# DIPLOMARBEIT

## Using Organic Computing to Control Bunching Effects

von

**Oliver Ribock**

**eingereicht am 22.10.2007 beim**

**Institut für Angewandte Informatik**
**und Formale Beschreibungsverfahren (AIFB)**
**der Universität Karlsruhe (TH)**

**Referent: Prof. Dr. Hartmut Schmeck**
**Betreuer: Dipl.-Wi.-Ing. Urban Richter**

| Studienanschrift: | Heimanschrift: |
|---|---|
| Kreuzstraße 33 | Waxensteinstraße 11 |
| 76133 Karlsruhe | 82140 Olching |

E-Mail: oliver@ribock.de

# Acknowledgement

Above all, I would like to thank my parents for making my studies possible by their continuous financial support and their moral assistance in critical moments.

I thank Prof. Dr. Hartmut Schmeck for giving me the opportunity to conduct my work. I also wish to thank Urban Richter, advisor of my diploma thesis, for his guidance, inspiration, and patience during the development of my work. Furthermore, I gratefully acknowledge all the valuable suggestions by the participants of the graduands course.

I am especially indebted to Andreas, Anne, Elli, Kaj, Michael, Oliver, and Urban for proofreading my work and for giving numerous advices. I appreciate their effort which is by no means a matter of course.

# Contents

# List of Figures

# List of Tables

# 1 Preface

In all probability, the reader of this thesis possesses at least one personal computer (PC). Maybe this PC is additionally joined by a cell phone, a personal digital assistant (PDA), or a mp3 player which can connect itself to the internet. By now, many people drive cars having internet access and being able to connect to the driver's smartphone. Additionally, advanced, wearable devices like wrist watches resembling small computers are capturing markets. This list could be considerably expanded, especially by various achievements of the entertainment electronics industry. However, its essence has probably become clear: in the meantime, we take it for granted to be surrounded by a heap of cross-linked high-tech equipment. When looking back in time, this situation has not always been as it is today, though. A couple of years ago, we did not enclose ourself in a wall of high-tech gadgets. Neither was the internet as widespread as it is today. At that time, the interconnection of PCs was mostly limited to local area networks making them de facto isolated devices by today's standards. In the early days of computing there were even no PCs and computing was limited to a few mainframe computers shared by many users.

The essence of this short historical retrospect can be stated as follows: computer industry is in the middle of an ongoing process affecting our relationship to technology. This process is characterized by two major trends. On the one hand decreasing bandwidth limits and cost have increased the interconnection of computing devices by, at the same time, dramatically changing our life. On the other hand falling prices for transistors and exponential growing processing speed have formed the basis for the proliferation of computer devices. There is no end in sight: according to Moore's Law (see [Int07]) these trends will endure for quite some more time driving this process forward. Moreover, fostered by this development computer devices act and intercommunicate to an increasing extend without user interaction. It is foreseeable that computer devices will most likely become even more embedded into our daily life, supporting our everyday business, and relieving us of more cumbersome tasks.

Aside from these benefits we already enjoy today, this process also rises some major challenges which have largely remained hidden from the common user by now. The most notably insight regarding this issue is that *the permanently increasing number of interconnected devices bears the problem of controllability.* While being not acute in our private life yet, other fields are already suffering heavily from this problem. At present, even hordes of professional system administrators struggle in managing "their" highly organized IT infrastructure (see [IBM07a]). Automotive manufacturers have major difficulties to manage the coaction of numerous electronic components within a vehicle faultlessly (see [Sta05]). Many other topical examples for failures of complex systems could be given that sustain barely controllable complexity. Moreover, even if not being a problem for our everyday life yet, this process puts a big question mark over our future relationship to computing devices: how can we expect an average user to manage countless connected and automated devices, if even professionals fail in their field?

All given examples seem to indicate the incapability of present organization methods to cope with increasing complexity of technical systems. Possible relief could be afforded by endowing technical devices with life-like abilities like self-organization. Particularly the fact that nature seems to manage extremely complex systems like the human body without any problems motivates several research programmes. This thesis is embedded into one of these programmes: it is part of the Organic Computing programme, which will be introduced in the next chapter. *The goal of this thesis is to gain information of how self-organizing systems can be controlled.* Especially experience in mastering unpredictable phenomenons typically emerging in these systems should be attained within this thesis.

In order to fulfil our intended goals, a lift control scenario is used as a testbed for our investigations. We use this scenario as a representation of a complex technical system that shows an unpredictable phenomenon. Our main motivation is to derive general statements from the perceptions gained at controlling the lift system (the actual job definition is given in the next section).

Perhaps one day research performed in this field will allow us for using computer technology in a different way as it is usual today. In the future we will possibly utilize a large number of computers without further agonizing over how to operate them. At this time computing will not only be a necessary part of our daily lives as is is today; computer devices will be operated with ease and its integration will be as seamlessly that we will not notice

them any more. Maybe there will even be a time where self-organizing lifts are put into operation.

## 1.1 Job Definition

Basis of this thesis is the existing lift simulation developed by Christoph Pickardt [Pic06]. Within this lift simulation an unpredictable phenomenon called bunching emerges. *The goal of this thesis is to enhance this lift simulation programme in a way that bunching can be quantified and controlled.* The enhancement should be done in terms of the general concepts provided by *Organic Computing* (OC). In particular a so-called *observer/controller architecture* should be implemented above the existing simulation programme.

The functionality of the implemented observer/controller architecture should be verified by experiments. Especially, the effectiveness of the controller on the bunching effect has to be examined. Additionally, the existence of possible influencing factors on the controller effect has to be investigated.

## 1.2 Approach and Structure

This first chapter gives a general introduction to the subject of this thesis. This thesis is based on several theoretical concepts and is embedded into a relatively young research programme. These underlying terms and concepts, along with a couple of related fields of research, are introduced and illustrated in chapter 2. This chapter also gives a short introduction to lift group control together with general terms used in this field. Additionally, at the end of chapter 2, an introduction to bunching is given with the development of a measure to quantify bunching.

Chapter 3 covers the actual implementation of the lift control simulation. At the beginning, it is dealt with the underlying simulation framework and the original lift simulation, followed by a determination of the programme requirements. All relevant characteristics of the lift simulation model are described hereafter. The main part of this chapter addresses the lift simulation itself. Especially the development of a bunching measure as well as means to control the system are presented in this place.

Subsequently, the used experimental design is introduced in chapter 4. In chapter 5 the actual experiments are performed. The focus of these experiments lies on the determination of the effect of the observer and the controller on the system. Moreover, the developed bunching measure is tested for its functioning within this chapter.

Hereafter, all findings are summarized and reviewed in chapter 6. In addition, a short outlook on possible further work and improvements is given. At the end, a list of used abbreviations is given in appendix A.

# 2 State of the Art

This chapter provides a concise overview of the general concepts and ideas on which this thesis is based.

First of all, the motivation and the goals of *Organic Computing*, the field of research in which this thesis is nested in, are characterized in section 2.1. Two fundamental components of Organic Computing, *emergence* and *self-organization*, are described in section 2.2. One method of controlling the emergent phenomenons appearing in self-organizing systems is the *generic observer/controller architecture*. This architecture is implemented within this thesis and described in section 2.3. Techniques connected to Organic Computing in such a manner that they share the same ideas or try to master the same challenges are introduced in section 2.4. A short introduction in the history of lift traffic control systems is given in section 2.5. *Bunching* is a phenomenon appearing in lift traffic control scenarios and a form of emergence. A definition of bunching, measurement methods and means to cope with this phenomenon are given in section 2.6.

## 2.1 Organic Computing

*Organic Computing* (OC) is a vision of future information processing systems. The term "Organic Computing" was formed in 2002 as a result of a workshop aiming at future technologies in the field of information processing. The outcomes of the workshop are outlined in [AM04]. The priority programme "Organic Computing" of the German Research Foundation (DFG) tries to find answers arising within this vision. Embedded in this priority programme is the present diploma thesis. This programme was initiated in 2004 and is scheduled to last for a maximum of six years. The superior goal is the realization of new computer and system architectures, which follow the OC vision and implement the approaches stated above. Besides the priority programme there are other programmes similarly motivated and concerning themselves with the same topics. Worth mentioning

is the OC initiative propagated by von der Malsburg, which is focusing amongst others on building new computing systems based on utilizing biological principles (see [OCo07]). Additionally, also the computer industry is taking action: IBM for example initiated the Autonomic Computing programme introduced in section 2.4.1.

### 2.1.1 Motivation of OC

The motivation of OC is based on the insight that in foreseeable time current principles of organization of technical systems will not be able to handle the accompanied complexity any more. There are two main reasons for the rise of complexity:

On the one hand, there is a general trend towards smaller, more intelligent and more numerous devices surrounding everybody in his everyday life. This outlook is amongst others given by the paradigm of ubiquitous computing: future information processing will be integrated into a broad range of everyday objects ("everywhere"). The term "ubiquitous computing" was first introduced by Mark Weiser in [Wei91].[1] All these devices will be interconnected and communicate over various communication channels. Thus, networks of intelligent systems will arise, whose behaviour will no longer be predictable due to the effects of interaction (see [Sch05a]).

On the other hand, the increase in the number of small devices will lead to changing system compositions, since these devices will not be permanently logged into the same system. Therefore, these systems will perform their tasks in dynamically changing environments. This requires the ability to work properly within unforeseen situations (see [Sch05b]). This ongoing development leads to a dramatic increase in system complexity and will render it impossible to make a statement on the current system state because the system state itself becomes too complex to be described. At this point the global behaviour of these systems might become unexpected and therefore current organizational methods for technical systems are going to surrender.

### 2.1.2 Vision of OC

Driven by this outlook, the vision of OC presents a way to cope with these challenges (see [Sch05b]): we will soon be surrounded by autonomous systems, which organize and adjust

[1] For a comprehensive introduction to the ubiquitous computing paradigm visit the overview given by Mark Weiser at [Wei07] or refer to [Gre06].

themselves to permanently changing environmental conditions.[2] These future systems will possess certain degrees of freedom to handle unforeseen situations and act flexibly and independently. In consequence, these systems will exhibit self-organized behaviour which makes them able to adapt to a changing surrounding: they show life-like characteristics. That is why in the field of OC these systems are called *"organic"*. Hence, an *"organic computing system"* is a system, which adapts dynamically to the current conditions of its environment (see [DFG]), but still obeys goals set by humans. In addition to this environmental awareness, systems providing services for humans will adjust themselves to the users' requirements (not the other way round). For that reason context sensitive user interfaces will receive great attention (see [AM04]).

### 2.1.3 Approach of OC

Hence, the goal of OC is to build systems in a way that they are still manageable under uncertain conditions and do not show undesirable behaviour. The key to fulfil this goal is hereby to grant the system some degrees of freedom to adapt to a changing environment. In the first instance, this freedom evokes unpredictability which will raise the complexity of the system to an even higher level. As a consequence, the system must be *self-organizing* to be able to cope with this complexity on its own and thus relieving the users from maintenance issues. The idea of self-organization is the basis for the organic behaviour of the system and will be covered in section 2.2. OC systems will show the so-called self-x properties as utilized by *Autonomic Computing*[3]: they will be self-organizing, self-configuring, self-healing, self-protecting and self-explaining (see [KC03]). These traits make an OC system much more robust than a classical system of corresponding scope could ever be.

Implementing the ability to learn into an OC system is another very promising outlook. This would enable the system to adapt to changing conditions not only by maintaining a fault free operation by performing a simple reconfiguration of the remaining components in case of failure but rather to fully adjust itself to an altering environment by self-optimization.

Furthermore, OC systems will be aware of themselves and their environment, they will be self-aware. Because of all these properties an OC system does not need to be intensively

[2] Many possible examples are described in [AM04, section two]

[3] The idea of Autonomic Computing and the self-x properties are covered in section 2.4.1.

controlled from an external place, instead it is able to manage itself and only requires objective definitions given by the user. For this reason the *generic observer/controller architecture* as introduced in section 2.3 has been developed. The basic intent of OC is not only to fight the unpredictable behaviour, but also to try to channel these phenomenons into a desired development and make use of them. These phenomenons are called *emergence* and will be discussed in section 2.2 in more detail. To fully profit from the alluring opportunities of intelligent networks, the insight has been evolved that emergence (besides self-organization) must be a key issue in OC systems.

### 2.1.4 Challenges of OC

However, besides this fascinating outlook, the "materialization" of the OC vision depends on several crucial factors (see [Sch05b]):

- We have to guarantee, that a self-organizing system does not show undesired behaviour. This is particularly important when malfunction can have disastrous consequences as for example in safety critical applications. The generic observer/controller architecture as described in section 2.3 seems to be a promising approach in asserting certain functionality and additionally in keeping the system at an efficient state of operation. OC systems will only get accepted if we can trust them. Therefore trust could turn out to be the most important prerequisite for acceptance.
- Closely related is the inevitability of providing means for the user to monitor and influence the system: it must be guaranteed that it is still the user who guides the overall system. Therefore the designer/maintainer must be able to control the system which means that there has to be a possibility to take corrective actions from outside the system. The generic observer/controller architecture as described in section 2.3 attends to this task.
- We have to determine appropriate rules and patterns for local behaviour in large networks of smart devices in order to provide some requested global functionality. An important topic in the design of a self-organizing system is hereby to utilize emergent phenomena arising at local level in such a way that the system shows the desired behaviour at global level. Therefore, the task is to derive a set of behavioural and interaction rules that, if embedded in individual autonomic elements, will induce a certain global characteristic (see [KC03]). The inverse direction of anticipating the

global system behaviour based on known local decision rules is also very important in this regard.

- An OC system interacting with humans has to show context sensitive characteristics and has to filter information and services according to the current user's needs. This refers to methods for artificial vision and is addressed for instance by von der Malsburg [vdM04].
- By designing an OC system one must carefully think about how to open up the necessary degrees of freedom for the intended adaptive behaviour. Certain degrees of freedom are needed to enable self-organization, but it is easily imaginable that allowing the subsystems a too broad range of possible (re-)actions in a specific situation could lead into uncontrollable chaos.
- The implementation of a learning ability in an OC system provides not only great chances but also bears serious problems. Learning systems can make mistakes. In fact they are going to make mistakes if no countermeasures are taken. Additionally we have to guide the attempt of the learning system. We must assure that the system does not develop itself in an undesired manner.

This list could be considerably expanded, but it already represents the most important topics. Even though OC is not an established technology yet, promising approaches have already been developed in this field. The generic observer/controller architecture is a good example for such a concept.

## 2.2 Emergence in Self-Organizing Systems

As stated in section 2.1, OC systems are *self-organized* to cope with the challenge of proliferating complexity. Self-organizing systems typically exhibit *emergent behaviour*. A key issue of OC is therefore the technical utilization of *emergence* in OC systems (see [DFG]). Emergence and self-organization will be introduced in the next sections.

### 2.2.1 What is Self-Organization?

*Self-Organization* is the basis of OC systems and many other technical systems. It denotes a way to organize systems consisting of several entities/subsystems. Intuitively, self-organization refers to exactly what is suggested: systems that appear to organize

themselves without external direction, manipulation, or control (see [Dem98]). In the field of natural science a common definition of self-organization is stated as follows:

> *"Self-organization is the spontaneous emergence of (new) spatial and temporal patterns within dynamic systems, which is based on interaction of subsystems." [...] "Within a self-organizational process order and complexity arises from the system itself"* (translated from [Bib07b])

In other words: self-organization is the process of generating order from disorder based on local interactions without explicit pressure or involvement from outside the system.

A *system* is defined as a group of interacting parts functioning as a whole and distinguishable from its surroundings by recognizable boundaries (see [CAL07]). A *self-organizing system* is a system that is based upon a self-organizing process. That means the system structure is not "hard-coded" but arises from the interaction of the system constituents.

The term *complexity* used in the above definition refers to the difficulty to predict the dynamics of a self-organizing process. This is based particularly on the problem how to map local decision rules of the subsystems to global behaviour. Self-organizing systems belong to the vague field of *(dynamic) complex systems* (see [Bib07a]).

Beyond self-organization as defined above, OC utilizes a concept termed *controlled self-organization*: to guarantee that the system evolves in compliance with the user's goals, a goal definition in the form of an objective function is stored in the system. However, as self-organizing systems possess no central authority, the objective function has to be distributed among the intelligent agents. Furthermore the user is able to change the goal definition of the system. The means for doing so are described in section 2.3.

There are several approaches to measure the level of a system's self-organization. A possible way of quantifying self-organization is to observe the extend of *complexity reduction* of a self-organizing system. In this context, "complexity" is utilized in terms of *information entropy*[4] at which a system is more complex if more information is needed to describe its state (see [Bib07a]). This approach is introduced along with an overview of other practices in [CMMS+07]. It is focused on OC systems featuring the generic observer/controller architecture described in section 2.3. Systems of this kind can be controlled from outside by setting parameters. The set of all possible configurations is called configuration space.

[4] The concept of information entropy was introduced by Shannon in [Sha48] (Probably easier to find is a reprint of the original article [Sha01]).

There is an external and an internal configuration space: the external is visible to the external controller (for example a human user) and comprises all possibilities to control the system from outside; the internal represents the set of all possible configurations of the system. The cardinality of a configuration space corresponds to its complexity (or information entropy, respectively), as more information is needed to describe a larger configuration space. If the system is self-organizing, it will manage itself to a certain extend and thus the external complexity will be smaller than the internal. The difference of the two cardinalities is called the complexity reduction and therewith the degree of self-organization of the system.

Mentionable in this regard are the works of Werner Ebeling [Ebe91, EFS98] which provide a easily understandable outline on the general thoughts in this regard. A deeper insight in this subject is given by Haken [Hak00] especially by putting emphasize into concepts dealing with self-organizing systems. Additionally, Haken [Hak04] provides an extensive introduction to the related field of synergetics. Although focusing mainly on the emergence of collective phenomena, Schweitzer [Sch97] gives an extensive view on self-organization of complex systems and applications of self-organization theory.

### 2.2.2 What is Emergence?

Emergence as a general concept is probably explained best by an example: imagine an electromagnetical resonance circuit consisting of a capacitor, a resistor and an inductor. Its components exhibit defined attributes such as capacity (capacitor), resistance (resistor) and inductance (inductor). In contrary, the overall circuit shows properties as resonance frequency and damping factor. These system properties cannot be derived from any of the components of the circuit instead they develop from the interaction of the system components. The development of these system properties is called an *emergent phenomenon*. This phenomenon is referred *emergence*. The bottom line of this example could be the following familiar statement:

*"The whole is more than the sum of its parts." (from "Metaphysics", Aristotle)*

Expressed in a more scientific way emergence is a property of a system that cannot be derived from the simple summation of the properties of the constituent subsystems (see [MS04]). According to this, an example of a non-emergent "phenomenon" would be: the weight of a car is the sum of the weights of its components. The total weight is no

emergent phenomenon because it obviously is the simple summation of the properties of the constituents.

Additionally, emergence can be observed in many complex systems in the technical field (like the internet), in physics (like colour of objects) as well as in nature (like the path finding behaviour of ants or the shape of weather phenomena). Moreover, emergence is discussed in the domain of philosophy of mind in terms of emergentism. These pretty different origins of emergent phenomena lead to different and controversial definitions of emergence. For a discussion of several views on emergence, it is worthwhile to have a look at [MSS06], [Cor02], [Fro04] and **(Achim Stephan - insertref)**.

Common to all these flavours of emergence basically are four characteristic traits: emergent phenomena are based on (1) the interaction of mostly large numbers of individuals, (2) some kind of order arises from disorder within the system (at macro level), (3) this order emerges without external intervention, central control nor is it explicitly programmed into the single individuals (at micro level), and (4) this macro-level order is somehow different from the micro-level processes within the system (see [MS04, MSS06, Gol99]). The connotations of these traits are very definition-specific though. Particularly (4) is very vague and the meaning of this "difference" varies widely within the views on emergence.

Even the distinction between "emergence" and "self-organization" is not always clear in the literature. De Wolf and Holvoet define the difference between these terms as follows:

> *"Emergence emphasises the presence of a novel coherent macro-level emergent (property, behaviour, structure, ...) as a result from the interactions between micro-level parts. Self-organization emphasises the dynamical and adaptive increase in order or structure without external control."* (see [DWH05])

In other words: emergence is the result of the order generating process in contrast to the process itself. As in OC systems the order generating process is self-organization it can be said that within an OC system emergence is the result of the self-organization process. This disambiguation will be used in this thesis.

The general acceptation of emergence in the context of OC systems is pretty good described by the general aspects (1) to (4), stated above. However, there exists no universal accepted definition of emergence in the OC context beyond these general characteristics. As for the self-organizing nature of OC systems, emergent phenomena are most likely the outcome

of a self-organizing process. Therefore, in general and within this thesis it is assumed that emergence in OC systems is caused by self-organization.

There is another mentionable contribution to emergence in the domain of OC: in order to allow quantification of emergence in technical systems, Mnif and Müller-Schloer introduce in [MMS06] a further definition of emergence termed *quantitative emergence*. They define emergence as self-organized order that can be observed in terms of spatial and/or temporal patterns. It is based on Shannon's theory of information entropy. The measurement rests upon the comparison of a system's entropy[5] value at the beginning and at a later moment. Mnif and Müller-Schloer propose that only quantifiable phenomena resulting in a (self-organized) increase of order deserve to be accepted as emergence. A limitation of this definition is, however, the mandatory need for an observable pattern. The aforementioned resonance frequency does not constitute such a pattern, it is rather a property of such a pattern. In a situation, where the observed phenomenon is totally different from the behaviour of the system constituents, a type of preprocessing could be necessary within the observation process: the resonance frequency can be obtained by observing the system behaviour after a Fourier analysis.[6] Quantitative emergence in comparison to other historical and novel definitions is outlined in [MSS06].

Additional to the books and papers presented so far, there are lots of other works worth reading: Fromm [Fro05] specifies a universal taxonomy and comprehensive classification of the major types and forms of emergence in multi-agent systems. In [RMS05] the difference of the bottom-up phenomenon emergence and the classical top-down design process is discussed. At [Fac07] a brief overview is given on philosophical and technical concepts and definitions of emergence.

### 2.2.3 What Impact do Emergence and Self-Organization have on OC Systems?

As stated in section 2.1, self-organizing systems have several advantages over classical, centrally controlled systems. The failure of a single component does not cause a global

[5] As stated in 2.2.1, the entropy of a system is defined as the amount of information needed to describe the state of a system. Entropy decreases with increasing order as a structured system is easier to describe.

[6] Fourier analysis is used to decompose a periodic function into its basis functions: the function hereafter is expressed as an infinite sum of sines and cosines. See [Bib07c] for a brief definition or [But05] for detailed introduction.

malfunction, as the system is able to adapt to changing circumstances. These fail-safe characteristics are shown for example at the process of routing in the internet: The failure of a single router is dynamically compensated by diverting the traffic affected by the failure over the remaining routers. At doing this, the proper operation of the system is restored. For more information on internet routing refer to [Hui00].

Besides this self-healing ability, the advantages of OC systems basically are the other self-x properties mentioned in section 2.1. As a result, self-organizing systems are a means of reducing the "complexity" of computer systems. In fact, as stated in section 2.2.1, self-organizing systems are complex systems. However, the user does not have to manage this complexity as the system manages itself, that means the "visible complexity" is lesser. Referring to the above stated example the interaction of the nodes (routers) to maintain a fault-free operation is certainly a complex process, however, this process is performed by the system itself and does not have to be initiated by a human operator.

Regardless, self-organization and emergent phenomena rise new problems unknown in classical technical systems:

- On the one hand emergence is characterized as a bottom-up phenomenon in contrary to the usual top-down design process. Therefore local decision rules, which cause emergent phenomena, have to be deducted from a desired global behaviour. To be able to design such systems it is also necessary to understand the inverse direction: How does local behaviour map to global behaviour? As global emergent behaviour is a nonlinear combination of local behaviour this is a highly nontrivial task (see [KC03]). These questions have not been answered by research yet.
- On the other hand the fact that in a self-organizing system global behaviour cannot be deducted from local properties raises the problem of controllability: How can we guarantee that the system acts in a certain, desired manner? How can the user revise the system goals? Within the usual top-down design process constraints are propagated from the top-level to the subordinate instances. However, this is contradictory to the bottom-up process of self-organization (see [MSvdMW04]). *Controlled self-organization* is a method addressing this problem. It is introduced in section 2.3.

## 2.3 The Generic Observer/Controller Architecture

Computer systems should not simply self-organize, but they should use self-organization to achieve a certain goal. Therefore systems featuring *controlled self-organization* are aspired in OC. Moreover, OC systems will always need some external high level control by a human operator. On the highest level, humans have to decide which goals the system will strive for. Furthermore the system has to be capable to deal with (unanticipated) emergent behaviour and to adapt to changing situations. In other words, some method to fulfil the self-x properties and to guarantee that the system aims at the desired goals has to be implemented in the system. The *generic observer/controller architecture* (o/c architecture) promises to satisfy all these demands. More information in this regard can be found at [RMB+06] and [BMMS+06]. The following paragraphs are based on this sources.

A classical control theory approach to keep properties of a system in preferred regions is to implement a *closed control loop* (see [FPEN02]): Observe certain attributes of the system and act according to an evaluation of the observation; if a parameter exceeds the desired range, take action to direct the system back into its desired range; observe the effect of the intervention and take further action, if necessary. An example for this is a car's cruise control, which regulates the speed at a constant level: if the observed speed falls below a certain threshold, some action to raise the speed has to be taken; if it raises to high, do something to slow the car down. The actual action depends on the evaluation of the observation: In case of falling speed, there are two obvious actions: shift one gear down or push the throttle. The action is chosen which appears to be the most reasonable. By this feedback mechanism, disturbances to the system can be compensated.

The generic o/c architecture is basically an implementation of the closed control loop approach. The regulated system is termed *system under observation and control* (SuOC). A control loop like the one described before is created on top of the SuOC: An *observer* monitors the state of the underlying system by measuring, quantifying, and predicting the parameters and conditions of the system components (see figure 2.1). Afterwards the observer reports the gathered information to the controller. Then, the *controller* evaluates the information regarding to a given objective function and takes appropriate actions to influence the SuOC with respect of the system goal. The SuOC together with an o/c control loop will be referred to as *OC system* in the following.

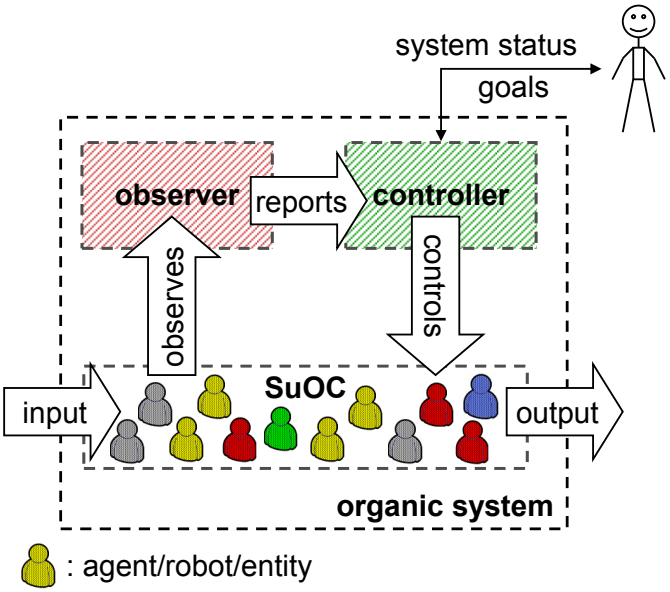



Figure 2.1: The generic observer/controller architecture (from [CMMS+07])

Additionally, the controller reports the current system status to the user. Thus, if the system evolves in an undesired direction, the user is able to influence the system by changing the system goal from outside. This assures that the user has the ultimate control over the system.

### 2.3.1 Observer

The purpose of the *observer* is to generate a precise summary of the global state (called the *situation parameters*) and the dynamics of the SuOC. For this reason, the observer does not only report on the current system parameters but also predicts the future status of the system. In the following a concise overview of the functionalities of the observer will be given. The overview is based on the concept of the observer as described in [RMB+06] and [BMMS+06].

The observation process contains the steps *monitoring*, *pre-processing*, *data analysis*, *prediction* and, *aggregation* and is guided by an observation model. Its architecture is outlined in figure 2.2 and described in the following:

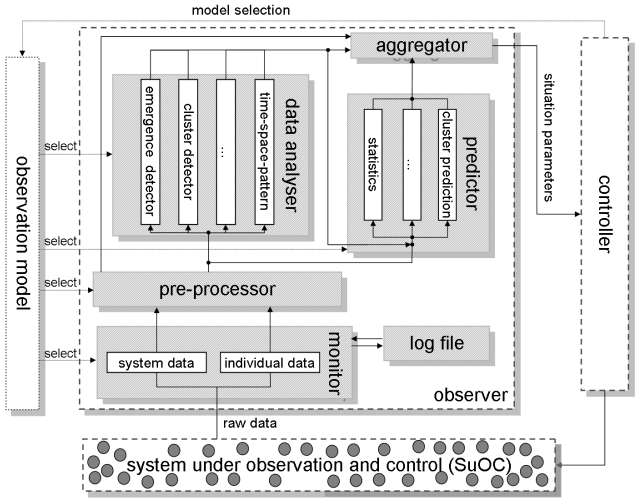



Figure 2.2: Generic observer architecture (from [RMB+06])

- The observer *monitors* attributes of single system elements as well as system data at a given sampling frequency. As a result raw data from the system is acquired. The obtainable data is constituted by the sensory equipment of the observer.
- All measured data is stored in a *log file* for every loop of observing/controlling the SuOC.
- Subsequently, a *pre-processor* derives attribute data from the raw data obtained by the monitor (for example an element's velocity can be calculated from the changes of the $x$ and $y$ coordinates over time).
- The *data analyser* applies a set of detectors to the attributes given by the pre-processor. These detectors look for certain patterns indicating for example phenomena as clustering or emergence among the system elements. The result is a global description of the system state.
- The *predictor* uses information from the data analyser and data from the pre-processor to estimate the future state and dynamics of the system. For this reason, the predictor has access to methods from the data analyser besides performing calculations by itself. It also features a memory to be able to analyse the system history and thereof derives an estimation of the future.
- The *aggregator* combines information from the data analyser and the predictor as well as raw data coming from the pre-processor to the so called *situation parameters*. They are filtered by the aggregator to eliminate the effects of noise. The situation parameters constitute a system-wide description of the current state and the dynam-

ics of the SuOC. The aggregator hands over the filtered situation parameters to the controller.

An observation model guides the observer by selecting for example sampling frequency, observable attributes, data analyser patterns, and predictor methods. By altering the observation model the controller can influence the observer. Thereby, it is possible to adapt the observer to different scenarios.

### 2.3.2 Controller

The role of the *controller* is to guide and control the self-organizational process of the SuOC based on the informations received from the observer (the described concept of the controller is based on [RMB+06] and [BMMS+06]). However, the controller will only interfere when necessary, as for example when an attribute is detected to be outside of its destined bounds. Assuming a system containing self-organizing agents the controller can influence the SuOC basically in three different ways:

1. Change the local decision rules of the agents. This will lead to a different local behaviour, which may result in a different global behaviour.
2. Manipulate the perception of the agent's neighbourhood. This can be done by changing the neighbourhood structure of the system elements as for instance hiding some attributes/elements of the agent's surrounding. Another possibility would be to feign attributes of the neighbourhood. As a result, the global behaviour of the system will change.
3. Alter the environment of the agents directly. This allows indirect control of the SuOC. As a matter of course this type of control works only if the agents have sensors to realize the change of the environment and the controller has means to exert influence on the SuOC. However, in this theoretical model the environment cannot be altered at all. Nevertheless, some agents could be declared as parts of the controller. Then they could be used as actuators.

There are certainly more types of control imaginable, but these three represent the most general ones.

