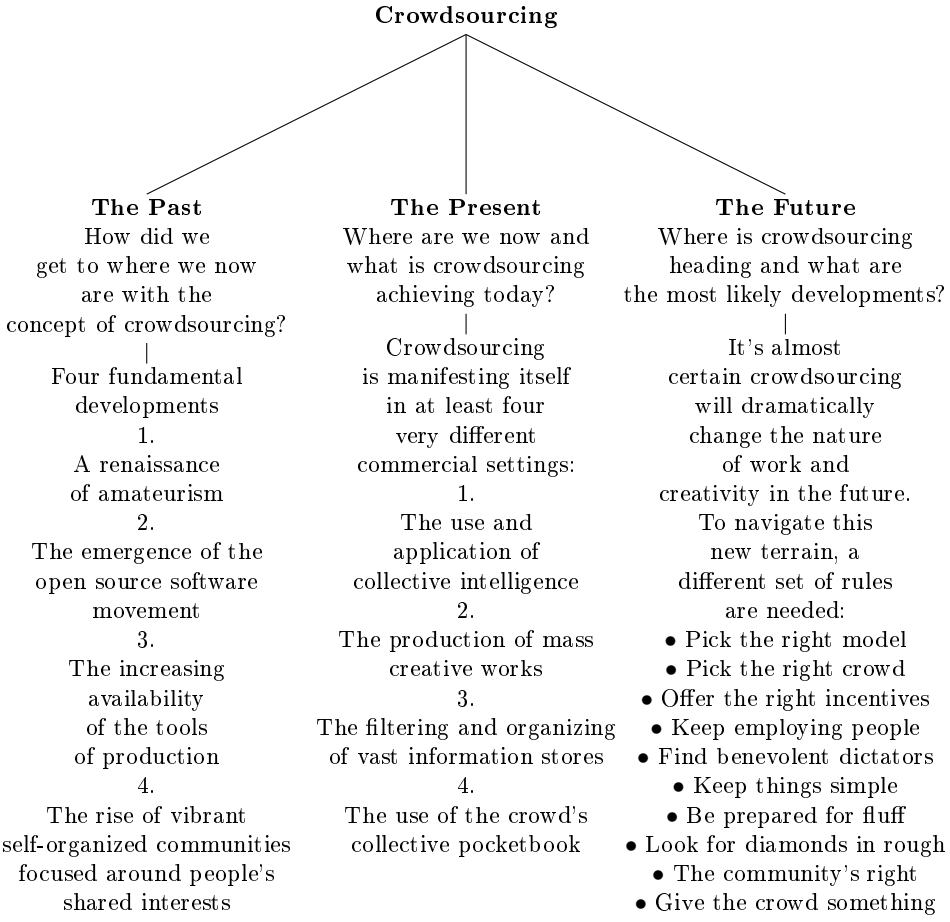


Fig. 1. The development of crowdsourcing over time. A shortened version of [6].

information. As the name "responder" implies it also interacts with the *user* by responding queries posed by the *user* through S. Furthermore it provides *answer authentication information*, consisting of "the latest structure authentication information issued by the source [...] and a *proof* of the answer".
- The *user* utilises S to ask questions without directly contacting the *source*. Instead it interacts with the *responder*. As already mentioned the *responder* is not trustworthy. So in order to verify the returned information the *user* needs the accompanied *answer authentication information*.

Now this needs to be matched to the NABU example used in this paper. If authenticated data structures are a way to solve data protection issues in crowdsourcing it could be like this: The source could be the group of participants that takes part in the project, meaning that they provide the information which serves as the source. The responder would probably be any server to be able to perform the task of passing on the information. Finally the user would be the NABU that wants to gather all the information the participants provided as the source. A schematic display of the example is shown in figure 2.

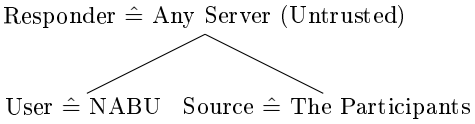


Fig. 2. Schematic Display of the NABU Example set into an Authenticated Data Structures Context.

