Multi Agent System Optimization in Virtual Vehicle Testbeds

Patrick Lange, Rene Weller, Gabriel Zachmann
University of Bremen
{lange,weller,zach}@cs.uni-bremen.de

ABSTRACT

Modelling, simulation, and optimization play a crucial role in the development and testing of autonomous vehicles. The ability to compute, test, assess, and debug suitable configurations reduces the time and cost of vehicle development. Until now, engineers are forced to manually change vehicle configurations in virtual testbeds in order to react to inappropriate simulated vehicle performance. Such manual adjustments are very time consuming and are also often made ad-hoc, which decreases the overall quality of the vehicle engineering process. In order to avoid this manual adjustment as well as to improve the overall quality of these adjustments, we present a novel comprehensive approach to modelling, simulation, and optimization of such vehicles. Instead of manually adjusting vehicle configurations, engineers can specify simulation goals in a domain specific modelling language. The simulated vehicle performance is then mapped to these simulation goals and our multi-agent system computes for optimized vehicle configuration parameters in order to satisfy these goals. Consequently, our approach does not need any supervision and gives engineers visual feedback of their vehicle configuration expectations. Our evaluation shows that we are able to optimize vehicle configuration sets to meet simulation goals while maintaining real-time performance of the overall simulation.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architectures—*Domain-Specific Architectures*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent System*; I.3.1 [**Computer Graphics**]: Hardware Architecture—*Graphics Processor*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Parallel*

Keywords

Virtual Testbed, Vehicle Simulation, Multi Agent System, Discrete Event Simulation, GPU, CUDA, Domain Specific Modelling

1. INTRODUCTION

Typically, the engineering of vehicles consists of three main steps: First, a mission scenario and corresponding requirements are defined, then engineers construct a vehicle that should fulfill these specifications and finally, test runs for fine-tuning of vehicle parameters, like fuel level or sensor orientation, are performed (see Figure 1).

While the basic principle remained the same since the beginning of modern engineering, the process has changed substantially during the past 40 years: The design moved from the drawing board to the computer and test runs are not only performed with physical prototypes in real hardware environments [19] but have also moved more and more to the computer.

In recent years, Virtual Reality (VR) has emerged as a key technology for improving and streamlining the conceptualisation and design of vehicles by simulation [11][10]. These testbeds give engineers the opportunity to interact with the simulated vehicle in order to gain comprehensive understanding of possible design flaws as early as possible during the design process. They are frequently used in fields like space robotics [10][21], underwater vehicles [7], or military ground vehicles [19]. In addition, it is widely recognized that simulation is pivotal to vehicle development, whether manned or unmanned [17]. Virtual testbeds are constituted by a sophisticated physically-based simulation of both the vehicle and its designated environment, as well as real-time, immersive rendering and 3D interaction techniques for the direct feedback and manipulation of the vehicle. In addition to the vehicle design process, the same virtual testbeds can often be re-used directly for later mission stages like training and supervision. Furthermore, virtual testbeds are a cost-efficient alternative e.g. for planetary exploration missions, or other autonomous vehicle applications in which setting up real mockups for testing and verifying is too expensive [11].

However, even in state-of-the art virtual testbeds, it is still challenging to identify the origin of errors. Engineers can waste a lot of time in the fine-tuning vehicle parameters while the error is in the vehicle design or even in the mission specifications.

.

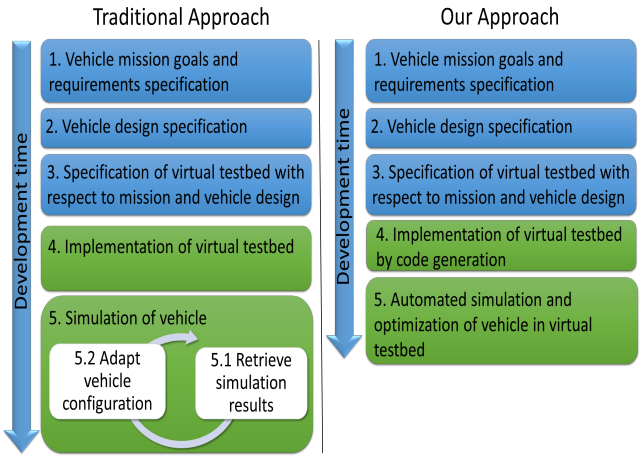

Figure 1: Integration of virtual testbeds with the vehicle engineering process: Simulation results are used to change vehicle configurations in order to meet engineer expectations if the simulated vehicle performance does not meet vehicle mission or engineering expectations. In the traditional approach, virtual testbed development as well as actual vehicle simulation take the most of the overall engineering process time (left). Our approach (right) reduces this development time significantly by introducing source code generation as well as autonomous vehicle optimization without supervision.

We propose a novel approach that overcomes these drawbacks. The main idea is to remove the last feedback-loop of the current design process completely (see Figure 1). To do that, we replace the manual parameter fine-tuning by an optimization method that adjusts these parameters automatically. This avoids the time-consuming and error-prone manual tweaking and helps the engineers to concentrate on their real work: the construction, testing and validation of the vehicle. In order to give engineers direct access to the virtual testbed, we additionally present a novel easy-to-use domain specific interface that allows the intuitive definition of simulation goals as well as vehicle parameters.

Consequently, we present a novel comprehensive approach to modelling, simulation, and optimization of such vehicles which greatly benefits the overall engineering process of autonomous vehicles.