As stated above, the controller tries to direct the self-organizing process of the SuOC. At doing this, it periodically decides what action has to be taken regarding to the situation parameters received from the observer. A generic architecture of the control process is

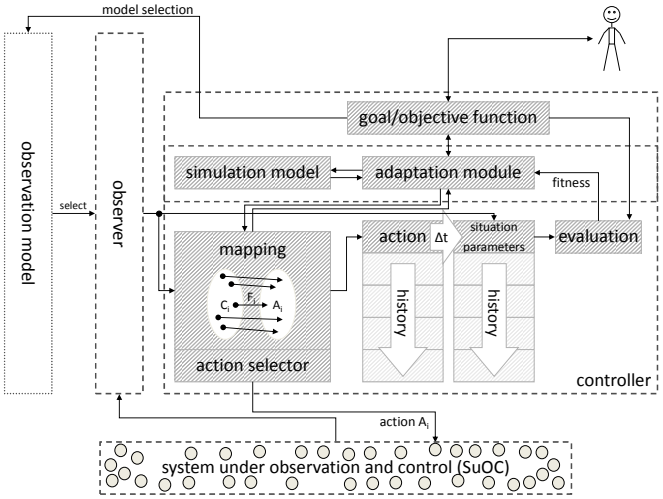



Figure 2.3: Generic controller architecture (from [RMB+06])

outlined in figure 2.3. However, the generic architecture presented here has to be adapted to the actual scenario and is thus only a general guideline. It is structured into the following subprocesses:

- The *action selector* is the heart of the controller. It chooses the best known action out of the set of given actions, whereas the best action is identified by a mapping module. This module contains a set of situation parameters $C_i$, a set of possible actions $A_i$ and a mapping function assigning each rule $(C_i, A_i)$ a fitness value $F_i$. Subsequently the action selector forwards the selected action to the SuOC.
- Concurrently to the action selection process the controller keeps track of *history* data to review the success of the chosen actions. Therefore for every action at time $t$ the consequent situation parameters at time $t + \Delta t$ (as reported by the observer) are stored into a memory. Then the tuples of actions and resulting situation parameters are used to *evaluate* new fitness values $F_i$. At last the *adaptation module* updates the old fitness values in the mapping table with the newly evaluated ones.
- Additionally to an observation based update of the fitness values $F_i$ for a known $(C_i, A_i)$, the controller can also generate completely new rules and actions by applying mechanisms of machine learning. If the observer reports for example so far unknown situation parameters, no matching action can be found by the action selector in the first instance. However, by using a *simulation model* the adaptation module could either evaluate a suitable action for the current situation from the

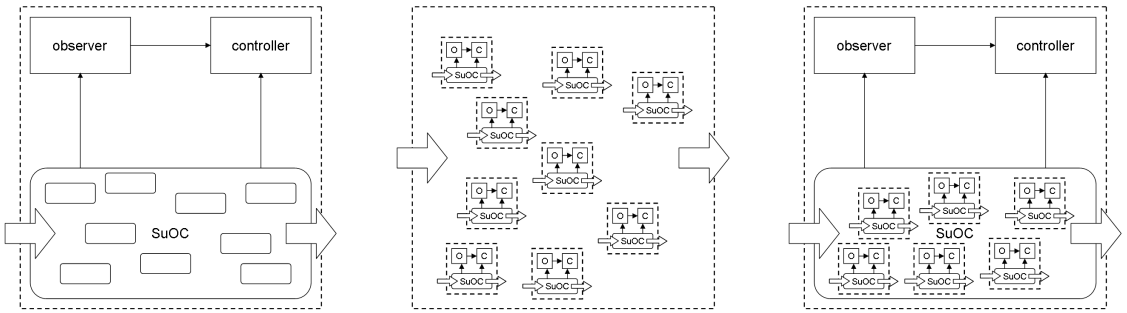
set of known actions (i. e. create a new control rule) or generate a completely new action. The details of the used machine learning mechanisms are not defined in the generic architecture: adaptation can occur online and/or by a model based internal learning process; possible methods could be artificial neural networks, learning classifier systems, reinforcement learning, or evolutionary algorithms. In any case, if new rules and actions are created, a simulation model should be used to predict the possible outcome of the modification before actually applying them to the SuOC. For this purpose, the simulation model must have access to a system model. If time permits, this would allow an internal response optimization.

Note, however, that the presented architecture is of generic nature. Practical implementations do not have to contain all functionalities described above. A controller consisting only of an action selector and a mapping table is possible but very limited in its possibilities. This type of controller would not be able to adapt to changing environmental situations on its own, though. Moreover, it would not be able to adjust the mapping table to different system goals. Actually, it could not react to a system goal change at all.

As seen in figure 2.3, the controller offers three interfaces to exchange information with other components: (1) The aggregated data obtained from the observer. (2) The system objectives imposed by the user. (3) An interface to interact with the SuOC. Information about the SuOC depends on the scenario, is predetermined, and therefore must be given to the controller.

Apart from the three (partially) concurrent subprocesses described above, the control process as a whole passes through the following steps:

- At first the controller decides if an intervention is necessary. This estimation is based on reference values for the indicators and on possible degradation of the objective function.
- Then the controller chooses the best known action $A_i$ for the reported situation parameters $C_i$.
- The controller keeps track of recent actions and waits some time $\Delta t$ after the last executed action before interfering again. This prevents overshooting control.
- If the controller knows no suitable control rule for observed situation parameters $C_i$, it has to create a new one. If it has access to a system model it can create new actions and optimize existing actions utilizing a simulation model.




Central Decentral Multi-Level

Figure 2.4: Observer/controller realization (from [BMMS+06])

### 2.3.3 General Characteristics of an Observer/Controller Architecture

The observing and controlling process is executed in a continuous loop. This ensures that the system acts in the desired way. Additionally, the system as a whole will adapt to changes within the SuOC as well as changes in the surrounding of the system. In [CMMS+07] three different architectural approaches are stated (see figure 2.4):

- Central: One observer/controller for the whole system.
- Decentral: One observer/controller for each subsystem.
- Multi-level: One observer/controller for each subsystem as well as one (or more) for the whole system.

In particular, at larger and more complex systems it will be necessary to build hierarchically structured OC systems[7] instead of trying to manage the whole system with one o/c pair. In the case of multiple o/c levels a low-level SuOC will consist of "simple" elements like single software or hardware modules. However, at higher levels a SuOC will comprise subsystems at which each of these subsystems represents an OC system[8] on its own. The further one steps up in the hierarchy the more abstract and general are becoming the goals of the controller. This corresponds to the management of a company where high-level administration units are not bothered by low-level decisions.

In the business world it is necessary to separate low-level from high-level management tasks in the way mentioned above to handle the complexity of management tasks. This paradigm can be transferred to the field of OC systems: Regarding to [CMMS+07] the need for multi-

[7] As stated before, an OC system contains a SuOC, an observer and a controller.
[8] Including an own observer/controller.

tier o/c architectures is based on complexity in terms of variability. *Variability* is the number of possible, by an observer detectable configurations of a SuOC, and is apparently larger if an SuOC contains more elements. Implementing multi-tier o/c architectures is a means of greatly reducing variability and therewith complexity. For further informations regarding this topic see [CMMS+07].

The main objective of the o/c architecture is to achieve a controlled self-organized behaviour. This means that it allows for self-organization but at the same time provides means to control the emerging global behaviour of self-organizing systems. In this regard it is important to note that an organic system continues to work and does not break down if observer and controller stop working (see [RMB+06, page 3]). Furthermore in organic scenarios the underlying system does not need to have a strict structure. The SuOC can contain very different elements of very different granularity[9]. Unlike classical system design the o/c approach enables OC systems not only to cope with predetermined situations but also to adapt to emergent phenomena for which they have not been programmed explicitly.

## 2.4 Approaches Similar to OC

Besides OC, there are programmes motivated in a similar way and dealing with alike challenges. Two of them will be discussed in the following: The Autonomic Computing approach propagated by the IBM corporation is introduced in the next section and the Collaborative Research Centre 614 funded by the German Research Foundation (DFG) in section 2.4.2.

### 2.4.1 Autonomic Computing

*Autonomic Computing* (AC) is an approach to self-managing computer systems without human interference (see [IBM07a]). The term *Autonomic Computing* is a metaphor based on biology. It refers to the autonomic nervous system of the human body. This connection will be made clear in the following sections.

The programme was initiated by the IBM Corporation in 2001 [IBM01] and is by no means a closed project: Other companies or research institutions are welcome to join the

[9] In terms of multi-tier o/c architectures as described above.

initiative. Good resources regarding AC are the official AC homepage [IBM07b] and the AC research homepage [IBM07a]. A very good and broad overview is provided in [KC03] and [SPTU05]. AC is in some areas closely related to OC or based on similar principles, as presented in the following.

### Motivation of AC

AC is motivated by the insight that complexity of computing systems is increasing more and more and the IT industry is therefore heading towards a "complexity crisis". Increasing complexity refers hereby not only to individual software components consisting of billions lines of code, which demands a horde of software maintainers and administrators to keep them running. Complexity will in an even greater manner arise due to the integration of several different computing systems scattered around the globe and interconnected via the internet. Today the IT industry deals with the increasing complexity by also increasing the number of IT workers. This "solution", however, is by no means forward-looking and will only lead to a further increase in IT costs. Additionally, the rising administration effort erodes more and more the dependability on computer systems. However, especially in server infrastructures reliable systems are essential, hence the need for a change seems to become even more inevitable (Check up [IBM07a] for further details).

Although not directly addressed, the AC initiative is aware of the ongoing trend of ubiquitous computing escalating the quantity of linked components. Though ubiquitous computing does not appear on IBM's current agenda, it probably will in the future (see [KC03, Bey07]).

### Vision of AC

In order to overcome the quagmire of proliferating IT management costs and to fulfil the desire for reliable software systems IBM defined the main goals of AC as follows (see [IBM07a]):

- Reduce the management effort humans have to perform and increase productivity. This includes hiding complexity from the users by building intuitively maintainable IT systems.
- IT systems should be less reliant on human interventions to increase overall stability.

- Build systems that are able to adjust to varying circumstances and optimize themselves.

To achieve these aims AC postulates a paradigm shift within computing. According to the AC vision future computing systems will not be strictly controlled by a (or a few) central instance(s) and will not need to be heavily maintained by legions of IT personnel any more. In contrast, AC aims at creating self-managed computing systems with a minimum need of human interference. The concept of self-management is hereby adopted from the autonomic nervous system of the human body which covers involuntary key functions like respiration and heart rate without human interference. This is performed by motor neurons sending messages to organs at a sub-conscious level. That means key functions are covered without conscious interference. The aim of AC is to realize something similar in computing: create a self-managing network of smart components that relieves the users from low-level management activities. The IBM autonomic computing research website [IBM07a] is a good resource for a detailed introduction in this field.

### Architecture of an AC system

An self-managing AC system features several self-x properties, most notably self-configuring, self-healing, self-optimizing, and self-protecting (see [Ste05]). These properties are basically the same as the self-x properties of OC systems and are for that reason not described in detail here. IBM has introduced a generic framework for an AC infrastructure in [IBM04]. The main concepts of this blueprint will be pointed out in the following:

To meet the properties of self-management, AC systems contain elements called an *autonomic manager* (AM). An AM manages so-called *managed resources* (MR). MRs can be either a *managed component* (MC), a group of MCs, or a group of other AMs. The MCs can be any type of resource (hardware and software) as for example a printer, a server, or a database. The MCs form the lowest level of the architecture and are basically the same as components in a non-autonomous system. As stated above, an AM can also manage other AMs. This means, in terms of "having authority over another AM" several AMs can be hierarchically ordered. At lower levels, the range of internal behaviours and relationships with other elements is probably very limited and hard-coded. There are used most likely well-established techniques that rarely fail. The higher an AM resides in the hierarchy the more hard-coded behaviours will give way to high-level objectives such as "optimize network usage". Both low-level behaviours and high-level goals of an AM are

either defined by the designer, by elements having authority over the element or by contracts with peer elements. IBM terms this set of goals and considerations an AM follows *policy*. The policy is set from outside the AM and is stored into its memory. Policies are defined in a standard way, making it possible to share them across different AMs. Thus, policies are the method for human operators of taking influence on the system.

Both MCs and AMs provide *sensor/effector interfaces* for other AMs and other components to use. Using these interfaces components can be composed together in a manner that is transparent to the managed resources. By this means elements can communicate with each other via standard interfaces. To manage other elements, each AM utilizes a *control loop*. The control loop is an automatic method to collect details from the system by its sensors and to autonomously keep the parameters of the managed elements by its effectors in the desired range. Although the internal structure of an AM is not prescribed, it must implement the following functions to provide the control loop (see [IBM04]):

- A *monitor function* that collects, aggregates, filters and reports details gathered from a managed resource (by sensors).
- An *analyse function* to correlate and model complex situations and to enable the AM to predict future situations.
- A *plan function* that derives the actions needed to achieve the objectives.
- An *execute function* to execute the current plan (by effectors).

IBM refers these four parts as the *MAPE* (monitor, analyse, plan, execute) model. Additionally, data obtained by sensors/effectors as well as knowledge created out of this data is stored inside an AM as *knowledge*. Thus the creation of knowledge can incorporate the ability to learn (by the analyse function). Therefore the knowledge of an element consists of problem determination knowledge, policy knowledge as mentioned above and knowledge of the topology surrounding an element. An overview of the generic architecture of an AM featuring the MAPE model is shown in figure 2.5.

#### So, what is the difference between OC and AC?

Both AC and OC are motivated by the looming complexity crisis in computer systems design: the proliferating number of interconnected devices and subsystems is assumed to confront the classical approach of system design with an insolvable problem. AC as well as OC search answers to address these upcoming challenges. Compared to OC, the

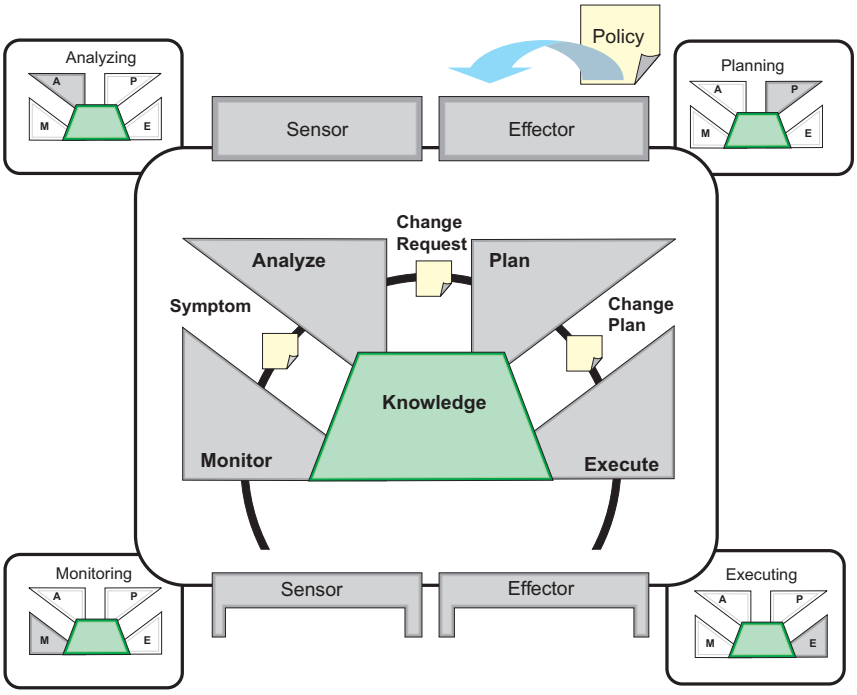



Figure 2.5: Generic architecture of an autonomic manager (from [IBM04, page 19])

AC approach focuses on the challenges arising from managing future server architectures whereas OC attends in the complexity evolving from ubiquitous computing.

The postulation derived from the complexity increase, however, is basically the same in both approaches: They ask for organically inspired computer systems that manage themselves and adapt to changing environmental conditions. For this reason these systems feature several self-x properties as for instance the ability to heal themselves in case of error. Both AC and OC present a means to control the self-organizing behaviour of the system: AC utilizes the MAPE approach whereas OC applies the observer/controller architecture. These two concepts work basically in the same way. They provide the opportunity to direct the process of self-organization: in OC the controller is guided by a goal function while in AC an AM is guided directly.

A difference in view of the way the systems handle emergent phenomena is notable though: OC systems look upon emergence favourably as well as negatively. An OC system makes explicit use of emergent phenomena and will foster its occurrence if the phenomenon is desired; emergence is a central concept in OC systems. AC puts much less emphasize

on studying the mechanisms of emergence but does consider emergent behaviour to be of importance at the design of an AC system.

Both OC and AC aim at self-organizing computer systems. However, the expected benefit from this vision is slightly different: The overarching goal of OC is to simplify the use of equipment and services whereas AC intends mainly to optimize performance and exploit redundancy. AC is much more business focused and regards self-organization as a chance to shorten IT management costs. Additionally, IBM expects the enabling of new business models like the so-called "e-sourcing" from self-organization. In this model, software modules/services can easily be integrated into (and removed from) an existing computer system, allowing for pay-per-use based business models.

### 2.4.2 Collaborative Research Centre 614

#### Overview of the Approach

The Collaborative Research Centre 614 (SFB 614) "Self-Optimizing Concepts and Structures in Mechanical Engineering" is a research project funded by the German Research Foundation (DFG). The SFB 614 has the long term goal to open up the active paradigm of self-optimization to mechanical engineering systems and to create a set of tools to support the development of such systems. It was set up in July 2002 and is scheduled to run for ten years. For further details we refer to the project homepage [Uni02].

For the future, the SFB 614 expects machines with partial inherent intelligence enabling them to *self-optimize*. Self-optimization of a technical system is here understood as (1) the ability of a system to adapt endogenously its objectives to changing environmental conditions and (2) to purposively adapt its parameters, its structure and thus its behaviour. The inherent intelligence will allow these machines to act and react autonomously and flexibly to changing operating conditions to a greater extend than familiar rule-based and adaptive strategies.

Intelligent mechanical engineering systems aspired by the SFB 614 are based on mechatronic systems as suggested by Lückel in [HSNL97] extended by the aspect of self-organization. At the lowest level such a system is constituted by "mechatronic function modules" (MFM), consisting of a basic mechanical structure, sensors, actuators, and a local information processor containing the controller. The task of such a controller is to assure

the desired function of the MFM by processing measurements and reacting accordingly.[10] MFMs can be hierarchically structured with one MFM being the actuator of another. A combination of several mechanical or logical[11] connected MFMs forms an "autonomous mechatronic system" (AMS). An AMS does also possess a controller, which deals with high-level tasks such as monitoring, fault diagnostics, and maintenance decisions. It interacts with the environment and manipulates the controllers of the underlying MFMs. A combination of some AMSs constitute a so-called "network mechatronic system" (NMS). In such a system, the single AMSs are coupled solely logically by information exchange and comprise a controller carrying out higher-level functions in the same way as that of the AMS.

In order to realize the above-mentioned extension of the classical architecture by self-organization, the controllers of the MFMs, AMSs and NMSs can be replaced by enhanced controllers implementing the so-called *Operator-Controller Module* (OCM). The concept of the OCM is described in the next section.

### The Operator-Controller Module (OCM)

The ambition is to construct complex mechatronic systems, which self-organize calls for a system architecture with inherent partial intelligence. The *Operator-Controller Module* (OCM) is the core element to answer this ambition. An OCM is basically an advanced "conventional" controller enhanced with the ability to self-optimize. Its basic structure is shown in figure 2.6 and and will be considered in more detail now (see also [Gau05] and [HOG04]).

**Controller:** This is the lowest level of the OCM. It is a control loop that processes signals and produces direct control output. Therefore it is called "motor loop". It works quasi-continuously, i. e. under hard real-time conditions. The controller may be made up of several controllers which can be interchanged.

**Reflective Operator:** The reflective operator monitors and controls the controller. It modifies the controller instead of accessing the system directly by actuators. It can change parameters of the controller or switch between certain controller configurations (represented by "configuration control" in figure 2.6). Its close connection

[10] For a concise outline of the architecture of a mechatronic system in terms of SFB 614, refer to [Uni02, Gau05, OHKK02].

[11] In terms of a connection based on information technology.

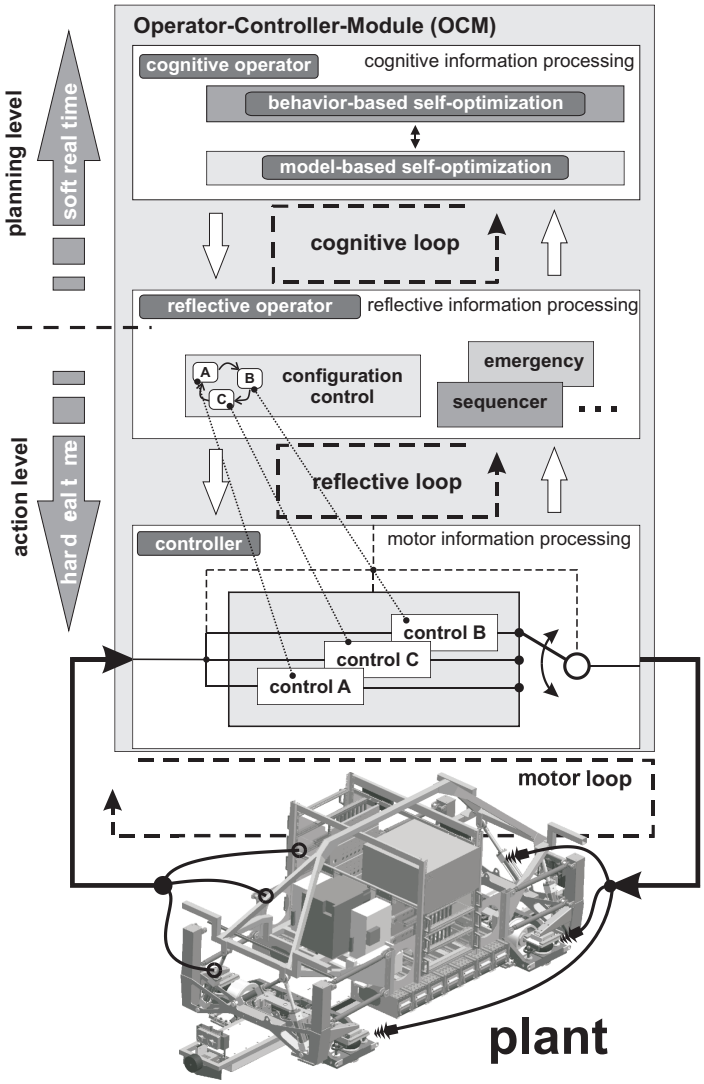



Figure 2.6: Operator/controller module (from [HOG04])

to the controller demands a hard real-time processing of events by the reflective operator. Additionally, it serves as an interface between the controller and the cognitive operator which operates in soft real-time. It filters incoming signals and inputs them to the subordinated levels. It also handles the communication to other OCMs in real-time.

**Cognitive Operator:** On the top of the OCM the system can employ a variety of methods (such as learning methods, model-based optimization or knowledge-based systems) to use information about itself and its environment in order to improve its own behaviour.

The *self-optimization process* can be carried out in several ways in the OCM architecture. If it has to fulfil real-time requirements, self-optimization is performed in the reflective operator in a comparatively simple manner. Systems that do not need real-time adaptation can perform self-optimization asynchronously in the cognitive operator using more complex methods. In this case the reflective operator synchronizes the adaptation with the controller. There are also hybrid forms where the two described forms of self-optimization take place simultaneously. Normally, optimization is performed not in real-time, i. e. without permanent interaction with the system.

As the intelligence[12] of the overall system resides within each OCM, it has to be distributed among all of them. Consequently external goal definitions are distributed at runtime top-down to the OCM through the hierarchy. These external goals form in conjunction with objectives defined at design time the internal objectives of each OCM.

### Difference to OC

The OCM approach reminds of the o/c architecture to a certain extend. In fact, it pursues a similar aim: To enable the system[13] to adapt to changing conditions. For this reason OC systems feature the o/c architecture and the SFB 614 introduced the OCM. However, there are considerable differences among these two concepts:

- The OCM focuses on the self-optimization of the system whereas the o/c approach puts much stronger emphasize on self-organization and with it the self-x properties. In fact self-optimization is merely a goal at all in OC. Its foremost aim is rather to

[12] In other words: the ability to self-optimize.
[13] At this comparison the term system refers to the SuOC and the mechatronic system, respectively.

construct systems that remain manageable at future developments than to provide a near-optimal functionality.

- On MFM level each OCM controls one single element whereas the o/c supervises a whole system of components (the SuOC) on each level of the hierarchy. An OCM controlling an AMS or NMS, respectively, exerts influence on multiple components, though.
- In contrast to the described split-up of the "control loop" of an OC system into an observing and a controlling instance, the OCM is divided into the three levels. The o/c is separated in a logical way whereas the OCM is segmented in units of different "distances" from the system: "distances" in terms of how direct their influence[14] is on the underlying system. However, the potential effect of both generic concepts appears to be the same (on condition of identical aims).

### 2.4.3 Model Predictive Control

*Model Predictive Control* (MPC) is an advanced method of process control. Camacho and Bordons define MPC in [CB04] as follows:

> *"The term Model Predictive Control does not designate a specific control strategy but rather an ample range of control methods which make explicit use of a model of the process to obtain the control signal by minimizing an objective function. These design methods lead to controllers which have practically the same structure and present adequate degrees of freedom."*

MPC is based on the following ideas (as stated in [CB04]):

- MPC makes use of a model of the underlying system to predict the future outcome of the system. The model is mostly obtained by *system identification* (see [Mac02] for further details)
- A control sequence consisting of the planned, future moves is calculated by minimizing a defined objective function.
- MPC follows a *receding strategy*, which means that at each instant the horizon is shifted one step towards (in the future). Only the first step of the strategy is implemented, then the system state is sampled again and the calculation of the control sequence is repeated.

[14] Another possible statement could be: "distances" in terms of how far reaching their actions are.

Therefore, MLP is also called *receding horizon control*. As a solid introduction in MPC would go far beyond the scope of this work, this very brief outline of the general concept of MPC should be considered as an incentive for further reading in this area. A comprehensive overview of MPC is given for example by [CB04] and [Mac02].

## 2.5 Lift Group Control

In the previous sections an introduction of the theoretical context in which this thesis resides was given. On the whole, the introduction focused on self-organization including accompanied phenomena and methods/concepts to cope with them. Now the focus will be shifted to the field of lift control and some of its (for this thesis) relevant peculiarities.

### 2.5.1 General Terms

There are some common technical terms within the domain of lift traffic control that will be used in the following. Most of them should be self-explanatory, however, the following terms might need some explanation:

**Landing/Hall Call** Landing calls (also referred as hall calls) represent a floor at which a passenger is waiting to be served. They are registered by passengers from outside the lift for example by activating the "up button" at the main terminal. If up and down buttons are installed, *up hall calls* and *down hall calls* are distinguished.

**Car Call** A car call represents the desired destination of at least one passenger within a lift. It is mostly registered inside a single lift by a passenger pushing the button corresponding its desired destination. Some present lift systems provide means to register car calls from outside the lift (see section 2.6.5). Within lift traffic control, a lift cabin is often called "car".

In the field of lift control the so-called *traffic pattern* is often of major importance on the effectiveness of control strategies (see [Bar03, section 4.4]):

**Uppeak** The lifts arrive at the main terminal, load with passengers, and move up the building stopping often until the last stop when they express return to the main terminal. This pattern is observed at the beginning of a workday and thus often called *morning uppeak*.

**Downpeak** The lifts stop at a few floors in the building, loading with passengers and then perform an express run back up the building. This pattern is observed at the end of a workday and thus often called *evening downpeak*.

**Balanced/Random Interfloor Traffic** There is no discernible pattern for a balanced interfloor traffic condition. This is the normal course of operation during the majority of a workday.

### 2.5.2 A Brief History of Lift Group Control

In the following a brief overview of historic and state-of-the-art lift group operation systems will be given to grant a better understanding of lift group control in general. The overview is based on [PH98] and [Bar03].

In the early days there were no lift systems at which the lifts had been controlled as a group. Lifts were operated manually as "one lift systems". At the very beginning hydraulic or steam-driven lifts were operated by a rope that ran the length of the hoistway and actuated a valve in the basement (*"shipper rope"*).

With the introduction of electric lifts, the rope was replaced by a switch in the car which when operated would electrically start the car in the up or down direction, and if centred would stop the lift. This so-called *car switch operation* was available in office buildings until the early 1920s. The switch was still handled by human operators, but the passengers could register their landing calls at the single floors and hereupon a human "lift starter" at the main floor could direct the lifts. He had an information panel at which the landing calls and the current positions of the lifts were shown and informed the operators within the lifts by signalling systems where they had to stop. Thus, the lifts were (manually) controlled as a group by a central authority. The lift starter also tried to maintain a time spacing of the lifts to overcome bunching by letting certain lifts wait for some time at the lobby. However, due to the manual control this was faulty and not very effective.

To realize higher travelling speeds of the lifts, in the 1920s *signal operation* enhanced car switch operation by automating the starting and stopping procedure of the lifts. Now, the lifts stopped at the floors where a landing call had been registered directed from the lift starter at the main terminal. Anyhow, there were still human operators within the lifts to safely close the doors. Parallel *automatic operating systems* were installed in apartment houses which provided a fully automatic lift control. Whereas they were only able to serve

one call at the same time in the first instance, they were even able to "collect" several landing calls automatically later on.

In 1949 the first completely automatic, operatorless lift was introduced combining the advantages of the automatic operating systems within residential buildings and the signal operation systems within large office buildings. Within this so-called *group automatic operation* the human lift starter as the former central dispatching authority was replaced by an automatic scheduling controller. Also elaborate dispatching strategies were developed[15] and several rudimentary provisions to cope with bunching were introduced[16]. The potential of those fully automatic systems surpassed that of the human lift starter since many more factors could be considered and weighted for a better system performance.

Finally, beginning in 1975 the use of *microprocessors* with programmable features changed the lift operational controllers dramatically: The controllers needed no longer to be hardwired with a myriad of relays to translate the control schemes that become highly sophisticated in the meantime. Moreover, the microprocessor provided means to easily change the control logic afterwards to adapt the system to changed requirements. It also allowed for more advanced control strategies due to the higher computing power. For example, "relay logic" dispatching systems allocated landing calls largely based upon the spacial distance between the lifts and the landing calls and hence did not consider the actual travelling time. The travelling time, however, is influenced by car calls that could delay lifts to a great extend making it possible that the spacial closest lift is not necessarily the temporal closest. This was a major drawback that very often led to bunching[17] and was overcome by microprocessor controllers that based their allocation on how fast it takes a lift to answer a landing call. Other significant improvements are real-time monitoring of lift activity as well as intelligent dispatching algorithms which adapt to changing traffic patterns and predict the traffic demand for the near future. One characteristic still remains the same though: as each of the above presented lift group control systems, state-of-the-art microprocessor controlled systems feature a central authority directing the lifts.

Over the years many features have been incorporated in lift systems to answer the demands especially of high-rise buildings. Some of the most important and widespread will be mentioned in the following:

[15] A good overview of classical group traffic control is given in [Bar03, section 10.4].
[16] As for example *zoning* which is introduced in section 2.6.5.
[17] The phenomenon of *bunching* will be explained in section 2.6.

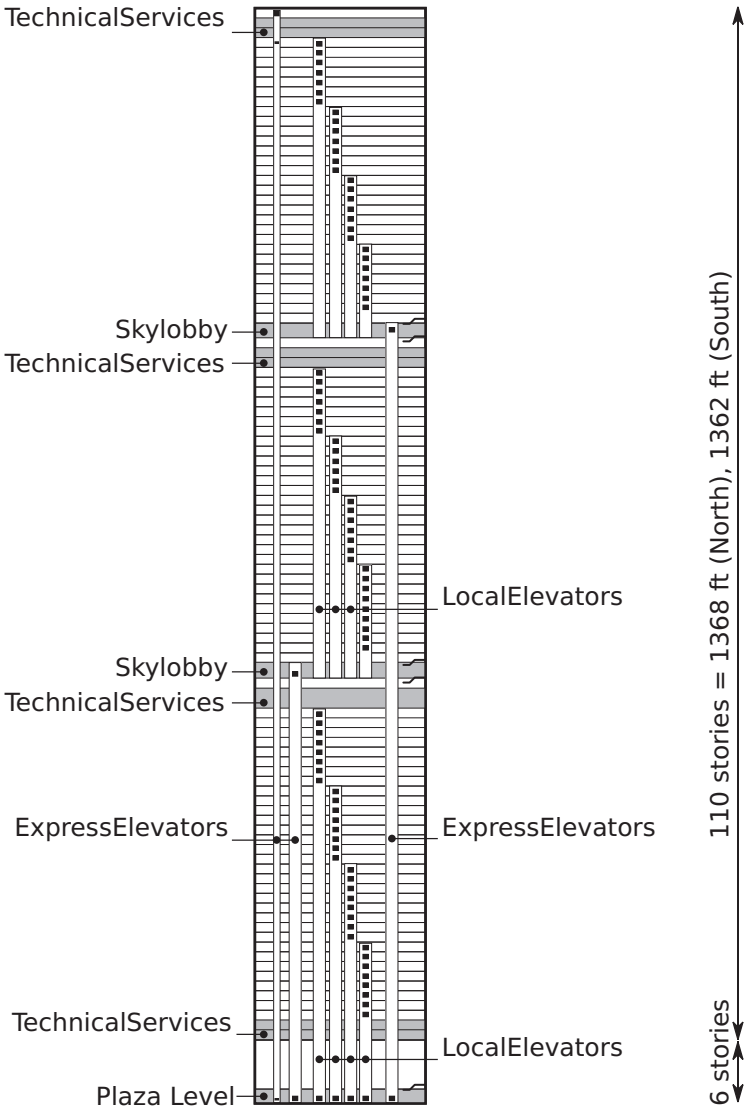



Figure 2.7: The lift arrangement of the World Trade Center (from [WTC07])

So-called *sky lobbying* is used in many very large buildings to reduce the space needed for lift hoistways (see [Str98, chapter fourteen] and [Bar03, chapter eight]). The building is separated in "sub-buildings" called *stacks* with an own terminal floor (the *sky lobby*) and local lifts serving the floors above the sky lobby. The main lobby is connected with the sky lobbies via express lifts serving only these lobby floors. A typical example for a building with sky-lobbies was the World Trade Center (WTC). The lift arrangement of the WTC is shown in figure 2.7. Sky lobbying greatly reduces passenger travel times in very tall buildings due to the reduced number of lift stops. Additionally, the local lifts of the single stacks use the same hoistway, reducing the core space needed for the lifts at the floors.

