New Concepts for Virtual Testbeds

Data Mining Algorithms for Blackbox Optimization based on Wait-Free Concurrency and Generative Simulation

Dem Fachbereich Informatik
der Universität Bremen
eingereichte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

Patrick Draheim

Referenten der Arbeit: Prof. Dr. Gabriel Zachmann
Prof. Dr. Marc Erich Latoschik

Tag der Einreichung: 11. Oktober 2017
Tag des Kolloquiums: 05. Dezember 2018

Für meine Familie

Acknowledgments

Finishing this thesis would not have been possible without many of the people who accompanied me during the time of its creation. Some of them directly influenced my work, while others accompanied me in life beyond. To each and everyone of them I owe the fact that I was able to successfully make it through my Ph.D. time.

First and foremost, I am very grateful to my supervisor, Prof. Dr. Gabriel Zachmann, for providing me with the opportunity to pursue my thesis. He always allowed me to follow my own ideas, providing helpful discussions and advice whenever needed. Moreover, I am thankful for being given the opportunity to join his research group and for the trust he placed in me during the past years.

I also would like to express my gratitude to Prof. Dr. Marc Erich Latoschik for accepting the co-advisorship.

Undoubtedly, this thesis would not exist the way it does, if I had not been supported by my colleague René Weller, who was a constant companion while conducting my research. I am sincerely grateful for his support. I would also like to thank all members of my old research group at the University of Bremen. I always enjoyed our very friendly atmosphere and interesting discussions.

I owe the greatest dept of gratitude to my family which supported me all the time, endured all my stays abroad and extra hours, while constantly pushing me in my interests and ideas. This thesis is not my sole merit and belongs to you, too.

Abstract

Virtual testbeds have emerged as a key technology for improving and streamlining complex engineering processes by delivering long-term simulation and assessment of complex designs in virtual environments. In contrast to existing simulation technology, virtual testbeds focus on long-term physically-based simulation of the overall design in its (virtual) environment instead of only focussing on isolated, specific parts for short periods of time. This technology has the major advantage that costly testing, prototyping, and assessment in real-life environments are replaced by a cost-efficient simulation in virtual worlds for comprehensive and long-term analysis of designs.

For this purpose, engineering models and their requirements are abstracted into software simulation models and objectives which are executed in virtual assessments. Simulation models are used to predict complex, real systems which can be further a subject to random influences. These predictions are used to examine the effects of individual configuration alternatives without actually realizing them and causing possible negative effects on the real system. Virtual testbeds further offer engineers the opportunity to immersively and naturally interact with their simulation model in these virtual assessments. This enables a greater and comprehensive understanding of possible design flaws early-on in the design process for engineers because they can directly assess their design in the virtual environment, based on the simulation objectives. The fact that virtual testbeds enable these realtime interactive virtual assessments, makes their underlying software infrastructure very complex.

One major challenge is to minimize the development time of virtual testbeds in order to efficiently integrate them into the overall engineering process. Usually, this can be achieved by minimizing the underlying concurrency of the testbed and by simplifying its software architecture. However, this may result in a degradation of their very concurrent and asynchronous behavior, which is usually required for immersive and natural virtual interaction.

A major goal of virtual testbeds in the engineering process is to find a set of optimal configurations of the simulation model which maximizes all simulation objectives for the specified virtual assessments. Once such a set has been computed, engineers can interactively explore it in the virtual environment. The main challenge is that sophisticated simulation models and their configuration are subject to a multiobjective optimization problem, which usually can not be solved manually by engineers or simulation analysts in feasible time. This is further aggravated because the relationships between simulation model configurations and simulation objectives are mostly unknown, leading to what is known as blackbox simulations.

In this thesis, I propose novel data mining algorithms for computing Pareto optimal simulation model configurations, based on an approximation of the feasible design space, for deterministic and stochastic blackbox simulations in virtual testbeds for achieving above stated goal. These novel data mining algorithms lead to an automatic knowledge discovery process that does not need any supervision for its data analysis and assessment for multiobjective optimization problems of simulation model configurations. This achieves the previously stated goal of computing optimal configurations of simulation models for long-term simulations and assessments. Furthermore, I propose two complementary solutions for efficiently integrating massively-parallel virtual testbeds into engineering processes. First, I propose a novel multiversion wait-free data and concurrency management based on hash maps. These wait-free hash maps do not require any standard locking mechanisms and enable low-latency data generation, management and distribution for massively-parallel applications. Second, I propose novel concepts for efficiently code generating above wait-free data and concurrency management for arbitrary massively-parallel simulation applications of virtual testbeds. My generative simulation concept combines a state-of-the-art realtime interactive system design pattern for high maintainability with template code generation based on domain specific modelling. This concept is able to generate massively-parallel simulations and, at the same time, model checks its internal dataflow for possible interface errors. These generative concept overcomes the challenge of efficiently integrating virtual testbeds into engineering processes.

These contributions enable for the first time a powerful collaboration between simulation, optimization, visualization and data analysis for novel virtual testbed applications but also overcome and achieve the presented challenges and goals.

Zusammenfassung

Virtuelle Testumgebungen haben sich als Schlüsseltechnologie zur Verbesserung und Rationalisierung von komplexen Entwicklungsprozessen, durch die Bereitstellung von Langzeitsimulationen und -bewertungen von komplexen Entwürfen in virtuellen Umgebungen, entwickelt. Im Gegensatz zu existierender Simulationtechnik setzen virtuelle Testumgebungen auf eine langzeitig, physikalisch-basierte, Simulation des gesamten Entwurfs in seiner (virtuellen) Umgebung, anstatt nur spezifische Teile des Entwurfs isoliert für kurze Zeitspannen zu betrachten. Diese Technologie hat den großen Vorteil, dass teure Tests, Prototypentwicklung und deren Beurteilung in realen Umgebungen durch eine kostengünstige Simulation in virtuellen Welten für umfassende Langzeitanalysen der Entwürfe ersetzt wird.

Zu diesem Zweck werden Ingenieursmodelle und deren Anforderungen in softwareseitige Simulationsmodelle und -ziele abstrahiert und in virtuellen Assessments ausgeführt. Diese Simulationsmodelle werden verwendet, um komplexe, reale Systeme vorhersagen zu können, die unter dem Einfluss stochastischer Prozesse stehen. Diese Vorhersagen werden verwendet, um die Effekte einzelner Konfigurationsalternativen zu untersuchen und zu realisieren, ohne mögliche negative Auswirkungen auf das reale System zu verursachen. Virtuelle Testumgebungen ermöglichen es Ingenieuren zudem, in den virtuellen Assessments immersiv und natürlich mit ihrem Simulationsmodell zu interagieren. Diese Interaktion ermöglicht ein umfassenderes und tiefergehendes Ingenieurverständnis für mögliche Konstruktionsfehler bereits im Entwicklungsprozess, weil Ingenieure ihre Entwürfe in der virtuellen Umgebung direkt, auf der Grundlage der Simulationsziele, bewerten können. Die Tatsache, dass virtuelle Testumgebungen diese echtzeit-interaktiven, virtuellen Assessments ermöglichen, macht ihre zugrunde liegende Software-Infrastruktur aber sehr komplex.

Eine große Herausforderung besteht hierbei, die Entwicklungszeit von virtuellen Testumgebungen zu minimieren, um sie effizient in den gesamten Entwicklungsprozess integrieren zu können. In der Regel kann dies durch die Minimierung der Nebenläufigkeit der Testumgebung und Vereinfachung ihrer Softwarearchitektur erreicht werden. Dies kann aber zur Folge haben, dass ihr sehr nebenläufiges und asynchrones Verhalten beeinträchtigt wird, welches aber normalerweise für eine immersive und natürliche virtuelle Interaktion benötigt wird.

Ein wichtiges Ziel virtueller Testumgebungen im Entwicklungsprozess ist es, eine Reihe von optimalen Konfigurationen des Simulationsmodels zu finden, welche alle Simulationsziele für die angegebenen virtuellen Assessments maximiert. Sobald ein solche Zusammenstellung berechnet wurde, können Ingenieure diese interaktiv in der virtuellen Umgebung explorieren. Die größte Herausforderung besteht hierbei, dass anspruchsvolle Simulationsmodelle und deren Konfiguration einem multikriteriellem Optimierungsproblem unterliegen, das von Ingenieuren oder Simulationsanalysten, in annehmbarer Zeit, nicht manuell gelöst werden kann. Diese Problematik wird weiterhin dadurch verschärft, dass die Beziehungen zwischen Simulationsmodellkonfigurationen und Simulationszielen meist unbekannt sind und dies zu sogenannten Blackbox-Simulationen führt.

In dieser Arbeit stelle ich neue Data-Mining Algorithmen für die Berechnung von Pareto-optimalen Simulationsmodellkonfigurationen vor, basierend auf einer Annäherung des zulässigen Entwurfraums, für deterministische und stochastische Blackbox-Simulationen in virtuellen Testumgebungen. Diese neuen Data-Mining Algorithmen führen zu einem automatischen Knowledge-Discovery Prozess, der keine Beaufsichtung der Analyse und der Ergebnisse für multikriteriellen Optimierungsproblemen von Simulationsmodellkonfigurationen benötigt. Damit wird das obige Ziel erreicht, optimale Konfigurationen von Simulationsmodellen für Langzeitsimulationen und -bewertungen zu berechnen. Darüber hinaus stelle ich zwei komplementäre Lösungen für die effiziente Integration von massiv-parallelen virtuellen Testumgebungen in Entwicklungsprozessen vor. Erstens stelle ich ein neues warte-freies Daten- und Nebenläufigkeitsmanagement basierend auf Hash-Maps vor. Diese warte-freien Hash-Maps erfordern keine Standard-Sperrmechanismen und ermöglichen eine Datenverarbeitung, -verwaltung und -verteilung mit geringer Latenzzeit für massiv-parallele Anwendungen. Zweitens stelle ich neue Konzepte für die effiziente Codegenerierung meines warte-freien Daten- und Nebenläufigkeitsmanagement für beliebige massiv-parallele Simulationsanwendungen von virtuellen Testumgebugen vor. Mein generatives Simulationskonzept kombiniert ein modernes Entwurfsmuster von echtzeit-interaktiven Systemen für hohe Wartbarkeit mit Template-Codegenerierung basierend auf domänenspezifischer Modellierung. Dieses Konzept ist in der Lage, massiv-parallele Simulationen zu generieren bei gleichzeitiger Modellprüfung des internen Datenflusses für mögliche Schnittstellenfehler. Dieses generative Konzept ermöglicht es virtuelle Testumgebungen effizient in Entwicklungsprozesse zu integrieren.

Diese Beiträge ermöglichen erstmals eine leistungsfähige Zusammenarbeit zwischen Simulation, Optimierung, Visualisierung und Datenanalyse für neue Anwendungen von virtuellen Testumgebungen und somit die dargestellten Herausforderungen und Ziele zu überwinden und zu erreichen.

Contents

Part I

Virtual Testbeds: Challenges and Thesis Contributions

Ignorance is the curse of God, knowledge the wing wherewith I fly to heaven.

William Shakespeare

1 Introduction

1.1 Motivation and Challenges

The motivation of this thesis is the development of new algorithms and methods for improving the efficiency and efficacy of virtual testbeds within modern engineering processes by solving the current two main challenges (see Sections 1.1.1 and 1.1.2). In the following, I will introduce the motivation, requirements, and challenges of virtual testbeds that lead to my contributions to the field.

The concept of simulation is widely accepted as the third pillar of science, on par with theory and experiments, and has been further successfully evolved into the modeling, simulation and optimization (MSO) concept [17]. Not surprisingly, MSO fills a crucial role in the development within arbitrary engineering processes. The ability to compute, test and assess simulation model configurations within arbitrary engineering problems reduces development time and cost [87, 89, 152]. Therefore, engineering methods have substantially changed in this context and evolved continuously over the past 40 years [87, 152] (see Figure 1.1). In the past, engineered entities such as assembly lines, supply chains in logistics, robotics or vehicles were designed, prototyped, configured and tested in real life hardware environments (RLHEs), for instance in hardware-in-the-loop processes.

This traditional approach leads to several major disadvantages: it is not only very difficult up to impossible to prepare suitable RLHEs in order to adequately cover all required MSO constraints but it is also very expensive [87, 88, 152]. Consequently, MSO in the form of simulation based optimization (SBO) has emerged as a key technology for solving this problem because it enables a computer-based execution of the engineered entities as simulation models that does not require any RLHE. This can drastically reduce development cost and time depending on both, the entity to be engineered and its' corresponding and required SBO scenarios.

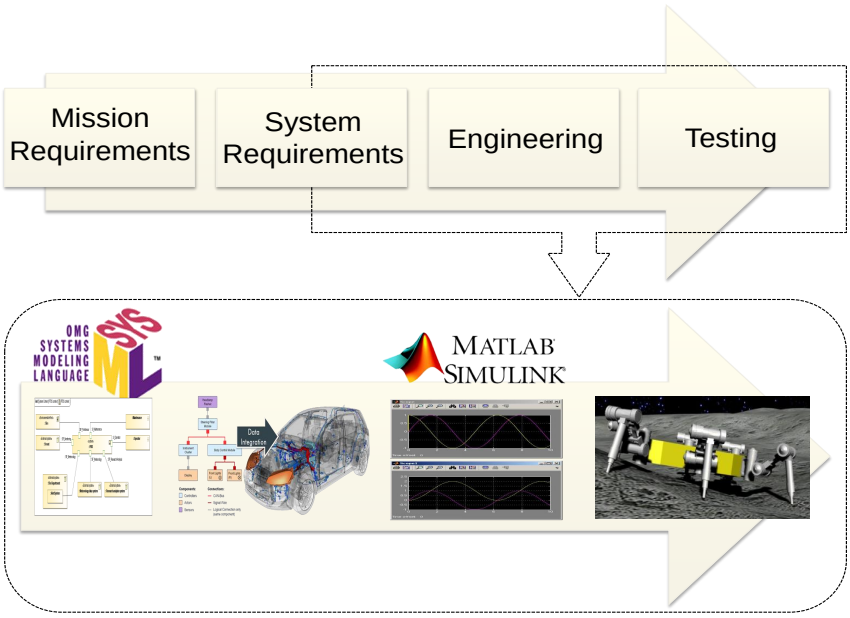


Figure 1.1: The evolution of the engineering process started with requirement based system modelling and implementation with UML and SysML, went over enhanced CAD model approaches [116] and MATLAB/Simulink based evaluations, and culminated in current novel virtual testbeds (e.g. [207]) for advanced engineering process. In contrast to traditional approaches, virtual testbeds combine simulation, visualization and interaction to enable a more in-depth analysis of simulation models.

With the advancement of SBO in recent years, based on the advancement of algorithms and computing technology, realtime interactive systems (RISs), namely virtual testbeds, have emerged as a key player for improving and streamlining such problems as vehicle design, manufacturing and development processes [87, 88, 152]. Virtual testbeds derive from 3D simulation technology (3DST)[1] based approaches and have been extended in the last years with virtual reality (VR) based interaction for better usability. The major goal of virtual testbeds is to bring SBO to the extreme by implementing the simulation as close as possible to reality in a virtual world.

For this purpose, engineering models and their requirements are abstracted into software simulation models and objectives. These simulation models are executed in the virtual testbed as virtual assessments. In these assessments, the simulation model performance is recorded and mapped the simulation objectives. Simulation models are used to predict complex, real systems which can be further a subject to random influences. These predictions are used to examine the effects of individual configuration alternatives without actually realizing them and causing possible negative effects on the real system.

The idea is, in contrast to existing SBO applications (see Section 2.2 and Figure 1.2), to focus on long-term physically-based simulation of the overall design in its (virtual) environment instead of only focussing on isolated specific parts for short periods of time. This technology has the major advantage that costly testing, prototyping, and assessment in real-life environments are replaced by a cost-efficient simulation in virtual worlds for comprehensive and long-term analysis of designs. This concept has the major advantage that testing, prototyping and the assessment in RLHEs can be replaced by a cost-efficient virtual world. This virtual world is, most often, generic and can be re-used for arbitrary purposes or enables the quick generation of additional required scenarios.

[1] This terminology encompasses the combination of a simulation and its' visualization in three-dimensional virtual environments.

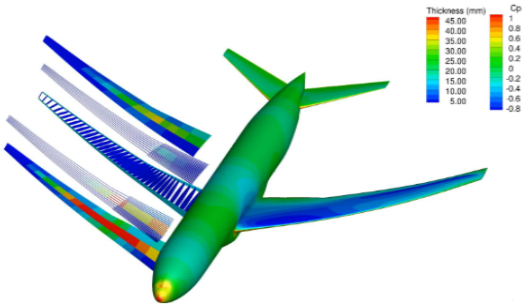


Figure 1.2: Example of traditional optimization of specific parts of a (simulation) model: optimization of plane wings with respect to air drag [199]. The work presented in this thesis focusses on long-term simulation and optimization of whole simulation models. In contrast to traditional approaches, the proposed idea is based on understanding the simulation model as a set of unknown contradictory objective functions that can be approximated. By solving the approximated objective functions, the simulation model in its' entirety is optimized under multi objective constraints.

This means that costly investments for RLHE setups can be replaced by a cost-efficient software implementation. This leads to three fundamental advantages of virtual testbeds with respect to traditional SBO applications:

- First, they give engineers the opportunity to immersively and naturally interact with their entity as a simulation model in the virtual environment. These interactive virtual assessments can enlarge the understanding and recognition of possible design flaws in different settings already in the engineering process. This interaction is usually done via 3D interaction metaphors using either state-of-the-art VR input devices such as body and hand tracking, head mounted displays or traditional input devices (keyboard, mouse). This interactivity in the designated virtual environment with the simulation model is not possible in standard SBO approaches, e.g. [199].

- Second, virtual testbeds enable the generation of arbitrary environmental settings for the simulation model. These settings can cover conditions which are not reproducible in RLHEs, for instance arbitrary extreme conditions for the simulation model. Such extreme conditions (e.g. test cases in which the real world entity could easily break) are essentially required in SBO studies to ensure full test coverage of the simulated entity.

- Third, the resulting simulation data of these scenarios can be recorded, replayed and analyzed for debriefing and in-depth analysis purposes. In addition to the already known advantages of such data [124, 125, 127], this simulation data can be farmed with my novel data mining techniques in order to approximate the behavior of the given simulation model (see Figure 1.3). This simulation model approximation is very important because it can be used to predict the simulation model performance, even without actually simulating the underlying simulation model anymore (see Chapter 5 for the related contributions of this thesis).

These three advantages lead to the aforementioned cost-effectiveness by reproducible, interactive virtual assessments which can cover arbitrary environmental challenges for arbitrary simulation models. As a result of this technological advancement, hardware-focused testing is becoming more and more testing in virtual testbeds [148, 152].

Consequently, virtual testbeds are used, researched, and developed by different research and industrial fields (e.g. supply chains and logistics [188], autonomous space robotics [14, 87, 88], unmanned vehicles [133, 148], automotive [61], manufacturing [141] and military operations [20, 148]) (see Figure 1.4).

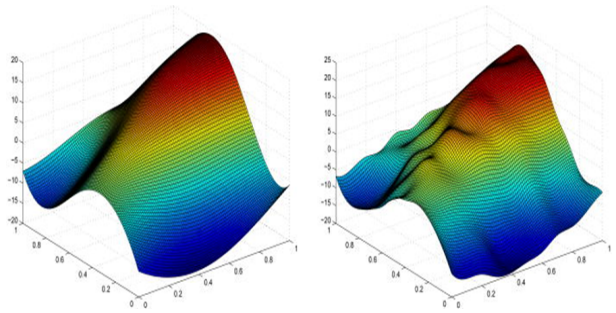


Figure 1.3: The approaches presented in this thesis allow for the first time to approximate simulation models in virtual testbeds and their unknown objective functions. The approximation (left) of the unknown objective function (right) via simulation model parametrizations leads to a loss of information (visible by the smoothing of the surface), which however has no negative effect on the quality of the solutions due to the presented optimization approaches in this work.

In these research fields, expensive hardware is connected with novel complex (mostly autonomous) control algorithms. Due to its complexity and novelty, this development can rarely rely on pre-production experience (such as within the development of regular serial production). Consequently, mostly feasibility studies are required. Therefore, the development, prototyping, and testing of these systems are extremely expensive and error-prone. Virtual testbeds have been established especially in these fields of application because they enable the aforementioned cost-efficient virtual assessments of the system designs prior buying expensive hardware [87, 89, 152].

However, virtual testbed research is still young and, thus, growing. Therefore, two important areas of interest are emerging which I denote as the integration and workflow challenge. These challenges have not yet been mastered and prevent virtual testbeds not only from being fully integrated into the engineering process but also their extraordinary potential for SBO studies can not be exploited. Both challenges are presented in the following sections. It is immensely important and desirable to tackle these challenges: overcoming these would result in a better (quicker) integration of virtual testbeds into the engineering process and to a full exploitation of the virtual testbed opportunities for multiobjective optimization (MOO) in complex and large SBO studies.

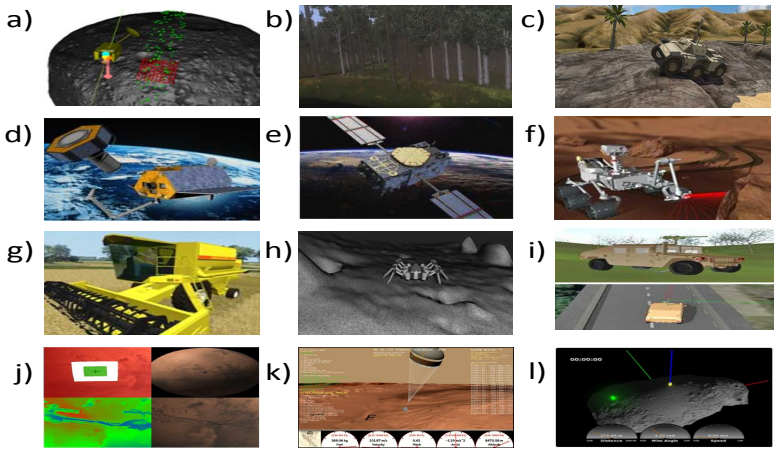


Figure 1.4: Examples of state-of-the art virtual testbeds: a) Autonomous spacecraft navigation for deep space navigation [149] b) Virtual forest testbed [89] c) Autonomous navigation virtual environment laboratory (AUTONOMOUS NAVIGATION VIRTUAL ENVIRONMENT LABORATORY (ANVEL) [148] d) The rendezvous and capture test facility for spacecraft operations (INVERITAS) [85] e) Testbed for intelligent building blocks concept for on-orbit satellite servicing, iBoss [136] f) Self-localization of mobile robots on planetary surfaces, Selok [137] g) Virtual harvesting testbed [47] h) Virtual crater testbed for robotic exploration [207] i) The mobility testbed for unmanned army ground vehicles [99] j) High fidelity dynamics simulator for spacecraft entry, descent and surface landing [74, 128] k) Mars sample transfer testbed [98] l) Testbed for landing and gravity campaigns for spacecraft operations [73].

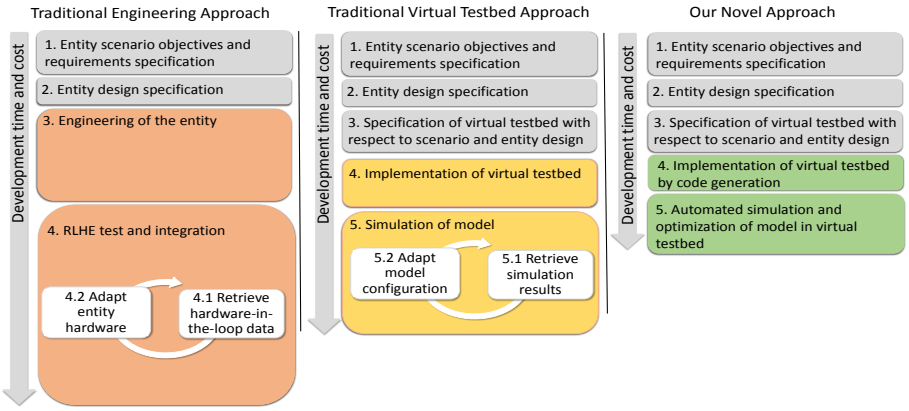


Figure 1.5: The integration challenge: traditional hardware-based testing (left) is currently (partly) replaced with state-of-the-art virtual testbed technology (middle). By integrating virtual testbeds into the process, the aforementioned interactive exploration and analysis of simulation model behavior is possible. However, too much time is required for efficiently integrating these virtual testbeds into the engineering process because of their complexity. My approach (right) aims at 1) efficient development of massively parallel virtual testbeds based on code generation techniques, 2) at automatic optimization of the simulation model. This decreases the development costs and time. Even more, it removes the closed-loop of manual simulation model adaptation (see Chapter 5).

1.1.1 The Integration Challenge

Until now, virtual testbeds are usually integrated in linear fashion into the engineering process [3], e.g. in a five step approach [87, 88, 152] (see Figure 1.5). The development of virtual testbeds requires a huge amount of time as they need to incorporate the complete entity (e.g. structure, sensors, actuators, guidance and control algorithms) and its environment (e.g. physically-based simulation) very accurately in order to guarantee that the simulation results are close to reality. Otherwise, the simulation results are not feasible, leading to unusable simulation data and virtual assessments. Implementing such a required sophisticated simulation involves many disciplines for adequately covering all physically-based aspects. This results in interdisciplinary research and development of virtual testbeds. However, this interdisciplinary development also entails a lot of integration and interface work because the involved disciplines have a different understanding of physical and simulation model properties (e.g. reference frames) and their requirements (e.g. continuous and discrete simulation, realtime behavior) which are required in the simulation.

This integration and interface work is a typical source of error in complex software projects [159]. This system complexity is further aggravated by the functional and non-functional virtual testbed requirements (see Chapter 3). Therefore, the development of the virtual testbed software requires often much time. This leads to the integration challenge because the main idea of virtual testbeds is to provide engineers with simulation results rapidly and constantly in the design part of the engineering process.

However, in current virtual testbeds approaches, the engineers have to wait for the completion of the virtual testbed development while urgently requiring simulation results. This means that the development of virtual testbeds within the engineering process is part of the critical project path[2] and determines the time-to-market[3] of the engineered product.

[2] In project management, a critical path is the sequence of project network activities which add up to the longest overall duration, regardless if that longest duration has float or not. This determines the shortest time possible to complete the project [96].

[3] Time to market is the length of time it takes from a product being conceived until its being available for sale [96].

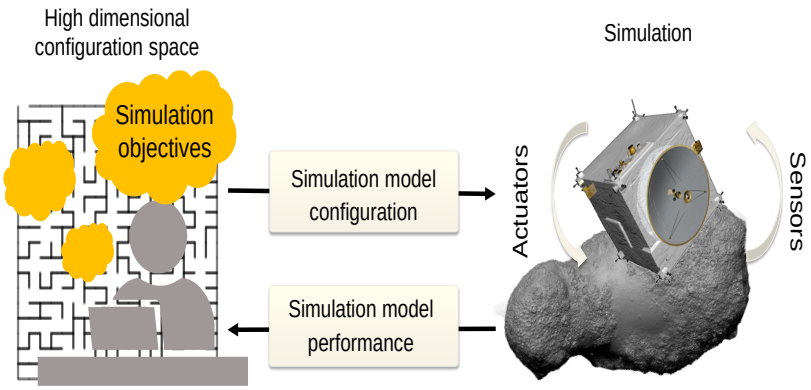


Figure 1.6: The workflow challenge: in the traditional manual approach, engineers are trapped in a closed-loop in order to find a suitable solution for the multiobjective optimization problem (MOP) with unknown objective functions governing the simulation model. Due to the complexity of the optimization and the high dimensional configuration space, no (Pareto) optimal configurations of simulation models can be calculated manually. Another complicating factor is that current virtual testbeds can't perform in-depth analysis of simulation models and their dependencies on parameterizations, which could support engineers in their analysis and decision making.

1.1.2 The Workflow Challenge

In addition to the integration challenge, a second challenge arises. Today, sophisticated simulation models are dominated by a MOP because they model real world problems. These problems involve decisions based on multiple and conflicting criteria [79], very closely. The goal of MOO is to compute optimal decisions that consider the best trade-off among such criteria. MOPs can be found in the presented applications of this thesis but also, for example, in product design where several criteria must be simultaneously satisfied [25, 170, 187].

MSO in the form of SBO in virtual testbeds is used to compute a (optimal) solution of the MOP or to determine the feasible design space (FDS) of the MOP (see Section 2.2) which governs the simulation model.

Such MOPs govern the simulation model because they directly determine the result of the defined simulation objectives. In state-of-the art simulations and virtual testbeds the analysis of the model behavior and the determination of the valid design space, respectively, is usually done manually by simulation experts or the engineer himself.

Due to the increasing complexity of state-of-the-art simulations within virtual testbeds, objective functions are not always available. Even more, there are many technical complex systems whose long-term behavior can not be described by a set of equations. Therefore, it is extremely difficult to compute a set of Pareto solutions or to find a local or global extrema. This kind of SBO problem is called blackbox simulation problem because the objective functions are unknown. Obviously, there exist some optimization approaches (e.g. [43]) which can solve a MOP for known and unknown simulation objectives.

However, they do not consider data mining and analysis based approaches, as presented in this thesis, which can provide viable heuristics. These heuristics originate from a very large pool of data which covers much simulation model information that is usually unknown to the engineer. Thus, these data-driven heuristics often lead to better solutions of the optimization solvers [9, 153, 156].

Generally, the model behavior analysis in these blackbox simulations and the determination of the FDS, respectively, is done manually by simulation experts. The goal is to approximate the behavior of the objective function in order to interpolate and extrapolate the model behavior. The manual analysis method is widely used [83] although it yields many disadvantages because it is based on subjective judgements from the simulation engineer [156, 203]: the simulation expert usually takes an educated guess based on his experience which parameters might be influential on the simulation scope and varies these cleverly in multiple simulations run in order to do both: approximate the FDS and reduce the process complexity [141].

The resulting simulation results are later manually analyzed with respect to the given simulation constraints. This workflow can be partly supported (e.g. visualization of parameter sets, clustering analysis of results) by different tools [43, 113, 141] that can be used by the engineer or simulation expert. However, state-of-the-art blackbox simulation problems are dominated by a MOP, as stated above. Such blackbox simulations can not be determined by manual analysis [79]. Therefore, [83] refers to this as the "trial-and-error approach" because the engineers are literally trapped within the endless possible simulation model configurations (see Figure 1.6).

1.1.3 Requirements

The integration and workflow challenge are further governed by several requirements. These requirements originate from two different research fields. First, by the virtual testbed and its functional software architecture and non-functional system requirements (see Chapter 3), for instance re-usability, adaptability and maintainability. These requirements mainly dominate the integration challenge because they determine the development time. Second, by the underlying SBO requirements (see Chapters 5 and 6) which are required to overcome the workflow challenge.

In the first case, we need to understand what aspects constitute virtual testbeds. Virtual testbeds are essentially RISs because they are designed as very interactive systems with the focus on real-time performance. Within this scope, virtual testbeds strive for being an efficient RIS in order to enable a better understanding of simulation and optimization results. Therefore, they partially yield core aspects of the VR system domain such as the immersive and interactive way to explore computer-generated environments.

As a result of this, their (software) development follows the best-practices of RIS and VR engineering which imposes several functional and non-functional requirements on the virtual testbed implementation. These requirements strive not only for high quality interaction but also on functional aspects. These functional requirements are (realtime) performance, responsiveness, scalability, maintainability, consistency and (re-)usability [55, 56, 119–121, 150, 151, 183, 184].

It is very challenging to achieve these requirements within RIS development because they are usually asynchronous and use highly parallel software architectures in order to satisfy their performance requirements [55, 56, 119–121, 183, 184, 184]. Consequently, RIS research and development strives for reusable patterns and software architectures in order to increase the satisfaction of the above mentioned requirements, especially for massive amounts of RIS software components [117, 140]. Such massive numbers of software components are required by sophisticated simulations and impose additional requirements from the research field of thread synchronization and data consistency. Usually, such design patterns can also directly reduce the development time of RIS because they primarily enhance the re-usability and maintainability of existing software code.

In the second case, we need to understand the underlying MOP that governs the simulated model behavior. Traditional SBO approaches [79, 107] usually require known objective functions which directly describe the influence of all simulation input parameters on the specified simulation objectives (denoted as model behavior).

Optimization toolsets (e.g. [43]) can use these objective functions (e.g. ordinary differential equations) in order to approximate the FDS or to compute local and/or global minima which satisfy given constraints. Even more, in some optimization problems, engineers and simulation analysts are interested in best trade-off solutions among arbitrary criteria. These trade-off solutions, so-called Pareto solutions (see Section 2.2), define a subset of the input parameter space. However, it is extremely difficult to compute a set of Pareto solutions or to find a local or global extrema for the aforementioned blackbox simulations. Here, the objective functions are unknown to both: the simulation engineer and consequently optimization toolset. Nevertheless, a solution for the underlying MOP has to be efficiently computed.

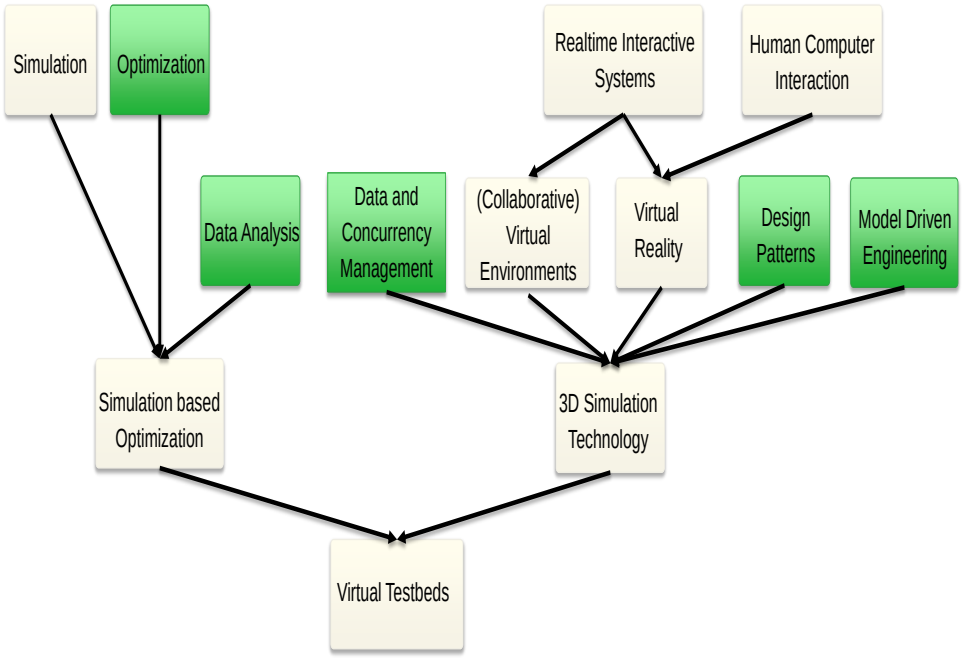


Figure 1.7: Virtual testbeds are an interdisciplinary approach that combines many fields of research ranging from simulation to optimization to 3D simulation. The approaches and contributions presented in this thesis are therefore also located in different research fields that are marked in green.

1.2 Thesis Goal

My motivation for this work is the vision of new approaches and concepts for novel virtual testbeds which overcome the above described integration and workflow challenge. These concepts are summarized in data mining algorithms for blackbox optimization based on wait-free concurrency and generative simulation. This summary directly shows that I make several contributions in different research areas (see Figure 1.7), which directly aim at tackling the presented challenges.

My contributions are technically explained in detail in the following sections. They lead at a high level, in summary, to

- fast development of re-usable and maintainable virtual testbeds based on generative concepts. These virtual testbeds execute simulation, visualization, data analysis and optimization based on wait-free concurrency control,
- efficient approximation of the simulation model behavior based on novel data mining algorithms and to
- automatic optimization of simulation model configurations within blackbox simulations for deterministic and stochastic simulations.

Due to the generative aspect of my contributions, virtual testbeds can be efficiently developed and integrated. In addition, my wait-free concurrency and data management minimizes required software interfaces and enables low-latency data exchange in scalable and massively-parallel virtual testbed applications. Even more, my contributions remove the error-prone closed-loop concept of state-of-the-art virtual testbeds, resulting in an improved efficacy. Consequently, my contributions lead to a novel type of virtual testbeds which are cost-efficient and highly productive for advanced engineering processes.

There are many important applications for my novel virtual testbeds: supply chains and logistics [188], autonomous space robotics [14, 14, 87, 88], unmanned vehicles [53, 133, 148], automotive [61], collaborative training [20] or vehicle design, manufacturing [141] and development processes [152]. I already outlined in the previous sections how my approach is connected to these research fields and what benefits I contribute.

1.3 Overview and Summary of Contributions

In summary, this thesis aims at tackling a number of difficult and interesting challenges by introducing several novel concepts (see Figure 1.8). The following sections of this chapter summarize my novel concepts with their contributions in more detail. In short:
• **Overcoming the integration challenge of virtual testbeds (see Section 1.1.1)** I introduce a highly reusable, maintainable, generic, massively parallel software architecture for virtual testbeds. It enables wait-free parallel execution of SBO applications for simulation, optimization and visualization by a hash map based multi version concurrency control (MVCC). Even more, it leads to increased cohesion and decreased coupling (C&C) by specifying all required interfaces of encapsulated software components within a centralized data management. It further satisfies the requirements of RISs and consists of:

1. A novel efficient, low-latency, wait-free data structure based on hash maps for massively parallel execution of simulation and optimization software components with little synchronization overhead to achieve (above) real-time simulation execution.

2. A novel centralized data architecture in order to avoid quadratic number of interfaces and to decrease coupling between software components. This data architecture can further scale to massive amounts of software components.
3. A novel highly re-usable design concept based on the state-of-the-art entity component system pattern (ECS) design pattern in combination with my hash map concept from above. It specifies the key-value exchange of the hash map and increases the cohesion of the virtual testbed software component.
4. A novel generative concept based on my template based code generation (TBCG) approach within a domain specific modelling language (DSML) concept to quickly generate from platform independent models (PIMs) (based on above wait-free software infrastructure) arbitrary virtual testbed applications (platform specific models (PSMs)) in order to minimize the required virtual testbed development time.

• **Overcoming the workflow challenge of virtual testbeds (see Section 1.1.2)** I remove the closed loop between virtual testbed and engineers with novel data mining algorithms for an automatic knowledge discovery process (KDP), and hierarchical multi agent system (MAS) optimization toolset. Both constitute a novel concept for SBO and MOO in virtual testbeds for deterministic and stochastic blackbox simulations which are governed by a MOP. This concept consists of:

1. Novel data mining algorithms for a novel completely automatic KDP which uncovers hidden relationships between simulation input and simulated model behavior. It uses novel data mining methods which can approximate unknown objective functions in deterministic and stochastic blackbox simulations. These data mining methods are able to approximate the FDS and to compute gradients towards the Pareto front.
2. A novel optimization toolset based on a hierarchical MAS. The agents utilize above FDS approximation in order to compute an optimal solution for the given MOP.

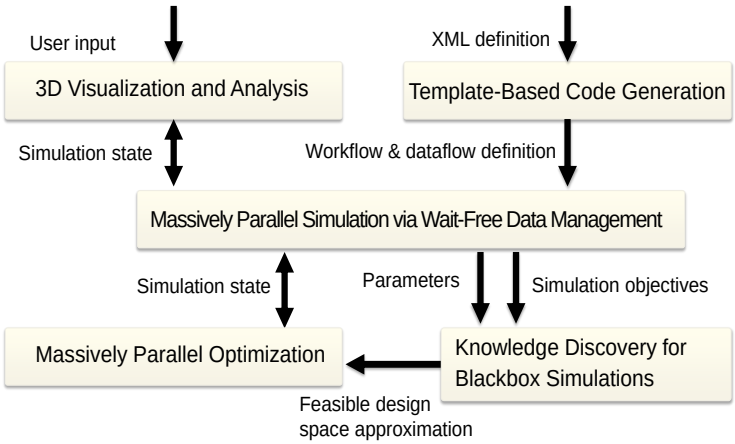


Figure 1.8: Overview of my contributions: a massively parallel simulation based on my wait-free data management is generated by my code generation approach with templates. This code generation includes the automatic realization of the complete work- and dataflow of the virtual testbed as well as model checking. The resulting massively parallel simulation is analyzed by my proposed knowledge discovery process for blackbox simulation models. The result of this analysis, the approximation of the feasible design space, is used for computing Pareto optimal solutions of simulation model configurations by my massively parallel optimization, based on multi agent systems. Finally, simulation and optimization are interactively visualized and presented to the user.

1.3.1 Wait-Free Data and Concurrency Management for Massively Parallel Virtual Testbeds

A central part of RISs, such as collaborative virtual environments (CVEs), virtual environments (VEs), VR systems and 3DST based applications, is the generation, management, and distribution of all relevant data (resp. simulation) states. In modern manifestations of these systems, usually many independent, inhomogeneous[4] software components need to communicate and exchange data in order to simulate the given model as well as to provide data for the visual feedback [150, 151, 154]. In detail, such systems usually consist of many different components such as graphics rendering, sound, several input devices, haptic rendering, physically-based simulation, model behavior, etc. All these components have to share and communicate some kind of data. For instance, the physically-based simulation gathers data from the input devices and passes its results to the graphics, haptic and sound rendering. This requires some kind of interface for the data exchange between the components. This data can be extremely large, for instance in spacecraft and spaceflight simulations [14, 149] (see Section 3.6), where the position of thousands of celestial bodies changes continuously by Newtons' laws of motion. All transformations have to be passed from the simulation to the rendering component. This data exchange is usually done concurrently in highly parallel manner in order to preserve a fast simulation and immersive visual feedback to the user [150, 151, 184]. Therefore, current RISs rely on some kind of data management or structure which is concurrently shared between all software components. Thus, RISs, in general, require a data and concurrency management that is easy to handle and guarantees fast access to data for both, reading and writing while maintaining a consistent simulation state even in heavily concurrent access scenarios [126].

My contribution within above research field is a novel concurrency control management (CCM) and data management that manages concurrent multi-threaded access to shared data in any RIS. My approach does not only reduce the number of required interfaces from $O(n^2)$ of standard approaches to $O(n)$ which benefits better maintainability. Even more, it minimizes the synchronisation overhead compared to standard locking approaches and guarantees simultaneous, wait-free read and write operations. It is based on a wait-free hash map data structure which does not suffer from traditional problems such as deadlocks or thread starvation. This wait-free hash map implements a MVCC based on double buffering. This data structure achieves high scalability because the wait-free concept only introduces small latencies even for massively parallel access. In addition, it provides high performance because it is completely wait-free and in-memory resident. Additionally, it provides high adaptability because it stores all simulation data in efficient object-oriented structures within a graph-based look-up infrastructure. In contrast to the state-of-the-art 3DST applications which utilize full-fledged database technology [124, 125, 127], the time-consuming serialization for data exchange as well as table-based coordination and separation of relational databases are eliminated. My approach stores static and dynamic parts of a simulation model, distributes changes caused by the simulation and logs the simulation run. Even more, my approach implements the same functionality as state-of-the-art relational databases such as aggregate queries and caching strategies. As a consequence, my approach overcomes the associated drawbacks of relational database technology for sophisticated 3DST applications. Additionally, my approach has several advantages compared to other state-of-the-art decentralized methods [124, 125, 127], such as persistence for simulation state over time, fast object and data identification, standardized interfaces for software components as well as a consistent world model for the overall simulation system.

[4] This terminology encompasses differences in type (graphics, physics, haptics, sound, input, craft, and many more) and frequency of the software component.

At last, my data structure incorporates a versioning mechanism which generates a queryable archive of the complete simulation. As a result, simulation components can be used in an online viewing mode to replay a simulation run step by step, allowing analysis and debriefing. From the software engineering point of view, this approach (in combination with my generative concept, see Sections 1.3.2 and 4.2) leads to C&C by forcing the complete dataflow to be accessed by encapsulated software classes (see Section 3). For more details, see chapter 3.

1.3.2 Generative Concepts for ECS based Virtual Testbeds

As introduced in the previous section, a central part of RISs is the generation, management, and distribution of the global simulation a.k.a world state. Usually many independent software components need to communicate and exchange data in these modern systems systems in order to generate this global state. These components and their corresponding performance within a RIS are governed by the functional as well as non-functional requirements of modern RIS development such as (realtime) performance, responsiveness, scalability, maintainability, consistency and (re-)usability [55, 56, 119–121, 150, 151, 155, 183, 184]. It is very challenging to achieve these requirements within RIS development because they are usually asynchronous and use highly parallel software architectures in order to satisfy their performance requirements. Consequently, the need for suitable design patterns which enable reusable software architectures that aim for massive amount of RIS components is rapidly growing in RIS development and research [117, 140].

In the past, the ECS approach has become a major design pattern used in modern architectures for RISs [55, 56, 119–121, 183, 184]. This pattern strives for high re-usability and architectural scalability. The main idea of ECS is to decouple high-level modules such as physics, rendering or sound from the low-level objects with their corresponding data. Therefore, ECS introduces three software architectural objects: `Entities`, `Components` and `Systems`. These are used to describe objects of a RIS via composition instead of object-oriented inheritance. It has been shown that this design pattern can be used to directly enhance the software quality of RISs by introducing semantics [55, 56, 119–121], leading to highly maintainable and re-usable applications and frameworks.

My contribution is an extension of the ECS pattern with my high performance wait-free hash maps with efficient memory management which enables low-latency data exchange within the pattern and reduces their memory consumption. Consequently, I enable the previously described non-locking read and write operations for any ECS pattern based system. Thus, leading to a highly responsive low-latency data access for every `System` while maintaining a consistent state even for structured `Components`. Simultaneously, my contribution greatly reduces the memory footprint of my wait-free hash maps by introducing novel garbage collection techniques. These deallocation techniques directly aim at minimizing the memory footprint for RIS based applications. My novel approach is easy to implement and fits perfectly into the implementation of wait-free hash maps without altering the ECS pattern. My approach therefore greatly benefits the overall RIS performance when using the ECS pattern for improved maintainability and re-usability.

Even more, my approach is combined with the concept of model driven engineering (MDE), namely DSMLs. I developed a DSML that is able to define PIMs for arbitrary virtual testbed applications from which, via code generation, various PSMs can be generated. My approach allows easy and quick modelling of my ECS pattern based wait-free hash map software infrastructure for virtual testbeds. This leads to efficient definition, implementation via code generation, and model checking of virtual testbeds.

My code generation is able to generate the workflow and dataflow for my wait-free hash map based virtual testbeds which use the ECS pattern. This workflow and dataflow is model checked, this means that, at code generation time, possible errors in the dataflow will be found and reported to the user. This minimizes the tedious integration work in the interdisciplinary virtual testbed engineering. At the same time, it uncovers hidden flaws in the designed software architecture of the virtual testbed application. For more details, see chapter 4.

1.3.3 Data Mining Algorithms for Simulation based Optimization in Virtual Testbeds

Traditional SBO approaches [107, 114] usually require pre-defined objective functions which directly describe the influence of all simulation input parameters on the specified simulation objectives (denoted as model behavior). Optimization toolsets, for instance [43], use these objective functions (e.g. ordinary differential equations) in order to find a local or global minimum which satisfies given constraints. As a consequence of the increasing complexity of state-of-the-art simulations within virtual testbeds, such objective functions are not always available. Even more, there are many technical complex systems whose long-term behavior can not be described by a set of equations (e.g. the behavior of autonomous systems in changing environments). This kind of SBO problem is called blackbox simulation problem because the objective functions are unknown to both: the simulation engineer and consequently optimization toolset. There is already a huge number of computational methods for solving MOPs (e.g. [39, 49, 58, 59, 174]) which usually do not consider the generation of vast amounts of simulation model behavior results that can be derived from a KDP in simulations. Usually, the traditional approaches use heuristics of the unknown objective functions for their algorithms. However, these approaches converge much better to local or global minima when they are enhanced with additional information about the MOP [203]. I propose for this purpose an approximation of the complete simulation model behavior.

In contrast to state-of-the-art approaches, which are not able to automatically analyze blackbox MOPs in simulations, my approach automatically actively builds a model between simulation input and simulation objectives. This approximation of the simulation model behavior directly leads to an approximation of the FDS of the simulation model configuration space for a Pareto based MOO. My novel data mining algorithms and concepts lead to a novel KDP that uncovers unknown causal relations in large parameter sets between simulation input and model behavior which are assumed to be unknown non-linear objective functions. In detail, it approximates objective functions (resp. the FDS) in arbitrary deterministic and stochastic blackbox simulations as B-spline surfaces. It computes a gradient from this FDS approximation towards concave, convex or interrupted Pareto fronts. This gradient is used by my hierarchical MAS approach, as described in the next section. My approach is completely automatic, it does not need any supervision from simulation experts. Another advantage of my approach is its performance. It gains its efficiency from a novel spline-based sampling of the parameter space in combination with a novel forest-based simulation dataflow analysis. Another main advantage of my approach is that the evaluation of my B-spline surface based FDS approximation for simulation model behavior data is computationally very fast and can, thus, replace costly simulation-based evaluations which are usually required. Consequently, my approach can also delivers a performance boost when computing a solution for the given MOP. Furthermore, my approach is very generic. It can be easily incorporated into existing SBO approaches which already use a KDP. Even more, the computed Pareto solutions from my evaluations are close to the Pareto front for both, deterministic and stochastic simulations.

Another advantage of my approach are the provided optimization strategies which cover different aspects such as reliability of the solution or the coverage of the input space by my data mining approach. These strategies can be used by state-of-the-art MOO solvers in order to investigate a larger bandwidth of the simulated model behavior. For more details, see chapter 5.

1.3.4 Multi-Agent System for Massively Parallel Optimization

In order to utilize my FDS approximation for computing an optimal simulation model configuration solution, I use a highly parallel optimization system based on my wait-free data management. This optimization system is a hierarchical MAS which aims at dynamically tuning all given input configuration parameters with respect to the approximated FDS, which is retrieved from my KDP. Such hierarchical MAS have already proven their feasibility for solving MOP [43, 103, 109, 152, 164]. My main idea is that every agent introduces a part-wise modelling (singleobjective optimization (SOO) and MOO constraints per input parameter) of the problem and its behavimy and communication to other agents is used to solve the global (MOO) problem.

Instead of using a costly evolutionary approach (such as proposed by [103, 109, 164]), my MAS directly utilizes my cost-efficient FDS approximation and can converge in only a few iterations to the solution (see Section 6.6).

My MAS is composed of several agent organizations. Each of these organizations aim at optimizing a subset of configuration parameters for one or more simulation objective, each one represented by my FDS approximation. Within these organizations, the agents follow the principle of negotiation based agent communication. In my case, all agents negotiate based on the provided FDS approximation from which the agents compute gradients towards the Pareto front. Every agent organization is managed by an specific negotiation agent. It handles requests between the other agents in order to satisfy the existing MOO constraints. These requests are based on a heuristic which the agents apply. Based on the corresponding subset of configuration parameters, this heuristic determines whether or not a better solution is reachable and approximates a corresponding cost. This cost is the number of negatively affected simulation objectives by changing the corresponding configuration parameter set.

My MAS perfectly harmonizes with my ECS based, wait-free, massively parallel data management approach because it delivers the required low-latency communication and adaptability for many homogeneous agents. In detail, all agents can communicate and exchange data very quickly, increasing the overall performance of the optimization process. Additionally, agents can be added and removed at runtime to the optimization system without the need of restarting the optimization run. For more details, see chapter 6.

In the following, I will explain the previously introduced concepts in more detail. Following a quick overview of theoretical background in chapter 2, I explain my hash map based MVCC for RISs in chapter 3. Further, I introduce in chapter 3 how the ECS pattern can be adopted for the previously described hash map based data management and how this combined concept can be efficiently implemented by using a DSML. I present my KDP in chapter 5 and the accompanying MAS based optimization approach in chapter 6. I conclude with a summary of my contribtions and directions for future work in chapter 7. Finally, a summary of my publications and awards is given.

Listen to many, speak to a few.

William Shakespeare

2

Brief Overview of Techniques for Data Management and Analysis, and Simulation based Optimization

In this chapter, I introduce several topics related to virtual testbeds namely RISs, SBO, MOPs, and knowledge discovery in databases (KDD) (resp. data mining within KDPs). I recall some of the most relevant research articles that have appeared in the international literature related to these topics. The presented state-of-the-art does not have the purpose of being exhaustive; it aims to drive the reader to the main problems of this interdisciplinary work and the approaches to solve them.

2.1 Data Management: Concurrency Control in Realtime Interactive Systems

As previously described, a central part of RISs, such as CVEs, VEs, VR systems and 3DST based applications, is the generation, management and distribution of the global simulation state. In state-of-the-art RIS and 3DST applications, usually many independent software components are required to execute the simulation model as well as to provide data for the visual feedback. This data exchange is usually done concurrently in highly parallel manner in order to preserve a fast simulation and immersive visual feedback to the user [154]. Consequently, when conceptualizing and implementing above applications in massively parallel manner, efficiently managing concurrency control is inevitable.

Usually, these data management and concurrency control concepts (summarized as a CCM),are related to database research. However, CCMs also have been applied on data management in virtual testbed research, thus, I treat the terms database and data management equally within this section as database management system (DBMS).

A CCM ensures correct results for concurrent operations on shared data, while getting those results as quickly as possible. When components operate concurrently by messaging or by sharing accessed data, a certain components' consistency may be violated by another component. This means that either a corrupted state is generated or that information is lost when no concurrency control is introduced. In this case, one could never rely on any parallel execution of operations.

The general area of CCM provides rules, methods, design methodologies, and theories to maintain the consistency of components operating concurrently while interacting, and thus the consistency and correctness of the whole system. These concurrent operations are denoted as transactions[1].

[193] defined the atomicity, consistency, isolation, durability (ACID) properties which each transaction has to obey when executed in CCM system, based on the work by [78]:

- **Atomicity** Each transaction is "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the DBMS state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes. To the outside world, a committed transaction appears (by its effects on the DBMS) to be indivisible ("atomic"), and an aborted transaction does not happen.

- **Consistency** Every transaction must leave the DBMS in a consistent (correct) state, i.e., maintain the predetermined integrity rules of the DBMS (constraints upon and among the objects of the DBMS). A transaction must transform a DBMS from one consistent state to another consistent state (given that the transaction itself is correct, i.e., performs correctly what it intends to perform while the predefined integrity rules are enforced by the CCM). Since a DBMS can be normally changed only by transactions all the states of the DBMS are consistent.

- **Isolation** Transactions can not interfere with each other (as an end result of their executions). Moreover, usually (depending on the concurrency control method) the effects of an incomplete transaction are not even visible to another transaction. Providing isolation is the main goal of concurrency control.

- **Durability** Effects of successful (committed) transactions must persist through crashes (typically by recording the transactions' effects and its commit event in a non-volatile memory).

Therefore, a transaction, by definition, must obey the ACID properties. If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency exists.