In detail, our contributions are:

- A GPU based multi-agent system (MAS) for the automatic vehicle parameter adjustment: It computes objective and utility values for evaluating the current vehicle configuration with respect to fuzzy logic based simulation goals.
- A Domain Specific Modeling Language (DSML) which allows the visual development of a vehicle configuration and simulation as well as MAS based parameter optimization.
- A massively parallel domain framework, which enables wait-free discrete event simulation of autonomous vehicles. Additionally, this domain framework minimizes the synchronization overhead to the GPU to a minimum for fast parallel computations, without affecting the vehicle simulation performance.

In addition, our virtual testbed implements a 3D geometric visualization and functional vehicle simulation loop incorporating internal vehicle subsystems, sensors, actuators, and environmental physical influences.

In order to test, validate, and verify the navigation, control and localization algorithms of vehicles, engineers simply have to specify the simulation goals and the vehicle parameters in our novel DSML. Our virtual testbed supports interactive real-time adjustment of vehicle specifications like adding obstacles to the environment. The system computes an optimal set of parameter for these new specifications on-the-fly so that the engineer can inspect the results of the changes immediately. Consequently, our approach reduces the development and testing time significantly while simultaneously improving the quality of the parameter setting. We have applied our new approach to an autonomous spacecraft simulation. Our results show a real-time performance even for large parameter sets.

In order to achieve this real-time performance and easy code generation, we propose the use of a multi-agent system as it can be easily integrated into our underlying wait-free simulation system while maintaining easy code generation. Furthermore, multi-agent systems fully utilize the advantages of our wait-free simulation system due to its decentralized, parallel solving process behavior.

2. RELATED WORK

The main task of virtual testbeds in the vehicle engineering process is to support engineers with simulation results, based on simulation scenario and vehicle configuration. For each simulation run, the vehicle configuration has to be adjusted if the simulated vehicle performance does not meet mission or engineering expectations. Currently, this adjustment of vehicle parameters in virtual testbeds is either done externally by engineer experts that need to guide the adaptation, or the application has a number of scenarios. These scenarios are pre-defined by engineer experts to cover almost all aspects of the simulation.

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. Furthermore, pre-defined scenarios may lead to less optimal adaptation [13]. Consequently, it would be beneficial to automatically adjust the simulation online without the need of an expert guiding this process. Additionally, this online adaptation can run in vast amounts of simulation runs to find a vehicle configuration which can satisfy both mission and engineering requirements.

However, the increasing complexities of mission scenarios, goals and requirements as well as virtual testbed development process demands sophisticated development approaches which should perform in distributed, parallelized manner to attain real-time simulation behavior. Furthermore, the development approach should improve the overall development time of the virtual testbed in order to support the vehicle engineers as soon as possible in the engineering process with simulation results. Additionally, mission goals 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.

Multi-agent systems (MAS) can 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 [15]. Furthermore, interactions between agents can give birth to emergent phenomena (e.g. patterns, organizations, behaviors) [15]. We consider the vehicle 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.

The optimization system presented in this paper is therefore based on a modular hierarchical MAS in which self-organization principles are used to make the collective behavior emerge from local ones.

The use of GPU based MAS has been successfully advocated as a means to deal with this kind of complexity in various other highly sophisticated simulation applications such as astro-physics [8][27], graph algorithms [20], molecular dynamics [29], serious game adaptations [13], and traffic simulation [6]. These approaches, and other evaluation of GPU based MAS such as [3], show that GPU based MAS can incorporate a drastic performance improvement up to an average factor of 60 with respect to purely CPU based implementations.

In addition, advancements in software engineering can improve the development of sophisticated virtual testbeds. Model Driven Development (MDD) allows aspects of virtual testbeds to be represented formally as an abstract graphical model which can be automatically transformed into software artefacts and subsequently into complete simulation applications. MDD enables domain experts through a Domain Specific Modelling Language (DSML) to produce virtual testbeds for autonomous vehicles easily and quickly, as MDD notably promises great benefits to its practitioners. From a software development context, MDD offers an increase in productivity, promotion of interoperability and portability among different technology platforms, support for generation of documentation, and easier software maintenance [1]. 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 [26], which is highly desirable for researchers, engineers, and industry. Consequently, DSML lowers the development time and increases overall comprehension of simulation and optimization aspects of our virtual testbeds.

However, previous work on virtual testbeds for autonomous vehicles focussed highly on specialized partial aspects such as real-time simulation with state-of-the-art graphics [19][17], visualization for mission analysis [7] or sensor data generation for space robotic applications [11]. As the ever-increasing demand of sophisticated simulation as well as appealing graphical visualization increases the amount of interacting virtual testbed components, state-of-the-art virtual testbeds introduce centralized simulation data structures which are concurrently shared by all virtual testbed components [19][22][21][16][14]. Centralized data storages deliver many advantages for virtual testbeds such as smart simulation data logging, access rights management and homogeneous access to simulation data for the vehicle, environment or rendering. Nevertheless, such centralized data storages introduce a bottleneck to the overall virtual testbed as the concurrent access to it has to be managed. Especially multi-agent systems introduce a huge amount of software components which data access has to be successfully synchronized. In addition to this unsolved problem for state-of-the-art centralized virtual testbeds, none of the existing approaches addressed the complex development process of such virtual testbeds and how this development process could be integrated into the vehicle engineering process. Moreover, these virtual testbeds are highly specialized and therefore present most diverse software architectures for vehicles within their designated environments. Furthermore, none of the approaches involved any adaptation or optimization of the simulated vehicle. All state-of-the-art virtual testbeds need engineer expert supervision if vehicle configurations has to be adjusted as their primary objective is to give visual feedback of the simulated vehicle. Consequently, existing approaches are simply visual resemblances of the underlying simulation with no more benefits to the engineers.