To improve the handling capacity of a hoistway, *double-deck lifts* were introduced for the first time in the early 1930s but did not take hold not until the 1970s (see [Str98, chapter fourteen]). A double-deck lift consists of two lift cabins with one attached on top of the other. Such a lift can serve two consecutive floors at the same time. Typically, passengers destined for the odd-numbered floors enter the bottom deck and those for the even-numbered floors enter the upper deck. A double-deck lift has one main advantage: it provides more passenger handling space per hoistway, making it possible to reduce the number of lift shafts in a building at a constant handling capacity of the lift system. This results in less core area for the lifts, thereby increasing the rentable space in a building. Another advantage is the excellent uppeak performance of such systems as the number of typical stops is halved. Therefore double-deck lifts are used in many buildings, especially in very tall skyscrapers as the Taipei 101 (see [Tai07]) or the Burj Dubai (see [Bur07], presently under construction). For the Burj Dubai, engineers considered installing the world's first *triple-decker lifts* (see [Bui07]). However, the actual building will use double-decker lifts. When construction is finished, the Burj Dubai will be the tallest man-made structure in the world measuring 808 meters (estimated height of antenna, see [Bui07]). The observary lifts in the Burj Dubai will be the worlds fastest lift installation and will have the the worlds longest travel distance (see [Bur07]).

Another recent approach on increasing system performance is *TWIN* from ThyssenKrupp Elevator (see [Thy, Thy07]), the first lift control system operating two lifts independently in a single hoistway. TWIN incorporates destination control, i. e. passengers register their target floor before boarding the lift. It increases the handling capacity of the lift system at a constant number of lifts or reduces the number of required hoistways with the same handling capacity: 25 percent less hoistways are required as claimed by ThyssenKrupp

(see [Thy07]). TWIN is recommended for buildings heights from 50 to 200 meters and performs best if two main entrances exist within a building. Although first attempts with two cars in one shaft were made in the early 1930s, TWIN was introduced seventy years later in 2003.

One of the newest developments in the field of lift systems are *lifts driven by magnetic levitation* (maglev) instead of steel cables (see [Löf07]). This technology is similar to maglev trains like the Transrapid (see [Tra07]) and could be the answer to a difficulty occurring at future very tall skyscrapers: because of the immense height of such structures, the steel cables used to hoist the lifts are becoming such heavy, that they could finally tear. The reason for that is simple: a cable has to lift its own weight which grows with increasing length. As the function of maglev lifts is completely unaffected by the building height, they seem to be perfectly suited for future "supertall" skyscrapers. A first step to maglev lifts will be made by the Toshiba Elevator and Building Systems Corporation, who introduced a lift system at which guide rails and steel springs are displaced by "non-contact magnetic suspensions" (see [Tos07]). For now, fully maglev driven lifts operate only in testing environments, though.

## 2.6 Bunching

OC systems are designed to cope with emergent phenomena as described in section 2.2 by the o/c architecture described in section 2.3. This diploma thesis covers the implementation of a lift group control scenario in terms of OC. At this scenario, as well as at (simple) lift group control systems in practice, arises under certain conditions a phenomenon called *bunching*.

### 2.6.1 What is Bunching?

The probably most common definition of bunching is given by Al-Sharif in [ASB92a]:

> *"When the time interval between cars leaving the main terminal is not equal, bunching occurs and degrades the performance of the lift system. [...] A typical case of bunching can be seen in lift systems when the lifts start following each other (or even frog-leaping), as they answer adjacent calls in the same direction. This has a detrimental effect on passenger waiting time. The ultimate case is*

> *when all the lifts in the group move together, acting effectively as one huge lift with a capacity equal to the summation of the capacities of all the lifts in the group."*

Bunching is defined by Barney [Bar03] and Strakosch [Str98] in a very similar way. This definition is based on observing the lift arrivals at the main terminal and gives no global overview of the spatial distribution of the lifts throughout the building at a certain point of time. Hence it is only of limited use for the intended online monitoring of bunching.

Bunching can also be observed in a different way: for example simply by discovering an "unequal distribution" or "synchronization" of the lifts throughout the building. This "definition" is used for example by Hikihara and Ueshima in [HU97].

Another possibility is a definition derived from Al-Sharif (stated above): observe the departure interval at an arbitrary floor (in contrast to the main floor as in Al-Sharif). This definition is stated in the following and will be regarded as the basis for future use of the term "bunching":

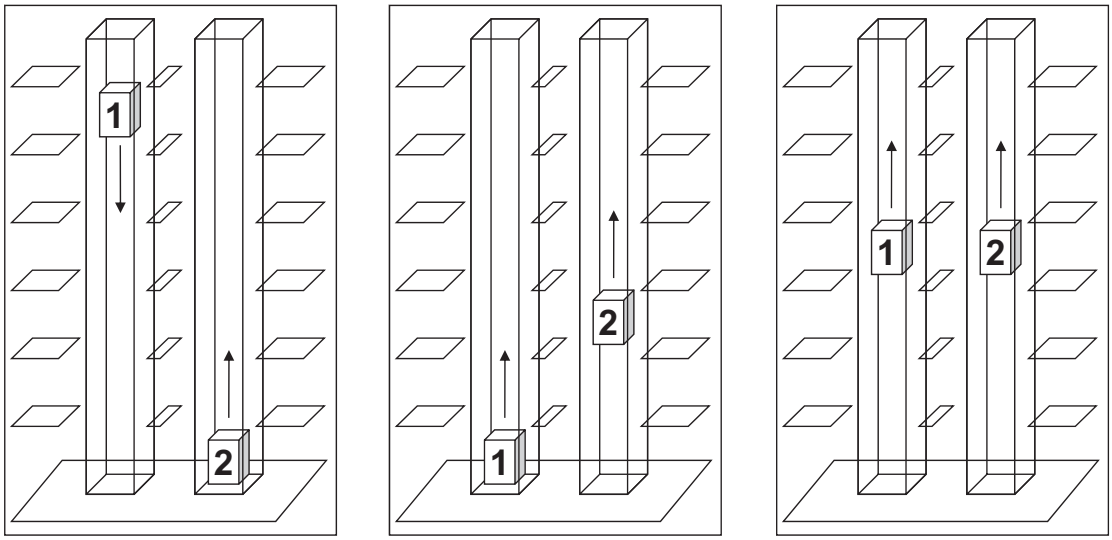
> *"Ideally, lifts arrive at a floor with a certain separation time equal to an average interval.[18] However, under heavy traffic load conditions the lifts tend to "synchronize" and serve the floors at unequal intervals in form of a wave. In an extreme case the lifts behave like a huge, single lift with the capacity equal to the sum of the individual lifts. This phenomenon is called bunching."*

Although sounding almost equal to Al-Sharif's definition, there is a need for this definition. It will become clear at the description of the developed bunching measure. A situation where two lifts exhibit bunching is shown in figure 2.8: on (a) the lifts are "well distributed" throughout the building, on (b) lift 1 begins to follow lift 2, and on (c) the two lifts move like a single lift.

As bunching itself cannot be expected beforehand nor can its occurrence be predicted from the system description, it is considered as an *emergent phenomenon* as defined in 2.2.2. See [HU97] for further details.

There will be defined a measuring metric for bunching later on in section 3.4.1 which is used within the implementation of the lift simulation.

[18] The reason for this aim will be explained in section 2.6.2.




(a) No bunching (b) Partial bunching (c) Complete bunching

Figure 2.8: Bunching of lifts

### 2.6.2 In which Way Does Bunching Affect System Performance?

Lift traffic control systems are trying to maintain an equal distribution of the lifts throughout the building. This is done because under usual conditions the best system performance is achieved when every floor is served by a lift at an equal interval. Every deviation from the equal distribution leads to a lower system performance.[19] In the context of lift systems, *system performance* is measured in terms of *average waiting time* (AWT) of the passengers (see [Bar03, section 11.4.2]). This performance measure is also called *quality of service* (see [Bar03, section 6.2]) and is the standard objective at office buildings. Another reasonable measure could be the *quantity of service*, i. e. how many people can use the lift system over a defined period of time. It is represented by the *handling capacity*. There are other performance measures as for instance the total car travel time or the consumed energy of the lift system, which are normally not suitable in practice, though. In general, the passengers' AWT is accepted as the most important performance indicator of a lift system. See [Bar03, section 11.4.2] for details.

Even though bunching degrades the system performance in terms of AWT, it does not necessarily reduce the system's handling capacity. If all waiting passengers are served,

[19] A sample calculation is given in [AS93].

the handling capacity of the lift system is not affected despite bunching. However, due to unequal lift arrival intervals the number of waiting passengers will vary among the intervals. As a constant passenger arrival rate is assumed[20] at every floor, the waiting queue will be larger at longer intervals. If by this reason the number of waiting passengers exceeds the space available in the lift, the handling capacity is degraded as less passengers can be transported at a certain time. Furthermore, this effect does also compromise the quantity of service to a greater extend as now some passengers have to wait until the next lift arrives.

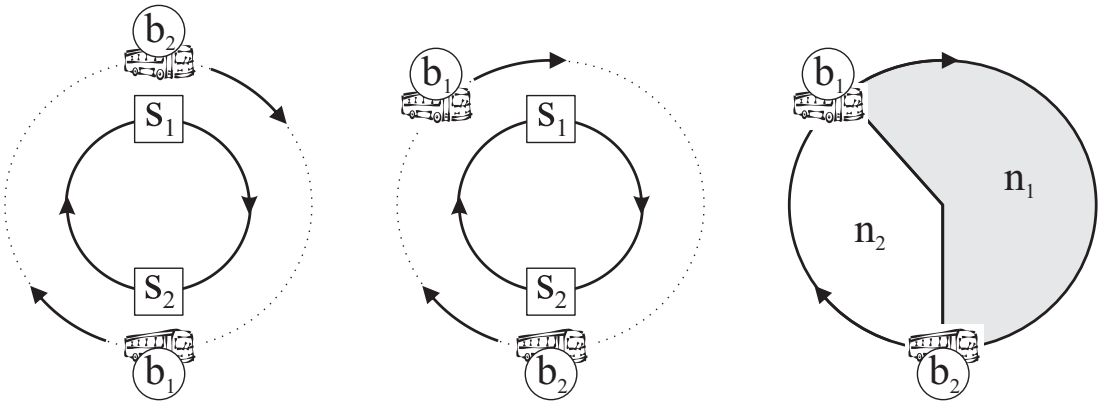
However, there are situations where bunching could be absolutely desirable. For example the following scenario could be imaginable: within an office building there is a floor containing several meeting rooms. At the end of a meeting there probably will be a crowd of people wishing to have a snack at the cafeteria located some floors apart. If in this situation several lifts arrive at the same time or closely consecutive all participants could board (nearly) at the same time, resulting in a very low average waiting time of the passengers. However, if not all meeting participants are able to enter the lifts the remaining have to wait a considerable amount of time. Below the line the average waiting time could still be less than at spaced lift arrivals if service to the other floors of the building is not affected too much. Although intuitively in a way reasonable, bunching is presumably counter-productive at the morning up-peak as passengers have to wait a serious amount of time between the lift arrivals and thus raising the waiting time on the average. All in all bunching could make sense but at most conditions it probably does not.

### 2.6.3 How Does Bunching Arise?

Now, that it has been clarified in which way bunching affects the system performance, still one question remains: how does bunching actually arise? The effect is probably best explained by means of a simple example considering buses (The presented example is derived from [She06]):

Imagine two buses $b_1, b_2$ serving a route with length $l$ containing two stops $s_1, s_2$. The route is in form of a loop and both buses circle the loop in the same direction (within this example clockwise, w.l.o.g.). The two stops are "equally distributed" over the loop, i. e. the distance between them is in each case half the loop: $h^*_{12} = [s_1, s_2] = \frac{l}{2}$ and

[20] This is a reasonable assumption for general conditions. Refer to [Bar03, section 5.6] for further details.




(a) Initial setup (b) bunching begins (c) waiting passengers

Figure 2.9: Bunching with buses

$h^*_{21} = [s_2, s_1] = \frac{l}{2}$. The buses travel with the same speed, and it takes for both buses the same time $t^*$ to travel from one stop to the other, i. e. the interarrival times $t_{12}$ and $t_{21}$ between buses $b_1$ and $b_2$ (or between $b_2$ and $b_1$, rsp.) at stop $s_1$ or $s_2$ are equal to $t^*$. To minimize the AWT, buses depart from the stops at equal time intervals $t^*$, i. e. they maintain a distance of $t^*$ or $\frac{l}{2}$, respectively. This initial setup is illustrated in image (a) of figure 2.9.

Let's assume that, caused by a random event as for example an old person who takes longer to alight, $b_1$ falls slightly behind schedule. Now the distances among them are shifted a little bit: $b_2$ seems to "chase" $b_1$. This case is illustrated in image (b) of figure 2.9. As a constant passenger arrival rate is assumed and $h_{21} < \frac{l}{2} < h_{12}$ and thus $t_{21} < t_{12}$, there will be more people waiting for $b_1$ than for $b_2$ at the next stop. In fact, for the number of waiting passengers $n_i$ for bus $i$ will hold: $\frac{n_1}{n_2} = \frac{t_{12}}{t_{21}} = \frac{h_{12}}{h_{21}}$. This is visualized in picture (c) of figure [RMB+06] with the sectors of the circle representing the $n_i$. Now $b_1$ has to spend more time than on average on picking the passengers up whereas $b_2$ saves time at every stop (since more passengers need more time to board). As a result, $b_2$ will catch up and pretty soon (with decreasing $t_{12}$ (or $t_{21}$)) the two buses will travel in a kind of a convoy with the first bus full and the second bus empty.

This scenario can easily be adopted to lift traffic control where events like holding the door open to let another passenger board causes delays within the schedule and causes the lifts to bunch. The common objective is to keep lifts in the group as far apart as possible

during usual traffic conditions. Within the example stated above, bunching is clearly a self-aggravating situation that after a slight incentive develops on its own (like wildfire for example). This observation holds also in practice as stated in [Bar03, section 12.6].

Above all, bunching depends on the relative system load. Particularly at higher loads lift systems tend to bunch if no countermeasures are taken. This correlation can be seen in figure 2.10 as explained subsequently. Another source of information for bunching detection is Hikihara and Ueshima in [HU97].

Aside the above stated bunching cause, another possible factor worth mentioning could foster bunching. It is based on the different number of passengers in the lifts. In the mean, in delayed lifts there will be more passengers in the lifts than in early lifts as more passengers are waiting at the stops for delayed lifts (analogous to the above explained bus example). More passengers will probably force the lift to stop more often as more passengers are most likely destined to a higher number of different floors. This way, a delayed lift could be slowed down in comparison to other early lifts. However, this factor will be less serious in heavy traffic conditions when most lifts are near to fully loaded and on most floors being landing calls registered forcing the lifts to stop very often.

In addition to these common causes for bunching, the actual factors being responsible for bunching lift behaviour will be explained in section 3.3.5.

### 2.6.4 Measuring Bunching Quantitatively

Measuring the extent of bunching quantitatively is quite essential in order to compare the performance of different systems and to be able to take appropriate countermeasures. There are some methods to measure bunching. Notable in this field is the work of Al-Sharif, who has published several articles concerning the bunching of lifts and in particular its measurement. He focuses on *uppeak* traffic conditions.[21] However, the insights gained from the observation of bunching at uppeak can be adopted to general traffic conditions.

Al-Sharif distinguishes between *simulation based* and *formula based* measurement (see [AS96]). Simulation based methods measure bunching afterwards by calculating a kind of a "performance index" derived from observed property values. Formula based methods measure bunching during operation by monitoring/calculating certain values and comparing them with predefined ideal values. The calculated deviation indicates the amount of

[21] An overview of different traffic conditions is given in section 2.5.1.

bunching. These two measurement concepts will be presented in more detail now (see [ASB92a, ASB92b, AS93]).

Within the calculation formulas some variables will be used that might need explanation:

**RTT** The *round trip time* is the time in seconds for a single lift trip around a building from the time the lift doors open at the main terminal, until the doors reopen, when the lift has returned to the main terminal floor (see [Bar03, Definition 4.8]). The round trip time comprises boarding and alighting of passengers. Therefore, it depends on current passenger arrivals.

**INT** The *interval* is the average time between successive lift arrivals at the main terminal floor with lifts loaded to any travel (see [Bar03, Definition 4.9]). The following equation is valid for perfectly distributed lifts: $INT = \frac{RTT}{L}$ at which $L$ is the number of lifts in a group.

**AWT** The *average waiting time* is the average time in seconds a passenger has to wait for a lift. It is measured by the time span from the instant that the passenger registers the call (pushing the button outside the lift) to the instant the passenger can enter the lift (see [Bar03, Definition 6.6]). The following equation holds for a uniform passenger arrival rate and perfectly distributed lifts: $AWT = \frac{INT}{2}$.

### Simulation Based Measurement

The performance of a lift system can be expressed by the ratio of $AWT$ to $INT$ at which $AWT$ is an observed and $INT$ a calculated, theoretical value (see [AS93]). The division by $INT$ is a method of normalization. As seen in figure 2.10, $\frac{AWT}{INT}$ is for loads below 50 percent a linear function but raises increasingly for higher loads. The loading factor is expressed as the ratio between the number of passengers and the rated capacity. The linear relationship can be extrapolated as shown in figure 2.10 by the dotted line. It is proposed that the departure from the linear behaviour is caused by bunching of the lifts. On this basis a measure of bunching is defined as the bunching coefficient $bx = \frac{AWT}{AWT^*}$ at which $AWT$ is the observed waiting time and $AWT^*$ is the theoretical waiting time of the passengers at perfectly distributed lifts.

A $bx$ value of 1.0 represents no bunching conditions and a value higher than 1.0 indicates bunching. Additional to the dependence of $bx$ from the system load, $bx$ is also a function of the number of lifts in the group and will increase as the number of lifts increases. A

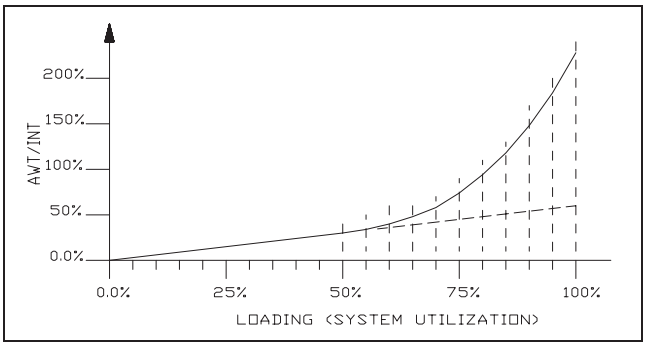



Figure 2.10: Relationship between system loading and AWT/INT, with a straight extrapolation of the linear part below 50% loading (from [AS93]).

drawback of $bx$ is that a single lift system can show "bunching" as $AWT$ still increases slightly with increasing system loading. But bunching by definition cannot apply to a group with one lift.

#### Formula Based Measurement

Additionally Al-Sharif developed a formula based bunching measure that can be measured directly from the lift system behaviour. He defines an ideal behaviour of the lift system under no bunching conditions, calculates the deviation from the ideal state and uses this deviation as an indicator for bunching (see [ASB92a, AS93]).

The ideal behaviour of a lift system is defined as the departure of the lifts from the main terminal at equal time intervals $INT = \frac{RTT}{L}$ at which $L$ is the number of lifts in the group.

The time between lift number $i$ and lift number $i+1$ is defined as $t_{i,i+1}$ and in the ideal case should hold $\frac{RTT}{L} = t_{i,i+1}$. Thus the deviation of $t_{i,i+1}$ from the ideal behaviour can be expressed as $(t_{i,i+1} - \frac{RTT}{L})$. Since positive as well as negative deviations are equally harmful, the absolute value is taken: $|t_{i,i+1} - \frac{RTT}{L}|$. The sum of all these differences among the lifts in the group is an indication of the system's deviation from ideal performance:

$$|t_{1,2} - \frac{RTT}{L}| + |t_{2,3} - \frac{RTT}{L}| + \cdots + |t_{L-1,L} - \frac{RTT}{L}| + |t_{L,1} - \frac{RTT}{L}| \qquad (2.1)$$

In order to normalize this bunching representation it is divided by its highest possible value, that is at "perfect" bunching. In this case all the lifts depart from the main terminal at

the same time. So all the $t_{i,i+1}$ will be zero, because there is no time lapse between the lifts, except for the last term $t_{L,1}$ which has the value of $RTT$ because full $RTT$ elapses between the (collective) departure of all the lifts and the next (collective) arrival of the group.

$$\begin{aligned} & |0 - \frac{RTT}{L}| + \cdots + |0 - \frac{RTT}{L}| + |L - \frac{RTT}{L}| \\ = \; & (L-1) \cdot \frac{RTT}{L} + (\frac{L \cdot RTT}{L} - \frac{RTT}{L}) \\ = \; & (L-1) \cdot \frac{RTT}{L} + (N-1) \cdot \frac{RTT}{L} \\ = \; & 2 \cdot (L-1) \cdot \frac{RTT}{L} \end{aligned} \tag{2.2}$$

From this follows the normalized bunching coefficient $BC_1$:

$$BC_1 = \frac{|t_{1,2} - \frac{RTT}{L}| + |t_{2,3} - \frac{RTT}{L}| + \cdots + |t_{L-1,L} - \frac{RTT}{L}| + |t_{L,1} - \frac{RTT}{L}|}{2 \cdot (L-1) \cdot \frac{RTT}{L}} \tag{2.3}$$

Al-Sharif introduces another bunching measure, that penalizes larger deviations to a greater extend. This is done by squaring the differences $(t_{i,i+1} - \frac{RTT}{L})$. Following the same calculations that led to $BC_1$, the normalized bunching coefficient $BC_2$ is:

$$BC_2 = \sqrt{\frac{(t_{1,2} - \frac{RTT}{L})^2 + (t_{2,3} - \frac{RTT}{L})^2 + \cdots + (t_{L-1,L} - \frac{RTT}{L})^2 + (t_{L,1} - \frac{RTT}{L})^2}{L \cdot (L-1) \cdot (\frac{RTT}{L})^2}} \tag{2.4}$$

The value of $RTT$ is continuously measured during the operation and thus adapting to changes in the system (for example changes in the passenger arrival rate). The values of $BC_1$ and $BC_2$ lie between 0.0 and 1.0 whereas 1.0 represents full bunching.

### 2.6.5 Methods to Cope with Bunching

To cope with bunching is an important task as it can greatly decrease system performance. In particular in the past within classical lift traffic control systems without a powerful computer controller this was a demanding problem. In general it can be distinguished between means within and means aside the landing call assigning algorithm.

#### Means aside the Landing Call Assignment Process

In the early days lift systems were controlled by a human operator or a rudimentary automatic controller. It was tried to overcome bunching by introducing a *variable lift dwell*

*time*, which is varying the minimum door-hold open time: To achieve an even distribution of the lifts throughout the building, the lifts dispatched from the main terminal at equal intervals with "early" lifts wait a longer amount of time than "late" lifts before their departure (see [Str98, chapter four]). The effect, however, is a decrease in handling capacity since lifts spend some time waiting unproductively at the lobby.

Control techniques like the one above are also built in modern controllers, however, with utilizing their far greater operational scope: The controller is able to let lifts dwell a little bit longer at any stop, to vary door open/close times or the maximum acceleration/velocity (if supported by the "hardware") .

Another relatively simple but very drastic approach is the *zoning* and *spotting* of the lifts (see [Str98, chapter four]). *Zoning* is used mainly at up-peak where bunching occurred to the greatest extend. The idea of zoning is to let the lifts serve only certain floors of the building: For example, in a group of six lifts let three of them serve the lower and three the upper floors. Zoning increases handling capacity at the expense of interval (INT, see section 2.6.4). Additionally it is cumbersome to travel from one zone to another as a stopover at the main terminal is required to change the lift and thus increasing passenger travel time. Therefore, zoning should only be used at periods of peak demand and never at normal traffic conditions. *Spotting* is a rather similar approach where each lift is designated to serve a limited number of floors. With the advent of microprocessor operators a technique called *channelling* was introduced: This is a sort of spotting with dynamically designating floors to the lifts based on statistical analysis. The floors served by a lift are announced by monitors at the main terminal. For a more detailed introduction in zoning, spotting, and channelling refer to [Str98, chapter four].

*Sectoring* is another classic however far more elaborate practice to keep the lifts apart. Each lift is assigned to certain floors of the building at which it exclusively collects the registered landing calls. The partition of the floors into sectors can be either static or dynamic; an introduction to static and dynamic sectoring is given in [Bar03, chapter ten]. At basic dynamic sectoring landing calls are grouped in variable sectors reaching from one lift to the next lift ahead either free or travelling in the same direction. When a sector reaches the terminal floor, it continues in the opposite direction. A free but moving lift is taken out of the normal sectoring scheme. A free lift is given two zones, one in each direction. Sectoring performs well under normal interfloor traffic whereas it has shown to be much less effective at uppeak and downpeak conditions.

Aside from distributing lifts being in service, a *distributed parking* of idle lifts at certain floors is essential for a good lift system performance. This means reduces the average distance from a lift to newly registered landing calls and thus average waiting time of newly arriving passengers. However, this strategy will result in higher energy cost due to the higher distance travelled by the lifts.

#### Means within the Landing Call Assignment Process

Another provision to reduce the effects of bunching is to assign landing calls to lifts with regard to the expected resulting bunching (which has to be measured/estimated somehow). An example is given in [MA88]: calculate the *estimated arrival time* of the lifts to travel from its current lift position to a landing call. Calls are assigned based on this calculation with respect to maintain an equal spacing of the lifts. Also already assigned landing calls should be reassigned to other lifts if the expected bunching would be significantly less.

The uncertain destinations of the boarding passengers at every landing call are a problem within landing call allocation. Therefore some modern lift control systems provide means for the passenger to register their destination outside the lifts before entering. Based on this information the passengers are grouped with regard to their destinations and then assigned to lifts (i. e. the passengers have to move to an indicated lift). The only caveat is that once a passenger entered a lift it is not possible for him to change his travelling destination anymore. Passengers have to get used to this paradigm shift which may be a problem at buildings with transient population as a hotel. However, the advantage of these systems is evident: They will not suffer from bunching as calls can be assigned to lifts based on (near-)[22]complete information. This has especially great advantages during uppeak traffic as passengers can be grouped to common destinations, if there are many of them. For more information see [Bar03, chapter twelve] and [PH98].

#### General Thoughts

The presented provisions aside landing call assignment differ in their *dependence of the actual traffic pattern*. At normal interfloor traffic, zoning shows very poor performance as the passenger travel time is unnecessarily increased without reducing the AWT (see

[22]The controller does not know what landing calls will be registered in the future. This is not a problem, however, since the controller will assign future landing calls in a bunching preventing way.

[Str98, chapter four]). Sectoring shows contrary behaviour: it performs well under inter-floor traffic, but is shown much less effective under downpeak (see [Bar03, section 10.4.2]). Therefore it is advisable to use zoning and sectoring only under appropriate traffic conditions. Intelligent parking of the lifts and introducing small variations like altering door open/close times exhibit nearly no substantial disadvantage on each traffic pattern, however, at the same time being less effective due to the gentle nature of their interference (see [Str98, chapter four]).

Hence, it can obviously be very beneficial for the performance of the lift system if the controller *recognizes the current traffic pattern* as soon as possible. Knowing the traffic pattern, the controller can safely initiate provisions which could be harmful if applied in an improper situation. For an introduction of possible provisions check up [Bar03, Str98].

Particularly at anti-bunching means based on landing call assignment it seems to be also very useful for the central controller to gather as much *information on the system* as possible as more information allow a better estimation of the current situation. If for instance the destinations of the passengers are known, the controller is able to calculate more exactly estimated arrival times of the lifts used for call assignment. Additionally passengers can be assigned to lifts in consideration of their destinations granting the controller with powerful means to influence the lifts.

However, controlling the lift from a central place is a very demanding task. In fact, this task may become even more complex when more information is available, since the state space of the system grows larger. Calculating an optimal routing strategy is especially hard as the result has to be computed in very short time: when a new landing call is registered, the controller has to decide in virtually the same moment on the routing strategy. This very short response time calls for a so-called *online optimization*. Additionally, the routing strategy has to be continuously adapted to the changing situation (new landing calls are registered etcetera). More information on online-optimization and the challenges on optimizing lift routing can be found in [GKR99].

# 3 Implementation

In this chapter the actual implementation of the observer/controller architecture developed within this thesis will be introduced. The implementation is based on an existing lift simulation program. This programme along with its underlying simulation framework *Repast* will be introduced in section 3.1. Subsequently, the *requirements* made on the simulation program, formed by the generic concepts introduced in the preceding chapter, will be outlined in section 3.2. Hereafter a description of the *characteristics of the lift simulation model* follows in section 3.3, including its limitations and peculiarities. Finally the *general structure of the program* will be outlined in section 3.4, focusing on the observer and the controller.

## 3.1 Starting Basis

In this chapter the characteristics of the underlying simulation framework *Repast* and the initial lift simulation programme is introduced. The originating programme was created by Christoph Pickardt [Pic06] and formed the starting basis of this thesis. The simulation framework Repast is presented in the next section, the underlying simulation programme in section 3.1.2.

### 3.1.1 Repast

The *"REcursive Porous Agent Simulation Toolkit" (Repast)* is one of the most used agent based modelling toolkits that is available (see [Rep07, SM06]). It was originally developed by Sallach, Collier and others at the University of Chicago in 2000. Subsequently it was expanded by Argonne National Laboratory as a reusable software infrastructure to support "rapid social science discovery" based on extensive computational experimentation. Repast is now managed by the non-profit organization ROAD (Repast Organization

for Architecture and Development). In the following a general overview of Repast and afterwards a short introduction on building models with Repast is given in section 3.1.1.

### General Overview

The Repast system is available directly from the internet at [Rep07] in binary and source code form and it is free of charge. It is implemented in pure Java (Repast J), pure Microsoft .Net (Repast .Net), and in a version supporting Python scripting (Repast Py). The Java based versions, Repast J and Repast Py, are available for virtually all modern computing platforms including Windows, Linux, and Mac OS. Both, Christoph Pickardt's lift simulation model and the lift control model created within this thesis, use the Java based implementation of Repast 3.

Repast borrows many concepts from the *Swarm* simulation toolkit (see [Gro07]). However, Repast surpasses Swarm in the number of features since it delivers for example built-in genetic algorithms and regression functions (see [NCV06]). Furthermore, Repast is published under a derivation of the Berkeley Software Distribution (BSD) license, which makes it possible to distribute simulation models in binary form without the need to release the source code. In contrast, Swarm is distributed under the GNU General Public License (GPL), which requires developers to make the source code for their entire model available.[1]

The latest development in the scope of Repast is *Repast Simphony* (Repast S). It is intended to extend the current Repast portfolio (Repast J, Repast .Net and Repast Py) by offering a new approach to simulation development and execution. However, it is not replacing the existing tools but it is rather complementary to them. Repast S provides new features like agent storage and display management to simplify the design process. At later releases, Repast Simphony will offer the ability to create basic models without Java knowledge (see [Rep07]). Actually Repast S is available as an alpha release and is still under development. More information about Repast S can be found in [NHCV05, TNH+06].