However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur, such as:

- **The lost update problem** A second transaction writes a second value of a data-item on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.

- **The dirty read problem** Transactions read a value written by a transaction that has been later aborted. This value disappears from the DBMS upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.

[1] A transaction symbolizes a unit of work performed within a DBMS against a shared data structure, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in a DBMS.

- **The incorrect summary problem** While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Most high-performance transactional systems need to run transactions concurrently to meet their performance requirements. Thus, without CCM such systems can neither provide correct results nor maintain their DBMSs consistent.

The main categories of CCMs are:

- **Optimistic** Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g. serializability and recoverability) until its end, without blocking any of its (read, write) operations, and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once).

- **Pessimistic** Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.

- **Semi-optimistic** Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking (if needed) to transaction's end, as done with optimistic.

These categories provide different performance, i.e., different average transaction completion rates (throughput), depending on transaction types mix, computing level of parallelism, and other factors. If selection and knowledge about trade-offs are available, then the category and method are usually chosen to provide the highest performance.

The mutual blocking between two or more transactions (where each one blocks the other) results in a deadlock, where the transactions involved are stalled and can not reach completion. Most non-optimistic mechanisms (with blocking) are prone to deadlocks which are resolved by an intentional abort of a stalled transaction (which releases the other transactions in that deadlock), and its immediate restart and re-execution. The likelihood of a deadlock increases with the number of transactions. Blocking, deadlocks, and aborts result in performance reduction.

Many CCM methods exist which reside within one of the main categories above (see Figure 2.1). The major methods [144] have many variants and overlay in some cases:

- **Locking** Controlling access to data by locks assigned to the data. Access of a transaction to a data item (DBMS object) locked by another transaction may be blocked (depending on lock type and access operation type) until lock release.

- **Serialization graph checking/Precedence graph checking** Checking for cycles in the schedules' graph and breaking them by aborts.

- **Timestamp ordering** Assigning timestamps to transactions, and controlling or checking access to data by timestamp order.

- **Commitment ordering** Controlling or checking transactions' chronological order of commit events to be compatible with their respective precedence order.

- **Multiversion concurrency control** Increasing concurrency and performance by generating a new version of a DBMS object each time the object is written, and allowing transactions' read operations of several last relevant versions (of each object) depending on scheduling method.

- **Index concurrency control** Synchronizing access operations to indexes, rather than to user data.

- **Private workspace model (deferred update)** Each transaction maintains a private workspace for its accessed data, and its changed data become visible outside the transaction only upon its commit.

Many DBMSs rely upon locking strategies to provide ACID capabilities. In more detail, locking means that the transaction marks the data that it accesses so that the CCM knows not to allow other transactions to modify it until the first transaction succeeds or fails. The lock must always be acquired before processing data, including data that is read but not modified. Non-trivial transactions typically require a large number of locks, resulting in substantial overhead as well as blocking other transactions. For example, if software component A is running a transaction that has to read a row of data that software component B wants to modify, component B must wait until component As' transaction completes. There a several variants of these locking approaches such as LCN (Locking a Central Node), G2PL (Global Two-Phase Locking), Weaker Consistency or PCL (Primary Copy Locking) [139] (see Figure 2.1). An alternative to this kind of locking approach is MVCC, a powerful CCM concept, which is described in the next section.

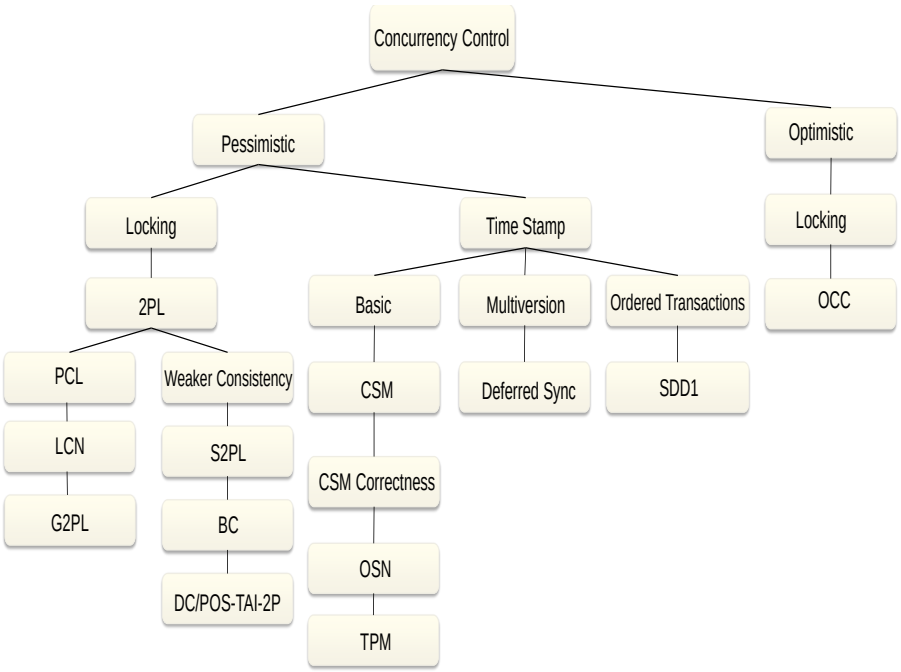


Figure 2.1: Overview of current CCM approaches for ensuring ACID for transactions (adopted from [139]). The approaches follow classical pessimistic (synchronized or merge based execution of transactions) or optimistic (direct execution of transactions and rollout in case of detected conflicts) approaches. Pessimistic approaches such as the two phase lock approaches based on 2PL, which allow only low data throughput, are widely used. In this thesis, a classical timestamp based approach to multiversion concurrency for hash maps is researched and pursued.

2.1.1 Multiversion Concurrency Control

MVCC is a commonly used CCM by DBMSs programming languages to implement transactional memory[2]. Since 1980, MVCC has been researched in various implementations and has already confirmed its efficiency and performance countless times [32, 34, 40, 46, 66, 67, 129, 143, 157, 158, 179, 181, 197, 198, 206, 208].

It is an alternative to locking, in which the DBMS provides each reading transaction the prior, unmodified version of data that is being modified by another active transaction. This allows readers to operate without acquiring locks, i.e. writing transactions do not block reading transactions, and readers do not block writers.

In detail, when MVCC needs to update an item of data, it will not overwrite the old data with new data, but instead marks the old data as obsolete and adds the newer version elsewhere. Thus, there are multiple versions stored, but only one is the latest. This allows readers to access the data that was there when they began reading, even if it was modified or deleted part way through by someone else. It also allows DBMSs to avoid the overhead of filling in holes in memory or disk structures but requires (generally) the system to periodically sweep through and delete the old, obsolete data objects.

MVCC provides point in time consistent views. Read transactions under MVCC typically use a timestamp or transaction ID to determine the latest version of every DBMS object. Read and write transactions are thus isolated from each other without any need for locking. Writes create a newer version, while concurrent reads access the older version.

In its most basic implementation, MVCC ensures a transaction (T) never has to wait to read a DBMS object (O) by maintaining several versions of the object. Each version of object O has both a read timestamp (RT) and a write timestamp (WT) which lets a particular transaction T_i read the most recent version of the object which precedes the transaction's read timestamp $RT(T_i)$. If transaction T_i wants to write to object O, and there is also another transaction T_k happening to the same object, $RT(T_i)$ must precede the $RT(T_k)$, i.e., $RT(T_i) < RT(T_k)$, for the object write operation to succeed. A write can not complete if there are other outstanding transactions with an earlier RT to the same object and are therefore either merged or sequentially applied.

The drawback of MVCC is the cost of storing multiple versions of objects in the DBMS. On the other hand, the access is never blocked, which can be important for workloads mostly involving reading values from the DBMS. MVCC is particularly adept at implementing true snapshot isolation, something which other methods of concurrency control frequently do either incompletely or with high performance costs.

MVCC is widely used in several popular DBMSs, such as ArangoDB, Berkely DB, CouchDB, IBM DB2 and Cognos, eXtremeDB, MemSQL, Microsoft SQL, MongoDB, MySQL, Oracle, PostgreSQL [21, 26, 70, 71, 135, 138, 161].

[2] Transactional memory uses transactions rather than locks to synchronise processes that execute in parallel and share memory. It uses designated code sections as transactions and guarantees for consistent execution of parallel transactions. It does this by monitoring their access to designated transactions variables. If the systems detects a conflict, it will initiate a rollback of the transaction, without noticing the user of the transactional memory.

2.2 Simulation based Multiobjective Optimization

The idea of SBO is to combine simulation models with an optimization component that varies certain variables of a simulation model to minimize or maximize a set of objective functions. Thus, SBO combines mathematical optimization techniques with simulation (analysis).

Simulation models are used to predict complex, real systems which can be further a subject to random influences. Typically, simulation models are used to examine the effects of individual configuration alternatives without actually realizing them and causing possible negative effects on the real system. The classical method for selecting an optimal configuration alternative for a given simulation model and simulation scenario is to carry out large simulation experiments and to make the selection manually.

SBO aims to make this selection of the optimal or "best" configuration alternatives automated by means of an algorithm using an underlying simulation model. The basic idea of SBO is to cleverly vary the configuration alternatives in order to approximate or compute the optimal configuration alternatives for the given scenario. State-of-the-art simulations increase steadily and rapidly in their complexity. Therefore, the underlying objective functions become difficult and expensive to evaluate.

Thus, it is vital for a SBO application that the simulation is executed very quickly and that the number of simulation executions is minimized. Even more, in many complex systems, objective functions can not be formulated.

In analogy to "classical" optimization, the simulation result, corresponds to the objective function of an optimization problem, and the variables of a simulation model correspond to the variables of an optimization model. A major difference is, however, that the objective function can be stochastic, which means that it is subject to random fluctuations, depending on which scenario is considered in a simulation run. Even more, objective functions can be completely unknown, which means that, for instance, no ordinary differential equations or partial derivatives ar given, leading to so-called blackbox simulations.

Specific SBO methods can be chosen based on the decision variable types (see Figure 2.2). Within this context, SBO exist in two fashions, either the optimization is dynamic per simulation state (optimization control) or static for the complete simulation run (parametric optimization). Both approaches aim at maximizing or minimizing a set of functions.

2.2.1 The Simulation based Optimization Process

A SBO study or process can be subdivided into three phases, including a preprocessing phase, an optimization phase and a post processing phase [16] (see Table 2.1).

The preprocessing phase plays a significant role in the success of the SBO study. In this phase, the most important task is the formulation of the optimization problem (resp. the definition of objective functions), if possible. This task is not trivial because it is very interdisciplinary: it lays directly between mathematics and the corresponding engineering field of the given simulation model. Therefore, it requires rich knowledge of mathematical optimization and domain knowledge. It is valuable to note that, within my virtual testbed concept, the simulation model should not be simplified too much in order to prevent the risk of simplification and/or inaccurate modeling of the problem. Conversely, such complicated models may severely delay the whole SBO study because the simulation requires more computational time.

In the optimization phase, the most important task of analysts is to monitor convergence of the optimization and to detect errors which may occur during the whole process.

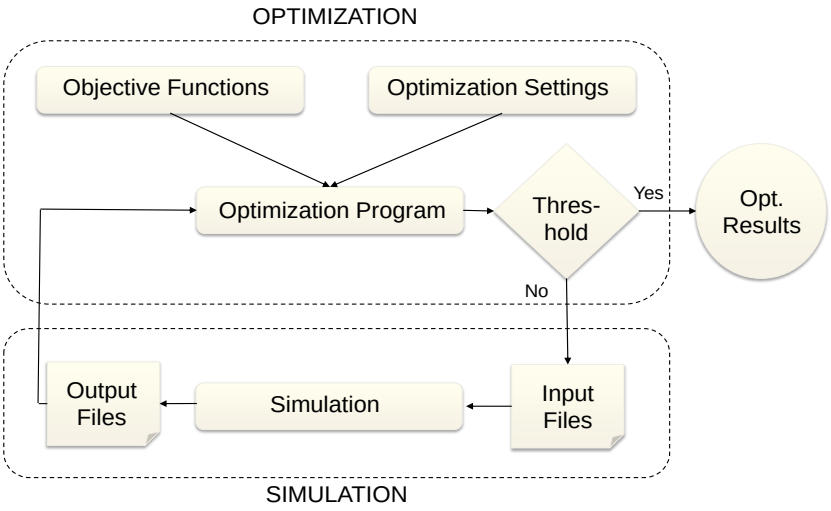


Figure 2.2: The coupling loop between simulation and optimization (adopted from [16]). In state-of-the-art approaches, objective functions are known and required. The simulation is parametrized, executed several times via an optimization algorithm and the objective functions are evaluated. This is done until a minimum or maximum value is reached. In contrast to these approaches, my work replaces this coupling loop by an approximation of the simulation model because the objective functions are unknown.

It is necessary to note that a convergent optimization process does not necessarily mean the global minimum (or minima) has been found. Convergence behaviors of different optimization algorithms are not trivial and are a crucial research area of computational mathematics [16]. Errors during the optimization process may rise from insolvable solution spaces, infeasible combination of variables, etc. Even more, a single simulation failure may crash the entire SBO process. These errors can be detected by monitoring the optimization progress, considering simulation time report, simulation data retrieval, convergence behavior or optimization termination criteria [16]. There are a great number of termination criteria which are mostly dependent on the used optimization algorithm, e.g. maximum optimization time, objective function convergence and approximation, population convergence, change of variables, step size reductions or maximum iterations. An optimization may have more than one termination criterion and the optimization process ends if at least one of these criteria is satisfied. The termination criteria must be set correctly unless the optimization will fail to converge to a stationary solution or result in useless evaluations, thereby extra optimization time. However, in sophisticated SBO applications, it is often impossible to identify whether a global optimum is reached by the optimization.

Phase	Major tasks
Pre-processing	Formulation of the optimization problem: - Building the simulation model - Setting objective functions and constraints - Selecting and setting independent (design) variables and constraints - Selecting an appropriate optimization algorithm and its settings for the problem - Coupling the optimization algorithm and building the simulation program - (Optional) Screening out unimportant variables by using sensitivity analysis to reduce the search space and increase efficiency of optimization[15, 24, 167] - (Optional) Creating a surrogate model (a simplified simulation model) to reduce computational cost of the optimization [24, 60, 172]
Running optimization	Monitoring convergence Controlling termination criteria Detecting errors or simulation failures
Post-processing	Interpreting optimization results - (Optional) Verification [10] and comparing optimization results of surrogate models and "real" models for reliability [24] - (Optional) Performing sensitivity analysis on the result [54] Presenting the results

Table 2.1: Major phases in simulation based optimization studies. Adopted from [16].

Nevertheless, even if the optimization results in a non-optimal solution, one may have obtained a better building performance compared to not running any optimization [16].

In the post processing phase, the analysts have to interpret optimization data into charts, diagrams or tables from which useful information of optimal solutions can be derived [16].

The simulation part of the SBO application can be implemented as a discrete event simulation or continuous simulation. A discrete event simulation models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation can directly jump in time from one event to the next. This contrasts with continuous simulation in which the simulation continuously tracks the system dynamics over time. Instead of being event based, this is called an activity based simulation; time is broken up into small time slices and the system state is updated according to the set of activities happening in the time slice. Because discrete-event simulations do not have to simulate every time slice, they can typically run much faster than the corresponding continuous simulation [186].

2.2.2 Multiobjective Optimization

Today, simulation models are dominated by a MOP because many real world problems involve decisions based on multiple and conflicting criteria [79] (see Section 1.1). Therefore, it is necessary survey the definition and constraints of MOPs. I define MOP according to [79]:

Given a subset X of $\mathbb{R}^n$ and p functions $f_j : X \to \mathbb{R}$ for $j = 1, 2, ..., p$, MOP is defined as:

$$(MOP) \max_{x \in X} F(x) = (f_1(x), f_2(x), ..., f_p(x)) \tag{2.1}$$

where $F : X \to \mathbb{R}^p$ is the objective function vector. I assume that X is of the form $X = \{x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n : a_i \leq x_i \leq b_i, i = 1, 2, ..., n\}$, where a_i and b_i are the lower and upper bound of the ith component of variable x, respectively. When the objective functions conflict with each other, no single solution can simultaneously minimize all scalar objective functions $f_j(x), j = 1, ..., p$. In these scenarios, the goal of MOP is to identify a subset of the Pareto optimal points ($\mathcal{P}*$) which are able to represent the Pareto front or to compute a single trade-off solution $x \in \mathcal{P}*$ (see Figures 2.3a and 2.3b) [153]. The definition of Pareto optimality can be provided by using Pareto dominance relation [2]:

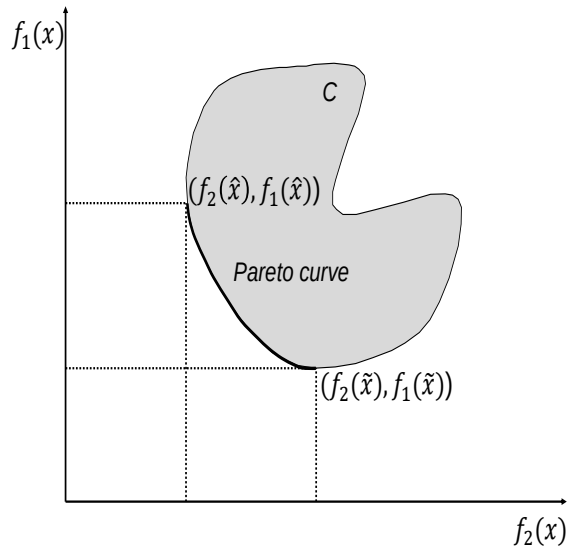
- Let $x_u, x_v \in X$ be two decision vectors (samples of X). $F(x_u)$ dominates $F(x_v)$ (denoted $F(x_u) \prec F(x_v)$) if and only if $f_i(x_u) \leq f_i(x_v)\ \forall i \in \{1, 2, ..., p\}$ and $f_j(x_u) < f_j(x_v)\ \exists j \in \{1, 2, ..., p\}$
- A point $x* \in X$ is globally Pareto optimal if and only if there is no $x \in X$ such that $F(x) \prec F(x*)$. Then, $F(x*)$ is called globally efficient. The image of the set of globally efficient points is called the Pareto front. In general, computational methods can not guarantee global Pareto optimality [82], but at best local Pareto optimality that is defined as:
- A point $x* \in X$ is locally Pareto optimal if and only if there exists an open neighborhood of $x*$, $B(X*)$, such that there is no $x \in B(x*) \cap X$ satisfying $F(x) \prec F(x*)$. $F(x*)$ is then called locally efficient. The image of the set of locally efficient points is called the local Pareto front.

In general, identifying the set of all Pareto optimality points is not a tractable problem and mostly impossible, particularly when the knowledge on the structure of the problem is very minimal or not available [79].

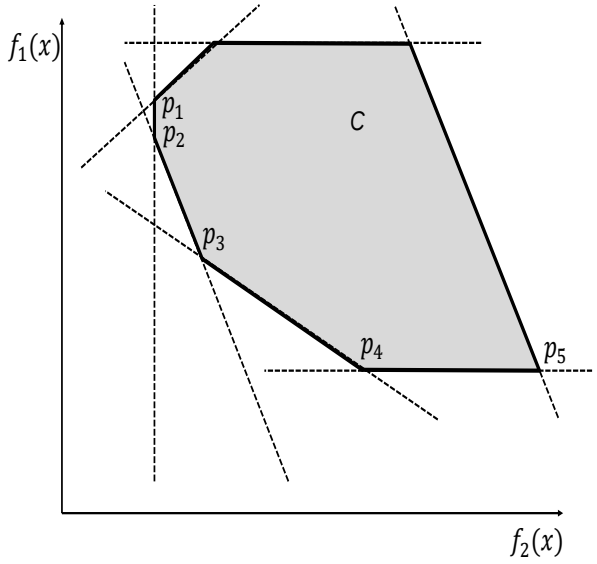
An example of a convex Pareto front is illustrated in Figure 2.3a. Here, all the points between $(f_2(\hat{x}), f_1(\hat{x}))$ and $(f_2(\tilde{x}), f_1(\tilde{x}))$ define the Pareto front. These points are called non-inferior or non-dominated points.

An example of local and global Pareto optima is illustrated in Figure 2.3b. The points p_1 and p_5 are local Pareto optima; points p_2, p_3 and p_4 are global Pareto optima.

There a numerous approaches in order to solve MOPs in order to find a single Pareto solution or to determine the Pareto front. Tables 2.2 and 2.3 give an overview of the mostly-used algorithms. Obviously, all algorithms have in common that they achieve better results (in terms of number of found solutions or distance to Pareto front), when the objective functions are known. In addition, they do not consider data mining and analysis based approaches which can provide viable heuristics [153, 156], thus, leading to better solutions of the optimization solvers [153].



2.3a: Example of a Pareto front (black) of the feasible design space. Adopted from [115]. This Pareto front yields all best trade-off solutions of the MOP. In this thesis, solutions shall be computed that are very close or directly at the Pareto front.



2.3b: Example of local (p_1, p_5) and global (p_2,p_3,p_4) Pareto solutions that reside on the Pareto front. Adopted from [115]. Normally, these global Pareto solutions are shown to the user who can then decide on a suitable trade-off.

Family	Strength and weakness	Typical algorithms
Direct search (including generalized pattern search methods)	-Derivate-free methods -Can be used if cost function has small discontinuities -Some algorithms can not exactly compute minimum -May be attracted by a local minimum -Coordinate search methods often have problems with non-smooth functions	Exhaustive search, Hooke-Jeeves, coordinate search, mesh adaptive search, generating set search, simplex algorithm
Integer Programming	Solving problems which consist of integer or mixed-integer variables	Branch and bound methods, simulated annealing, tabu search, hill climbing method
Gradient-based	-Fast convergence, a stationary point can be guaranteed -Sensitive to discontinuities in the cost function -Sensitive to multi-modal function	Bounded BFGS, Levenberg-Marquardt, discrete Armijo gradient, CONLIN method, etc.

Table 2.2: Classification of mostly-used optimization algorithms in SBO studies (1). Adopted from [16].

Family	Strength and weakness	Typical algorithms
Meta-heuristic methods	-Do not get stuck in local optima -Large number of cost function evaluations -Global minimum can not be guaranteed	Evolutionary optimization family: GA, genetic programming, differential evolution Swarm intelligence: Particle swarm optimization, ant and bee colony algorithm, intelligent water drop
Trajectory search	-Easy implementation even for complex problems -Appropriate for discrete optimization problems (continuous variables can also be used), e.g. travelling salesman problems -Only effective in discrete search spaces -Unable to tell whether the obtained solution is optimal or not -Problems of repeated annealing	Simulated annealing, tabu search, hill climbing
Other		Harmony search, firefly, invasive optimization
Hybrid	Combining the strength and limiting the weakness of above mentioned approaches	PSO-HJ, GA-GPS, CMA-ES/HDE, HS-BFGS

Table 2.3: Classification of mostly-used optimization algorithms in SBO studies (2). Adopted from [16].

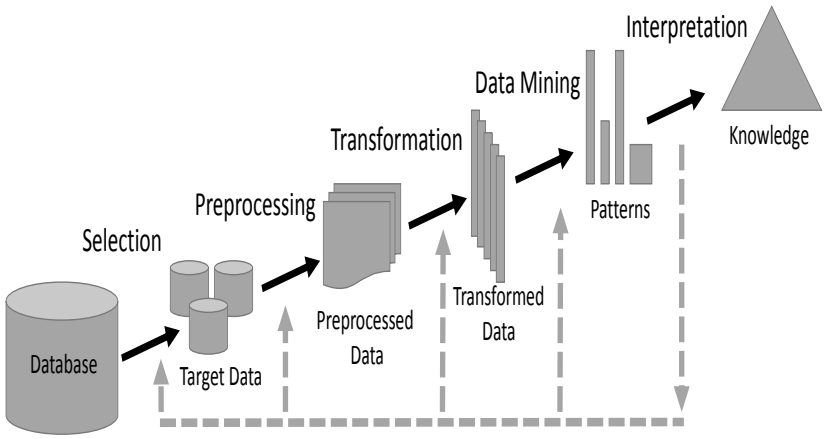


Figure 2.4: The traditional KDD process (adopted from [200]): low level data is extracted from a database and after preprocessing and transformation of the data, data mining methods generate specific representations and patterns of the data. Extensive manual evaluation of these representations leads finally to low level data knowledge. In this thesis, a novel automatic KDD for virtual testbeds is presented.

2.3 Knowledge Discovery Processes and Data Mining

KDD is motivated by the trend that arbitrary data is being collected and accumulated at a dramatic pace across a wide variety of research and industrial fields. There is an urgent need for a new generation of computational theories and tools to assist humans in extracting useful information (knowledge) from the rapidly growing volumes of digital data. These theories and tools are the subject of the emerging field of KDD [200]. Originally, KDD is defined as making sense of data collections that are too big to manually review each and every single record. Input sources for such kinds of data are complex simulations, graphs, or data warehouses [203]. [200] describe the KDD process as multiple steps to ultimately transform low level data into useful knowledge (see Figure 2.4). This knowledge might be more compact (for example, a short report), more abstract (for example, a descriptive approximation or model of the process that generated the data), or more useful (for example, a predictive model for estimating the value of future cases) [200]. Consequently, KDD deals with the non-trivial task of identifying valid, novel, potentially useful, and ultimately understandable patterns in data by the application of specific data mining methods for pattern discovery and extraction [200]. The actual combination of diverse data mining methods in order to obtain above illustrated knowledge for a specific use case is summarized as a KDP.

In detail, the KDP is usually a highly interactive five-step-process that requires many decisions made by the user. Some of these steps (e.g. target data selection or interpretation of patterns) have to be iteratively repeated by the user for convincing results. Hence, the KDP is usually semi-automatic because the user is ultimately responsible for interpretation and evaluation of mining results. This particularly applies for the evaluation of the usefulness of the generated knowledge [200].

Generally, a KDP can be separated into five steps:

1. **Selection** In the first step, the target data set (e.g. selecting a data set or focusing on a subset of variables) has to be defined on which the necessary discovery is to be performed.

2. **Preprocessing** In the second step, the target data is cleaned and preprocessed: removing noise if appropriate, collecting the necessary information to model or account for noise, deciding on strategies for handling missing data fields, and accounting for time-sequence information and known changes.

3. **Transformation** In the third step, the preprocessed data is transformed (e.g. reduced) for finding useful features to represent the data depending on the goal of the task. With dimensionality reduction or transformation methods, the effective number of variables under consideration can be reduced, or invariant representations for the data can be found.

4. **Data Mining** In the fourth step, the goals of the KDP are matched to a set of data-mining methods, e.g. classification, regression or clustering, as described in the next section, and patterns are extracted.

5. **Interpretation** In the last step, the mined patterns are interpreted. This step can also involve visualization of the extracted patterns and models or visualization of the data given the extracted models. This step can lead to another iteration of the KDP until the mined patterns yield useful results.

The most interesting and challenging step is the data mining, which is further outlined in the next section.

2.3.1 Data Mining

Data mining is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and DBMSs. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Aside from the raw analysis step, it involves DBMS aspects, data pre-processing, model and inference considerations, metrics, complexity considerations, post-processing of discovered structures, visualization, and online updating [200]. Within the above mentioned KDP process, data mining methods are chosen based on the KDP goals.

Generally, these goals can be differentiated into verification and discovery. With verification, a KDP is used to verify a users' hypothesis while for discovery, the KDP has to find new patterns. The discovery goal can be further subdivided into prediction, where the KDP has to find patterns for predicting future behavior of some entities, and into description, where the KDP has to find human-understandable representations for unknown patterns. However, the boundaries between prediction and description are not sharp (some of the predictive models can be descriptive, to the degree that they are understandable, and vice versa) but the distinction is useful for understanding the overall discovery goal.

Data mining involves six common classes of tasks (see Figure 2.5):

- **Anomaly detection (outlier/change/deviation detection)** is the identification of unusual data records, that might be interesting or data errors that require further investigation.

- **Association rule learning (dependency modelling)** searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.

- **Clustering** is the task of discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data. Examples of clustering applications in a knowledge discovery context include discovering homogeneous sub-populations for consumers in marketing DBMSs, identifying subcategories of spectra from infrared sky measurements [146] or the classification of cost-drivers in air fleet management [132].

- **Classification** is the task of generalizing known structure to apply to new data. Examples of classification methods used as a part of knowledge discovery applications include the classifying of trends in financial markets [27], the automated identification of objects of interest in large images DBMSs [202], or the classification of "legitimate" or "spam" mails in e-mail programms.

- **Regression** attempts to find a function which models the data with the least error. Regression has many applications, for instance, estimating cost drivers in airfleet management [132].

- **Summarization** provides a more compact representation of the data set, including visualization and report generation. A simple example would be tabulating the mean and standard deviations for all fields. More sophisticated methods involve the derivation of summary rules [162], multivariate visualization techniques, and the discovery of functional relationships between variables [171]. Summarization techniques are often applied to interactive exploratory data analysis and automated report generation.

Every data mining algorithm (which consists of a set of above classes) contains three primary components: model representation (data structure used to describe the discoverable patterns), model evaluation (quantitative statements or fit functions of how well a particular pattern meets the goals of the KDP process) and search (once the model representation and the model-evaluation criteria are fixed, then the data mining problem has been reduced to purely an optimization task: find the parameters and models that optimize the evaluation criteria) [200].

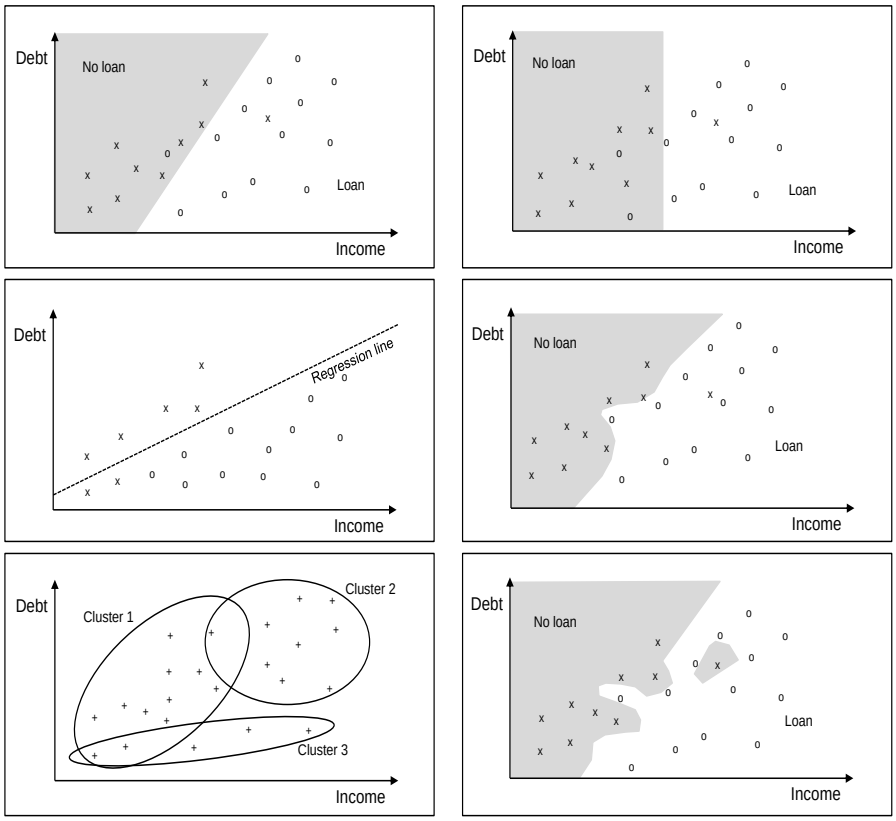


Figure 2.5: Typical examples of data mining algorithms (left: linear classification, linear regression and gaussian clustering; right: non-linear classification and nearest-neighbor classification) for a simple data set with two classes. Adopted from [200].

Part II

Wait-Free Data and Concurrency Management Enabling Massively Parallel Virtual Testbeds

Nothing is worth more than this day.

Johann Wolfgang von Goethe

3

Wait-Free Data and Concurrency Management

In this chapter, I introduce my novel wait-free data and concurrency management based on hash maps. I recall some of the most relevant research articles that have appeared in the international literature related to this topic and emphasize my contributions.

A central part of RISs, such as CVEs, VEs, VR systems and 3DST based simulations, is the generation, management, and distribution of all relevant data (resp. simulation) states. In modern manifestations of these systems, usually many independent, inhomogeneous[1] software components need to communicate and exchange data in order to simulate the given model as well as to provide data for the visual feedback [150, 151, 154]. In detail, such systems usually consist of many different components such as graphics rendering, sound, several input devices, haptic rendering, physically-based simulation, model behavior, etc. All these components have to share and communicate some kind of data.

For instance, the physically-based simulation gathers data from the input devices and passes its results to the graphics, haptic and sound rendering. This requires some kind of interface for the data exchange between the components. This data can be extremely large, e.g. think about a spacecraft simulation in the asteroid main belt [14, 149] (see Section 3.6), where the position of thousands of asteroids changes continuously by Newtons' laws of motion. All transformations have to be passed from the simulation to the rendering component. This data exchange is usually done concurrently in highly parallel manner in order to preserve a fast simulation and immersive visual feedback to the user. Therefore, current RISs rely on some kind of data management or structure which is concurrently shared between all software components. Thus, RISs, in general, require a data and concurrency management that is easy to handle and guarantees fast access to data for both, reading and writing while maintaining a consistent simulation state even in heavily concurrent access scenarios [126]. In order to achieve above objectives, the problem can be tackled from two different perspectives. It can be explored either by traditional VR, VE, RIS engineering research or by current virtual testbed (resp. 3DST) research.

[1] This terminology encompasses differences in type (graphics, physics, haptics, sound, input, craft, and many more) and frequency of the software component.

Figure 3.1: Collaborative virtual assembly of cars as an example of RIS consistency (interaction and changes of continuously changing model), scalability (amount of users) and responsiveness (rendering of the continuously changing assembly) [22].

In the traditional VR based research, RISs, i.e. standard VR approaches, describe and encode VE in the classic fields-and-routes based dataflow paradigm, as for instance specified in VRML and X3D [150]. In this paradigm, "wires" are implemented between the in- and output fields of the components and the data is routed through these wires. Consequently, the number of interfaces grows quadratically with the number of components. This is manageable, as long as the number of components is relatively small. However, modern VR systems or VEs (or above mentioned related technologies) are often not restricted to a single user or software component but may contain thousands of users or software components which synchronously or asynchronously access the shared global data state. Additionally, adding new components to the system requires changes to the interfaces of many other components and the implementation of new routes. This reduces the maintainability significantly and can affect the performance of the overall system negatively. A second major challenge for modern VR systems is the data consistency: for instance, in multi-user VEs, all users interact simultaneously but the system has to provide a consistent view on the VE to all users.

This problem also arises in multi-threaded single-user VEs, for instance when a haptic rendering thread runs at 1000 Hz and the graphical rendering requires only 30Hz. As a result, RISs require a method to synchronize the data exchange between the components. This required massively parallel software architecture can be obtained by design patterns or toolkits for high performance multiprocessing for RIS applications such as [86]. These classic VR systems often use locks on shared data to avoid race conditions and barriers for data synchronization. The approaches presented in this part of my thesis are not directly derived from these concepts but can be used to implement wait-free concurrency in such architectures.

Unfortunately, locking approaches decrease the performance of the whole system because components have to wait until all other components have finished their operations. Consequently, it is extremely complicated to guarantee time-critical access to data for the components (see Chapter 2.1). In case of haptics this may result in poor immersion or even damage of the expensive devices. Furthermore, modern CVEs like large-scale virtual cities or open space training have increased in popularity tremendously over the past years.

All of these applications have in common that a large number of users or components interact simultaneously in real-time in a shared virtual world. Interaction usually means that either users can manipulate objects or that software components algorithmically change data in the VE. In order to maintain a common and consistent state of the CVE for all users, interactions made by one user have to be made visible to all other users immediately. This requires a high responsiveness of the system, i.e. system changes have to be distributed with low-latency. Actually, experiments have shown that a bad responsiveness (high-latency) can lead to frustrated participant experience [51] and even to users completely losing interest in the application [4]. However, distributing shared data with low-latency is not enough to keep a shared VE plausible and fair.

A second challenge is the consistency of the system. This challenge is essential if several users or software components are allowed to manipulate the same object simultaneously, e.g. in above mentioned open space training scenarios, when a group of users jointly solve tasks, or in collaborative virtual sculpting tools [63]. Here, interactions of one user directly influence the simultaneous interactions of the other users. Finally, the demand for larger CVEs, i.e. VEs that allow a higher number of participants, more artificial intelligence components, or more interactive objects, increased significantly during the past years. Consequently, modern CVEs (and above related technologies) should allow a high scalability in order to be prepared for today but also future applications. Figure 3.1 illustrates these key requirements for a virtual assembly scenario.

Obviously, these three key requirements responsiveness, consistency and scalability are not independent of each other and apply for any RIS. For instance, low-latency is the prerequisite for the consistency while higher scalability is often contradictory to high responsiveness (see Figure 3.3). Usually, this functionality of handling all simultaneous user an software component interactions and allowing access to the common parts of the shared VE is implemented within a CCM. This CCM enables and maintains parallel, dynamic behavimy of the CVEs shared world state. A perfect CCM should fulfil all three partly conflicting requirements. Most current CCMs for VR based applications use a simple locking mechanism for simultaneous access to shared objects: if a component wants to manipulate an object, it locks the object, manipulates it, and when it has finished his manipulation, it releases it. This mechanism guarantees a consistent system and avoids race conditions; but if many components try to access the same object, it results in a bad responsiveness because other users have to wait until they gain access to the object, and it limits the scalability. Actually, locking mechanisms serialize concurrent user access, hence, they are only limited CCMs.

Virtual testbed research or 3DST applications tackle above described problem differently. The goal of these applications is usually to simulate a given model and to provide the users visual feedback, most often in real-time. Like in above research fields, many independent inhomogeneous software components need to communicate and exchange data in order to simulate the model as well as to provide data for the visual feedback [150, 154]. Again, this data exchange is usually done concurrently in highly parallel manner in order to preserve a fast simulation and immersive visual feedback to the user. In state-of-the-art virtual testbed research, relational databases are often used for the task of data management and distribution [91, 126] (see Figure 3.2). They are well-researched, easy-to-use and deliver out-of-the-box functionality for a consistent data management. Unfortunately, they also have some drawbacks when considering 3DST applications. For instance, they do not scale well to massively parallel access due to their inherent serialization of access queries.

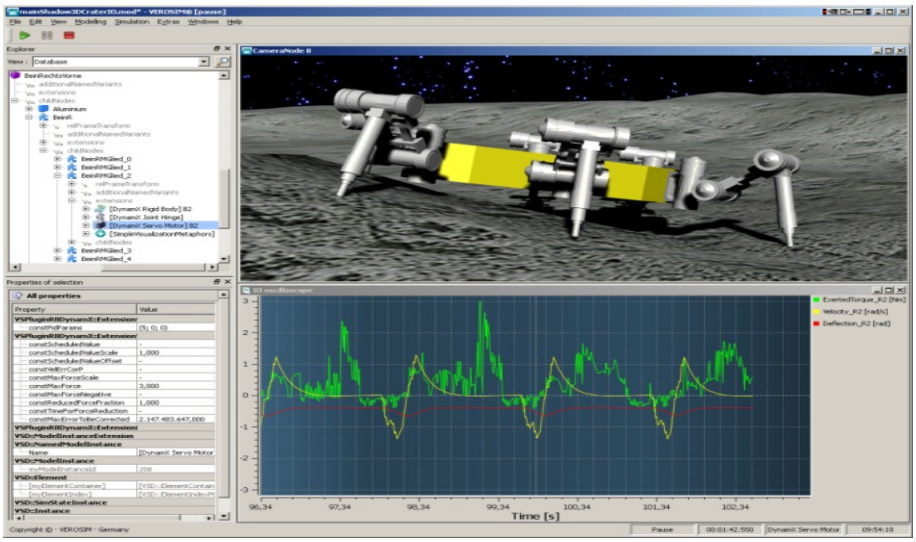


Figure 3.2: Example of a virtual testbed based on relational database technology: the virtual crater testbed [207].

Moreover, the relational data model requires a strict definition of a schema (consisting of tables with the defined data fields in row-column format) prior to storing any data. This constraints typical simulation engineering tasks such as capturing new simulation data which was previously not considered or introducing simulation behavior changes due to new data formats and content.

Finally, simulation application developers usually use object-oriented programming languages to build 3DST applications as handling object-oriented data is nowadays most efficient. In contrast to this, the data needs to be collected from many tables (often hundreds or thousands in todays' simulation applications) and combined before it can be provided to the application. Similarly, when writing data, the write access needs to be coordinated, separated and performed on many tables [38]. This results in a fundamental mismatch between the way a simulation application would like to see its data and the way it's actually stored in a relational database.

My contribution for above research field perspectives is a novel centralized CCM and data management that manages concurrent multi-threaded access to shared data in any RIS, such as CVEs, VEs, massively parallel 3DST applications or VR systems.

The main idea is to implement a wait-free CCM based on centralized hash maps which implement MVCC and support the ACID properties. My approach contributes many benefits. It reduces the number of interfaces from $O(n^2)$ to $O(n)$ which benefits better maintainability and lower synchronisation overhead. Even more, it enables wait-free read and write operations. It is based on my novel wait-free hash map data structure which does not introduce unwanted side effects such as possible deadlocks or thread starvation. Additionally, this wait-free access allows high performance access even for massive numbers of concurrent read and write operations. Within above concept, my hash map stores static and dynamic parts of (for instance a simulation model), distributes changes caused by the simulation and logs the simulation run. Here, I use my novel wait-free hash map techniques in graph-based schema-less, in-memory resident manner in order to store object-oriented content. As a result, the time-consuming serialization as well as table-based coordination and separation of relational databases are eliminated. Additionally, my approach implements the same functionality as state-of-the-art relational databases such as aggregate queries and caching strategies. As a consequence, my approach overcomes the associated drawbacks of relational database technology for sophisticated 3DST applications.

Additionally, my approach has several advantages compared to other state-of-the-art decentralized methods, such as persistence for simulation state over time, object identification, standardized interfaces for software components as well as a consistent world model for the overall simulation system.

At last, my data structure incorporates a versioning mechanism which generates a queryable archive of the complete simulation. As a result, simulation components can be used in an online viewing mode to replay a simulation run step by step, allowing analysis and debriefing. Therefore, my approach fulfils all todays' needs for data and thread management in virtual testbeds.

Concluding, my approach satisfies the three main requirements of data managements for RIS and 3DST applications. It incorporates a highly responsive low-latency data access for any number of simulation components. It achieves high scalability because the wait-free access for all simulation components does not require high coupling. The dependencies between the components are managed via unique keys which define the component communication. My implementation further avoids data inconsistencies based on a few atomic operations for this type of wait-free access.

3.1 Related Work

In the following, I introduce related work from the domains of VR systems and 3DST applications. Both domains consider and treat the problem of data management for RISs and present different approaches.

3.1.1 Data Management in Virtual Reality Systems

When introducing any data management concept, such as my proposed centralized approach, for any RIS, one immediately encounters the well-known problem of concurrent shared data structures and race conditions (see Section 2.1). Consequently, a CCM has to be implemented which manages the access to the centralized data management of all software components. A distinctive characterization of CCMs is whether they are locking or non-locking (see Chapter 2.1). In this section, I investigate these concepts closer from the RIS engineering point of view, outlining the current approaches for VR and CVE applications.

Locking approaches usually allocate resources exclusively by using various well-studied techniques such as mutexes, semaphores or condition variables. Concurrent threads have to wait until a resource has been released. This may result in a loss of efficiency and parallelization or even deadlocks.

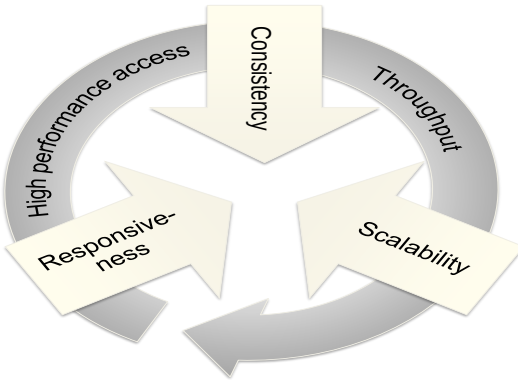


Figure 3.3: All presented CCM approaches of RIS related research try to achieve three main requirements (scalability, consistency and responsiveness) while providing high performance access. These requirements are contrary to each other (for instance, high scalability with many components leads to poor responsiveness in traditional locl-based approaches). In the context of this work, these requirements are considered equally from the aspect of data throughput for high performance access.

However, a main advantage of locking CCMs is that they avoid race conditions and naturally guarantee consistency of the system. Many scenegraph systems like OpenSG or toolkits such as IRIS [86] use blocking mechanisms for synchronization. Most classic CCMs like [28] relied directly on standard locking approaches. Unfortunately, [201] reported that the locking approach scales only to at most ten peers on a local area network. This is mainly because of the problem that concurrent threads have to wait until a resource has been released. This may result in a loss of efficiency because problems like thread starvation or deadlocks can occur. Consequently, more modern CCMs like [63, 142, 145] tried to avoid this problem by extending the basic locking mechanism. [145] used a simple first-come-first-serve locking, in which a central server granted manipulating access to a user on request. All other participants could only work on a local copy but their local changes were not transmitted to the server site. Hence, only one user of the CVE could really interact whereas all other users could only observe the changes. [63] further presents a lock-based approach for the special case of collaborative sculpting. They split a mesh into different regions. For each region, a lock could be acquired. This allowed several users to work in parallel on the same mesh but at different parts. However, only one user could modify a region at the same time, multiple access was again not possible due to the lack of an suitable algorithm which could solve for parallel access. Additionally, the approach does not scale well with the number of users because the lock acquirement is slow. Filtering approaches basically offer a more general approach such as [45, 95].

The main idea is to reduce the acquirement latency by restricting the number of users which can request a lock. To do that, constraints have to be defined that are used to filter the requests. Even if the basic idea is generic, the constraints have to be adjusted for each individual application. Moreover, defining constraints which can not be met by all users simultaneously is challenging if not impossible in CVEs. Therefore, these filtering constraints are not generally valid for all kind of CVE applications. They also also do not solve the inherent problem of lock acquirement latency. Other approaches for collaboration in VEs or generic VR system architectures (e.g. [19, 51, 62, 192]) neglect the problem of efficient data access to a shared world state.

Non-blocking approaches avoid this exclusive allocation of resources by introducing very smart designs mostly using a few atomic operations [44, 123, 131]. These atomic operations, like compare and swap (CAS), are usually directly supported by the processor. Consequently, non-blocking data structures avoid inconsistencies and deadlocks. Non-locking approaches can be further classified into lock-free and wait-free methods. Both approaches avoid locks when solving concurrent access. Lock-free approaches guarantee progress of at least one of the threads accessing the shared data structure. All other threads can be arbitrarily delayed. In most approaches, all reading operations can happen in wait-free manner whereas all writers are delayed. Unfortunately, this can lead to thread starvation of the writers. Today, there exist efficient non-blocking implementations for almost any common data structure [122, 123, 176]. However, due to the restriction to processor-supported primitive data types that allow atomic operations, they can be hardly extended to RISs, such as CVEs, VEs, VR systems and 3DST based simulations, that require more complex data structures, e.g. matrices to store transformations. Additionally, memory management has to be taken into account: the design has to ensure that under no circumstances memory is freed, which is still in use by a concurrent thread. Actually, non-blocking approaches can be further classified into lock-free and wait-free methods. Lock-free approaches do not use any locks and guarantee progress of at least one of the threads accessing the shared data structure. Lock-free approaches incorporate that some threads can be delayed arbitrarily, in most cases the producer, which waits until every reader (which is in most cases wait-free) has finished its operations on the shared data structure.

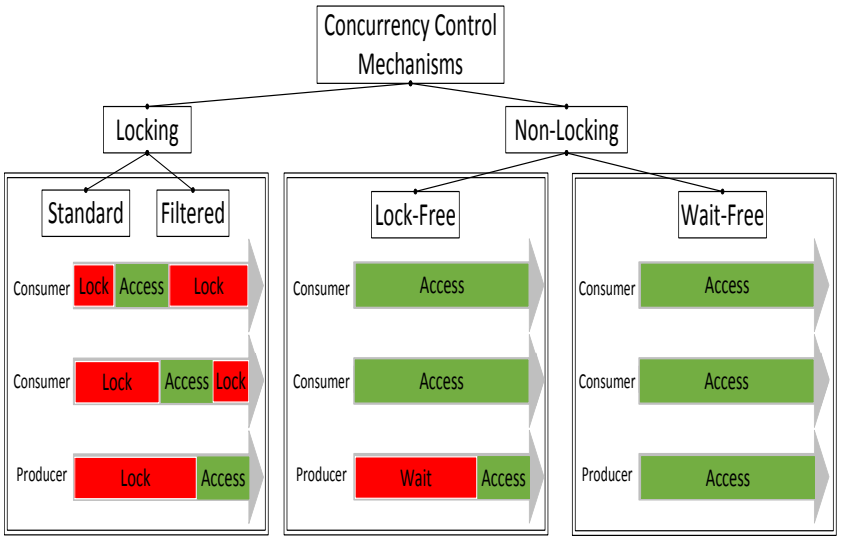


Figure 3.4: Classification of CCMs. Left: traditional locking approaches can delay threads arbitrarily. Middle: lock-free approaches that only delay the producer. Right: my novel wait-free approaches that do not delay other threads at all.

This approach is vulnerable for dead-locking the producer, however, statistically all threads will make progress [122].

Wait-free approaches guarantee each thread access to the shared data structure in a finite number of steps, regardless of other threads accessing the shared data structure [122]. Wait-free approaches have been developed, for instance, for queues [160] or linked lists [189]. Some lock-free solutions have also addressed this challenge, including per-thread timestamps [5, 195], reference counters [8], expensive CAS approaches [44] or global pointers, such as [123, 131].

With regard to the overhead needed for synchronizing the access of threads, wait-free approaches offer the least overhead, while lock-free mechanisms incur more overhead, and lock-based mechanism the most. Wait-free approaches additionally support different thread cycle times because no thread is blocked by another, promising high performance when asynchronously accessing a shared data structure in parallel. Such kind of asynchronous, parallel access most often occurs in RIS (see Chapter 2.1). Some of the above stated approaches had been compared by Hart [191]. Hart summarizes, that the reclamation overhead of non-blocking schemes can dominate the overall execution time of these approaches, decreasing the performance boost with respect to traditional blocking approaches. Hart also concludes, that for accessing single data sets a pointer[2] based approaches (for instance, the hazard pointer scheme [131]) perform very well, except when these data sets have to be traversed. He further concludes, that it is desirable to create a pointer inspired scheme that avoids per-element atomic instructions. Such a wait-free approach, based on pointers, could guarantee access to the shared data structure in a finite number of steps for each thread, regardless of other threads accessing the shared data structure [122]. Unfortunately, most of them are restricted to a single writer, as described above. However, RISs and 3DST applications require, simultaneous writing and readings operations are essential in order to achieve performance requirements (see Chapter 2). Figure 3.4 illustrates the classification CCMs for locking and non-locking approaches.

[2] The concept of a pointer denotes here a identifier for a memory block, e.g. a standard pointer from the programming lanuage C++.

3.1.2 Data Management in Virtual Testbeds

The subject of data management was also investigated more extensively in the research area of 3DST applications, namely virtual testbeds. Here, the combination of database technology, simulation and rendering methodology has attracted increasing interest in the last decade because databases have been integrated into virtual testbeds in many different ways. Though many attempts have been made to incorporate database technology into 3DST systems, to my knowledge, no one has used in-memory schema-less technology with wait-free access behavior before.

State-of-the-art research in the integration of database technology into 3DST systems use standard full-fledged SQL databases because they are easy-to-use and deliver out-of-the-box functionality for a consistent data management. [91, 126] introduced schema and data synchronization for distributed 3DST systems with a versioning interface. In more basic applications, databases have been used to store additional meta-information (e.g. about scene objects [23, 196]). More sophisticated approaches use the database to store the scene data itself [196], where some do support collaboration [37, 57, 100, 177, 205] while others do not [18, 52]. A flexible support for different data schemata is not widespread among these systems [37, 52, 205]. The simplest realizations allow schema alteration by adding attributes to generic base objects [68]. The more advanced systems support different static [80] or dynamic [52] schemata. However, these data management approaches can only alter their relational table schema based on a new schema delivered by another simulation architecture component (e.g. a simulation server). Consequently, this schema alteration is done manually by hand and is only distributed automatically. In all applications, the table schema alteration is complex and computationally expensive.

To summarize, the above mentioned related studies were focussed on combining traditional relational databases with simulation technology. However, traditional database technology has three main technical limitations:

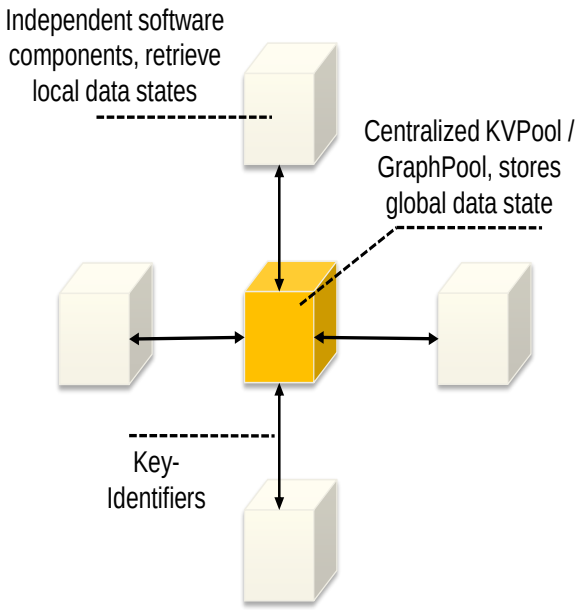
- the adaptability to object-oriented data due to rigid table-based schema,
- the scalability to massive amounts of components accessing the database in real-time manner,
- the performance with respect to massively parallel read and write operations due to serialization of access queries.

The database research community established in-memory resident databases and the not only SQL (NoSQL) methodology to compensate for these technical limitations shared by the majority of relational database implementations. NoSQL started out as industry developments in companies such as Amazon, Google, Twitter or Facebook which discovered these serious limitations of relational database technology [64].