However, system adaptations can be found in other simulation related fields, such as military training in serious games. Simulation and serious gaming is a strongly related field [18] in which system parameter adaptation is well-founded. Current adaptation concerns automated scenario generation, e.g. for military training purposes [2]. Such adaptive applications can be effectively modelled with MAS [12]. Furthermore, [13] showed in general how agents can be used to define serious game applications.

3. VEHICLE SIMULATION APPROACH

Enabling MAS based vehicle configuration requires a virtual simulation environment for the parameter optimization to take place. In this section, we will describe our simulation environment as well as domain framework and how they are related to our DSML. Our modelling approach is based on the lightweight DSML [24] 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 vehicles or even other domains, complete source code generation is no prerequisite.

Our lightweight DSML approach is used to define generic components of a virtual vehicle simulation: the environment, vehicle sensors and actuators, internal vehicle control components, virtual objects as well as multi-agent system. All components are situated in a vehicle simulation loop, which ensures that the vehicle can only perceive its environment via its sensors and that only actuator information are routed to the environment. Our lightweight DSL enables modelling of these components in a graphical manner as described in the next sections.

From our DSML approach, graphical Platform Independent Models (PIM) can be modelled by domain experts. These PIM are then used to automatically generate the source code of our virtual testbed, the Platform Specific Model (PSM), via horizontal model transformations [26].

Figure 2 depicts our vehicle simulation and optimization within our lightweight DSL approach.

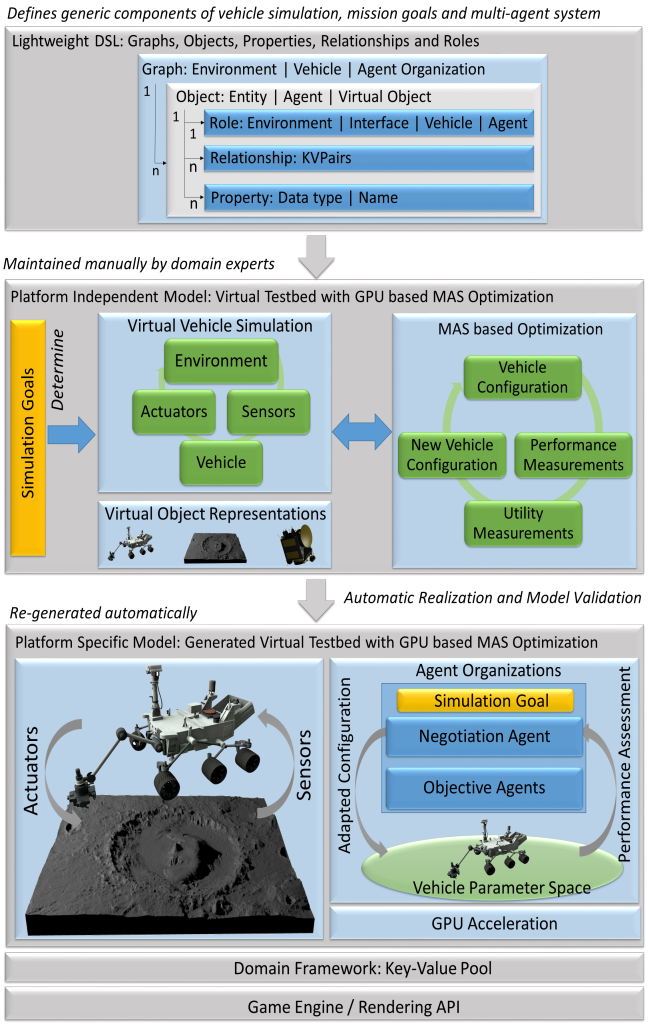

Figure 2: Overview of our proposed approach. From a lightweight DSL, graphical Platform Independent Models (PIM) are modelled which are used to generate our virtual testbed with its vehicle simulation and configuration optimization.

We will describe in the next sections our domain framework with its data flow for vehicle simulation and how it benefits the integration of GPU based multi-agent systems. Additionally, we will describe the infrastructure, optimization solving process and implementation of our GPU based MAS.

3.1 Domain Framework and Data Flow

Within domain specific modelling, a domain framework is used to execute generated code artefacts. Additionally, a domain framework 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 (JRE) [25].

The domain framework used in our approach should be a centralized solution in order to maintain software engineering advantages from current virtual testbed approaches [17][16] but should not introduce a bottleneck to the overall simulation and multi-agent system optimization. We presented a wait-free concurrency control management approach in [22][23][21]. This approach introduces a centralized data storage that drastically outperforms traditional approaches by several orders of magnitude. Consequently, we use our concurrency control management as the basis of our domain framework. We will describe in this section how we implemented a discrete event simulation and multi-agent system optimization of vehicles within this control management approach.

The core of our approach is a global dictionary, called key-value pool (*KVPool*), a centralized data storage that maintains the complete shared world state of the virtual testbed. Each simulation component that reads or write data to the KVPool is summarized as a *Entity*.

Each Entity that needs to share data to other Entities registers this shared data to the KVPool. Examples for such shared data are simulation time, vehicle position, sensor measurements or actuator commands. Registering shared data means the creation of a key-value pair (*KVPair*) in the global KVPool with a unique key which is required for fast identification. If Entities want access to the data, they simply have to pass this key to the KVPool. Each KVPair can be composed of arbitrary content, such as vectors, matrices, arbitrary numerics or geometry. Consequently, one KVPair can have an arbitrary amount of member data which makes a KVPair universally usable.

We presented in [22][23] how Entities can access the complete shared world state in wait-free manner. In contradiction to traditional lock-based concurrency approaches which are used by current virtual testbeds, Entities do not have to acquire a lock before manipulating our concurrently shared world state. Consequently, no synchronisation overhead is needed when solving for concurrent access. This leads to a dramatic performance increase with respect to traditional locking approaches for massively parallel access up to several orders of magnitude.