Thus, the problem statement of this paper, is to examine the chosen types of authenticated data structures regarding their ability to solve the data protection aspects in this setting.

In this paper the efficiency of the introduced kinds of authenticated data structures will not be distinguished, because for this work it is only interesting to examine them regarding their ability to solve the problem statement.

3 Different Types of Authenticated Data Structures

In this section different types of authenticated data structures in general will be introduced and explained. Afterwards their corresponding implementations will be presented.

3.1 Static Authenticated Dictionaries

As already mentioned, there are different ways to create an authenticated data structure. Roberto Tamassia introduces some in his paper on authenticated data

structures [11]. According to his work, the first motivation for working on something like that came from the problem of *certificate revocation* in public key infrastructures. It was focused on *authenticated dictionaries*, which were used to perform membership queries [11]. An authenticated dictionary, as schematically shown in Figure 3, provides the possibility to perform queries on data storage with verifiable integrity, especially when this data is located on an untrusted server [1].

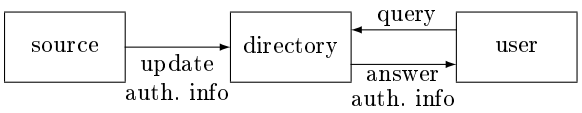


Fig. 3. Authenticated dictionary [3].

There are three main goals to be appreciated when setting up an authenticated dictionary [3]:

- *low computational cost*: This is achieved best by making all internally performed computations by each of the three parties in Figure 3 plain and quickly. Another important point is to make sure that the necessary memory space to carry out the computations needed is as slight as possible.
- *low communication overhead*: Every kind of information whether it is *update authentication information* or *query authentication information* is to be kept at the minimal possible size.
- *high security*: This means, that the reliability to be sure that the received answers are authentic is as high as possible.

Goodrich and Tamassia formalize these goals in [3] as an algorithmic problem. This problem is to minimize six different cost parameters of such an authenticated dictionary. First, is the *space* that the data structure is applied to. Secondly, it is the *time* the directory needs to fulfil an update which was started by the source. Thirdly, the *size* of the update authentication information which was sent by the source during an update, the so-called "source-to-directory communication". Fourthly, the *time* the directory needs to respond to a query and return the query authentication information to prove that the answer is reliable. Fifthly, the *size* of the answer from the directory sent along with the query authentication information. Last but not least, the *time* the user needs to verify a given answer to a query [3].

Furthermore there are two different kinds of authenticated dictionaries *static* and *dynamic*. Each type needs another way to be implemented, which will be presented now.

A way to implement *static* authentication dictionaries is the *hash tree* scheme by Merkle [7]. The concept of a hash tree T is that it is developed from the

bottom up. This means that there are leaves which equal data blocks. Each data block/leave has an assigned hash value forming one internal node of the tree structure. Two of these hash values are children of another hash value which is part of another set of children and so on, until the *top hash* is reached. [5] An example of this structure is given in Figure 4.

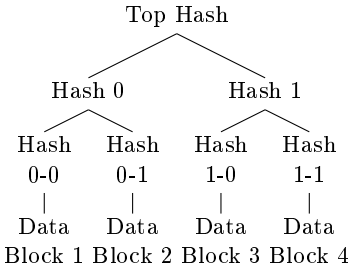


Fig. 4. Example of a simple hash tree.

Aim of this method is to have only one digest to verify the correctness of an answer. This is done by only using the "top hash". By this means the user is able to be sure that the given answer is reliable without the responder getting too much knowledge about the original information in the source. [5]

3.2 Dynamic Authenticated Dictionaries

To implement *dynamic* authenticated dictionaries other methods become necessary. One way to accomplish that are *skip lists* and *commutative hashing* like introduced by Goodrich and Tamassia in [3]. At first, a closer look will be taken on skip lists.