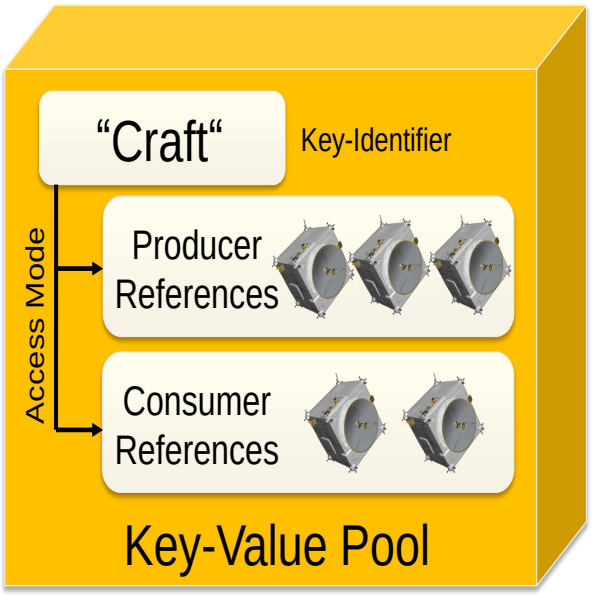
In order to overcome these limitations, database architects had sacrificed many of the most central aspects of relational databases, such as joins and fully consistent data, while introducing many complex and fragile pieces into the operations puzzle. They simplified the database schema and introduced various query caching layers. Finally, schema devolved from many interrelated fully expressed tables to something much more like a simple key/value look-up in an attempt to address these new requirements [38].

Relational and NoSQL data models are very different. The relational model takes data and separates it into many interrelated tables consisting of rows and columns. These tables reference each other through foreign keys that are stored in columns as well. Every piece of data is then stored only once in one table.

Consequently, the relational model minimizes the amount of storage space required, which was a key requirement when relational database were created due to expensive hardware [38].



3.5a: In my centralized data management approach, software components communicate via unique keys. The centralized hash map contains the global data state (every data that has to be shared within the application). Each software retrieves from this global state a temporary local state that is used for computation. The access is modelled via a key look-up.



3.5b: The access to the global data state is implemented with a unique key and access mode. The unique key refers to the required data and the access mode defines which version of the data should be returned. The data is stored in two versions in order to implement the proposed wait-free access: for producing and consuming operations.

However, space efficiency comes at expense of increased complexity when inserting and looking up data. Developers generally use object-oriented programming languages to build 3DST systems as handling object-oriented data is nowadays most efficient. In contrast to this, the data needs to be collected from many tables (often hundreds or thousands in todays' simulation applications) and combined before it can be provided to the application. Similarly, when writing data, the write needs to be coordinated, separated and performed on many tables [38]. Consequently, a fundamental mismatch exists between the way a simulation application would like to see its data and the way it's actually stored in a relational database.

Another major difference is that relational technologies have rigid schemas while NoSQL models are schema-less [38]. The relational data model requires a strict definition of a schema (consisting of all tables with the defined data fields in row-column format) prior to storing any data. This requirement makes typical simulation engineering tasks such as capturing new simulation data which was previously not considered or introducing simulation behavior changes due to new data formats and content extremely disruptive and frequently avoided.

This is the exact opposite of the desired behavior in the area of simulation and modelling, where developers need to rapidly, and constantly, incorporate new types of data to enrich their simulation models and applications. In comparison, schema-less databases allow the format of the data being inserted or changed at any time, without application disruption [38].

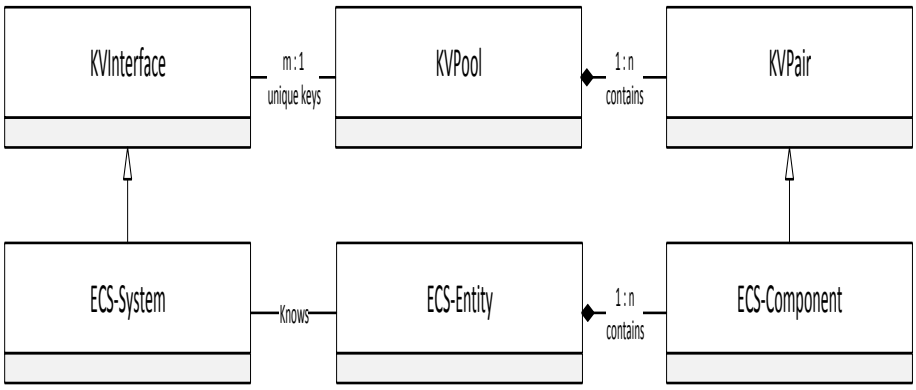


Figure 3.6: Integration of my wait-free hash map based data management into the ECS pattern only requires two abstract software classes: the KVInterfaces encapsulates the presented wait-free access via producer and consumer version based on copy on write (COW). The abstract KVPair encapsulates the value that should be stored in the key value pool (KeyValuePool). Both classes are realized by the ECS architectural concepts by implementing the actual behavior of the System and by defining member variables of the Components within the key value pair (KeyValuePair) concept.

3.2 Hash Map based Data Management

In the following chapters, I use the notations KeyValuePool and `KVPool` in order to differentiate between the concept and the actual software implementation, in order to avoid confusion.

The core of my data management and concurrency control is a wait-free hash map which I denote as `KVPool` (KeyValuePool). It stores all data that can be accessed concurrently by all software components which I define within the ECS pattern (see Figure 3.6 and Section 4.2). In general, the proposed hash map based data management implements a MVCC (see Section 2.1.1) within my centralized `KVPool` concept for the ECS pattern and solves for ACID. (see Figures 3.5a and 3.5b).

The shared data is stored in the pool in form of `KVPairs` (KeyValuePairs). Basically, the key of such a KeyValuePair. is the identifier for the `Systems`. There is no global main loop required; each `System` can access the data, i.e. read or write, at any point in time.

The main challenge is to avoid inconsistencies, as my CCM has to ensure that no data that is currently read by an `System` will be overwritten by another `System` that concurrently writes the data. Moreover, all these access operations are be performed without the necessity to wait for any `System` within my novel wait-free CCM. This CCM, the KeyValuePool itself, is implemented in three manifestations, each handling and tackling different constraints and challenges. In general, all manifestations are based on a COW mechanism which implements a double-buffering of KeyValuePairs, guarded by atomic operations and a delay of memory de-allocation. This double-buffering implements a MVCC - for each write operation a clone is generated.

Within my proposed CCM, I differentiate between consumers `Systems` that just read the data, e.g. the rendering thread that reads the transformations of the 3D objects in the scene, and producers `Systems` that are allowed to write data, like the physically-based simulation that changes the transformations.

These producer and consumer `Systems` retrieve the `Components` of each `Entity`, which is stored in the corresponding KeyValuePool manifestation of the hash map. These different manifestations of my hash map based data management are introduced in the following sections.

Here, the Systems retrieve a local simulation state (LSS) for their computations, while the consistent global simulation state (GSS) is stored in the hash map. The LSSs, copies of key-related sub-parts of the GSS, allow the implementation of my wait-free MVCC.

The corresponding wait-free read and write behavior for the LSSs is achieved via my global atomic marker (GAM) or local atomic marker (LAM) concepts and is described in the following sections.

In general, a major advantage of my global KeyValuePool approach is that it reduces the many-to-many interface of classic approaches to a simple many-to-one interface. One of the standard approaches to describe and encode VR systems is the classic fields-and-routes-based dataflow paradigm, as for instance specified in VRML and X3D. In that paradigm, one has to draw "wires" between the in- and output fields of the components and route the data through these wires. Consequently, the number of interfaces grows quadratically with the number of components. This is manageable, as long as the number of components is relatively small. However, modern RISs are often not restricted to a single user or software component but may contain an arbitrary number of them. In addition, adding new components to the system requires changes to the interfaces of many other components and the drawing of new routes. This reduces the maintainability significantly and can affect the performance of the overall system negatively.

In contrast to this approach, my basic idea is very simple: first, I identify all kinds of data that need to be shared between different components, e.g. the transformations of the objects in the scene. Instead of drawing a quadratic number of routes between the software components, I assign a single unique KeyValuePair to each data packet. I register the *key* of the KeyValuePair to my global KeyValuePool and reserve memory for the data. My global KeyValuePool holds the complete shared GSS of the system. If any System wants to access the data, it simply has to look up the *key* from the Component. Note that I actually do not require a full-fledged SQL database. Instead I rely purely on very fast hash maps.

Moreover, adding new interfaces between existing software components or adding new software components (resp. Systems) to the application, is extremely simple and does not necessarily require the introduction and implementation of more interfaces. I just introduce a new KeyValuePair to the KeyValuePool (see Figure 3.7).

Overall, I get a highly adaptable wait-free system for massively-parallel access in ECS based RIS implementations. Obviously, my approach does not directly rely on the ECS pattern and can also be used in any other pattern. However, in combination with the ECS pattern we directly C&C by the KVInterface encapsulation and the corresponding constraint on the centralized data access.

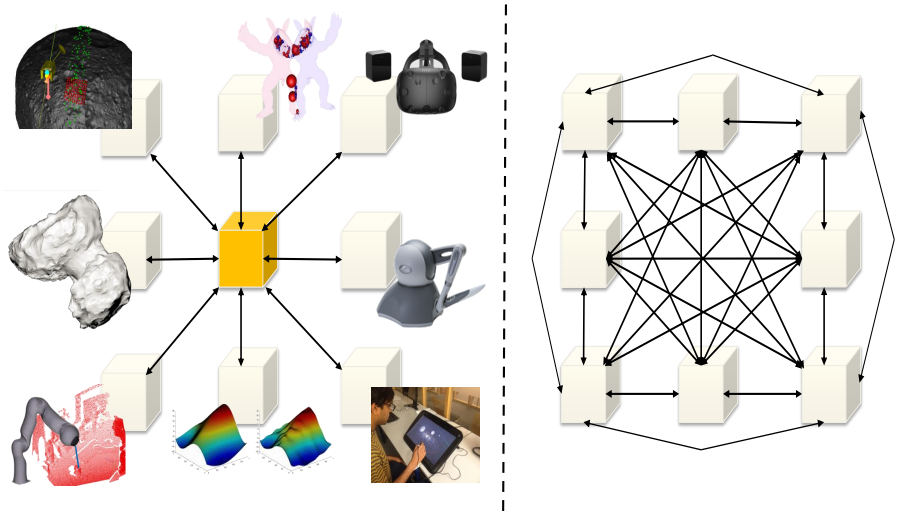


Figure 3.7: Comparison of my approach (left) to the standard approach (right). Standard approaches (such as the fields-and-routes based dataflow) for data exchange and communication incur a quadratic number of communication interfaces. My approach reduces this significantly because its a centralized approach based on key identifiers. This improves the maintainability of the overall architecture directly because the complete data- and workflow is modelled with a minimum number of interfaces and these interfaces can be implemented very efficiently by using a small set of key exchanges.

3.3 Global Atomic Markers: Single Producer, Multiple Consumer

Even if the previously introduced basic concept is relatively simple, its actual implementation holds some challenges. In the following, I will describe these challenges and present solutions for the use-case of single producer and multiple consumers, based on my novel GAM based CCM implementation for the KeyValue-Pool. The global `KVPool` provides three core functionalities:

- Putting values into the `KVPool`
- Getting values from the `KVPool`
- Release of unused memory

The `put` function is used to insert a `KVPair` into the `KVPool`. If the key is not already stored in the pool, it simply creates a new `KVPair`. Otherwise the existing `KVPair` will be updated. The value can be retrieved in constant time using a simple hash function. Moreover, the `put` function contains the proposed COW mechanism: when I update the value for a key, the consumer copy of the value is updated by a clone of the producer reference. Note that the memory of the old consumer copy is untouched and no memory is freed at this point, only the pointer has been replaced. This implements the copy-on-write mechanism, the MVCC.

The `get` function is used to retrieve an existing `KVPair` from the `KVPool`. However, I have to indicate whether I have to return the producer reference or the consumer copy. To do that, I additionally include a variable that indicates the access right. Finally, the pool implements the `scan` function to free allocated memory, based on Michaels approach [131]. The `scan` function checks whether the entries of a given list of retired `KVPairs` (retired `KVPairs`, which are returned by a put function call) are currently under use. More specifically, if there is any hazard pointer from a concurrent thread pointing towards a given entry. The memory of the `KVPairs` will be freed if there is no intersection between the list of a threads retired `KVPairs` and the global hazard pointer list (see Michael [131] for more details). Each software component (resp. `System`) that can access the `KVPool` is considered as a thread.

Algorithm 1 KVPool::put(key,value)

```
1: if key in map then
2:     slot = map.get(key)
3:     slot.producerreference = value
4:     retired = slot.consumercopy
5:     slot.consumercopy = value.copy()
6:     return retired
7: else
8:     map.insert(key,value)
9: end if
```

Algorithm 2 KVPool::get(key,access)

```
1: if key not in map then
2:     return empty
3: else
4:     slot = map.get(key)
5:     if access is producer then
6:         return slot.producerreference
7:     else
8:         return slot.consumercopy
9:     end if
10: end if
```

I define an abstract class called `KVInterface`, which serves as a wrapper for every software component that wants access to the `KVPool`. It provides two wrapper functions of the `KVPools'` `put` and `get` function. These wrapper functions additionally provide the management of the thread-local lists of acquired hazard pointer as well as retired consumer copies. The `KVInterface::put` wrapper calls the `put` function of the `KVPool` and retrieves the retired `KVPair`, if available. The retired `KVPair` is inserted into a thread-local list of retired `KVPairs`, which is later used to free the retired working copy. Similar to the `get` function of the `KVPool`, the `KVInterface::get` wrapper function distinguishes whether the access is from the producer or from a consumer of the `KVPair`. If the producer of the `KVPair` access the `KVPool`, the corresponding `KVPair` (replaced consumer copy) is returned.

Algorithm 3 KVInterface::put(key,value)

```
1: retired = pool.putPool(key,value)
2: if retired is not null then
3:     retiredKVPairs.add(retired)
4: end if
```

Algorithm 4 KVInterface::get(key,access)

```
1: if access is consumer then
2:     value = pool.getPool(key,consumer)
3:     hp = &value
4:     acquiredHazardPointers.add(hp)
5:     return value
6: end if
7: return pool.getPool(key,producer)
```

As there is only one producer for each `KVPair` no memory management is needed because the producer works on the producer reference of the `KVPool` record. If a consumer wants to access the `KVPool`, a hazard pointer is created for the record and saved to a thread-local list of used hazard pointers, indicating that no other concurrent thread should free the memory of the `KVPair`.

Algorithm 5 KVInterface::release()

```
1: for all acquired hazard pointers hp do
2:     release hp
3: end for
4: pool.scan(GlobalHazardPointers, RetiredPairs)
```

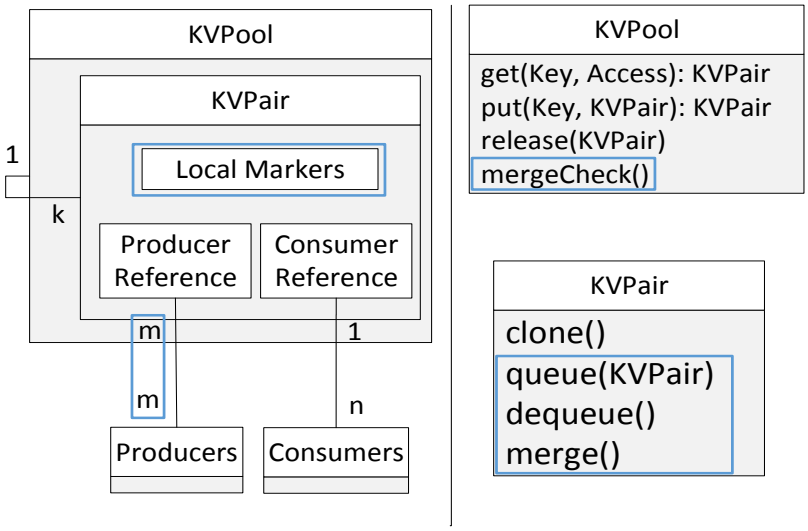


Figure 3.8: UML class diagrams of the actual KeyValuePool implementation illustrating the difference between LAM and GAM based concept: the newly introduced LAM based concepts are marked in blue. The KeyValuePair introduces the new merging concept for multiple writing operations and several producer versions. The KeyValuePool itself has to provide a new function for ensuring that the required merge operations of KeyValuePairs are called whenever a producer version is outdated. At last, the atomic references are no longer stored in the KeyValuePairs itself because they can't themselves determine anymore if they should be merged because other KeyValuePairs are in other thread scopes.

The calling of several `KVInterface::put` and `KVInterface::get`, inserts arbitrary `KVPairs` (old consumer copies of threads, produced by updating `KVPairs`) and hazard pointer (references to used `KVPairs`) to the thread-local lists. After finishing all operations in one thread cycle, the `KVInterface::release` function is called. This `release` function releases all acquired hazard pointers from `KVInterface::get` calls, indicating other threads that the memory can be safely freed.

For producers that additionally call the `KVInterface::put` function, the second part of the release function tries to free old consumer copies of the thread produced `KVPairs`, by calling the scan function of the central `KVPool`. This is similar to the deletion of retired pairs from the `KVPair` list. After each data access of each `KVInterface` all used references to `KVPairs` are released. It may happen that an arbitrary number of old consumer copies could not be freed because some concurrent threads are still using them. Due to the wait-free access of the `KVPool` that guarantees progress of each thread, every claimed memory of a `KVPair` will be freed after some time. The maximum time is defined by the thread cycle time of the slowest `KVPair` consumer.

Each `Component` implementation has to define its own pool record by implementing the abstract class `KVPair` and by defining arbitrary member variables. These member variables can be accessed via a key, which is determined when the `KVPair` is first stored in the `KVPool`. This allows us to generate an arbitrary number of complex values for each key and it reduces the amount of required `KVPairs`. Moreover, it avoids unnecessary calls of `get` functions.

3.4 Local Atomic Markers: Multiple Producer, Multiple Consumer

In this section, I present an extension of my GAM KeyValuePool based CCM which overcomes the limitation of single producers. Namely, I present a novel method that avoids the global hazard pointer management, and I present a mechanism that also allows wait-free write access for several concurrent producers. Still, the approach is a MVCC and leads to C&C.

In the GAM based KeyValuePool concept, I used hazard pointers to avoid race conditions. If a consumer wants to read a consumer copy, it generates a hazard pointer that will be destroyed when the reading operation has been finished. All hazard pointers are stored in a global list of the KeyValuePool. I used a wait-free list with atomic operations to organize this global hazard list. The main problem of this implementation are the relatively large amount of memory – each reading operation requires a pointer – and the time that is needed to release the pointers – I have to iterate through the list to find the pointer before I can release it. Moreover, after each hazard release, I have to search the list if there are more hazards pointing to the same consumer copy in order to decide whether to delete the copy or not.

The main idea to avoid this problem is to use LAM instead of global hazard pointers. Basically, a LAM is a single atomic integer value stored with each consumer copy. If a consumer wants to read the copy, it simply increments this LAM with an atomic operation. When it has finished the reading, it decrements the marker. Obviously, the local copy can be deleted if and only if the marker equals zero, because in this case, no reader is reading it any more.

The advantage is that I need only one single atomic integer for each consumer copy and not a hazard pointer for each consumer that access the copy. Moreover, the time consuming search for other hazards accessing the same copy can be omitted and I do not need complicated data structures like the wait-free list.

In detail, if a producer wants to modify data in a `KVPair`, the `KVPool` returns it a copy of the current data. Each concurrent producer gets its individual copy and all copies are labeled with a time code. Obviously, concurrent reading copies will get the same time code. After finishing the modification, the producers informs the `KVPool` by writing back the data. Now, two possible cases may happen: first case, both the current consumer copy and the new producer copy have the same time code. In this case, I can simply replace the consumer copy and assign the current time code to it.

The second case is more interesting: the time code of the consumer copy and the producer copy are different. This means, the consumer copy has been already replaced by another producer. In this case, I combine the data of both copies to a new consumer copy. In order to combine both copies, the creator of the `KVPair` has to define an individual merge function which is used by the `KVPair` to solve the conflict.

This function allows a high flexibility and covers all cases that may happen in a concrete RIS implementation. For instance, it can simply overwrite all values, compute the maximum or minimum of both values, or in case of virtual sculpting or assembly it can summarize the data via performing a linear interpolation between modified vertices. Obviously, as a special case, it can implement the first-come-first-serve strategy of [145] by simply keeping the first change and throwing away all others, but also the local lock mechanism described by [63].

Several producers are allowed to write back their copies simultaneously. Because of this, I have to buffer this backwriting with a wait-free queue for each `KVPair`. Copies that are put into the queue are merged sequentially by the `KVPool`, outside the producers and consumers thread scope, using the `merge` function.

Algorithm 6 KVPool::get(key,access)

```
1:  if key not in map then
2:      return empty
3:  else
4:      slot = map.get(key)
5:      if access is producer then
6:          return slot.producerreference.clone
7:      else
8:          slot.consumerreference.localMarker++
9:          return slot.consumerreference
10:     end if
11: end if
```

Algorithm 7 KVPool::put(key,value)

```
1:  if key in map then
2:      slot = map.getValue(key)
3:      if slot.timecode = value.timecode then
4:          slot.producerreference = value
5:          retired = slot.consumerreference
6:          slot.consumerreference = value.clone
7:      else
8:          slot.producerreference.queue(value)
9:          retired = slot.consumerreference
10:         pool.notify
11:     end if
12: else
13:     map.insert(key,value)
14: end if
15: return retired
```

Consequently, this sequential merging does neither influence the wait-free read, nor the wait-free write capability of my data structure. All components can still access all `KVPairs` for reading and writing. Figure 3.8 illustrates the above stated LAM concept as well as the relationship of producers and consumer with respect to the `KVPairs` stored in the `KVPool`.

The LAM based KeyValuePool, implemented as a hash map like the GAM one, offers two access functions for the components: `put` and `get`. If consumer wants to read a value, it calls the `get` function and the LAM based `KVPool` returns the current consumer copy. Moreover, it increments the LAM (see Algorithm 6).

Algorithm 8 KVPool::mergeCheck()

```
1: while slots are being marked do
2:     for all marked slots of map do
3:         slot.producerreference.dequeue
4:         slot.consumerreference = slot.producerreference.clone
5:     end for
6: end while
```

If the consumer has finished reading, the `release` function will be called that decrements the LAM again. Additionally, it checks whether the LAM is zero and, probably, allows the deletion of the consumer copy if no other consumers still reads it.

Writing access also begins with a call of the `get` function. However, in this case, the LAM based `KVPool` returns a clone of the producer copy. When the producer has finished writing, it calls the `put` function (see Algorithm 7). The `put` function ensures the above stated wait-free reading access: it either replaces the old consumer copy by the new value or it collects those `KVPairs` that need to be merged (see Algorthm 7 line 9). Actually, the compare of the time code has to be performed atomically in order to avoid race conditions during concurrent writes. Collected `KVPairs` are put in a queue that the LAM based `KVPool` maintains for each entry individually.

Additionally, the `put` function allows the insertion of new `KVPairs` to the LAM based `KVPool`. Moreover, it returns the old consumer reference as retired. This allows the deletion of this pair by the `release` function.

If the put function recognizes concurrent writes, i.e. a queue for a KVPair is not empty, the LAM based KVPool calls a mergeCheck function (see Algorithm 8) that processes the merges of those KVPairs.

Each KVPair can store arbitrary data, hence, the merge function has to be implemented per KVPair. The merge function operates on the defined member variables of the KVPair which should be merged. Obviously, the users can implement their own functions to provide the required merge for their KVPairs.

When introducing a merging function for my concurrency control management, I have to define those operations on KVPairs which can be conducted in the proposed wait-free manner. This means that the concept does not allow to implement new parallelizations but only to operate existing approaches (e.g. [86]) with the highest possible throughput. These wait-free operations must be abelian because there is no synchronization between the write processes.

In abstract algebra, an abelian group, also called a commutative group, is a group in which the result of applying the group operation to two group elements does not depend on the order in which they are written. This means that the group obeys the axiom of commutativity. Commutative operations are, for instance, insert, deletion, overwrite, addition, and multiplication.

Mathematically, an abelian group is a set, A, together with an operation $\cdot$ that combines any two elements a and b to form another element denoted $a \cdot b$. The symbol $\cdot$ is a general placeholder for a concretely given operation. To qualify as an abelian group, the set and operation, $(A, \cdot)$, must satisfy five requirements known as the abelian group axioms:

- **Closure:** $\forall a, b \in A$, the result of the operation $a \cdot b$ is $\in A$.
- **Associativity:** $\forall a, b, c \in A$, the equation $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ holds.
- **Identity element:** $\exists e \in A$, such that for $\forall a \in A$, the equation $e \cdot a = a \cdot e = a$ holds.
- **Inverse element:** $\forall a \in A$, there exists an element $b \in A$ such that $a \cdot b = b \cdot a = e$, where e is the identity element.
- **Commutativity element:** $\forall a, b \in A$, the equation $a \cdot b = b \cdot a$ holds.

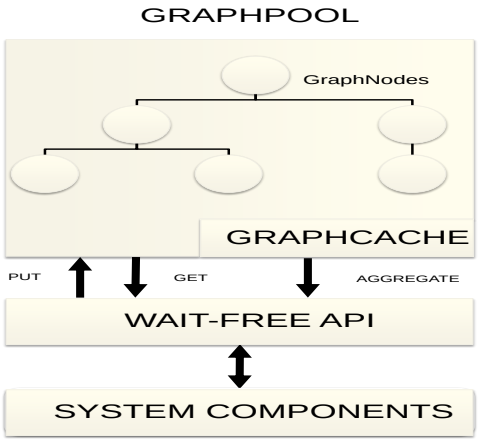


Figure 3.9: The graph based key value pool (GraphPool) has two access workflows for all system components (for instance threads or any software class that implements the required interface). The stored data is available is via the previously introduced put and get functionalities of the KeyValuePool as well as aggregate functions. These aggregate functions imitate relational queries and also the graph based key value pool cache (GraphCache) that stores results from previously executed aggregate calls for fast data retrieval.

3.5 Graph based Nested Hash Maps

Based on my GAM and LAM wait-free hash maps within the ECS pattern, I present in this section the last development stage: graph based nested hash maps, the GraphPool. The GraphPool is a data management approach for virtual testbeds, that, in addition to the previously described benefits, eliminates the disadvantages of DBMS based approaches for 3DST applications. In order to do so, I extend the original GAM and LAM KeyValuePool approaches with relational database technology, specifically aggregate queries and caching.

This combination solves the main requirements of data managements for virtual testbed research: scalability, adaptability, and performance [154]. My wait-free data access enables high scalability while requiring only low latencies, even for massive concurrent data transactions. In addition, my hash maps store object-oriented data formats. As a result of this, the time consuming separation of data into interrelated tables of relational database technology is eliminated. This leads to a better adaptability of new data formats.

In order to implement these concepts, my approach uses the previously introduced GSS, based on wait-free access using my GAM or LAM KeyValuePool concept. The GSS is stored in my GraphPool, resulting in a centralized data management. This means, the GraphPool is used for storing and managing all parts, dynamic as well as static, of the shared simulation data (e.g. a simulation model) in a consistent object-oriented data schema. This object-oriented approach correlates to typical 3D construction and environmental data used in 3DST applications.

During runtime, every simulation component can replicate any parts of the GSS into its LSS while local changes are tracked and written back into the GSS. These read and write processes execute in wait-free behaviour, without synchronisation (see Sections 3.3 and 3.4). As mentioned above, not only all static parts of a 3D simulation model (e.g. the 3D environment) are stored in the GraphPool, but also all dynamic objects which are changed by the running simulation. These changes are likewise written to the GSS, hence communicating the new state of the simulation model to all software components.

Consequently, the GraphPool drives the simulation itself as it represents the central communication (dataflow and workflow) hub.

Many more advantages arise from using centralized data management system for a 3DST application [126]: different applications (e.g. for authoring) can be employed using its standardized interfaces, an inherent rights management provides means for fine grained access control, consistent data schema, solution to object identification ("id problem"), and (spatial) queries allow very selective loading and changing of the simulation state and simulation model.

In order to use my wait-free CCM for all data transactions, I use my proposed framework consisting of ECS pattern and KeyValuePool concept. Thus, I identify all kinds of data that need to be shared between different components, e.g. simulation time or the transformations of the objects in the scene. In addition to the original concept, I further arrange this data into logically structured data packets. This allows use to define aggregate queries on the data (see Section 3.5.2). For instance, the 3D geometry of a car but also its' individual components in a car simulation, such as mass, position, velocity or acceleration are logically arranged into one data packet. I store these data packets in my GraphPool. I further assign a set of unique key-identifiers (*object-key* and *member-keys*) to each of these data packets. The member-key references the complete data packet while member-keys reference a specific data type within the packet. I denote this combination of identifiers and GraphPool as graph based key value pool node (GraphNode).

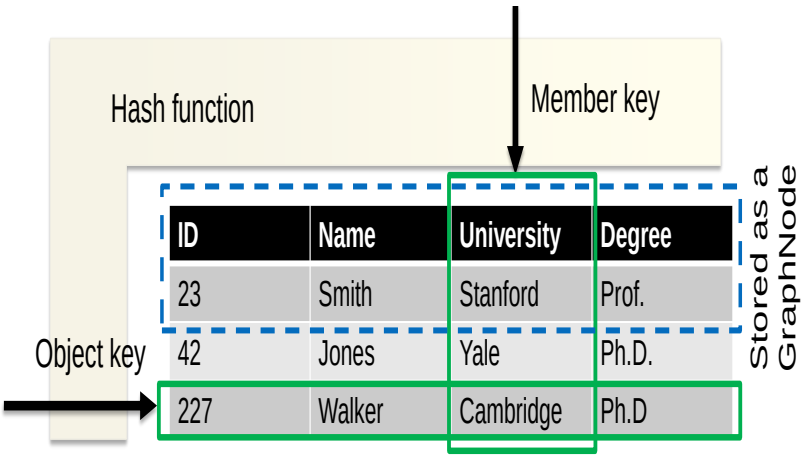


Figure 3.10: Access query example for my property graph model: member- and object-key can be used like SQL-ish aggregate functions. These aggregate queries of my GraphPool retrieve a data set (marked in green). This data set is collected from an arbitrary amount of GraphNodes (blue) as indicated by the used keys. This imitates a table-based storage of data as it is done by traditional databases. However, instead of using a rigid table schema, my approach uses object-oriented GraphNodes leading to a NoSQL like data management. This facilitates a faster data access and removes the previously mentioned drawbacks of relational databases for virtual testbeds.

All GraphNodes are registered in my GraphPool and memory is reserved for the data, like in the original KeyValuePool approach. However, the GraphPool connects all GraphNodes into a graph-based lookup structure and construct thereby the GSS. Consequently, the GraphPool is technically a wait-free nested hash map because each GraphNode internally implements a hash map which can be obtained from the GraphPool.

Hash maps outperform other container types (e.g. lists or arrays) due to their constant lookup, insertion and deletion time of $\mathcal{O}(1)$ which makes them perfectly suitable for a high performance data management. My nested hash map approach enables row and column based queries for the stored data and is a NoSQL data management. From a relational database point of view, a GraphNode is a 1-by-n schema-less table in which all columns n can be accessed separately by n member-keys. Figure 3.10 illustrates this concept with an simple person data example while Figure 3.9 illustrates the main concept of the GraphPool consisting of GraphNodes.

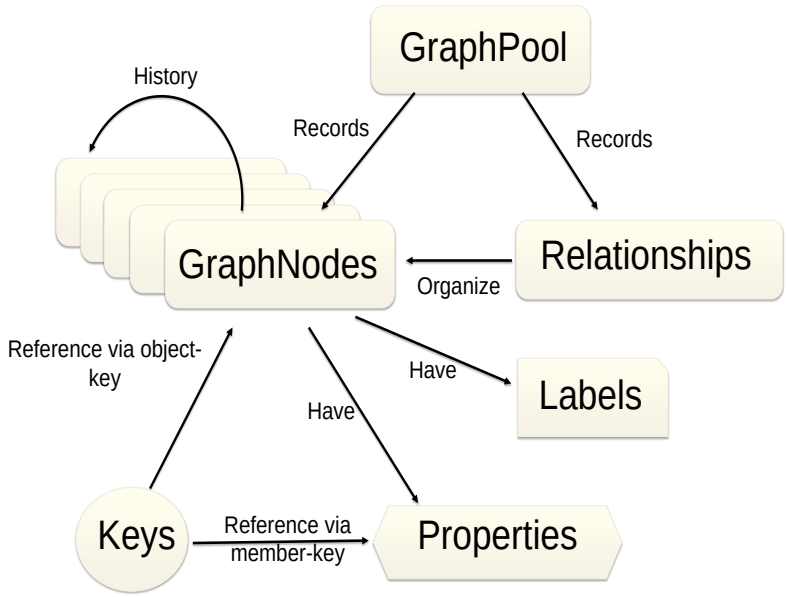


Figure 3.11: The building blocks of my property graph model: the GraphPool consists of linked GraphNodes. Each GraphNode has a querable history and can be, in its' entirety, retrieved with a unique object-key. Each property of the GraphNode can be retrieved via a unique member-key. GraphNodes are structured with relationships in the graph and are semantically annoated with labels.

As all intermediate states of the simulation are made persistent in the GraphPool, a simulation run can easily be captured by my versioning mechanism. This versioning mechanism generates a time-stamped history of all GraphNodes. These recorded time-stamped GraphNodes represent a query-able archive of the complete simulation.

Every simulation component can be used in an off-line viewing mode to replay a simulation run step by step, allowing analysis and debriefing of the complete simulation. Consequently, my approach allows the subsequent replay, archiving, debriefing and analysis of the complete simulation. Additionally, simulation components can also be used in an off-line viewing mode to replay a simulation run step by step, allowing analysis and debriefing.

3.5.1 Property Graph Model for Nested Hash Maps

In order to allow for relational core and aggregate functions, I arrange the GraphNodes in my property graph model (PGM) structure. I define this graph structure as $\mathcal{G} = \{\mathcal{N}, \mathcal{R}, \mathcal{K}, \mathcal{P}\,\mathcal{L}\}$ with $\mathcal{N}$ nodes, $\mathcal{R}$ relationships, $\mathcal{K}$ keys, $\mathcal{P}$ properties and $\mathcal{L}$ labels.

The property graph contains all GraphNodes as a connected graph which can hold any number of properties within its hash map. GraphNodes can be tagged with labels representing their different roles in the simulation domain. Labels can serve as a contextualization for GraphNode and relationship properties. Furthermore, labels may also denote constraint or metadata information of GraphNodes.

Every relationship provides a directed, named semantically relevant, connection between two GraphNodes. A relationship always has a direction, a start node, an end node and a type.

The relationship type can be arbitrary, for instance a weight, cost, time interval, distance or inheritance/tree structure.

GraphNodes can share any number or type of relationships because they are stored efficiently, without sacrificing performance. In order to enable fast traversion of the GraphNodes, the GraphPool can navigate between GraphNodes regardless of relationship direction.

Furthermore, I follow the consistent rule that no broken links shall be present in the graph. Since a relationship always has a start and end GraphNode, a GraphNode can not be deleted without also deleting its associated relationships. Consequently, an existing relationship will never point to a non-existing endpoint. Moreover, the GraphPool provides a versioning mechanism that archives every previous state of the GraphNodes as a time-stamped version. This mechanism provides transparent access to these historic states. A user interface element or any other component of the simulation system can then set a reference time and the versioning interface takes care of reloading the appropriate versions of the object data. Furthermore, there is no global main loop required; each simulation component can access the GraphNodes, i.e. read or write, at any point in time. Figures 3.10 and 3.11 illustrate my property graph structure.

3.5.2 Relational Core & Aggregate Queries

In this section, I described how relational core and aggregate queries can be implemented within my PGM structure with caching. The `GraphPool` has to provide two relational core functionalities:

- Pushing a LSS to the GSS, respectively putting values into the `GraphPool`
- Retrieving a LSS from the GSS, respectively getting values from the `GraphPool`

The GSS is thereby defined by the complete set of `GraphNodes` which are stored in the `GraphPool`. As a result, the LSS is a subset of these `GraphNodes` which a simulation component can access by the `GraphPools` `put` and `get` function.

The `put` function is used to update a `GraphNode` via its object-key in the `GraphPool`. If the object-key is not already stored in the pool, it simply creates a new `GraphNode`. Otherwise the existing `GraphNode` will be updated. The value can be retrieved in constant time using my hash function as described below. The `get` function is used to retrieve an existing `GraphNode` from the `GraphPool`.

As previously described (see Section 3.4), also the `GraphPool` differentiates between consumer and producer simulation components. Furthermore, it also uses the proposed LAM concept for wait-free transactions. The principle applies here to the same extent: consumer components only read a set of `GraphNodes` whereas producer components read and write a set of `GraphNodes`. Therefore, the `GraphPool` maintains two copies of the `GraphNodes`, a producer reference and a consumer reference, like in my proposed KeyValuePool approach. This means, that read requests for a `GraphNode` will return the consumer reference and that write requests for a `GraphNode` will return the producer reference.

If a consumer wants to read a value, it calls the `get` function and the `GraphPool` returns the current consumer copy. This is decided via an access request which every get query contains. Moreover, it increments a LAM (see Algorithm 10) of the consumer reference. If the consumer has finished reading, the consumer decrements the LAM again. In addition, it checks whether the LAM is zero and, in case no consumer is reading it anymore, deletes the consumer reference. If the memory can not be directly deleted, the `GraphPool` will take care of releasing the memory at a later time point as described in Section 3.4. Writing access also begins with a call of the `get` function in order to retrieve the data which should be manipulated. In this case, the `GraphPool` returns the producer reference and sets the ownership of the simulation component.

Algorithm 9 GraphPool::put($\mathcal{K}$ object-key, $\mathcal{V}$ value)

1: $\mathcal{R}$ retired graph node
2: **if** $\mathcal{K} \in$ GraphPool **then**
3: $\mathcal{N}$ graph node = GraphPool[$\mathcal{K}$]
4: **if** $\mathcal{V}_{Id}$ = $\mathcal{N}$.*Producer.Id* **then**
5: $\mathcal{N}$.*Producer* = $\mathcal{V}$
6: $\mathcal{R}$ = $\mathcal{N}$.*Consumer*
7: $\mathcal{N}$.*Consumer* = $\mathcal{V}_{clone}$
8: **else**
9: $\mathcal{N}$.*Producer.Queue*($\mathcal{V}$)
10: $\mathcal{R}$ = $\mathcal{N}$.*Consumer*
11: GraphPool.notify
12: **end if**
13: **else**
14: GraphPool.insert(pair($\mathcal{K}$,$\mathcal{V}$))
15: **end if**
16: GraphCache.update($\mathcal{K}$)
17: return $\mathcal{R}$

Algorithm 10 GraphPool::get($\mathcal{K}$ object-key, $\mathcal{A}$ access)

1: **if** $\mathcal{K} \notin$ GraphPool **then**
2: return empty
3: **else**
4: $\mathcal{N}$ graph node = GraphPool[$\mathcal{K}$]
5: **if** $\mathcal{A}$ is producer **then**
6: $\mathcal{N}$.*Producer.Id* = $\mathcal{A}$.Id
7: return $\mathcal{N}$.*Producer.Clone*
8: **else**
9: $\mathcal{N}$.*Consumer.MarkerIncrement*
10: return $\mathcal{N}$.*Consumer*
11: **end if**
12: **end if**

This ownership is an atomic id of the producer reference. It is set and checked in the `get` and `put` function. In the get function, the producer reference is marked with the corresponding producer id. When the write operation is conducted, the producer checks whether its id is the current one. If this is not the case, another producer has updated the producer reference in the meantime (see Algorithm 9). This means that another system component has changed the `GraphNode` and the changes have to be merged in order to preserve a consistent `GraphNode` state. The required merge is then implemented as described in section 3.4.

Algorithm 11 aggregate($\mathcal{K}$ object-keys, $\mathcal{I}$ member-keys, $\mathcal{A}$ aggregator)

1: $\mathcal{H}$ = getHash($\mathcal{K}$, $\mathcal{I}$, $\mathcal{A}$)
2: **if** $\mathcal{H}_{valid}$ **then**
3: **return** GraphCache.get($\mathcal{H}$)
4: **end if**
5: $\mathcal{C}$ = empty collection
6: **for** $\mathcal{K}_i \in \mathcal{K}$ **do**
7: $\mathcal{N}$ GraphNode = GraphPool[$\mathcal{K}_i$]
8: $\mathcal{C}$ += $\mathcal{N}[\mathcal{I}]$
9: **end for**
10: $\mathcal{R}$ = $\mathcal{A}$($\mathcal{C}$)
11: GraphCache.set($\mathcal{H}$,$\mathcal{R}$)
12: **return** $\mathcal{R}$

In short, conflicting producer references are sorted into a producer queue and the `GraphPool` calls a merge function that processes the merges of those `GraphNodes`. In order to do so, every `GraphNode` contains a merge strategy (e.g. first-come first-serve or averaging the values). Algorithms 9 and 10 illustrate the implementation.

In contrast to the above introduced relational core functionalities which use single data, relational aggregate functions use multiple data. Aggregate functions are essential functions of relational databases. These functions collect in their original database implementation the values of multiple columns and rows. They use this collection as input on certain criteria which further filter the result. Typically, selective (equal, not, smaller, greater, between) and numerical (average, min, max, sum) operators are most commonly used for aggregate functions.

Algorithm 11 illustrates the general implementation of an aggregate function in the PGM structure. First, the corresponding hash is determined. If the result of the aggregate function was computed before, I take the value from the my cache structure, the GraphCache. If not, I recalculate the result of the aggregate function. In order to do so, I collect the corresponding data and apply the associated aggregate function onto this data and store the result in the GraphCache. I detail the GraphCache in the next section.

3.5.3 Wait-Free Caching

Caching is widely used in database technology to store results of expensive aggregate query results. This enables the database to quickly deliver previously computed results. I also provide a caching strategy based on a tree data structure, called GraphCache.

The GraphCache supports two types of workflows. First, if a GraphNode is updated in the GraphPool from a software component by calling the put function, the associated stored data in the GraphCache is marked as outdated. Second, if an aggregate query is used, either a cached result is returned or the associated nodes in the GraphCache are marked as valid and the corresponding data is updated.

For the first case, the GraphPool has to support a GraphCache traversal via object-key in order to find those hash values which (partly) consist of the given GraphNode. For the second case, the GraphPool needs to support a traditional cache traversal via hash value in order to find the corresponding cached query result.

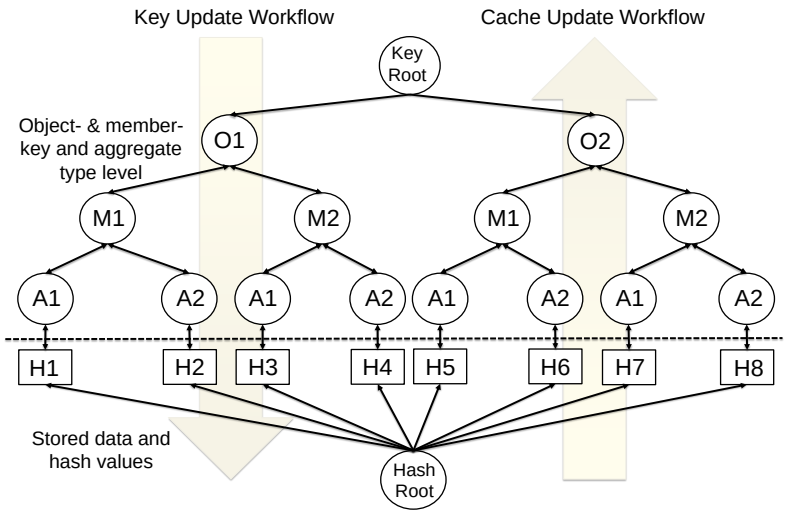


Figure 3.12: The GraphCache stores results from previously executed aggregate function calls. When an aggregate function is called, the GraphCache ic checked to see whether the result has already been calculated once before. If not, the result is recalculated, returned and stored in the GraphCache. For this procedure the GraphCache is traversed from the hashroot. If GraphNodes change in the GraphPool, the pre-calculated results in the GraphCache are no longer valid and may no longer be used. To do so, the GraphCache is traversed from the keyroot and the cache entries are marked invalid.

Algorithm 12 initGraphCache($\mathcal{O}$ object-keys, $\mathcal{M}$ member-keys , $\mathcal{A}$ aggregate types)

```
1: KR = key root
2: CR = cache root
3: for O_i ∈ O do
4:    R = node with O_i
5:    for M_i ∈ M do
6:       T = node with M_i
7:       for A_i ∈ A do
8:          U = node with A_i
9:          T_i−childs += U
10:      end for
11:      R_i−childs += T
12:      CR_childs += hash(O_i, M_i,A_i)
13:   end for
14:   KR_childs += R
15: end for
```

Algorithm 13 hash($\mathcal{K}$ object-keys, $\mathcal{M}$ member-key, $\mathcal{A}$ aggregate-type)

```
1: V = empty hash value
2: P = prime number
3: H = hash function
4: for K_i ∈ K do
5:    V = V · P + H(K_i);
6: end for
7: V = V · P + H(M);
8: V = V · P + H(A);
9: return V
```

Consequently, the GraphCache is accessible via two root nodes: the key-root and the hash-root. This enables fast access because unnecessary tree traversal is avoided (see Figure 3.12).

Due to the main principle of wait-free access of the underlying CCM (see Sections 3.3 and 3.4), I propose a wait-free caching approach in order to maintain overall wait-free access control of the GraphPool. When a wait-free data management system is implemented, every access workflow to the stored data has to be wait-free, in order to guarantee the wait-free behavior of the complete system. Therefore, I initialize the complete GraphCache at application startup. This initialization at startup has the advantage that cache entry insertion and deletion does not have to be implemented in wait-free manner, but only the update. Implementing efficiently wait-free insertion and deletion tree data structures is hard to achieve and, up to now, does not yield any performance boosts [154].

However, the required update process can be implemented with an atomic boolean, which is used as an indicator whether a cache entry is outdated or not. The GraphCache initialization involves all possible combinations of object-, member-keys and aggregate types because the queries conducted by the system components are unknown at application startup. This results in a tree structure with $o \cdot m \cdot a$ nodes, where o is the number of object-keys, m is the number of member-keys and a is the number of aggregate types.

In detail, the initialization of the GraphCache involves the creation of the aforementioned two root nodes, the key-root $\mathcal{KR}$ and cache-root $\mathcal{CR}$. These roots are created at first. The GraphCache further consists of three node levels: object-keys, member-keys and aggregate types. For each of possible combinations of these keys, I have to generate one hash value, in order to uniquely identify the cached data. In order to generate these hash entries, I iteratively concatenate them as nodes within the GraphCache. Object-keys are added to the $\mathcal{KR}$ as nodes and all member-keys are added to the object-key nodes. Finally, all aggregate types are added to the member-key nodes (see Algorithm 12). After the initialization, the GraphCache can be directly used for caching operations.

The GraphCache contains a large number of cache entries for sophisticated simulations. I use a uniform distribution of hash values in order to avoid collisions for cache lookup. In order to deliver such a uniform distribution of hash values, even for massive amounts of cache entries, I use a prime-based hash generation in order to generate unique hash values for all concatenations of object- and member-keys with respect to all defined aggregate functions (see Algorithm 13).

However, most of these possible key combinations will never be used during runtime. In order to reduce the memory overhead, I propose a pruning strategy that removes unused nodes from the GraphCache. The main idea is to remove those nodes which have not been used by the application after a predefined timespan (resp. amount of cache operations).

3.6 Applications

The presented KeyValuePool and GraphPool based approaches enable the implementation of very different categories of virtual testbed applications. Exemplarily, I present the application of a high fidelity dynamics and spacecraft EDL (entry, descent and landing) end-to-end spaceflight mission simulator called KaNaRiA [14, 149]. However, there are currently many more applications in their design phase, e.g. for swarm-based exploration of planetary surfaces.

KaNaRiA (from its German acronym: Kognitionsbasierte, autonome Navigation am Beispiel des Ressourcenabbaus im All) is a joint venture of the University of Bremen and the Universität der Bundeswehr in Munich financed by the German Aerospace Centre (DLR - Deutsches Zentrum für Luft- und Raumfahrt). KaNaRiA comprises the major goal to develop a software mission simulator which serves as a platform for test, verification and validation of novel autonomous spacecraft navigation algorithms. This mission simulator, a virtual testbed for feasibility studies, concerns the following spacecraft mission concepts: long cruise phases, multi-body fly-bys, planetary approach and rendezvous, orbiting in a-priori unknown dynamic environments, controlled descent, precise soft landing, docking or impacting, surface navigation or hopping.

In order to develop the KaNaRia virtual testbed, I implemented a simplified version of ESAs ARCHEO-E2E system [29] that defines a reference architecture for spacecraft engineering feasibility studies within my proposed software architecture.

I defined, modelled and implemented all required modules of the testbed ass `Systems`, `Entities` and `Components`. These modules of the ECS pattern communicate via my KeyValuePool approach, as described previously. Examples of these modules are:

- **different spacecraft types** ranging from orbiter to lander designs,
- **instruments and actuators of the spacecraft** such as spectrometers, startrackers, cameras, interferometers, range-finders, etc.,
- **guidance, navigation and control concepts** are modelled as `Systems` which can easily exchange transactions via the KeyValuePool,
- **space environment** including the spacecraft orbit and attitude.

The sensor input (e.g. camera and range finder measurements) for the instruments is synthesized from the simulated environment. In my implementation, all this synthesized data and the current world state (e.g. spacecraft pose, positions of celestial bodies, sensor configurations, scene nodes) are represented as `Components` in my central KeyValuePool. The instruments and the physically-based simulation read and write the entries periodically. Consequently, this scenario has a large amount of concurrent read- and write operations on my KeyValuePool based architecture. Currently, measurements show that up to 1000 transactions per simulation step are conducted. Figure 3.13 shows the visual output of the simulation in which a spacecraft conducts scientific experiments while orbiting an asteroid.

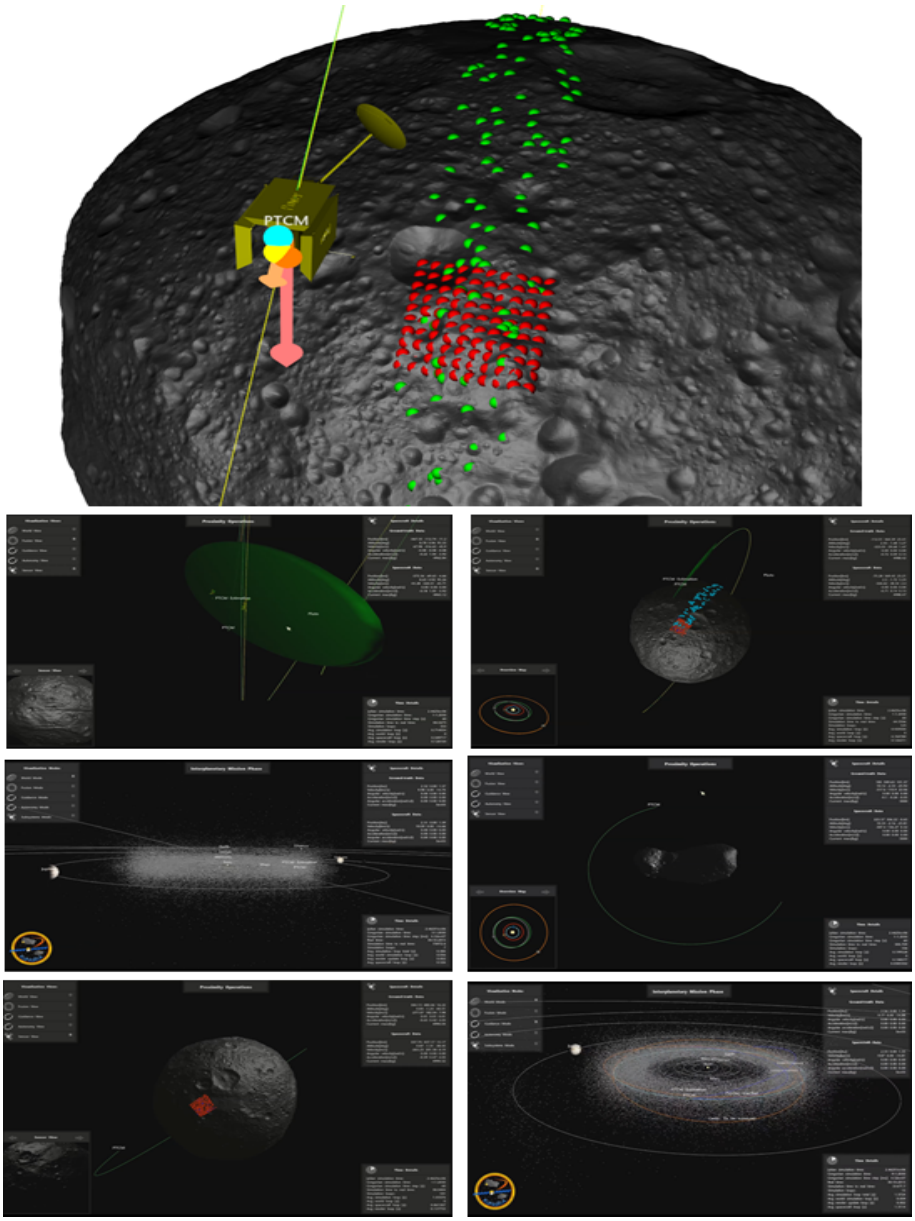


Figure 3.13: Use case study: the KaNaRiA simulator [14, 149] is using the presented approaches for simulating long-term spacecraft missions to asteroids. The figures show how a spacecraft is orbiting an asteroid. Rangefinder (red) and landmark (green) measurements are generated for spacecraft self-localization [48, 75] purposes.

3.7 Results

I implemented my new KeyValuePool based CCMs in C++. I performed experiments on a machine with an Intel Core i7 4-core processor (3.4 GHz) with enabled Hyperthreading, using the Microsoft Visual C++ 14 compiler with all optimizations, operated by Windows 7 64 bit and 8GB of RAM. Due to the limitations of the competitive approaches I had to limit the data pairs to basic primitives like doubles, integers and pre-allocated lists with fixed size of 256 Bytes, for the evaluation setup for the KeyValuePool evaluation. I performed 50.000 read- and write-operations for each test. Additionally, I repeated each individual test 100 times and averaged the resulting timings.

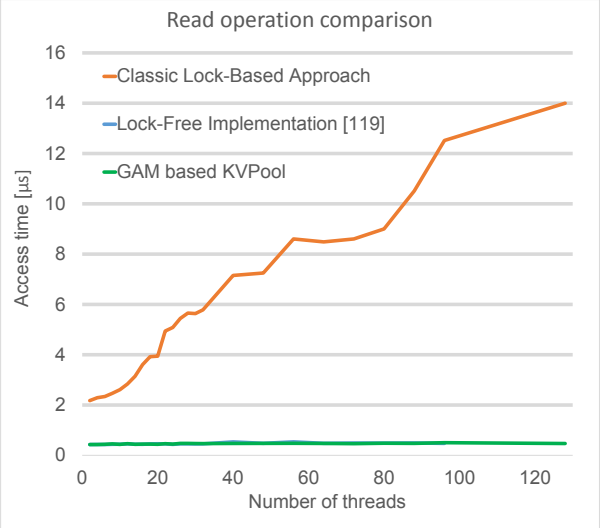
The KeyValuePool and GraphPool based implementations contained 50.000 KeyValuePairs or GraphNodes, each representing a virtual object with its properties. The access to the KeyValuePool and GraphPool was modelled with different numbers of concurrent components, ranging from 2 to 128. To prevent caching, I inserted for each test run the KeyValuePairs and GraphNodes at random positions. The key size was set to 12 Bytes.