In addition to the wait-free access of the KVPool, the approach delivers a homogeneous interface for accessing the simulation state as well as for Entity communication. Such relationships are defined by a set of KVPair read and write operations. This encapsulation as KVPairs leads to easier DSL modelling concepts as the code generation for accessing simulation state as well as for simulation component data exchange can be represented by simply delivering the corresponding key for read and write access. Current virtual testbed data storage approaches even use full-fledged SQL databases [16][14][5]. In contrast to our KVPool, such approaches would make code generation way more complicated as complete SQL-queries would need to be generated for accessing the simulation state.

Consequently, our concurrency management approach delivers real-time performance as well as high software cohesion which facilitates code generation of the overall virtual testbed.

Within our KVPool approach, we describe a discrete event simulation loop of our domain framework as follows:

A state $S = (t, C, P, E)$ of our virtual testbed consists of simulation time t and set of Entities C, key-value pairs P, and events E. Additionally, a transition function δ is defined:

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

In each transition, the key-value pairs in the key-value pool are updated by the Entities and new transition events are generated by the system. To enable the modelling of vehicle environment interaction, Entities can incorporate one of four types:

- Environment: Defines simulation aspects from the environment in which the vehicle 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 vehicle, e.g. thrusters, gyroscopes, cameras, or range finder sensors.
- Vehicle: Defines internal vehicle components such as control loops, localization concepts, sensor fusion algorithms, BDI-structures or target detection.
- Agent: Defines an agent for observing and optimizing an associated vehicle configuration parameter.

For delivering a highly generic framework which can be adapted to many hardware configurations, those Entities can be situated in our discrete event simulation loop in one of two ways: First one being a CPU-GPU hybrid, second one being a purely CPU based approach.

In both versions, Entities that simulate the vehicle in its designated environment are directly mapped to our generic vehicle simulation loop, imposing a clear engineering concept even on source code level.

The versions differentiate in the way that the computation of the multi-agent system is either done on the GPU or CPU. In the first case, the multi-agent system will be computed on the GPU and the data transfer is done via one Entity. In the CPU based approach, every agent is modelled as one Entity. In both versions, all Entities benefit from our massively parallel domain framework, which enables access to the KVPool with no synchronisation overhead.

Figure 3 additionally depicts these two approaches.

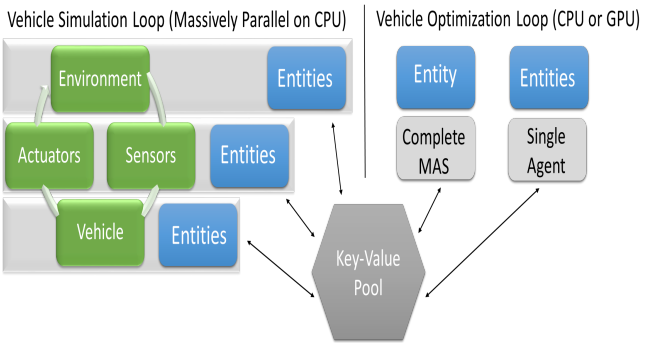

Figure 3: Entities are mapped to our generic vehicle simulation loop, consisting of the environment, sensors, actuators and internal vehicle components. Our MAS is either computed on the CPU with n Entities or on the GPU as one Entity with a single CUDA kernel for all agents.

4. OPTIMIZATION OF VEHICLE CONFIGURATION

The adaptation system proposed here is based on a hierarchical MAS which aims at dynamically tuning all the vehicle parameter values with respect to the simulated vehicle performance.

We will start with an overview which describes the overall MAS infrastructure with its modular agent organizations and relationships to the vehicle simulation. Following this, we describe the input and output data as well as communication structure of the agents. Additionally, we will describe the adaptation solving process with its negotiation mechanisms and how even contradictory simulation goals can be satisfied.

Our MAS is composed of several agent organizations which aim at optimizing each a subset of vehicle parameters to attain a pre-defined simulation goal by improving the simulated vehicle performance. These agent organizations are defined for each specified simulation goal and consist of a hierarchy of two agent types: objective and negotiation.

A objective-agent is used to measure one vehicle parameter. Consequently, there can be an arbitrary amount of objective-agents measuring the same vehicle parameter, each one measuring different simulation goal satisfaction values. Each agent type implements a certain functionality in order to maximize the related simulation goal, even if they are contradictory. For example, one simulation goal could be to minimize the vehicle fuel consumption. Therefore, the simulation goal could refer to the fuel tank capacity parameter of the vehicle, the propulsion fuel consumption efficiency or the number of allowed thrust ignitions. If more than one objective-agent is attached to a vehicle parameter, the corresponding optimization can be contradictory. For example, increasing the vehicle fuel capacity has an impact on the overall vehicle mass which reduces the main thrust efficiency. Therefore, negotiation-agents are used if several objective-agents refer to the same vehicle parameter. In addition to their objective value, objective-agents also deliver a utility value. These values are described in the next section. Figure 4 illustrates this agent organization concept with respect to the overall vehicle simulation.