[1] For a comparison of several agent-modelling-toolkits refer to [SD02, GB02].

### Building Models with Repast

Repast is an agent based modelling toolkit providing several means and features to assist the user in building simulation models. In the following the general design ideas for building simulation models with Repast 3 is briefly illustrated on a rather abstract and simplified level. For more information on developing models with Repast refer to [Rep07]. An overview of the structure of the Repast framework itself is given in [NCV06].

As Repast is fully object-oriented, a simulation model is usually separated into several classes. In general, the heart of a simulation is a class called the *model class*. It contains for example these typical components:

- Infrastructure and representation variables concerning the whole simulation like parameter values as "how long does it take for a lift to open the door".
- Means to initialize and, if a restart is desired, to reset all objects and parameters of the simulation.
- A scheduling functionality responsible for all the state changes within the simulation. Repast utilizes a discrete time model: time elapses by cycling through so-called "ticks" at which a list of registered actions is performed. Actions are registered within a *schedule*.
- Means to display graphical output, write data to a log file, and import parameter values from a parameter file can be integrated into the model class by using Repast standard tools/libraries. The displays are realized as inner classes within the model class.
- It contains references to the agents (i. e. the agent objects, described in the following) and general relationships among them. Agents are often embedded into a *space* like for example a two dimensional grid (as a chessboard) to express their spatial relation.

A few of the above outlined points are strictly prescribed by the framework as for example the implementation of initialization and setup methods, whereas the suggested realization of some parts is "only" good practice. It depends on the preferences of the user and the demands of the model in which way the actual implementation is realized. For instance, in more complex simulation models it often makes sense to extract the agent space from the model class to a separate class to clarify the division of the corresponding methods. The goal at the development of Repast was granting the user the flexibility to build a model

following his needs and at the same time supporting him in standard tasks like scheduling single actions.

Repast can be run in *interactive* and *batch mode*. At interactive mode the user can interact with the simulation for instance by adjusting parameters or pausing/resuming the simulation. Within batch mode the simulation runs one or more times cycling through a parameter file containing several parameter values. Most notably in this regard are nested parameter variations, where several simulation runs are executed consecutively with all possible combinations of parameter settings without the need for user interactions. While doing this, parameter screening experiments as described in chapter 4 can easily be performed. During a batch run the user has no possibility to intervene the simulation and usually no graphical output is given. Especially the absence of the graphical user interface in batch mode leads to a faster operation compared to interactive mode. This is very useful if only the actual response for a predefined parameter set has to be evaluated (e. g. in form of logfile data) and no feedback during the simulation is needed. Altogether, the batch mode is a very powerful tool supporting the execution of experiments especially when several parameter combinations have to be tested.

*Agents* are typically realized as objects in Repast. The single agents are instances of an *agent class*. If agents of different types are present, there is an agent class for every type. Within a simple predator-prey example simulation (as introduced in [TNH+06]) there are wolfs (predator agents) and sheeps (prey agents) and hence a wolf agent class and a sheep agent class. The actual wolfs and sheeps are instances of the wolf or the sheep class, respectively. The state of the agents is stored within instance variables. Common behaviours of the agents as well as means to modify the state of the agents are implemented in form of methods within the agent classes. Repast provides several standard agent classes which can be extended to create own agents easily.

Furthermore Repast comes with libraries for genetic algorithms, neural networks, random number generation, specialized mathematics, social network modelling support tools and much more. Additionally, it is possible to integrate third party libraries into Repast models easily (for more information visit the project homepage [Rep07]).

### 3.1.2 The Underlying Lift Simulation

The lift group control simulation programme developed within this thesis is based on an existing simulation programme by Christoph Pickardt [Pic06] resulting from a student research project. The functionality of the programme is stated in the following paragraphs. It is presented in very short form; for more information consult the term paper.

This programme simulates a group of lifts serving passengers arriving at different floors at a user-defined rate. The number of lifts, the number of floors and many other parameters can be chosen freely. The particular strategy of the lifts can be selected from a set of predefined strategies. Unlike classical lift control systems, which are controlled by a central authority, here the lifts act fully autonomously: there is no authority controlling the lifts. The lifts in the simulation show bunching accompanied by an increasing AWT of the passengers. Further information about general characteristics of the programme can be found in section 3.3, where the advanced simulation programme is introduced.

In the following remarks, the general structure of the originating programme will be briefly outlined, as far as it is relevant for this context. Unless noted otherwise, after this section the term "simulation program" will refer to the programme developed within this thesis. The primal programme will be referred as "original (simulation) program".

The original programme is realized with Repast J and hence written in Java. It keeps quite well with the generic buildup outlined in the previous section: there is a base model class "ElevatorSimModel" that is responsible for model initialization and tasks as scheduling. The *passengers* and the *lifts* are agents of the simulation and therefore realized as instances of the classes "ElevatorSimAgent" (for the passengers) or "Elevator" (for the lifts). They are part of the *space* class "ElevatorSimSpace" representing the building. The lifts serve the passengers according to a *strategy* which can be set at the beginning of the simulation for every lift in the file "elevators.cfg". Each lift strategy is realized as a Java class which is used by the lifts.

Logging is performed by writing the current lengths of the passenger waiting queues, the positions of the lifts, the load of the lifts, and other data to the logfiles "elevatorData.txt" and "passengerData.txt".

The parameters as for example the number of floors, the number of lifts, or the passenger arrival rate are stored in the file "ESParams.txt", which is parsed at the beginning of the simulation.

## 3.2 Programme Requirements

The just outlined original simulation programme does "simply" simulate a lift system without providing any means for the user to intervene in the course of the simulation. Within the limits of this thesis, the existing simulation programme has to be extended to incorporate additional functionality. The major *requested modifications* are as follows:

- A means to control the lift system had to be implemented. This has to be done using the *generic observer/controller architecture* described in section 2.3. Hence, an observer and a controller have to be implemented.
- The *observer* has to have an ability to recognize bunching and quantify it. The quantification is necessary, because it is the prerequisite for the interventions of the controller. Additionally, the observer was provided with a means to predict the near future. This ability presumes either the existence of a system model within the observer or means to approximate the future course of the system.
- The *controller* has to be able to intervene somehow in the course of the system if bunching is detected by the observer. Interventions of the controller or its control strategies should reduce bunching.

The goal at implementing an observer/controller architecture above the lift system is, on the one hand, to create an instrument to observe the development of bunching in the system in a quantitative way (i. e. to measure bunching). On the other hand, means to control this bunching and thus improving the system performance should be implemented. System performance is expressed in terms of the passengers' AWT for reasons stated in section 2.6.2. The observer and the controller have to satisfy the general guidelines of OC. Thus, these components have to be realized with respect to certain *contraints*. These constraints are mainly the properties of the generic o/c architecture described in section 2.3:

- The controller should never be a central authority like in classical lift control systems. Therefore, the interventions of the controller have to be only of a guiding nature as presented in 2.3.2 – the autonomy of the lifts as given in the original simulation programme must not be compromised. In section 3.4.2 these interventions will be described in detail.
- Ordinary operation of the lift system has to be guaranteed if observer and controller are turned off. This constraint results from the principle that the lifts (i. e. the

SuOC) do not rely on the "control" of the controller. Most certainly there will be more bunching in absence of the controller, however, the lifts are able to fulfil their strategy further on and serve the calls.

In which way these requirements are realized within the programme will be shown in the next sections. Observer and controller will be described in sections 3.4.1 and 3.4.2 in detail.

## 3.3 Characteristics of the Lift Simulation Model

In the following the general characteristics of the simulation programme is outlined. At first, substantial characteristics like attributes of the lift system or the type of the building is stated in the next section. In section 3.3.2, the fundamental difference between lift systems in practice and this simulation model is made clear. At the simulation model, there are several simplifications compared to "real" lift controllers. They are explained in section 3.3.3. The strategies, based on which the lifts act in the building are illustrated in section 3.3.4. Finally, the cause of bunching within this model is outlined in 3.3.5.

### 3.3.1 Substantial Characteristics of the Simulated Lift System

In the following a brief overview of substantial characteristics of the simulation model is given. All the following attributes are also characteristic for the original simulation program. In fact, the originating lift simulation model has only been altered in a minor way not affecting the nature of the program. The simulated lift system exhibits the following *general attributes*:

**Type of the building** The building in which the lifts operate consists of one main lobby and an arbitrary number of floors. The lobby is the lowermost floor of the building. This is the floor, where passengers arrive during uppeak and passengers are travelling to during downpeak. There is no sky lobby or other "special" floor like a basement or cafeteria floor which could require especial attention. Every floor of the building is equal in terms of relevance for the passengers and the lifts. The main lobby would only be different from the other floors at uppeak and downpeak, however, as these traffic patterns are not used within the simulation program, the main lobby is alike to every other floor.

**Lift system** The programme simulates a group of lifts. The number of lifts can be chosen as desired, however, values should range from two to eight. Two is a reasonable minimum as a "group" of one lift does not constitute a group. Eight is the maximum number of lifts efficiently operating together in a group in practice (see [Bar03]). As stated above, every floor is served by the lifts in the same way.

**Traffic pattern** Within this diploma thesis, only balanced interfloor traffic with different passenger arrival rates is used (see next paragraph).

**Passenger arrival model** Passengers' arrival depends on the traffic pattern. During normal interfloor traffic, passengers arrive with a certain arrival rate. In this situation, the arrival model follows the *Poisson distribution*: the time from one passenger arrival to the next arrival is based on an average passenger arrival rate $\lambda = \frac{average\ number\ of\ passengers}{tick}$. This time is independent of the time elapsed since the last arrival (see [Lin04] for more information). It is shown in the next section ("Simplifications") why this arrival model is reasonable. On passenger arrival, both the floor where the passenger arrives as well as the target floor are chosen randomly. The choice is based on a *(discrete) uniform distribution*, i. e. one floor out of all floors is chosen with the same probability (see [YMS03] for more information).

**Lift strategies** The lifts act based on their currently assigned strategy. The different strategies are discussed later on.

### 3.3.2 Fundamental Difference to Classical Lift Systems

In contrast to classical lift control systems as mentioned in section 2.5, within this simulation the lifts are not supervised by a central authority. In fact, they are not controlled at all (in the classical way). Every lift acts on its own behalf and serves passengers according to its assigned strategy. That means, a lift "decides" by itself which landing call to serve (or more general: what action to perform next). This replacing of the central control authority by the intelligence of autonomous cooperation constitutes a *paradigm shift* in lift control. However, the "decisions" of the lifts are of a very simple nature and based on passing a sequence of primitive conditions. At most of the control strategies, every lift acts as if it was the sole lift in the system and the lifts do not communicate among each other.[2] The lifts act fully autonomously.

[2] Lifts acting according to the strategy "LeftNeighbourOrientation" communicate with their left neighbour and decide hereafter how to serve the available landing calls.

### 3.3.3 Simplifications

In addition to this fundamental difference, the simulation programme presents a partly simplified image of real lift behaviour. The most important simplifications are as follows:

- Acceleration/deceleration times of the lifts are completely ignored: the lifts consume always the same time to travel between two successive floors. However, this simplification does not have a significant impact on the outcome of the simulation as acceleration can be easily expressed in terms of longer dwell times. Deceleration times could render it impossible for a lift to stop at suddenly registered landing calls ahead of the lift. This could be taken into account within the lift strategies but it should intuitively not have a serious impact on the simulation outcome.
- Additionally, passenger boarding times are not taken into account. Passengers board the lifts in no time, i. e. within one tick. It does not matter how many passengers wait in front of a lift in order to board. Therefore, doors need a constant time to close (three ticks in the default setting). Passengers are able to board at every tick during the close time. This assumption contrasts with the general cause for bunching described in section 2.6.3. Reasons for doing this are given in section 3.3.5.
- Another simplification from real passenger behaviour is the fact that passengers would wait forever, if not served by a lift. That means, once a passenger registers a landing call, he will wait at the very floor until a lift picks him up. Furthermore, the passenger will not change his destination afterwards. The assumed premise of patient passengers complies with overall real-world passenger behaviour and thus is no significant restraint for the simulation model.
- As mentioned in the above sections, the passenger arrival process is modelled as a Poisson probability process which is made up of an average arrival rate. The process could have been represented by a constant arrival rate, which would certainly be an intuitive and very easy choice. However, the randomness within the Poisson arrival process is truly desired: the effect of the randomness is to cause the lifts to take different times to carry out a round trip and thus being a reason for emerging bunching. The use of the Poisson probability process for modelling passenger arrival is generally accepted. In respect of the arrival process, Barney states, “Although there may be other theoretical distributions which might better accommodate the data, the Poisson distribution must be considered as a good approximation to the actual empirical distribution.” ([Bar03, chapter five]) All in all, the Poisson arrival

model seems to be a good approximation of an arrival process in reality for normal conditions. However, special events like the end of a convention cannot be modelled by this way.

### 3.3.4 Lift Strategies

The lifts act based on their currently assigned strategy: the strategy provides the lifts with the next action to be performed taking into account the current system state. Each strategy is a sequence of simple decisions resulting in an order to be executed by the lift. No matter what strategy a lift is following, it always has to obey a set of basic rules, which the passengers accept and understand (see [Bar03, section 11.2.1]):

1. "Car calls *always* take precedence over landing calls."
2. "A lift *must not reverse* its direction of travel with passengers in the car."
3. "A lift *must stop* at a passenger destination floor (it must not pass it)."
4. "Passengers wishing to travel in one direction *must not to enter* [sic] a lift committed to travel in the opposite direction."
5. "A lift *must not stop* at a floor where no passengers wish to enter or leave the car."

These rules are obeyed by real world lift control strategies as well as by all strategies incorporated within the simulation program. In the originating simulation programme there are five strategies implemented:

- Collective control (Default) – used in the program
- Local search
- Collective control with information exchange (LeftNeighbourOrientation) – used in the program
- Zoning (upper zone)
- Zoning (lower zone)

From these five strategies, "collective control" and "collective control with information exchange" are used within the advanced simulation program. In the following, they will be called "Default" and "LeftNeighbourOrientation". These terms correspond to the names of the Java classes of the strategies. Due to the functionalities added to the program, these two strategies had to be changed in some minor way. Hence, the remaining three strategies cannot be used in the simulation programme without applying these slight changes. Anyway, these modifications are solely caused by technical issues and do not

affect the nature of these strategies. These modifications could be easily performed to analyse the other strategies.

In the following the strategies "Default" and "LeftNeighbourOrientation" will be outlined. Since not used in the simulation programme and for the sake of brevity, we refer to [Pic06] for a description of the other three strategies.

**Collective control ("Default")** "Default" is a rather simple strategy, as within this strategy the lifts serve the landing calls on all floors sequentially in travelling direction. That means, there is no ordering or sorting of the calls: the calls are served in travelling direction one after another. During operation (i. e. with passengers inside), the lift stops at every floor where a landing call is registered if no intervention by the controller exists. These interventions are described later on in section 3.4.2. If empty, the lift searches for landing calls firstly in travelling direction and in the opposite direction afterwards. In compliance with the basic rules stated above, this is the only situation in which the lift can change its travelling direction. The lift will remain on the current floor, if no landing call is found (and the lift is empty). Additionally, a lift following this strategy is not affected by the behaviour of the other lifts in the group. It will act, as if it would be the sole lift serving the landing calls in the building. The exact sequence of decisions forming the "Default" strategy is shown in figure 3.1.

**Collective control with information exchange ("LeftNeighbourOrientation")** The strategy "LeftNeighbourOrientation" corresponds to "Default" in every step but extending it by a means for communication with another lift. Each lift is able to communicate with its left neighbour. The leftmost lift does not have a left neighbour and thus does not have a communication partner. This is caused by implementation issues. At every tick of the simulation, the lifts choose their actions from left to right, i. e. the leftmost lift decides first. If the leftmost lift would take the action of the rightmost lift into consideration, it would orient itself by actions of the last tick. Communication is performed as follows: when a lift following "LeftNeighbourOrientation" finds a landing call, it checks if its left neighbour is already serving this call. If the landing call is served by the neighbour, the call is ignored – otherwise it is served. The exact sequence of decisions forming the "LeftNeighbourOrientation" strategy is shown in figure 3.2.

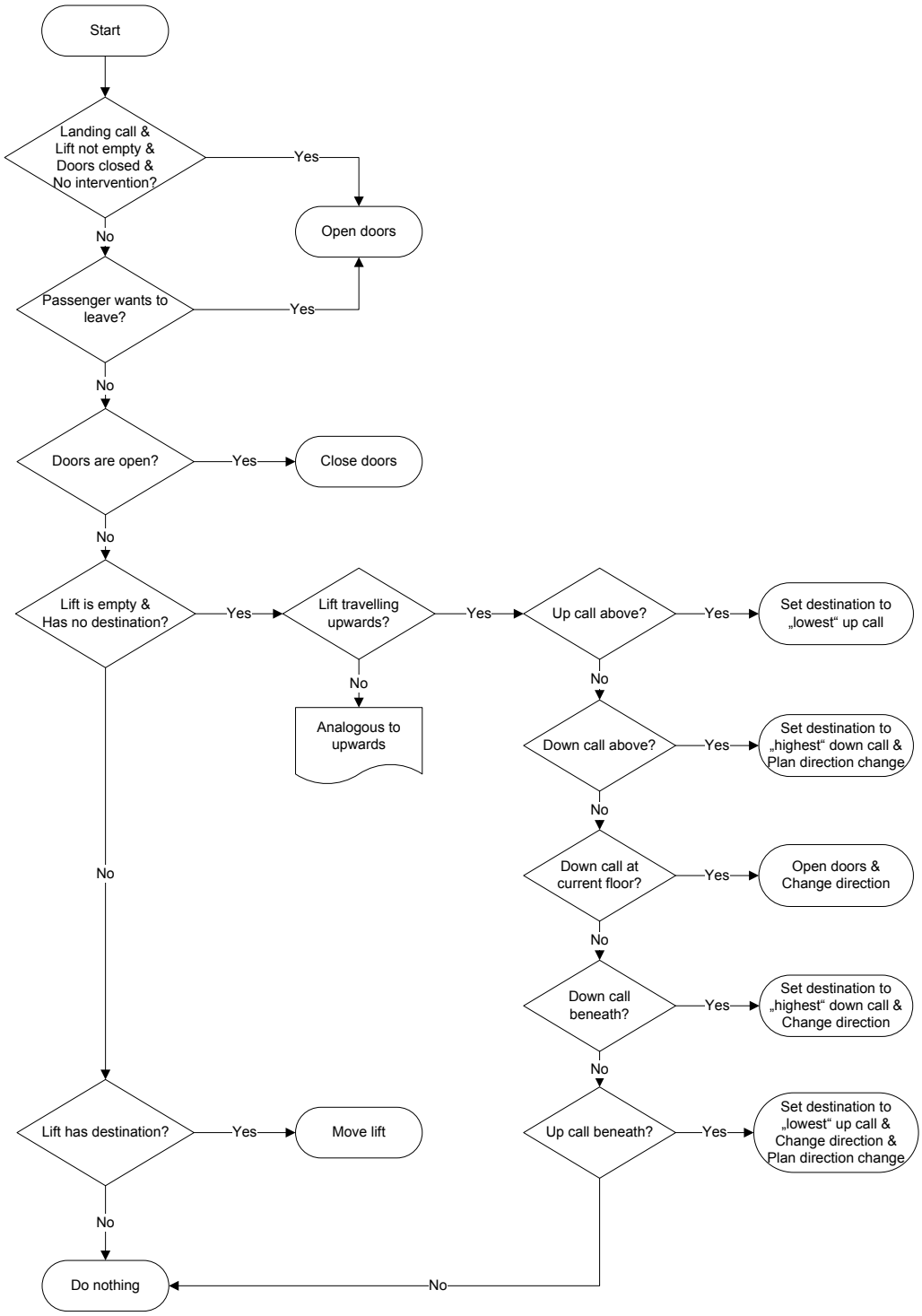



Figure 3.1: Collective control (“Default”)

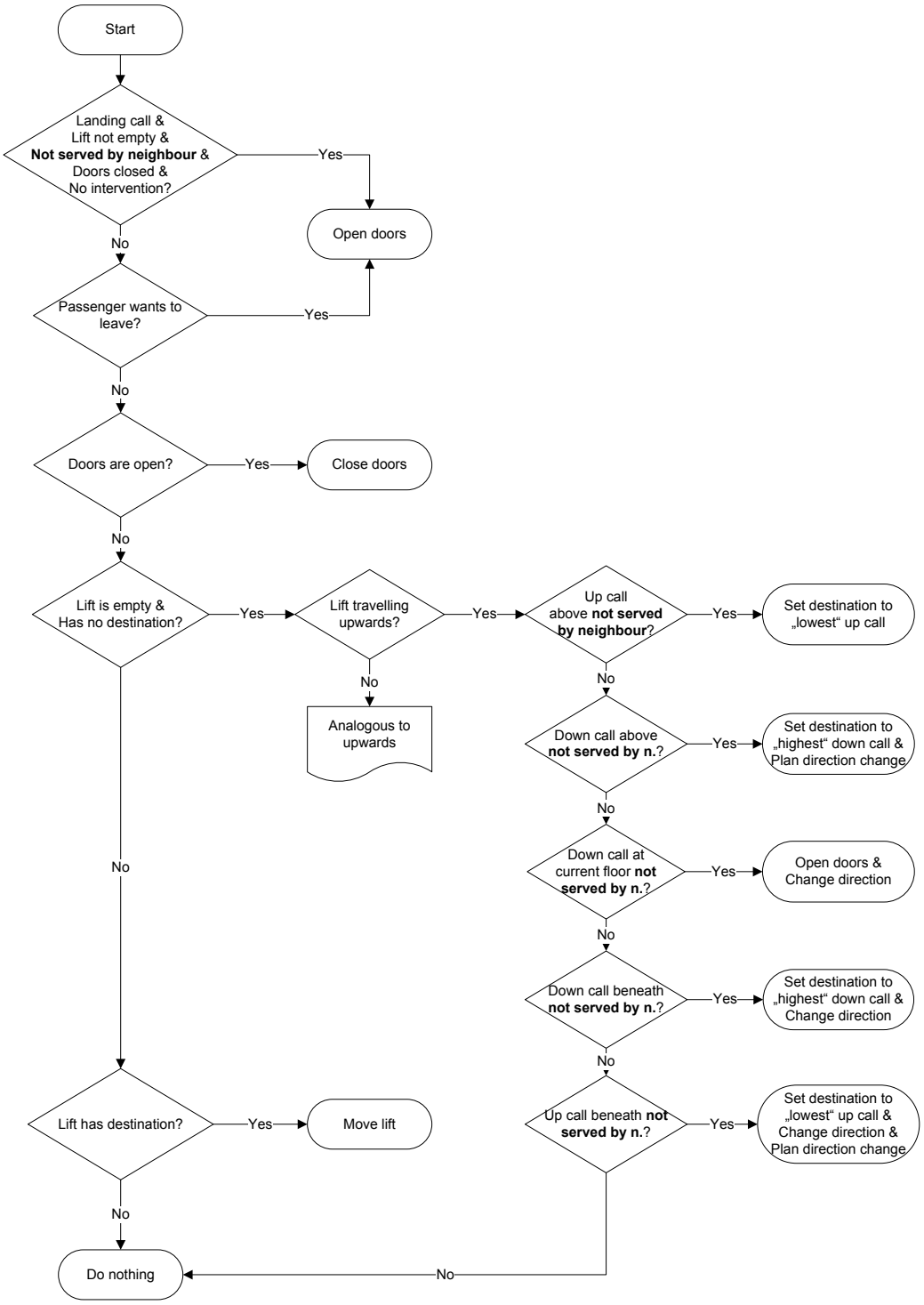



Figure 3.2: Collective control with information exchange (“LeftNeighbourOrientation”) – Differences to “Default” are marked with bold letters.

### 3.3.5 What Is the Cause of Bunching?

Bunching arises within the lift system for the most part independent from the current lift strategy. Considering the details given in section 2.6 and due to the relatively primitive strategies used, this is not a remarkable observation. However, within this simulation program, bunching is probably caused in a slightly different than in the classical way which was stated in section 2.6.3. In classical systems showing bunching phenomena, bunching is caused by different round trip times of the lifts based on the occurrence of random events (one lift has to serve more passengers than the other). Once the spacing of the lifts is "disturbed", bunching will arise due to its self-aggravating nature: as a longer timespan elapsed since the departure of the last lift, bunching occurs due to the higher number of passengers waiting for delayed lifts.

Random events do also occur in the simulation program[3], however, these events do not lead automatically to lift bunching, as this situation is *not* self-aggravating in the way described above and in section 2.6.3. In the simulation program, passengers always consume the same time to board/alight a lift independent of their number. Thus more passengers waiting (in the mean) for a delayed lift do not slow this lift further down at boarding. Additionally, the number of passengers in a delayed lift will probably be higher than in a early lift. However, due to the characteristics of the model, the total passenger alighting time will not be affected by the higher number.

So, why does bunching arise in the simulation programme anyway? The reason most likely causing bunching is the absence of communication among the lifts: if a landing call is registered, all lifts could travel to the corresponding floor (depending on the situation). This probably leads to a spatial concentration of the lifts and thus the arise of bunching. When the lifts are operating with "LeftNeighbourOrientation", this phenomenon is probably less severe because of the limited communication. There is another factor potentially enforcing bunching also mentioned in section 2.6.3: more passengers in a lift will most likely force the lift to stop more often than an early lift. This could slow down the delayed lift in comparison to the other lifts and foster the unequal spacing between the lifts. However, this theory does not apply if during heavy traffic conditions all lifts, even the early ones, are fully loaded as in this case the expected number of stops will be the same.

[3] This is caused by the Poisson arrival process.

The reason why passenger boarding times have not been introduced is that lifts also show bunching within the current implementation. Additionally, it is not very important which is the actual cause of bunching within this simulation model. Since the lift simulation serves only as a testbed for our research, the most important fact is that bunching *does* occur. The present bunching phenomenon is based on local interactions of the lifts and cannot be predicted from the system description beforehand. Thus it is considered to be also an emergent phenomenon and suited to verify the o/c architecture. Nevertheless, future work with the simulation programme should introduce passenger boarding times, as this could introduce new possible intervention methods.

## 3.4 General Structure

As mentioned in section 3.2, the simulation programme is an implementation of an observer/controller architecture following the generic model presented in section 2.3. The SuOC is formed by the lifts travelling in the building, the passengers waiting at the floors, and passengers travelling in the lifts. However, the passengers are neither controlled by the controller nor are they monitored by the observer. The passengers are placed "out of range" of the controller, because a real lift control system does also have no substantial abilities to give orders to certain passengers other than signalling the travelling direction of a lift and thus preventing passengers waiting for the other direction to board.[4] The observer does presently not monitor passenger movements at all. Observations like the detection of full lifts is already incorporated in the lift system and is thus utilized within the lift strategies. However, if certain control strategies of the controller would require some kind of passenger monitoring, it could easily be implemented.

The programme itself runs in form of a loop; during operation, the loop is cycled once every tick. After programme initialization the simulation begins iterating trough the loop: at first, (a) the passengers act, i. e. enter a lift, then (b) the lifts fulfil their orders like for example opening their doors; after this begins (c) the observing and (d) the controlling of the system. We omit a description of step (a), as there is not really anything to describe: the actions of the passengers are pretty straightforward and are limited to entering/alighting lifts. Step (b) has already been specified in section 3.3.4. The steps (c) and

[4] The up/down signalling stated in the example is already part of the lift system, i. e. it is incorporated in the behaviour of the passengers.

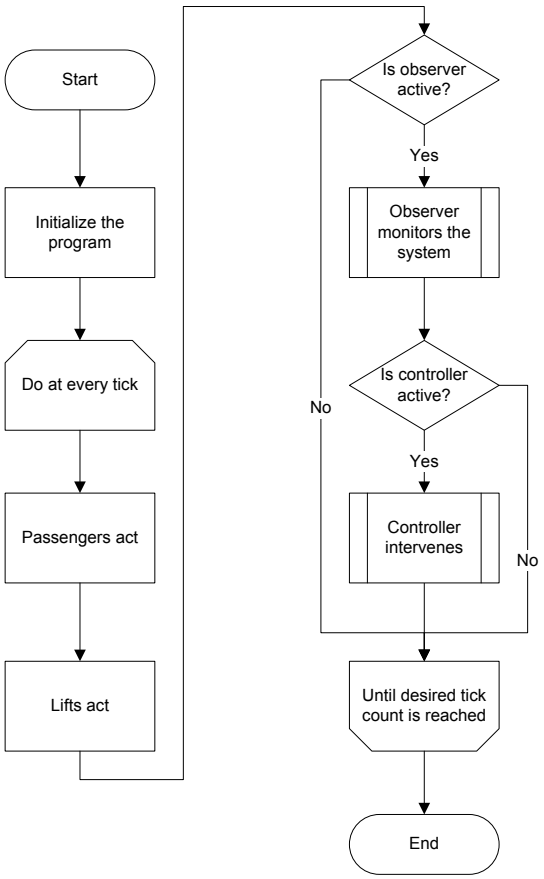



Figure 3.3: Operational sequence of the simulation program. The contained monitoring and controlling steps are shown in figure 3.4 and 3.13, respectively.

(d) are introduced in sections 3.4.1 and 3.4.2, respectively. Before the beginning or during the simulation the user is able to turn the observer and the controller on/off. Deactivating the observer does also deactivate the controller as the monitoring process of the observer is a necessary prerequisite for the controller. The function for deactivating only the controller is the possibility to observe the system by the methods of the observer without intervening in the system. As postulated in section 3.2, the simulation will run technically flawless[5] without the observer/controller. The just outlined course of operations is shown in figure 3.3.

[5]Most likely there will be more bunching without a controller. Nevertheless, the simulation is runnable without the controller.

The actual structure of the simulation programme basically corresponds to the structure of the original simulation programme. The main differences are the new modules *observer*, *controller*, and *simulator*, which are realized as Java classes. The functionalities of the observer are outlined in section 3.4.1. Details concerning the controller will be presented in section 3.4.2. The simulator is a constituent part of the observer forming the predictor component and will thus also be explained in the context of the observer within section 3.4.1. We will not go into the simulator module and leave it at the description of the predictor, though. The rough dependencies among the modules will be explained in the following sections.

### 3.4.1 Functionality of the Observer

The primary task of the observer is to monitor the system and generate the system parameters based on the current system state (see section 2.3.1). The relevant system state for the controller is mainly determined by the present degree of bunching as the controller has to alleviate it (see section 3.2). Therefore the observer has to have means to measure bunching and quantify it. The provisions, implemented in the programme for this reason, are described later on. Additionally, the observer passes several other parameters of the system to the controller, which are needed for the interventions of the controller. Furthermore, the observer is able to predict the future system development and uses this prediction to calculate an estimated bunching value for the near future. This estimation can be used as a basis for controller interventions. It is described at the end of this section.

In the following, the procedures taking place within the observer as well as the actual parameters being monitored by the observer will be outlined.

#### How Does the Observer Work?

The observer monitors the system in real-time, i. e. the status of all relevant system components is checked every step within the monitoring process. For this purpose, the *monitoring* process of the observer is made up of four steps: *preprocessing*, *data analysing*, *predicting*, and *aggregating*. The monitoring process is illustrated in figure 3.4.

During *preprocessing*, basic computations are performed like calculating the distances between the lifts and transforming the actual lift positions in the building to a measure

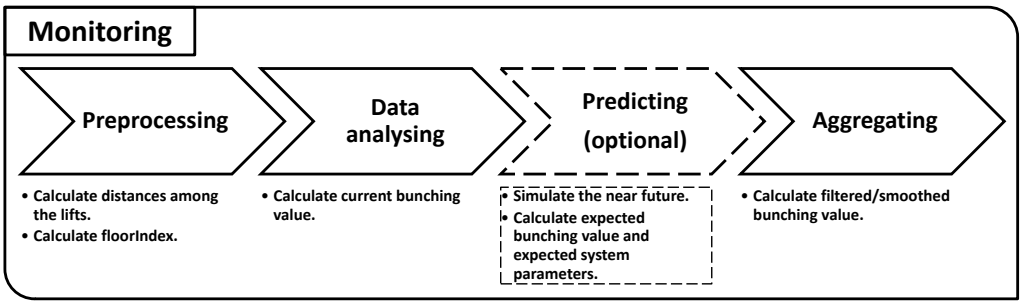



Figure 3.4: Steps of the monitoring process

called "floorIndex" (described later on). These computations are based on raw input data like the landing calls present in the building or the current positions of the lifts. Subsequently, the current bunching value is calculated within the *data analysing* step. The bunching measurement as well as the transformation of the lift positions will be described in the next section. Data analysing is followed by the *predicting* step where the development of the system in the near future is estimated via a simulation model. Within this step expected future system parameters as well as an estimation of the future bunching value are obtained. Predicting is an optional step and is omitted if the predictor functionality is turned off. The predictor functionality is illustrated at the end of this section. Finally, in the *aggregating* step the obtained calculation results are filtered and smoothed to reduce the effects of noise.