3.7.1 Global Atomic Marker Concept

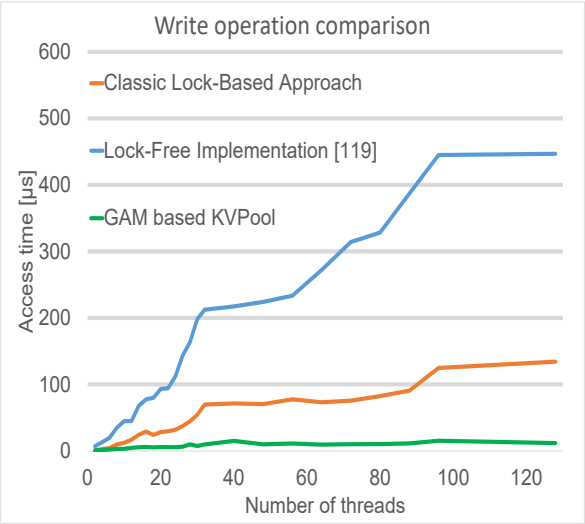
I compared the performance of my GAM based KeyValuePool to two different existing methods. The first competitor was a standard blocking hash map. I used the well-known boost library [35] that uses shared mutexes and allows multiple readers and a single writer accessing the hash map. Additionally, I adopted a lock-free hash map of the original hazard-pointer algorithm that supports wait-free reading and lock-free writing [131]. The GAM based KeyValuePool outperforms both competitors for reading as well as writing operations (see Figures 3.14a and 3.14b). More precisely, it is more than two orders of magnitude faster than the traditional lock-based hash map for reading operations. Obviously, the speed-up increases with an increasing number of threads because the concurrent thread access is limited by the locks in the standard approach (see Figure 3.14a). Even more interesting is the comparison with the lock-free hash map. The GAM based KeyValuePool wait-free method, as well as the lock-free method, support wait-free concurrent reading access of the data. Consequently, both methods perform almost identically for reading operations (see Figure 3.14a). However, the GAM based KeyValuePool also allows wait-free writing access using the COW mechanism instead of CAS, used by the lock-free hash map.

In that case the GAM based KeyValuePool outperforms the lock-free competitor by an increasing factor, depending on the number of threads. Surprisingly, the lock-free hash map is even slower than the traditional lock-based approach in writing operations. This is mainly because the lock-free method was implemented with a spinlock-wait until it can CAS the KeyValuePair between reading operations. The lock-based approach was implemented using boost mutexes [35], which do sleep-waiting, while claiming the writer-lock. They allow other threads to continue with their operations, explaining these results. The GAM based KeyValuePool is independent of the number of concurrent threads. Consequently, the performance boost, compared to both competitors, increases with an increasing number of threads.

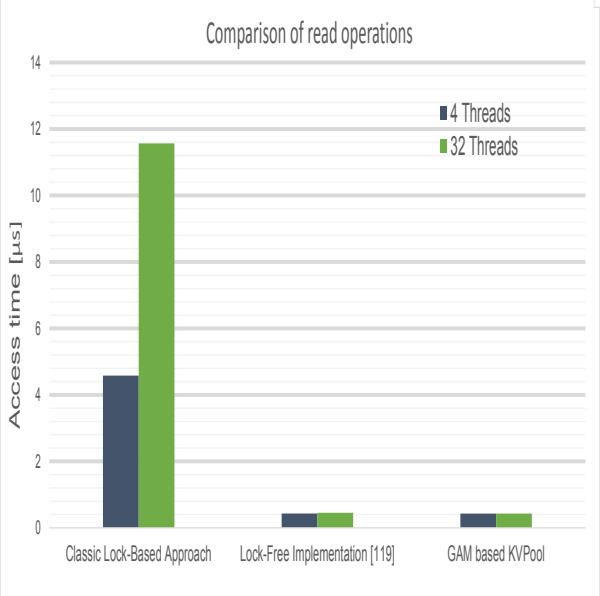
Figure 3.15a and 3.15b show how the concrete timings of the approaches, when using 4 or 32 threads. The illustrations go along with my previous findings, indicating the performance decrease of the lock-based and lock-free approach, explaining the performance boost of the GAM based KeyValuePool. The only bottleneck is the COW mechanism during write operations because it clones the current data. Finally, Figure 3.16 shows that the GAM based KeyValuePool incurs only a very small performance decrease while the KeyValuePair size increases. This means that the COW mechanism mainly determines the access timing.



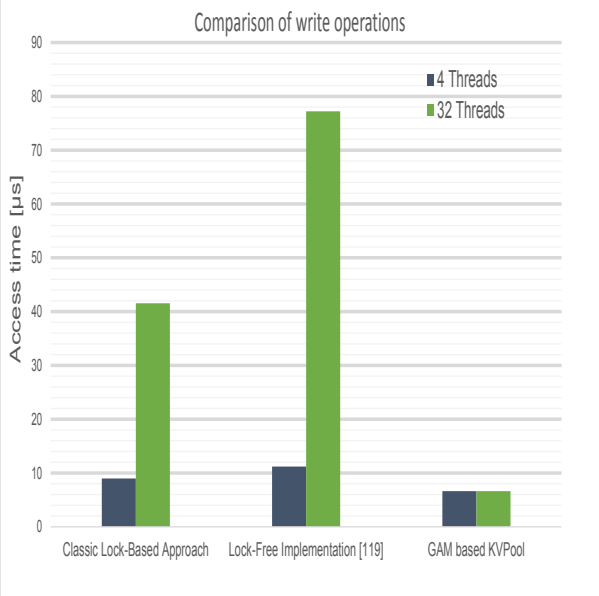
3.14a: Reading KeyValuePairs from the GAM based KeyValuePool. My approach outperforms locking and non-locking competitors.



3.14b: Writing KeyValuePairs to the GAM based KeyValuePool. My approach outperforms locking and non-locking competitors.



3.15a: Overview of timings compared to my GAM based KeyValuePool for writing operations.



3.15b: Overview of timings compared to my GAM based KeyValuePool for reading operations.

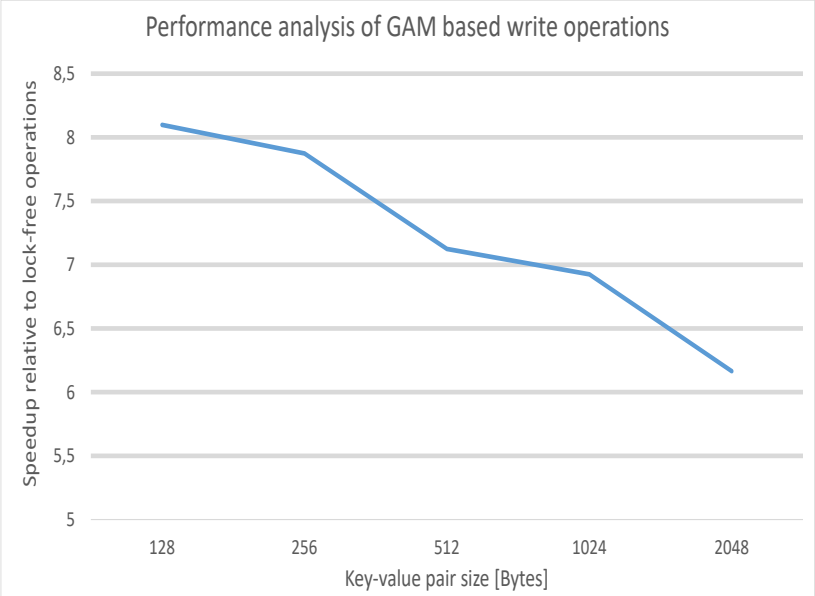


Figure 3.16: Dependency of the GAM based KeyValuePool performance of data package sizes with respect to the lock-free implementation.

3.7.2 Local Atomic Marker Concept

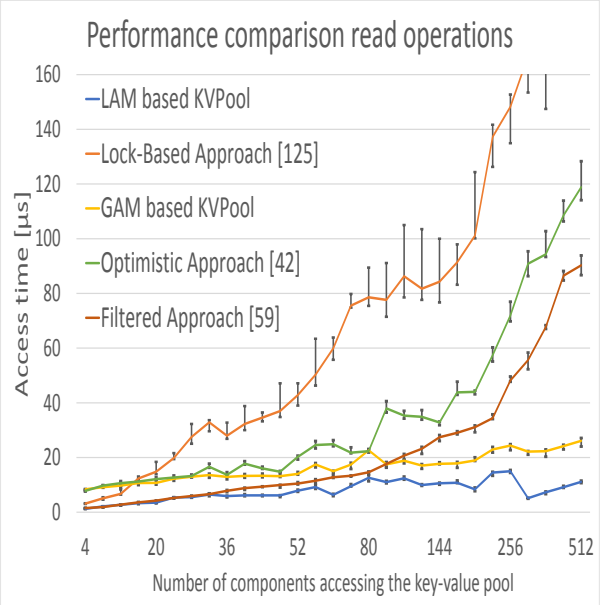
I applied two test scenarios for my LAM based KeyValuePool, in addition to the previous evaluations. In the first scenario the components were equally divided into producers and consumers of the KeyValuePairs. The second scenario involved more producers, namely twice as much as consumers.

I compared the performance of the LAM based KeyValuePool to fmy different existing methods, which I adopted to the KeyValuePool scheme. The first competitor was a standard locking scheme based on the boost locking library [35]. The second approach was a typical filtered concurrency locking implementation based on [63]. The third competitor was an optimistic concurrency locking implementation which simply recomputes the values in case of a concurrent write based on [45]. Finally, I compared LAM based KeyValuePool to the GAM based KeyValuePool.

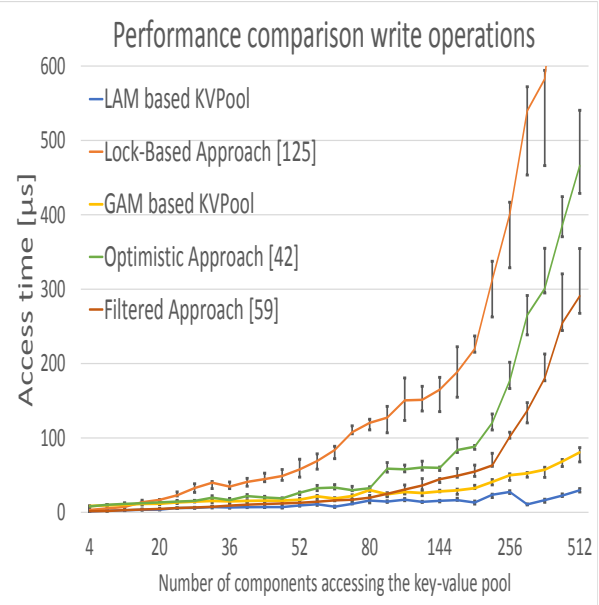
Figures 3.17a and 3.17b show a comparison of the performance with respect to the two proposed test cases.

My evaluations show that the LAM based KeyValuePool outperforms all other competitors. Obviously, its' speed-up increases with an increasing number of components accessing the LAM based KeyValuePool. My approach is almost independent of the number of concurrent components. Additionally, the LAM based KeyValuePool outperforms also the GAM based KeyValuePool. For less than approximately 28 components that concurrently access the LAM based KeyValuePool, the filter-based approach performs almost identically to my approach. However, if more than 30 components concurrently share data, the GAM and LAM based KeyValuePools, easily outperform the filter-based approach.

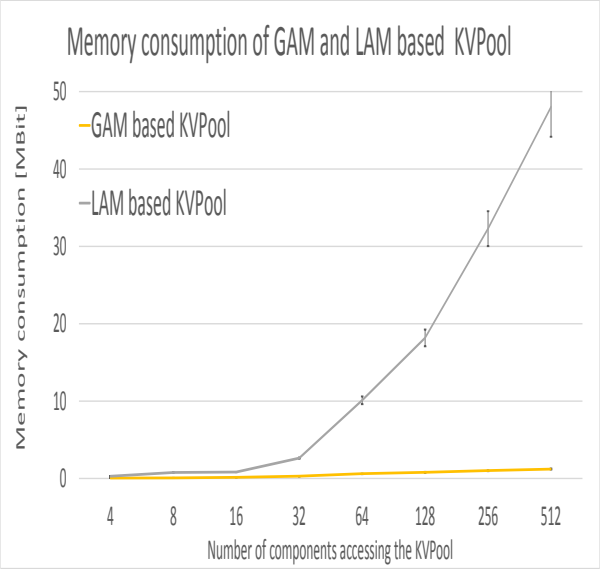
Figure 3.18a shows a comparison of the memory usage with respect to a distributed KeyValuePair with 128 Byte size. Surprisingly, the LAM based KeyValuePool uses less memory than GAM based KeyValuePool implementation, though using multiple COW clones for the concurrent write operations. Consequently, I expected a higher memory usage. However, the LAM based KeyValuePool performs much better due to the LAM. This results in a faster release of retired KeyValuePairs and benefits therefore the overall memory demand of my new approach.



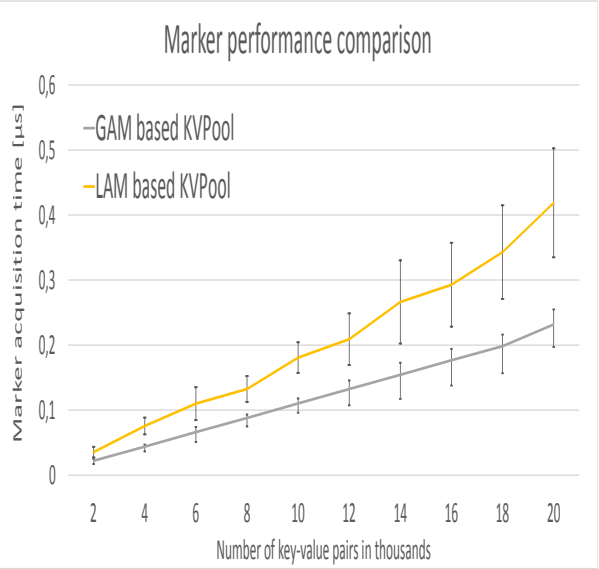
3.17a: Timings for a combined read and write operation with an equal producer consumer distribution for the LAM based KeyValuePool.



3.17b: Timings for a combined read and write operation with twice as much producers as consumers for the LAM based KeyValuePool.



3.18a: Memory consumption of the GAM and LAM based KeyValuePool.



3.18b: Performance gain of the LAM approach with respect to the GAM one.

Figure 3.18b shows the performance gain of the LAM based KeyValuePool with respect to the GAM concept by comparing the timings of hazard pointer acquisition and marker usage for the above stated test cases. In average, the LAM concept performs nearly twice as fast as GAM based approach with hazard pointers. Overall, the LAM concept makes approximately 16% of the overall performance gain. The multiple wait-free writing constitutes 84% of the overall performance boost.

With respect to the overall performance, I can conclude that the LAM based KeyValuePool outperforms the competitors (in average between a factor of 2 and 35) while using less memory than the original approach (on average 74%).

3.7.3 Graph based Nested Hash Map

I conducted different experiments to measure the performance as well as the quality of my GraphPool For the quality measurement, I used the use case scenario described in Section 3.6. However, as the scenario is domain-dependent, it can be hardly used to evaluate the performance of my approach. Hence, I additionally implemented a synthetic benchmark for performance measurements.

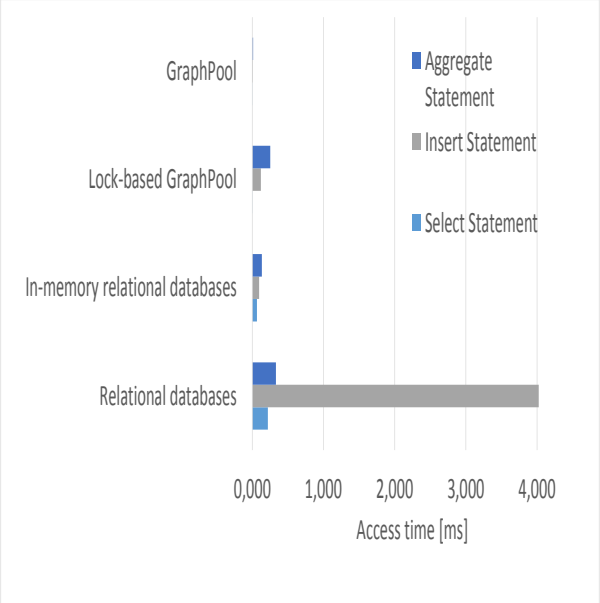
The GraphPool contained 1000 GraphNodes for the synthetic benchmark. I performed 10.000 read-, write- and aggregate queries for each test. Each test was additionally repeated 100 times and I averaged the resulting timings. The access to the GraphPool was modelled with an equal read/write distribution of concurrent system components. The transactions to the GraphPool and its competitors varied in size from 1 Byte to 1 Megabyte. I compared the performance of my new approach with three competitors. The first competitor was a lock-based implementation of my GraphPool. The other two competitors were a relational SQLite database and a MySQL database.

I compared the performance with the traditional on-disk option as well as in-memory resident versions of the databases. Furthermore, I validated if the results from my GraphPool implementations yield the same results as the database competitors.

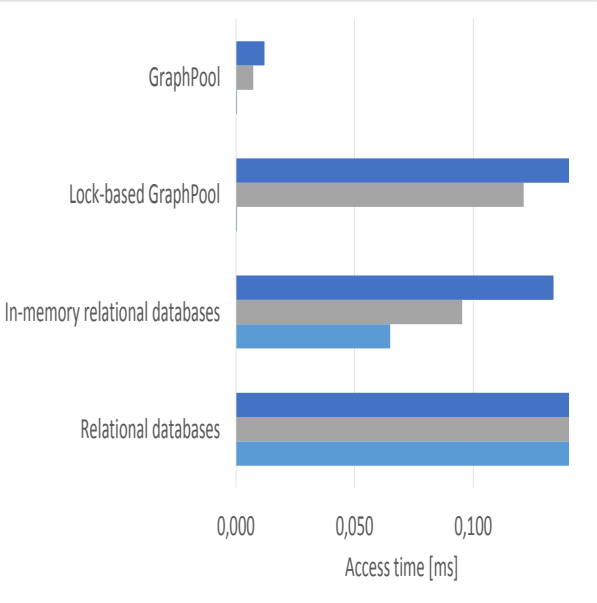
My results show, that, in case of a single component, my wait-free GraphPool outperforms all in-memory and relational databases for every query type in several orders of magnitude. However, the traditional lock-based implementation of the GraphPool is slightly outperformed by in-memory resident relational databases. In this case, the lock acquisition introduces a computational overhead with respect to the wait-free implementation. In addition, standard relational databases can not compete with the in-memory databases and GraphPool implementations (see Figures 3.19a and 3.19b).

This performance gain of my GraphPool increases with an increasing number of simulation components accessing the data management systems. In this case, the wait-free access shows its strengths when several components simultaneously access the GraphPool. Like in the single access case, the in-memory relational database slightly outperforms the lock-based GraphPool implementation. The in-memory and relational databases are again outperformed by the wait-free GraphPool by several orders of magnitude (see Figures 3.20a and 3.20b).

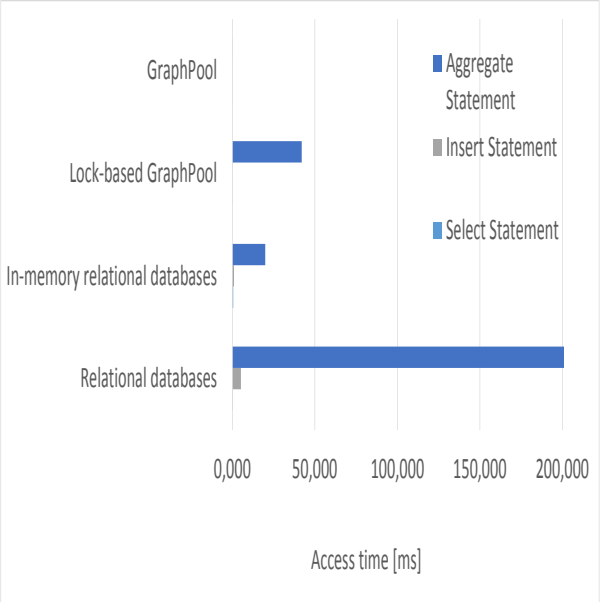
Overall, my evaluation underlines the aforementioned technical limitations of current 3DST applications which rely on relational databases: The relational databases scale not very well with many concurrent components accessing them. Furthermore, my evaluation shows that the hash map backbone of my GraphPool can effectively solve the problems of the rigid table format of relational databases because no transformations of object-oriented data into tables is necessary. Additionally, the wait-free access of the GraphPool improves the overall performance even for massive concurrent read and write operations. In summary, my approach improves the overall system performance of 3DST applications by several orders of magnitude in all access query cases.



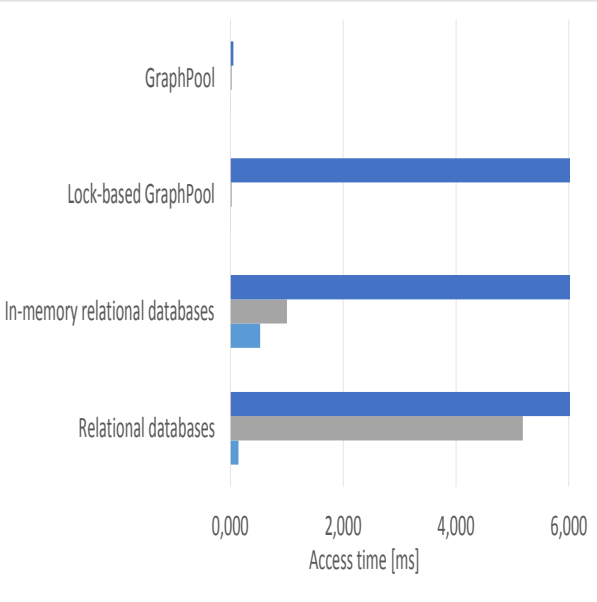
3.19a: Performance comparison of core & aggregate queries for single access scenarios: overall, my GraphPool outperforms all competitors for all query types for multi-component access.



3.19b: Detailed view from the same evaluation: in-memory resident relational databases outperform the traditional lock-based implementation of the GraphPool.



3.20a: Performance comparison of core & aggregate queries for multi access scenarios: overall, my GraphPool outperforms all competitors for all query types for multi-component access.



3.20b: Detailed view from the same evaluation: in-memory resident relational databases outperform the traditional lock-based implementation of the GraphPool.

Pure mathematics is, in its way, the poetry of logical ideas.

Albert Einstein

4

Concepts for Generative Virtual Testbeds

In this chapter, I introduce my novel approaches for generative virtual testbeds. They are based on my wait-free hash maps, the KeyValuePool and GraphPool, within the ECS pattern and a novel DSML. I recall some of the most relevant research articles that have appeared in the international literature related to this topic and emphasize my contributions.

As introduced in the previous chapters, a central part of RISs is the generation, management and distribution of the global simulation a.k.a world state. Usually many independent software components need to communicate and exchange data in these systems in order to generate this global state. These components and their corresponding performance within an RIS are governed by the functional as well as non-functional requirements of typical RIS development such as (realtime) performance, responsiveness, scalability, consistency and (re-)usability [151, 155]. Consequently, RIS research and development strives for reusable patterns and software architectures in order to increase the satisfaction of the above mentioned requirements, especially for massive amounts of RIS software components [117, 140].

Hence, diverse approaches have been presented to tackle this kind of challenge. In the past, the ECS pattern has become a major design pattern used in modern architectures for RIS [56]. This pattern strives for high re-usability and architectural scalability. The main idea of ECS is to decouple high-level modules such as physics, rendering or sound from the low-level objects with their corresponding data. Therefore, ECS introduces three software architectural objects: `Entities`, `Components` and `Systems`. These are used to describe objects of a RIS via composition instead of object-oriented inheritance. Many advantages arise when using ECS. For instance, the behavior of `Entities` can be changed at runtime by adding or removing `Components`. Moreover, `Systems` are likewise interchangeable as they are not part of the `entity` nor `Component` implementation. This allows even the quick swap of e.g. a physics engine. This leads to the elimination of ambiguity problems of wide and deep inheritance hierarchies, often encountered in traditional RIS development. As a result of this, the ECS pattern and variations of it have been applied to many RISs [55, 56, 117, 119–121, 140, 155]. However, the ECS pattern does not aim at satisfying the performance requirement of RIS architectures as it does not specify any low-level implementation.

For instance, the System access implementation to the Components is not defined. Usually, every System iterates over a container (e.g. an array) of all Entities and applies the Systems behavior on the corresponding Components. In fact, modern RIS can consist of hundreds or thousands of Components and Systems. Therefore, an access parallelization (e.g. in the form of threads or OpenMP[1] support) is necessary in order to maintain realtime performance of the whole application. Consequently, this container of Components (and therefore every Component) has to be implemented as a shared data structure. Such shared data structures can quickly become bottlenecks of modern RIS applications as they typically use crucial software synchronization patterns such as mutexes, semaphores and consequently synchronization which can lead to thread starvation or system deadlock [150]. The ECS pattern does not cover any guidelines or specifications for effectively solving this problem but my previously described wait-free hash maps, the KeyValuePool and GraphPool, can be integrated into the ECS pattern.

As stated before, the KeyValuePool and GraphPool guarantee access to a shared data structure in a finite number of steps for each System (eg. as a traditional thread or OpenMP implementation), regardless of other System accessing the shared data structure.

My contribution is an extension of the ECS pattern with my high performance wait-free hash map with efficient memory management which enables low-latency, massively parallel access within the pattern and, additionally, reduces its memory consumption. Hence, my contribution allows non-locking read and write operations of Systems, leading to a highly responsive low-latency data access while maintaining a consistent state even for structured Components. Simultaneously, my contribution greatly reduces the memory footprint of my wait-free hash maps by introducing novel garbage collection techniques. My novel approach is easy to implement and it fits perfectly into the implementation of wait-free hash maps without altering the ECS pattern itself. My approach therefore greatly benefits the overall RIS performance. This integration into the ECS pattern further enables a high cohesion of the KVInterfaces as Systems in the ECS pattern. In addition to the previous contributions of the hash map based data management, it C&C. Even more, I show how above approach can be further improved with the concept of MDE, namely DSMLs. My ECS pattern based wait-free hash map concept can be easily modelled with a DSML, leading to efficient definition and implementation of virtual testbeds via code generation.

4.1 Related Work

Research in increasing scalability, maintainability, re-usability and performance as well as managing concurrency within RIS frameworks has attracted increasing interest in the last decade [55, 56, 119–121]. This research can be broadly classified into two classes: high-level and low-level concepts. High-level concepts describe the overall RIS software architecture in terms of software classes which use standard libraries for solving parallelization and concurrency of the low-level implementation. Examples for such high-level concepts are Simulator X [117] with its actor model [183] or other high-level approaches, e.g. based on dataflows [194]. Further comprehensive high-level work in the area of reusability, scalability for RIS was carried out by [55, 56, 119–121]. My work is complementary to this work because my concepts improve the low-level communication with the wait-free hash map concept. This means that the high-level concepts can be linked with my work and that the achieved advantages can be used in combination with my work, too. Further, [31] gave an overview of high-level architectures aiming at solving consistency and concurrency for CVEs. However, this overview neglected wait-free synchronization approaches.

[1] OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, processor architectures and operating systems.

In contrast to these high-level concepts, low-level concepts investigate programming language specific implementations of synchronization approaches for RIS challenges. In the past, low-level concepts mainly introduced lock-based CCM approaches for RIS, VR, and CVE frameworks in order to efficiently implement the high-level concepts, i.e., [86].

A distinctive characterization of CCMs is whether they are locking or non-locking (see Chapter 2.1). To recap, locking approaches allocate resources exclusively by using various well-studied techniques such as mutexes, semaphores or condition variables. A main advantage of locking CCMs is that they avoid race conditions and naturally guarantee consistency of the system.

Many traditional RISs, especially CVEs, such as [28, 201] used lock-based approaches until [201] reported that the locking approach scales only to at most ten components. This is mainly because of the problem that concurrent threads have to wait until a resource has been released. This may result in a loss of efficiency because problems like thread starvation or deadlocks can occur. Consequently, more modern CVEs like [63, 142, 145] tried to avoid this problem by extending the basic locking mechanism, e.g. by a first-come-first-serve locking [145]. Further, more sophisticated concurrency control approaches introduced fine-grained locks per object for single-write and multiple-read operations [44, 45, 95]. Due to the limitations of lock-based approaches, I proposed a wait-free approach based on hash maps for RIS (see Chapter 3).

Wait-free approaches guarantee access to the shared data structure in a finite number of steps for each thread, regardless of other threads accessing the shared data structure by introducing a few atomic operations [122]. This means that these approaches do not need any traditional locking mechanism in order to preserve a consistent data state. My extensive evaluations have shown a superior performance of wait-free approaches with respect to traditional locking approaches (see Section 3.7). My wait-free approaches not only support structured `Components` such as arrays or lists but also use fast hash key operations in order to find and retrieve the stored `Component` inside the used hash table. Due to their excellent scalability, they are perfectly suited for RIS frameworks which need to support massive amounts of `Systems`, such as MAS based VR and RIS applications [152, 209]. Consequently, using wait-free data structures as a data access backbone can highly improve the performance and scalability of RIS frameworks.

However, my wait-free hash maps come at a cost: In order to achieve wait-free behavior of read and write operations, I use double-buffering for write operations. This means that every write access on the shared data structure is preceded by a double-buffering which clones the data [150, 151]. All ongoing read operations can still access the old data state while new read queries are directly routed to the new (manipulated) data state.

Therefore, after a given timespan, the old data will not be used any more. When all read operations on the old data state are finished, the data is released. Hence, the amount of cloned data directly responds to the amount of write operations (see Sections 3.3 and 3.4).

In the following sections, I will describe the integration of double-buffered wait-free hash maps into the ECS pattern. Furthermore, I will describe my novel memory management for these data structures which reduces their memory demand greatly. Therefore, my integration and novel memory management overcomes the limitations of the presented related work.

In addition, advancements in software engineering can improve the development of sophisticated virtual testbeds. MDE allows aspects of RISs, specifically virtual testbeds, to be represented formally as an abstract graphical model which can be automatically transformed into software artefacts and subsequently into complete RIS applications.

MDE enables domain experts through a DSML to produce RISs for arbitrary scenarios easily and quickly, as MDE notably promises great benefits to its practitioners. From a software development context, MDE offers an increase in productivity, promotion of interoperability and portability among different technology platforms, support for generation of documentation, and easier software maintenance [7].

In addition, it can also lead to production of better code quality and reliability due to integration of domain rules into the DSML. Such domain rules minimize modelling errors and increase the reliability of mapping from model to code [178], which is highly desirable for researchers, engineers and industry. Consequently, a DSML can decrease the development time and increases overall comprehension of simulation and optimization aspects of my virtual testbeds.

4.2 Wait-Free Hash Maps for the Entity-Component-System Pattern

As previously outlined in the introduction, RIS research and development strives for reusable patterns, for instance the ECS pattern. As outlined in the introduction, this pattern has become a major design pattern used in modern architectures for RIS. However, the ECS pattern does not aim at satisfying the performance requirement of RIS architectures as it does not specify any low-level implementation. This leads to the problem that the container of `Entities` becomes are heavily shared data structure [155]. In order to solve this concurrent data access, I propose my KeyValuePool and GraphPool which deliver high performance access even for massive numbers of concurrent read and write operations.

However, as a drawback they accompany a large memory footprint because they rely on a double-buffering approach with atomic operations in order to achieve wait-free behavior. This double-buffering creates for every write access to the shared data structure a clone of the manipulated data. When all read operations on the cloned data are finished, the cloned data is released. Hence, the amount of cloned data directly responds to the amount of write operations (see Chapter 2.1).

I present a novel solution to this challenge; my ECS based approach allows concurrent read- and write access even for highly data driven RIS applications (resp. RIS applications which inherit many `Components` and/or `Systems`). Moreover, it can even handle multi-modal RIS applications in which different `Systems` interact with each other in different frequencies.

In detail, my contribution is

- an extension of the ECS pattern for high performance double-buffered wait-free hash maps (KeyValuePool, GraphPool) with
- centralized as well as decentralized approaches for efficient memory management of these data structures which greatly reduces their memory consumption.

My contribution allows non-locking read and write operations of `Systems`, leading to a highly responsive low-latency data access while maintaining a consistent global state even for structured `Components`. Simultaneously, my contribution greatly reduces the memory footprint of my KeyValuePool based data management. My novel memory management is easy to implement and it fits perfectly into the KeyValuePool / GraphPool implementations without altering the ECS pattern itself. Thus, my approach greatly benefits the overall RIS performance.

4.2.1 The Entity-Component-System Pattern

In the past, the ECS pattern has become a major design pattern used in modern architectures for RISs [56]. The main idea of ECS is to decouple high-level modules such as physics, rendering or sound from the low-level objects with their corresponding data. Therefore, ECS introduces three software architectural objects: `Entities`, `Components` and `Systems`:

- The `Entity` is a general purpose object which is usually defined as a unique id. These `Entities` can be further described via composition of `Components`.
- The `Component` is the raw data for one aspect (e.g. a position, velocity or sprite) of general purpose objects.
- The `System` performs global actions on every Entity that possesses a `Component` with the same aspect as that `System`. Each `System` thereby runs continuously (e.g. as a thread).

These concepts are used to describe objects of a RIS via composition instead of object-oriented inheritance. The traditional way to implement simulation or game objects within RISs was to use object-oriented programming. Each object was modelled and implemented within a typical class hierarchy which intuitively allowed for an instantiation of these classes. This enabled simulation or game objects to extend to other objects through polymorphism. However, with an increasing complexity of the RIS, this leads to large, rigid class hierarchies.

These wide and deep hierarchies become consequently increasingly difficult to maintain. In addition, placing a new simulation or game object into the hierarchy is further complicated if the object needs a lot of different types of functionality from different domains. Figure 4.1 illustrates this limitation in a game-based RIS scenario. Usually, the conflicting code is then moved to the base class which results in super classes. These super classes gradually decrease the maintainability and scalability of the overall RIS architecture [155].

Typically, these deep and wide inheritance structures can be vertically decomposed with the ECS pattern (see Figure 4.2). This allows greater flexibility and adaptability in defining simulation or game objects (e.g. vehicles, sensors, enemies, etc.) as every object is an `Entity`. Every `Entity` consists of one or more `Components` which add aspects (e.g. position, velocity, sprite, etc.) to the `Entity`. Within this context, the behavior of an `Entity` can be changed at runtime by removing or adding `Components` [155].

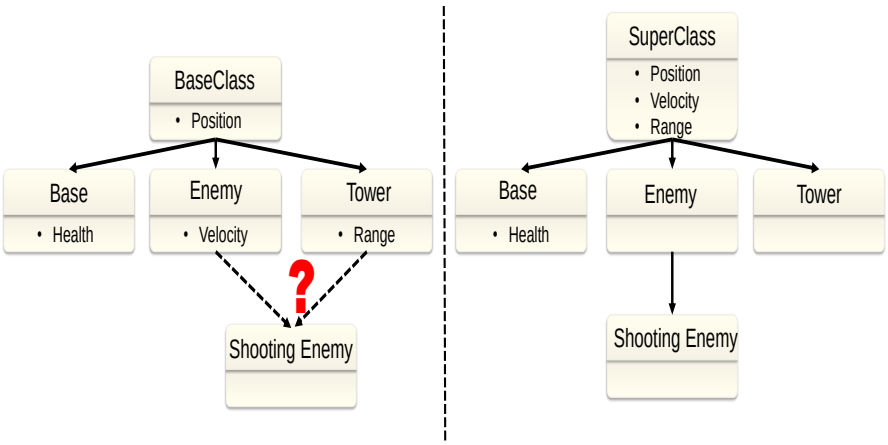


Figure 4.1: Evolving problems in inheritance based object design in RIS applications: In order to preserve class-wise consistency, super classes are constructed. These super classes degenerate maintainability and reusability of the whole application.

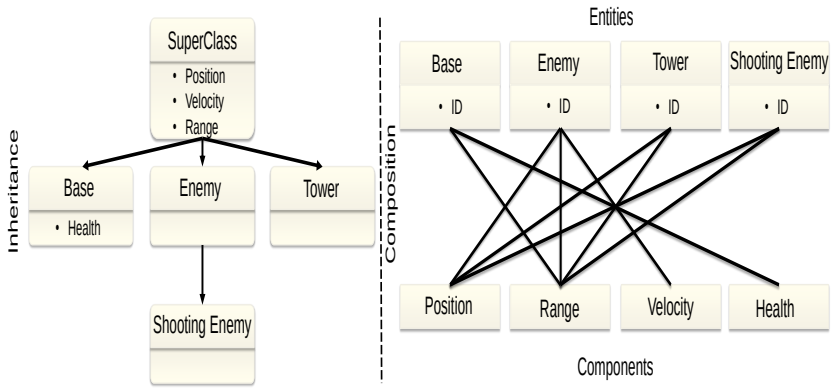


Figure 4.2: Deep and wide inheritance structures arise in traditional RIS development that is based on object-oriented design (left). The ECS pattern decouples the data and algorithms via composition into Entities and Components (right).

Furthermore, even `Systems` are decoupled as each `System` applies its computation on the same `Component` types which are referenced via `Entities`. As an example, think of a physically-based simulation for gravitational forces: the corresponding `System` will apply Newton's law of gravity every time on the same `Component` types (position and velocity) but it does not concern any more properties of the `Entity`.

As a consequence, even `Systems` can be easily added, removed or changed as they are not part of `Entity` or `Component` implementation. Actually, object-oriented problems of deep and wide inheritance structures as mentioned above are eliminated.

4.2.2 Integration of Wait-Free Hash Maps

As previously stated, the ECS pattern does not specify the low-level implementation of the `System` access to the `Components`. In this section, I demonstrate the applicability of my wait-free hash maps for this `System` access to the `Components` as they promise high performance even for massive numbers of concurrently acting `Systems`.

My wait-free hash maps (KeyValuePool and GraphPool) can be easily integrated in to the ECS pattern as follows: All `Components` (which can represent primitive or structure data) are stored inside the wait-free hash map. These `Components` are accessible via a unique key which is generated for every `Component`. All `Systems` and `Entities` can refer to these `Components` via their unique keys which retrieve the `Component` from the hash map. Every `Entity` has a list of keys which defines the `Component`-wise composition. Adding and removing `Components` from the `Entity` are implemented as insertion and deletion operations on this key list. Every `System` iterates over the `Entities` and uses the stored keys in order to retrieve the corresponding `Components` for computation. This integration of wait-free hash maps does not alter the original ECS approach (see Figure 4.3). Overall, the `System` access to all `Components` is implemented straightforward, for instance with OpenMP support (see Algorithm 4.1). Note, that no locking operations are needed in order to maintain consistency of the hash map and consequently of all `Components`.

The `Component` access of the wait-free hash map is implemented in accordance to the previously described double-buffering approach (see Sections 3.3 and 3.4): Every `System` can retrieve a `Component` from the hash map by looking up the corresponding key. Within the double-buffering approach, every `Component` is stored as a producer and consumer version in the hash map.

For all read operations, the hash map returns a pointer to a dedicated consumer version of the `Component`. Consequently, all read operations work on the same memory as they can not affect each other. All actively reading `Systems` notify their access by incrementing (read operation starts) and decrementing (read operation has ended) an atomic marker of the consumer version. For write operations, `Systems` retrieve a producer version of the `Component` which is a different memory object than the consumer version. After modifying a `Component`, a `System` can notify its changes to all other `Systems` by storing the `Component` back to the hash map. This write process uses the aforementioned double-buffering and returns the cloned `Component`. In detail, the whole write operation of a `System` can be decomposed into six steps: A `System` wants to modify a `Component` which is stored inside the hash map.

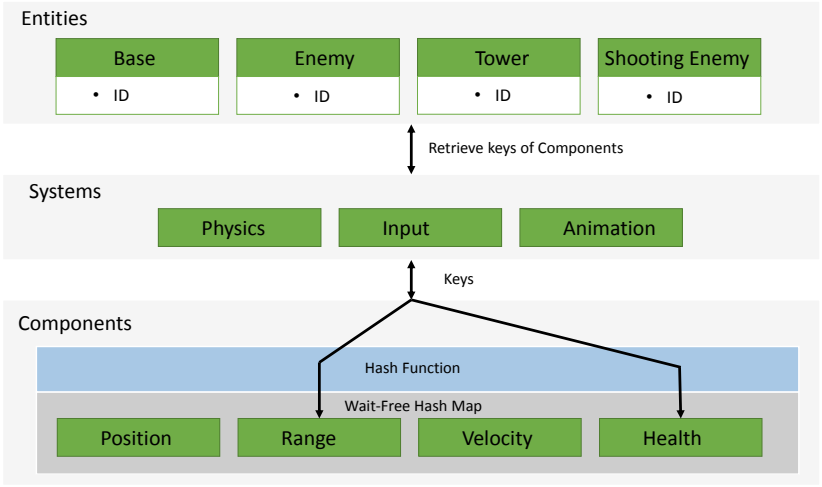


Figure 4.3: The ECS pattern does not specify a high performance access of *Systems* to *Entities* and *Components*. The integration of my wait-free hash map concept into the ECS pattern enables high throughput. For integration, all *Components* reside inside my hash map which is concurrently shared by all *Systems*. The access is managed via unique keys that also belong to the ECS required IDs of *Components*.

To do that, the corresponding hash map access returns the producer version of the `Component`. The `System` can then modify the `Component` and stores it back to the hash map. In order to notify all other `Systems` about these modifications, this write operation creates automatically a clone of the producer version which is used as the new consumer version. All concurrent read operations are routed to the old consumer version as long as the actual write operation of the hash map lasts. When the new consumer version is available, all concurrent read operations are routed directly to the new consumer version.

```
// Define OpenMP parallelization with n threads
#pragma omp parallel for num_threads(n)
for ( auto Entity in Entities )
{
  if (Entity.Type == System.Type)
  {
    ComponentKeys = Entity.WriteKeys
    for ( auto CompKey in ComponentKeys )
    {
      Component c = Hashmap.get(CompKey)
      System.ApplyComputation(c)
      Component clone = Hashmap.set(c)
      // Delete clone after all concurrent
      // read operations have finished
    }
    ComponentKeys = Entity.ReadKeys
    for ( auto CompKey in ComponentKeys )
    {
      Component c = Hashmap.get(CompKey)
      c.incrementReadMarker
      // Use data as long as needed
      // without altering it
      // ...
      c.decrementReadMarker
    }
  }
}
```

Listing 4.1: System access on Entities and Components in C++ pseudocode

In the meantime, the old consumer version is not deleted to prevent memory failures. When all ongoing read operations on the old consumer data are finished, the corresponding memory will be deleted, hence completing the double-buffering principle. Parallel write operations are merged when required (see Section 3.4). Figure 4.4 illustrates this double-buffering approach. The main challenge remains the efficient release of the old `Component` data which is described in the next section.

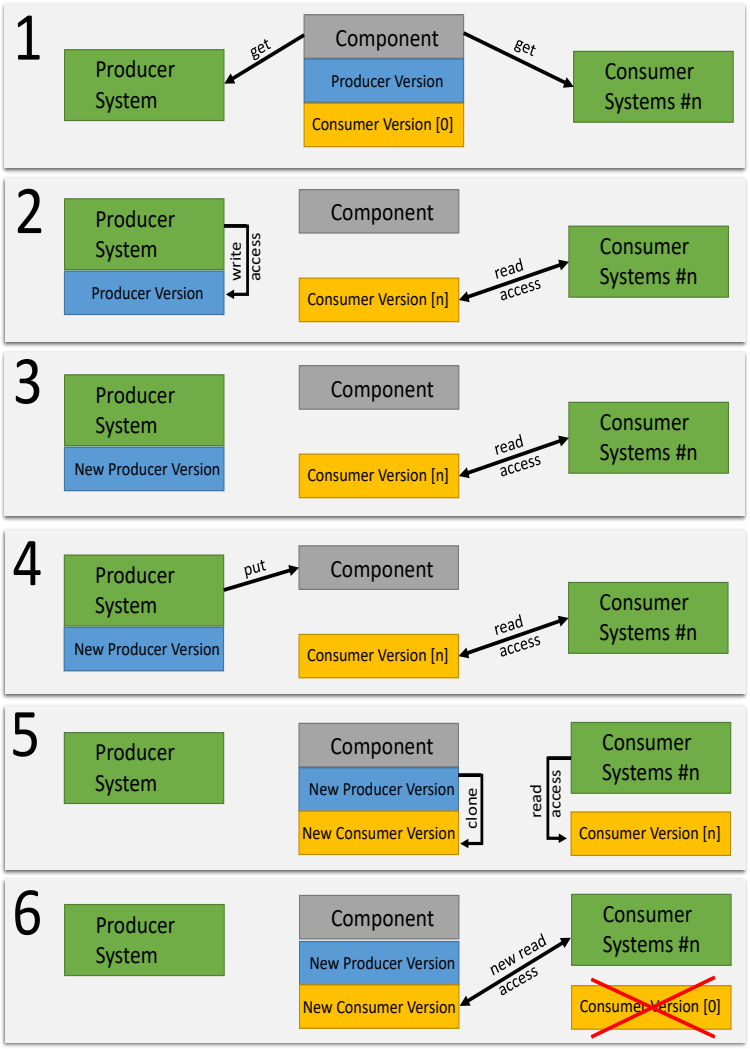


Figure 4.4: The double-buffering approach applied to the ECS pattern, simplified into six steps: Every write access is preceded by a cloning process of the *Component* data. When all parallel read operations are finished, the old data is deleted. In detail, 1) producing *Systems* retrieve the producer version and (n) consuming *Systems* retrieve the consumer version. 2) parallel write access is done without any locking while read operations may take place. 3) read and write operations may overlap or not. 4) as soon as the producing *System* has finished, the data is updated regardless of any read operations. 5) the consumer version is updated, while the consuming *Systems* operate on their local copies. 6) new read operations of consuming *Systems* are directly routed to the new data and the old local copy is freed.

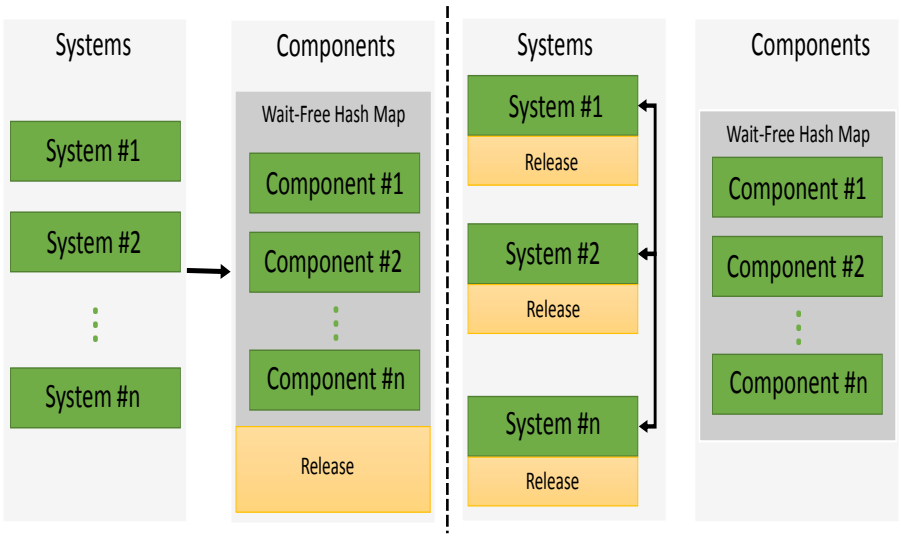


Figure 4.5: The centralized and decentralized memory management approaches: In the centralized approach, every *System* notifies the central hash map via atomic markers. The hash map itself releases the memory in either periodic or threaded implementation (left). In the decentralized approach, all *Systems* notify themselves about ongoing and finished read operations. In addition, every *System* itself takes care of releasing its cloned *Component* data (right).

4.2.3 Memory Management of Wait-Free Hash Maps

In this section I present centralized as well as decentralized approaches for the memory management of cloned `Component` data in wait-free hash maps for the ECS pattern. First, I present two centralized approaches based on periodic memory release as well as threaded memory release. These approaches are located in the central hash map itself and rely on finished read notifications that are triggered by all `Systems`. Second, I present a decentralized approach which defines an individual memory release per `System`. In this case, every `System` itself takes care of releasing unused `Components`.

All approaches follow the presented main principle of LAM (see Section 3.4): all `Systems` notify each other when they read or write data via notifications. This notification is implemented as an LAM which is increased when the read access begins and which is decreased when the read access has finished. Consequently, if this LAM is zero, the data can be safely deleted. Note that the proposed integration can also be implemented with the GAM approach.

The basic challenge is the management, i.e. the saving and the efficient release, of the generated `Component` clones. The nature of RIS applications leads to different generation, update and deletion frequencies of different `Components`. For instance, a collision detection query is updated at 1000 Hz and an animation of a scene node is played when a certain event has triggered in the RIS. In these cases the resulting `Components` (e.g. the resulting collision volume and rotation matrix) are more frequently or rarely updated.

Consequently, different `Components` types (e.g. player-input, physically-based simulation results or animators) are more frequently updated than other `Components` types. This means that also the corresponding clone data is more frequently generated. In this case, it is desirable to handle the memory release per `Component` type. This leads to my approach which introduces `Component`-wise queues for storing the cloned `Component` data per `Component` type. These queues basically split all cloned `Components` into smaller chunks which can be faster checked than a single container for all cloned `Components`. Every cloned `Component` data is directly sorted into the corresponding queue immediately after its creation.

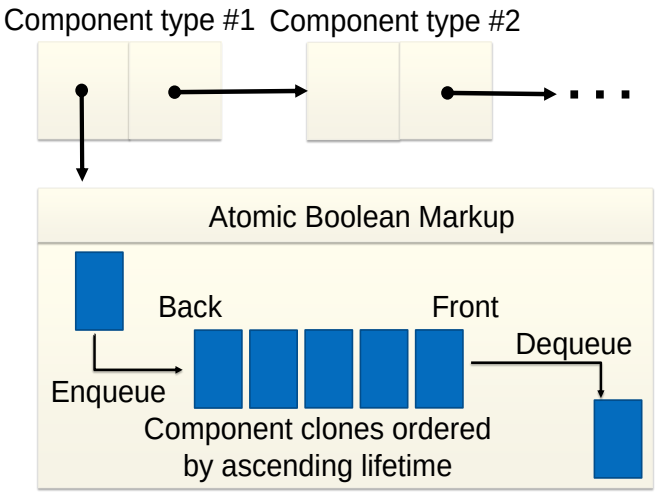


Figure 4.6: The *Component*-wise queue concept: for every *Component* type a corresponding queue of cloned *Component* data is created. This avoids searching unstructurally the cloned data. Instead, my memory management can now iterate for every *System* only the corresponding queues of related *Component* types. This speeds-up the search for deletable memory immensely.

Additionally, every queue itself has a atomic boolean markup. Every time a `System` notifies a finished read operation (by decrementing the corresponding LAM of the `Component`), this queue markup will be set to true. The marker will be set to false when the release function has finished its checks (see Algorithm 4.2). This release function iterates over all queues and checks their boolean markers. If a queue marker is set to true (resp. a `System` has finished its reading operation on a cloned `Component` data within the queue) it will further iterate the `Component` queue. After a queue check, the release function will set the corresponding queue markup to false (see Figure 4.6). Algorithm 4.2 shows how these `Component`-wise queues can be iterated in order to efficiently release the unused cloned `Component` data. The efficient iteration over these `Component`-wise queues remains challenging. As mentioned above, I present two approaches for tackling this challenge: centralized and decentralized.

```
for ( auto CompQueue in ComponentQueues )
{
  if (CompQueue.Markup == true)
  {
    for (Component c = CompQueue.peek();
         CompQueue.size() > 0;
         c = CompQueue.peek())
    {
      if (c.ReadMarker == 0)
      {
        CompQueue.pop()
        delete c
      }
    }
    CompQueue.Markup = false
  }
}
```

Listing 4.2: Cloned Component data access via queues in C++ pseudocode

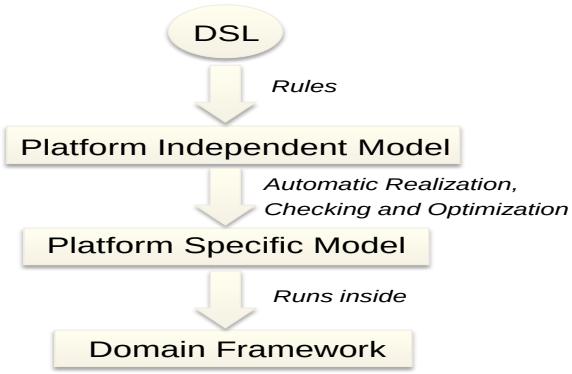


Figure 4.7: The main concept of my generative approach: a DSML defines generic PIMs for specific use-cases of virtual testbeds in XML. These PIMs enable the automatic generation of PSMs for specific computing platforms via automatic realization for the programming language C++. The required domain framework for these PSMs are based upon the previously presented ECS pattern with my hash map concept.

The centralized memory management approach is implemented in three variations: periodic, continuously threaded and on-demand threaded. In all cases, the memory management is located within the central wait-free hash map and all `Systems` notify the hash map with respect to the above mentioned atomic memory markup. The continuously threaded approach implements a complete separate thread that runs constantly in parallel within the RIS and checks the queues as shown above. The on-demand threaded approach implements a complete separate thread that is only activated when one `System` has finished its operations and generated new cloned data. After one check for all queues, it goes back to sleep state until it is notified by another `System` again. The periodic centralized approach is called in accordance to the `System` frequencies. `Systems` usually apply their computations to the `Components` periodically: The physically-based simulation (e.g. collision detection) runs traditionally at 1000 Hz while animations require 30 Hz or 60 Hz. Hence, it is also favourable to directly couple the frequency of the memory release with the actual implemented `System` frequencies of the RIS. I propose three frequencies for the periodic memory release: First, the memory release can be performed at the slowest frequency of all `Systems`.

Second, the memory release can be performed at the fastest frequency of all `Systems`. At last, the memory release can be performed at the average frequency of all `Systems`. The periodic approach builds upon application domain knowledge. For instance, if it is known to the RIS developer that mostly fast-paced physically-based simulations are present in the application, also a rapid periodic check for memory release could be useful and vice versa. In contrast, the decentralized memory management is located within the `System` implementation. All `Systems` notify each other with respect to the above mentioned atomic memory markup. Every `System` actively checks on his own after each completed computation whether memory can be deleted or not (see Figure 4.5).