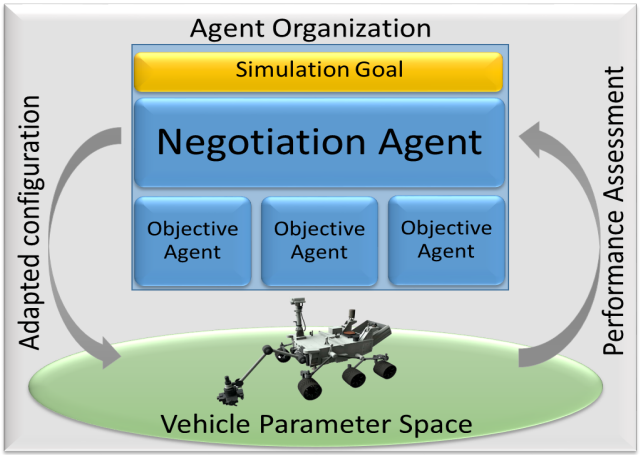

Figure 4: One agent organization is generated for each simulation goals. Each organization searches through the corresponding vehicle parameter space by measuring the vehicle performance and by adapting the configuration until the organization finds a satisfying solution.

4.1 Parameters, Objectives and Utilities

When building a vehicle configuration optimization system, all manipulable vehicle parameters needs to be uniquely identified. Additionally, a range for each of these parameters is given, defining the allowed values for this specific parameter. All adjustable vehicle parameters PAR constitute the input of our multi agent system; it is defined as follows:

$$PAR = \{p_1 ... p_n\}$$

$$p_i \in \{p_{min} ... p_{max}\} \subset \mathbb{R} \quad (1)$$

Analogous, objectives of the control system need to be defined. Objectives of simulations are often multiple and contradictory. Instead of defining a single general goal state of the simulation, several independent objectives are here considered, each one related to a parameter $p \in PAR$.

Requirements on autonomous systems are hard to express in an numerical way as humans often only have a general idea of their system expectations. Diverse solutions based on fuzzy logic have been proposed to solve this mapping from human-made requirements to measurable numerics [28]. This paper proposes the use of such fuzzy predicates to allow the smart definition of objectives as they involve a specific measure on the simulation which should not be necessarily complete satisfied or unsatisfied.

Satisfaction functions are introduced in order to determine the objective satisfaction for a given parameter. A satisfaction function takes as input a parameter and returns a satisfaction value in fuzzy predicate. This predicate is constituted within the [0;100] interval and yields every objective satisfaction value between unsatisfied (0) and completely satisfied (100). Depending on the relationship between the parameter and objective, a satisfaction function can be strictly or loosely following a parameter in (inverted) linear, quadratic or exponential manner. We introduce, in addition to the satisfaction function, an optional linear weighting term $*w + b$ which can scale the satisfaction value in order to increase the impact of the parameter within the solving process. The set of all objectives OBJ is defined as follows:

$$OBJ = \{o_1 ... o_n\} \rightarrow \{0, 100\}$$

$$o_i = satisfaction(p_i)[*w + b] \quad (2)$$

In addition to the objectives to be satisfied, a control system should consider the possible utility when changing the vehicle parameters with respect to the current simulation state. Consequently, we introduce a utility function. It is used to calculate the benefit for the current simulation state, if a parameter $p \in P$ would be adapted accordingly to the corresponding objective value o. The utility value is then later used by the negotiation agent to select the most appropriate action.

The set of all utility values is defined as follows:

$$UTL = \{u_1 ... u_n\} \rightarrow \{0, 100\}$$

$$u_i = 100 - o_i[*w + b] \quad (3)$$

The aim of the parameter adjustment control system is to find the values of PAR that maximize all objectives OBJ and minimizes all utilities UTL. In other words, it is to tune all the parameters so all the constraints and objectives are satisfied.

4.2 Solving Process Principle

When designing a multi-agent system, the focus is set on agents behaviors and communications in order to cover isolated parts of the global problem. Each agent tackles an isolated sub-problem and emergence is used to solve the overall problem. Therefore, the solving process is distributed among all agents. Every agent introduces a part-wise modelling of the problem and its behavior and communication to other agents is used to solve the global problem. Consequently, the definition of agent behaviors is also one of the key aspects of our multi-agent system and is described hereafter.

- Objective-agents compute a satisfaction value for a target parameter. An objective-agent has a rather simple behavior: it observes its target parameter and uses a satisfaction function to compute a objective satisfaction value. The goal of every objective-agent is to iteratively find a simulation state which yields a satisfaction value above a given minimum threshold. In order to do so, it writes a KVPair to its negotiation-agent, requesting a modification of its observed parameter in a proper way; this KVPair contains only the current objective satisfaction, utility and requested parameter variation sign (either positive or negative). Additionally, objective-agents compute the utility value of their proposed request. The utility value is basically a weighted inverse of the corresponding objective value of a parameter. Therefore, a utility value represents the degree to which a certain non-satisfied objective should be tackled, resulting in its utility value.
- Negotiation-agents monitor the objectives and utilities of all corresponding objective-agents of their agent organization. When several requests are received, it will choose the most appropriate parameter modification based on the highest utility value, implementing the English auction principle.

5. DOMAIN SPECIFIC MODELLING LANGUAGE

For defining our graph-based modelling language, the industry notation MetaCase+ GOPRR (Graphs, Objects, Properties, Relationships and Roles) [9] was used. All objects used within GOPRR are drawn into graphs that contain the object's 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.

This section will give a short overview of our GOPRR-notation based modelling approach. The aim of our modelling approach is to graphically describe simulation components as well as agents of our MAS.

Therefore, graphs in our notation can either be the environment, the simulated vehicle or a agent organization. Every graph contains objects: agent organizations are constituted of agents whereas vehicle and environment are represented by Entities and VirtualObjects. VirtualObjects are three-dimensional objects which are used to represent certain simulation aspects in the virtual testbed, e.g. a CAD vehicle model. Every object involves three main concepts: a role, relationships and properties. Every Entity has one of four roles: environment, interface, vehicle or agent.