Before the observer is able to preprocess raw data it has to retrieve this data first. Data retrieval raises the question what parameters can actually be "seen" by the observer, i. e. how far the "visibility range" of the observer reaches. The visibility of parameters is certainly not a serious constraint in the present straightforward scenario, but it could be a rather important factor at more comprehensive systems. In the following an overview of possibly observable parameters as well as effectively observed parameters will be given. Afterwards, it will also be shown how the observer calculates a bunching measure based on this observations.

### Observation Range

In order to develop metrics to evaluate bunching, first of all the parameters that can actually be observed and that are relevant for attaining the objective have to be specified.

For this purpose, we separated the parameters into three "domains": *lifts*, *passengers*, and *general model parameters*. These parameters are listed in table 3.1.

| **Lifts** | **Passengers** | **Model** |
|---|---|---|
| Position ★ | Positions/directions of landing calls ★ | Number of lifts ★ |
| Travelling direction ★ | Length of waiting queues | Number of floors ★ |
| Passengers in the lifts | Destinations of car calls ★ | Passenger arrival rate ▼ |
| Lift capacity | Destinations of landing calls ◊ | ... |
| | Individual waiting times ◊ | |

Table 3.1: Observable parameters. Parameters marked with ★ are observed by the observer. Parameters marked with ◊ are only observable at systems utilizing destination control. Parameters marked with ▼ are solely used within the predictor component of the observer.

The table shows potentially observable parameters as well as parameters which are actually observed by the observer. As mentioned above, these parameters are classified in three domains for the sake of clarity. The listed parameters should be rather self-explanatory, however, some minor annotations may be helpful. The *number of passengers in the lifts* and derived filling level of the lifts is not monitored by the observer, however, it is already taken into account in the lift strategies. The positions and directions of the landing calls are recognized by the observer, but the actual number of passengers waiting at a floor and forming the *waiting queues* is not known by the observer.

*Individual waiting times* of the passengers and *destinations of the landing calls* could easily be observed in the programme, however, lift systems in practice may accomplish this observation only when destination control systems as introduced in section 2.6.5 are utilized. As the lift strategies in the present simulation programme do not comprise destination control, these parameters are also not considered by the observer.

The visibility of model parameters to the observer is ambiguous. Within this theoretical model these parameters could be made visible to the observer easily, however, in real-world systems of larger scale such a request would be infeasible in many cases. In this case, the observer knows the number of floors in the building and the number of lifts. Knowledge of the effective *arrival rate* underlying the simulation model is certainly questionable as knowledge of this rate is completely out of range of a real system. Within the current implementation of the observer the predictor component knows this rate to generate a

precise estimation of the future. This is done to survey if the use of prediction could lead to an advantage. In later implementations of the observer the knowledge of the arrival rate should definitely be reconsidered. An estimation of the arrival rate could be derived from the observation of other parameters[6], but this is currently not performed within the observer.

The parameters differ in their variability: some parameters like the number of floors or the capacity of the lifts will definitely remain constant during operation; other parameters as the number of lifts and the passenger arrival rate could vary over time when a lift failure occurs or the traffic pattern changes, respectively.[7] These constant parameters are not "observed" but rather "known" by the observer, if they are visible and their observation is desired.

#### Developing Metrics to Measure Bunching

During operation, the observer is ought to measure the current level of bunching among the lifts. This intend raises the question of how to actually evaluate the "amount" of bunching. Measurement theory deals with this subject and there have been developed several concepts addressing this question. However, we will not provide an introduction to this field of science here but rather outline the general thoughts which lead to the actual measure. For more information about measurement theory have a look at [KLST71, SKLT89, Luc90], which give an extensive overview of this field. A more concise outline is given by [SAS97]. As for measurement theory, we leave it at the following saying stated by lord Kelvin which gives an idea of the essence of measurement:

> *"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of science."* (William Thomson, "Lord Kelvin")

Thus, roughly speaking our aim is to express bunching "in a number". We will use this "number" to compare different system states and the effectiveness of different control strategies and parameter variations within the controller. Therefore, forming this measure

[6] Like the temporal distances among landing call registrations etcetera.

[7] Both lift failure as well as changing traffic patterns or changing passenger arrival rate during operation are not considered within this thesis.

consists of two steps: at first we have to define the requirements on the bunching measure given by the intended use; afterwards, we must develop a metrics which fulfils these requirements. Our requirements on the bunching measure are largely the same as Al-Sharif took as a basis on developing his bunching coefficients in [AS93]:

**Normalization** The bunching measure has to be normalized to be independent of the parameters of the observed system. Therefore the measure can be used to evaluate changes in the system, e. g. varying the number of lifts, with regard to their effect on the bunching behaviour.

**Limited range** The measure must have a defined range of values to be perspicuous and self-explanatory. In this case we set the range from 0.0 to 1.0, where 0.0 represents no bunching and 1.0 full bunching. No bunching would correspond to a perfect distribution of the lifts throughout the building and full bunching to the situation where all lifts arrive at a floor at the same time and behave like a huge lift.

**On-line measurement** The metrics have to give an instantaneous view of the current bunching situation. This is essential, as the controller interventions are also on-line and rely on the information about the current system state given by the observer.

**Monotony** The bunching measure takes values in such a form that the order of the numbers reflects an order relation defined on the attribute, i. e. a bunching value of 0.4 represents less bunching than a value of 0.8. However, the statement "a value of 0.4 represents half the bunching than a value of 0.8" would be meaningless at an order relation. As we are not interested in statements like this the order relation is sufficient for our needs.

After the definition of the requirements we are now able to construct a bunching measure. The general idea is to define an ideal lift system behaviour in terms of ideal parameter values. The observer compares these ideal values to the actual parameter values obtained by observation. At doing this, the deviation of the current state from the ideal values is calculated. This deviation is expressed as a number and indicates the current amount of bunching present in the lift system.

At first, one has to decide which parameters to utilize in order to evaluate bunching. For this purpose it is necessary to remind the *characteristic traits of bunching phenomena* in order to identify the parameters that constitute these traits: at bunching, lifts synchronize and become unequally distributed throughout the building, i. e. the distances among

them alters. This raises another problem: what is the actual *distance* between two lifts (w. l. o. g.)? There are two possible ways to express the distance: description in either a spatial or in a temporal way. Spatial distance refers to the number of floors lying between the lifts. It can very easily be evaluated by counting the number of floors between the two lifts. The temporal distance between the lifts refers to the time one lift would need to travel to the other lift. It depends on the spatial distance between the lifts and on the number of stops the lifts have to perform. The number of stops depends solely on the car calls and landing calls present in the system, as the time required for a stop is constant in this model. Therefore, evaluating temporal distances between lifts results in calculating estimated travel times of the lifts. The observer currently only considers spatial distances between the lifts, however, calculating temporal distances should not be a problem. Besides, there would probably not be a large difference between spatial and temporal distances as the time required for a stop is constant. Hence, the term "distance" will refer to spatial distance in the following (unless noted otherwise). Parameters which express spatial distribution of lifts are the positions of the lifts and their travelling directions. Thus, these parameters have to be used to form the measure. The measure will be derived in the following. The approach on measuring bunching developed hereafter follows essentially the United States Patent number 5447212 (see [Pow94]).

| Term | Denotation |
|---|---|
| $N$ | Number of floors |
| $L$ | Number of lifts |
| $l_i$ | One lift of the group of $L$ lifts |
| $f_i$ | The floorIndex of a lift |
| $d_i$ | Distance from lift $i$ to the next lift $i+1$ |
| $d^*$ | Distance at equal distribution of the lifts |
| D | Sum of the distance deviations |
| BV | Bunching value |

Table 3.2: Terms used for the calculation of the bunching value

In order to calculate the distances among the lifts a distance function has to be defined. All terms used in the following are listed in table 3.2 along with their denotations. As stated before, the distance between two lifts should be the number of floors lying between them. However, it has to be defined how to calculate distances between lifts travelling in

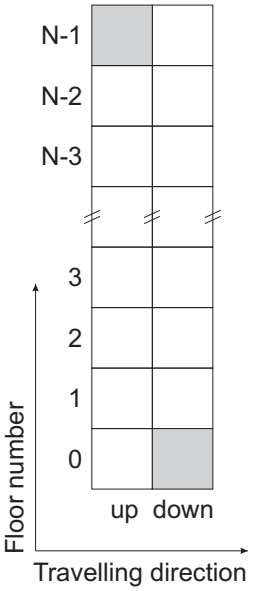



Figure 3.5: Displaying lift positions and travelling directions in form of a grid

| | | | | |
|---|---|---|---|---|
| 0 *up* | 1 *up* | ... | $N-2$ *up* | $N-1$ *up* |

Figure 3.6: Lowermost floor to topmost floor pictured as a line segment

different directions. Mathematically speaking the lifts move in a two-dimensional space: it can be imagined as a grid with the direction (up/down) being the $x$-coordinate and the floor being the $y$-coordinate ranging $[0; N-1]$[8] with $N$ being the number of floors (see figure 3.5). As measuring distance between lifts within this space is nowhere near self-evident, our idea is to transform this two-dimensional grid into an one-dimensional space where the distance can easily be defined. For this purpose, we separate the lifts in two groups, which each group heading into a different direction. Herewith we get two one-dimensional segments with equal length (the number of floors, $N$). The first segment ranges from the lowermost to the topmost floor (viewed upwards) pictured in figure 3.6. And the second segment from the topmost to the lowermost floor (viewed downwards) pictured in figure 3.7.

[8]The terminal floor is commonly assigned zero as this represents the ground level. Thus unity is assigned to the first floor above ground. Since $N$ being the number of floors *including* the terminal floor, the topmost floor is assigned number $N-1$.

| | | | | |
|---|---|---|---|---|
| $N-1$ *down* | $N-2$ *down* | ... | 1 *down* | 0 *down* |

Figure 3.7: Topmost floor to lowermost floor pictured as a line segment

Each of these two segments can be shortened by one floor: there will never be an upwards travelling lift at the topmost floor and a downwards travelling lift at the lowermost floor. As lifts in a building change their travelling direction under heavy traffic conditions rarely (see section 3.3.4), it is very likely lifts travel to the topmost or lowermost floor, before changing their direction. For example a lift at floor $N-2$ will likely travel to floor $N-1$, change its direction, move to $N-2$, and will subsequently travel further down. In this regard a lift being at floor $N-2$ *up* is two floors away from a lift being at floor $N-2$ *down*. Therefore, we join the two segments of length $N-1$ at their "upper ends" to form a combined, one-dimensional "bar" of length $2(N-1)$. This bar forms the lower row in figure 3.8.

| $f_i$ | 1 | 2 | ... | $N-1$ | $N$ | $N+1$ | ... | $2N-1$ | $2N-2$ |
|---|---|---|---|---|---|---|---|---|---|
| **Floor** | 0 *up* | 1 *up* | ... | $N-2$ *up* | $N-1$ *down* | $N-2$ *down* | ... | 2 *down* | 1 *down* |

Figure 3.8: Calculation of the floorIndex

On the bar every lift $l_i$ can be assigned an index $f_i$ with $i \in [1; L]$ and $L$ being the number of lifts in the group. For the sake of brevity a mathematical formulation of the assignment process is omitted in this place. The maximum value of $f_i$ results as follows:

$$f_{max} = 2(N-1) = 2N-2 \tag{3.1}$$

This index refers non-ambiguously to a lift's current position as well as its travelling direction. We call this index $f$ *"floorIndex"*. For example, a lift being at the first floor in upwards direction is assigned floorIndex 2. How floorIndexes are related to the actual floor is shown in the upper row in figure 3.9. Lifts travel on this bar towards increasing indices until reaching the end and leaping to the beginning of the bar. This forms an unwanted discontinuity.

We resolve this discontinuity in the same way we built the bar in figure 3.8: we join the two ends of the bar (just as we did it before) to obtain continuous lift movements around the terminal floor. The resulting circle can be seen in figure 3.9. Within this representation of the building, the lifts seem to never change their travelling direction: they always travel

clockwise around the circle. If a lift changes its direction in the building, it will perform a leap within the circle representation.

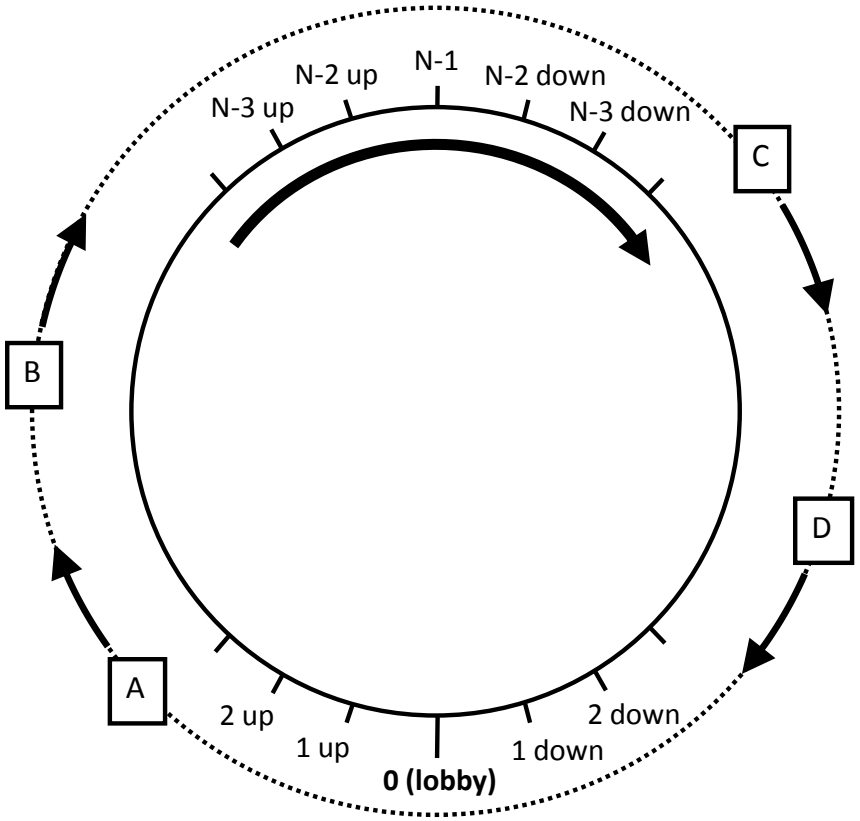



Figure 3.9: Representation of travelling lifts as a circle (circle representation)

Within the circle representation the *distance* between two lifts can now be calculated very easy by determining the minimum number of floors lying between them. In the present case, the observer calculates for each lift $l_i$ the distance $d_i$ to the next lift $l_{i+1}$ in the circle representation. For this purpose the lifts are ordered by ascending floorIndexes, i. e. $f_i < f_{i+1}$.

$$d_i = \begin{cases} f_{i+1} - f_i & \text{if } i < L \\ f_0 + (f_{max} - f_L) & \text{if } i = L \end{cases} \tag{3.2}$$

As stated at the beginning, we measure bunching by comparing the current situation to the ideal situation with respect to the requirements listed further up. Ideally, the lifts are perfectly distributed throughout the building, i. e. they are equally distributed in the

circle representation. Thus in an ideal case consecutive lifts are separated by a constant distance

$$d^* = \frac{f_{max}}{L} = \frac{2N-2}{L} \tag{3.3}$$

with $L$ being the number of lifts present in the system and $f_{max}$ as derived in (3.1).

If $L$ lifts are present in the building, there are also $L$ distances among these lifts. In order to compare the current situation with ideal conditions, the difference of each distance $d_i$ from the ideal distance $d^*$ is summarized. The absolute value of the difference is taken to assign equal costs to positive and negative deviations from the ideal:

$$D = \sum_{i=1}^{L} |d_i - d^*| \tag{3.4}$$

At last, the deviation sum $D$ has to be *normalized* in order to fulfil the requirements stated at the beginning. This is done by division by the worst possible value $D$ could take which occurs when all lifts are at the same time at one floor (which represents full bunching). Hereby, $L-1$ distances would be equal to zero and exactly one distance would be equal to the full circle, which is $f_{max}$. So, we get for the worst case:

$$D_{worst} = (L-1) \cdot |0 - d^*| + 1 \cdot |f_{max} - d^*| = (L-1) \cdot d^* + |(2N-2) - d^*| \tag{3.5}$$

Thus, from (3.4) and (3.5) we receive the current bunching value:

$$BV = \frac{D}{D_{worst}} = \frac{\sum_{i=1}^{L} |d_i - d^*|}{(L-1) \cdot d^* + |(2N-2) - d^*|} \tag{3.6}$$

At last, the BV is filtered to compensate sudden outlier among the reported BVs in the aggregating step. For this purpose an average of the last reported values is calculated and provided to the controller; this process is called smoothing. In the present program, the *exponential moving average* is calculated. This smoothing method is very similar to the standard moving average technique utilized in numerous applications, except that it puts greater emphasize on more recent values. At moving average the mean value over the last ticks is calculated. This "window" is moved one step further every tick. More information in this regard can be found in **(insert reference to smoothing methods - insertref)**.

#### The Predictor Functionality

If the predictor functionality is turned on, an estimation of future system behaviour is evaluated in addition to the calculation of current bunching. In order to perform the

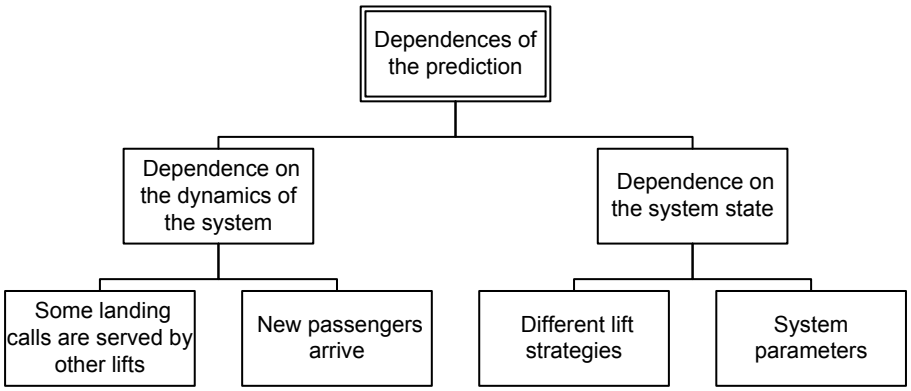



Figure 3.10: Dependences of the prediction

estimation, the observer contains a complete *model of the system*. Based on this model the observer performs a simulation of future system behaviour. The outcome of the simulation is the knowledge of a future system state. Therefore, especially the knowledge of the future BV as well as the future lift distribution is desired. How the simulation is performed and the reasons for realizing the prediction this way will be outlined within the next paragraphs.

At first, it has to be considered in which way the observer is able to calculate a prediction abstracted from the actual realization of the predictor. A prediction has to be based on the current system state comprising all system parameters and lift/passenger positions as well as the lift strategies. However, the future course of the system does not only depend on the current system state but also on prospective interactions of the system constituents. These interactions are certainly a result of the current system state, but they are very difficult to predict because of the self-organizing nature of the system. Thus the development of the system is a highly dynamic operation making the prediction by no means trivial. A brief outline of the stated dependencies is given in figure 3.10.

In particular the dynamics of the system cause notable changes of the system state and affect the choice of the predictor realization. An intuitive attempt would be to simply evaluate the current situation (i. e. the current system state) in a *static* way in order to generate a bunching estimation. In this regard, "static way" refers to a plain computation based on the current system state and assumptions of future events. A very simple computation scheme for example would only use the current lift distribution, system parameters, and landing calls to approximate the future by an evaluation function. Additionally, the

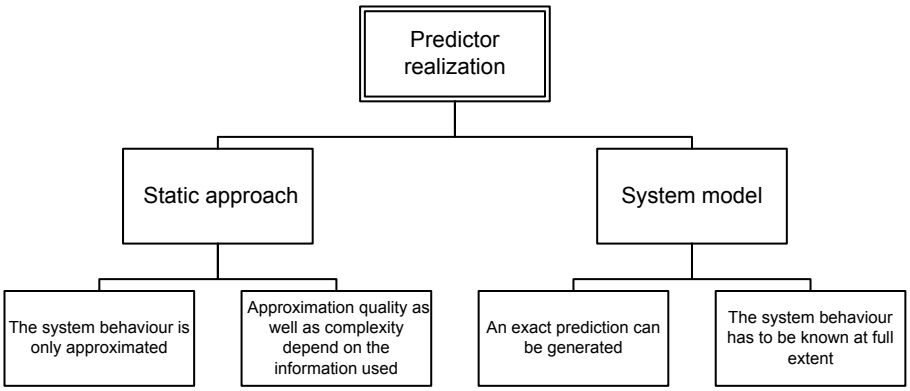



Figure 3.11: Considered ways of realizing the prediction

arrival and disappearing of landing calls could be guessed with a probability function to improve the quality of the estimation. However, such a static calculation scheme would always make uncertain assumptions and could not estimate the future correctly.

A completely different approach is to embed a *system model* in the observer. While doing so, the observer has total knowledge of the procedures within the system. This comprises the decision rules of every system constituent and thus their expected actions and reactions. Therefore an exact prediction of the future system behaviour can be made. However, utilizing such a system model is not without drawbacks: it presumes the complete knowledge of the system and will most likely be much more extensive to evaluate than a plain, static calculation scheme. A brief outline of these two approaches is given in figure 3.11.

There are certainly more methods of realizing a predictor functionality than presented. However, a simulation of the system behaviour based on a system model is doubtless the most powerful and promising approach (in terms of prediction quality). In particular the absent consideration of lift interactions at a static model would most likely render a prediction based on such a scheme useless. To enhance such a model to an extent that it could approximate the interaction sufficiently seems not reasonable as it could not be adapted to changes easily within the system. And compared with a static model, a system model can potentially be reused when developing a learning method within the controller. For these considerations we decided to use a model based prediction for the observer.

The actual prediction takes place as follows. The observer evaluates at every tick $t_u$ based on the present system state $s_u$ the predicted, future system states $s_{u,v}$. The index $u$

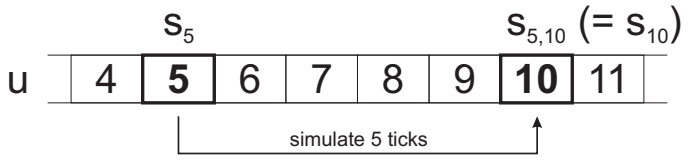



Figure 3.12: An example for prediction: at $t_5$ the future system state $s_{5,10}$ is simulated.

refers to the current tick count of the system and $v$ to the tick count of the predicted tick starting from tick $u$ (i. e. based on $s_u$). For $v$ holds $v \in [u; u + h]$; $h$ is the predictor horizon. Figure 3.12 gives an example: at current tick count $u = 5$ the system state $s_{5,10}$ of tick 10 is simulated. As stated before, in the present case the simulation performed on the system model corresponds to the real system behaviour, i. e. the simulated state $s_{u,k}$ is exactly the same as the real, future system state $s_{u+k}$. However, hereby the controller is *no* part of the system model and is thus not taken into account within the simulation. Therefore in the given example $s_{5,10} = s_{10}$ would only hold if the controller does not intervene in the system from tick 5 to 9 and thus altering its course. The controller is not simulated in the "prediction environment", as this would require further changes within the implementation of the controller. It is up to further works to implement these changes and perform tests to investigate the potential improvement over the current situation. The performance of the current implementation will be examined in chapter 5.

After the simulation the future system state $s_{u,k}$ is known to the observer. However, the observer does not use the full information of the state description; instead only the future current and smoothed BVs as well as the future floorIndexes and distances among the lifts at tick $t_k$ are handed over to the controller. This is done as the controller does presently not need more information. Of course all this can be altered, if desired.

At last it has to be mentioned, that the current concept of the prediction functionality is not without disaccord: in the current implementation, the system model knows the real passenger arrival rate present in the system. This is not practical for real systems as the arrival rate is a true probabilistic process which can only be approximated. In order to make the prediction as accurate as possible this ability of the observer is accepted for this implementation. However, for future research it has to be considered to change the predictor if necessary.

### 3.4.2 Functionality of the Controller

After the observer has finished evaluating the current system state including the BV, the controller becomes active. As stated before, the task of the controller is to maintain a proper system behaviour. The controller assesses the system situation based on the information given by the observer. Afterwards the controller intervenes in the course of the system, if considered to be necessary. For this purpose two control strategies have been implemented and will be described later on. Before doing so an overview of the controlling process will be given in the following section.

#### How Does the Controller Work?

Like the observer the controller acts at every tick. The controlling process starts right after the observer has finished its work. It consists of a *receiving* and an *action selecting* step. The process is depicted in figure 3.13.

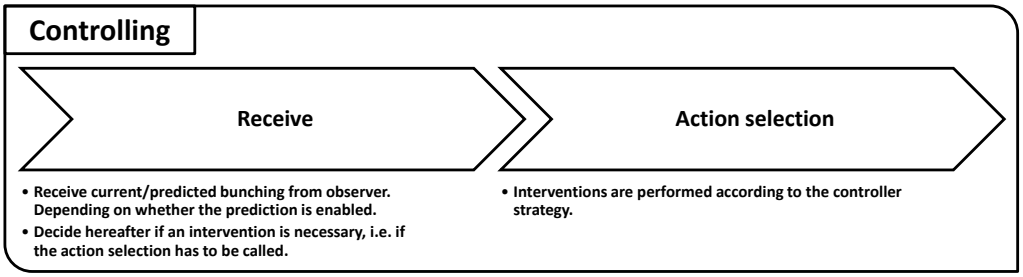



Figure 3.13: Steps of the controlling process

At first the controller has to assess the situation. This is done within the *receiving* step. It decides based on the situation parameters received from the observer if the situation is in accord with the desired course of the system. In the present case the decision is based on the BV reported by the observer, as the task of the controller is to prevent bunching as stated in section 3.2. The situation is considered undesirable if this value surpasses a certain predefined threshold. Thus, if the value exceeds this threshold, the controller is going to intervene. If otherwise, the controlling process comes to an end without further action. The controller checks the BV every tick, so it is decided at every tick anew, if interventions have to be started, ended, continued, or to be omitted further on. During the receiving step either current situation parameters or predicted parameters are used,

depending on whether the predictor functionality is turned on or off. The information used by the controller within the two implemented strategies is shown in table 3.3.

| **Strategy:** | **softIntervention** | **strongIntervention** |
|---|---|---|
| **Decision is based on:** | current BV ★ | current BV ★ |
| **Used information:** | lift distances ★<br>lift positions ★<br>landing calls | lift distances ★ |

Table 3.3: Information used within the control strategies. Predicted values of items marked with a ★ are used, if prediction is enabled.

Finally the *action selector* step follows, if the controller decides to intervene. Within this step the controller performs its interventions. The actual form of the interventions depends on the used control strategy. Within the current programme two control strategies have been implemented. These strategies will be outlined in the following paragraphs.

#### Controller Strategies

Before developing controller strategies we should remind that the set of possible interventions is determined by the constraints stated in section 3.2 and the general design thoughts of the generic observer/controller architecture as illustrated in section 2.3.2. The bottom line of these thoughts is to keep the self-organizing nature of the system and realize the controller as a guiding instance, which is correcting the course of the system by slight interventions. Thus, we do not intend to implement orders like *"Lift A, go to floor 5"* within the control strategies. Instead, we focus on interventions which influence the lift behaviour indirectly. An example for such an indirect intervention would be the hiding of the landing call at floor 4 from lift $A$ and thus making lift $A$ to travel directly to floor 5. Therefore, most means outlined in section 2.6.5 are no suitable control strategy as they do not comply with the requirements stated in 3.2. We will not go into detail for this reasons.

Taking into account all considerations mentioned, what are the possible controller interventions? In order to answer this question we should recall the aim of the controller: to maintain an equal spacing of the lifts. Therefore, when the spacing departs from the

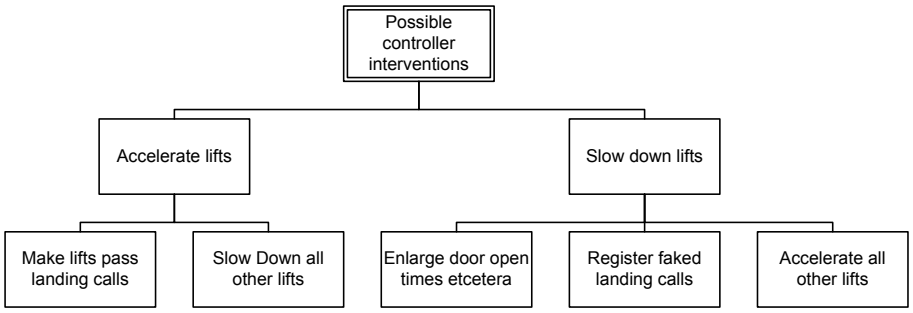



Figure 3.14: Possible controller interventions with regard to exerting influence on lifts

ideal, the controller has to "do something" to restore the initial situation. Within the current scenario we assume that only lifts (and not passengers) can be influenced by the controller. With respect to the constraints and considerations stated before, the controller has roughly two possible (abstract) ways to intervene: it could *accelerate lifts* and/or *slow down lifts*. Most likely it is reasonable to accelerate *delayed* lifts and slow down *early* lifts. A lift is considered as delayed if the distance to the next lift in travelling direction is significantly higher than the distance at perfect lift distribution (early lifts accordingly). Possible means in this regard, with respect to general lift system practice as well as the opportunities given by the programme, are shown in figure 3.14. However, no matter what provisions out of these two fields are taken by the controller, there will most likely be a trade-off between the minimization of the future bunching factor and the response time to landing calls (or other dimensions) at the present time.

With respect to these considerations, two strategies have been implemented within this thesis: *"strongIntervention"* and *"softIntervention"*. These two strategies represent different control approaches and should be regarded as exemplary implementations of these concepts. The basic idea behind both of these strategies is to accelerate delayed lifts. The reason of focusing on lift acceleration is the expected lower impact on the handling capacity of the system than strategies trying to slow down the lifts. This is just an estimation and requires investigations in further research, though. Probably strategies combining both approaches will deliver the best results, which has to be examined in further research alike. In the following the two actually implemented strategies are described. A rough summary of the differences among the strategies is given in table 3.5.

**strongIntervention** represents a rather simple strategy causing potentially relatively significant impact on the system due to the severe nature of its actions. As stated before, the underlying concept of this strategy is to accelerate delayed lifts, i. e. lifts having a large distance to the next lift in travelling direction. The actual idea for "strongIntervention" is to accelerate lifts by making them *blind* with regard to landing calls. A blind lift will not serve any landing calls in the building and will thus stop less frequent. The result is a relative acceleration of a blind lift as all other lifts "loose time" by stopping at landing calls. The duration of this lift blindness is discussed later on. Furthermore, "strongIntervention" exhibits additional functionalities: instead of blinding only one lift, several lifts can be blinded simultaneously or partially. The actual sequence of operations is shown in figure 3.15 and is explained in the following.

At first the controller sorts all lifts $l_i$ according to their distance $d_i$ to the next lift in travelling direction. After sorting, we represent the lifts in the form of $o_j = l_i$ with $j$ being the order of the lift: $o_1$ refers to the lift with the largest distance $d_i$, $o_2$ to the lift with the second largest distance and so on. Depending if the predictor is turned on or off, the controller uses either current lift distances or predicted distances. Now the controller blinds the half of the lifts having the most distance ahead, i. e. the lifts $o_1 \dots o_{\lceil L/2 \rceil}$ (with $L$ being the number of lifts), with decreasing intensity. This strategy affects more lifts at a time to be independent from the number of lifts in the building.[9] The amount of blindness $b(o_j)$, which is assigned to the lifts, is defined as follows:

$$b(o_j) = \begin{cases} 1 - (\frac{1}{\lceil L/2 \rceil} \cdot (j-1)) & \text{if } j \leq \lceil L/2 \rceil \\ 0 & \text{else} \end{cases} \tag{3.7}$$

The blindness $b(o_j)$ of a lift determines the probability this lift will serve a landing call: for example a lift being 50 percent blind will pass a landing call with a probability of 50 percent. This relationship will probably be more obvious if explained with an example. For this we assume a building is containing $L = 5$ lifts $l_1 \dots l_L$. These lifts are now sorted by their distance $d_i$. Thus we receive the ordered lifts $o_1 \dots o_L$. These lifts are now blinded with the function $b(o_j)$ defined at (3.7). The result of the blinding is shown in table 3.4.