4.3 Domain Specific Modelling for ECS based Virtual Testbeds

In this section I present my novel comprehensive approach to modelling, simulation and optimization of arbitrary simulation models within virtual testbeds. This approach encompasses several contribution of my work, namely my KeyValuePool and GraphPool based ECS software infrastructures for RIS based applications, MAS based optimization (see Chapter 6), and KDPs for SBO (see Chapter 5). My approach is based upon the concept of a DSML, a software engineering methodology for designing and developing complex systems.

It represents the various facets of a system in a PIM. From these PIMs, PSMs are generated via code generation techniques (see Figure 4.7).

Enabling SBO applications require a virtual simulation environment for the parameter optimization to take place. In this section, I will describe my simulation environment as well as domain framework and how both are related to my DSML. My modelling approach is based on the lightweight DSML [1] approach. It introduces more generalized model artefacts to be used in order to decrease the overall development time when using DSML concepts. Generalized model artefacts do not support complete source code generation as their modelling concept focusses mainly on the dataflow. However, for a generic virtual testbed, which should support highly different simulation models or even other domains, complete source code generation is no prerequisite.

My lightweight DSML approach is used to define generic components of a simulation. For instance, a robot or craft simulation should cover the environment, craft sensors and actuators, internal craft control components, and the corresponding three-dimensional representations. These components are situated in a simulation model loop which ensures that the craft can only perceive its environment via its sensors and that only actuator information are routed to the environment. My lightweight DSML enables modelling of such components in a straightforward manner as described in the next sections. My approach enables domain experts to easily model generic PIMs for various simulation applications. These PIM are then used to automatically generate the source code of the corresponding virtual testbed, the PSM, via horizontal model transformations [178]. This contribution enables an efficient development of a virtual testbeds based SBO. Even more, my approach model checks the given simulation model for interface (dataflow and workflow) errors. Thus, this leads to less errors in the development of virtual testbeds. It utilizes my KeyValuePool and GraphPool based software infrastructure for massively parallel execution of simulation and optimization. In addition, it uses my MAS based optimization (see Chapter 6) and automatically integrates my KDP for SBO studies (see Chapter 5). Within my DSML approach, the overall development time of the virtual testbed is greatly reduced because

- it allows for quick modelling of the simulation and optimization modules within the virtual testbed,
- it enables code generation for arbitrary PSMs from the specified PIMs,
- it conducts a complete model checking for dataflow and workflow of the simulation in order to detect possible errors from the given simulation requirements and architecture.

Consequently, my DSML represents high-level modelling concepts for expressing SBO related functionality while providing model checking within my high performance wait-free software infrastructure (see Figure 4.8).

4.3.1 Domain Framework and Dataflow

Within my DSML approach, a domain framework is required to execute the generated code artefacts. Such a domain framework additionally reduces the amount of generated software clones as the domain framework encapsulates common interfaces and data structures. This approach is well-known from other applications, such as the Java Runtime Environment [169], and incorporates many advantages.

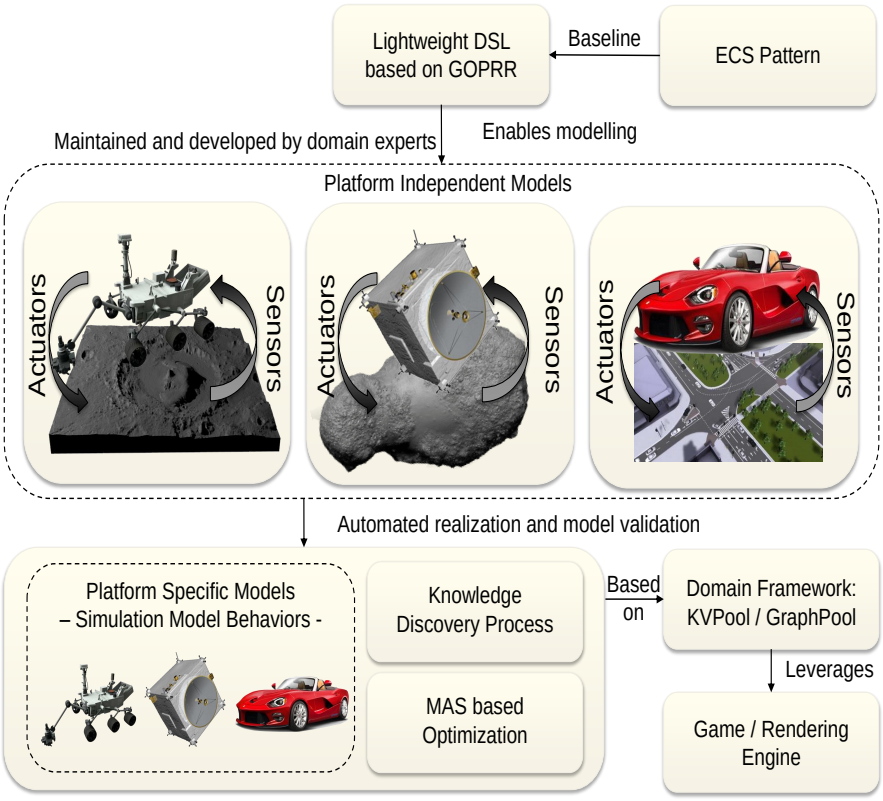


Figure 4.8: Overview of my proposed approach. From a lightweight DSML, arbitrary domain-unrelated PIMs for specific virtual testbeds are defined which are used to generate the corresponding actual virtual testbed implementation (PSM) with my proposed SBO methodologies in C++. The used domain framework is my previously described hash map based data management. The baseline, the ECS pattern, is also used to define the descriptive objects of the lightweight DSML in order to enable a efficient implementation of the overall code generation process.

The required domain framework should be a solution that satisfies state-of-the-art software engineering advantages from current virtual testbed approaches [126, 133] but should not introduce a performance bottleneck to the overall simulation and optimization (see Chapters 2.1 and 3).

I presented in the previous chapters my wait-free software infrastructure concept based on KeyValuePool and GraphPool with its extensive benefits and contributions. Consequently, I use my KeyValuePool and GraphPool as the domain framework. The domain framework is responsible for driving the simulation and optimization in a consistent manner. I describe in the following how a discrete event simulation (DES) can be implemented within the proposed domain framework. In addition, I describe the integration of the ECS pattern within the DSML and the incorporation of SBO concepts.

The core of the proposed DSML approach is the domain framework, namely my KeyValuePool and GraphPool based software infrastructure. Therefore, every domain component that is defined in the PIM is a `KVInterface` (resp. a `System`), as introduced in Chapter 3. Consequently, every domain component benefits from my proposed wait-free, ECS based software infrastructure.

In addition to this wait-free access, the KeyValuePool and GraphPool software infrastructure delivers a homogeneous interface for accessing the simulation state as well as for `KVInterface` communication. These relationships are defined by a set of KeyValuePair read and write operations. The read and write operations are easily represented by two sets of the correspondig `Component` keys which indicate the producer and consumer of the data.

This encapsulation leads to simpler DSML concepts because the code generation for accessing simulation states and for simulation component data exchange can be represented by simply delivering these two sets of keys per domain component.

In contrast to my straightforward approach, current virtual testbed data managements use full-fledged SQL databases [36, 126, 205]. These approaches introduce additional complexity to the code generation process because complete SQL-queries would need to be generated for accessing the simulation state but also for domain component interaction. Consequently, my software infrastructure delivers not only fast low-latency data access performance but, even more, high software cohesion which facilitates efficient code generation of the overall virtual testbed.

Within this concept, I describe a DES loop of my domain framework as follows: A state $S = (t, I, P, E)$ of my virtual testbed consists of simulation time t and sets of `KVInterfaces` I, KeyValuePairs P and events E. Additionally, a transition function δ is defined:

$\delta(t_n, I, P, E) = (t_{n+1}, I, P', E')$

In each transition, the KeyValuePairs (resp. the GSS) are updated by the `KVInterfaces` and new transition events are generated by the system. In order to enable a generic modelling concept for arbitrary simulation models, `KVInterfaces` can be grouped into various groups which can be defined within the DSML. These groups define synchronization barriers between the `KVInterfaces`, if required. Within each group, all domain components run completely in parallel, for instance as a OpenMP implementation (see Chapter 4.2.2).

As an example of above concept, domain components for a vehicle or robot based SBO application could incorporate these five groups (see Figure 4.9):

- Environment: Defines simulation aspects from the environment in which the craft is situated. These aspects can incorporate physical forces such as gravitation, air drag, pressure, kinematics or even Virtual Reality based approaches like collision detection.
- Interface: Defines the sensors and actuators of a craft, e.g. thrusters, gyroscopes, cameras or rangefinder sensors.
- Craft: Defines internal craft components such as control loops, localization concepts, sensor fusion algorithms, BDI-structures or target detection.
- Optimization: Defines a SBO for craft related configurations.
- Simulation Analysis: Defines the KDP properties for analyzing the simulation model behavior.

4.3.2 Domain Specific Modelling Language

My DSML is based on the industry MetaCase+ graphs, objects, properties, relationships, roles (GOPRR) notation [84]. All objects used within GOPRR are drawn into graphs that contain the objects role and the relationships thereof. GOPRR objects can be for example a process, a thread, a class or an instance of class. A property describes features of graphs, objects, roles and relationships. A relationship connects objects by assigning them roles in the activity of the object. The aim of my DSML is to describe all required SBO aspects, namely simulation model, simulation model analysis and MOO, within GOPRR. Using GOPRR has the major advantage that there are existing applications for the creation of GOPRR based languages.

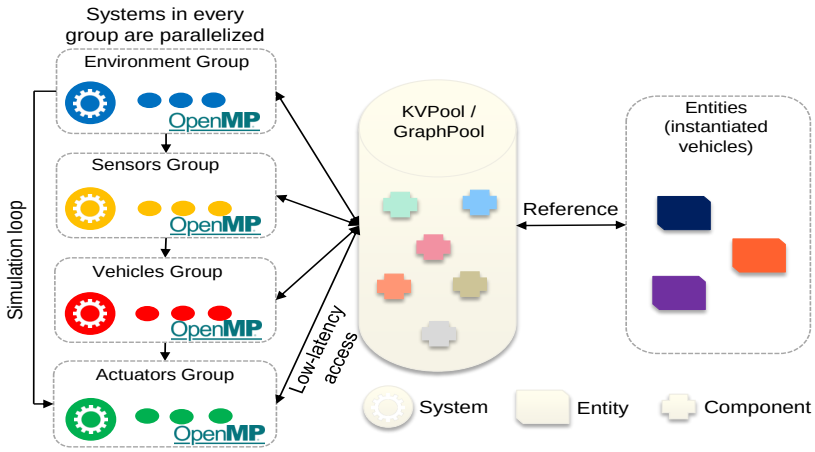


Figure 4.9: Generated vehicle simulation example: synchronization groups (e.g. a classical sensor-actuator loop here) are mapped to my ECS based software infrastructure with wait-free CCM. Within each synchronization group, all domain components (Systems that apply operations on instantiated vehicles as Entities) run in parallel (OpenMP parallelization) with high throughput based on my wait-free hash map concept.

Therefore, the required functionality is defined in the GOPRR notation: Graphs in my DSML are synchronization groups which can belong to either simulation, optimization or visualization. Every graph contains a set of objects, domain components, namely `KVInterfaces`. Every object involves three main concepts: a role, relationships and properties. There are arbitrary roles which can be defined for every domain component and define the belonging of each domain component to a specific synchronization group.

Relationships between objects are expressed by KeyValuePair exchanges (resp. as two key sets, see above). Furthermore, object properties (resp. `KVInterface` member variables) can be arbitrary data, such as numbers or strings.

Formally, let $G = (\text{SynchronizationGroup}, \{G\}, \{O\})$ be a graph definition with child graphs {G} and objects {O}, where $O = (\texttt{KVInterface}, Ro, \{Re\}, \{P\})$ is an object with $P = \{(\text{Data type, name})\}$ variables, role $Ro = $ (Generic set, defined by domain expert) and relationships $Re = \{KVPair\}$ (see Figure 4.10).

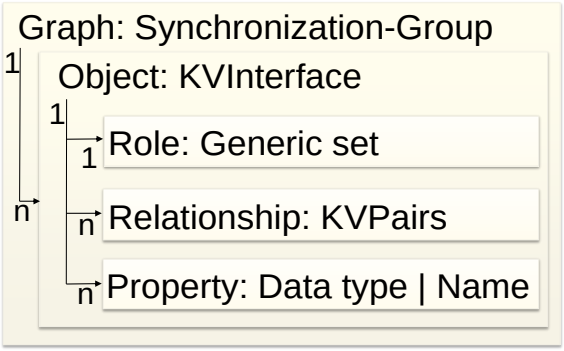


Figure 4.10: Simplified hierarchical depiction of my lightweight DSML approach, based on the GOPRR notation, for the ECS pattern. GOPRR specifies abstract DSML concepts: the graph and the objects with their role, relationships and properties. The graphs are the previously introduced synchronization groups in which the objects (actual implementations of KVInterfaces, i.e., Systems) are located. Each object has a role (Entity type and id) and relationships to other KVInterfaces represented as keys of KeyValuePairs as well as the corresponding properties (represented as Components).

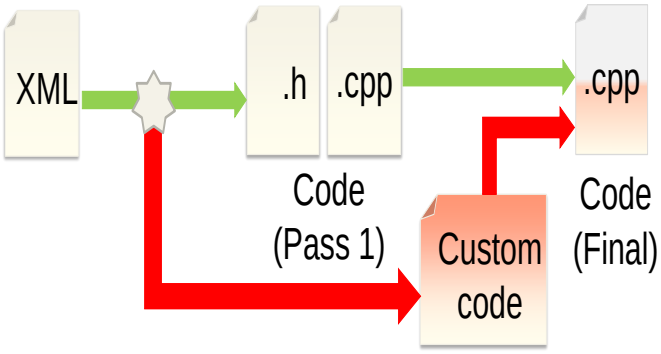


Figure 4.11: My two-pass TBCG approach: templates with all read and write operations are generated (green) and the interface behavior is added manually (red) (see Algorithm 14).

4.3.3 Code Generation

I use TBCG [33] in order to generate source code for my PSMs from my PIM. TBCG is a generative technology that transforms a given model into source code, through the use of templates in two passes (see Figure 4.11). These templates provide a high level of flexibility for the generated output required by custom generation scenarios. Furthermore, it is extensively used throughout the industry [33], leading to good documentation and continuous development. A template thereby consists of imperative control and structural source code patterns such as loops or conditional statements. As everything in my virtual testbed is implemented as homogeneous interface to my KeyValuePool and GraphPool based software infrastructure, my TBCG directly aims at generating all read and write interfaces of the corresponding `KVInterfaces`. Currently, PSM can be generated for the programming language C/C++ (see Pseudocodes 4.3 and 4.4).

In order to implement above concept, the following information is derived from the DSML:

- ω : all specified `KVInterfaces`
- α : name of one `KVInterface` $\in \omega$
- β : role (represented as base class) of one `KVInterface` $\in \omega$
- γ : list of properties per `KVInterface`, structured as tuples with data type and name
- δ : list of KeyValuePairs which are read by the `KVInterface`, represented as a key list
- ϵ : list of KeyValuePairs which are written by the `KVInterface`, represented as a key list
- ζ : `KVInterface` unique synchronization group id and name

Algorithm 14 GenerateKVInterfaceImplementation($\alpha, \beta, \gamma, \delta, \epsilon, \zeta$ from $\forall \omega_i \in \omega$)

1: Generate header file with derived class α from β
2: **for** $\gamma_i \in \gamma$ **do**
3: Generate private variable $\gamma_i - name$ with $\gamma_i - type$ in header file declaration
4: **end for**
5: Generate cpp file with include to class α
6: Generate empty read, write and work function of α
7: **for** $\delta_i \in \delta$ **do**
8: Generate **key-value pool read** of key δ_i for variable $\gamma_i - name$ in read function
9: **end for**
10: **for** $\epsilon_i \in \epsilon$ **do**
11: Generate **key-value pool write** of key ϵ_i for variable $\gamma_i - name$ in write function
12: **end for**
13: Add KVInterface to synchronization group ζ in simulation loop

```
class System−α : public β
{
public:
 // Generic access functions for data
 void read();
 void write();
 void work();

private:
 //Generate for each γi ∈ γ:
 γi − type γi − name;
 //Synchronization group
 unsigned int ζ
 //KVPool or GraphPool reference
 Pool p
};
```

Listing 4.3: KVInterface header class generation

```
#include "βα.h"

void α::read()
{
 //Generate for each δi ∈ δ:
 γi − name = p−>get(δi);
}

void α::write()
{
 //Generate for each ϵi ∈ ϵ:
 p−>put(ϵi);
}

void α::work()
{
 //TODO: Implement behavior
}
```

Listing 4.4: KVInterface cpp generation

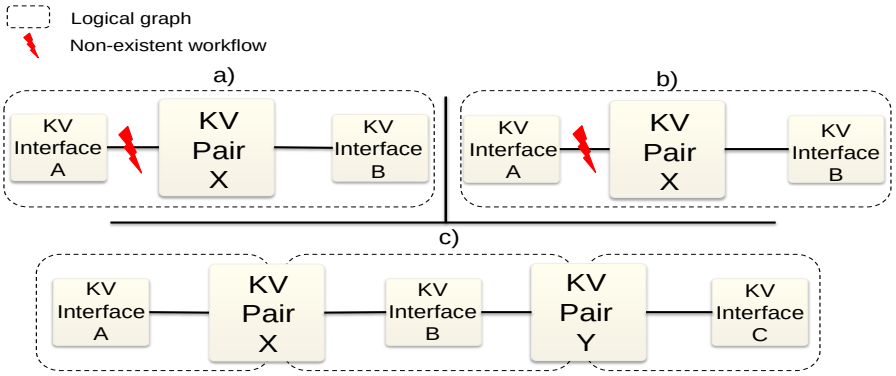


Figure 4.12: Model validation of the simulation dataflow besides the evident checks a) and b), also interface communication is checked: c) if data from domain component *A* should be perceived by *C*, which is not situated in the same logical graph (synchronization group) as *A*, it has to be transmitted by an interface domain component *B* which is placed between the two logical graphs.

4.3.4 Model Validation

Model validation is, besides code generation, one of the main aspects and benefits of a DSML based approach [178]. It aims at checking whether a model conforms to its specified requirements. As mentioned earlier, the huge complexity of virtual testbeds with the ever increasing amount of software interfaces between simulation, optimization, user interaction and visualization makes interface development tedious and often stressful (see Chapter 1). I therefore validate the simulation dataflow as it exactly constitutes the internal interfaces of the simulation. This validation check is modelled as finite state machines which are generated by the overall KeyValuePair access of all `KVInterfaces` (see Figure 4.12). I validate three simulation dataflow constraints. If a KeyValuePair is defined and written by one `KVInterface`, at least one other `KVInterface` must read it, otherwise the modelling and generation of this KeyValuePair is unnecessary. Analogous, if a KeyValuePair is read by at least one `KVInterface`, another `KVInterface` must create this KeyValuePair beforehand. In addition to these evident requirements, I also validate whether the simulation and optimization is itself not violated: `KVInterfaces` can only communicate with other `KVInterfaces` within their logical graph. For instance, my SBO approach (see Chapter 5) must only change its designated parameters, a simulated vehicle must only perceive its environment by simulated sensor measurements and the simulated environment must only retrieve actuator information from the vehicle.

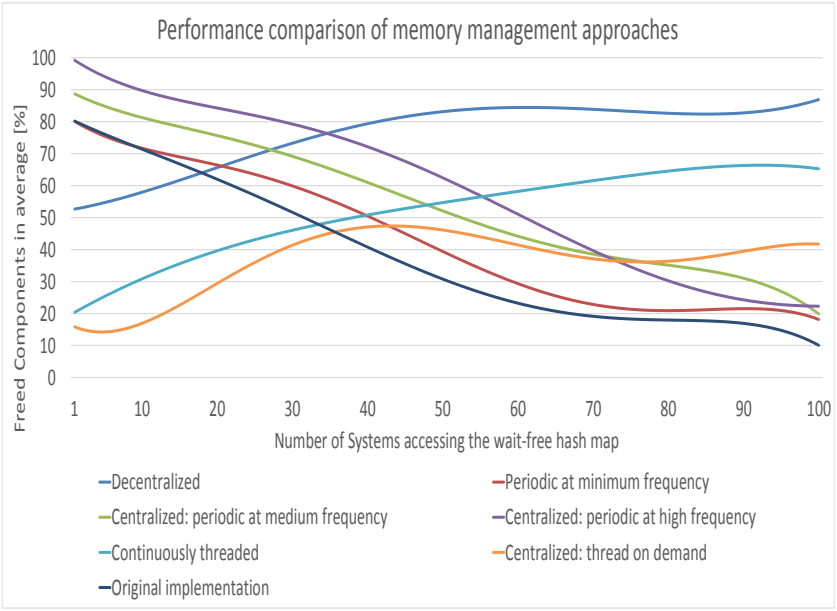


Figure 4.13: Performance comparison of the proposed memory management approaches. None of my approaches delivers 100% performance for arbitrary numbers of *Systems*. However, diverse use cases emerge in which certain approaches perform best. This leads to best practices as described in the evaluation.

4.4 Results

I conducted different experiments to measure the management efficiency my ECS based KeyValuePool and GraphPool approach. The basis of these experiments is the end-to-end spaceflight mission simulator, as described in Section 3.6. I implemented my ECS based software infrastructure with KeyValuePool and GraphPool as well as DSML in C++. I performed experiments on a machine with an Intel Core i7 4-core processor (3.4 GHz) with enabled Hyperthreading, using the Microsoft Visual C++ 14 compiler with all optimizations, operated by Windows 7 64 bit and 8GB of RAM

I compared my different memory management strategies in several set-ups of my use case. My evaluation concerned different amounts of active `Systems` within the use case study as well as varying `Component` types. The `Components` represented simple three-dimensional positions, 3x4 and 4x4 matrices, point clouds (ranging between 1.000 and 40.000 points), three-dimensional line segments and geometry as well as standard programming language objects such as strings or integers. Furthermore, the `Components` varied in size between a few Byte and several Megabyte. I performed 10.000 read- and write operations for each test. In order to avoid caching effects I repeated all tests 50 times and I averaged the resulting timings.

Figure 4.13 illustrates the performance of my novel memory management. Here, I evaluated how many unused `Components` are deleted each simulation step in the virtual testbed scenario. Clearly, my novel memory management outperforms the original implementation. Actually, all of my proposed strategies outperform the original implementation but they also perform diverse for varying numbers of `Systems`. Each implementation, whether centralized or decentralized, exhibits a sweet spot in which it outperforms the competitors. In detail, the decentralized approach outperforms the centralized implementations the more `Systems` access the `Components`. In case of less active `Systems`, the centralized approaches, especially the frequency dependent variations, perform better. I believe that an increasing number of `Systems` increases the overall memory dependency between the `Systems`.

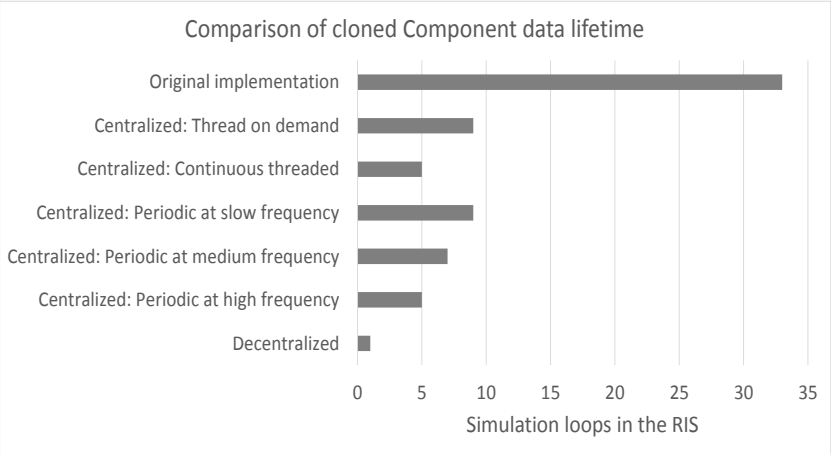


Figure 4.14: Comparison of the unused *Component* data lifetime before they are deleted in a RIS setting for virtual testbeds with 100 *Systems*. The original implementation is based upon the initially described memory management of the wait-free hash maps without the ECS integration. This is a configuration excerpt from the previous evaluation to show how strongly the individual approaches differ in their performance.

This means that the more active `Systems` a RIS inherits, the more `Systems` are likely to use the same `Components`. Therefore, they are "blocking" the release of the cloned `Component` data.

Consequently, the responsibility of releasing the memory shifts from the overall collective of `Systems` more to the single `System`, resp. to the decentralized approach. At last, the centralized approaches always outperform the original KeyValuePool and GraphPool based implementation which uses a standard global list for the management of unused `Components`.

Clearly, my novel memory management approach outperforms the original implementation also for minimizing the lifetime of cloned `Components` (see Figure 4.14). In detail, the decentralized approach outperforms all competitors while the centralized approach with high frequency can nearly compete with it. It can be further observed that the frequency of the periodic centralized implementation directly relates to the lifetime of the unused `Component` data.

I also compared the actual access performance of the presented enhanced memory management with the original implementation and a lock-based implementation. It can be observed, that my enhanced memory management for the wait-free hash maps does not introduce any performance bottleneck to the original implementation and it behaves almost identical for wait-free read and write operations. Furthermore, my results show, in accordance to my other evaluations (see Section 3.7), that the wait-free access implementation gradually outperforms the traditional lock-based implementation with an increasing amount of `Systems` by several orders of magnitude.

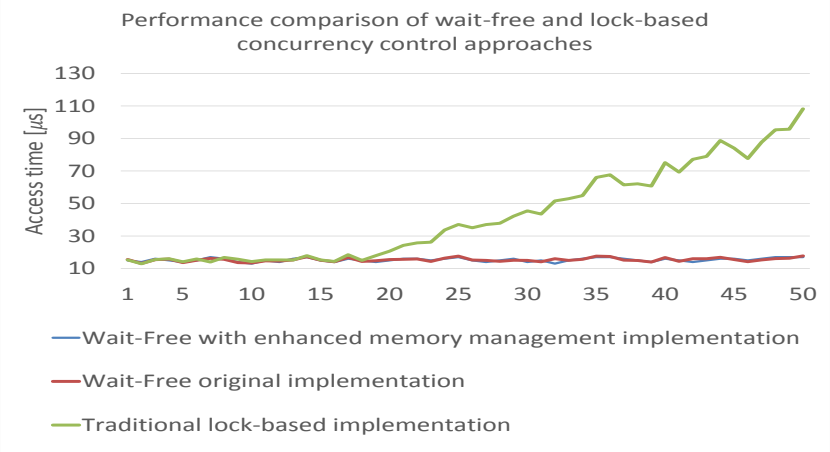


Figure 4.15: Performance comparison of my memory management enhanced wait-free implementation, original wait-free implementation and standard locking approach. The integration of ECS does not influence the wait-free performance and does perform equally. Both wait-free approaches clearly outperform a traditional lock-based implementation.

4.4.1 Best Practices

It can be summarized, that my evaluation revealed different advantages of the presented memory management approaches. In detail, for only a few active `Systems` in the RIS, the centralized (periodic with any frequency) approaches outperform their competitors. However, if the RIS inherits more than 30 active `Systems`, the decentralized approach outperforms all competitors. Furthermore, the memory is deleted fastest in the decentralized approach, followed by the centralized (periodic with high frequency and continuously threaded implementation) approaches with an increasing number of active `Systems`. I can derive from this some best practices for RIS development which use my wait-free hash maps with double buffering. Tables 4.1 and 4.2 illustrate my findings for different use cases, namely: RIS applications which inherit few/many `Systems` as well as RIS applications which inherit small (e.g. only primitives like vectors, matrices) or large `Component` data (such as large structured data such as point clouds or triangles).

Few *Systems*	
Small *Component* Data	Large *Component* Data
Centralized (periodic with any frequency) management	Centralized (periodic with high frequency) management

Table 4.1: Guideline for memory management approaches for few *Systems* within a RIS.

Many *Systems*	
Small *Component* Data	Large *Component* Data
Decentralized management	Decentralized management

Table 4.2: Guideline for memory management approaches for many *Systems* within a RIS.

Part III

Algorithms and Concepts for Blackbox Optimization in Virtual Testbed Simulations

Wise men speak because they have something to say; Fools because they have to say something.

Plato

5

Data Mining Algorithms for Pareto based Multiobjective Optimization

In this chapter, I introduce my novel data mining algorithms and concepts for my novel automatic KDP for MOO in virtual testbeds. This approach is able to uncover unknown causal relations in simulation models and to approximate unknown objective functions. This information is gathered with a novel data mining concept and enables a MOO of the high-dimensional input space, for parametric optimization (see Section 2.2) of the simulation model. Such approximations and optimizations are currently not available for virtual testbeds. I recall some of the most relevant research articles that have appeared in the international literature related to this topic and emphasize my contributions.

Traditional SBO approaches [107, 114] usually require pre-defined objective functions which directly describe the influence of all simulation input parameters on the specified simulation objectives (denoted as model behavior). An optimization toolset (e.g. [43]) uses these objective functions (e.g. ordinary differential equations) in order to find a local or global minimum which satisfies given constraints. As a consequence of the increasing complexity of state-of-the-art simulations, such objective functions are not always available.

Even more, there are many technical complex systems whose long-term behavior can not be described by a set of equations (e.g. long-term behavior of autonomous systems in changing environments). This kind of SBO problem is called blackbox simulation problem because the objective functions are unknown to both: the simulation engineer and optimization toolset (see Chapters 1 and 2). In these blackbox simulations, and in regular simulations, is the analysis of the model behavior and the determination of the valid design space, respectively, usually done manually by simulation experts, guided by mathematical methods which introduce heuristics on the unknown objective functions.

This manual analysis is generally performed by identifying a few distinct parameters according to the simulation project scope given as a set of simulation objectives. An optimization is not conducted by a simulation itself, but rather through execution of multiple simulation runs.

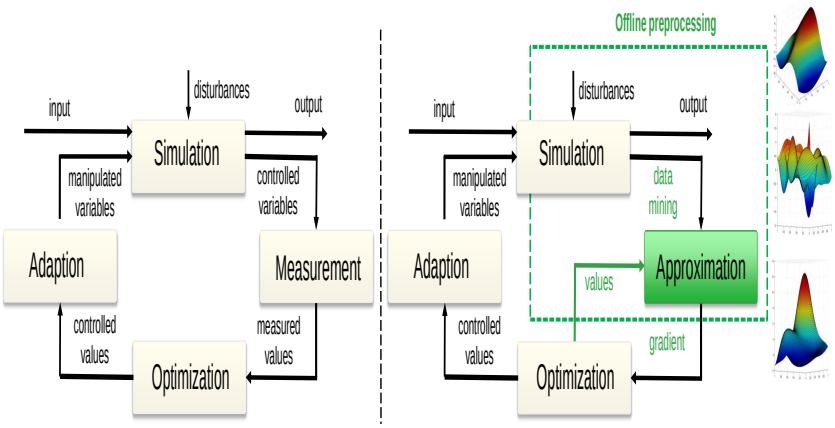


Figure 5.1: My KDP approach introduces a offline pre-processing step (right) to the traditional SBO loop consisting of simulation, optimization, objective measurement and configuration adaptation (left). This pre-processing unveils hidden relationships in the simulation model, approximates the unknown objective functions based on B-Spline surfaces and delivers finally a FDS approximation for a optimization toolset.

In order to reduce complexity and number of runs, the input parameters of each single run have to be varied cleverly [141]. The simulation expert usually takes an educated guess based on his experience which parameters might be influential on the project scope and therefore time and effort is invested in experimenting with these focus parameters in a fixed system configuration environment. Hence, for each simulation run, the input configuration has to be adjusted if the simulated model performance does not meet scenario or engineering expectations. Currently, this adjustment of input parameters in simulations is either done externally by simulation experts that need to guide the simulation process, or by defining a number of scenarios. These scenarios are pre-defined by simulation experts to cover almost all aspects of the simulation model. The use of expert guidance can lead to quite effective simulation results. However, such experts are rare and expensive. Additionally, its not always feasible to have an expert available for configuring and supervising the simulation. Nevertheless, this approach is widely used [83] but yields many disadvantages as this workflow is based upon subjective judgment of simulation results. These judgments are insufficient for efficiently solving this problem because they can not survey the whole underlying MOP of the simulation model, especially for blackbox simulations.

[83] refers to this as the "trial-and-error approach" to finding a good solution and recommends that simulation experts should spend more time in analyzing than building the model. Furthermore, pre-defined scenarios may lead to less optimal adaptation [92].

Consequently, it would be beneficial to automatically compute suitable input configurations for a given simulation model without the need of an expert guiding this process [152]. In addition, recent simulation models are dominated by a MOP because many real world problems involve decisions based on multiple and conflicting criteria [79].

There is already a number of computational methods for solving MOPs, for instance [59, 174], available. Some of these computational methods also work for some use cases in blackbox simulations. However, they usually do not consider the generation of vast amounts of simulation model behavior results that can be easily derived from simulation data farming via a KDP. Although, this data could be used to deliver additional (gradient) information to traditional MOP solving approaches. In the best case, optimization approaches can directly benefit from this information.

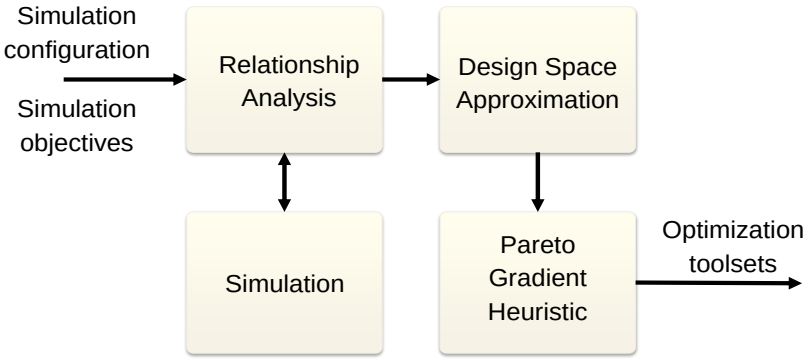


Figure 5.2: Simplified steps of my proposed KDP: a relationship analysis is conducted based on simulation data. After discovering unknown relations between simulation input and simulation model behavior, the design space of the simulation model input is approximated and the information are given to an optimization toolset.

Unlike traditional approaches for solving MOP in blackbox simulations, these KDPs in simulations are not limited to a static, pre-determined input dataset for model behavior and optimization. Instead, the simulation is used as a generator for new data by a simulation sampling process.

By definition, KDPs incorporate some kind of data mining process which samples the simulation input parameter space [113, 141]. This enables a KDP to investigate the simulated model behavior in more detail and larger bandwidth [153] via its data mining scheme. This data mining scheme directly determines the efficiency and quality of the resulting objective function approximation because it defines the simulation sampling process.

I present a different approach, a generic KDP for simulations (see Figure 5.2), which is directly based on this observation and the idea of KDD [200]. Unlike traditional approaches for solving MOP, KDD resp. KDPs in simulations are not limited to a static, pre-determined input dataset for model behavior and optimization. Instead, the simulation can be used as an generator for new data by itself.

This enables the investigation of the whole bandwidth or at least the largest part of possible model behavior by conducting cleverly designed simulation data farming in order to discover surprises and potential [65, 180] which can be re-used in the MOP solving process, too. More precisely, I adopted techniques from KDD research to a generic KDP for multiobjective Pareto-based optimization in deterministic and stochastic simulations with unknown objective functions. My KDP approximates these objective functions via a novel data mining concept.

State-of-the-art KDP approaches do not support stochastic simulation behavior and are therefore restricted to deterministic simulations. Nevertheless, sophisticated simulations involve real-world scenarios which incorporate stochastic processes or properties.

Approximating objective functions in stochastic simulations is more difficult and complex because the underlying noise of the stochastic process involves variations in the simulation and consequently in the data farming process. State-of-the-art studies model stochastic processes as deterministic ones which leads not only to inferior approximation of the objective functions but also to inferior solutions of the MOO toolset as I will show in my evaluation.

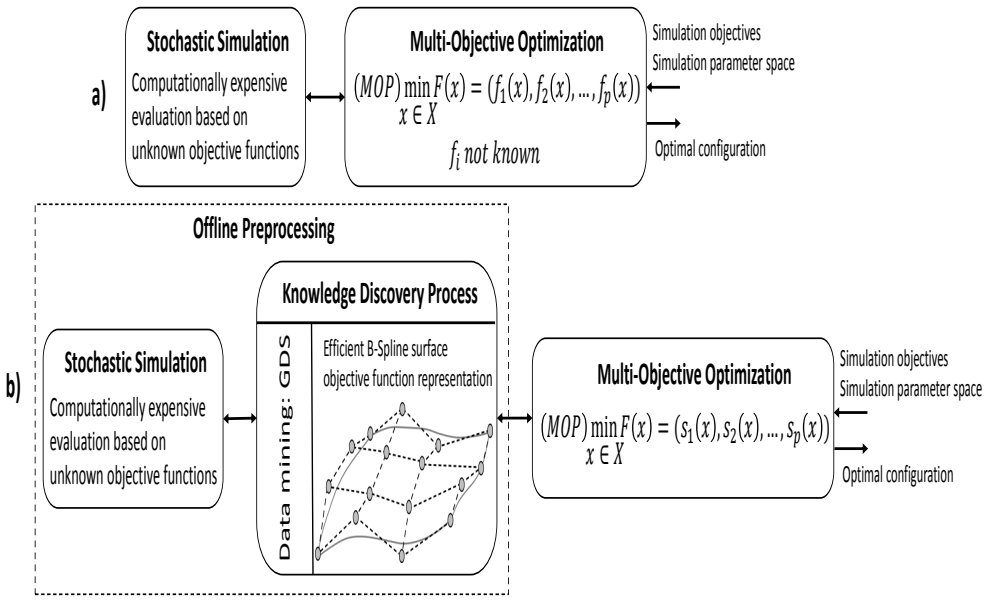


Figure 5.3: My data mining approach introduces an computationally efficient objective function approximation based on my gradient based density spline surface (GDS) concept within my offline KDP via novel B-spline response surface representations for deterministic and stochastic simulations. These response surfaces are used to replace the unknown objective functions for MOO of the simulation model.

My approach automatically builds an active model between simulation input and simulation objectives which is capable of

- uncovering unknown causal relations in large parameter sets between simulation input and model behavior which are assumed to be unknown non-linear objective functions,
- approximating objective functions (resp. the FDS) in arbitrary deterministic and stochastic blackbox simulations as B-spline surfaces,
- computing a Pareto gradient from the FDS approximation for concave, convex or interrupted Pareto fronts, which can be used with different optimization strategies,
- computing a Pareto solution from the FDS approximation with my proposed hierarchical MAS approach (see Chapter 6).

My approach is completely automatic. It does not need, in contrast to existing approaches, any supervision from simulation experts. Another advantage of my approach is its performance. It gains its efficiency from a novel B-spline based sampling of the parameter space in combination with a novel forest-based simulation dataflow analysis. Another main advantage of my approach is that my B-spline surface based FDS approximation evaluation is computationally very fast and replaces costly simulation evaluations which are usually required. Consequently, my approach also delivers a performance boost when computing a solution for the given MOP. Furthermore, my approach is very generic. It can be easily incorporated into existing SBO approaches. Even more, the computed Pareto solutions are close to the Pareto front for deterministic and stochastic simulations. Additionally, of my approach provides three different optimization strategies. These strategies can be used by state-of-the-art MOO solvers in order to investigate a larger bandwidth of the simulated model behavior.

Within my KDP, I present my GDS data mining algorithm. GDS is able to approximate arbitrary unknown objective functions in deterministic and stochastic blackbox simulations, for convex and concave as well as interrupted Pareto fronts.

GDS runs as a pre-processing step before the actual SBO process and replaces the traditional variable based simulation result measurement step (see Figure 5.1). GDS consists of three main concepts:

- B-spline surface representation of the relationship space (see Section 5.4.1),
- density based clustering of objective function samples which determines the noise behavior of the stochastic simulation (see Section 5.4.2),
- gradient based sampling of the parameter space which reduces the required amount of samples (see Section 5.4.3).

The result of GDS is an efficient approximation of the objective functions as a set of B-spline surfaces which can be used in various SBO scenarios, such as complex MOPs. This efficient approximation is computationally very inexpensive compared to usually very costly simulation evaluations (see Figure 5.3). It can be further effectively utilized in optimization toolsets, e.g. by my hierarchical MAS based optimization approach (see Chapter 6).

5.1 Related Work

Research in combining KDD and simulation methodology has attracted increasing interest in the last decade. There are numerous applications, with the main aim of assisting the simulation analyst and engineer, e.g. by visualizing simulation data with annotations, analyzing existing Pareto solutions or by clustering simulation output.

However, the main interest is to approximate objective functions in order to interpolate or extrapolate simulation model behavior. Mostly, such approximations are implemented as a sophisticated data mining approach within a complex KDP.

The research can be classified into two groups: KDP approaches aiming at SOO or MOO. All presented KDP approaches have in common, that they only support deterministic simulations. Current KDP approaches for SOO problems reduce the optimization problem, for instance [113] explored the landscape characterization problem with a support vector machine by analyzing the complete input parameter space.

The approach assumed a non-objective oriented simulation, in which the simulation model can be reduced to a single function f which updates the simulation state x with parameter set θ via $x_{k+1} = f(x_k, \theta)$. They defined the landscape characterization problem by determining the set of points θ in which a predefined simulation state is achieved. This approach can neither be applied to SOO nor MOO based simulations in which the simulation model is governed by a set of (possible) contradicting functions $f_i, \ldots, f_n$ as the approach does not concern such structures within the simulation. [12] determined dynamic adaptation strategies for agent-based traffic simulations via supervised learning. They extracted parameter patterns in the from of decision trees in stochastic simulation by simulating the simulation model several times. These generated decision trees are valid for linear relationships between input parameters and model "what if" studies. The approach is further restricted to a small number of simulation input parameters because it involves a runtime which is quadratic in the number of input parameters.

Likewise, [132] neglected MOO properties. They investigated the application of KDD in simulation of aircraft engine fleet management. They applied a linear regression to all input parameters $x_i, \ldots, x_n$ for one simulation objective state y resulting in a model of the form $y = C + \alpha_1 x_1, \ldots, \alpha_n x_n$. This model was used to determine the cost drivers in aircraft fleet management. These cost drivers were then classified by a clustering algorithm into low or high cost classes, describing the main cost drivers for the given fleet management simulation.

[141] proposed an approach for uncovering unknown relationships in model behavior. They conducted large scale experiments by replicating pre-defined experiment definitions. The resulting simulation data output was clustered and presented in various plots and charts in order to reveal unknown relationships for the simulation experts and is consequently highly depending on the simulation expert. KDP approaches based on MOO (respectively Pareto) based optimizations for simulations (such as [30, 104, 130, 173]) focussed on extracting additional information from pre-determined Pareto sets or analzying these sets within the simulation. Consequently, they can be used to neither approximate the FDS nor to compute a Pareto solution itself. Other approaches concerning SBO in virtual testbeds such as [105, 106] work within the field of deterministic simulations for eRobotics use cases [90]. Other KDP approaches based on MOO in simulations [30, 104, 130, 173] focussed on extracting additional information from pre-determined concave Pareto sets or by analyzing these sets within the simulation. Consequently, they can not be used to approximate the FDS nor to compute a Pareto solution itself.

In summary, all above mentioned studies focused on building passive models between simulation input and objective-related simulation output while minimizing the simulation parameter scope or by focusing on single-objective linear simulation models in deterministic simulations. These passive models describe the simulation without enabling interpolation or extrapolation of the simulated model behavior for SBO purposes. Instead they aim is to describe the output of the simulation in a human-understandable format. They deliver coarse granularity parameter relationship information which can not be used to approximate the FDS nor to compute a Pareto gradient information (e.g. gradient information of the analyzed data with respect to the Pareto front), especially for stochastic simulations. Consequently, they can not be used as input for MOO algorithms in order to compute suitable configurations. Concluding, all above studies are restricted to deterministic simulations which leads to several disadvantages as stated before.

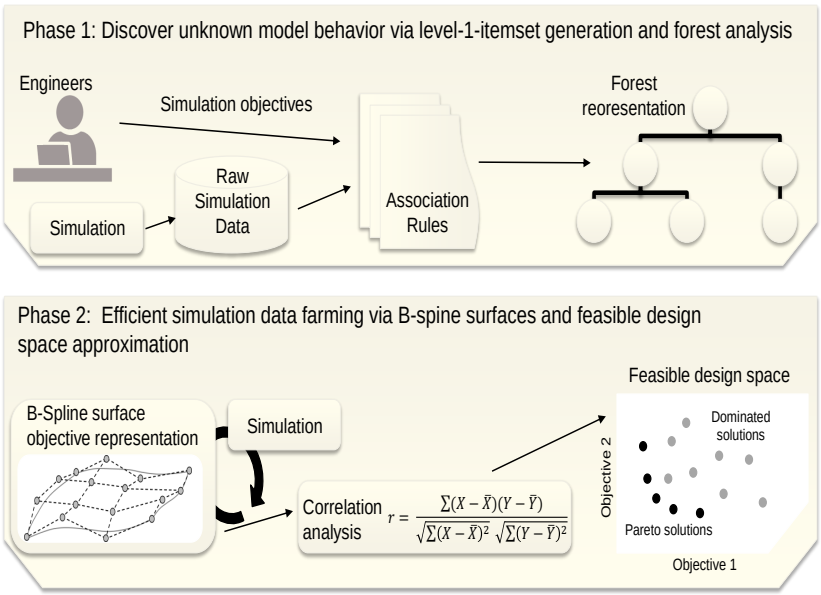


Figure 5.4: My automatic KDP: first, causal relations between simulation input parameters and simulation objectives are revealed via my association rule mining approach with forest based dataflow analysis. The used simulation objectives are defined by the engineers. Second, simulation data farming is efficiently conducted (by structuring the dataflow with my forest approach) in order to approximate the unknown objective functions and the FDS. This approximation is checked for correlation and used to compute Pareto gradient information and multiobjective solutions.

5.2 Process Overview

Originally, KDD is defined as making sense of data collections that are too big to manually review each and every single record. Input sources for such kinds of data are complex simulations, graphs, or data warehouses [203]. [200] describe the KDD process as multiple steps to ultimately transform low level data into useful knowledge. In detail, the KDD process is a highly interactive five-step-process that requires many decisions made by the user. Some of these steps (e.g. target data selection or interpretation of patterns) have to be iteratively repeated by the user for convincing results. Hence, KDD is a semi-automatic process because the user is ultimately responsible for interpretation and evaluation of mining results. This particularly applies for the evaluation of the usefulness of the generated knowledge [200] (see Section 2.3).

Today, simulation models are dominated by a MOP because many real world problems involve decisions based on multiple and conflicting criteria [79]. I already outlined the concept and challenges MOP (see Section 2.2.2). In summary, the optimal decisions have to consider the best trade-off among these criteria. These best trade-offs are called Pareto solutions. Computing the Pareto front (set of Pareto solutions) or a single Pareto solution is actually the goal of MOO. Such MOPs can be found in many situations, for example, in product design where several criteria must be simultaneously satisfied [25, 170, 187].

My motivation is to adapt the original KDD process as a novel generic KDP for SBO applications in virtual testbeds. Therefore, I present the application of an completely automatic KDP to reveal causal relationships between simulation input parameters and pre-defined simulation objectives with respect to blackbox simulations with an underlying multiobjective model behavior. The result of my KDP is an approximation of the FDS as well as Pareto information which can be directly used for solving the MOP of the simulation model.

Three main challenges arise when applying KDD techniques to these problems. First, engineers who specify the simulation model as well as simulation experts have limited and hence incomplete knowledge about the simulation model behavior with respect to the complete parameter input space. Consequently, the assumed relations between pre-defined simulation objectives and parameter space input are incomplete or wrong [83]. Unfortunately, efficiently computing viable solutions for MOP requires at least a correct approximation of the relationship between the parameter input space and the objective functions.

This means, all relations between a simulation input parameter and a pre-defined simulation objective within the simulated model behavior have to be determined. Second, KDD usually requires extensive simulation data farming in order to yield useful results. This simulation data farming can lead to a computationally very expensive KDD process because this complexity usually grows at least quadratically with the amount of input parameters [12]. In addition, objective function are usually costly to evaluate (see Section 2.2). Hence, the simulation data farming constraints (selection and sampling of convenient input parameters) have to be minimized. Third, diverse algorithms exist for computing a solution to MOP, such as gradient descent [72, 77], simulated annealing [49, 174] or evolutionary algorithms [58, 59]. The proposed KDP should yield Pareto information in such a way that this information can be directly used in such different optimization approaches.

In order to overcome these challenges, my KDP differs in many ways from the above described standard KDD process (see Figure 5.4). Basically, my process is split into two main phases: First, association rule mining (ARM) based dataflow and forest based workflow analysis. Second, simulation data farming with relationship analysis.

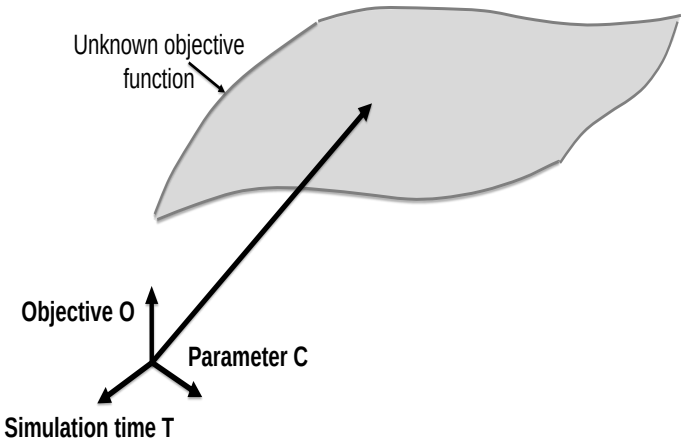


Figure 5.5: Example of a three-dimensional space that indicates how a simulation input parameter C relates to a given simulation objective over time. This unknown model behavior is represented as a three-dimensional surface.

The first phase reveals unknown model behavior and constructs a simulation objective based forest data structure which enables fast simulation data farming. The second phase utilizes this forest in order to analyze the unknown model behavior. This analysis is used to approximate the FDS and consequently to compute the needed Pareto information for MOP optimization.

5.3 Unveiling Hidden Relationships: Forest based Association Rule Mining

My KDP starts with the determination of possible causal relations between simulation input parameters and simulation objectives. The overall idea is to decompose the large high-dimensional input space into sub-parts and to prove correlation for these parts. Correlating parts are then aggregated back into the higher dimensional input space. These aggregated information are then used for optimization (see Figure 5.7).

I assume that every relationship between a parameter $C = c_0, ..., c_m$ with size k and objective value $O = o_0, ..., o_m$ with size m can be formally represented as a continuous function $f : C, T \mapsto O = f(c, t) \mapsto o_{\mathcal{O}}$ which maps the parameter space to a given simulation objective $\mathcal{O}$ with its objective function space at a given time step $t \in T$ of the simulation. It would be possible to perfectly determine the behavior of f with respect to C by brute-force sampling the whole parameter with $s \geq k$ samples. However, in real world applications k can be arbitrary large or continuous and the simulation evaluation computationally very expensive. Therefore, a brute-force sampling of the parameter space is infeasible. Consequently, it is necessary to reduce the amount of needed samples: $s \ll k$. Overall, the relationship constitutes a large three-dimensional cartesian space $\mathbb{R}^3$, spanned by C, O and T (see Figure 5.5).

I define a possible causal relation with an existing dataflow inside the simulation denoted as $d_j\{x_i, ..., x_n\} \mapsto \mathcal{O}_j$ where $\mathcal{O}_j$ is a pre-defined simulation objective which maps the parameters $\{x_i, ..., x_n\}$ with a dataflow d_j to a satisfaction value or objective state.

My approach concerns blackbox simulations, thus, no mapping between parameters $\{x_0, ..., x_n\}$ and simulation objectives $\{\mathcal{O}_o..., \mathcal{O}_j\}$ as well as explicit forms of $\{f_o, ..., f_j\}$ is known in advance. The simplest, and computationally most expensive, approach would be to brute-force analyze all given parameters for every simulation objective in order to reveal unknown model behavior.

This would result in a simulation data farming computational complexity of:

$$\mathcal{O}((n^2 - n) \cdot s \cdot g) \tag{5.1}$$

where

g : number of simulation objectives
s : sampling size of the parameter
n : number of simulation model input parameters

Sophisticated simulations easily inherit hundreds or thousands of input parameters with large parameter spaces. This would result in computationally very expensive brute-force analysis of the complete KDP.

In order to overcome this limitation, I present several ideas to accelerate the computation. I start with a fast ARM which uncovers the complete dataflow $\{d_o, ..., d_j\}$ of the simulation by analyzing all dataflow transactions. Within the simulation, a dataflow transaction d_i has to exist for making f_i possible. If there is no dataflow, no function could be defined which models this relationship. Consequently, these transactions can be used to determine the parameter mapping $\{x_i, ..., x_n\}$ as well as for identifying f_j of $f_j\{x_i, ..., x_n\} \mapsto \mathcal{O}_j$.

The main idea is to use the traditional ARM Apriori [163] algorithm with low support and high confidence settings. In order to enable data farming parallelization and pruning of the analyzed workflow, I transform the original list output of the Apriori algorithm into a disjoint union of tree data structures called forest.

Following the original definition by [163], the problem of ARM is defined as: Let $I = i_i, i_2, ..., i_n$ be a set of n binary attributes called *items*. Let $D = t_1, t_2, ..., t_n$ be a set of transactions called the database. Each transaction in D has a unique transaction ID and contains a subset of the items in I. An association rule is an implication expression of the form $X \Rightarrow Y$, where X and Y are disjoint itemsets, i.e., $X \cap Y = 0$. Further, $X, Y \subseteq I$.

To illustrate these concepts, I use a small example from the supermarket domain: $\{butter, bread\} \Rightarrow \{milk\}$ meaning that if butter and bread are bought, customers also buy milk.

The strength of an association rule can be measured in terms of its support and confidence. The support value of X with respect to T is defined as the proportion of transactions in the database which contains the item-set X given as $\sigma(X) =\mid \{t_i \mid X \subseteq t_i, t_i \in T\} \mid$. Confidence, on the other hand, measures the reliability of the inference made by a rule. Both are mathematically defined as:

$$Support, s(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \tag{5.2}$$

$$Confidence, c(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{5.3}$$

Apriori [163] is an algorithm for frequent item set mining and ARM over transactional databases or sets. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The output of the Apriori algorithm is a list of level-k-itemsets: $\{\{X_0, ..., X_k\} \Rightarrow Y\}_{s,c}$.