These define the Entity's role with respect to the aforementioned simulation loop. Relationships between objects are expressed by key-value pair exchanges. Furthermore, object properties can be arbitrary data, such as numbers or strings. Formally, let

$$G = (\{Environment|Vehicle|Agents\}, \{G\}, \{O\}) \quad (4)$$

be a graph definition with child graphs {G} and objects {O} defined as

$$O = (\{Entity|Agent|VirtualObject\}, Ro, \{Re\}, \{P\}) \quad (5)$$

with

$$P = \{(Datatype, name)\} \quad (6)$$

a set of variables,

$$Ro = \{Environment|Interface|Vehicle|Agent\} \quad (7)$$

a role definition and

$$Re = \{KVPair\} \quad (8)$$

set of key-value pairs.

All modelling aspects are depicted in Figure 5.

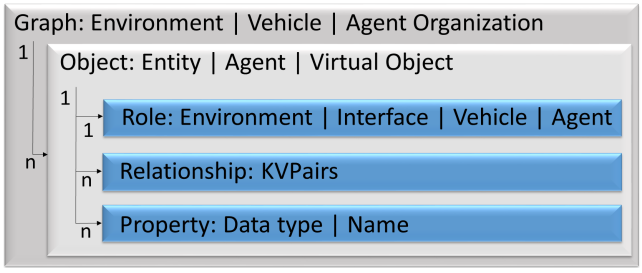

Figure 5: Simplified hierarchical depiction of our lightweight DSL approach, based on the GOPRR-notation.

5.1 Code Generation

Template-Based Code Generation (TBCG) [4] is used to generate source code from our PIM. TBCG is a generative technology that transforms a given model into source code, through the use of templates. 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 [4]. A template thereby consists of imperative control and structural source code patterns such as loops or conditional statements.

As everything in our virtual testbed is an Entity with an homogeneous interface to the overall shared simulation state by accessing a set of key-value pairs, our TBCG aims at generating the read and write processes of all Entities. Additionally, the CUDA code for the MAS is directly generated for the specified agent into the corresponding Entity specification.

Pseudo-code 1 gives a brief overview of the Entity class generation. The following parameters are derived from the DSL:

- ω : all specified Entities
- α : name of the Entity
- β : role of the Entity (respectively base class)
- γ : list of properties, structured as tuples with data type and name
- δ : list of key-value pairs which are read by the Entity
- ϵ : list of key-value pairs which are written by the Entity
- ζ : Entity type (agent or simulation component)
- η : Agent data list (non-zero if Entity type is agent), structured as tuples with data type and name

Algorithm 1 GenerateEntityImplementations

for $\omega_i \in \omega$ **do**
  Generate header file with derived class α from class β
  for $\gamma_i \in \gamma$ **do**
    Generate private variable $\gamma_i - name$ with $\gamma_i - type$ in header file declaration
  end for
  Generate cpp file with include to class α
  Generate empty read, write and work function of α
  for $\delta_i \in \delta$ **do**
    Generate **key-value pool read** of key δ_i for variable $\gamma_i - name$ in read function
  end for
  for $\epsilon_i \in \epsilon$ **do**
    Generate **key-value pool write** of key ϵ_i for variable $\gamma_i - name$ in write function
  end for
  if ζ = Agent **then**
    Generate work function:
    Generate **CUDA device memory** for *device-$\eta_i - name$ with $\eta_i - type$ and η_{size} with **cudaMalloc**
    Generate **host to CUDA device memory transfer** for device-η_i-name, η_i-name and η_{size} with **cudaMemcpy**
    Generate **CUDA kernel** call with η_{grid} η_{block} and list of device memory device-η-names
    Generate **CUDA device to host memory transfer** for objective, parameter and utility values
  end if
end for

5.2 Model Validation

Model validation is, besides code generation, one of the main aspects and benefits of domain specific modelling [26]. 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 rendering makes interface development tedious and often error- prone. We therefore validate the simulation data flow 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 key-value pair access of all Entities. We validate three simulation data flow constraints: If a key-value pair is defined and written by one Entity, at least one other Entity must read it, otherwise the modelling and generation of this key-value pair is unnecessary. Analogous, if a key-value pair is being read by at least one Entity, one other Entity must create this key-value pair. In order to prevent invalid user modelling of the virtual testbed, we also validate whether the simulation requirements are not violated: Our MAS must only change its designated parameters and the simulated vehicle must only perceive its environment by simulated sensor measurements. Furthermore, the simulated environment must only retrieve actuator information from the vehicle. Figure 6 illustrates these three model validation checks.

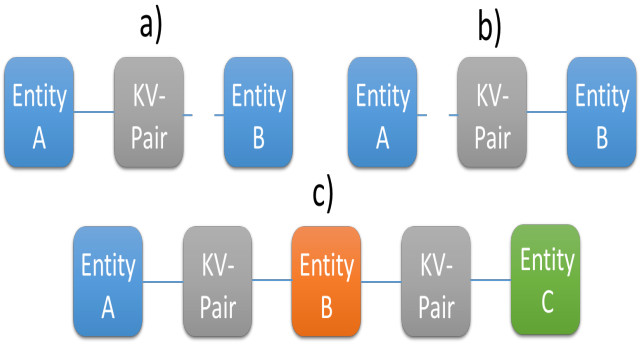

Figure 6: Model validation for simulation data flow: a) If a key-value pair X is written by Entity A, at least one other Entity B must read it from the same graph b) If a key-value pair X is read by Entity B, it must be written by another Entity A from the same graph c) If data from Entity A should be perceived by Entity C which is not situated in the same graph as Entity A, it has to be transmitted by an interface Entity B.

6. APPLICATION

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 our proposed approach as it consists of several environmental influences and vehicle internal control loops.