The blindness of a lift lasts until (a) the BV falls below the bunching threshold and the controller thus stops its interventions, (b) the distances $d_i$ change in a way that other lifts have now the largest distance ahead and therewith follows a change in the order of the lift

[9] For example blinding one lift in a two lift system would result in a certain impact. At an eight lift system, blinding one lift would have probably much less effect.

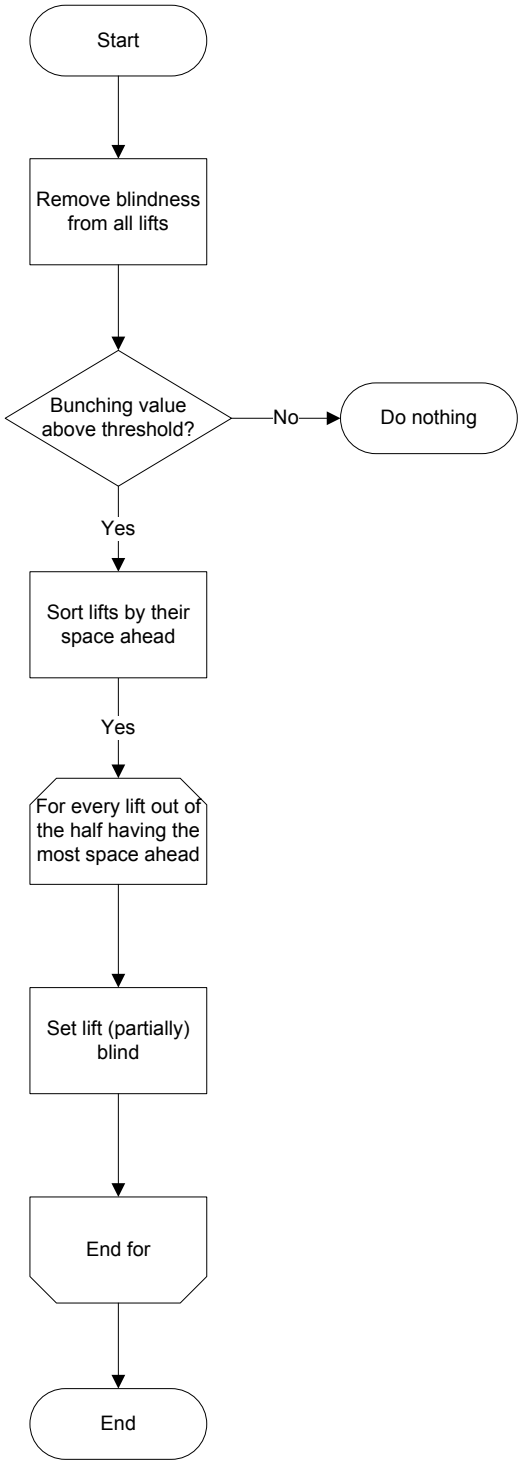



Figure 3.15: Controller strategy: “strongIntervention”

| **Lift** $o_j$ | $o_1$ | $o_2$ | $o_3$ | $o_4$ | $o_5$ |
|---|---|---|---|---|---|
| **Blindness** $b(o_j)$ | 100% | 66% | 33% | 0% | 0% |

Table 3.4: An example of lift blinding with five lifts

from $o_k$ with $k \in [1, \lceil L/2 \rceil]$ to $o_l$ with $l \in [\lceil L/2 \rceil + 1, L]$, (c) the lift changes its travelling direction, or (d) if the controller is turned off by the user.

At comparing "strongIntervention" with the generic interventions introduced in section 2.3.2, it cannot be clearly associated to one of those generic interventions. On the one hand this process could be regarded as hiding landing calls from a lift and therefore as a manipulation of the perception of a lifts environment. On the other hand, this represents a change of a lifts decision rules, since a blind lift does not distinguish between the separate landing calls, i. e. its behaviour can be described in form of a changed strategy. Therefore, it seems plausible to consider "strongIntervention" as manipulating (a parameter of) the decision rules of the lifts. With "strongIntervention" following this generic scheme, it differs from "softIntervention", which will be described in the following paragraph.

There are a few possible *drawbacks* if the controller acts based on "strongIntervention". This strategy incorporates no means against so-called *starvation* of some passengers. Starvation refers to a problem that occurs when the lifts repeatedly pass a landing call for some reasons. The passengers waiting for this call to be answered are pictorially suffering from starvation. This is an undesired characteristic which most likely occurs at approaches which rely on the acceleration of lifts and thus on lifts passing landing calls. This drawback can be formulated more generally: there is no distinction between the landing calls – they are all treated the same way. Another point regards the fact, that always half of the lifts are blinded (if the controller is active). Hence, it is possible that more lifts than necessary are set blind leading to an increase of the AWT of the passengers.

**softIntervention** represents a more sophisticated strategy performing relatively precise interventions compared to "strongIntervention". As with "strongIntervention", the basic idea of "softIntervention" is to accelerate delayed lifts to get back the equal spacing for reasons described before. The acceleration is hereby performed by *hiding landing calls* from delayed lifts. Within this strategy means against the so-called *starvation* of some passengers are also incorporated. Starvation refers to a problem that occurs when the

lifts repeatedly pass a landing call for some reasons. Passengers waiting for this call to be answered are pictorially suffering from starvation. This is an undesired characteristic which most likely occurs at approaches like “softIntervention” which rely on the acceleration of lifts and thus on lifts passing landing calls. The way this is realized will be described in the following. Figure 3.16 shows the actual sequence of operations forming “softIntervention”.

Following “softIntervention”, first of all the controller builds up the set of lifts $L$ which have a distance $d_i$ larger than the ideal distance $d^*$ multiplied by a variable $c$ :

$$L = \{l_i | d_i > d^* \cdot c\} \tag{3.8}$$

We introduced the variable $c$ to make an adjustment of the strategy possible. In the following chapters this variable is referred to as *multiplier*. By varying the parameter the impact of the strategy can be defined. Setting the multiplier for example to a higher level results in setting less landing calls “tabu”.

After building up $L$, for every lift $l_i \in L$ the next landing call in travelling direction is searched and set *tabu* for this lift, if no *tabuTabu* exists (the meaning of “tabuTabu” will be explained later on). The “tabu” setting of landing calls represents the above mentioned hiding of landing calls from the lifts. A lift will not serve a landing call being “tabu” for this lift and will thus pass it. A “tabu” is always referred to both a landing call and a lift, i. e. a landing call being “tabu” for lift $l_1$ is not necessary “tabu” for lift $l_2$ (only if another “tabu” for $l_2$ exists). In the actual implementation every lift maintains a private “tabu” list where every landing call being “tabu” is registered. This list does only apply to the lift which keeps the list.

In addition to the hiding of landing calls, there are the aforementioned means to prevent starvation of passengers. The prevention is performed by setting certain landing calls *tabuTabu*: a landing call being “tabuTabu” may not be passed by lifts anymore and thus must be served at the next possible time. The process of setting landing calls “tabuTabu” is constituted as follows: there is a counter for every floor; every time a lift passes a “tabu” floor, this counter is increased by one. If this counter exceeds a certain predefined threshold, the call is considered “tabuTabu” and may not be passed again. In addition to increasing the counter at passing a “tabu” landing call, the “tabu” for this lift is removed, since the lift will probably not pass the landing call again in the near future. The “tabuTabus” are maintained as a central list being equal for all lifts. The counter is also central and increased if *any* lift passes the call.

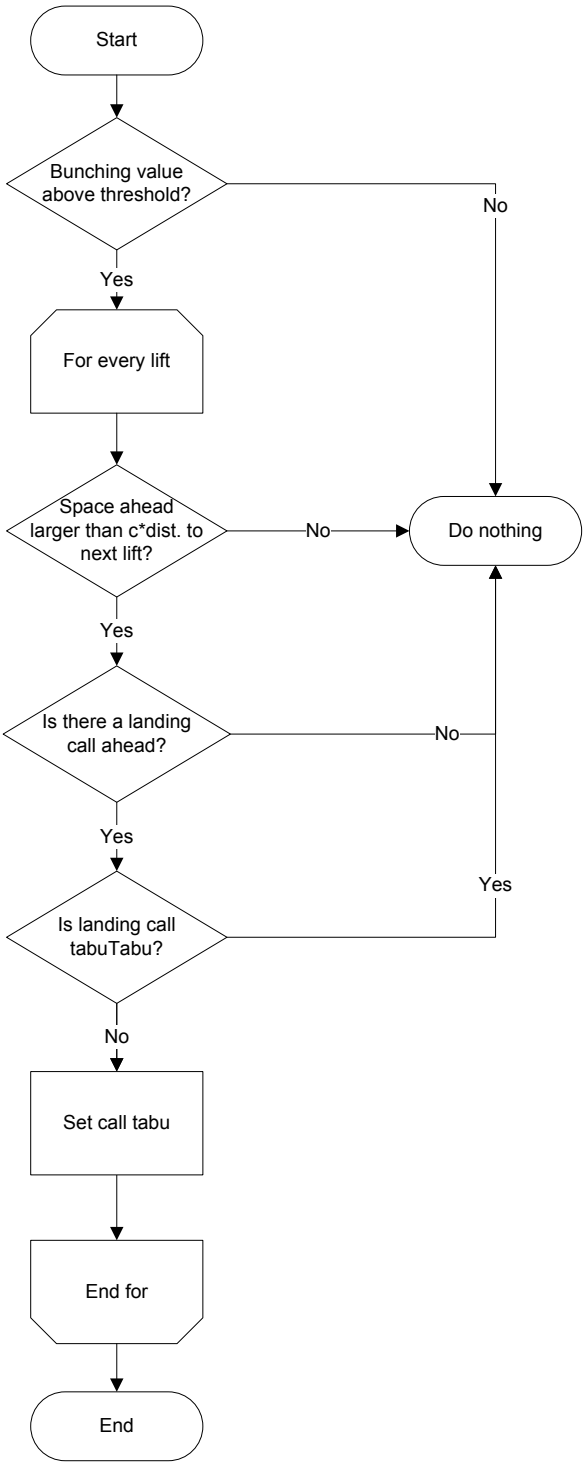



Figure 3.16: Controller strategy: "softIntervention"

Once being "tabuTabu", a landing call remains at this status until it is served by a lift. Additionally, its "tabuTabu" status is removed, if the controller is stopped for any reasons (for example by the user) or the BV falls below the bunching threshold and thus the controller ceases its interventions. Nearly the same thing applies for the "tabu" landing calls. When the controller is stopped or the BV falls below the bunching threshold, the "tabus" of all landing calls in all lifts are deleted. When a landing call is served, all possibly existing "tabus" for this call in all lifts are deleted. Additionally, a "tabu" is deleted in the "tabu" list of the passing lift (*not* in the "tabu" lists of the other lifts), when the lift passes this "tabu" landing call. Furthermore all "tabus" registered for a lift are deleted, if this lift changes its travelling direction.

Compared to the generic interventions introduced in section 2.3.2, "softIntervention" is clearly an approach of changing the lifts perception of its surrounding. Certain landing calls are hidden from some lifts without altering the environment (the landing calls are only hidden, but they exist further on) or changing the decision rules of the lift. The decision rules are not modified, only their input changes due to the hiding of calls. This means the behaviour of a lift being influenced by the controller cannot be expressed in form of a lift strategy showing equivalent behaviour. On the contrary, when "softIntervention" is active, the lifts general strategy is not changed; they rather react to a feigned neighbourhood.

**Comparison of the strategies** The two strategies share many characteristics. Most notably, they use the idea of accelerating lifts to restore an equal lift spacing. Their approach to do this is different, though. "StrongIntervention" is based on altering (parameters of) the lift strategy whereas "softIntervention" relies on altering the lifts' perception of their environment. Another difference between the strategies is the basis of their interventions. "SoftIntervention" intervenes on a per landing call basis by hiding single landing calls from certain lifts. "StrongIntervention" acts based on lifts with disregard of landing calls. A list of these traits along others is given in table 3.5. It also has to be kept in mind that the current implementation of the controller forms only a very basic realization of the generic concept. Neither self-optimizing abilities nor learning methods that enable self-adaptation have been implemented or investigated.

| | **strongIntervention** | **softintervention** |
|---|---|---|
| **Type of intervention:** | Change decision rules of lifts | Alter the lift's perception of the environment |
| **Actual intervention:** | Delayed lifts are blinded and thus serve no landing calls | Landing calls are hidden from delayed lifts |
| **Basis of the intervention:** | On a per lift basis | On a per landing call basis |
| **Range of the intervention:** | Affects multiple landing calls | Affects single landing calls |
| **Additional functionalities:** | Lifts can be partially blinded, i. e. landing calls are served with a probability | Means to prevent starvation of passengers |

Table 3.5: A short comparison of the controller strategies "strongIntervention" and "softIntervention"

# 4 Design of Experiments

We have developed an observer/controller architecture on top of a lift simulation programme comprising the concepts and components described in chapter 3. These concepts have been implemented in the programme with the objective of altering the course of the system. In order to survey if these components fulfil their intended tasks and to examine the impact of the developed observer/controller architecture on the system behaviour, tests have to be performed.

Within this chapter, general design decisions concerning the experiment are outlined; the actual analysis of the results is performed in chapter 5. At first overall design guidelines are mentioned in section 4.1. Hereafter, general thoughts in designing the experiments like defining the exact aim of the experiments are presented in section 4.2. Also, the assessment of the response variables relevant for the intended goal and the choice of the design factors are carried out within this section. At last, the utilized experimental design is introduced in section 4.3.

## 4.1 Design Guidelines

Before performing an experiment, special attention has to be exercised in planning and developing an experimental design. This is especially important as a bad or improper approach can easily render the whole experiment meaningless. For our study, we follow the guidelines for designing experiments given by Montgomery in [Mon05], displayed in table 4.1. However, we will not go into details of experimental design at this place, but only outline our approach and some essential underlying concepts in very short form. For further information Montgomery [Mon05] is a viable source of information.

| Guidelines for Designing an Experiment |
|---|
| (1) Recognition of and statement of the problem |
| (2) Selection of the response variable |
| (3) Choice of factors, levels, and ranges |
| (4) Choice of experimental design |
| (5) Performing the experiment |
| (6) Statistical analysis of the data |
| (7) Conclusions and recommendations |

Table 4.1: Guidelines for designing an experiment according to Montgomery (see [Mon05])

The tasks (1) recognition and statement of the problem, (2) selection of the response variable, and (3) choice of factors, levels, and ranges is performed in the next section. The (4) choice of experimental design is made in section 4.3. Chapter 5 will cover steps (5) to (7).

## 4.2 Pre-experimental Planning

The first step is to define our *objectives* for the experiment. As mentioned before, we want to survey if the developed observer/controller architecture delivers the intended result: to reduce bunching within the lift system and thus to increase system performance. The component exerting influence on the system behaviour, and hence producing measurable effects, is the controller. Therefore, we will test if the controller does actually alleviate bunching within the lift system and thus reduces the AWT of the passengers. At doing this, we also want to examine the influence of controller and system parameters, as for example the choice of different control strategies on the system. Another goal is detect the influence of the prediction functionality on the system performance, as this functionality will probably influence controller actions. In addition we will check the correlation of the BV to the AWT, which could give us an indication of the quality of the bunching measurement with regard to representing the amount of bunching. Thus, our goals can be summarized as shown in table 4.2. However, due to the nature of this test scenario, it would certainly be not worth the effort to perform the experiments with perfect precision including extensive preliminary investigations. For this reason, we will keep the used methods at an appropriate level.

| Goals of the Experiments | |
|---|---|
| (1a) | Gain insights of the influence of the controller on system performance |
| (1b) | Gain insights of the influence of system parameters on system performance |
| (1c) | Gain insights of the influence of the prediction on system performance |
| (2a) | Check the correlation of AWT and BV |

Table 4.2: Goals of the experiments

The second step is to define *response variables*, i. e. variables that provide information of the process under study. The objectives (1a) to (1c) deal with the influence on system performance. As discussed before (see 2.6.2), we measure system performance in terms of AWT (in this regard, optimal performance stands for the AWT being as small as possible). Therefore, varying parameters which influence system performance will also change the AWT, which means the AWT is a measure of the effects of parameter variations on system performance. Additionally, at changing parameters in a way that the AWT changes, the BV should also change if the correlation assumption does hold: as stated before, we consider bunching as the main effect degrading system performance in the present simulation and thus assume the existence of this correlation.

The third step is to consider the factors which may influence the performance of the system. These are *nuisance factors* and potential design factors which split up into *design factors*, *held-constant factors*, and *allowed-to-vary factors* (see [Mon05]). In this case, the only nuisance factor present in the system is the random *seed* for the random numbers generator determining passenger arrival. Design factors are factors actually selected for study in the experiment, i. e. whose influence on the system should be examined. Held-constant factors are factors which are not varied within the experiment. Indeed, they may exert some influence on the system, however, for the purpose of the experiment they are not of interest. Also held constant are the allowed-to-vary factors: these factors are mostly very difficult to vary and have only negligible effect on the system. If those factors exist they are often balanced out by repeatedly performed experiments. However, those factors do not exist in the present case. A list of all factors affecting the system is given in figure 4.1.

As stated before, we will not vary all of those factors as not all of them will affect the system in a manner that justifies the accompanied effort. Factors which will be held constant

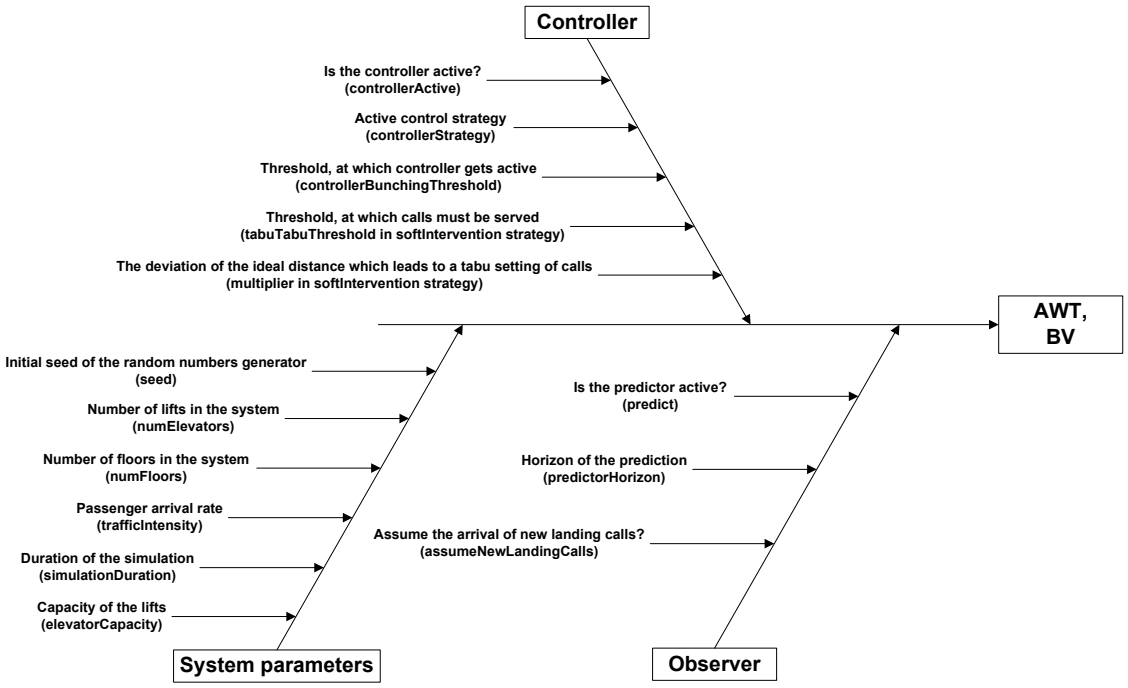



Figure 4.1: Cause-and-effect diagram of the simulation programme

are (a) the assumption of new landing calls by the observer during prediction (*assumeNewLandingCalls*, see section 3.4.1), (b) the threshold at which landing calls will be set tabuTabu (*tabuTabuThreshold*, see section 3.4.2), (c) the deviation of the ideal distance which leads to a tabu setting of calls (*multiplier*, see section 3.4.2), (d) the simulated time (*simulationDuration*), and (e) the capacity of the lifts (*elevatorCapacity*). The parameter *assumeNewLandingCalls* will be held constant as we try to keep the prediction as powerful as possible: a limitation of the predictor abilities rests on further works, as stated before. *tabuTabuThreshold* is kept constant for reasons discussed in chapter 5. The *multiplier* and the *simulationDuration* have shown to have little impact on the outcome of the simulation, why they are held constant, too. The capacity of the lifts, *elevatorCapacity*, is held constant for the sake of simplicity as its variation is not expected to contribute valuable insights. A full list of held-constant and design parameters is given in table 4.3. Actual values of the parameters used within the experiments are discussed in section 4.3.

All remaining factors will be varied. However, there are some peculiarities. The initial *seed* of the random numbers generator is considered as a nuisance factor that has to be averaged out by replication, i. e. by repeated runs of the experiment under the same

| Category | Design factors | Held-constant factors |
|---|---|---|
| Observer | predictorHorizon<br>(predict) | assumeNewLandingCalls |
| Controller | controllerStrategy<br>controllerBunchingThreshold<br>(controllerActive) | tabuTabuThreshold<br>multiplier |
| System parameters | numElevators<br>numFloors<br>trafficIntensity<br>lift strategies | simulationDuration<br>elevatorCapacity |

Table 4.3: Classification of the factors influencing the system. A translation of these coded factors gives figure 4.1.

conditions. Lift strategies are held constant at all but one experiment to simplify matters: one experiment is performed to investigate the effect of lift strategies (see section 5.2.4). Therefore lift strategies are considered as a design factor. Due to the implementation of the random numbers generator it does not make a difference for the outcome which values are actually chosen, as far as they differ from each other. Among the other parameters there are some dependences that one has to be aware of. On the one hand, *ontrollerStrategy* and *controllerBunchingThreshold* depend on *controllerActive*, i. e. if the controller is activated. A turned off controller corresponds to a *controllerBunchingThreshold* of 1.0 and therewith the variable *controllerActive* is not explicitly considered for analysis. On the other hand, the *predictorHorizon* depends on the status of the prediction functionality (turned on or off). Here, a turned off prediction corresponds to a *predictorHorizon* of zero and therewith the variable predict is also not explicitly considered for analysis. For this reason these dependencies disappear.

At last, the *levels and ranges* of the design parameters have to be chosen. For *ontroller-Strategy*, level and range are obvious as this parameter takes only two discrete values (the two strategies). Same applies to the lift strategies. The variables *numElevators* and *num-Floors* will probably take values corresponding to real lift systems: the number of lifts is limited to eight lifts in a group (see [Str98, chapter two]); the number of floors will

probably never exceed fifty floors in real buildings[1]. However, referring to real world lift systems is not necessary as this simulation is mainly a test scenario and not a perfect image of a real lift system. In this case, we will stick to this constraints, though. Reasonable values for the traffic intensity depend on the number of lifts as the arrival rate is absolute and expressed in $\frac{passengers}{tick}$. Therefore, a rate easily manageable for a system of eight lifts will for instance most likely overcharge a lift system containing of two lifts. Hence, the arrival rate should presumably be chosen with respect to the number of lifts. Another technical constraint restricts the choice of the arrival rate: at the current implementation, the rate cannot be set above 1.0 $\frac{passengers}{tick}$. Even so, this is not a relevant restriction for the current experiment as higher rates would overload the system anyway. However, it might be necessary to alter the implementation for surveys of higher buildings or a greater number of lifts. The *controllerBunchingThreshold* can be chosen from zero to unity as it is a percental value. We will discuss the actual values chosen for those parameters in the next section, dealing with the choice of the experimental design, as it is heavily influenced by the design characteristics.

## 4.3 Choice of the Experimental Design

The choice of the experimental design basically depends on the initial situation and on the desired goals of the experiment. We do not have any previous knowledge of the influence of the parameters on the system as we built this observer/controller architecture from scratch. In fact, we want to discover the effects of parameters on the outcome and the dependencies among the parameters: how much does the response variable change when each factor is varied and does it make a difference to vary factors together instead of one at a time. In the literature this request is called *"parameter screening"* and forms the basis for further investigations (see [Mon05]). Under these conditions, Montgomery suggests the usage of a *factorial experimental design*, a very efficient approach to investigate the parameter influence with respect of parameter interactions. We do not consider this in detail but rather refer to the viable introduction Montgomery gives into this field, among presentations of some other experimental designs, in [Mon05]. For this reasons, we use factorial experimental design for our initial experiments.

[1] The total number of floors in some real buildings does exceed this number, however, not all floors are served by one lift group due to sky lobbies (see section 2.5.2).

After the factorial design step we will perform further studies depending on the outcome of parameter screening. All these further steps will be described in chapter 5 together with the actual experiments. Additionally, besides parameter screening, we also want to gain insights about the quality of the BV. The approach for doing this is outlined after the following description of the used factorial experimental design.

At factorial design, all possible subsets of the design parameters are varied with the other parameters held constant. Since parameters are varied together, interactions of the parameters can be revealed. Within experiments aiming at parameter screening, it is general practice to perform so-called $2^k$ *factorial experiments* to keep the experimental effort within certain bounds. Within such an experiment, each of $k$ parameters is tested at two levels. These levels are most likely at a relatively high and a relatively low value of a parameter and can be quantitative as well as qualitative (for example a functionality turned on or off). Therefore such a design requires "only" $2 \cdot 2 \cdot 2 \cdot \ldots \cdot 2 = 2^k$ observations. It is the most efficient way to analyse both the influence of single parameters as well as their interactions on a given parameter set (see [Mon05]). Analysis of only two levels for each factor requires an approximately linear response over the range of the chosen factor levels. We verified this assumption for every factor in preliminary examinations utilizing several factor levels. The chosen levels for the factors have also been obtained by prior investigations. The levels have been chosen in a manner that they produce a significantly different response in *one-factor-at-a-time experiments*. In one-factor-at-a-time experiments, one factor is varied apart from the others in every run of the experiment.[2] We also do not go into the very details of these investigations but only present the parameter levels actually used. All parameters including their levels are listed in table 4.4.

The *predictorHorizon* is either turned off (horizon of zero) or quite short sighted (horizon of two), as a too far horizon has shown to be counter-productive. This is investigated in chapter 5 in detail. The parameters *controllerActive* and *ontrollerStrategy* have only two possible values, which hence is used within the factorial experiment. The remaining design factors take values which have shown to produce significantly different response at one-factor-at-a-time experiments. Additionally the compliance with the linearity assumption has been verified for the used range of values (experiments have shown a deviation from linearity for extreme values in some cases). Upper values of *numElevators*, *numFloors*, and *trafficIntensity* had initially been chosen twice as high as the lower value in each case.

[2] However, at doing this no insights of parameter interactions can be attained.

| Parameter | Values |
|---|---|
| **Design factors** | |
| predictorHorizon | 0, 2 [ticks] |
| controllerActive | false, true |
| controllerStrategy | softIntervention, strongIntervention |
| controllerBunchingThreshold | 0.4, 0.8 [percent] |
| numElevators | 3, 6 [quantity] |
| numFloors | 10, 15 [quantity] |
| trafficIntensity | 0.3, 0.6 [passengers/tick] |
| **Held-constant factors** | |
| assumeNewLandingCalls | true |
| tabuTabuThreshold | 2 [quantity] |
| multiplier | 1.3 [percent] |
| simulationDuration | 50,000 [ticks] |
| elevatorCapacity | 10 [quantity] |
| Lift strategies | Default, (LeftNeighbourOrientation)▲ |
| **Nuisance factors** | |
| seed | 20 differing values per parameter set |

Table 4.4: Used parameter values within factorial experiments. (▲) Lift strategies are only varied within one experiment. Unless otherwise stated, this parameter is held constant at "Default".

As these factors are considered to heavily interact with each other in a more or less linear way. For example, a passenger arrival rate twice as high will likely call for about twice as much lifts in the building to keep the system at roughly the same functional level. However, setting the number of floors to twenty led to a whole collapse of the lift system. Therefore we chose ten as the lower and fifteen as the higher value for *numElevators* during the factorial experiment. The circumstances of the collapse will be investigated in chapter 5.2.2.

Levels of the held-constant parameters have been chosen based on preliminary studies just as the levels of the design parameters. As mentioned before, *elevatorCapacity* and assumeLandingCalls are held constant for reasons stated in section 4.2. We kept the *elevatorCapacity* at the same level as in the initial lift simulation programme. The *multiplier* as well as the *tabuTabuThreshold* is kept at levels which have shown good results beforehand. However, especially in respect of the constant-holding of *tabuTabuThreshold* further explanations are required, see chapter 5. The *simulationDuration* is set to a value that has shown to be long enough to lead to a low standard deviation of the response (longer durations could not provide a significant improvement). To balance out the dependence on the initial lift distribution the first $10,000$ ticks are omitted at logging and are not considered for the calculation of the AWT and the average BV. Thus, effectively only $40,000$ ticks are considered in the analysis.

Besides examining the dependencies of the input variables on the system performance we also want to verify our bunching measure. This will be performed with a regression model considering the AWT of the single runs as the predictor and the corresponding BV as the response.[3] Details can be found in section 5.3.

During all experiments, *nuisance* is balanced out by *replication* of the experiment. Replication means an independent repeat of each factor combination. The number of replicated runs performed within all experiments is twenty, unless otherwise stated. This value has shown to be sufficient to balance out the nuisance factors.

[3] Within regression models, a predictor is a factor influencing the response variable, i. e. the response is dependent on the predictors.

# 5 Realization of the Experiments

Based on these general thoughts stated in chapter 4 we performed several experiments. At the beginning extensive preliminary investigations have been carried out to find reasonable levels for the parameters. We do not go into the details of these investigations but only mention their results. In our opinion an extensive exposure of all performed experiments appeared not to be necessary due to the nature of the whole scenario. The process of parameter level determination with its accompanied problems are touched within section 5.2.2.

After these preliminary investigations, we performed factor screening experiments to gain insights of dependences of the simulation from the input parameters. These experiments can be found in section 5.1. Subsequently, several experiments have been carried out to gain further insights based on those observations. These are described in section 5.2. Finally, we examined the developed BV with respect to its quality to represent bunching in the lift system. This experiment is described in section 5.3.

## 5.1 Factor Screening

For factor screening, we performed a $2^k$ factorial experiment as described in section 4.3. Hereby, we used parameter values shown in table 4.4. To balance out the influence of the random *seed*, which is determining arrival and passenger actions, we performed twenty replicated runs of the experiment for each factor combination to gain a sufficiently low variance.

The actual execution of this experiment is covered in section 5.1.1. Based on insights gained from this initial experiment, the influence of building parameters is discussed in section 5.1.2 while the effect of controller parameters are subject of section 5.1.3.

### 5.1.1 Full Factorial Experiment

Using regression models, we calculate how all main factors as well as all possible subsets of factor combinations affect the response in order to determine which parameters are influencing the system. We chose the AWT of the passengers as the response variable. Therefore, a lower response corresponds to a better system performance. The impact of these parameters can be visualized with a normal probability plot of residuals, as shown in figure 5.1, to easily determine the statistically *active*[1] effects. An effect is either a *main effect* or a *n-way interaction* of $n$ main effects. At this type of plot, the $x$-axis shows the standardized residual of an effect. The $y$-axis shows the percentage of effects being less or equal to the residual, shown on the $x$-axis. The straight line indicates a potential normal probability distribution of the residuals. Unimportant effects tend to be centred around zero and rest on the line. If points depart from this straight line, the normality assumption may be invalid and the corresponding effect may exert influence on the system. In figure 5.1 all important effects are marked with a cross, whereas an effect is important, if it is influencing the response to a probability of at least 95 percent.[2] Additionally, the complete list of coefficients is presented in the appendix in table B.1. Summarizing, this factorial model fits quite well to the observed values as we obtained a $R^2$-value of 98.10 percent, which means that 98.10 percent of the residuals can be explained by the coefficients of the effects. Due to the sheer number of possible interactions, figure 5.1 is not intended to identify all important effects; its purpose is rather to visualize the differing impact of the effects.