Algorithm 15 GenerateForestStructure

1: $\mathcal{O}$ = list of objective references $\{X_0, ..., X_g\}$
2: $\mathcal{L}$ = list of level-1-itemsets rules: $\{X\} \Rightarrow Y\}$
3: $\mathcal{F}$ = forest root node
4: **for** $\mathcal{O}_i \in \mathcal{O}$ **do**
5: $\quad \mathcal{M}$ = tree root node with $\mathcal{O}_i$
6: $\quad \mathcal{M}_{childs}$ = GenerateTree($\mathcal{M}$, $\mathcal{L}$, $\mathcal{O}_i$)
7: $\quad \mathcal{F}_{trees}$ += $\mathcal{M}$
8: **end for**

Algorithm 16 GenerateTree

1: $\mathcal{R}$: read relations of $\mathcal{O}_i$
2: $\mathcal{R}$: $\bigcup_{X \Rightarrow Y \in \mathcal{L}} := \{X \mid \forall Y = \mathcal{O}_i\}$
3: **for** $\mathcal{R}_i \in \mathcal{R}$ **do**
4: $\quad \mathcal{C}$ = child node of $\mathcal{M}$ with $\mathcal{R}_i$
5: $\quad \mathcal{C}_{childs}$ = GenerateTreeStructure($\mathcal{C}$, $\mathcal{L}$, $\mathcal{R}_i$)
6: **end for**
7: **return** $\mathcal{M}$

In this scenario, Iam only interested in direct relations represented as consistent association rules. More precisely, in level-1-itemset rules which have high confidence c and low support s as they describe direct parameter relations [163].

Due to the inherent structure of a simulation dataflow, which is constituted by the simulation workflow, repeating patterns of data access emerge. For instance, a physically-based simulation of Newton's law will always modify the position and velocity of certain simulated objects. This physically-based simulation will update the corresponding objects every time step in the simulation, generating such repeating patterns. These patterns especially appear when different simulation objectives are related to the same parameters, namely in MOPs. Consequently, the list-based output of the Apriori algorithm is not suitable for efficiently analyzing the simulation dataflow as it can not represent these repeating patterns for quick algorithmic access. These patterns lead to additional effort in the simulation data farming process because they would be analyzed multiple times.

I present a novel idea based on forest data structures in order to overcome these repeating patterns. The main idea is to generate for every simulation objective $\mathcal{O}$ a tree which denotes the level-1-itemset dataflow result of the ARM process for this particular simulation objective. Within these trees, repeating transactions will manifest as duplicated sub-trees which I effectively prune (see Figure 5.6).

My utilization of the simulation transaction data and forest data structure reduces original computational complexity from $\mathcal{O}((n^2 - n) \cdot s \cdot g)$ to $\mathcal{O}(k \cdot s \cdot g)$ (where n is the total number of simulation input parameters and k is the number of related simulation input parameters for g objectives) because it enables the upcoming data mining step to analyze only a subset of the parameter space with $k \leq n$. Each simulation input parameter of this subset has a confirmed dataflow within the simulation for a simulation objective. In the next step of my KDP, each of these dataflows is used to analyze whether or not a causal relation is given.

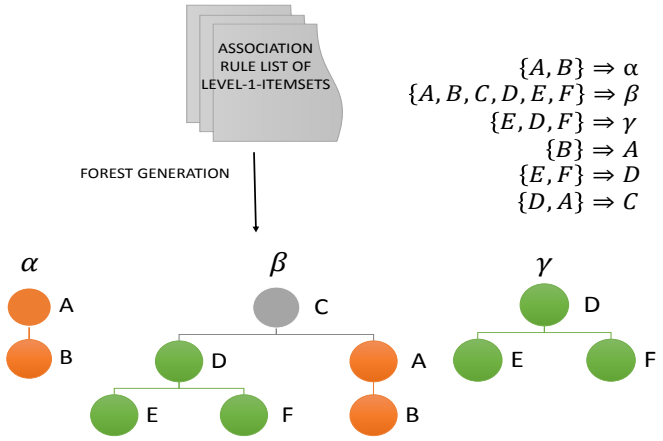


Figure 5.6: Level-1-itemsets are generated by the Apriori algorithm for each simulation objective (example given in the upper right). The list based Apriori output is transformed into a forest structure which facilitates efficient simulation data farming. Repeating data access patterns from the simulation workflow result in prunable (green & orange) sub-trees of my forest.

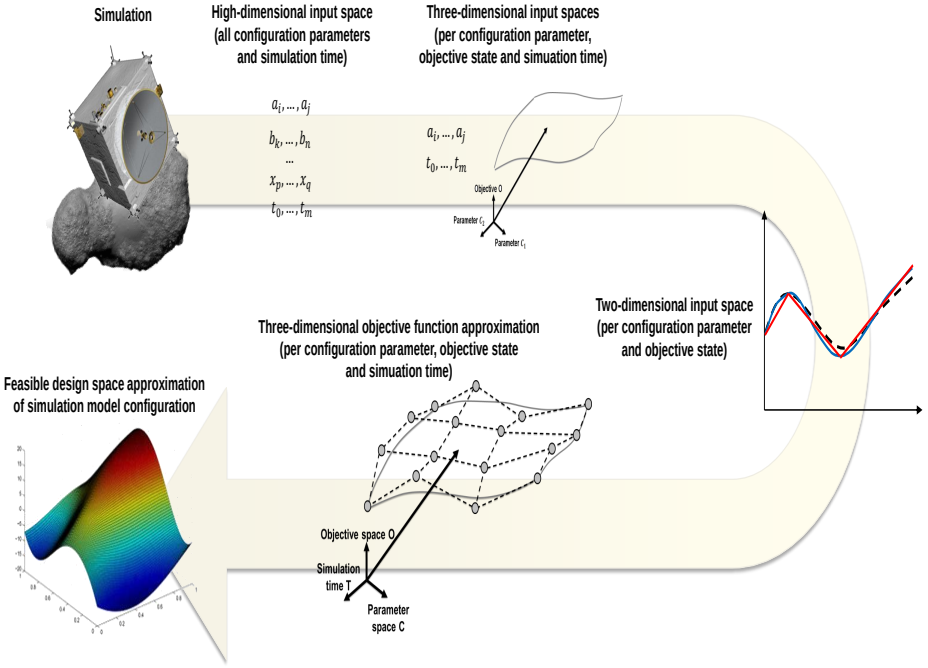


Figure 5.7: The goal of my approach is to accurately approximate the unknown objective functions in order to formulate a FDS. My approach conducts a dimensionality reduction of the high dimensional input space down to three-dimensional and two-dimensional representations of the unknown objective functions for precise approximation. The gained knowledge about the unknown objective functions is then aggregated back to the high dimensional input space via my B-spline surfaces and FDS approximation.

5.4 Approximating Unknown Objective Functions

In order to determine causal relations between simulation input, simulation model and simulation objectives, my KDP needs to farm simulation data after the data- flow determination of the blackbox simulation. As stated before, a brute-force analysis would lead to computationally very expensive behavior of the complete KDP.

In this section, I present the next step of my KDP which involves my efficient farming of simulation data for the given parameter and simulation objective relations (see Figure 5.3). This process is based on the previously introduced forest based data structure and allows a top-down analysis (see Algorithm 17).

In the following, I present the required algorithms and concepts to discover these causal relations as well as to minimize the sampling rate of the parameter space without losing objective value information. First, I introduce a B-spline surface based approximation of the unknown objective function and second, I describe my recursive correlation analysis (RCA) in section 5.4.4.

Algorithm 17 Approximation the unknown objective functions based on the forest data structure forestBasedAnalysis($\mathcal{T}$ association rule tree, ϵ minium correlation coefficient)

```
1:  for each tree 𝒯 ∈ ℱ do
2:    for each node 𝒩 ∈ 𝒯 do
3:      𝒮_objective = basisSplineSurfaceApprox(𝒩, 𝒯_objective)
4:      𝒮_parent = basisSplineSurfaceApprox(𝒩, 𝒩_parent)
5:      𝒞_objective = recursiveCorrelationAnalysis(𝒮_objective)
6:      𝒞_parent = recursiveCorrelationAnalysis(𝒮_parent)
7:      if 𝒞_parent < ϵ or 𝒞_objective < ϵ then
8:        Remove all subgraphs of 𝒩 in ℱ
9:      end if
10:   end for
11: end for
```

5.4.1 Relationship Definition

As stated previously, I assume that every relationship between a parameter $C = c_0, ..., c_k$ with size k and objective value $O = o_0, ..., o_m$ with size m can be formally represented as a continuous function $f : C, T \mapsto O = f(c, t) \mapsto o_{\mathcal{O}}$ which maps the parameter space to a given simulation objective $\mathcal{O}$ with its objective function space at a given time step $t \in T$ of the simulation. It would be possible to perfectly determine the behavior of f with respect to C by brute-force sampling the whole parameter with $s \geq k$ samples. However, in real world applications k can be arbitrary large or continuous and the simulation evaluation computationally very expensive. Therefore, a brute-force sampling of the parameter space is infeasible. Consequently, it is necessary to reduce the amount of needed samples: $s \ll k$. Overall, the relationship constitutes a large three-dimensional cartesian space $\mathbb{R}^3$ (spanned by C, O and T), which I denote as relationship space (see Figure 5.8). In order to overcome the challenge of minimizing the number of required samples while precisely approximating the objective function, I propose an approach based on splines.

A spline is a function that is piecewise defined by low-degree polynomials. Splines are often preferred in interpolation problems over higher-degree polynomial interpolation approaches because spline interpolation avoids the problem of Runge's phenomenon, i.e. oscillations that occur in interpolations between points when using high degree polynomials. Furthermore, even splines based on cubic polynomials can accurately approximate a given non-linear function. In addition, only a few samples are required to precisely define a spline [153].

The general idea of cubic splines is to represent a function by a different cubic function on each interval between data points. For n data points, the spline $S(x)$ is the function

$$S(x) = \begin{cases} F_1(x), x_0 \leq x \leq x_1 \\ F_i(x), x_{i-1} \leq x \leq x_i \\ F_n(x), x_{n-1} \leq x \leq x_n \end{cases} \tag{5.4}$$

where each F_i is a cubic function.

The most general cubic function has the form

$$F_i(x) = a_i + b_i x + c_i x^2 + d_i x^3 \tag{5.5}$$

In sophisticated simulations, the same simulation model (parameter) configuration will contribute to different objective values at different time steps in the simulation (e.g. a fuel state/configuration in a car simulation which changes over time). Because of configurations like this, I prefer to define a spline of the objective function for each simulation time step individually. This results in a list of splines: $S_{t_0}(C), ..., S_{t_n}(C) = O_{t_0}, ..., O_{t_n}$ for n simulation time steps with:

$$S_{t_i}(C) = O_{t_i} \tag{5.6}$$

where

t_i : simulation time
C : complete parameter space based on the dataflow analysis
O : objective values of the corresponding objective function

I use these splines to formulate a cubic B-spline surface:

$$C(u) = \sum_{i=0}^{n} p_i N_{i,3}(u), 0 \leq u \leq 1 \tag{5.7}$$

$$s(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} N_{i,3}(u) N_{j,3}(v), 0 \leq u, v \leq 1 \tag{5.8}$$

where $P_{ij}(i = 0, 1, ..., m; j = 0, 1, ..., n)$ are the control points of the surface which are determined by $S_{t_0}(C), ..., S_{t_n}(C) = O_{t_0}, ..., O_{t_n}$ via a uniform coverage of the splines. u, v are the knot vectors in the direction of u or v and $N_{i,3}(u), N_{j,3}(v)$ is the B-spline basis (see Figure 5.8).

This B-spline approximation (see Figure 5.8) replaces the unknown objective function $f : C, T \mapsto O = f(c,t) \mapsto o_{\mathcal{O}} = s(c,t) \mapsto o_{\mathcal{O}}$ and is efficiently used to define the required gradient information for optimization purposes (see Section 5.5 and Chapter 6).

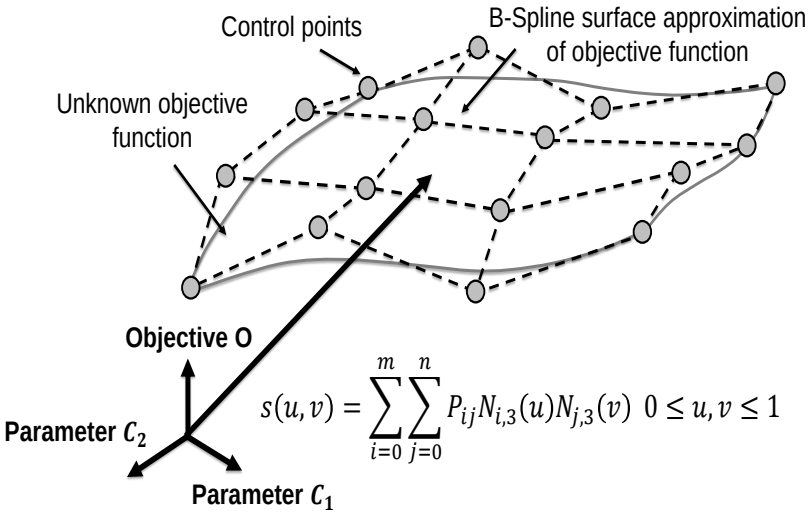


Figure 5.8: B-Spline surface representation of the three-dimensional space constructed by two simulation input parameters C_1, C_2 and objective O. One of the main goals of my approach is to compute the required control points as close as possible to the unknown (in most cases stochastic) objective functions. Usually, a cubic B-Spline surface is used because my evaluations for virtual testbeds have shown that from this surface representation smooth gradients for optimization purposes can be efficiently derived.

5.4.2 Density Splines

In order to enable a precise B-spline surface approximation of the relationship space, the splines which define $s(u, v)$ must be very close to the unknown objective function. However, as stated above, stochastic simulations are governed by diverse noise behavior which makes it hard to approximate these unknown objective functions. Therefore, I define every spline sampling point (c, o) via a density clustering from n simulation samples. This density clustering detects noise outliers and therefore enables my splines to approximate the unknown objective function more precisely.

In order to incorporate this density clustering for the unknown noise distribution of the objective function, I extend the general spline definition (see Equation 5.4). Every spline is defined with a triple, consisting of the cubic function F_i, variance of the computed density cluster θ and certainty of measurement p. In detail, every F_i is constructed with the centre point of the most dense cluster of every simulation sample set per sampling configuration of the simulation. In order to incorporate the simulation noise behavior, every centre point is associated with the variance θ of the corresponding cluster.

My spline definition will interpolate some of the objective values due to the gradient-based sampling of the parameter space (see Section 5.4.3) because I sample only a small subset of the parameter space. This means that, by definition, some approximated objective values are more likely precise (based on simulation samples) than others (based on the spline interpolation). Therefore, p indicates whether or not the resulting approximated objective value is interpolated resp. close (in parameter space) to a drawn simulation sample.

The sampled objective values are clustered with the density based spatial clustering of applications with noise (DBScan) data clustering algorithm [118] (see Figure 5.9). DBScan has several advantages with respect to other clustering approaches (such as [13, 41, 111]) because

- it does not require a specification on the expected cluster amount,
- it can find arbitarily shaped clusters,
- it has a notion of noise which makes it robust to outliers.

This density clustering is important because it enables a more accurate approximation of the unknown objective function via its detection of noise outliers. Therefore, just averaging ψ is insufficient. Even more, the computed DBScan clusters can be used to retrieve the standard deviation and variance of the measurement. I further utilize this information in my optimization process in order to investigate arbitrary MOPs from different optimization perspectives (see Section 5.5).

Therefore, for n sampling points, the spline $S_{t_i}(c)$ is the function:

$$S_{t_i}(c) = \begin{cases} (F_1(c), \Theta(\psi), p(c,\theta)), c_0 \leq c \leq c_1 \\ (F_i(c), \Theta(\psi), p(c,\theta)), c_{i-1} \leq c \leq c_i \\ (F_n(c), \Theta(\psi), p(c,\theta)), c_{n-1} \leq c \leq c_n \end{cases} \tag{5.9}$$

where

$\Theta(\psi)$: variance $\frac{1}{k-1} \cdot \sum_{i=0}^{k} (o_i - \bar{o})^2$

$p(c,\theta)$: certainty of measurement: 1 if $c \in \theta$, 0 otherwise

Ω : DBScan core cluster of ψ

ψ : n samples of c $\{(c,o_i),...,(c,o_n)\}$ with c $\in \theta$

θ : sampling configurations

$F_i(x)$: cubic function based on θ: $a_i + b_i x + c_i x^2 + d_i x^3$

c_j : closest previous sample point $\in \theta$ to c

When using my proposed density clustering, two scenarios can occur which I cover with a heuristic (see Figure 5.10). In the first case, DBScan determines one core cluster and, clearly distinguished, two noise groups. Here, I use the one core cluster for computing $S_{t_i}(c)$. In the other case, DBScan determines three core clusters because the noise distributes very densely around the unknown objective function. In this case, I use the middle cluster for computing $S_{t_i}(c)$.

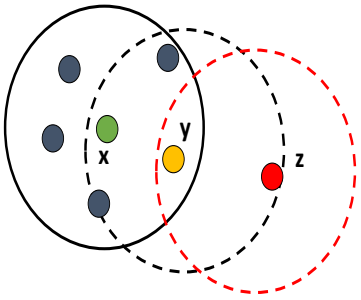


Figure 5.9: Illustration of the DBScan algorithm: a data point is inside a dense region (core point, x), on the edge of the region (boundary point, y) or in a spare region (noise point, z). The example is given for at least six neighbours. Adopted from [118].

5.4.3 Gradient Sampling

The main idea of my gradient-based simulation data sampling is to minimize the amount of samples s (see Section 5.4.1) which are required approximate the original behavior of f. In order to do so, I iteratively approximate the unknown objective function f with my spline definition for a certain simulation time t. This spline is iteratively updated with more sampled data until the spline approximates f within a specified error degree.

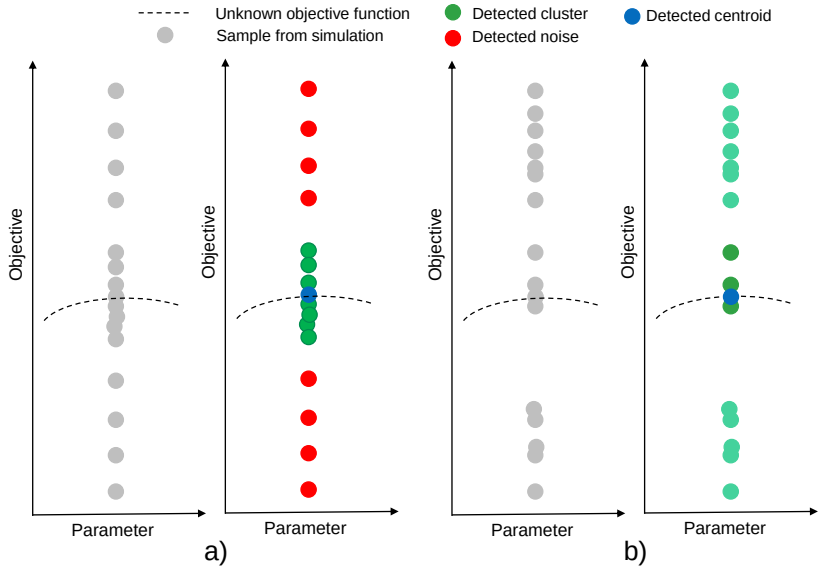


Figure 5.10: Two scenarios occur when applying DBScan for approximating an unknown objective function (dotted line): either it directly determines a close core cluster (a) or it determines several clusters (b). In both cases, I apply a heuristic for finding the appropriate DBScan core cluster. This heuristic either chooses the middle cluster if three clusters are formed or the mean value if only one cluster is found.

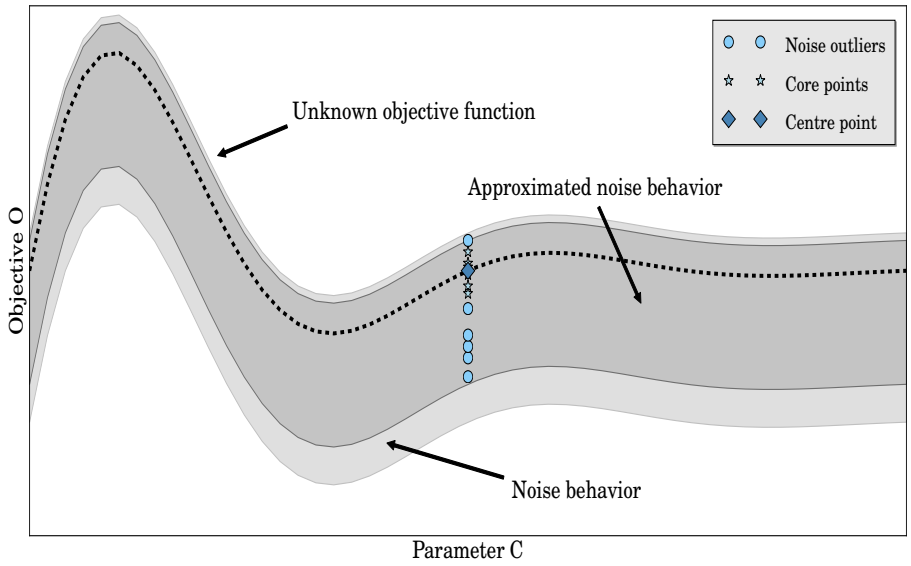


Figure 5.11: Approximating an unknown objective function with my density spline: Noise outliers are detected and the cluster centre point is used to construct the spline. Subsequently, the variance Θ of the clustering is propagated through the spline. This variance approximates the noise behavior of the unknown objective function.

In order to implement above concept, I utilize a property of spline-based interpolation. When interpolating with splines, the spline can change drastically when updated with new sample points at interpolated gradient minima and maxima. This is due to the fact that splines ensure that the first and second derivative of the spline will match at the knot points. Therefore, it is desirable to determine the spline gradient while approximating the unknown objective function in order to find large spline gradient changes. These gradient changes are used to draw a new sample from the parameter space which will more likely change the adjacent spline knots and therefore reduce the amount of required samples due to the aforementioned inherent behavior of spline interpolation (see Algorithms 18,19 and Figure 5.12).

Algorithm 18 Objective function approximation via density splines splineApprox(amount of samples $\mathcal{N}$, parameter $\mathcal{C}$ with space $\mathcal{K}$, spline error threshold ϵ, simulation time $\mathcal{T}$)

$\mathcal{D} = c_0, c_{\frac{k}{2}}, c_k \in \mathcal{C}$, sampling configurations
$\mathcal{F}$ = simulation results of $\mathcal{D}$ with $\mathcal{N}$ samples
$\mathcal{K}$ = Dbscan clusters of $\mathcal{F}$
$\mathcal{S}$ = spline based on $\mathcal{D}, \mathcal{K}$
$\mathcal{R}$ = amount of remaining samples: $k - 3$
$\mathcal{E}$ = empty list of rejections
while $\mathcal{R} > 0$ **and** $\mathcal{E} < \epsilon_{rejections}$ **do**
  $\mathcal{C}$ = gradientConfiguration($\mathcal{S}$,$\mathcal{D}$)
  $\mathcal{D}$ += $\mathcal{C}$
  $\mathcal{O}$ = empty list of simulation results
  for $\mathcal{N}$ samples **do**
    $\mathcal{O}$ += simulation result of $\mathcal{C}$ at $\mathcal{T}$
  end for
  $\mathcal{P}_{spline} = \mathcal{S}(\mathcal{C})$
  $\mathcal{A}$ = largest Dbscan cluster of $\mathcal{O}$
  $\mathcal{P}_{sim}$ = centre of $\mathcal{A}$
  $\mathcal{P}_{\Theta}$ = variance of $\mathcal{A}$
  $\mathcal{P}_p = 1$
  if $| \mathcal{P}_{sim} - \mathcal{P}_{spline} | < \epsilon_{deviation}$ **then**
    $\mathcal{E}$ += $\lceil$
  end if
  $\mathcal{S}$ = rebuild spline based on $\mathcal{D}, \mathcal{O}$
  $\mathcal{R} = \mathcal{R} - 1$
end while
return $\mathcal{S}$

Algorithm 19 Sampling of the parameter space based on gradient information: gradientConfiguration(current spline definition $\mathcal{S}$, sampled configuration $\mathcal{D}$)

$\mathcal{C}$ = return configuration
$\mathcal{T}$ = maximum threshold: 0
for $\mathcal{D}_1, \mathcal{D}_2, \in \mathcal{D}$ **do**
  $\nabla_{\mathcal{D}_1} = \dot{\mathcal{K}}(\mathcal{D}_1)$
  $\nabla_{\mathcal{D}_2} = \dot{\mathcal{K}}(\mathcal{D}_2)$
  if $|\nabla_{\mathcal{D}_1} - \nabla_{\mathcal{D}_2|} > \mathcal{T}$ **then**
    $\mathcal{C} = \frac{d_1 + d_2}{2}$
    $\mathcal{T} = |\nabla_{\mathcal{D}_1} - \nabla_{\mathcal{D}_2}|$
  end if
end for
return $\mathcal{C}$

After successfully approximating the unknown objective function, I combine the variances of all clusters for each spline in order to gain an accurate approximation for the noise distribution of the relationships. Given two clusters, C_1 and C_2, with their respective mean and variance of their largest cluster, $\bar{X_1}, \bar{X_2}$ and ${S_1}^2, {S_1}^2$ with n_1, n_2 observations, I compute the combined variance (see Equation 5.10). I iteratively apply this formula on all clusters of each spline and retrieve the overall variance of the spline approximation.

$$\begin{aligned} {S_c}^2 &= \frac{n_1 {S_1}^2 + n_2 {S_2}^2 + n_1(\bar{X_1} - \bar{X_c})^2 + n_2(\bar{X_2} - \bar{X_c})^2}{n_1 + n_2} \\ &= \frac{n_1[{S_1}^2 + (\bar{X_1} - \bar{X_c})^2] + n_2[{S_2}^2 + (\bar{X_2} - \bar{X_c})^2]}{n_1 + n_2} \end{aligned} \tag{5.10}$$

where

$${X_c}^2 = \frac{n_1 \bar{X_1} + n_2 \bar{X_2}}{n_1 + n_2} \tag{5.11}$$

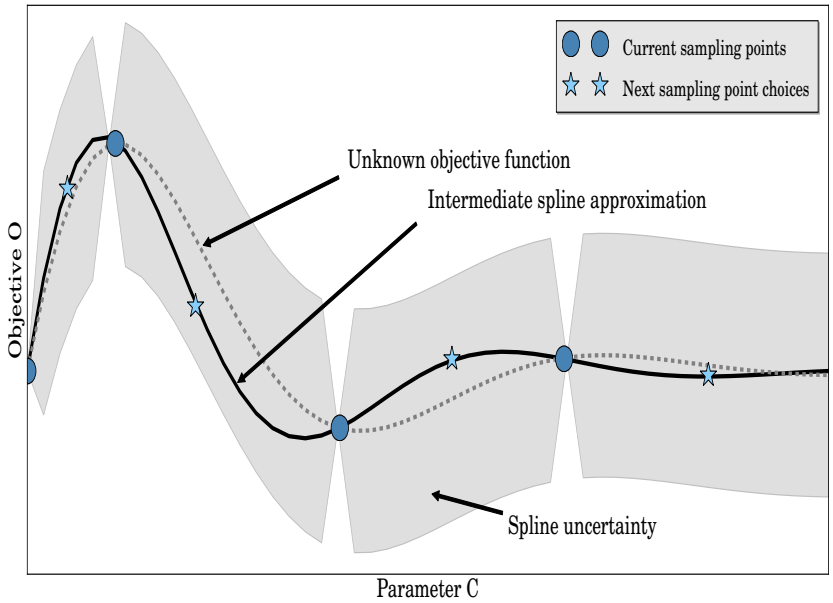


Figure 5.12: Spline approximation of an unknown objective function f: The spline is iteratively updated until it approximates the unknown objective function within a certain error degree. The spline uncertainty p is propagated through the spline. This uncertainty (with respect to unseen simulation behavior for the configurations) defines those spline segments which have been inter- or extrapolated.

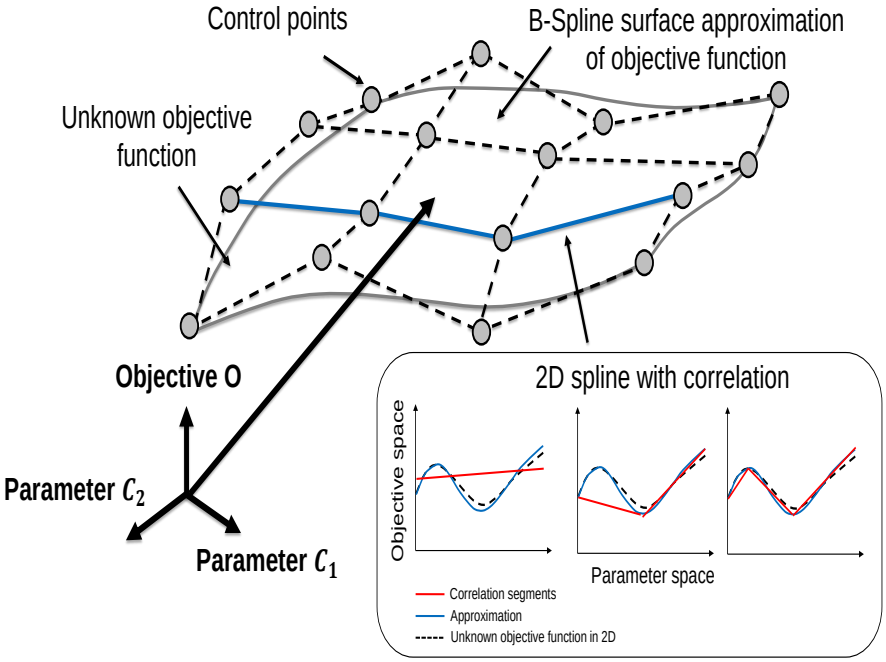


Figure 5.13: Checking the B-Spline surface as a whole for (partial) correlations is computationally complex. Therefore, I divide the surface into individual splines (blue line) and check for correlation for each spline. This check is performed efficiently in parallel across splines and the results are merged.

5.4.4 Recursive Correlation Analysis

After constructing the B-spline surface approximation of the objective function f output $o = \{o_o, ..., o_n\}$ for a given parameter space $c = \{c_o, ..., c_n\}$, the next step is to determine whether or not a causal relation is present between c and o. This correlation analysis is required because the B-spline surface itself does not contain any correlation or causality information and can therefore approximate any given signal.

Due to the fundamental property of self-containment of a simulation, confounding variables can be neglect at this step because they would also be part of the workflow which would be uncovered by my ARM approach (see Section 5.3). Therefore, correlation alone can be used to determine the causal relation between c and o. In order to do so, I split the B-spline surface into several cubic spline representations s_i for each simulation time step t_i. I approximate every s_i with segments which prove correlation, based on my RCA: if the complete cubic spline s_i can be represented with such segments, I assume correlation and therefore causality between c and o. This approach has the major advantage that I can check for correlation in parallel for every simulation step (see Figure 5.13).

Correlations between variables can be measured with the use of different indices (coefficients), such as the Pearson product correlation coefficient r [101]. It measures the linear correlation between two variables X and Y, giving a value between +1 and -1. +1 describes total positive correlation, zero no correlation and -1 total negative correlation.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5.12}$$

where

n : number of elements in X resp. Y
x,y : elements of X and Y
$\bar{x}, \bar{y}$: sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ (analogously for $\bar{y}$)

However, non-linear objective functions (e.g. polynomials with degree ≥ 2) can not be correctly modelled with r. I therefore propose to recursively compute the Pearson coefficient with my RCA approach: if the signal can not be described with r, I recursively split the signal in the middle and analyze the remaining signals. The causal relation is proven when the complete signal can be described with $\{r_k\}$, where k is the number of coefficients. Algorithm 20 and Figure 5.14 illustrate this concept:

Algorithm 20 RecursiveCorrelationAnalysis($\mathcal{S}$ spline)

1: $\mathcal{C}$ empty correlation segments
2: r_{xy} = pearson coefficient of $\mathcal{S}$[0,n]
3: **if** $|r_{xy}| >= r_{threshold}$ **then**
4: $\quad\mathcal{C}$ += $\mathcal{S}$[0,n]
5: **else**
6: $\quad\mathcal{C}$ += recursiveCorrelationAnalysis($\mathcal{S}[0,\frac{n}{2}]$)
7: $\quad\mathcal{C}$ += recursiveCorrelationAnalysis($\mathcal{S}[\frac{n}{2},n]$)
8: **end if**
9: **return** $\mathcal{C}$

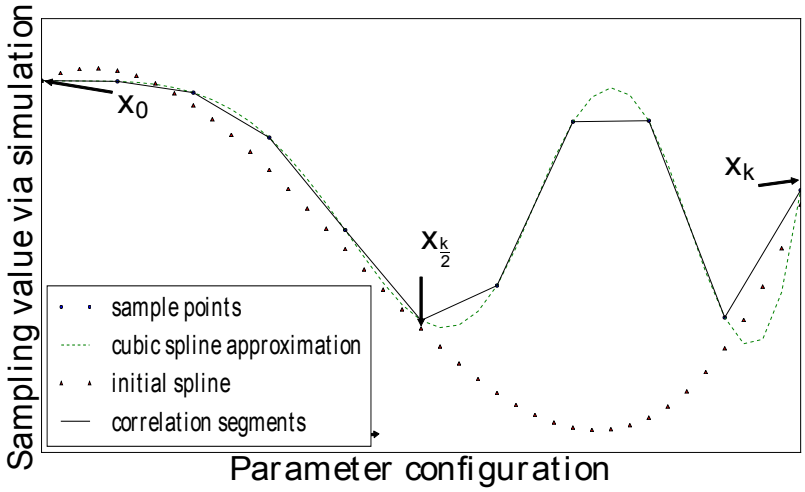


Figure 5.14: Result of the RCA: Segments which prove correlation are computed for the spline representation of the objective function.

Figures 5.15 and 5.16 illustrate the intermediate results from our KDP for a simple MOP based on four variables x_0, x_1, x_2, x_3 and noise N:

$$\begin{aligned} o_0 &= x_0 * sin(x_0) - 5 * x_1 + N \\ o_1 &= x_3^2 + -x_2^2 - 10 * cos(x_0) + N \end{aligned} \tag{5.13}$$

In this example, three out of the four parameters have a direct influence on o_0 and/or o_1 The goal of my KDP in these scenarios is to approximate the relationships of x_0, x_1, x_2, x_3 and o_0, o_1, as described in the previous sections. The figures show the individual relationships of x_0, x_1, x_2, x_3, the combined relationships of x_0, x_1, x_2, x_3 and the resulting B-spline surfaces with the approximated feasible solutions per parameter x_i. These B-spline surfaces are then used for MOO (see Section 6).

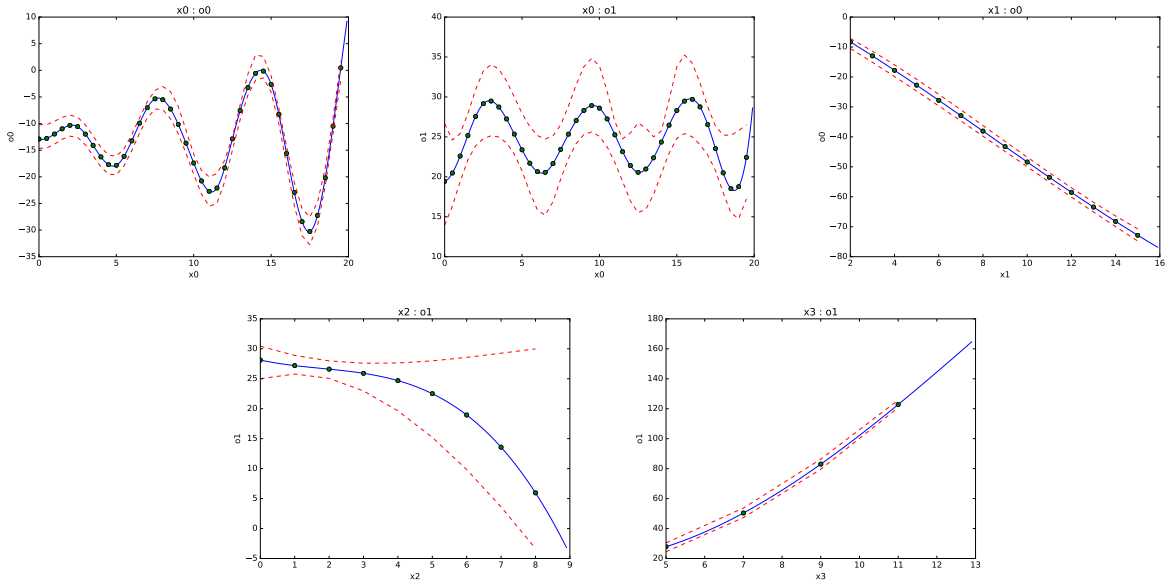


Figure 5.15: Two-dimensional approximation (straight blue line) of one parameter and one objective, the confidence interval is given by the red dotted lines.

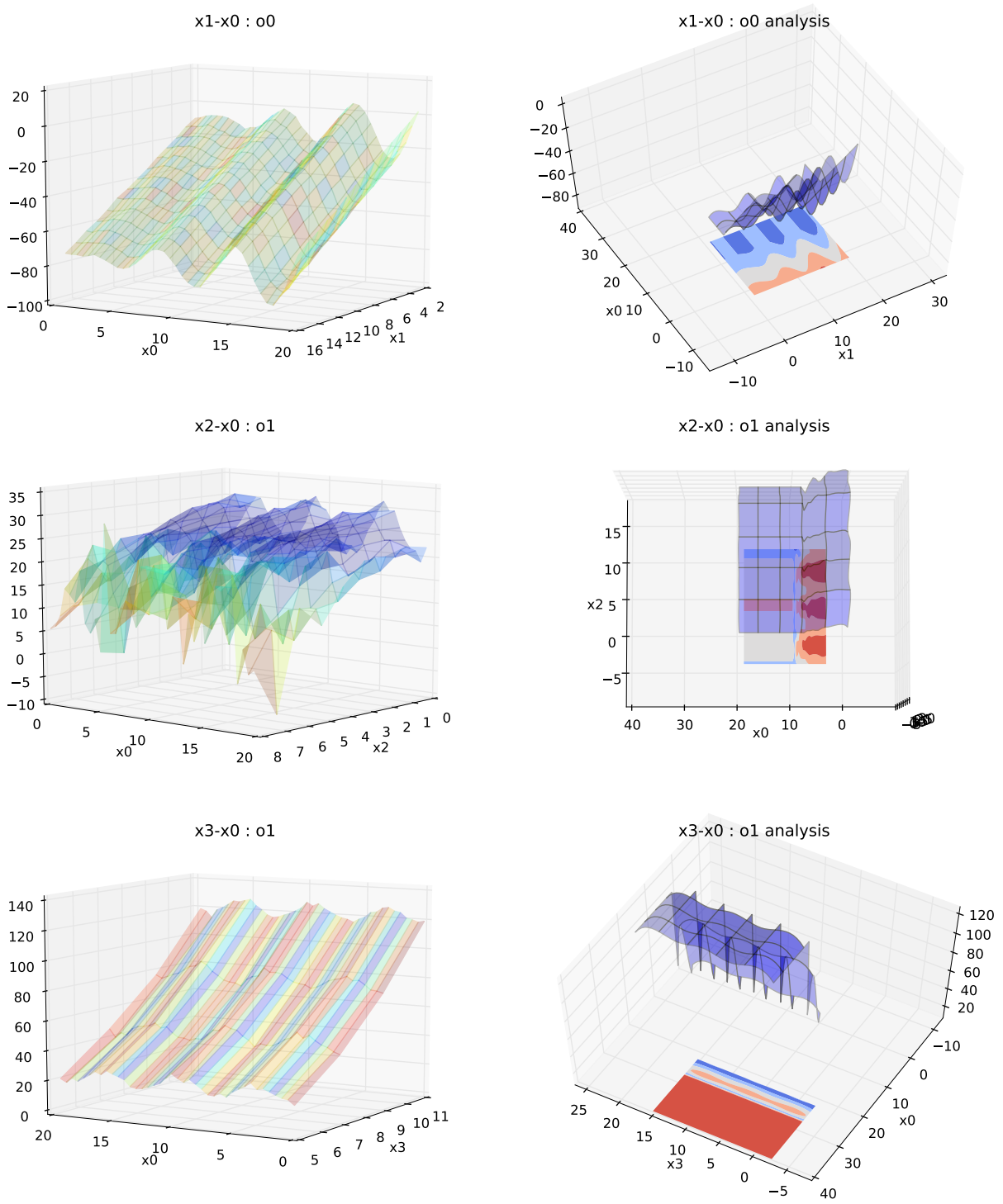


Figure 5.16: (Left) Combining the two-dimensional intermediate results from 5.15 for every parameter configuration lead to three-dimensional approximations. The computed variance of the approximation from my GDS approach is colored coded. (Right) The B-spline surfaces of the rough approximations from the left smooth the approximation and are used to compute feasible solutions (projected red area).

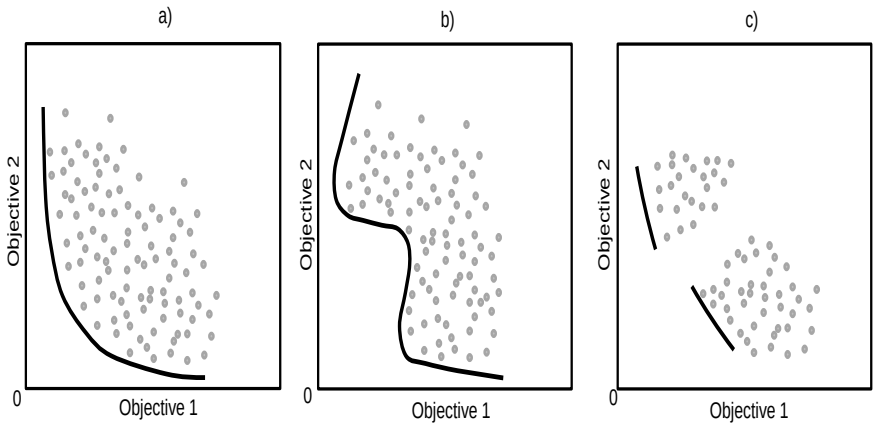


Figure 5.17: Illustration of Pareto optimal solutions: concave (a), convex (b) or interrupted Pareto fronts (c) can occur. My approach efficiently approximates all possible frontiers.

5.5 Multiobjective Optimization

As previously outlined simulation models are dominated today by a MOP. Within simulation based MOPs, engineers are interested in several different optimal solutions, e.g. which are reliable in many scenarios or which maximize the objective function for certain aspects. Therefore, my approach enables a Pareto solution of a MOP with three different optimization strategies which determine different FDSs while maintaining Pareto efficiency:

- Compliance strategy: Determination of the parameter space which maximizes or minimizes the objective function.
- Reliability strategy: Determination of the parameter space which maximizes the sampling probability.
- Closeness strategy: Determination of the parameter space which minimizes the clustering variance.

Due to my relationship definition, I can substitute the unknown objective functions $f_j(x), j = 1, ..., p$ from Equation 2.1 with my B-spline surface approximation:

$$\begin{aligned} &(MOP)\ \max_{x \in X} F(x) = (f_1(x), f_2(x), ..., f_p(x)) \\ &\qquad\qquad \Leftrightarrow \\ &(MOP)\ \max_{c,t \in C,T} F(c,t) = (s_1(c,t), s_2(c,t), ..., s_p(c,t)) \end{aligned} \tag{5.14}$$

My strategies determines a different FDS within above definition, namely a sub-set $\{\{c_i, ..., c_j\}, ..., \{c_n, ..., c_m\}\} = C_{Strategy} \subseteq \{c_o, ..., c_k\} = C_k$ with $o \leq i, i < j, n < m, j < n, m \leq k$. In detail, the compliance strategy (see Equation 5.15) maximizes objective function above a given minimum threshold, m, $\forall t \in T$ where T are all objective thresholds. In contrast to this, the reliability strategy (see Equation 5.16) and closeness strategy (see Equation 5.17) either maximize the probability p or minimize the variance Θ of each measurement. Depending on the strategy, different sub-sets are determined. Each sub-set is transformed into a FDS approximation ω, depending on the MOO constraints of the specific configuration parameter, namely the set of objective functions which are influenced by the parameter.

$$C_{compliance} = \{c_i | \forall s_q(c_i, T).F \geq m\} \tag{5.15}$$

$$(c_i, \overline{p_i}) = (c_i, \frac{\sum_{q=0}^{q=k} \sum_{n=0}^{t} s_q(c_i, t_n).p_i}{k}) \tag{5.16}$$
$$C_{reliability} = max_p\{(o_0, p_0), ..., (o_k, p_k)\}$$

$$(c_i, \overline{\Theta_i}) = (c_i, \frac{\sum_{q=0}^{q=k} \sum_{n=0}^{t} s_q(c_i, t_n).\Theta_i}{k}) \tag{5.17}$$
$$C_{closeness} = min_\Theta\{(o_0, \Theta_0), ..., (o_k, \Theta_k\}$$

$$\omega_i(c,t) = \sum_{p=0}^{p=k} \Theta_p \cdot |\frac{o}{n} - s_p(c,t) \cdot \frac{o}{\sum_{q=0}^{q=k} |t_q - s_q(c,t)|}| \tag{5.18}$$

where

$0 \leq o \leq 1$: weighting factor
k : the number of related objective functions of i
t : the objective threshold
Θ : the Pareto weighting factor
c : configuration from corresponding strategy

Based on $\omega_i(c,t)$ I can substitute every $s_i(c,t)$. Finally, I aggregated every information from my KDP into my FDS approximation:

$$\begin{gathered}(MOP) \max_{x \in X} F(x) = (f_1(x), f_2(x), ..., f_p(x)) \\ \Leftrightarrow \\ (MOP) \max_{c,t \in C,T} F(c,t) = (s_1(c,t), s_2(c,t), ..., s_p(c,t)) \\ \Leftrightarrow \\ (MOP) \max_{c,t \in C,T} F(c,t) = (\omega_1(c,t), \omega_2(c,t), ..., \omega_k(c,t))\end{gathered} \tag{5.19}$$

Consequently, my approach is able to find either a qualitative solution (see Equation 5.15), a reliable solution (see Equation 5.16) or the most dense solution (see Equation 5.17) that can be directly used in order to investigate the MOP from different perspectives.

The Pareto gradient $\nabla\omega$ is defined with unit vectors i, j, k which span the FDS:

$$\nabla\omega_{pareto} = \frac{\partial\omega}{\partial c}i + \frac{\partial\omega}{\partial t}j + \frac{\partial\omega}{\partial o}k \tag{5.20}$$

Figure 5.18 illustrates this concept for the compliance strategy in a simple two-dimensional example. This approach works for convex, concave and non-continuous Pareto fronts (see Figure 5.17).

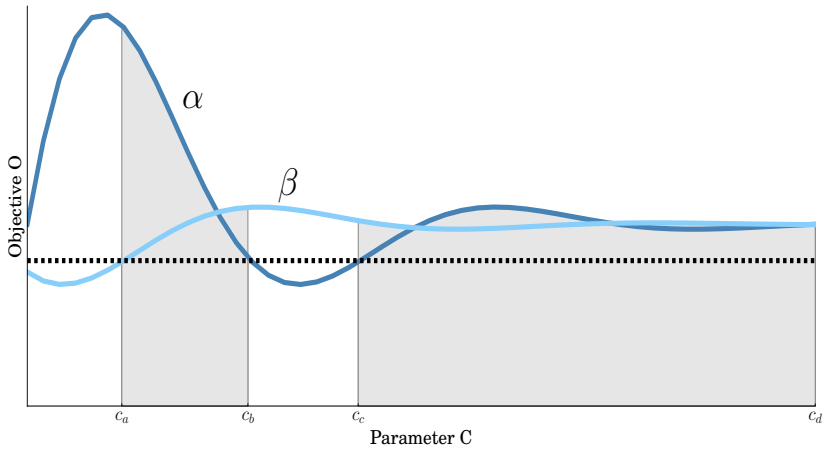


Figure 5.18: Approximation of the FDS for a parameter space, simulation objectives (α, β) and given minimum objective value (dotted line): $(c_a, c_b), (c_c, c_d)$ determines the feasible parameter configurations with $\Theta_\alpha = \Theta_\beta = 1$.

5.6 Use Case Studies

I present two diverse use case studies from the fields of engineering and biology to illustrate the broad range of applications of my presented approach. First, I optimize a interplanetary cruise flight trajectory of a spacecraft based on Kepler's orbital mechanics Second, I optimize a prey-predator system using the non-linear differential Lotka-Volterra equations [11]. These equations are frequently used to describe the dynamics of biological systems in which to species interact, one as a predator and the other as prey. In both use cases, the model behavior is unknown to my KDP. Known to my approach are only the available input parameters as well as simulation objective measurements. The goal in both use cases is to approximate the unknown objective functions in order to compute a optimal simulation model input configuration.

5.6.1 Spaceflight Orbit Optimization

Scenario

Spaceflight navigation solutions, especially autonomous interplanetary cruise flights, usually use optical measurements of reference bodies (e.g. Sun, Earth, Mars, Jupiter) to estimate their position [14]. Onboard optical systems take pictures of these reference bodies with respect to stars with known celestial locations. These images are used to compute the angular position of a spacecraft with respect to the reference bodies. These measurements are essential as spacecraft self-localization is required throughout the complete mission [14]. Consequently, spacecraft trajectory calculation to target destinations has to consider possible reference bodies.

The main idea of this use case is to compute a cruise orbit for a interplanetary cruise flight to a target body in such a way that optical measurements to two reference bodies are guaranteed in order to ensure that spacecraft self-localization is possible.

Methodology

In celestial mechanics, Kepler's orbital elements can be used to uniquely identify a specific orbit in space. A Keplerian orbit is an idealized, mathematical approximation of an orbit for a particular time span.

Each Kepler orbit is defined with six elements (see Figure 5.19), namely eccentricity e, inclination i, semimajor axis a, longitude of ascending node Ω, argument of periapsis ω and mean anomaly at epoch M.

Solving Keplers equation for a given orbit defines the cartesian position $\vec{r}$ of the orbiting body (see Equation 5.21).

$$M = E - e \cdot sin(E) \tag{5.21}$$

where

M : mean anomaly
E : eccentric anomaly
e : eccentricity

Solving Keplers equation can be done with an appropriate method numerically, e.g. with Newton-Raphson:

$$E_n = E_{n-1} - \frac{E_{n-1} - e \cdot sin(E_{n-1}) - M}{1 - e \cdot cos(E_{n-1})} \tag{5.22}$$

Using mean and eccentric anomaly, the orbiting body position, $\vec{r}$, is defined by the support vectors $\vec{P}, \vec{Q}$:

$$\vec{P} = \begin{cases} cos(\omega) \cdot cos(\Omega) - sin(\omega) \cdot sin(\Omega) \cdot cos(i) \\ cos(\omega) \cdot cos(\Omega) + sin(\omega) \cdot cos(\Omega) \cdot cos(i) \\ sin(\omega) \cdot sin(i) \end{cases} \tag{5.23}$$

$$\vec{Q} = \begin{cases} -sin(\omega) \cdot cos(\Omega) - cos(\omega) \cdot sin(\Omega) \cdot cos(i) \\ -sin(\omega) \cdot sin(\Omega) + cos(\omega) \cdot cos(\Omega) \cdot cos(i) \\ cos(\omega) \cdot sin(i) \end{cases} \tag{5.24}$$

$$\vec{r} = a \cdot (\vec{P} \cdot (cos(E) - e) + \sqrt{1 - e^2} \cdot \vec{Q} \cdot sin(E)) \tag{5.25}$$

The two target celestial bodies for navigation purposes and the spacecraft are represented by their Keplerian orbits around the Sun
$k_1 = \{e_{k1}, a_{k1}, i_{k1}, \omega_{k1}, \Omega_{k1}, M_{k1}\}$, $k_2 = \{e_{k2}, a_{k2}, i_{k2}, \omega_{k2}, \Omega_{k2}, M_{k2}\}$,
$k_{sc} = \{e_{ksc}, a_{ksc}, i_{ksc}, \omega_{ksc}, \Omega_{ksc}, M_{ksc}\}$.

The required field of view of the sensor measurements are represented by

$$\cos\alpha = \frac{(\vec{r_{target}} - \vec{r_{sc}}) \cdot \vec{p}}{|\ (\vec{r_{target}} - \vec{r_{sc}})\ | \cdot |\ \vec{p}\ |} \leq t \tag{5.26}$$

where

$\vec{p}$: the aligned payload vector
t : the allowed angle of the payload field of view
$\vec{r_{sc}}$: the spacecraft position
$\vec{r_{target}}$: the target position

The aim of my KDP is to approximate the impact of different Keplerian orbit configurations for $\vec{p}, \vec{r_{sc}}$ and $\vec{r_{target}}$ with the aim of maximizing the cruise flight period in which $\cos\alpha \geq t$.

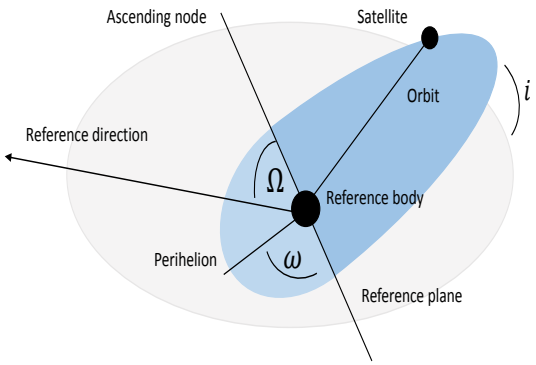


Figure 5.19: Illustration of the six Keplerian orbital elements. that uniquely describe a orbit of an celestial object based on Keplerian movement.

5.6.2 Lotka-Volterra Optimization

Scenario

The prey-predator system is a widely used simulation model of biological systems in which two species interact with each other. It consists of a dynamical non-linear system modeled by two differential equations, known as the Lotka-Volterra equations [11]. The equations model the evolution of two populations evolving in a common environment: prey and predators. Predators need to consume prey to survive, and prey spontaneously reproduce.

Due to the non-linear behavior of the Lotka-Volterra equations and further constraints (e.g. environmental conditions as the seasons which affect birth and death rate of the species) which can be added to the model, determining suitable input for observed real world ecosystem data is a challenging problem [108]. Therefore, the main idea of this use case is to determine a suitable input parameter set for the Lotka-Volterra equations in order to achieve a steady state between prey and predators for a given time span.

Methodology

The Lotka-Volterra model involves fmy parameters:

α : prey reproduction rate
β : prey death rate due to predators
δ : predators death rate in absence of prey
γ : predators reproduction rate according to consumed prey

The population evolution is given by these two differential equations:

$$\frac{dx(t)}{dt} = x(t)(\alpha - \beta y(t)) \tag{5.27}$$

$$\frac{dy(t)}{dt} = -y(t)(\delta - \gamma x(t)) \tag{5.28}$$

where $x(t)$ is the prey population at time t and $y(t)$ is the predator population at time t (see Figure 5.20). The aim of my KDP is to approximate the impact of different $\alpha, \beta, \delta, \gamma$ configurations for stabilizing the population of prey and predators at a static level.