We modelled a simplified landing procedure the following way. A spacecraft tries to land on a asteroid surface. The spacecraft is under the influence of solar radiation pressure as well as asteroid gravity. The spacecraft itself has an internal control loop which acquires sensor data: acceleration (accelerometer), orientation (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 timings of thrust ignitions. In order to regulate the landing velocity, the spacecraft autonomously fires its main engine if the acceleration exceeds a given threshold. 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 our optimization is to dynamically tune this spacecraft configuration in order to avoid a crash on the asteroid's 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 as the main thruster and landing gear should be perpendicular to the asteroid's surface. Figure 7 shows an example rendering of our virtual testbed.

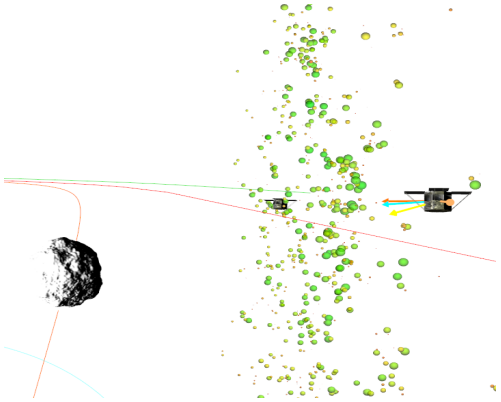

Figure 7: Example rendering of our approach within KaNaRiA [21]: On-board particle filter localization (color-coded spheres) before landing of our simulated spacecraft.

7. EVALUATION

We have implemented our vehicle simulation and optimization approach in C++ and CUDA 7. We performed our experiments on a machine with Intel Core i7 4-core processor with Hyperthreading enabled, Nvidia Quadro K1000M as well as Nvidia GTX 480 and 8GB of RAM.

We implemented two test scenarios. First, we modelled and generated, based on the previously introduced spacecraft application, our vehicle optimization and simulation. The first test scenario was used to evaluate whether the MAS based optimization solves for correct adaptations of the vehicle configuration. Second, we implemented synthetic benchmarks for our approach to evaluate overall system performance with respect to vehicle parameter optimization applications. The synthetic test scenario involved two competitors: a traditional non-parallel CPU implementation without the domain framework concept and our domain framework implementation without GPU support.

For our synthetic benchmarks, Figure 8 shows the mean average computation time for one complete simulation and optimization run. Our domain framework approach easily outperforms the traditional CPU based approach. Obviously, the speed-up of our approach increases with an increasing number of agents working in the system. Additionally, Figure 9 depicts the overall speed-up of our domain framework approach showing a speed-up of more than a factor of 60. Surprisingly, the difference between CPU based domain framework and GPU integrated version is very small for a few hundred agents. We believe that our wait-free domain framework pays-off well as the concurrent access to the KVPool is highly optimized with respect to the traditional CPU implementation.

For our spacecraft landing test scenario, Figure 10 shows the simulation objective satisfaction progress over time. Our approach is able to increase the satisfaction of all objectives until the simulation successfully ends.

Figures 11, 12 and 13 show how the main thrust level, fuel capacity and control thrust ignition configuration change over time. Our simulation successfully optimizes these parameters until the spacecraft safely lands after seven completed simulation loops with overall 1500 simulation steps.

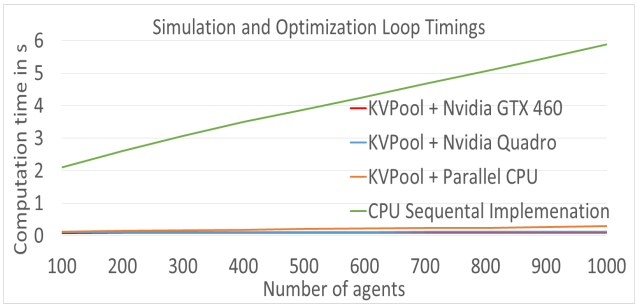

Figure 8: Computation time for one simulation and optimization run.

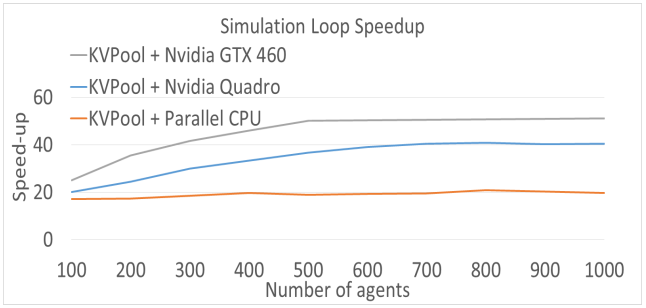

Figure 9: Performance speed-up for one simulation and optimization run with respect to the traditional CPU based approach.

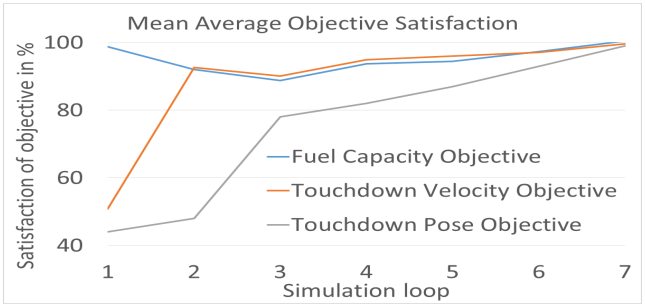

Figure 10: Our approach successfully changes the vehicle configuration in order to increase the simulation goal satisfaction.