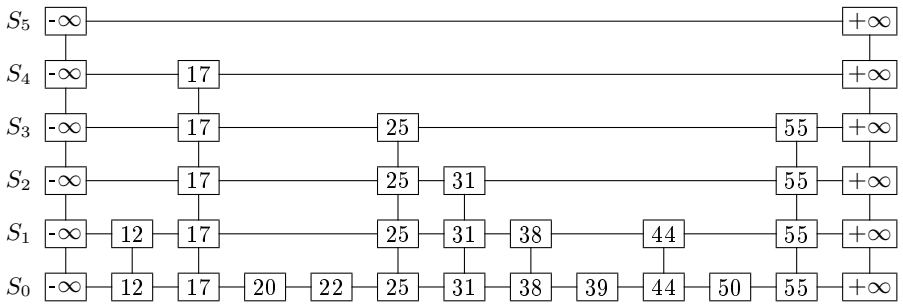


Fig. 5. Example of a skip list [3].

"The *skip list* data structure is an efficient means for storing a set S of elements from an ordered universe" [3]. There are three operations that can be executed by a skip list. **find(x)** to see if element x is part of S. **insert(x)** to insert an element x into S. **delete(x)** to delete an element x from the set S. [3]

A skip list, as exemplarily shown in Figure 5, files a set S of elements in a a couple of linked lists $S_0, S_1, S_2, ..., S_t$. S_0 is the so-called base list. In it all elements of S in their specific order are stored along with sentinels and their adjoining special elements $-\infty$ and $+\infty$. Each list S_i, for $i \geq 1$, contains a sample of the elements of S_{i-1}. When nothing is defined precisely, the above list simply chooses every element of S_{i-1} by a chance of $\frac{1}{2}$ to be in list S_i, which would be a *randomized skip list*. Another method would be to define a rule that "between any two elements in S_i there are at least 1 and at most 3 elements of S_{i-1}" [3] which would be a *deterministic skip list*. As shown in Figure 5, the sentinel elements are always a part of the above list and the top level list only contains these two elements. Therefore it is possible to define a *start node s* which is the one filing $-\infty$ in the top list S_t.

Furthermore there are two different types of nodes. The plateau and the tower nodes. In Figure 5, the plateau nodes are the ones without nodes in any higher level list: 20, 22, 39 and 50. The tower nodes are the ones building a 'tower': 12, 17, 25, 31, 38, 44 and 55. The architecture of a skip list does not allow any lists without plateau nodes. The only thing that differs is the number of plateau nodes according to the type of the skip list. *Deterministic*: There must be at least one and at most three plateau nodes between two tower nodes. *Randomized*: There has to be exactly one plateau node between two tower nodes. [3]

Further definitions for skip lists are the following list:

- v: A node in list S_i
- *elem(v)*: An element filed at v
- *down(v)*: A node under v in S_{i-1}, storing the same element as v, except when $i = 1$, then $down(v) = null$
- *right(v)*: Every node right of v in S_i, until far right node respectively the sentinel filing $+\infty$ is reached, then $right(v) = null$

Now that skip lists have been defined, *commutative hashing* will be introduced.

In general a cryptographic hash function is understood "as a function that takes two integer arguments, x and y, with the same number of bits and maps them to an integer $h(x; y)$ that is represented using a fixed number k of bits (typically fewer than the number of bits of x and y)" [3]. This method also works for larger sequences, meaning it is possible to hash $(x_1; x_2; ...; x_m)$ into $h(x_1; h(x_2; ...h(x_{m-2}; h(x_{m-1}; x_m))...))$. [3]

An important attribute of cryptographic hash functions is that they are collision resistant. This means that it is very hard and up to impossible to find a pair $(c; d) \neq (a; b)$ that fulfils the following $h(a; b) = h(c; d)$. In order to guarantee this one has to make sure that the time it would take a supercomputer to find such a pair is incredibly long. If h fulfils this requirement it is known as

collision resistant. In order to simplify this requirement in the verification process for any user of such an authenticated dictionary, Goodrich and Tamassio define collision resistance in the following way: "a hash function is commutatively collision resistant if, given $(a; b)$, it is difficult to compute a pair $(c; d)$ such that $h(a; b) = h(c; d)$ while $(a; b) \neq (c; d)$ and $(a; b) \neq (d; c)$" [3]. Furthermore they construct the following commutative collision resistant cryptographic hash function, h:

$h(x; y) = f(min\{x; y\}; max\{x; y\})$.