In order to get a better overview, we have investigated a subset of all possible effects first. For this reason, the coefficients of the main effects, as well as the ones having the most influence on the response, are listed in table 5.1 (based on table B.1). The $p$-value, given in this table, is the probability of an effect being equal to zero. Thus a $p$-value below an $\alpha$-level of five percent indicates important effects. The value of the coefficient indicates its influence on the response. A positive (negative) algebraic sign indicates a positive (negative) influence of the effect on the response, e. g. an increase of one lift will lower the AWT by 12.83 ticks (estimated, with all other factors held constant). The interaction between two factors *factor1* and *factor2* is represented by *factor1*factor2* (without regard of order); $n$-way interactions are termed accordingly. Interactions affect the response in

[1] We name an effect *active*, if its influence on the response is statistically significant.

[2] This corresponds to a so-called $\alpha$ level of five percent as most often used within hypothesis testing.

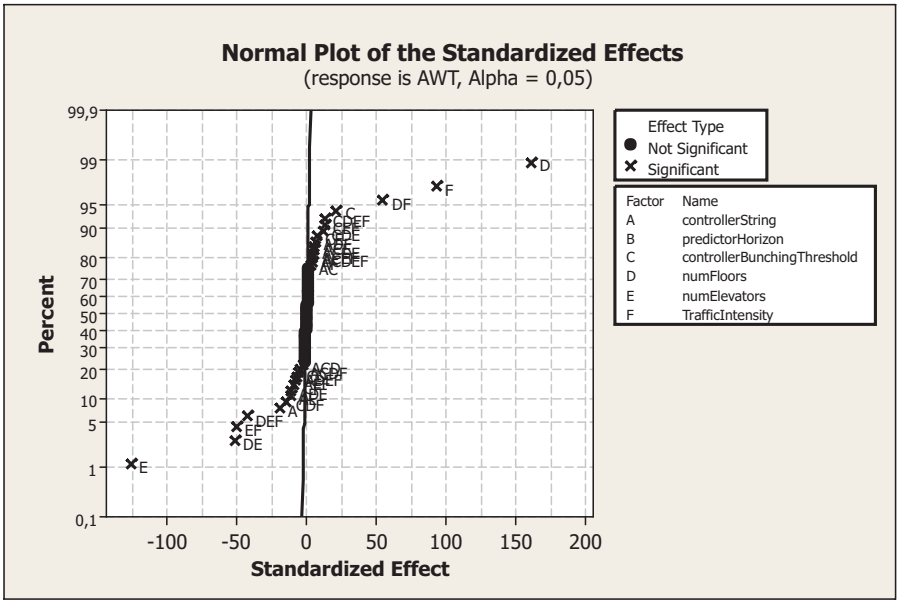



Figure 5.1: Normal probability plot of the effects

the following way: for example *numFloors*numElevators* unequal to zero indicates an interaction between these two factors. Its algebraic sign is negative, so an increase of both *numFloors* as well as *numElevators* leads to a decrease of the AWT resulting from this interaction. The actual variation of the AWT depends on the combined impact of the main effect variations and their corresponding interactions.

Huge differences in terms of influence on the response exist among these effects. Variations in *numFloors*, *numElevators*, and *trafficIntensity* have a far greater effect than changing controllerString, *predictorHorizon*, and *controllerBunchingThreshold*. The same applies for the $n$-way interactions with *numFloors*, *numElevators*, and *trafficIntensity* involved. In contrast, *predictorHorizon* even seems to have no effect at all on the AWT due to its very high $p$-value. We will examine the influence of *predictorHorizon* in detail later on (see section 5.2.3). Due to the dominance of the building parameters we will discuss their influence in section 5.1.2 separately from the other effects, which are examined in section 5.1.3.

| Term | Coefficient | $p$-value |
|---|---|---|
| controllerString | -1.92 | 0 |
| predictorHorizon | 0.06 | 0.54 |
| controllerBunchingThreshold | 2.19 | 0 |
| numFloors | 16.38 | 0 |
| numElevators | -12.83 | 0 |
| TrafficIntensity | 9.49 | 0 |
| numFloors*numElevators | -5.19 | 0 |
| numFloors*TrafficIntensity | 5.58 | 0 |
| numElevators*TrafficIntensity | -5.13 | 0 |
| numFloors*numElevators*TrafficIntensity | -4.33 | 0 |
| ... | ... | ... |

Table 5.1: An abridgement of estimated coefficients obtained from a factorial experiment. The full list is shown in the appendix in table B.1.

### 5.1.2 Effect of Building Parameters

The impact of varying *numFloors*, *numElevators*, or *trafficIntensity* is as intuitively estimated. Increasing the number of floors or the arrival rate result in a higher AWT. In contrast, more lifts lead to a lower AWT. There is also mutual aggravation among *numFloors* and *trafficIntensity*. If they are both increased/lowered simultaneously, the AWT responds to a higher extend than at unilateral deviation. Furthermore, the effect of changing the number of lifts seems to dominate the other effects, which is indicated by the negative effects of *numFloors*numElevators*, *numElevators*trafficIntensity*, and *numFloors*numElevators*trafficIntensity*. In respect thereof, one remarkable point in behaviour occurs if both, the number of floors and the arrival rate, are increased equally. In this case, the AWT tends to slightly decrease since not only *numElevators* is more influential than *trafficIntensity*, but also their interaction is significantly negative. Summarizing, the lift system shows intuitive response on variations of the building parameters. The arrival rate is of less importance compared to the other parameters. The influence of the building parameters on the system will be examined within another experiment in section 5.2.2.

| Parameter | Value |
|---|---|
| numElevators | 3 |
| numFloors | 10 |
| trafficIntensity | 0.6 |

Table 5.2: Parameters held constant at the second analysis of the factorial experiment

### 5.1.3 Effect of the Controller Parameters

As stated before, *numFloors*, *numElevators*, *trafficIntensity*, and their interactions exert heavy influence on the system and most likely superpose the other effects. Hence, we performed another analysis based on the results of the initial factorial experiment to get a better overview of the influence of *controllerString*, *predictorHorizon*, *controllerBunchingThreshold*, and their two- and tree-way interactions. Within this analysis, we do not take variations of *numFloors*, *numElevators*, and *trafficIntensity* into consideration. Instead, we keep those parameters at constant levels shown in table 5.2 and performed a factorial analysis based on the variations of the other parameters. As the foregoing analysis has shown only little interactions among these two "groups" of factors, this step can be done without oversimplification. The normal probability plot of the residuals is shown in figure 5.2. Table B.3 containing all coefficients is shown in the appendix.

Now, the effects of parameter variations within the o/c architecture become more obvious. All three main factors show statistically significant effect on the response. Most notably, a higher *controllerBunchingThreshold* increases the AWT of the passengers to a serious extend. This means, the more often the controller interacts, the better the system performance gets. It could be possible that best results are obtained with the controller being permanently active. We will investigate this observation in detail in section 5.2.1. The choice of the controller strategy does also influence the response at a reasonable level. We set "softIntervention" as the lower and "strongIntervention" as the higher value. Thus the negative coefficient of this effect indicates that the best system performance is archived with a controller based on "strongIntervention".

At the preceding analysis *predictorHorizon* did surprisingly show no significant effect. This time its effect is statistically significant, however, increasing the *predictorHorizon* actually decreases the system performance. The negative influence is of negligible scale

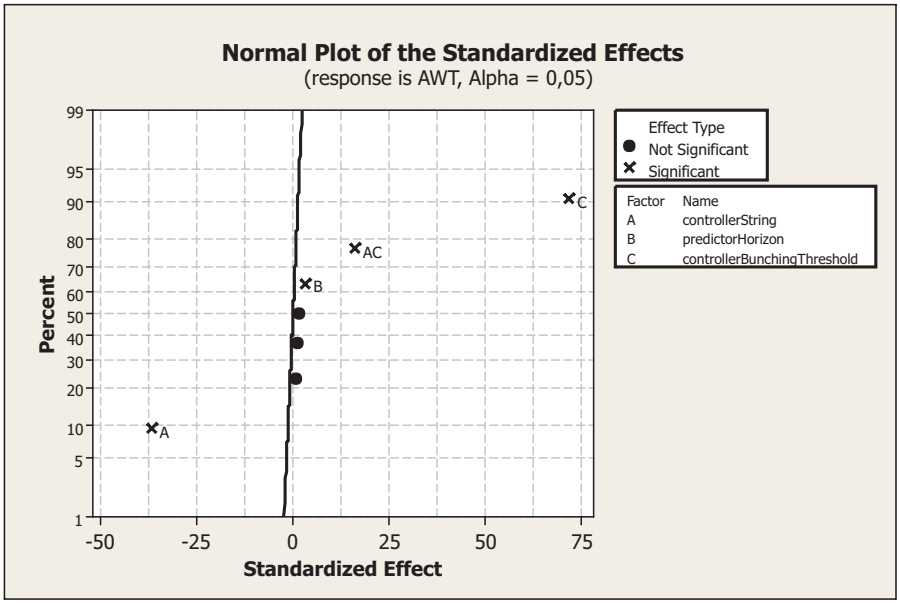



Figure 5.2: Normal plot of the standardized effects without varying *numElevators*, *numFloors*, and *trafficIntensity*

but nevertheless the absent improvement of system performance is very disappointing. This unexpected observation will be examined further on in section 5.2.3.

Additionally, a higher *controllerBunchingThreshold* seems to increase the AWT to a greater extend at "strongIntervention" than at "softIntervention", due to the negative algebraic sign of the corresponding interaction. This influence is caused most likely by an overall better performance of "strongIntervention" in comparison to "softIntervention", see section 5.2.1 for verification.

## 5.2 Further Experiments

Based on the perceptions gained within the factorial experiments further experiments are performed to verify the attained observations. At first, a deeper insight into the general effectiveness of the controller is given within section 5.2.1. Additionally, the influence of the building parameters is investigated in section 5.2.2 and the effect of the lift strategies in section 5.2.4. At last, the effect of the predictor on the system is discussed in section 5.2.3.

| Parameter | Values |
|---|---|
| controllerString | softIntervention, strongIntervention |
| predictorHorizon | 2 |
| controllerBunchingThreshold | 0.0 (always on), 0.1, ..., 0.9, 1.0 (always off) |
| numElevators | 3 |
| numFloors | 10 |
| trafficIntensity | 0.6 |

Table 5.3: Parameter settings for the examination of the controller influence

### 5.2.1 General Effectiveness of the Controller

To observe the control strategies, we performed another experiment varying the *controllerBunchingThreshold* and the control strategies. For this purpose the simulation has been executed with the parameter settings shown in table 5.3. A threshold of 0.0 corresponds to the controller being always activated independent from the reported BV. A threshold of 1.0 corresponds to a deactivated controller.[3] The choice of the *controllerString* has no effect without an active controller and thus a *controllerBunchingThreshold* of 1.0 results in the same AWT for each strategy.

The results are shown in figure 5.3. The system performance is always superior with an activated controller compared to an inactivated controller. Only the extend of the advantage in terms of AWT is different. As shown within the preceding analysis, the influence of the controller strategies on the response depends to a great degree on the *controllerBunchingThreshold*. Hereby, a lower threshold results in a lower AWT. However, when the controller is always activated (threshold equal to 0.0), the AWT finally increases again at both strategies. Most likely, this is caused by passing landing calls unnecessarily at low bunching conditions. As a result, the affected passengers have to wait for a longer time without any positive effect for the system, as the bunching is already at a low level. Additionally, the previously observed characteristic that "strongIntervention" is predominant to "softIntervention" is confirmed anew within this experiment.

[3] During the simulation, the controller is activated when the BV exceeds the threshold. However, a threshold of 1.0 (100 percent) can never be exceeded due to the nature of the BV calculation.

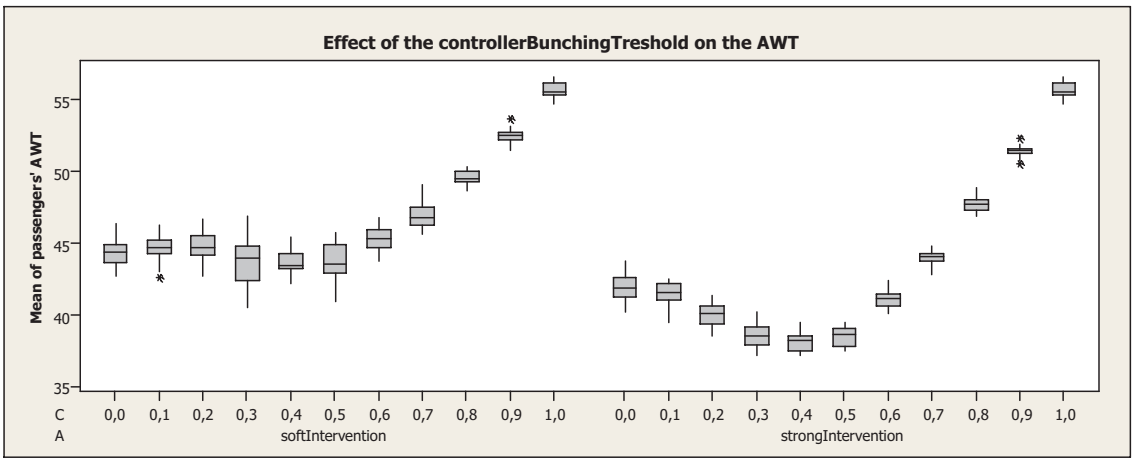



Figure 5.3: Effect of the *controllerBunchingTreshold* (C) on the AWT, grouped by *controllerStrategy* (A). *trafficIntensity* is set to 0.6.

| Parameter | Values |
|---|---|
| controllerString | softIntervention, strongIntervention (averaged out) |
| predictorHorizon | 2 |
| controllerBunchingThreshold | 0.4 |
| numElevators | 3, 6 |
| numFloors | 10, 15, 20 |
| trafficIntensity | 0.6 |

Table 5.4: Parameter settings for the examination of the influence of the number of floors

### 5.2.2 Influence of the Building Parameters

In real world lift systems, special care has to be taken in dovetailing general lift system parameters as for instance the number of lifts in the building. Barney puts this task aptly in [Bar03, chapter six]: *"the difficulty in planning a lift installation is not in calculating its probable performance, but in estimating the likely passenger demand."* Within real world lift systems, especially a correct estimation of the passenger arrivals is a crucial factor for choosing the number of lifts at a given number of floors. In the present case, we use a simple model without consideration of traffic patterns which would introduce varying passenger arrival rates over time. Also, no floors with differing passenger demand, i. e. a non-uniform distribution of passenger arrival among the floors, are considered.

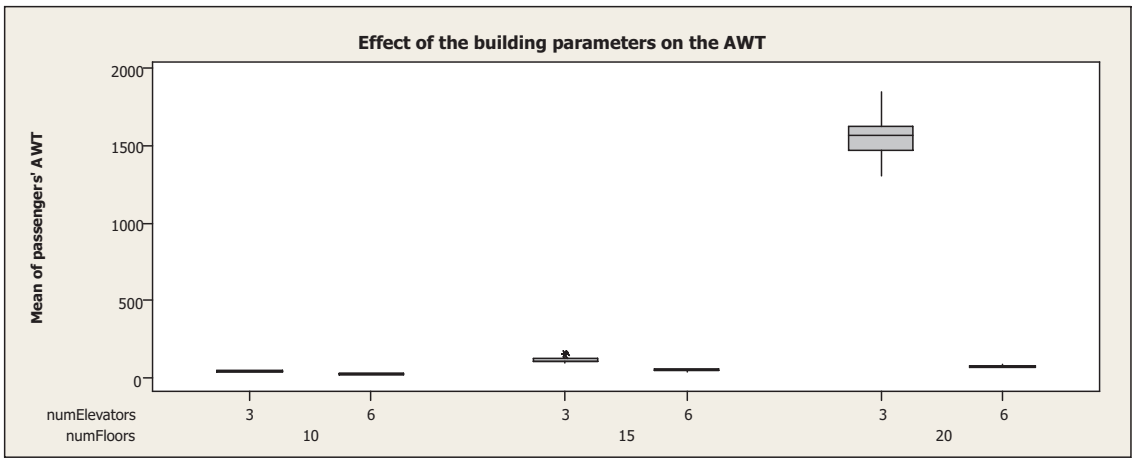



Figure 5.4: Effect of the building parameters on the AWT

During preliminary experiments and within the factorial analysis we observed a massive influence of the building parameters on the system performance. This influence is much greater than any influence of the controller parameters. In order to demonstrate this phenomenon, we perform another experiment with the same parameter settings as seen in table 5.4. We run the simulation with both controller strategies and afterwards, we average out their influence. All other controller parameters are held constant. Compared to the influence of the building parameters their impact is negligible anyhow. Among the building parameters, only the number of floors has been varied. However, the one-factor-at-a-time approach is sufficient to demonstrate the phenomenon of a collapsing lift system. This can be seen in figure 5.4.

It can be stated that the choice of parameter values has to be made with care. Otherwise, the system will not be able to fulfil its tasks in a reasonable manner. The implications of this observation are of larger extend. It is shown for example, that the effect of the emergent phenomenon has to be defined very carefully as it is by no means the only factor exerting influence on the system. In this case, the increase of the number of floors does certainly not amplify the AWT due to an increased bunching; the increase of the AWT is rather caused by the larger distances the lifts have to travel. Thus, “blaming” bunching for the escalating AWT is certainly wrong in this case and might result in drawing the wrong conclusions. The problem of assigning effects on the response to the bunching level is discussed in section 5.3.

| Parameter | Values |
|---|---|
| controllerString | softIntervention, strongIntervention |
| predictorHorizon | 0 (predictor turned off), 1, ..., 15 |
| controllerBunchingThreshold | 0.4 |
| numElevators | 3 |
| numFloors | 10 |
| trafficIntensity | 0.6 |

Table 5.5: Parameter settings for the examination of the influence of the predictor

### 5.2.3 Influence of the Predictor

Within the factorial experiments performed in section 5.1 we cannot verify that the prediction of the bunching value has a positive effect on the AWT. In fact, during the reduced factorial analysis we even observe a negative effect, if the predictor was activated. In order to examine this effect in more detail, we perform another analysis by varying only the *predictorHorizon* and the controller strategies. All other parameters are held constant at the levels shown in table 5.5. The result of the experiment is shown in figure 5.5.

While the controller is operating with "strongIntervention", no effect on the AWT is detectable. By contrast, the AWT grows slightly with increasing *predictorHorizon* using "softIntervention" strategy. The differing impact of those strategies can be explained by the nature of their intervention. Using "strongIntervention", lifts are blinded based on their distances to the lift in front whereas the predicted distance is used if the predictor is active. It blinds always half of the lifts, so the influence of the predictor is basically limited to "jumble" the two halves. Obviously, this perturbation occurs very rarely why prediction seems to have no effect on the controller actions while operating at "strongIntervention".

While operating with "softIntervention" the prediction exerts influence on the controller since the AWT varies with changing *predictorHorizon*. However, this influence is negative as indicated by an increasing AWT as the horizon increases. This observation puts the whole predictor functionality into question as the predictor was developed in order to reduce AWT. The reason for this behaviour cannot be detected definitely. At the present implementation of this strategy, calls are set tabu in front of lifts which will have a large distance ahead in the short future. Why this procedure could have a negative effect on the function of the controller could not been assessed, yet.

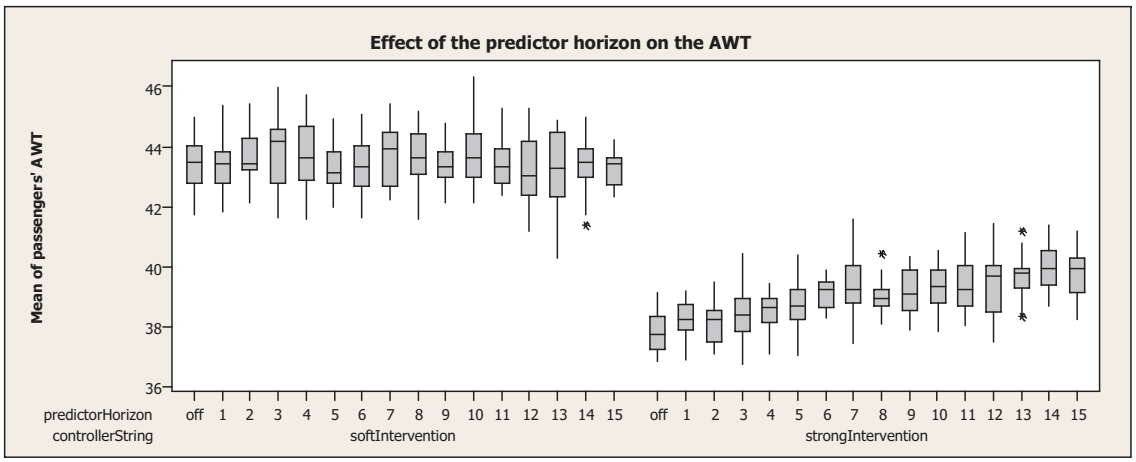



Figure 5.5: Effect of the predictorHorizon on the AWT

For now, the predictor simulates future behaviour which will never eventuate due to the disregard of future controller actions. Apart from this disregard the future is simulated exactly as discussed in section 3.4.1. However, we assume that this characteristic is the major cause for the failure of the predictor at both strategies. Therefore, it seems worthwhile to simulate controller interventions within the predictor in future research. At present time, the predictor seems to be useless.

### 5.2.4 Influence of Lift Strategies

Although focusing on the "Default" strategy at all other experiments, we performed an additional experiment to compare the general effectiveness of the two implemented strategies. This additional experiment is described within this section. We focused on the "Default" strategy at all other experiments to keep the experimental effort at an manageable level.

Within this experiment, we varied lift strategies, controller strategies, and the *controllerBunchingThreshold*. All other factors are held constant. A list of all parameter settings is shown in table 5.6. The result of the experiment is shown in figure 5.6. Label marks of *controllerBunchingThreshold* are not displayed in order to simplify the figure. Within each group, *controllerBunchingThreshold* starts with 0.0 (at the left), gradually increases by 0.1, and ends with 1.0 (at the right).

As observed in the previous experiments, the AWT depends on the choice of the controller strategy. While the controller is operating with "strongIntervention", the AWT is lower

| Parameter | Values |
|---|---|
| controllerString | softIntervention, strongIntervention |
| predictorHorizon | 2 |
| controllerBunchingThreshold | 0.0 (always on), 0.1, ..., 0.9, 1.0 (always off) |
| numElevators | 3 |
| numFloors | 10 |
| trafficIntensity | 0.6 |
| lift strategy | Default, LeftNeighbourOrientation |

Table 5.6: Parameter settings for the examination of the influence of lift strategies

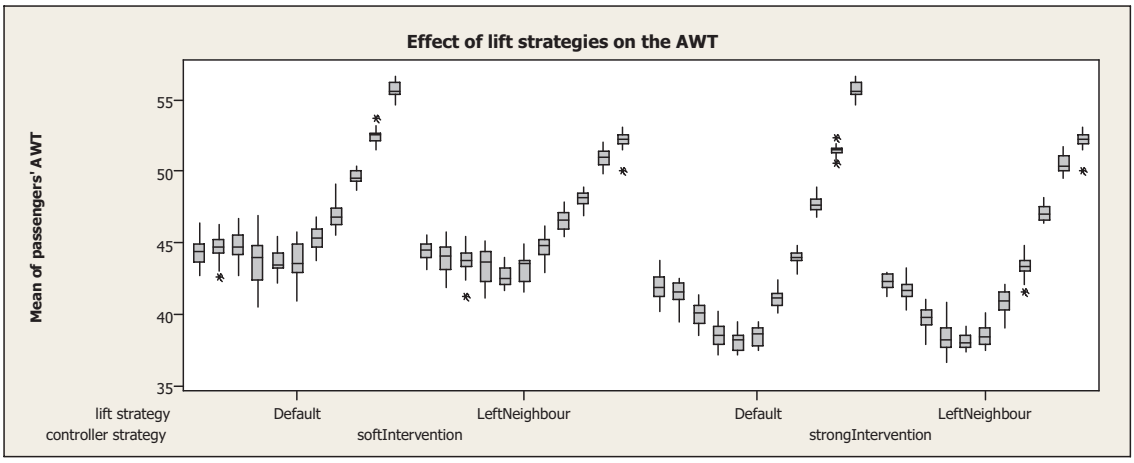



Figure 5.6: Influence of lift strategies on the AWT. Labels of the *controllerBunchingThreshold* are not shown at the $x$-axis.

independent from the lift strategies. This observation corresponds to all prior investigations. Looking at the performance of the lift strategies among both controller strategies, a similar development is visible. Worst performance is shown with an inactive controller, best performance around a *controllerBunchingThreshold* of 0.4, and a slight increase of AWT towards a threshold of 0.0. This general behaviour was also observed before (see section 5.2.1).

There are differences visible among the effect of the lift strategies. While the controller being inactive, "LeftNeighbourOrientation" performs clearly better than "Default". This advantage of "LeftNeighbourOrientation" becomes quickly smaller the more often the controller is activated (i. e. the smaller the *controllerBunchingThreshold* gets). In fact, with a

threshold of 0.8 and below, both lift strategies show nearly the same performance. Better performance of "LeftNeighbourOrientation" is not surprising, since "LeftNeighbourOrientation" is the more powerful strategy because of the lifts ability to communicate. However, the quick decrease of the advantage is not explainable intuitively.

Most likely this behaviour is caused by interaction of two phenomena. On the one hand, controller interactions reduce bunching. Figure 5.7 shows the development of the BV instead of the AWT in order to isolate this effect.[4] Now the development of the response (here the BV) is exactly the same at both lift strategies. Moreover even the level of the BV is identical. This means that "LeftNeighbourOrientation" does not have any effect on the bunching level compared to "Default". On the other hand, the different performance (AWT) at very high *controllerBunchingThreshold* is most likely caused by a characteristic of "LeftNeighbourOrientation": this strategy avoids unnecessary trips to landing calls which are already served by another lift. This does not have any effect on bunching due to the equal development of the BV with both lift strategies. The effect is apparently compensated by the controller. A reason for this could be the reduction of the number of unnecessary trips by the controller. However, this observation cannot be explained by the characteristics of the simulation model. Therefore, it seems more probable that the hiding of landing calls reduces the number of unnecessary stops "unintentionally" and thus levelling the advantage of "LeftNeighbourOrientation". This effect is relatively small, though.

## 5.3 Examination of the Bunching Value (BV)

So far, we have examined and discussed the influence of several parameters on the system performance. In other words, we measured the ability of the o/c architecture to reduce the AWT. Within this architecture, the control actions are based on the information given by the observer. In our case, this information is the amount of bunching in terms of the BV. Thus, it is crucial for our work that this BV represents the status of the system correctly, as all controller interventions are based on it. For this reason, we will verify the suitability of the BV to represent bunching.

[4] The suitability of the BV to represent bunching is proven in section 5.3. Thus, the use of the BV is applicable for the intended purpose.

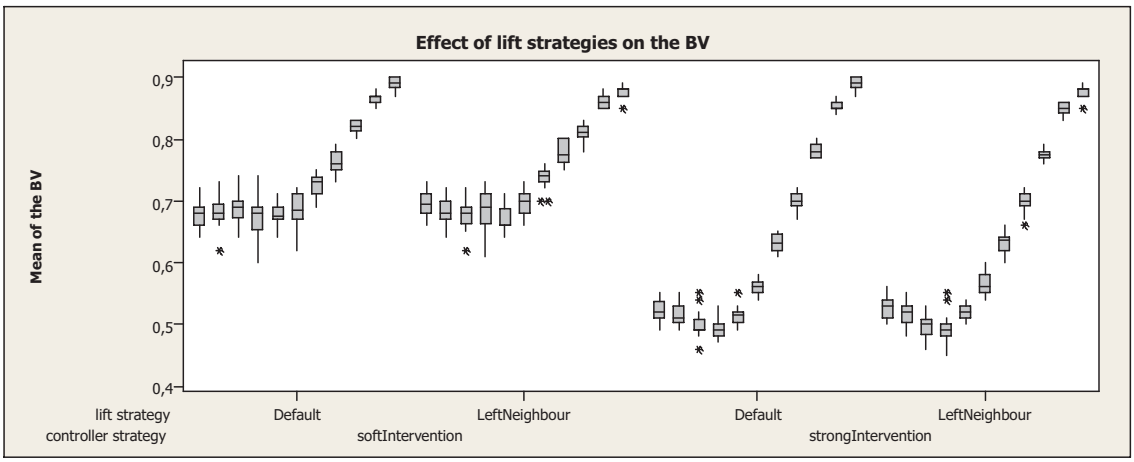



Figure 5.7: Influence of lift strategies on the BV. Labels of the *controllerBunchingThreshold* are not shown at the $x$-axis.

Verifying the quality of the BV is done by comparing current bunching with measured bunching (i. e. the BV). However, quantifying the current level of bunching is a difficult task, since we know no means other than the BV to measure it. Therefore, we use an indirect approach to quantify current bunching: deviations of lift system performance (expressed in a varying AWT) is caused by changing bunching under certain circumstances.

A general assumption on which the BV was developed was the presumption of bunching being the main phenomenon increasing the AWT. Following this assumption, a different AWT will be caused mainly by a different bunching level among different simulation runs. Needless to say, this connexion does only hold, if a different AWT is not caused by other reasons than by a different bunching level. For this reason, we distinguish between two different types of AWT changes. While varying building parameters (*numFloors*, *numElevators*, and *trafficIntensity*) the AWT will most likely change not by reason of different bunching levels. For example a higher number of floors will entail a higher AWT as the lifts have to travel longer distances. By contrast, when varying controller parameters the AWT is expected to change because of a varying bunching level. This results from the only effect of the controller: it reduces bunching by altering lift spacing. Hence, differing controller parameters change the effectiveness of the controller and thus change the bunching level. Therefore, changing controller parameters will change the level of bunching without

influencing the AWT in an other way.[5] Thus, there will be a verifiable connection between the bunching value and the AWT, if the bunching value works. Summarizing, the BV and the AWT will be positively correlated[6] under two conditions. On the one hand, the BV has to be a good representation of the amount of bunching. On the other hand, bunching has to be the main cause of performance degradation within the system. As stated before, the second condition is fulfilled at controller parameter variations.

For these reasons, we examined the correlation between the AWT and the mean BV at each simulation run. The mean BV is evaluated by calculating the arithmetic mean of the current BV observed for each tick. This evaluation is performed ex post due to the impracticality of calculating the AWT of the passengers online. On this account, we check if the aforestated positive correlation exists for different parameter settings. If a correlation exists, the BV is accepted as a good representation of bunching. At first, we performed a correlation analysis based on the data of the full factorial experiment carried out in section 5.1. As expected, it turned out that variations of the building parameters, *numElevators* and *trafficIntensity*, cause a change in the AWT without significantly affecting the BV. A variation of *numFloors* led to unreasonable results: an increase of the number of floors caused a rise of the AWT but a decrease of the BV. As stated in section 5.2.2, the 15 floor lift system is on the brink of an overall collapse so the rising AWT is probably not caused by bunching to a substantial level. For this reason the parameter *numFloors* will not be considered within this analysis and held constant at 10 floors. All other parameters are varied using the values shown in table 4.4. While doing this, the set of all simulation runs is divided into four subgroups defined by the variation of *numElevators* and *trafficIntensity* in order to isolate their effect on the AWT. This approach becomes more apparent when plotting the BVs against the AWT as shown in figure 5.8 which is analysed in the following.

Each point in figure 5.8 symbolizes the outcome of a simulation run in terms of its AWT and its mean BV. The outcome of every run is distinctive since it has been started with a unique input parameter combination (including the random *seed*). Within this plot, a linear relationship (i. e. points arranged in form of a line) would indicate a correlation of the

[5] In section 5.2.4 is was observed that the controller can influence the AWT caused not by a reduction of bunching under certain circumstances. For the *controllerBunchingThreshold* used within this experiment, this effect is negligible, though.

[6] Correlation refers to the Pearson correlation coefficient. It indicates the strength and direction of a linear relationship between two variables. See **insertref** for more information.