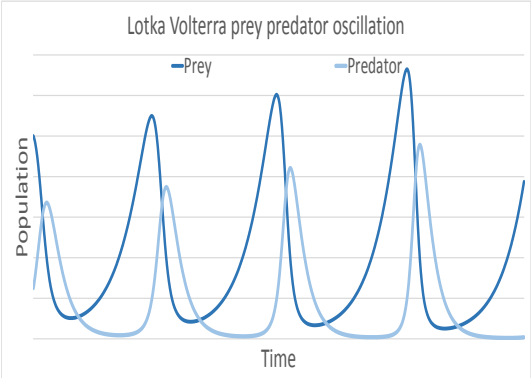


Figure 5.20: Illustration of the Lotka-Volterra equations: periodic oscillation between prey and predators occur.

5.7 Results

I implemented my KDP approach in C++. I performed experiments on a machine with an Intel Core i7 4-core processor (3.4 GHz) with enabled Hyperthreading, using the Microsoft Visual C++ 14 compiler with all optimizations, operated by Windows 7 64 bit and 8GB of RAM

I applied different experiments to measure the performance as well as the quality of my approach. For the quality measurement, I used the use case scenarios described above. Both use case simulations were used to evaluate whether the computed Pareto gradient information are suitable or not for converging towards the Pareto front.

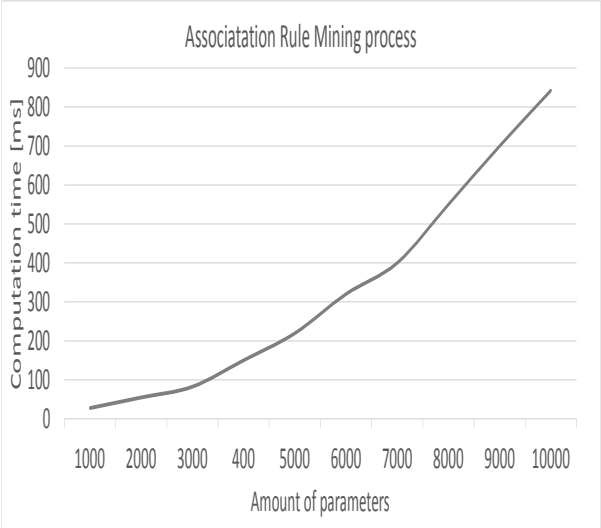
However, both scenarios are relatively small and can be hardly used to evaluate the performance of my approach. Hence, I additionally implemented a synthetic benchmark for performance measurements. I included three standard optimization algorithms: gradient descent, simulated annealing and evolutionary algorithms. The synthetic benchmark is based on blackbox simulations. I generated random objective functions for arbitrary simulation input parameters with mixed polynomials up to degree ten. These objective functions are further linked arbitrary times together with various simulation input parameters in order to generate multiobjective constraints.

Such MOPs do not have a single, accepted measure for solution quality [76], in contrast to single objective optimizations that have single global optimum, it is more complicated to measure the quality of any solution produced by a optimization algorithm.

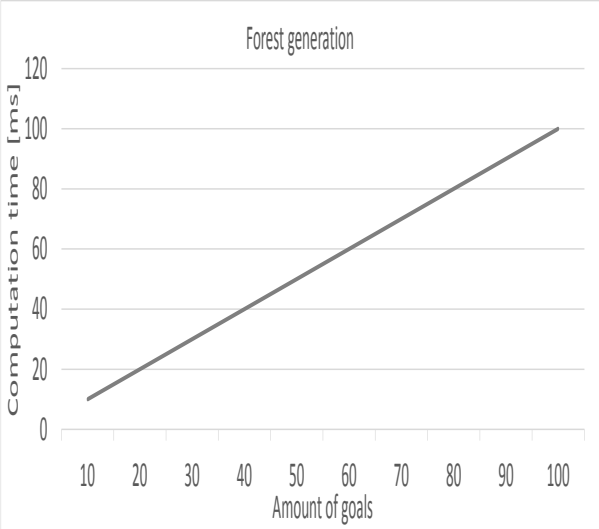
I use the GD (Generational Distance) measurement [39] that provides an estimation of the distance of the current solution to the Pareto front. In other words, $GD = 0$ indicates that all solutions are placed on the Pareto front. I first compute the minimum Euclidean distance $\delta_i, i = 1, 2, \ldots, n_p$ of each solution where n_p is the number of solutions found. The Generational Distance is then defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n_p} {\delta_i}^2}}{n_p} \tag{5.29}$$

Figure 5.21a shows the mean average computation time for the implemented ARM approach in my synthetic benchmark. My approach generates level-1-itemsets for more than 10.000 simulation input parameters in less than a second (see Figure 5.21a). Moreover, my approach is able to generate my forest data structure of up to 100 simulation objectives for these 10.000 parameters in less than 100 milliseconds (see Figure 5.21b). Consequently, my approach is able to analyze large scale simulations very effectively.



5.21a: My customized Association Rule Mining approach is able to analyze several thousands of parameters for generating level-1-itemsets in less than a second.



5.21b: My forest generation algorithm is able to generate the corresponding tree structures for each objective in less than 100 milliseconds for more than 100 objectives.

All three optimization algorithms I tested directly benefit from my Pareto information. Here, I compared how close the algorithms optimize towards the Pareto front when using my provided Pareto gradient information or not. When using the provided Pareto gradient information from my approach, the algorithms find solutions closer to the Pareto front by up to 38% for gradient descent, 44% for simulated annealing and 81% for a evolutionary approach (see Figure 5.22).

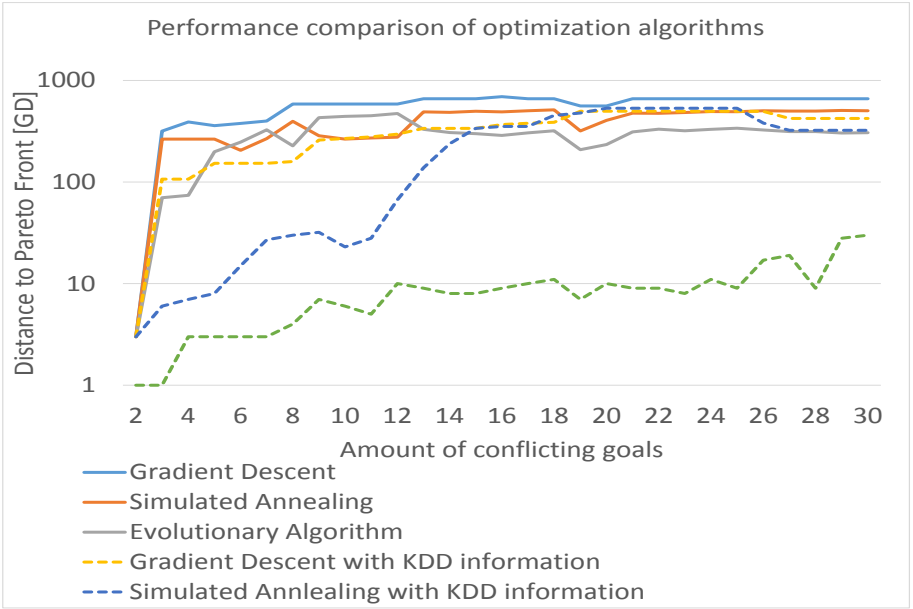
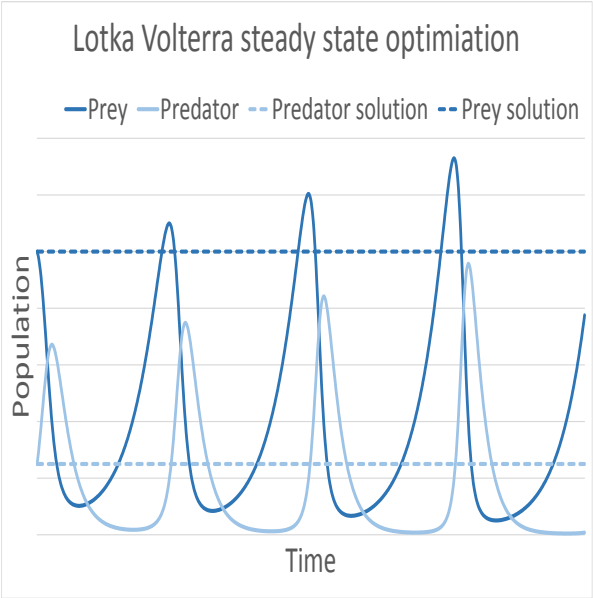
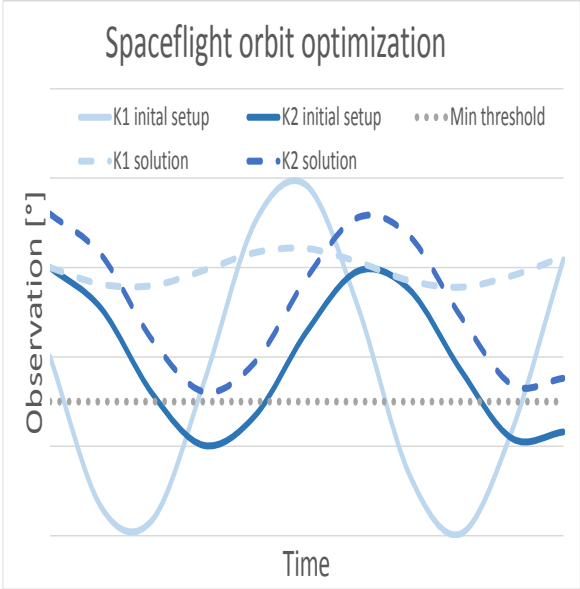


Figure 5.22: My Pareto information directly benefit gradient descent, simulated annealing and evolutionary approaches and deliver results closer to the Pareto front.



5.23a: Evaluation of the Lotka-Volterra use case study: my approach is able to compute a Pareto solution in order to achieve a steady state in the population.



5.23b: Evaluation of the spacecraft flight use case study: my approach is able to compute a Pareto solution in order to achieve the desired observation orbit.

Surprisingly, evolutionary algorithms benefit most from my Pareto information. I believe, that with an increasing number of conflicting objectives, even my Pareto space inherits many local minima, which adversely affect gradient descent and simulated annealing.

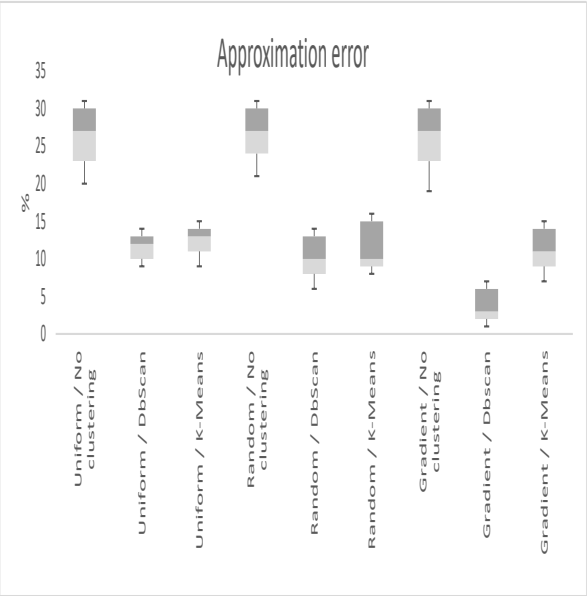
I used my use case scenarios to measure the quality of my approach. In the prey predator system, my approach was successfully able to compute a suitable input parameter set in order to achieve a steady simulation state. Figure 5.23a shows initial and the optimized model behavior. In my space flight scenario, my approach obtained an optimized flight orbit of the spacecraft with respect to the needed sensor measurements (see Figure 5.23b). In both use case studies, my approach computed suitable solutions based on my Pareto space and it achieved the desired simulation objective state.

I applied different experiments to measure the performance and quality of my B-spline approach within several synthetic benchmark scenarios. These synthetic benchmarks are based on generated blackbox simulations. Consequently, the objective functions of the simulation are unknown to all evaluated algorithms. The synthetic benchmarks compared the performance of my GDS algorithm to the approximation approach from [153] and different clustering (DBScan, k-means) and sampling approaches (uniform, random, gradient). In order to perform this evaluation, I generated two different types of random objective functions based on polynomials (f_p) and Gaussian functions (f_g).

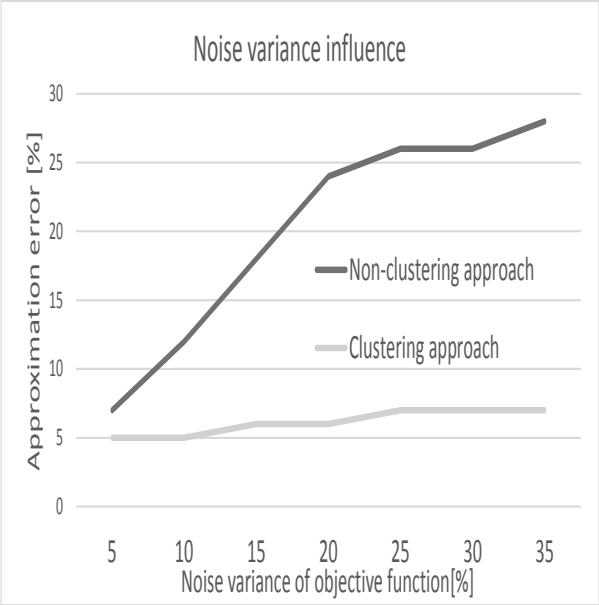
Furthermore, I added noise terms (N) for ten different noise distributions in order to obtain a stochastic simulation behavior. a, b, c, p, q are the known scalar values for each corresponding function:

$$f_p(c,t) = \sum_{i=0}^{n} a_i (c-p)^i + \sum_{j=0}^{m} b_j (t-q)^j + N \tag{5.30}$$

$$f_g(c,t) = \sum_{i=0}^{n} a_i e^{-\frac{(c-b_i)^2}{2c_i^2}} + \sum_{j=0}^{m} b_j e^{\frac{(t-b_j)^2}{2c_j^2}} + N \tag{5.31}$$



5.24a: My approach (Gradient/Dbscan) approximates unknown objective functions with less error and smaller error variance than its' competitors.



5.24b: Clustering based approaches do not suffer as much as non-clustering approach from the objective function noise behavior.

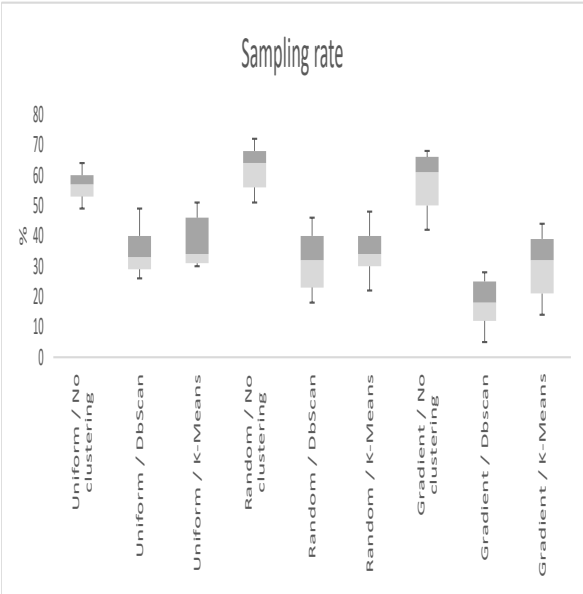
Based on above equations, I have evaluated my approach with more than 100 different versions of f_p and f_g in order to obtain a profound evaluation. These synthetic benchmarks have been supplemented by a qualitative evaluation in order to evaluate whether or not my B-spline surfaces can be used for optimization purposes.

I compared the mean approximation error for approximating an unknown polynomial objective function with a 30% noise variance (see Figure 5.24a) and a relationship space of 10.000. My GDS approach is able to outperform all its competitors up to a factor of four. This performance boost also increases with the noise variance of the unknown object function (see Figure 5.24b). This evaluation shows that the non-clustering approaches perform worse than any clustering-based approach.

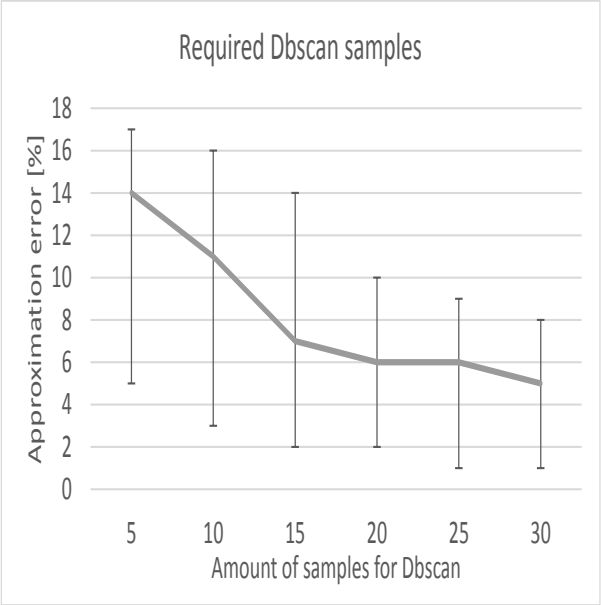
Even more, the error variance of my GDS approach is the smallest, especially when comparing to the uniform sampling approach without clustering [153] as well as for the random sampling. Therefore, it can be observed that a clustering is inevitable in order to approximate the unknown objective function accurately. In addition to the mean error evaluation, Figure 5.24b shows the mean average approximation error for an increasing noise behavior of the unknown objective function. It can be clearly observed that the clustering approaches adapt very well to an increased noise behavior of the unknown objective function while all other approaches clearly decrease in their performance.

Furthermore, all non-clustering based approaches need more samples than the clustering based competitors, when comparing their performance for the same required approximation error threshold (see Figure 5.25a). In this context, Figure 5.25b shows the impact of the sampling amount with respect to the clustering analysis. It can be observed that only a few samples ($\leq$ 12) per simulation configuration are required in order to precisely ($\leq$ 10%) approximate the given unknown polynomial objective function.

Overall, these evaluations strongly emphasize the quality improvement of my GDS approach with respect to its competitors. Furthermore, Figure 5.26 depicts the relationship between polynomial degree of the objective function and the GDS mean error of the objective function approximation. Surprisingly, my GDS approach is almost not affected by the polynomial degree of the objective function.



5.25a: My approach (gradient/Dbscan) requires less samples than its non-clustering based competitors. It further delivers both, best sampling variance and best approximation error.



5.25b: Only a few samples ($\leq$ 12) are required in order to efficiently ($\leq$ 10 % error) approximate the noise distribution of the synthetic unknown objective functions.

Therefore, even complex objective functions can be precisely approximated by my GDS approach. For deterministic simulations, all presented approaches perform very similar and obtain approximation errors less than 1% for arbitrary polynomial objective functions.

In summary, the following tables show a detailed overview of my synthetic benchmarks for both test functions f_p and f_g. It can be seen that my GDS algorithm outperforms its competitors for all given noise distributions in mean error.

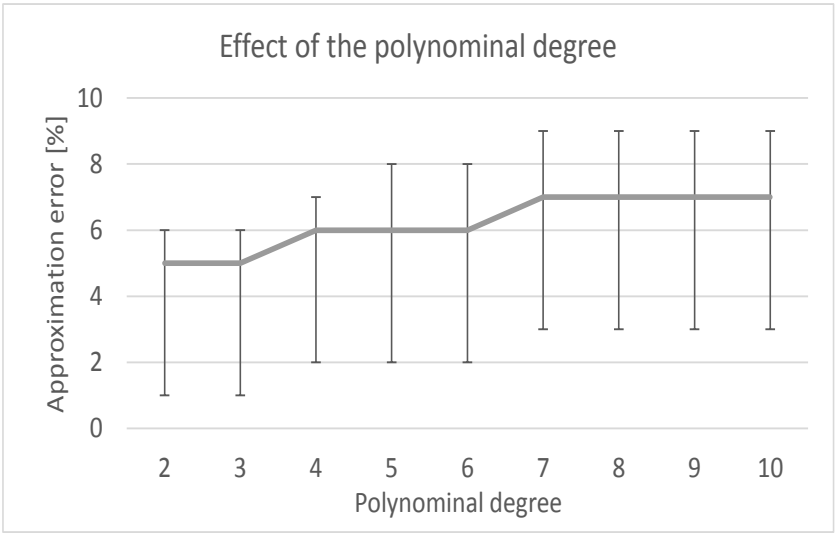


Figure 5.26: Effect of the polynomial degree of the unknown objective function on the mean error of my approach: the degree has almost no influence as the mean error varies around 6%.

Sampling:	**Uniform**					
Clustering:	**Det.**		**Dbscan**		**K-Means**	
	$\bar{e_p}$	$\bar{e_g}$	$\bar{e_p}$	$\bar{e_g}$	$\bar{e_p}$	$\bar{e_g}$
Probability mass functions						
Binominal, $t = 9.0, p = 0.5$ $P(i\|t,p) = \binom{t}{i} \cdot p^i \cdot (1-p)^{t-i}$	15.8	15.9	10.71	10.8	13.86	13.96
Geometric, $k = 0.3$ $p(i\|k) = k \cdot (1-k)^i$	15.66	15.71	10.61	10.67	13.34	13.58
Pascal, $k = 3.0, p = 0.5$ $p(i\|k,p) = \binom{k+i-1}{i} \cdot p^k \cdot (1-p)^i$	15.6	15.72	10.58	10.61	12.26	13.37
Uniform, $a = 0.1, b = 9.0$ $p(x\|a,b) = \frac{1}{b-a}$	15.33	15.56	10.29	10.41	13.44	13.49
Poisson, $\mu = 0.1$ $p(x\|\mu) = \frac{\mu^i}{i!} e^{-\mu}$	15.59	15.81	10.54	10.61	12.19	12.55
Probability density functions						
Cauchy, $a = 5.0, b = 1.0$ $p(x\|a,b) = \frac{1}{\pi \cdot b \cdot [1+(\frac{x-a}{b})^2]}$	15.1	15.3	10.44	10.48	12.42	12.56
Chi-squared, $n = 3.0$ $p(x\|n) = \frac{1}{\Gamma(\frac{n}{2}) \cdot 2^{\frac{n}{2}}} \cdot x^{\frac{n}{2}-1} \cdot e^{-\frac{x}{2}}$	15.7	15.9	10.61	10.63	12.11	12.46
Fisher-F, $m = 2.0, n = 2.0$ $p(x\|m,n) = \frac{\Gamma(\frac{m+n}{2})}{\Gamma(\frac{m}{2}) \cdot \Gamma(\frac{n}{2})} \cdot \frac{\frac{mx}{n}^{\frac{m}{2}}}{x \cdot (1+\frac{mx}{n})^{\frac{m+n}{2}}}$	15.35	15.51	10.84	10.88	12.71	12.83
Normal, $\mu = 5.0, \sigma = 2.0$ $p(x\|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	15.56	15.62	10.55	10.63	11.98	12.68
Exponential, $\lambda = 3.5$ $p(x\|\lambda) = \lambda e^{-\lambda x}$	15.65	15.7	10.62	10.67	13.46	13.66

Table 5.1: Synthetic performance comparison overview: my approach outperforms its competitors for all noise distributions (mean error $\bar{e}$ in %). (Part I)

Sampling:	**Random**					
Clustering:	**Det.**		**Dbscan**		**Dbscan**	
	$\bar{e_p}$	$\bar{e_g}$	$\bar{e_p}$	$\bar{e_g}$	$\bar{e_p}$	$\bar{e_g}$
Probability mass functions						
Binominal, $t = 9.0, p = 0.5$ $P(i\|t,p) = \binom{t}{i} \cdot p^i \cdot (1-p)^{t-i}$	16.68	16.71	10.55	10.59	13.42	13.56
Geometric, $k = 0.3$ $p(i\|k) = k \cdot (1-k)^i$	16.54	16.68	10.33	10.4	13.23	13.32
Pascal, $k = 3.0, p = 0.5$ $p(i\|k,p) = \binom{k+i-1}{i} \cdot p^k \cdot (1-p)^i$	16.27	16.32	10.32	10.4	13.67	13.7
Uniform, $a = 0.1, b = 9.0$ $p(x\|a,b) = \frac{1}{b-a}$	16.02	16.1	10.07	10.12	13.72	13.79
Poisson, $\mu = 0.1$ $p(x\|\mu) = \frac{\mu^i}{i!} e^{-\mu}$	16.42	16.51	10.3	10.38	13.21	13.32
Probability density functions						
Cauchy, $a = 5.0, b = 1.0$ $p(x\|a,b) = \frac{1}{\pi \cdot b \cdot [1+(\frac{x-a}{b})^2]}$	16.24	16.28	10.25	10.3	13.33	13.42
Chi-squared, $n = 3.0$ $p(x\|n) = \frac{1}{\Gamma(\frac{n}{2}) \cdot 2^{\frac{n}{2}}} \cdot x^{\frac{n}{2}-1} \cdot e^{-\frac{x}{2}}$	16.28	16.43	10.4	10.48	13.56	13.61
Fisher-F, $m = 2.0, n = 2.0$ $p(x\|m,n) = \frac{\Gamma(\frac{m+n}{2})}{\Gamma(\frac{m}{2}) \cdot \Gamma(\frac{n}{2})} \cdot \frac{\frac{mx}{n}^{\frac{m}{2}}}{x \cdot (1+\frac{mx}{n})^{\frac{m+n}{2}}}$	16.35	16.48	10.73	10.79	13.56	13.6
Normal, $\mu = 5.0, \sigma = 2.0$ $p(x\|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	16.4	16.53	10.39	10.44	13.67	13.72
Exponential, $\lambda = 3.5$ $p(x\|\lambda) = \lambda e^{-\lambda x}$	16.36	16.48	10.34	10.41	13.73	13.78

Table 5.2: Synthetic performance comparison overview: my approach outperforms its competitors for all noise distributions (mean error $\bar{e}$ in %). (Part II)

Sampling:	**Gradient**					
Clustering:	**Det.**		**Dbscan**		**K-Means**	
	$\bar{e_p}$	$\bar{e_g}$	$\bar{e_p}$	$\bar{e_g}$	$\bar{e_p}$	$\bar{e_g}$
Probability mass functions						
Binominal, $t = 9.0, p = 0.5$ $P(i\|t,p) = \binom{t}{i} \cdot p^i \cdot (1-p)^{t-i}$	15.65	15.72	8.67	8.71	11.34	11.41
Geometric, $k = 0.3$ $p(i\|k) = k \cdot (1-k)^i$	15.52	15.76	8.62	8.69	11.56	11.63
Pascal, $k = 3.0, p = 0.5$ $p(i\|k,p) = \binom{k+i-1}{i} \cdot p^k \cdot (1-p)^i$	15.65	15.86	8.53	8.6	11.23	11.38
Uniform, $a = 0.1, b = 9.0$ $p(x\|a,b) = \frac{1}{b-a}$	15.3	15.53	8.32	8.45	11.67	11.75
Poisson, $\mu = 0.1$ $p(x\|\mu) = \frac{\mu^i}{i!} e^{-\mu}$	15.56	15.62	8.56	8.61	11.85	11.93
Probability density functions						
Cauchy, $a = 5.0, b = 1.0$ $p(x\|a,b) = \frac{1}{\pi \cdot b \cdot [1+(\frac{x-a}{b})^2]}$	15.15	15.25	8.47	8.52	11.24	11.4
Chi-squared, $n = 3.0$ $p(x\|n) = \frac{1}{\Gamma(\frac{n}{2}) \cdot 2^{\frac{n}{2}}} \cdot x^{\frac{n}{2}-1} \cdot e^{-\frac{x}{2}}$	15.7	15.78	8.62	8.7	10.87	10.96
Fisher-F, $m = 2.0, n = 2.0$ $p(x\|m,n) = \frac{\Gamma(\frac{m+n}{2})}{\Gamma(\frac{m}{2}) \cdot \Gamma(\frac{n}{2})} \cdot \frac{\frac{mx}{n}^{\frac{m}{2}}}{x \cdot (1+\frac{mx}{n})^{\frac{m+n}{2}}}$	15.13	15.31	8.75	8.81	11.74	11.83
Normal, $\mu = 5.0, \sigma = 2.0$ $p(x\|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	15.52	15.59	8.48	8.54	11.52	11.6
Exponential, $\lambda = 3.5$ $p(x\|\lambda) = \lambda e^{-\lambda x}$	15.6	15.72	8.63	8.78	11.64	11.67

Table 5.3: Synthetic performance comparison overview: my approach outperforms its competitors for all noise distributions (mean error $\bar{e}$ in %). (Part III)

Science is not only a discipline of reason but, also, one of romance and passion.

Stephen Hawking

6

Multi-Agent System based Multiobjective Optimization

I present in this chapter how my FDS approximation and optimization strategies can be incorporated into a powerful optimization toolset for computing a Pareto solution for the simulation model input parameter space. The optimization system proposed here is based on a hierarchical MAS which aims at dynamically tuning all given input configuration parameters with respect to the approximated FDS. I recall some of the most relevant research articles that have appeared in the international literature related to this topic and emphasize my contributions.

The main task of virtual testbeds is to support engineers with simulation results, based on simulation scenario and simulation model configuration. Engineers are mainly interested in finding an optimal simulation model configuration. This means a configuration which is able to satisfy all given simulation objectives and requirements for given tresholds.

These simulation objectives and requirements are often contradictory and define only a subset of an overall system expectation. These system expectations are in general non-manageable non-technical system aspects which can not be easily formalized in a simulation. This leads to the aforementioned workflow problem (see Section 1.1.2) which drastically restrict engineers within their work.

MAS can effectively deal with such problems which are composed of a large number of interacting and contradictory sub-problems. Additionally, MAS allow a simpler modelling of the domain [108]. This is especially useful for solving the workflow problem because it conceptualizes the MOP which can not be formalized. Thus, using a MAS for a bottom-up modelling of the problem, based on my approximations of the unknown objective functions problem, is convincing. Furthermore, interactions between agents can give birth to emergent phenomena (e.g., patterns, organizations, behaviours) [108]. I consider the simulation model configuration optimization as a complex problem for which no specific solution exists beforehand. This makes MAS interesting for dealing with such problems for which no algorithmic solutions can be given in advance and therefore have to be designed in a bottom-up way.

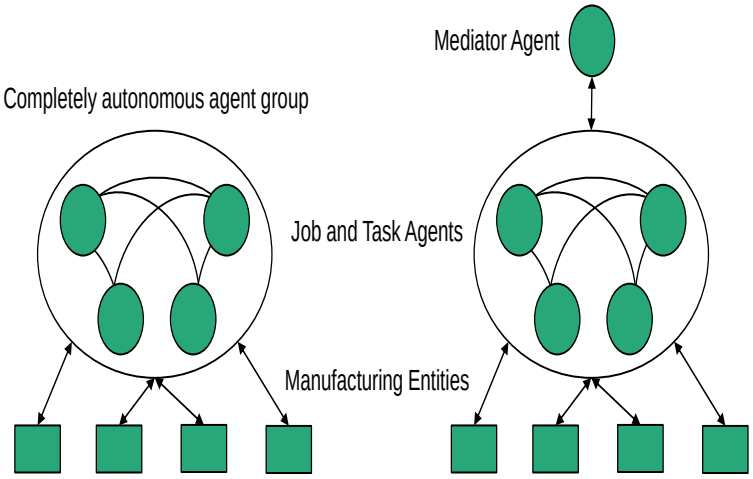


Figure 6.1: Classic MAS architectures: the autonomous agent-based architecture (left) and the mediator agent-based architecture (right). In the autonomous architecture the agents are not supervised by any hierarchy while the mediator framework introduces a hierarchical information exchange and operations.

Consequently, I propose my optimization system based on a modular hierarchical MAS in which self-organization principles are used to make the collective behavior emerge from local ones. I introduce with my optimization system (and corresponding FDS approximation) a novel type of virtual testbed which overcomes the aforementioned workflow challenge.

6.1 Related Work

Solving optimization problems with MASs emerged as a research field in the late 90's within the simulation of social behavior, e.g. based on particle swarm optimization [81] and is still a very active research field. There are numerous successful applications for these MASs, ranging from SOO to MOO for arbitrary use cases, e.g. scheduling problems, energy systems, autonomous architectures, transportation and logistics, and supply chain planning [112]. The main interest is to formulate the MOP more easily while maintaining an efficient process of converging. Various MAS approaches have been proposed in order to solve optimization problems, e.g. based on game theory [204] or on meta-heuristics for ant colony optimization [6]. Other approaches, e.g. [69] focussed only on convex objective functions. In addition, evolutionary aspects have been integrated into MASs for solving MOPs since 15 years [102]. [110] introduced extensions for evolutionary MASs for constrained MOO. Basically, they encoded constraints of an MOO into the evolutionary structure of MASs. In contrast, [165, 166] focussed on better agent population diversity by introducing the prey-predator model.

Approaches for basic system configuration adaptation (without the requirement of computing a solution to a MOP) can be found in other simulation related fields, e.g. [20]. Simulation and serious gaming is a strongly related field [134] in which system parameter adaptation is well-founded. Current adaptation concerns automated scenario generation, e.g. for military training purposes [20]. Such adaptive applications can be effectively modelled with MAS [92]. Furthermore, [93, 94] showed in general how agents can be used to define serious game applications. These approaches have in common that they rely on the classical approach of MAS based modelling or optimization purposes in which no information about the problem is available.

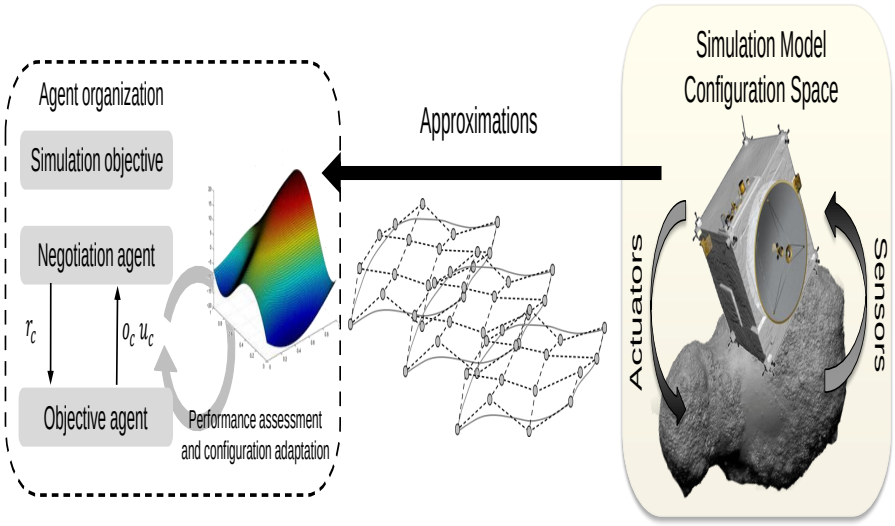


Figure 6.2: The proposed mediator-based MAS architecture operates in the approximated simulation model parameter space by changing the simulation model configurations based on gradient information from the GDS responses.

In general, above mentioned MAS approaches can be grouped into two classical architectures (see Figure 6.1). In completely autonomous MAS architectures, agents react locally to local changes, and interact directly with each other to generate global optimal and robust solution, e.g. for scheduling problems [50]. Despite the good performance of autonomous architectures, they usually face problems in providing globally optimized solutions in presence of a large number of agents (as highlighted by [147]), leading to above mentioned mediator based architecture in which mediator or negotiation agents influence the behavior and communication of other agents [112]. Therefore, I adopted above well-known techniques into a new MAS which directly operates within my FDS approximation based on a mediator agent-based scheduling architecture (see Figure 6.1).

6.2 Overview of Approach

Hierarchical MAS have already proven their feasibility for solving MOPs as previously described. My main idea is to show how my FDS can be incorporated in these approaches and that my wait-free data management further improves their performance due to wait-free communication between the agents. Based on the vision of my thesis, to conceptualize and implement an integrative approach, the MAS is implemented as a blackboard [42] with mediator agents (see Figure 6.2) in order to utilize the KeyValuePool or GraphPool. In order to accomplish SOO and MOO, every agent introduces a part-wise modelling (single and multiobjective constraints per input parameter) of the problem and its behavior and communication to other agents is used to solve the global (MOO) problem.

In the following, I describe at first the required MAS infrastructure with its modular agent organizations and their relationships to the FDS. Following this, I explain the input and output data as well as communication structure of the agents. At last, I outline the adaptation solving process with its negotiation mechanisms and how MOPs can be solved.

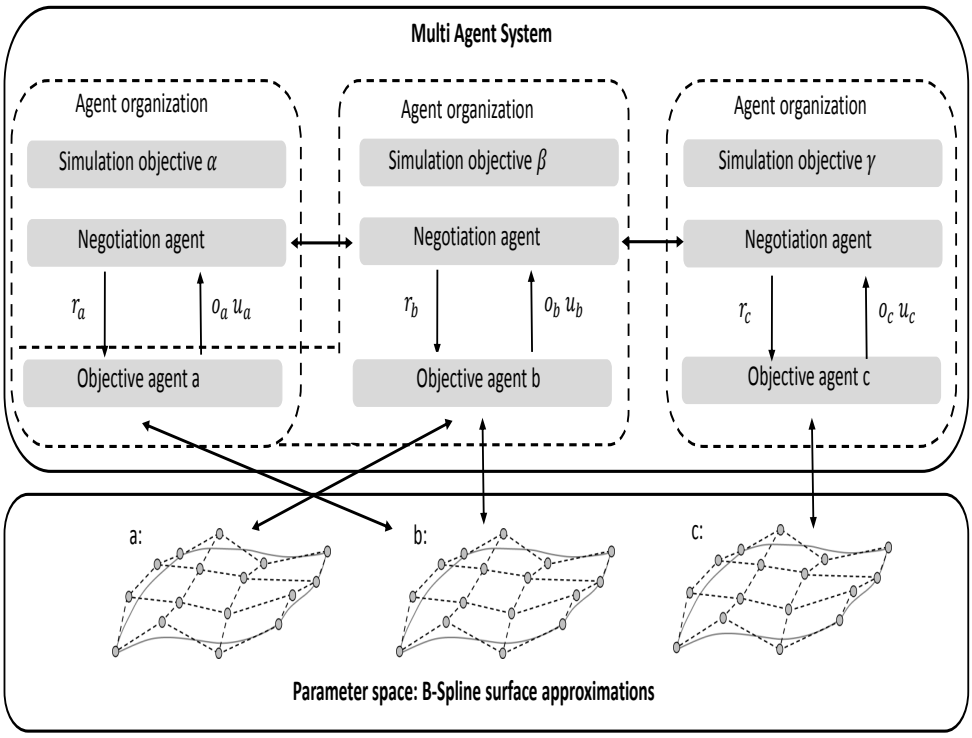


Figure 6.3: My MAS based optimization approach for a mixed objective problem statement (one multi-objective objective (β) and two single-objective problems (α, γ) with three input parameters): each agent organization optimizes the parameter set for one objective. Negotiation agents handle requests between the objective-agents in order to effectively find the optimal parameter configuration.

My MAS is composed of several agent organizations. Each of these organizations aims at optimizing a subset of configuration parameters to one or more simulation objectives, each one represented by my FDS approximation (see Figure 6.3).

These agent organizations are defined per specified simulation objective and consist of a hierarchy of two agent types: objective- and negotiation-agents. For each identified input parameter, one objective-agent is defined. Therefore, one objective-agent can belong to several agent organizations. The goal of every defined objective-agent is to maximize or minimize every attached simulation objective under Pareto constraints. Several optimization constraints arise because of the underlying MOP. Therefore, a negotiation-agent is defined for every specified objective. The goal of every negotiation agent is to manage requests between the objective-agents in order to satisfy the existing multi-objective constraints between the objective-agents.

Figure 6.3 illustrates this agent organization concept with respect to the overall proposed approach. The input and output data as well as communication structure of these agents is described in the next section. The idea is to use the emergence of the MAS in order to directly optimize towards the Pareto front (see Figure 6.4).

6.3 Parameters, Objectives and Utilities

Given a MOP, as defined in Equation 5.19, I can uniquely identify every adjustable input configuration parameter C with its valid range. These configuration parameters and the corresponding B-spline surface based FDS approximation constitute the input of my MAS. Additionally, I define the objectives and utilities of my MAS based optimization system.

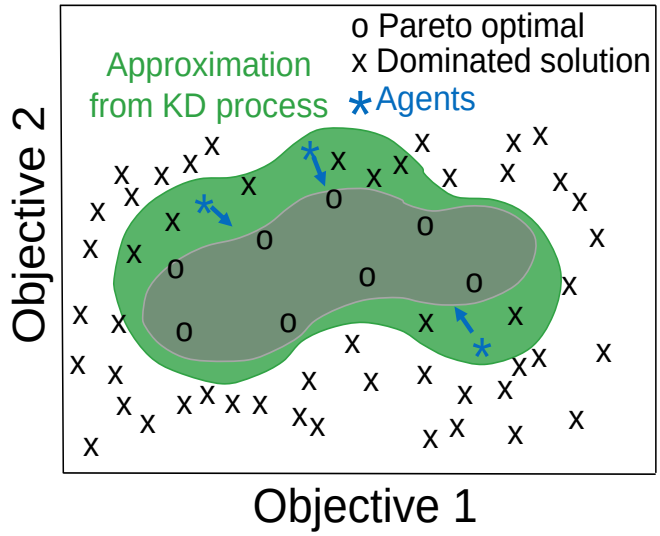


Figure 6.4: Visualization of my MAS approach. The agents are directly initialized close to the Pareto front because of the FDS approximation from my GDS approach. Further, the agents are forced to operate towards the Pareto front by using pre-computed Pareto gradient information from the response surfaces.

Each objective-agent strives for maximizing or minimizing each single-objective optimization problem of its attached input parameter under multi-objective constraints. Therefore, each objective-agents computes several objective values, one for each attached objective.

The set of all agent objectives OBJ is defined as follows:

$$OBJ = \{o_1 ... o_n\} \rightarrow \{0, 1\}$$

$$o_i = \omega(c, t) \tag{6.1}$$

where

ω : corresponding approximated FDS

In addition to the objectives to be satisfied, my MAS also considers the possible utility when changing the given parameter with respect to the current negotiation state among the agents. Consequently, I introduce a utility function. It is used to calculate the difference between current Pareto solution and highest achievable single-objective solution. This utility value is then later used by the negotiation-agent to select the most appropriate action.

The set of all utility values UTL is defined as follows:

$$UTL = \{u_1 ... u_n\} \rightarrow \{0, 1\}$$

$$u_i = s_i(c, t) - \omega(c, t) \tag{6.2}$$

where

s_i : corresponding approximated objective function
ω_i : corresponding approximated FDS

Therefore, the aim of my MAS based optimization system is to minimize *UTL* while maximizing *OBJ*. In other words, it is to tune all the parameters so all the constraints and objectives are satisfied.

In order to implement this efficiently in an agent-based organization, I define requests which are shared between the agents. These requests are used to lower or higher the Pareto weight or objective value threshold from Equation 5.18 if any agent has partially solved one objective:

$$\begin{aligned} REQ &= \{(\Theta_o, t_o), ...(\Theta_n, t_n)\} \\ \Theta_i &\rightarrow \{0, 1\} \\ t_i &\rightarrow \{0, 1\} \end{aligned} \tag{6.3}$$

where

Θ_i : corresponding Pareto weight for the associated objective
m_i : corresponding objective value threshold for the associated objective

6.4 Solving Process Principle

When designing a MAS based optimization process, the focus is set on agent behaviors and communications in order to cover the isolated parts of the global problem which each agent models. Each of my objective-agent tackles an isolated sub-problem (finding a solution to its attached single or multiobjective constraints) and emergence is used to solve the overall (MOO) problem. Therefore, the solving process is distributed among all objective-agents via negotiation-agents. Consequently, the definition of objective- and negotiation-agent behaviours is also one of the key aspects of my MAS and is described hereafter.

- Negotiation-agents monitor the objectives and utilities of all corresponding objective-agents of their agent organization and distribute requests between objective-agents: In the first step, they update (see below) the attached objective-agents if they have open requests. In this step, each objective-agent may achieve a new Pareto solution. In the second step, they collect new requests from each objective-agent and group them according to the multi-objective Pareto satisfaction:
  - a) if the objective is completely satisfied, it requests a change for the Pareto weight $\Theta = 0$ for every other attached objective-agent.
  - b) if the objective is partially satisfied, it requests a change for the objective threshold t in order of magnitude of current objective satisfaction for every other attached objective-agent.

  At last, the negotiation-agent will forward the requests (change in objective threshold t or Pareto weight Θ) of those objective-agents with the highest utility value.
- An objective-agent has a rather simple behaviour: it computes the objective and utility value for every attached objective for the current Pareto configuration (objective thresholds, Pareto weights and parameter configuration) based on my FDS approximation. It updates these values every time a request for change in Pareto weighting or objective threshold is received from an negotiation-agent.

6.5 Use Case Study: Spacecraft Landing Scenario

A spacecraft landing procedure is a very complex sequence of autonomous vehicle decisions, actuator commands and sensor data acquisition summarized as guidance, navigation and control. Such a landing procedure is an interesting testbed for my proposed approach as it consists of several environmental influences and vehicle internal control loops.

I modelled a simplified landing procedure within the previously introduced end-to-end spaceflight mission simulator (see Section 3.6). The aim of the procedure is to ensure a safely landing on an asteroid surface. The spacecraft is under the influence of several environmental forces which introduce trajectory perturbances: solar radiation pressure (Sun distance based tangential perturbance) and asteroid gravity (assuming a homogeneous gravity field).

The spacecraft itself is modelled with an internal control loop which acquires sensor data: acceleration (accelerometer), orientation (simplified Star Tracker) and distance to surface (range finder). The overall algorithmic internal spacecraft position estimation is simplified in such a way that ground truth data is used under application of Gaussian noise.

The spacecraft has three configurable parameters: main engine thrust level in Newton, fuel capacity in kg and control thrust ignition timings. Within the spacecraft control loop, the spacecraft autonomously fires its main engine, if the acceleration exceeds a given threshold, so that the landing velocity will be regulated. Additionally, the spacecraft continuously determines its orientation and fires its attitude thrusters, if the spacecraft orientation to the asteroid surface also exceeds a given threshold. Every time the main thrust or control thrust is fired, fuel is consumed and consequently the mass of the spacecraft is lowered.

The aim of my optimization is to dynamically tune this spacecraft configuration in order to avoid a crash at the asteroids surface as well as to ensure that enough fuel is left when the landing procedure is finished, to leave the asteroid again. Additionally, the orientation of the spacecraft has to be aligned to the asteroid surface so that the main thruster and landing gear should be perpendicular to the asteroids' surface.

In detail, the simulation objectives are defined as follows:

- The spacecraft velocity at touchdown should be less than 1 $\frac{m}{s^2}$. The thrust level must be between 1 - 2000 Newton.
- The spacecraft pose should be aligned with the landing terrain with less than 3 degree error. The maximum control thrust ignitions are 10 times within 1000 milliseconds.
- The spacecraft should have more than 10 % fuel left, when the landing procedure has ended. The fuel capacity must be between 500 - 1500 kg.

6.6 Results

I have implemented my MAS based optimization approach in C++ and CUDA 7. I performed experiments on a machine with an Intel Core i7 4-core processor (3.4 GHz) with enabled Hyperthreading, using the Microsoft Visual C++ 14 compiler with all optimizations, operated by Windows 7 64 bit and 8GB of RAM

6.6.1 Spacecraft Landing Scenario

I modelled, based on the previously introduced spacecraft application, my MAS based optimization and virtual testbed (see Chapter 4). This test scenario was used to evaluate whether or not the proposed MAS based optimization solves for correct adaptations of the spacecraft configuration.

My evaluation shows that the MAS is able to maximize the given simulation objectives until the simulation successfully ends (see Figure 6.5). The optimization required seven simulation runs (each containing 700 - 1000 simulation steps) in order to maximize all three simulation objectives.

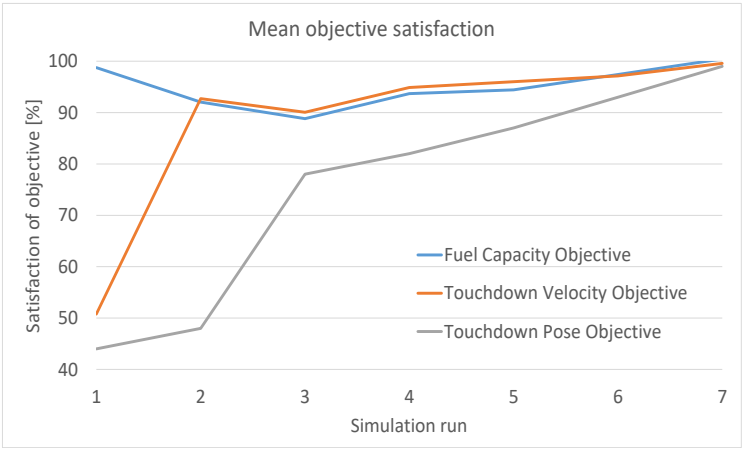
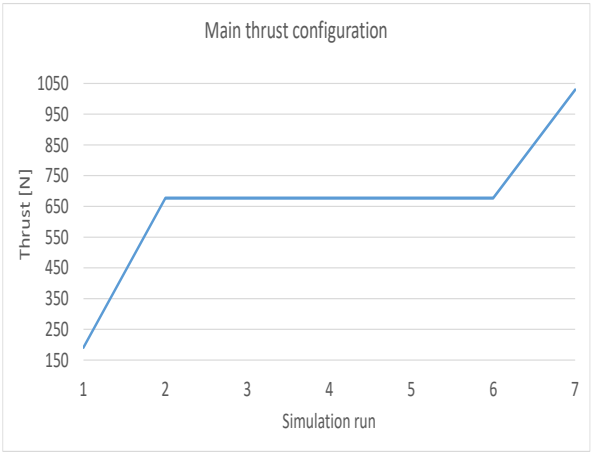
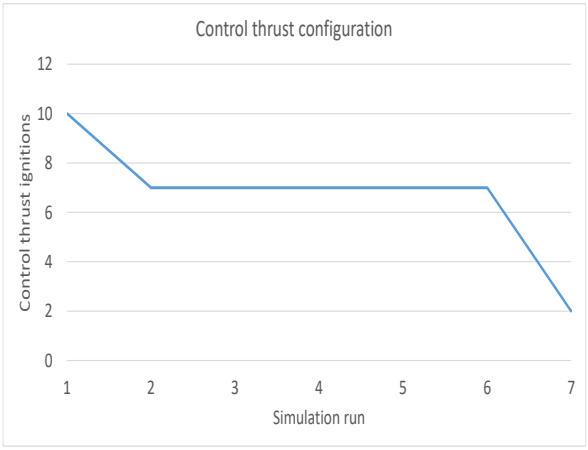


Figure 6.5: Objective satisfaction progress for all three optimized parameters: my approach successfully changes the vehicle configuration in order to increase the simulation goal satisfaction.



6.6a: Main thrust level parameter optimization over time. My MAS gradually increases the thrust level until the velocity can be adequately regulated for landing.



6.6b: Attitude thrust ignition optimization over time. My MAS gradually decreases the amount of ignitions as the overall thrust level is increased by another agent.

This study concerns the adaptation capabilities of the proposed MAS architecture in general. The second study in the next section is investigating the performance of my MAS based on a provided FDS approximation.

Figures 6.6a, 6.6b and 6.7 show how the main thrust level, fuel capacity and control thrust ignition configuration change over time. My approach successfully optimizes these parameters until the spacecraft safely lands after seven completed simulation loops with overall 1500 simulation steps.

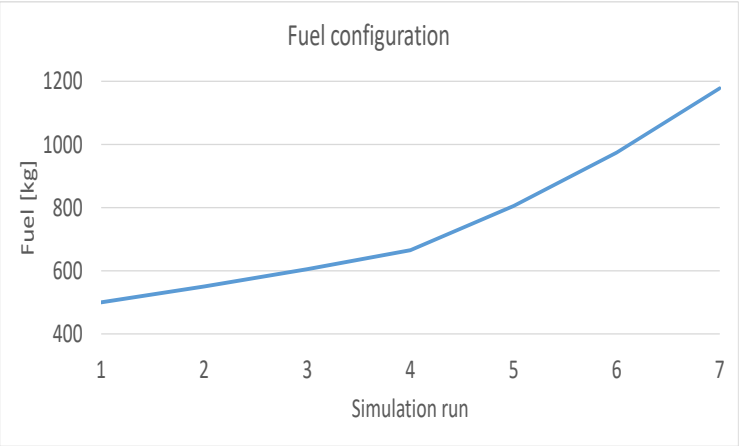


Figure 6.7: Fuel capacity parameter optimization over time. My MAS gradually increases the fuel capacity in order to maintain thrust while the landing procedure is conducted.

6.6.2 Multiobjective Optimization

In this section, I present the results of my MAS based optimization when using the provided FDS approximation from my KDP. For this evaluation, I considered the following standardized test function for MOPs from Binh and Korn [190] as a use case study. I further added noise terms to the functions in order to obtain a stochastic behavior:

$$\begin{aligned} f_1(x,y) &= 4x^2 + 4y^2 + N \\ f_2(x,y) &= (x-5)^2 + (y-5)^2 + N \\ &s.t. \\ g_1(x,y) &= (x-5)^2 + y^2 \leq 25 \\ g_2(x,y) &= (x-8)^2 + (y+3)^2 \geq 7.7 \end{aligned} \tag{6.4}$$

In this use case study, I approximate f_1, f_2 with my KDP approach and deliver the B-splines as input to my MAS. Two advantages from my approach can be observed in this use case study (see Figure 6.8):

First, the initial guess from the agents is directly the single objective solution of the problem, indicating that the approximated FDS is close to the Pareto front. This enables a much faster optimization process because my MAS requires less negotiations for converging to the correct solution. In addition, this reduces the probability of converging to a local instead of a global minimum. Second, already after a few negotiations (in this use case study: two negotiations) the objective-agents reached the Pareto front and returned one optimal configuration. This evaluation shows that my MAS directly benefits from the FDS approximation because, atleast in these evaluations, my MAS computes more efficiently and effectively a solution to the MOP

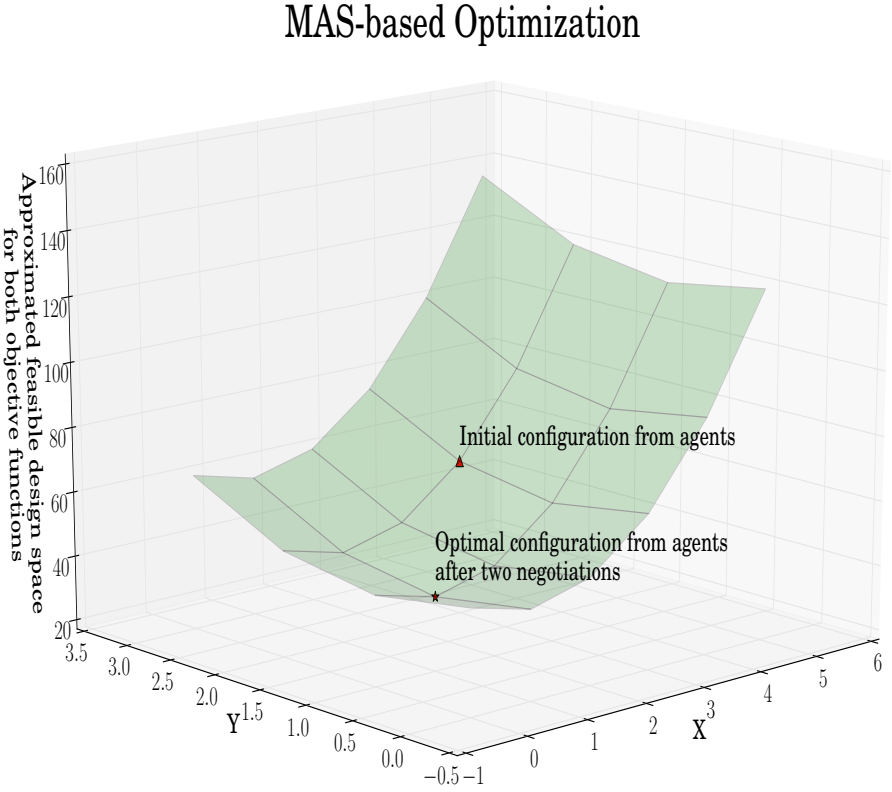


Figure 6.8: Evaluation of my use case study: My agents are directly initialized at the single-objective solution and converge fast to the multi-objective solution. Based on my FDS approximation, very simple surfaces can be generated which reduce complex optimization problems. On these surfaces even a simple gradient descent is often sufficient to solve the unknown functions.