A proof of why this function is collision resistant is given in [3]. Now in order to develop an authenticated dictionary Goodrich and Tamassia combine skip lists and commutative hash functions in the following way. Given a commutative cryptograhpic hash function h and an authenticated dictionary for a set S, it has three components [3]:

1. The skip list serves as the data structure in which the elements of S are stored.
2. Every node v of the skip list is labelled through a collection of values $f(v)$. With the help of the hash function h the items of S are computed accumulatively.
3. This equals the already mentioned "authentication information", as it is a signed statement from the source containing "of the timestamp of the most recent label $f(s)$ of the start node of the skip list"

The hash h is used to compute the label $f(v)$ of every node v in the skip list, without the sentinel nodes. Like the function of the top hash in hash trees, the start node stores a value $f(s)$ representing the whole skip list. How each label $f(v)$ accumulates the labels of nodes linked to it, is shown in Figure 6. In fact this implementation relates to a directed acyclic graph which is not to be confused with a tree structure. [3]

As one can see now this relates to the already introduced "hash tree scheme" but in a rather *dynamic* way.

Now that the structure is clear, it will be explained how the operations of authenticated data structures are performed in skip lists with commutative hashing. The source is in charge of updating the authenticated directory every time when something in the original version of the authenticated dictionary has changed. This means that it performs one of the operations *insertion* or *deletion* on a specified item x and provides the following authentication information. The new hash value of the start node s along with a timestamp, formed into a signed statement. If it is the case that the skip list is a randomized one, the source also sends the random bits it used to perform the update. After such an update was issued by the source all hash values in the skip list need to be updated accordingly. [3]

After the process of an update operation has been explained, the query and verification operation will be introduced. Briefly, one can say that the only purpose of this operation is to find out if a given element x is part of the skip list or not. In commutative hash functions this is done by either returning "yes" if

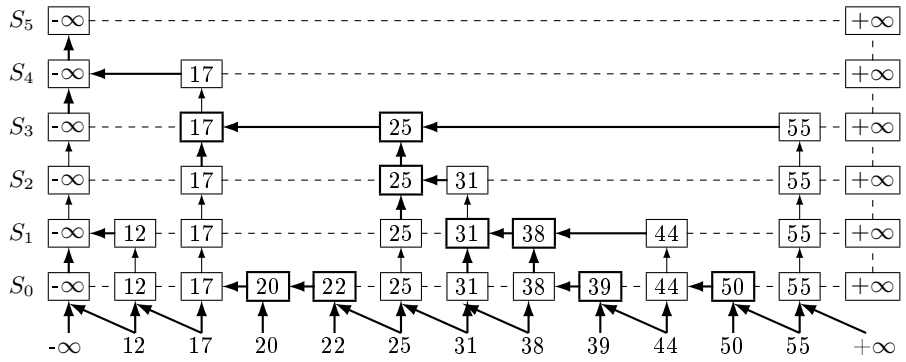


Fig. 6. "Flow of the computation of the hash values labeling the nodes of the skip list of Figure 5. Nodes where hash functions are computed are [the ones a thick arrow is pointing on]. Also, please note that the arrows denote the flow of information, not pointer values in the data structure." [3]

the element exists or showing the opposite by comparing the element x to two consecutive elements x' and x'' in the bottom level of the skip list. Thus, showing that $x' < x < x''$. [3]

3.3 The RSA Accumulator

Another option in contrast to authenticated dictionaries is to use the *RSA accumulator*. The RSA cryptosystem is a public key cryptosystem introduced by Rivest, Shamir and Adleman in a paper published 1977. It discusses the first interactions with "electronic mail" and the adjoining security problems [10]. The RSA accumulator is a method that leads to constant proof size and verification time. This efficient dynamic and distributed cryptographic accumulator is introduced in [4]. In this paper again three parties are mentioned: (1) the trusted information source, (2) the untrusted directory and (3) the user.