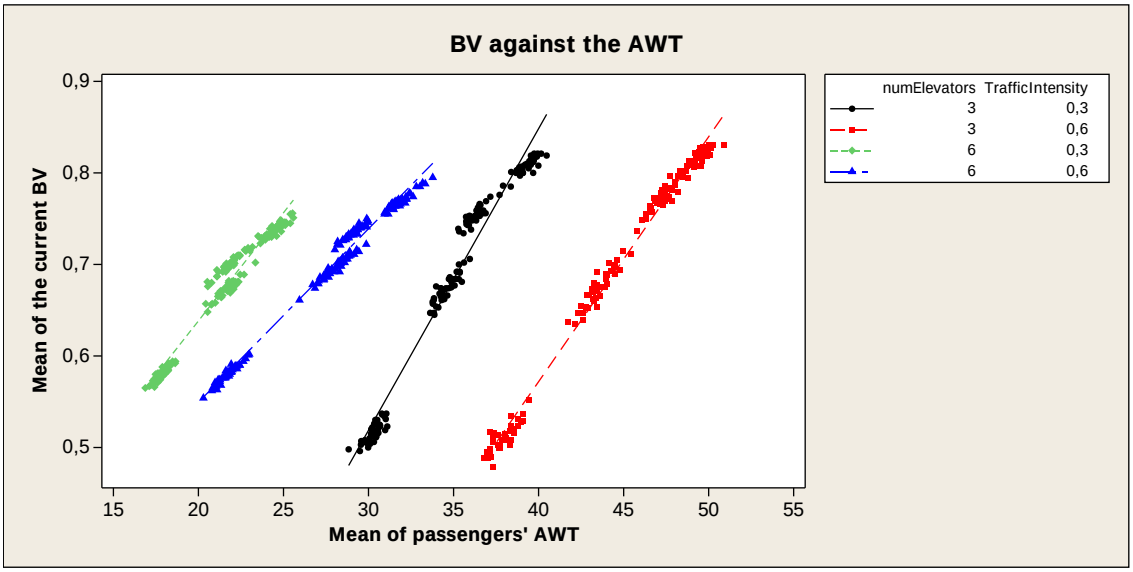



Figure 5.8: BV against the AWT

BV and the AWT. While looking at the scatterplot there is no overall linear relationship apparent. This observation was expected due to the strong influence of the building parameters. Within each of the four subgroups the BV follows a near linear relationship, which is also in accordance with the considerations discussed before. The observed linear correlation within the subgroups is verified by very high correlation coefficients as shown in table 5.7. Additionally, the four subgroups are arranged horizontally shifted in relation to each other. This shift is caused by the fact that a variation of *numElevators* and *trafficIntensity* only alters the level of the AWT without affecting the mean BV. This behaviour results from the BV's independence of the building parameters. For example increasing the number of lifts will result in smaller average distances between lifts – and therewith a smaller AWT – when the other building parameters are held constant. A rather similar effect applies to *trafficIntensity*. The arrival of more passengers leads to a higher AWT as now the lifts tend to stop more often in order to pick up additional passengers. These additional stops cause an increase of the AWT due to the additional door open/close operations. Doing this does not affect the BV as the spacing of the lifts remains the same.

In summary, within each of the four subgroups the BV follows a linear relationship. This behaviour is verified by the correlation coefficients shown in table 5.7 which are very high without exception. Therefore, while varying controller parameters a linear relationship

| Subgroup | | Correlation coefficient |
|---|---|---|
| numElevators | trafficIntensity | |
| 3 | 0.3 | 0.983 |
| 3 | 0.6 | 0.996 |
| 6 | 0.3 | 0.979 |
| 6 | 0.6 | 0.993 |

Table 5.7: Correlation coefficients (Pearson) between the AWT and the BV. The probability that no correlation exists ($p$-value) is zero in any case.

between the mean BV and the AWT has been proven. This connexion between the BV and the AWT complies with the initial expectations. Hence, the mean BV is an accurate representation of system performance (i. e. AWT). While varying controller parameters a changing AWT is mainly caused by a changed bunching level within the system. Thus, the BV can be considered as a very good approximation of the actual bunching level. Additionally, is has been shown that the BV is independent from the building parameters.

## 5.4 Summary of the Experiments

Within this chapter we have performed several experiments and gained cognizance of the influencing factors on the effectiveness of the observer/controller architecture. In the following is given a concise outline of the main insights.

During the initial factorial experiment a great influence of *building parameter*s compared to controller parameters has been observed. This influence has been examined in more detail within a further experiment. It turned out that these parameters exert influence on the system in a way intuitively anticipated. Within further examinations the choice of these parameters has shown to be of critical nature. For example setting the number of floors to an unreasonable value can easily lead the lift system to a full scale collapse. It is imperative to consider these effects by analysing for example the amount of bunching, as it is often by no means a trivial task to separate them.

Since building parameters have shown to superpose *controller parameters*, a separate analysis of the latter has been performed. Especially the choice of the controller strategy influenced the extend of bunching reduction within the system. The strategy taking more

severe actions, “strongIntervention”, turned out to perform better. In addition to these observations, the activation threshold of the controller has been varied. The activation of the controller has always shown to be superior to a system without regulation. However, a permanently active controller led to a slightly increased AWT, although performing significantly better than an uncontrolled system.

Another observation made within the factorial experiment was the apparent misfunction of the *predictor*. This especially is disappointing, as it stands in strong contrast to our expectations. Most likely this behaviour is caused in the predictor's disregard of controller interventions within its simulation. Nevertheless, at the current state the predictor seems to be useless.

Apart from the other experiments, the two different lift strategies have been compared. It has shown, that “LeftNeighbourOrientation” performed better than “Default” if the controller was inactive. While the controller being active, both strategies performed identical. The reason for this behaviour could be the “unintentionally” reduction of the number of unnecessary stops by the controller. This effect is considered negligible, though.

Finally the suitability of the $BV$ to represent the amount of bunching has been examined. This has been carried out by calculating the correlation between mean BV and AWT among different simulation runs differing by controller parameter variations. We regard a correlation as an indication of a good representation of bunching, since varying controller parameters will affect bunching which will in turn affect AWT. Within this analysis a Pearson's correlation coefficient of above 97 percent could be observed, which indicates a strong relationship between the mean bunching value and the AWT. Thus, the developed BV can be considered as a valuable indicator for bunching.

# 6 Summary and Outlook

This thesis investigated the potential of applying concepts of OC to a scenario of a group of self-organising lifts, showing a macroscopic behaviour that depends on local rules only. The lifts synchronise, move up and down together, and show the emergent effect of bunching. In order to reduce bunching we endowed a lift simulation with an observer/controller architecture. The observer monitors the lift system and provides the controller with information of the bunching level. For this purpose, a bunching measure, the bunching value, has been developed to quantify the level of bunching present in the system. The controller acts according to a global objective function: it tries to keep bunching below a certain threshold. If an exceed of this threshold is detected by the observer, the controller starts to intervene. In order to improve the reduction of bunching, the observer has been provided with an ability to predict the future behaviour of the lift system. For controlling the lifts we have implemented two simple methods that modify the perception of the environment and thus affect the local behaviour of the lifts.

Endowing the lift simulation with an observer/controller has shown to reduce bunching significantly in comparison to an uncontrolled lift system. This shows, that bunching can be prevented autonomously with respect to a global objective function. The overall effectiveness of the control strategies was tested by varying the activation threshold of the controller. It has shown, that activating the controller always led to a better system performance than an inactive controller, independent from the other parameters. However, a permanently active controller performed worse than setting the activation threshold to a middle value, letting the controller only intervene if necessary.

This interesting insight allows several implications: in general, interventions in the behaviour of the agents (lifts) succeed in reducing emergent phenomena (bunching) under preservation of the agents' autonomy. However, these interventions have to be well dosed to prevent overshooting control as observed when the controller was permanently active. Therefore, it has to be carefully defined at which time interventions are necessary. This

presumes knowledge of the underlying emergent phenomenon. If this phenomenon cannot be clearly defined and detected, a regulated control is not possible and controller interventions will most likely occur too rare or too frequently (as shown by setting the threshold to a "wrong" value). Additionally, absent understanding of the phenomenon makes directed interventions in all probability impossible, since in this case it is difficult to choose where and how to intervene.

Looking at the controller strategies, it has been shown that even our naive interventions have a significant impact. The strategy comprising more basic rules performed better in reducing bunching. However, this could also be caused by the relatively severe nature of its interventions. This observation is a starting point for further research. At this time, it is not definitely clear if sophisticated strategies are needed to cope with emergence or if relatively simple strategies could provide sufficient results. Additionally, it has to be investigated, if well directed interventions perform better than "unfocused", but severe actions. Therefore, implementing advanced strategies is certainly an opportunity for further research.

It also has to be kept in mind that the current implementation of the controller forms only a very basic realization of the generic concept. Neither self-optimizing abilities nor learning methods that enable self-adaptation have been implemented or investigated. With the current lift simulation model this is not a relevant constraint as no dynamics in the general conditions are considered: neither does the building change (number of floors, lifts), nor varies the passenger arrival model over time. However, adaptation to permanently changing conditions is most certainly one of the major benefits of OC systems. For this reason, endowing the lift system with adaptation capabilities is definitely a valuable starting point for further research. A very promising approach could be the implementation of *reinforcement learning* methods into the controller. Reinforcement learning has already shown good results within lift control scenarios as investigated by Crites and Barto in [CB96, CB98]. A possible implementation of reinforcement learning could use the existing simulation module developed for the prediction functionality. Another alternative would be the use of *Q-learning*, a reinforcement learning technique, where no model of the system is needed to be known by the learning instance.[1]

[1] More information on reinforcement learning as well as Q-learning can be found in Sutton and Barto in [SB02].

Learning is basically directing the degrees of freedom based on feedback of prior controller actions. The degrees of freedom comprise theoretically either the adaptation of controller strategies (by parameter variation), the selection of a strategy, or the creation of new strategies (which is probably an infeasible task). Each action of a controller is evaluated by observing its effect (feedback) and using this feedback to adapt future interventions. If for instance a controller intervention does not reduce bunching in a specific situation, the controller would intervene differently in the future when the system is in a similar state. The adaptation of controller strategies would presume the extension of current strategies or development of new strategies. Present strategies provide probably not enough room for adaptation. The introduction of learning abilities is especially a very interesting field, as several questions raise, to be answered. For example, it has to be assured that a learning system does not develop in an undesired direction. Another point is that a learning process involves making errors, which could cause serious consequences.

Implementing a prediction ability into the observer has shown very disappointing results. The performance was very poor in spite of using a system model for prediction, which was assumed to provide good results. Although simulating a precise estimation of the future, the present implementation of the observer does not take future controller actions into account. For this reason, the estimated future deviates from the "real" future. Moreover, this is most likely the reason why the prediction shows no usable effect. Implementing the consideration of future controller actions could be performed in a reasonable time. Maybe doing this provides further insights on the use of prediction. However, even if showing better performance, such a realization of a prediction module would eventually be far apart from being practicable for real systems. Within permanently changing environments, the observer will not be able to maintain a perfect model of the system needed for a precise prediction. The observer will rather only be able to guess the future with uncertainty. This could limit the practical use of the whole prediction approach.

In order to quantify the level of bunching present in the system, we developed a metric to measure bunching, the bunching value (BV). The BV forms the basis of all controller actions, as the controller decides depending on this value if interventions are necessary. Controller actions depend on the BV, as bunching is considered to be the major phenomenon influencing system performance. During our experiments we could verify the suitability of the BV to represent bunching. Actually, the quality of the representation has shown to be excellent. Moreover, the BV is relatively independent from building parameters, as for example the passenger arrival rate. This especially is necessary, since

otherwise a possibly changed situation (e. g. a different arrival rate) would require an adaptation of the BV.

As mentioned above, an effective control of the bunching effect is only possible, if it can be clearly defined. The BV is certainly a good measure to quantify bunching. However, decreasing system performance could also be caused by other effects than by emergent phenomena. For example setting "wrong" parameter values resulted in a collapse of the lift system without being caused by bunching. Therefore, not each decrease in performance should be associated automatically to the "standard" emergent phenomenon. Measures like the BV, which isolate one emergent phenomenon from other effects, are a good means to detect if other interfering effects may be present.

Implementing advanced lift strategies is another point which could be addressed by further research. Our naive implementation of a strategy comprising intercommunication performed very poor. The bad performance is probably caused by the very basic realization of intercommunication. Therefore, implementing more sophisticated communication means between the lifts could improve performance significantly. Additionally, intercommunication could provide interesting insights of controller effects on such a group of cooperating agents. Also, different traffic patterns or an arbitrarily varying arrival rate could be used to introduce changing environmental conditions. Furthermore, lift distances could be calculated based on estimated travel times instead of spatial distances. In addition, further work should focus on some technical issues. The programme should be altered to allow for passenger arrival rates above 1.0 per tick. Also the introduction of passenger boarding times would probably help to reproduce the bunching effect closer to reality.

Within the limits of this thesis several questions could be answered and many new have arisen worthwhile to be addressed by further research. The bottom line is a successful application of the o/c architecture within the lift simulation scenario. Controlled self-organization has shown to be a promising concept even though being supposably limited by the basic implementation of the current scenario.

# A List of Abbreviations

| List of general abbreviations/terms | |
|---|---|
| AC | Autonomic Computing |
| AM | autonomic manager |
| AMS | autonomous mechatronic system |
| AWT | average waiting time (see 2.6.4) |
| Car call | see section 2.5.1 |
| INT | interval (see section 2.6.4) |
| Landing call | see section 2.5.1 |
| MC | Managed component |
| MFM | Mechatronic function modules |
| MR | Managed resource |
| NMS | Network mechatronic system |
| o/c | observer/controller |
| OC | Organic Computing |
| RTT | round trip time (see 2.6.4) |

Table A.1: List of general abbreviations

| List of parameters of the lift simulation programme | |
|---|---|
| assumeNewLandingCalls | Considers the predictor new landing calls |
| controllerActive | Sets the controller active or inactive |
| controllerBunchingThreshold | Threshold, above which the controller intervenes |
| controllerStrategy | Strategy of the controller |
| elevatorCapacity | Capacity of the elevators in persons |
| multiplier | Parameter within softIntervention strategy |
| numElevators | Number of lifts in the system |
| numFloors | Number of floors in the building |
| predict | Sets the predictor active or inactive |
| predictorHorizon | Horizon of the predictor |
| seed | Seed for the random numbers generator |
| simulationDuration | Duration of a simulation run expressed in ticks |
| softIntervention | A controller strategy |
| strongIntervention | A controller strategy |
| tabuTabuThreshold | The number of times a landing call can be passed |
| trafficIntensity | Passenger arrival rate (passengers/tick) |

Table A.2: List of parameters of the lift simulation programme

# B Regression Results

| Term | Coefficient | p-value | Term | Coef. | p | Term | Coef. | p |
|---|---|---|---|---|---|---|---|---|
| Constant | 48.58 | 0 | A*B*C | -0.08 | 0.43 | A*B*C*D | -0.07 | 0.47 |
| A | -1.92 | 0 | A*B*D | 0.05 | 0.65 | A*B*C*E | 0.1 | 0.33 |
| B | 0.06 | 0.54 | A*C*D | -0.25 | 0.01 | A*B*D*E | -0.06 | 0.57 |
| C | 2.19 | 0 | A*B*E | -0.11 | 0.3 | A*C*D*E | 0.59 | 0 |
| D | 16.38 | 0 | A*C*E | 0.6 | 0 | A*B*C*F | -0.08 | 0.43 |
| E | -12.83 | 0 | A*D*E | -1.12 | 0 | A*B*D*F | 0.09 | 0.4 |
| F | 9.49 | 0 | A*B*F | 0.08 | 0.43 | A*C*D*F | -0.42 | 0 |
| A*B | 0.06 | 0.55 | A*C*F | -0.14 | 0.18 | A*B*E*F | -0.07 | 0.47 |
| A*C | 0.34 | 0 | A*D*F | 0.63 | 0 | A*C*E*F | 0.49 | 0 |
| A*D | 0 | 0.96 | A*E*F | -0.81 | 0 | A*D*E*F | -0.69 | 0 |
| A*E | -1.15 | 0 | B*C*D | 0.07 | 0.52 | B*C*D*E | -0.07 | 0.46 |
| A*F | 0.44 | 0 | B*C*E | -0.13 | 0.21 | B*C*D*F | 0.07 | 0.49 |
| B*C | 0.11 | 0.28 | B*D*E | 0.07 | 0.49 | B*C*E*F | -0.09 | 0.38 |
| B*D | -0.01 | 0.92 | B*C*F | 0.05 | 0.62 | B*D*E*F | 0.04 | 0.72 |
| B*E | -0.01 | 0.9 | B*D*F | -0.01 | 0.89 | C*D*E*F | 1.38 | 0 |
| B*F | 0.02 | 0.81 | B*E*F | 0.07 | 0.51 | A*B*C*D*E | 0.13 | 0.21 |
| C*D | -0.55 | 0 | C*D*E | 1.24 | 0 | A*B*C*D*F | -0.1 | 0.33 |
| C*E | 0.75 | 0 | C*D*F | -1.42 | 0 | A*B*C*E*F | 0.08 | 0.46 |
| C*F | -0.86 | 0 | C*E*F | 1.36 | 0 | A*B*D*E*F | -0.05 | 0.65 |
| D*E | -5.19 | 0 | D*E*F | -4.33 | 0 | A*C*D*E*F | 0.49 | 0 |
| D*F | 5.58 | 0 | | | | B*C*D*E*F | -0.11 | 0.3 |
| E*F | -5.13 | 0 | | | | A*B*C*D*E*F | 0.09 | 0.4 |

Table B.1: Coefficients of the regression model of the factorial experiment. The standard deviation of the coefficients is 0.1. The $R^2$ value is 98.10 percent. The coding scheme of the coefficients is shown in table B.2.

| Symbol | Coefficient |
|---|---|
| A | controllerString |
| B | predictorHorizon |
| C | controllerBunchingThreshold |
| D | numFloors |
| E | numElevators |
| F | trafficIntensity |

Table B.2: Coding scheme of the coefficients

| Term | Coefficient | p-value |
|---|---|---|
| Constant | 44.56 | 0 |
| A | -1.96 | 0 |
| B | 0.17 | 0 |
| C | 3.82 | 0 |
| A*B | 0.09 | 0.11 |
| A*C | 0.86 | 0 |
| B*C | 0.06 | 0.27 |
| A*B*C | 0.05 | 0.35 |

Table B.3: Coefficients of the regression model of the reduced factorial experiment. The standard deviation of the coefficients is 0.053. The $R^2$ value is 97.81 percent. The coding scheme of the coefficients is shown in table B.2.

# Bibliography


[AM04] D. Allrutz and S. Müller. Organic Computing, Computer-und Systemarchitektur im Jahr 2010. Technical report, VDE/ITG/GI-Positionspapier, 2004.

[AS93] L. Al-Sharif. Bunching in lift systems. In *Proceedings of the International Conference on Elevator Technology (Elevcon 93)*, 1993.

[AS96] L. Al-Sharif. Bunching in Lifts...: Why Does Bunching in Lifts Increase Waiting Time? *Elevator World*, 11:75–77, 1996.

[ASB92a] L. Al-Sharif and G. Barney. Bunching factors in lift systems (1). *Control systems centre report UMIST/Manchester*, (749), 1992.

[ASB92b] L. Al-Sharif and G. Barney. Bunching factors in lift systems (2). *Control systems centre report UMIST/Manchester*, (754), 1992.

[Bar03] G. C. Barney. *Elevator Traffic Handbook: Theory and Practice*. Spon Press, 2003.

[Bey07] H. Beyer. Dienstorientierte Ereigniskorrelation. Master's thesis, Technische Universität München, Institut für Informatik, 2007.

[Bib07a] Bibliographisches Institut & F. A. Brockhaus AG. Komplexe Systeme. `http://www.brockhaus-enzyklopaedie.de/`, visited 03.07.2007.

[Bib07b] Bibliographisches Institut & F. A. Brockhaus AG. Selbstorganisation. `http://www.brockhaus-enzyklopaedie.de/`, visited 03.07.2007.

[Bib07c] Bibliographisches Institut & F. A. Brockhaus AG. harmonische Analyse. `http://www.brockhaus-enzyklopaedie.de/`, visited 07.08.2007.

[BMMS+06] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck. Organic Computing – Addressing complex-

ity by controlled self-organization. In *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*, November 2006.

[Bui07] Emporis Buildings. Burj Dubai. `http://www.emporis.com/en/wm/bu/?id=burjdubai-dubai-unitedarabemirates`, visited 10.08.2007.

[Bur07] Burj dubai homepage. `http://www.burjdubai.com/`, visited 10.08.2007.

[But05] T. Butz. *Fouriertransformation für Fugänger*. Teubner, 4th edition, 2005.

[CAL07] CALResCo. The Complexity & Artificial Life Research Concept for Self-Organizing Systems. `http://www.calresco.org/`, visited 11.07.2007.

[CB96] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1017–1023. The MIT Press, 1996.

[CB98] R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2-3):235–262, 1998.

[CB04] E. F. Camacho and C. Bordons. *Model predictive control*. Springer, 2. ed. edition, 2004.

[CMMS+07] E. Cakar, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck. Towards a Quantitative Notion of Self-organisation. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 4222–4229, 2007.

[Cor02] P. A. Corning. The re-emergence of "emergence": A venerable concept in search of a theory. *Complexity*, 7(6):18–30, 2002.

[Dem98] B. Dempster. A Self-Organizing Systems Perspective on Planning for Sustainability MES Thesis. Master's thesis, University of Waterloo, 1998.

[DFG] DFG priority programm 1183: Organic Computing. `http://www.organic-computing.de/SPP`.

[DWH05] T. De Wolf and T. Holvoet. Emergence versus self-organisation: different concepts but promising when combined. In *Engineering Self Organising Systems: Methodologies and Applications*, volume 3464 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 2005.

[Ebe91] W. Ebeling. *Chaos, Ordnung, Information.* Deutsch (Harri), 1991.

[EFS98] W. Ebeling, J. Freund, and F. Schweitzer. *Komplexe Strukturen: Entropie und Information.* Teubner Verlag, 1998.

[Fac07] Fachgebiet Verteilte Systeme Universität Kassel. Dcs-wiki. `http://www.vs.uni-kassel.de/systems/index.php/Emergence`, visited 07.08.2007.

[FPEN02] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback control of dynamic systems.* Prentice-Hall, 4. ed. edition, 2002.

[Fro04] J. Fromm. *The Emergence of Complexity.* Kassel University Press, 2004.

[Fro05] J. Fromm. Types and Forms of Emergence. *Arxiv preprint nlin.AO/0506028*, 2005.

[Gau05] J. Gausemeier. Designing Tomorrow's Mechanical Engineering Products. In *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on*, pages 1–18, 2005.

[GB02] N. Gilbert and S. Bankes. Platforms and methods for agent-based modeling. In *Proceedings of the National Academy of Sciences of the USA*, volume 99, pages 7197–7198. National Academy of Sciences of the USA, 2002.

[GKR99] M. Grotschel, S.O. Krumke, and J. Rambau. Wo bleibt der Aufzug? *OR News Nr. 5*, pages 11–13, 1999.

[Gol99] J. Goldstein. Emergence as a Construct. *EMERGENCE*, 1(1):49–72, 1999.

[Gre06] A. Greenfield. *Everyware: the dawning age of ubiquitous computing.* New Riders Publishing, 2006.

[Gro07] SDG (Swarm Development Group). Swarm. `http://www.swarm.org/`, visited 30.07.2007.

[Hak00] H. J. P. Haken. *Information and Self-Organization: A Macroscopic Approach to Complex Systems.* Springer, 2000.

[Hak04] H. J. P. Haken. *Synergetics. Introduction and advanced topics.* Springer, 2004.

[HOG04] T. Hestermeyer, O. Oberschelp, and H. Giese. Structured Information Processing For Self-optimizing Mechatronic Systems. In *Proceedings of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004)*, 2004.

[HSNL97] U. Honekamp, R. Stolpe, R. Naumann, and J. Lückel. Structuring Approach for Complex Mechatronic Systems. In *Proceedings of the 30th Isata, Florence, Italy*, pages 16–19, July 1997.

[HU97] T. Hikihara and S. Ueshima. Emergent Synchronization in Multi-Elevator System and Dispatching Control. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 80(9):1548–1533, 1997.

[Hui00] C. Huitema. *Routing in the internet.* Prentice Hall PTR, 2. ed. edition, 2000.

[IBM01] IBM Research. Autonomic Computing Manifesto. `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf`, 2001.

[IBM04] IBM Corporation. An architectural blueprintfor autonomic computing. Technical report, IBM Corporation, 2004.

[IBM07a] IBM Corporation. IBM Autonomic Computing Research Website. `http://www.research.ibm.com/autonomic/`, visited 16.06.2007.

[IBM07b] IBM Corporation. IBM Autonomic Computing Website. `http://www.ibm.com/autonomic/`, visited 16.06.2007.

[Int07] Intel Corporation. Moore's Law. `http://www.intel.com/technology/mooreslaw/index.htm`, visited 06.06.2007.

[KC03] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[KLST71] D. H. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of measurement, Vol. I: Additive and polynomial representations.* Academic Press, 1971.

[Lin04] J. K. Lindsey. *Statistical analysis of stochastic processes in time.* Cambridge University Press, 1. publ. edition, 2004.

[Löf07] J. O. Löfken. Mit Magneten gen Himmel. `http://www.ftd.de/forschung_bildung/forschung/:Mit%20Magneten%20Himmel/178646.html`, visited 10.08.2007. Financiel Times Deutschland vom 27.03.2007.

[Luc90] R. D. Luce. *Foundations of measurement, Vol. III, Representation, axiomatization, and invariance*. Academic Press, 1990.

[MA88] R. C. MacDonald and E. Abrego. Anti-bunching method for dispatching elevator cars. `http://patft.uspto.gov/netacgi/nph-Parser?Sect2=PTO1&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=1&f=G&l=50&d=PALL&RefSrch=yes&Query=PN%2F4790412`, 1988. US patent No. 4790412.

[Mac02] J. M. Maciejowski. *Predictive control*. Prentice Hall, 1. publ. edition, 2002.

[MMS06] M. Mnif and C. Müller-Schloer. Quantitative Emergence. 2006.

[Mon05] D. C. Montgomery. *Design and analysis of experiments*. Wiley, 6. ed. edition, 2005.

[MS04] C. Müller-Schloer. Organic Computing – On the Feasibility of Controlled Emergence. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 2–5. ACM Press New York, NY, USA, 2004.

[MSS06] C. Müller-Schloer and B. Sick. Emergence in organic computing systems: Discussion of a controversial concept. In L. T. Yang, H. Jin, J. Ma, and T. Ungerer, editors, *ATC*, volume 4158 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2006.

[MSvdMW04] C. Müller-Schloer, C. von der Malsburg, and R.P. Würtz. Organic Computing. *Informatik-Spektrum*, 27(4):332–336, 2004.

[NCV06] M. J. North, N. T. Collier, and J. R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation*, 16(1):1–25, January 2006.

[NHCV05] M. J. North, T. R. Howe, N. T. Collier, and J. R. Vos. Repast simphony runtime system. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, ANL/DIS-06-1*, 2005.

[OCo07] The Organic Computing Page. `http://www.organic-computing.org/`, visited 16.06.2007.

[OHKK02] O. Oberschelp, T. Hestermeyer, B. Kleinjohann, and L. Kleinjohann. Design of self-optimizing agent-based controllers. In *Proceedings of the 3rd International Workshop on Agent-Based Simulation*, 2002.

[PH98] L. Pike and J. Halpern. Elevator operation and control. In Strakosch [Str98].

[Pic06] C. Pickardt. Implementierung unterschiedlicher Steuerungsstrategien für eine Fahrstuhlgruppe zur Untersuchung des Zusammenhangs von Bunching-Effekt und Leistungsfähigkeit. Master's thesis, Universität Karlsruhe, AIFB, 2006. Studienarbeit.

[Pow94] B. A. Powell. Measurement and reduction of bunching in elevator dispatching with multiple term objection function (usp 5447212), July 1994. Otis Elevator Company.

[Rep07] Repast agent simulation toolkit. `http://repast.sourceforge.net/`, visited 30.07.2007.

[RMB+06] U. Richter, M. Mnif, J. Branke, C. Mller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for Organic Computing. *INFORMATIK 2006*, P-93 of GI-Edition:112–119, September 2006.

[RMS05] F. Rochner and C. Müller-Schlör. Emergence in Technical Systems. *IT-MUNCHEN-*, 47(4):195, 2005.

[SAS97] SAS Institute Inc. Measurement theory: Frequently asked questions. `ftp://ftp.sas.com/pub/neural/measurement.html`, visited 21.08.1997.

[SB02] R. S. Sutton and A. G. Barto. *Reinforcement learning*. MIT Press, 4. print. edition, 2002.

[Sch97] F. Schweitzer. *Self-organization of complex structures*. Gordon & Breach, 1997.

[Sch05a] H. Schmeck. Organic Computing. *Künstliche Intelligenz*, pages 68–69, 2005.

[Sch05b] H. Schmeck. Organic Computing – A New Vision for Distributed Embedded Systems. In *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 201–203, 2005.

[SD02] A. Serenko and B. Detlor. Agent toolkits: A general overview of the market and an assessment of instructor satisfaction with utilizing toolkits in the classroom (working paper 455), 2002. McMaster University, Hamilton, Ontario, Canada.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Tech. J*, 27(3):379–423, 1948.

[Sha01] C. E. Shannon. A Mathematical Theory of Communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, 2001.

[She06] Y. Sheffi. Poker and random bunchin. *The Tech*, 126(50):5, 10 2006.

[SKLT89] P. Suppes, D.H. Krantz, R.D. Luce, and A. Tversky. *Foundations of measurement, Vol. II: Geometrical, Threshold, and Probabilistic Representations*. Academic Press, 1989.

[SM06] D. Samuelson and C. Macal. Agent-based simulation comes of age. *OR/MS Today*, 33(4):34–38, August 2006.

[SPTU05] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland. A concise introduction to autonomic computing. *Advanced Engineering Informatics*, 19(3):181–187, 2005.

[Sta05] K. Stamm. Zukünftige Systemarchitekturen im Kraftfahrzeug. In *25 Jahre Elektronik-Systeme im Kraftfahrzeug*. Bäker, B., 2005.

[Ste05] R. Sterritt. Autonomic computing. *Innovations in Systems and Software Engineering*, 1(1):79–88, 2005.

[Str98] G. R. Strakosch, editor. *The Vertical Transportation Handbook*. John Wiley & Sons, third edition, 1998.

[Tai07] Taipei 101 homepage. `http://www.taipei101mall.com.tw/`, visited 10.08.2007.

[Thy] ThyssenKrupp Elevator. Twin. `http://www.thyssenkruppelevator.com/twin.asp`.

[Thy07] ThyssenKrupp Elevator. Twin – the revolutionary system in elevator design. `http://www.thyssenkrupp-elevator.com/fileadmin/media/pdf/twin_english.PDF`, visited 10.08.2007.

[TNH+06] E. Tatara, M. J. North, T. R. Howe, N. T. Collier, and J. R. Vo. An introduction to repast modeling by using a simple predator-prey example. In *Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects*. Argonne National Laborator, 2006.

[Tos07] Toshiba Elevator and Building Systems Corporation. Toshiba Develops Worlds First Elevator Guiding System With Non-contact Magnetic Suspensions. `http://www2.toshiba-elevator.co.jp/elv/infoeng/pressrelease/20060117e.jsp`, visited 10.08.2007.

[Tra07] Transrapid International. Transrapid. `http://www.transrapid.de/`, visited 10.08.2007.

[Uni02] Universität Paderborn. Sonderforschungsbereich 614: Selbstoptimierende Systeme des Maschinenbaus. `http://www.sfb614.de/`, 2002. visited 06.06.2007.

[vdM04] C. von der Malsburg. Vision as an Exercise in Organic Computing. *Informatik*, 2:631–635, 2004.

[Wei91] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[Wei07] M. Weiser. Ubiquitous computing. `http://www.ubiq.com/hypertext/weiser/UbiHome.html`, visited 03.08.2007.

[WTC07] Lift arrangement of the wtc towers. `http://en.wikipedia.org/wiki/Image:World_Trade_Center_Building_Design_with_Floor_and_Elevator_Arrangment.svg`, visited 10.08.2007.

[YMS03] D. S. Yates, D. S. Moore, and D. S. Starnes. *The practice of statistics*. Freeman, 2. ed. edition, 2003.

**Erklärung**

Ich versichere hiermit wahrheitsgemäß, die Arbeit selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 22. Oktober 2007, Oliver Ribock