Part IV

Crossroads

There is nothing either good or bad but thinking makes it so.

William Shakespeare

7 Epilogue

7.1 Summary

In this thesis, I made several contributions to the area of virtual testbeds. The proposed concepts overcome the integration and workflow challenge of state-of-the-art virtual testbeds. My contributions have been described in detail in the previous chapters. My concepts include my wait-free hash map that implements a MVCC. It allows for low-latency massively parallel data exchange. Even more, it reduces the number of required interfaces and does not need any standard locking mechanism. It is the baseline for my generative virtual testbed concept. This concept involves the ECS pattern and my novel DSML. My DSML enables a TBCG of arbitrary virtual testbeds and the integration of my wait-free hash maps into the ECS pattern lead to C&C. My evaluations have shown superior performance of my CCM in many synthetic and real benchmarks.

My concepts further include my novel data mining algorithms for an automatic KDP and a hierarchical MAS optimization toolset. Both constitute a novel concept for SBO and MOO in virtual testbeds for deterministic and stochastic blackbox simulations which are governed by a MOP. My KDP unveils hidden relationships between simulation input and simulation model behavior. It uses my novel GDS approach for approximating simulation model behaviors for deterministic and stochastic simulations. My GDS approach can accurately approximate the FDS of arbitrary simulation models while using less samples than its' competitors. The resulting FDS approximation can be used to formulate gradients towards the Pareto front for MOO purposes. My evaluations have shown that standard optimization solvers benefit from my FDS approximation, e.g. my hierarchical MAS. Furthermore, they have shown that my data mining algorithms are able to efficiently analyze large scale simulations with tens of thousands of parameters in less than a second. Even more, the resulting FDS approximations are close to the Pareto front.

7.2 Future Work

In the following, I present several ideas to further improve the approaches of my thesis, and some ideas for additional methods that should be integrated in order to increase the overall performance. Generally, it would beneficial if all presented approaches could be validated and verified in more use case studies and evaluations. For instance, the presented KDP and the corresponding hierarchical MAS optimization could be evaluated in known synthetic problems of the SimOpt library [168] or in real world scenarios. Likewise, the proposed software infrastructure could be integrated in other existing RISs in order to evaluate its' performance, the memory allocation and de-allocation strategies. In the following, several additional technical advancements are outlined for the presented ideas of this thesis.

7.2.1 Wait-Free Data and Concurrency Management for Massively Parallel Realtime Interactive Systems

There are avenues for future work for the proposed software infrastructure approach based on wait-free hash maps.

The synthetic benchmarks of my wait-free hash map concept were restricted to a single machine, but Iam confident that my approach scales well for distributed applications like CVEs. In such a distributed CVE application, my approach would serve as a central time synchronising host, which updates the timestamps of the `KVPairs`. Typical distributed CVE problems such as network delays would not affect read or write operations. Delayed write operations would be merged and delayed read operations would still retrieve the newest data from the `KVPool`. In addition to that, a concurrent quality measure for data could define the boundaries and constraints for my proposed wait-free concept. Furthermore, I would like to apply current approaches based on ECS (e.g. [55, 56, 119–121]) that strongly strengthen the maintainability and re-usability of the systems by applying semantics.

With respect to the GraphPool approach, I would like to extend the `GraphCache` technique. It would be desirable to remove the initial setup phase of the `GraphCache` while maintaining its' wait-free behavior. Probably, this could be done by using unique prime numbers as identifiers for the `Systems` using the `GraphCache`. These identifiers could be used for unique cache access determination, which would result in "private" `GraphCaches` for every `System` accessing the `GraphPool`.

With respect to the presented application of the ECS pattern, I would like to extend the approach with intelligent `Component` cloning. This would incorporate that the write operation only clones the `Component` data when needed and not by default. This could be implemented with a `System`-based heuristic which notifies when the next read operation may happen. Furthermore, I would like to apply my wait-free hash maps in more RIS applications in order to enable a profound analysis of their capabilities.

Another interesting idea is based upon my DSML and TBCG approach. An important factor for RISs is latency. It determines the performance of the complete system and also, in many cases, the interaction quality. Especially the quality of interaction is for many RISs, e.g. VR systems, extremely important as high latencies make immersive interaction impossible. In this context, it would be a significant improvement of my system, if it could estimate the `Systems'` latency and their access to `Entities` already during the modeling and generation of the system. In order to accomplish this, the latencies, based on empirical values, are automatically specified during the modeling, based on estimations from the empirical data. This approach is similarly to [185] and would extend the presented ideas for my proposed wait-free concepts.

These experience-based latencies can be gained from existing RISs, which are based on my software infrastructure. Even more, analyzing the (wait-free) parallelism (e.g. similarly to [184]) of the system based on the modelled dataflow would also be a very interesting approach for efficiently modelling RISs.

Finally, I would also like to extend my approach with high-level concepts for adaptive memory management of my wait-free hash maps. It would be beneficial if a RIS framework itself could determine which memory management suits the current *System* and `Component` composition best. This would incorporate that the RIS framework monitors the usage of all `Components` from every `System` and adapts the memory management accordingly.

7.2.2 Knowledge Discovery Processes for Blackbox Simulations

In the future, I would like to further evaluate my KDP (and the corresponding MAS based optimization) with standard SBO problems using the SimOpt library [168]. Additionally, I would like to extend my KDP with several interpolation approaches: instead of using only my B-spline surface, I could analyze the unknown objective function with several approximations (linear, polynomial, spline) in parallel. After successfully analyzing the unknown objective function, the best (in terms of accuracy) approximation is used. This would lead to specific approximation types per simulation objective which could further minimize the approximation error. The motivation of this approach is the variability of the unknown objective functions. I believe that several approximations at once (partially or completely covering the objective function) could also be used for dealing with chaotic objective functions which I did not consider yet. Another interesting idea would be to replace my B-spline surface concept with a high dimensional input space. Here, I would need to extend my B-spline surface concept to B-spline volumes. These B-spline volumes could be directly used for high-dimensional optimization. Additionally, I would like to incorporate general-purpose computing on graphics processing units (GPGPU) programming into my data mining approach. I believe that such massively parallel implementation can be efficiently used to analyze large-scale simulations. Another interesting approach are deep neural networks which can be used to conduct a multi-class regression of the simulation model behavior. Usually, deep neural networks are not feasible in the previously described use cases of virtual testbeds due to the large configuration space. However, these neural networks would use my FDS approximations in order to generate adequate training data. My FDS approximations are really useful here because they already approximate the relations very well. I believe that such a multi-class regression can be used to approximate the complete Pareto front and not just a single solution, e.g. based on [175]. This approach would have the advantage that my proposed dimensionality reduction would not be required in the optimization; the input of the neural network (after training) could be the whole dependent simulation input space. The output would be an approximation of the whole simulation model behavior. Such deep neural networks can then be used by standard optimization toolsets to compute Pareto optimal solutions.

7.2.3 Application to Robotics and Evaluating the Reality Gap

Another interesting avenue for future work is the application of my work in reality. My KDP approach can be transfered into the real world, namely in robotics, in two different approaches: 1) as a valuable information for autonomous decision-making algorithms and 2) to determine and minimize the reality gap [182].

In detail, my FDS approximation can be incorporated into the knowledge base of real robots. The autonomous decision-making algorithm that controls the robot could thus decide also on my approximated model behavior. Here, the decision-making algorithm could make simple queries based on my FDS approximation. For instance, how often the robot was able, in its current configuration, to successfully solve the upcoming task in the simulation.

My FDS approximation can predict here the probability and standard deviation of the success for such queries based on simulation data. This could greatly benefit current autonomous decision-making algorithms because they would incorporate the complete knowledge of previously conducted simulations.

My second idea is based on above concept and aims at determining the simulation-to-reality gap and, if possible, to minimize it. The simulation-to-reality gap [97] describes the discrepancy between a simulation and its results to the actual model behavior in reality. No matter how sophisticated a simulation is, there will always be a difference to reality because simulations can never completely depict reality. However, in the scenario above, a robot with the knowledge of my FDS approximation could evaluate to what extent the predictions of the approximation are correct by testing and comparing the corresponding actions and their results. Deep-learning approaches, such as neural networks, can efficiently used for this task. Here, neural networks would learn the unknown differences in the simulation model configuration, which are necessary to correct the FDS approximations to the real world. This would result in a inverse sensitivity analysis of my results. This analysis can be used to evaluate the validity of my FDS approximation and the corresponding simulation results with respect to reality.

Furthermore, it is also possible to adapt simulations based on an inverse approach of my FDS approximation. Usually, the simulation-to-reality gap makes it hard to transfer learnt behaviors of robots in simulations to reality [97]. Most often, the simulation is tediously tuned to match real-world data more closer so that learnt behaviors can be directly transferred to real robots. In this scenario, the machine learning engineer has to know which parameters to tune, in order to adapt the simulation correctly. Here, my approach can be used to determine the parameters to be tuned - again as a inverse implementation. This would greatly improve the workflow of machine learning engineers to quickly adapt their simulation (results) to their desired transferred use cases.

Part V

Appendix

Publications and Awards

Some parts of this work have appeared previously in the following publications:

Patrick Lange, René Weller, Gabriel Zachmann. **GDS: Gradient based Density Spline Surfaces for Multiobjective Optimization in Arbitrary Simulations**. ACM SIGSIM PADS. Sinapore, Republic of Singapore, 2017.

Patrick Lange, René Weller, Gabriel Zachmann. **Intelligent Realtime 3D Simulations**. ACM SIGSIM PADS Ph.D. Colloqium. Banff, Canada, 2016.

Patrick Lange, René Weller, Gabriel Zachmann. **Knowledge Discovery for Pareto based Multiobjective Optimization in Simulation**. ACM SIGSIM PADS. Banff, Canada, 2016.

Patrick Lange, René Weller, Gabriel Zachmann. **GraphPool: A High Performance Data Management for 3D Simulations**. ACM SIGSIM PADS. Banff, Canada, 2016.

Patrick Lange, René Weller, Gabriel Zachmann. **Wait-Free Hash Maps in the Entity-Component-System Pattern for Realtime Interactive Systems**. IEEE VR: 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems SEARIS. Greenville, United States of America, 2016.

Alena Probst, Graciela Gonzales Peytavi, David Nakath, Anne Schattel, Carsten Rachuy, Patrick Lange, Joachimg Clemens, Mitja Echim, Verena Schwarting, Abhishek Srinivas, Konrad Gadzicki, Roger Förster, Bernd Eissfeller, Kerstin Schill, Christof Büskens, Gabriel Zachmann. **Kanaria: Identifying the Challenges for Cognitive Autonomous Navigation and Guidance for Missions to Small Planetary Bodies**. International Astronautical Congress (IAC). Jerusalem, Israel, 2015 .

Patrick Lange, René Weller, Gabriel Zachmann. **Multi Agent System Optimization in Virtual Vehicle Testbeds**. EAI SIMUtools 2015. Athens, Greece, 2015.

Patrick Lange, René Weller, Gabriel Zachmann. **Scalable Concurrency Control for Massively Collaborative Virtual Environments**. ACM Multimedia Systems, Massively Multiuser Virtual Environments (MMVE) 2015. Portland, United States of America, 2015.

Patrick Lange, Alena Probst, Abhishek Srinivas, Graciela González Peytavi, Carsten Rachuy, Anne Schattel, Verena Schwarting, Joachim Clemens, David Nakath, Mitja Echim, and Gabriel Zachmann. **Virtual Reality for Simulating Autonomous Deep-Space Navigation and Mining**. 24th International Conference on Artificial Reality and Telexistence (ICAT-EGVE 2014). Bremen, Germany, 2014.

Patrick Lange, René Weller, Gabriel Zachmann. **A Framework for Wait-Free Data Exchange in Massively Threaded VR Systems**. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)). Plzen, Czech Republic, 2014.

I was awarded with the *Ph.D. Colloquium Award* at ACM SIGSIM PADS 2016 for my Ph.D. research.

Glossary

3DST 3D simulation technology. 4, 14, 19, 35, 37–43, 52, 65

ACID atomicity, consistency, isolation, durability. 20, 22, 38, 44

ARM association rule mining. 97, 99, 100, 109, 117

C&C increased cohesion and decreased coupling. 12, 15, 45, 49, 68, 137

CAS compare and swap. 40, 41, 61

CCM concurrency control management. 14, 19–23, 37–41, 44, 46, 49, 52, 58, 61, 68, 69, 81, 137

COW copy on write. 44, 46, 61, 63

CVE collaborative virtual environment. 14, 19, 35–40, 68, 69, 138

DBMS database management system. 19–23, 31, 32, 52

DBScan density based spatial clustering of applications with noise. 104–106, 119

DES discrete event simulation. 79, 80

DSML domain specific modelling language. 13, 15, 17, 67–70, 77–82, 84, 85, 137, 138

ECS entity component system pattern. 13, 15–17, 44, 45, 52, 59, 67–75, 77, 79, 81, 85–87, 137, 138

FDS feasible design space. 9–11, 13, 16, 17, 92, 94, 96–98, 101, 112–114, 125–130, 132–134, 137, 139, 140

GAM global atomic marker. 45, 46, 48–50, 52, 61–64, 75

GDS gradient based density spline surface. 94, 95, 111, 119–121, 127, 129, 137

GOPRR graphs, objects, properties, relationships, roles. 80, 81

GPGPU general-purpose computing on graphics processing units. 139

GraphCache graph based key value pool cache. 51, 57–59

GraphNode graph based key value pool node. 52–55, 57, 61, 65

GraphPool graph based key value pool. 51–55, 57–59, 61, 65–68, 70, 72, 77–79, 82, 85, 86, 127, 138

GSS global simulation state. 45, 52, 53, 55, 80

KDD knowledge discovery in databases. 19, 30, 93, 95, 97

KDP knowledge discovery process. 13, 16, 17, 19, 30–32, 77, 78, 80, 91–100, 102, 110, 113–117, 133, 137–139

KeyValuePair key value pair. 44, 45, 48, 61–63, 79–82, 84

KeyValuePool key value pool. 44–46, 48–53, 55, 59–64, 67, 68, 70, 72, 77–79, 82, 85, 86, 127

LAM local atomic marker. 45, 48–52, 55, 63, 64, 75, 76

LSS local simulation state. 45, 52, 55

MAS multi agent system. 13, 16, 17, 69, 77, 78, 94, 95, 125–133, 137–139

MDE model driven engineering. 15, 68, 69

MOO multiobjective optimization. 6, 9, 13, 16, 17, 80, 91, 93–97, 110, 112, 126, 127, 130, 137

MOP multiobjective optimization problem. 9–11, 13, 16, 17, 19, 27, 28, 92–95, 97, 98, 100, 105, 110, 112, 113, 117, 125–128, 133, 137

MSO modeling, simulation and optimization. 3, 9

MVCC multi version concurrency control. 12, 14, 17, 22, 23, 38, 44–46, 49, 137

NoSQL not only SQL. 42, 43, 53

PGM property graph model. 54, 55, 57

PIM platform independent model. 13, 15, 77–79, 82

PSM platform specific model. 13, 15, 77–79, 82

RCA recursive correlation analysis. 102, 109, 110

RIS realtime interactive system. 4, 10–12, 14, 15, 17, 19, 35–41, 45, 49, 67–72, 75, 77, 86, 87, 138, 139

RLHE real life hardware environment. 3–5

SBO simulation based optimization. 3–6, 9–13, 16, 19, 24–26, 28, 29, 77–80, 84, 91, 92, 94–97, 137, 139

SOO singleobjective optimization. 17, 95, 126, 127

TBCG template based code generation. 13, 82, 137, 138

VE virtual environment. 14, 19, 35–38, 40

VR virtual reality. 4, 5, 10, 14, 19, 35–40, 45, 68, 69, 138

Bibliography

[1] . Pitkaenen, T. Mikkonen. Lightweight domain-specific modeling and model-driven development. *6th OOPSLA Workshop on Domain-Specific Modeling*, pages 159–168, 2006.

[2] A. Abraham, L. C. Jain, R. Goldberg. Evolutionary Multiobjective Optimization: Theoretical Advances and Applications. *Springer Verlag*, 2005.

[3] A. Albers, L. Nowicki. Integration der Simulation in die Produktentwicklung. Symposium Simulation in der Produkt- und Prozessentwicklung, 2003.

[4] A. Boukerche, N. J. McGraw, R.B. Araujo. A Grid-Filtered Region-Based Approach to Support Synchronization in Large-Scale Distributed Interactive Virtual Environments. *International Conference on Parallel Processing Workshops*, pages 525–530, 2005.

[5] A. Braginsky, A. Kogan, E. Petrank. Drop the Anchor: Lightweight Memory Management for Non-Blocking Data Structures. *In Proceedings of the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 33–42, 2013.

[6] A. Byrski, M. Kisiel-Dorohinicki. Decentralized Multi-Agent Optimization via Dual Decomposition. *International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, 2011.

[7] A. G. Kleppe, J. Warmer, W. Bast. MDA Explained: The Model Driven Architecture: Practice and Promise. *Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA*, 2003.

[8] A. Gidenstam, M. Papatriantafilou, H. Sundell, P. Tsigas. Efficient and Reliable Lock-Free Memory Reclamation Based on Reference Counting. *IEEE Transactions on Parallel and Distributed Systems*, 20, 2009.

[9] A, H. Ng, C. Dudas, L. Pehrsson, K. Deb. Knowledge Discovery in Production Simulation By Interleaving Multi-Objective Optimization and Data Mining. *The 5th International Swedish Production Symposium*, pages 461–471, 2012.

[10] A. Hasan, M. Vuolle, K. Siren. Minimisation of Life Cycle Cost of a Detached House using Combined Simulation and Optimisation. *Building and Environment*, 43:2022–2034, 2008.

[11] A. J. Lotka. Elements of Physical Biology. *Williams and Wilkins*, 1925.

[12] A. Lattner, J. Dallmeyer, I. Timm. Learning Dynamic Adaptation Strategies in Agent-Based Traffic Simulation Experiments. *Ninth German Conference on Multi-Agent System Technologies (MATES)*, pages 77–88, 2011.

[13] A. P. Dempster, N.M. Laird, D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

[14] A. Probst, G. Gonzales Peytavi, D. Nakath, A. Schattel, C. Rachuy, P. Lange, A. Srinivas, K. Gadzicki, K. Schill, C. Büskens, G. Zachmann. Kanaria: Identifying the Challenges for Cognitive Autonomous Navigation and Guidance for Missions to Small Planetary Bodies. *International Astronautical Congress (IAC)*, 2015.

[15] A. T. Nguyen. Sustainable Housing in Vietnam: Climate Responsive Design Strategies to Optimize Thermal Comfort. *Ph.D. thesis*, 2013.

[16] A.-T. Nguyen, S. Reiter, P. Rigo. A Review on Simulation-based Optimization Methods Applied to Building Performance Analysis. *Applied Energy*, 113:1043–1058, 2013.

[17] A. Tolk. Simulation and Modelling as the Essence of Computational Science. Summer Computer Simulation Conference, 2018.

[18] A. Vakaloudis, B. Theodoulidis. Spatiotemporal Database Connection to VRML. *Proceedings 9th UL Electronic Imaging & Visual Arts Conference*, 1998.

[19] A. Valadares, T. Debeauvais, C. V. Lopes. Evolution of Scalability with Synchronized State in Virtual Environments. *International Workshop on Haptic Audio Visual Environments and Games*, pages 142–147, 2012.

[20] A. Zook, S. Lee-Urabn, M. O. Riedl, H. K. Holden, R. A. Sottilare, K. W. Brawner. Automated Scenario Generation: Toward Tailored and Optimized Military Training in Virtual Environments. *Conference on the Foundations of Digital Games*, 2012.

[21] ArangoDB. ArangoDB's design objectives. 2012. URL `https://www.arangodb.com/2012/03/avocadodbs-design-objectives/`.

[22] AUDI MediaCenter. Audi tests gesture control for virtual assembly. 2015. URL `https://www.audi-mediacenter.com/en/press-releases/audi-tests-gesture-control-for-virtual-assembly-4904`.

[23] B. Damer, S. Gold, D. Rasmussen, et al. Data-Driven Virtual Environment Assembly and Operation. *NASA Ames Research Center: VIB Workshop Report*, 2004.

[24] B. Eisenhower, Z. O'Neill, S. Narayanan, V. A. Fonoberov, I. Mezic. A Methodology for Meta-Model based Optimization in Building Energy Models. *Energy and Buildings*, 47:292–301, 2012.

[25] B. Wilson, D. Cappelleri, T. W. Simpson, M. Frecker. Efficient Pareto frontier exploration using surrogate approximations. *Optimization and Engineering*, pages 31–50, 2001.

[26] Berkeley DB. Transactional Data Store Applications: Degrees of isolation. 2011. URL `https://docs.oracle.com/cd/E17275_01/html/programmer_reference/transapp_read.html`.

[27] C. Apte, S. J. Hong. Predicting Equity Returns from Securities Data with Minimal Rule Generation. *Advances in Knowledge Discovery and Data Mining*, 514–560, 1996.

[28] C. Carlsson, O. Hagsand. DIVE - A Multi-User Virtual Reality System. *Virtual Reality Annual International Symposium*, pages 394–400, 1993.

[29] C. de Negueruela, M. Scagliola, D. Giudici, J. Moreno, J. Vicent, A. Camps, H. Park, P. Flamant, R. Franco. ARCHEO-E2E: A Reference Architecture for Earth Observation end-to-end Mission Performance Simulators. *Simulation and EGSE facilities for Space Programmes*, ESA ESTEC, 2012.

[30] C. Dudas, A. H. C. Ng, H. Bostroem. Post-Analysis of Multi-Objective Optimization Solutions Using Decision Trees. *Intelligent Data Analysis*, 19:259–278, 2015.

[31] C. Fleury, T. Duval, V. Gouranton, B. Arnaldi. Architectures and Mechanisms to Maintain efficiently Consistency in Collaborative Virtual Environments. *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 2010.

[32] C. H. Papadimitriou, P. C. Kanellakis. On Concurrency Control by Multiple Versions. *ACM Transactions on Database Systems (TODS)*, 9(1):89–99, 1984.

[33] C. Henthorne, E. Tilevich. Code Generation on Steroids: Enhancing COTS Code Generators via Generative Aspects. *Proceedings of the Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques*, pages 8–14, 2007.

[34] C. Park, S. Park, S. H. Son. Multiversion Locking Protocol with Freezing for Secure Real-time Database Systems. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1141–1154, 2002.

[35] C++ Standards Committee Library Working Group. Boost Library. URL `http://www.boost.org/`.

[36] C. Watanabe, Y. Masunaga. Design and Implementation of a Multi-modal User Itnerface of the Virtual World Database System (VWDB). *7th International conference on Database Systems for Advanced Applications (DASFAA)*, pages 294 – 301, 2001.

[37] C. Watanabe, Y. Masunaga. VWDB2: A Network Virtual Reality System with a Database Function for a Shared Work Environment. *ISDB*, Acta Press:190 – 196, 2002.

[38] Couchbase. Why NoSQL? *Whitepaper*, 2014. URL `https://www.couchbase.com/binaries/content/assets/website/docs/whitepapers/why-nosql.pdf`.

[39] D. A. van Veldhuizen, G. B. Lamont. Evolutionary Computation and Convergence to a Pareto Front. *Late Breaking Papers at the Genetic Programming Conference*, 1998.

[40] D. Agrawahl, S. Sengupta. Modular Synchronization in Distributed, Multiversion Databases: Version Control and Concurrency Control. *IEEE Transactions on Knowledge and Data Engineering*, 5(1): 126–137, 1993.

[41] D. Arthur, S. Vassilvitskii. k-means++: The Advantages of Careful Seeding. *SODA '07 at 8th ACM-SIAM Symposium on Discrete algorithms*, 1027-1035, 2007.

[42] D. Corkill. Collaborating Software: Blackboard and Multi-Agent Systems and the Future. International Lisp Conference, 2003.

[43] D. Izzo. PYGMO and PYKEP: Open Source Tools for Massively Parallel Optimization in Astrodynamics. *International Conference on Astrodynamics Tools and Techniques (ICATT)*, 2012.

[44] D. L. Detlefs, P. A. Martin, G. L. Steele. Lock-Free Reference Counting. *ACM Symposium on Principles of Distributed Computing*, pages 190–199, 2001.

[45] D. Lee, M. Lim, S. Han. ATLAS - A Scalable Network Framework for Distributed Virtual Environments. *Presence*, 16:125–156, 2007.

[46] D. Lomet, A. Fekete, R. Wang, P. Ward. Multi-Version Concurrency via Timestamp Range Conflict Management. *IEEE 28th International Conference on Data Engineering (ICDE)*, pages 714–725, 2012.

[47] D. Losch, J. Rossmann. Simulation-Based Analysis of Mechanized Wood Harvest Operations. *International Conference on Industrial Engineering and Applications (ICIEA)*, 2017.

[48] D. Nakath, C. Rachuy, J. Clemens, K. Schill. Optimal rotation sequencies for active perception. In *Proc. SPIE Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2016*. SPIE Press, 2016.

[49] D. Nam, C. Hoon Park. Multiobjective Simulated Annealing: A Comparative Study to Evolutionary Algorithms. *International Journal of Fuzzy Systems*, 2, 2000.

[50] D. Quelhadj, S. Petrovic. A Survey of Dynamic Scheduling in Manufacturing Systems. *Journal of Scheduling*, 12:417–431, 2008.

[51] D. Roberts, R. Wolff. Controlling Consistency within Collaborative Virtual Environments. *International Symposium on Distributed Simulation and Real-Time Applications*, pages 46–52, 2004.

[52] D. Schmalstieg, G. Schall, d. Wagner, I. Barakonyi, G. Reitmayr, J. Newman, F. Ledermann. Managing Complex Augmented Reality Models. *IEEE Computer Graphics and Applications*, 27:48–57, 2007.

[53] D. T. Davis. D. Brutzman. The Autonomous Unmanned Vehicle Workbench: Mission Planning, Mission Rehearsal, and Mission Replay Tool for Physics-Based X3D Visualization. *Symposium on Unmanned Untethered Submersible Technology*, 2005.

[54] D. Tuhus-Dubrow, M. Krati. Genetic Algorithm based Approach to Optimize Building Envelope Design for Residential Buildings. *Building and Environment*, 45:1574–1581, 2010.

[55] D. Wiebusch, C. Zimmerer, M. E. Latoschik. Cherry-Picking RIS Functionality: Integration of Game and VR Engine Sub-Systems based on Entities and Events. In 10th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2017.

[56] D. Wiebusch, M. E. Latoschik. Decoupling the Entity-Component-System Pattern using Semantic Traits for Reusable Realtime Interactive Systems. *IEEE VR Workshop on Software Engineering and Architectures for Realtime Interactive Systems*, 2015.

[57] E. von Schweber. SQL3D - Escape from VRML Island. *SIGGRAPH VRML Consortium*, 1998.

[58] E. Zitzler. Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. *Ph.D. thesis*, Swiss Federal Institute of Technology Zurich, 1999.

[59] E. Zitzler, L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3:257–271, 1999.

[60] F. Boithias, M. E. Mankibi, P. Michel. Genetic Algorithms based Optimization of Artificial Neural Network Architecture or Buildings' Indoor Discomfort and Energy Consumption Prediction. *Building Simulations*, 5:95–106, 2012.

[61] F. Doin, J.-S. Chun, R. Robinson. Virtual Testbed for Assessing Probe Vehicle Data in IntelliDrive Systems. *IEEE Transactions on Intelligent Transportation Systems*, 2011.

[62] F. Steinicke, T. Ropinski, K. Hinrichs. A Generic Virtual Reality Software System's Architecture and Application. *ICAT Proceedings International Conference on Augmented Tele-Existence*, pages 220–227, 2005.

[63] F. W. Li, R. W. Lau, F. F. Ng. VSculpt: A Distributed Virtual Sculpting Environment for Collaborative Design. *IEEE Transaction on Multimedia*, 5:570–580, 2003.

[64] G. Burd. NoSQL. *Whitepaper*, pages 5–12, 2011. URL `https://www.usenix.org/legacy/publications/login/2011-10/openpdfs/Burd.pdf`.

[65] G. E. Horne, T. E. Meyer. Data Farming: Discovering Surprise. *Winter Simulation Conference*, pages 1082–1087, 2005.

[66] G. Lausen. Formal Aspects of Optimistic Concurrency Control in a Multiple Version Database System. *Information Systems*, 8(4):291–301, 1983.

[67] G. N. Buckley, A. Silberschatz. Obtaining Progressive Protocols for a Simple Multiversion Database Model. *VLDB*, pages 74–80, 1983.

[68] G. van Mare, R. Germs, F. Jansen. Integrating 3D-GIS and Virtual Reality. Design and Implementation of the Karma VI System. *10th Colloquium of the Spatial Information Research Center*, 1998.

[69] H. Terelius, U. Topcu, R. M Murray. Decentralized Multi-Agent Optimization via Dual Decomposition. *IFAC World Congress*, 2011.

[70] IBM. DB2 Knowledge Center Parallel Interaction. 2017. URL `https://www.ibm.com/support/knowledgecenter/SS9RXT_9.5.0/`.

[71] InnoDB. Multi-Versioning. 2017. URL `https://dev.mysql.com/doc/refman/5.7/en/innodb-multi-versioning.html`.

[72] J.-A. Desideri. Multi-Gradient Descent Algorithm (MGDA) for Multiobjective Optimization. *Comptes Rendus Mathematique*, 350:313–318, 2012.

[73] J. Balaram, J. Cameron, A. Jain, H. Kline, C. Lim, H. Mazhar, S. Myint, H. Nayar, R. Patton, M. Pomerantz, M. Quadrelli, P. Shakkotai, K. Tso. Physics-Based Simulator for NEO Exploration Anaysis and Modeling. *AIAA Space Conference and Exposition*, 2011.

[74] J. Balaram, R. Austin, P. Banarjee, T. Bentley, D. Henriquez, B. Martin, E. McMahon, G. Sohl. DSENDS - A High Fidelity Dynamics and Spacecraft Simulator for Entry, Descent and Surface Landing. *IEEE Aerospace Conference, Big Sky*, 2002.

[75] J. Clemens, T. Reineking, T. Kluth. An Evidential Approach to SLAM, Path Planning, and Active Exploration. *International Journal of Approximate Reasoning*, 2016.

[76] J. D. Siirola, S. Hauan, A. W. Westerberg. Computing Pareto Front Using Distributed Agents. *Computers and Chemical Engineering*, 29:113–126, 2004.

[77] J. Fliege, B. Fux Svaiter. Steepest Descent Methods for Multicriteria Optimization. *Mathematical Methods of Operations Research*, 3:479–494, 2000.

[78] J. Gray. The Transaction Concept: Virtues and Limitations. *7th International Conference on Very Large Data Bases (VLDB)*, 7:144–154, 1981.

[79] J.-H. Ryu, S. Kim, H. Wan. Pareto Front Approximation With Adaptive Weighted Sum Method in Multiobjective Simulation Optimization. *Winter Simulation Conference*, pages 623–633, 2009.

[80] J. Haist, V. Coors. The W3DS-Interface of Cityserver3D. *European Spatial Data Research: Next Generation 3D City Models*, pages 63–67, 2005.

[81] J. Kennedy, R. Eberhart. Particle Swarm Optimization. *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[82] J. Nocedal, S. J. Wright. Numerical Optimization. *Springer Verlag*, 1999.

[83] J. P. C. Kleijnen, S. M. Sanchez, T. M. Cioppa. A User's Guide to the Brave New World of Designing Simulation Experiments. *INFORMS Journal on Computing (Summer 2005)*, 17:263–289, 2005.

[84] S. Kelly J.-P. Tolvanen, R. Pohjonen. Advanced Tooling for Domain-Specific Modeling: MetaEdit+. *Sprinkle, J., Gray, J., Rossi, M., Tolvanen, J.P. (eds.) The 7th OOPSLA Workshop on Domain-Specific Modeling*, 2007.

[85] J. Paul, A. Detmann, J. Hilljegerdes, F. Kirchner, I. Ahrns, J. Sommer. INVERITAS: A Facility for Hardware-in-the-Loop Long Distance Movement Simulation for Rendezvous and Capture of Satellites and Other Autonomous Objects. *Acta Astronautica*, 116:1–24, 2015.

[86] J. Rohlf, J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. ACM SIGGRAPH, 1994.

[87] J. Rossmann, B. Sommer. The Virtual Testbed: Latest Virtual Reality Technologies for Space Robotic Applications. *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, pages 133–138, 2008.

[88] J. Rossmann, B. Sondermann, M. Emde. Virtual Testbeds for Planetary Exploration: The Self-Localization Aspect. *Symposium on Advanced Space Technologies in Robotics and Automation, ASTRA*, pages 1–8, 2011.

[89] J. Rossmann, M. Schluse, C. Schlette. The Virtual Forest: Robotics and Simulation Technology as the Basis for New Approaches to the Biological and the Technical Production in the Forest. *International Journal of Systemics, Cybernetics, and Informatics (JSCI)*, 8, 2010.

[90] J. Rossmann, M. Schluse, M. Rast, L. Atorf. eRobotics: Combining Electronic Media and Simulation Technology to Develop (Not Only) Robotics Applications. hapter in E-Systems for the 21st Century: Concept, Developments, and Applications, 2016.

[91] J. Rossmann, M. Schluse, R. Waspe, M. Hoppen. Real-Time Capable Data Management Architecture for Database-Driven 3D Simulation Systems. *Database and Expert Systems Applications*, pages 262 – 269, 2011.

[92] J. Westra, F. Dignum, V. Dignum. Guiding User Adaptation in Serious Games. *Agents for Games and Simulations II*, pages 117–131, 2011.

[93] J. Westra, H. van Hasselt, F. Dignum. On-line Adapting Games using Agent Organizations. *IEEE Symposium on Computational Intelligence and Games*, 2008.

[94] J. Westra, H. van Hasselt, V. Dignum. Adaptive Serious Games Using Agent Organizations. *The 10th International Conference on Autonomous Agents and Multiagent Systems*, 3:1291–1292, 2011.

[95] J. Yang, D. Lee. Scalable Prediction Based Concurrency Control for Distributed Virtual Environments. *Virtual Reality*, pages 151–158, 2000.

[96] J. Zimmermann, C. Stark, J. Rieck. Projektplanung - Modelle, Methoden, Management. Springer Verlag, 2010.

[97] J.B. Mouret, S. Koos, S. Doncieux. Crossing the reality gap: a short introduction to the transferability approach. ALIFE Workshop Evolution in Physical Systems, 2012.

[98] JPL Robotics. Virtual Mars Testbed. . URL `https://www-robotics.jpl.nasa.gov/tasks/showTask.cfm?FuseAction=ShowTask&TaskID=265&tdaID=700069`.

[99] JPL Robotics. Mobility Testbed. . URL `https://www-robotics.jpl.nasa.gov/tasks/showTask.cfm?FuseAction=showTask&TaskID=270&tdaID=700074`.

[100] K. Kaku, H. Minami, T. Tomii, H. Nasu. Proposal of Virtual Space Browser Enables Retrieval and Action with Semantics which is Shared by Multi Users. *21st International Conference on Data Engineering Workshops (ICDEW)*, pages 1259–1259, 2005.

[101] K. Pearson. Mathematical Contributions to the Theory of Evolution. *Philosophical Transactions of the Royal Society*, 187:253–318, 1896.

[102] K. Socha, M. Kisiel-Dorohinicki. Agent-based Evolutionary Multiobjective Optimisation. *IEEE Evolutionary Computation*, 2002.

[103] K. Socha, M. Kisiel-Dorohinicki. Agent-based Evolutionary Multiobjective Optimisation. *Evolutionary Computation*, 2002.

[104] K. Sugimura, S. Obayashi, S. Jeong. Multi-Objective Design Exploration of a Centrifugal Impeller Accompanied With a Vaned Diffuser. *ASME/JSME Joint Fluids Engineering Conference*, 2007.

[105] L. Atorf, C. Schorn, C. Schlette, J. Rossmann. A Framework for Simulation-based Optimization Demonstrated on Reconfigurable Robot Workcells. IEEE International Systems Engineering Symposium (ISSE), 2017.

[106] L. Atorf, M. Schluse, J. Rossmann. Simulation-based Optimization, Reasoning and Control: The eRobotics Approach Towards Intelligent Robots. 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), 2014.

[107] L. Jeff Hong, Barry L. Nelson. A Brief Introduction to Optimization via Simulation. *Winter Simulation Conference*, 2009.

[108] C. Bernon L. Pons. A Multi-Agent System for Autonomous Control of Game Parameters. *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2013.

[109] L. Siwik, M. Kisiel-Dorohinicki. Semi-elitist Evolutionary Multi-agent System for Multiobjective Optimization. *Computational Science - Lecture Notes in Computer Science*, 3993, 2006.

[110] L. Siwik, P. Sikorski. Efficient Constrained Evolutionary Multi-Agent System for Multi-objective Optimization. *IEEE Evolutionary Computation*, 2008.

[111] L. Xu, J. Neufeld, B. Larson, D. Schuurmans. Maximum Margin Clustering. *Advances in Neural Information Processing Systems*, 17:1537–1544, 2005.

[112] M. Barbatum G. Bruno, A. Genovese. Applications of Agent-based Models for Optimization Problems: A Literature Review. *Expert Systems with Applications*, 39:6020–6028, 2012.

[113] M. C. Burl, D. DeCoste, B. L. Enke, D. Mazzoni, W. J. Merline, L. Scharenbroich. Automated Knowledge Discovery from Simulators. *6th SIAM International International Conference on Data Mining*, pages 82–93, 2006.

[114] M. C. Fu. Optimization via Simulation: A Review. *Annals of Operations Research*, 53:199–248, 1994.

[115] M. Caramaia, P. Dell'Olmo. Multi-objective Management in Freight Logistics. *Springer Verlag*, 2008.

[116] M. Cohrs, S. Klimke, G. Zachmann. Streamlining Function-oriented Development by Consistent Integration of Automotive Function Architectures with CAD Models. *Computer-Aided Design and Applications*, 11:4, 2014.

[117] M. E. Latoschik, H. Tramberend. Simulator X: A Scalable and Concurrent Software Platform for Intelligent Realtime Interactive Systems. *Proceedings of the IEEE VR*, 2011.

[118] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *AAAI Press*, pages 226–231, 1996.

[119] M. Fischbach. Enhancing Software Quality of Multimodal Interactive Systems. *PhD Thesis*, 2017.

[120] M. Fischbach, D. Wiebusch, M. E. Latoschik. Semantics-based Software Techniques for Maintainable Multimodal Input Processing in Real-time Interactive Systems. In 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2016.

[121] M. Fischbach, D. Wiebusch, M. E. Latoschik. Semantic Entity-Component State Management Techniques to Enhance Software Quality for Multimodal VR-Systems. IEEE Transactions on Visualization and Computer Graphics, Vol. 23 (4), 2017.

[122] M. Herlihy. Wait-free Synchronization. *ACM Transactions on Programming Languages and Systems*, pages 206–209, 1991.

[123] M. Herlihy, V. Luchangco, M. Moir. Nonblocking memory management support for dynamic-sized data structures. *ACM Transactions on Computer Systems*, 23, 2005.

[124] M. Hoppen, J. Rossmann. A Database Synchronization Approach for 3D Simulation Systems. 6th International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2014), 2014.

[125] M. Hoppen, J. Rossmann. A Novel Distributed Database Synchronization Approach with an Application to 3D Simulation. IARIA International Journal on Advances in Software, 2014.

[126] M. Hoppen, M. Schluse, J. Rossmann, B. Weitzig. Database-Driven Distributed 3D Simulation. *Proceedings of the Winter Simulation Conference*, 2012.

[127] M. Hoppen, R. Waspe, M. Rast, J. Rossmann. Distributed Information Processing and Rendering for 3D Simulation Applications. International Journal of Computer Theory and Engineering (IJCTE), 2014.

[128] M. I. Pomerantz, A. Jain, S. Myint. Dspace: Real-time 3D Visualization System for Spacecraft Dynamics Simulation. *3rd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2009.

[129] M. J. Carey, W. A. Muhanna. The Performance of Multiversion Concurrency Control Algorithms. *ACM Transactions on Computer Systems (TOCS)*, 4(5):338–378, 1986.

[130] M. Liebscher, K. Witowski, T. Goel. Decision Making in Multi-Objective Optimization for Industrial Applications - Data Mining and Visualization of Pareto Data. *8th World Congress on Structural and Multidisciplinary Optimization*, 2009.

[131] M. M. Michael. Hazard Pointers: Safe Memory Reclamation for Lock-Free Objects. *IEEE Transactions on Parallel and Distributed Systems*, 16, 2004.

[132] M. Painter, M. Erraguntla, G. Hogg, B. Beachkofski. Using Simulation, Data Mining, And Knowledge Discovery Techniques For Optimized Aircraft Enginee Fleet Management. *Winter Simulation Conference*, pages 1253–1260, 2006.

[133] M. Rohde, J. Crawford, D. A. Horner. An Interactive Physics-based Unmanned Ground Vehicle Simulator leveraging Open Source Gaming Technology: Progress in the Development and Application of the Virtual Autonomous Navigation Environment (VANE) Desktop. *Unmanned Systems Technolgy XI SPIE*, 2009.

[134] M. Rohde, M. Toschlog. Toward the Fusion of Serious Simulation and Video Games. *Proceedngs of the 2009 Spring Simulation Multiconference*, 2009.

[135] Microsoft. SQL Server Features - Snapshot Isolation in SQL Server. 2017. URL `https://msdn.microsoft.com/en-us/library/tcbchxcb.aspx`.

[136] MMI Aachen. iBoss. . URL `https://www.mmi.rwth-aachen.de/projekt/iboss/`.

[137] MMI Aachen. SELOK. . URL `https://www.mmi.rwth-aachen.de/projekt/selok/`.

[138] MongoDB. Wired Tiger - Snapshots and Checkpoints. 2016. URL `https://docs.mongodb.com/manual/core/wiredtiger/`.

[139] N. Batra, K. Kapil. Concurrency Control Algorithms and its Variants: A Survey. AIP Conference Proceedings, 2010.

[140] N. F. Polys, S. S. Visamsetty, P. Bhattacharjee, E. Tilevich. The Value of Patterns in Deep Media Scenegraphs. *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 2009.

[141] N. Feldkamp, S. Bergmann, S. Strassburger. Knowledge Discovery in Manufacturing Simulations. *ACM SIGSIM PADS*, pages 3–12, 2015.

[142] O. Wongwirat, S. Ohara. Performance Evaluation of Compromised Synchronization Control Mechanism for Distributed Virtual Environment. *Virtual Reality*, 9:1–16, 2006.

[143] P. A. Bernstein, N. Goodman. Multiversion Concurrency Control - Theory and Algorithms. *ACM Transactions on Database Systems (TODS)*, pages 465–483, 1983.

[144] P. Bernstein, E. Newcomer. Principles of Transaction Processing: For the Systems Professional. *Morgan Kaufmann Publishers*, 1997.

[145] P. Buttolo, R. Oboe, B. Hannaford. Architectures For Shared Haptic Virtual Environments. *Computers & Graphics: Haptic Displays in Virtual Environments and Computer Graphics in Korea*, 21:421–429, 1997.

[146] P. Cheeseman, J. Stutz. Bayesian Classification (AUTOCLASS): Theory and Results. *Advances in Knowledge Discovery and Data Mining*, 1996.

[147] P. Davidsson, L. Henesy, L. Ramstedt, J. Törnquist, F. Wernstedt. On the Integration of Agent-based and Mathematical Optimization Techniques. *Lecture Notes in Artificial Intelligence*, 4496:1–10, 2007.

[148] P. Durst, M. Rohde, J. Crawford. A Real-Time, Interactive Simulation Environment for Unmanned Ground Vehicles: The Autonomous Navigation Virtual Environment Laboratory (ANVEL). *International Conference on Information and Computing Science (ICIC)*, pages 7–10, 2012.

[149] P. Lange, A. Probst, A. Srinivas, G. Gonzalez Peytavi, C. Rachuy, A. Schattel, V. Schwarting, J. Clemens, D. Nakath, M. Echim, G. Zachmann. Virtual Reality for Simulating Autonomous Deep-Space Navigation and Mining. *24th International Conference on Artificial Reality and Telexistence (ICAT-EGVE 2014)*, 2014.

[150] P. Lange, R. Weller, G. Zachmann. A Framework for Wait-Free Data Exchange in Massively Threaded VR Systems. *Journal of WSCG 2014*, 22:383–390, 2014.

[151] P. Lange, R. Weller, G. Zachmann. Scalable Concurrency Control for Massively Collaborative Virtual Environments. *ACM Multimedia Systems, Massively Multiuser Virtual Environments (MMVE)*, 2015.

[152] P. Lange, R. Weller, G. Zachmann. Multi Agent System Optimization in Virtual Vehicle Testbeds. *EAI SIMUtools*, 2015.

[153] P. Lange, R. Weller, G. Zachmann. Knowledge Discovery for Pareto based Multiobjective Optimization in Simulation. *ACM SIGSIM PADS*, 2016.

[154] P. Lange, R. Weller, G. Zachmann. GraphPool: A High Performance Data Management for 3D Simulations. *ACM SIGSIM Conference on Principles of Advanved Discrete Simulations (PADS)*, 2016.

[155] P. Lange, R. Weller, G. Zachmann. Wait-Free Hash Maps in the Entity-Component-System Pattern for Realtime Interactive Systems. *IEEE VR 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 2016.

[156] P. Lange, R. Weller, G. Zachmann. GDS: Gradient Based Density Spline Surfaces for Multiobjective Optimization in Arbitrary Simulations. *ACM SIGSIM PADS*, 2017.

[157] P. M. Bober, M. J. Carey. Multiversion Query Locking. *University of Wisconsin-Madison*, 1992.

[158] P. M. Bober, M. J. Carey. On Mixing Queries and Transactions via Multiversion Locking. *IEEE 8th International Conference on Data Engineering*, 1992.

[159] P. M. Duvall, S. Matyas, A. Glover. Continuous Integration: Improving Software Quality and Reducing Risk. *Pearson Education, Boston*, 2007.

[160] P. Stellwag, A. Ditter, W. Schröder-Preikschat. A Wait-Free Queue for Multiple Enqueuers and Multiple Dequeuers Using Local Preferences and Pragmatic Extensions. *In Proceedings IEEE Symposium on Industrial Embedded Systems*, pages 237–248, 2009.

[161] PostgreSQL. Version 7.1.13: Multi-Version Concurrency Control. 2017. URL `https://www.postgresql.org/docs/7.1/static/mvcc.html`.

[162] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, I. Verkamo. Fast Discovery of Association Rules. *Advances in Knowledge Discovery and Data Mining*, 307–328, 1996.

[163] R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between Sets of Items in Large Databases. *ACM SIGMOD Conference*, 1993.

[164] R. Drezewski, L. Siwik. Multi-objective Optimization Using Co-evolutionary Multi-agent System with Host-Parasite Mechanism. *Computational Science - Lecture Notes in Computer Science*, 3993, 2006.

[165] R. Drezewski, L. Siwik. Multi-objective Optimization Technique Based on Co-evolutionary Interactions in Multi-agent System. *Workshops on Applications on Evolutionary Computation*, pages 179–188, 2007.

[166] R. Drezewski, L. Siwik. Agent-Based Co-Evolutionary Techniques for Solving Multi-Objective Optimization Problems. *Advances in Evolutionary Algorithms*, 2008.

[167] R. Elvins, P. Pointer, R. Vaidyanathan, S. Burgess. A Case Study Exploring Regulated Energy Use in Domestic Buildings using Design-of-Experiments and Multi-objective Optimisation. *Building and Environment*, 54:126–136, 2012.

[168] R. Pasupathy. SimOpt: A Library of Simulation Optimization Problems. *Winter Simulation Conference*, pages 4075–4085, 2011.

[169] R. Radhakrishnan, N. Vijaykrishnan, L. K. John, J. Sabarinathan. Java runtime systems: Characterization and architectural implications. *IEEE Transactions on Computers*, 50:131–146, 2001.

[170] R. V. Tappeta, J. E. Renaud. An Interactive Multiobjective Optimization Design Strategy for Decision Based Multidisciplinary Design. *40th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, pages 78–87, 1999.

[171] R. Zembowicz, J. Zytkow. From Contingency Tables to Various Forms of Knowledge in Databases. *Advances in Knowledge Discovery and Data Mining*, 329–351, 1996.

[172] S. A. Kalogirou. Optimization of Solar Systems using Artificial Neural Networks and Genetic Algorithms. *Applied Energy*, 77:383–405, 2004.

[173] S. Bandaru, K. Deb. Automated discovery of vital knowledge from Pareto-optimal solutions: First results from engineering design. *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2010.

[174] S. Bandyopadhyay, S. Saha, U. Maulik, K. Deb. A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*, 12:269–283, 2008.

[175] S. Carcangiu, A. Fanni, A. Montisci. Multi-Objective Optimization Methods Based on Artificial Neural Networks. *Search Algorithms and Applications*, 2011.

[176] S. Feldmann, P. LaBorde, D. Dechev. Concurrent Multi-level Arrays: Wait-free Extensible Hash Maps. *International Conference on Embedded ComputEntityr Systems: Architectures, Modelling, and Simulation (SAMOS XIII)*, 2013.

[177] S. Julier, y. Baillot, M. Lanzagorta, D. Brown, L. Rosenblum. BARS: Battlefield Augmented Reality System. *NATO Symposium on Information Processing Techniques for Military Systems*, pages 9 – 11, 2000.

[178] J.-P. Tolvanen S. Kelly. Domain-specific modelling: enabling full code generation. *John Wiley and Sons*, 2008.

[179] S. Kumar, M. Baseer, S. S. Bhowmick. A Multi-Version Transaction Model to Improve Data Availability in Mobile Computing. *OTM Confederated International Conferences - On the Move to Meaningful Internet Systems*, pages 322–338, 2002.

[180] S. M. Sanchez. Simulation Experiments: Better Data, Not Just Big Data. *Winter Simulation Conference*, pages 805–816, 2014.

[181] S. Muro, T. Kameda, T. Minoura. Multi-version Concurrency Control Scheme for a Database System. *Journal of Computer and System Sciences*, 29(2):207–224, 1984.

[182] S. Nolfi, D. Floreano. Evolutionary Robotics: The Biology, Intelligence, and Technology. *MIT Press Cambridge*, 2000.

[183] S. Rehfeld, H. Tramberend, M. E. Latoschik. An Actor-based Distribution Model for Realtime Interactive Systems. *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 2013.

[184] S. Rehfeld, M. E. Latoschik. A Comparison of Parallelization Methods for Data Flow Networks. *Software Engineering and Architectures for Realtime Interactive Systems SEARIS, proceedings of the IEEE Virtual Reality 2010 workshop*, 2010.

[185] S. Rehfeld, M. E. Latoschik, H. Tramberend. Estimating latency and concurrency of Asynchronous Real-Time Interactive Systems using Model Checking. IEEE Virtual Reality (IEEE VR), 2016.

[186] S. Robinson. Simulation: The Practice of Model Development and Use. *John Wiley and Sons*, 2004.

[187] S. Shan, G. G. Wang. An Efficient Pareto Set Identification Approach for Multiobjective Optimization on Black-box Functions. *Journal of Mechanical Design 127*, 5:866–874, 2004.

[188] S. T. Enns, P. Suwanruji. A Simulation Testbed for Production and Supply Chain Modeling. *Winter Simulation Conference*, 2003.

[189] S. Timnat, A. Braginsky, E. Petrank. Wait-Free Linked-Lists. *ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pages 309–310, 2012.

[190] T. Binh, U. Korn. An Evolution Strategy for the Multiobjective Optimization. *2nd International Conference on Genetic Algorithms*, pages 23–28, 1996.

[191] T. E. Hart, P. E. McKenney, A. D. Brown, J. Walpole. Performance of memory reclamation for lockless synchronization. *Journal of Parallel and Distributed Computing*, 67, 2007.

[192] T. Feldmann, M. Kavakli. VaiR: System Architecture of a Generic Virtual Reality Engine. *Computational Intelligence for Modelling, Control and Automation - International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, 2:501–506, 2005.

[193] T. Härder, A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983.

[194] T. Knott, B. Weyers, B. Hentschel, T. Kuhlen. Data-flow Oriented Software Framework for the Development of Haptic-enabled Physics Simulations. *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 2014.

[195] T. L. Harris. A Pragmatic Implementation of Non-blocking Linked-Lists. *In Proceedings of the 15th International Conference on Distributed Computing*, pages 300–314, 2001.

[196] T. Manoharan, H. Taylor, P. Gardiner. A Collaborative Analysis Tool for Visualisation And Interaction With Spatial Data. *Proceedings of the Seveneth International Conference on 3D Web Technology*, pages 75 – 83, 2002.

[197] T. Merrifield, J. Eriksson. Conversion: Multi-Version Concurrency Control for Main Memory Segments. *8th ACM European Conference on Computer Systems*, pages 127–139, 2013.

[198] T. Neumann, T. Mühlbauer, A. Kemper. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. *ACM International Conference on Management of Data (SIGMOD)*, pages 667–689, 2015.

[199] T. R. Brooks, J. R. R. Martins, G. J. Kennedy. High-fidelity Multipoint Aerostructural Optimization of a High Aspect Ratio Tow-steered Composite Wing. *58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2017.

[200] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. From Data Mining to Knowledge Discovery in Databases. *AI Magazine (Fall 1996)*, 17:37–54, 1996.