(1) It defines a finite set S of items which is constantly updated by performing *insertions* and *deletions* of elements. In the directories copies of set S are maintained. As in the structures before the directories get time-stamped update information from S containing its current items. (2) It acts as the "middleman", answering the queries posed by the user about the elements of S by "saying" yes or no. Along with this answer a *query authentication information* consisting of a proof is been sent. This proof is established by matching statements which were previously signed by the source. (3) The user poses membership queries about S by asking if a certain element e is in the set of S through the directory maintaining S. In order to verify the proof, the user needs to rely on its trust in the source and the availability of public information about the source that makes it possible to verify the signature of the source. [4]

This data structures can be understood as an authenticated data structure.

After getting an overview of different types of authenticated data structures in general, in the next section they will be compared to each other and discussed whether one is more suitable than the other.

4 Comparison and Discussion of the Different Types

In this section the types of Authenticated Data Structures which were introduced in the previous section will be compared to each other.

	Hash Trees	Skip Lists and Commutative Hashing	RSA Accumulator
Components	trusted source, untrusted responder/directory, user/client	trusted source, untrusted responder/directory, user/client	trusted source, untrusted responder/directory, user/client
Updates	only the top hash is updated directly	only the start node is updated directly	the directories filing S are constantly being updated by the source with the current version of S
Queries	if element x is in the directory return "yes" along with a verification of the answer	if element x is in the directory return "yes" along with a verification of the answer	if element e is in set S return "yes" along with a verification of the answer

Table 1. Comparison of the Different Types of Authenticated Data Structures

As shown in Table 1 all of the introduced approaches on authenticated dictionaries are very similar. The question is, if they can provide what is needed for the problem with crowdsourcing. Especially in the third implementation it is clear that one needs to trust the source. The problem with that is, that in the given scenario the participants of the project "are" the source. This leads to the question, if the NABU trusts its participants/source, but furthermore if the participants are willing to trust the instance on which they provide their information to create said trusted source. Which leads to the following conclusion.

5 Conclusion

In order to provide the data the participants still need to put their trust in the database on which their provided information will be stored. But does that make any difference from the given implementation with a trustworthy middleman clearing the data of all personal information? In this paper one comes to the conclusion that the answer is no. Thus, if the introduced structures are used it is

not possible to solve data protection issues in crowdsourcing without a trusted middleman.

References

1. Crosby, S.A., Wallach, D.S.: Authenticated dictionaries: Real-world costs and trade-offs. ACM Transactions on Information and System Security 14(2) (2011)
2. Geere, D.: How the oxford english dictionary started out like wikipedia (January 2011), http://www.wired.co.uk/news/archive/2011-01/13/the-oxford-english-wiktionary, last viewed: 29.06.2014
3. Goodrich, M.T., Tamassia, R.: Efficient authenticated dictionaries with skip lists and commutative hashing (October 2001), http://cs.brown.edu/cgc/stms/papers/hashskip.pdf, last reviewed: 04.07.2014
4. Goodrich, M.T., Tamassia, R., Hasic, J.: An efficient dynamic and distributed cryptographic accumulator. ISC (2002), http://cs.brown.edu/cgc/stms/papers/isc2002.pdf, last reviewed: 06.07.2014
5. Goodrich, M.T., Tamassia, R., Triandopoulos, N.: Efficient authenticated data structures for graph connectivity and geometric search problems (September 2009)
6. Howe, J.: Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business. The International Achievement Institute (2008)
7. Merkle, R.: Secrecy, Authentication and Public Key Systems. Ph.D. thesis, Stanford University (June 1979)
8. NABU: Stunde der gartenvögel, http://www.nabu.de/aktionenundprojekte/stundedergartenvoegel/, last viewed: 30.06.2014
9. NABU: Stunde der gartenvögel infomaterial, http://www.nabu.de/imperia/md/content/nabude/vogelschutz/stundedergartenvoegel/gartenvoegel2014_code7000.pdf, last viewed: 30.06.2014
10. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems (1977), http://people.csail.mit.edu/rivest/Rsapaper.pdf, last reviewed: 02.07.2014
11. Tamassia, R.: Authenticated data structures. In: Battista, U.Z.G.D. (ed.) Algorithms — ESA 2003. Budapest, Hungary (September 2003)