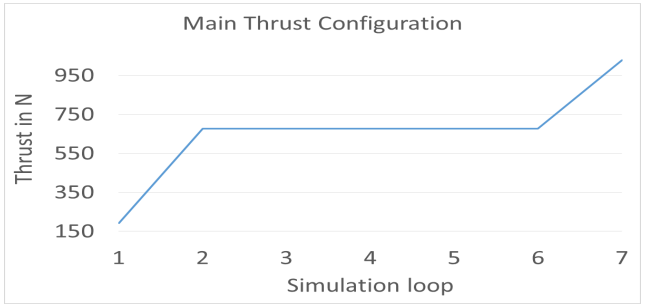

Figure 11: Our approach gradually increases the thrust level until the velocity can be adequately regulated for landing.

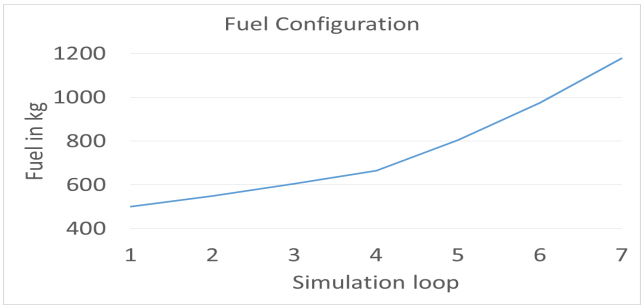

Figure 12: Our approach gradually increases the fuel capacity in order to maintain thrust while the landing procedure is conducted.

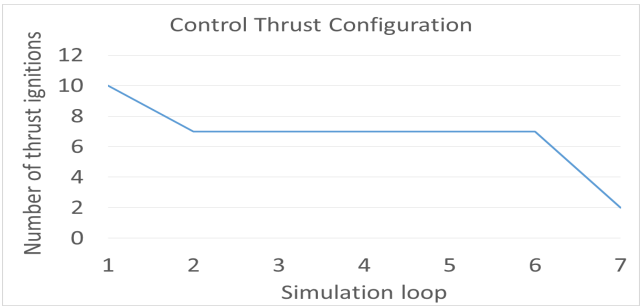

Figure 13: Our approach can gradually decreases the ignitions as the overall thrust level is increased by another agent.

Finally, we are able to generate almost 77 % source code of the aforementioned spacecraft test scenario. Currently, there are still some manual code changes for the internal vehicle control loops needed as well as for the satisfaction functions for the multi-agent system.

8. CONCLUSION

We have presented a novel comprehensive approach to modelling, simulation, and optimization of vehicles.

In our approach, engineers are no longer forced to manually change vehicle configurations. They can describe their vehicle expectations as simulation goals in our DSM framework. These simulation goals are then autonomously tracked and satisfied by our GPU based MAS which allows for optimization of vehicle configurations. Our MAS computes for each simulation step objective and utility values in order to compute an updated vehicle configuration until the specified simulation goals are satisfied. Synthetic benchmarks additionally show that our domain framework approach outperforms traditional approaches and that a majority of the virtual testbed source code can be generated from our models. This domain framework is based on our wait-free concurrency control management approach which additionally facilitates easy code generation due to homogeneous and simple access to the shared world state. Furthermore, due to the wait-free behavior of our key-value pool approach, even the CPU based implementation can be used for sophisticated real-time simulations.

However, the question remains unresolved if our MAS approach can solve any set of simulation goals as MAS do not necessarily compute

the global maximum of a parameter space. The emergence of MAS could also lead to wrong optimizations if, for example, the satisfaction functions are insufficiently implemented. Therefore, a generic modelling concept for objective satisfaction functions would greatly improve the overall usability as well as code generation of our approach.

In addition, the development of algorithms for autonomous retrieval of parameter - simulation goal relationships would further decrease the development time and increase the overall flexibility of our concept. To conclude, we also think that our framework, including domain specific modelling, GPU-assisted MAS and code generation, can be applied to other domains as long as parameter-based objective and utility values for simulation goals can be computed.

9. ACKNOWLEDGMENTS

This research is based upon the project KaNaRiA, supported by German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWi) under grant 50NA1318.

10. REFERENCES

[1] A. Kleppe, W. Bast, J. Warmer: MDA Explained: The Model Driven Architecture: Practice and Promise. In: *Addison Wesley Pub Co Inc* (2003)

[2] A. Zook, S. Lee-Urban, K. W. Brawner: Automated Scenario Generation: Toward Tailored and Optimized Military Training in Virtual Environments. In: *Proceedings of the International Conference on the Foundations of Digital Games* (2012), S. 164–171

[3] B. G. Alaby, K. S. Perumalla, S. K. Seal: Efficient Simulation of Agent-Based Models on Multi-GPU and Multi-Core Clusters. In: *SIMUTools 2010 3rd International Conference on Simulation Tools and Techniques* (2010)

[4] C. Henthorne, E. Tilevich: Code Generation on Steroids: Enhancing COTS Code Generators via Generative Aspects. In: *Proceedings of the Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques* (2007), S. 8–14

[5] C. Watanabe, Y. Masunaga: VWDB2: A Network Virtual Reality System with a Database Function for a Shared Work Environment. In: *Information Systems and Databases* (2002), S. 190–196

[6] D. Strippgen, K. Nagel: Using common graphics hardware for multi-agent traffic simulation with CUDA. In: *SIMUTools 2009 2nd International Conference on Simulation Tools and Techniques* (2009)

[7] D. T. Davis, D. Brutzman: The Autonomous Unmanned Vehicle Workbench: Mission Planning, Mission Rehearsal, and Mission Replay Tool for Physics-Based X3D Visualization. In: *Symposium on Unmanned Untethered Submersible Technology* (2005)

[8] E. Elsen, V. Vishal, M. Houston, B. Pande, P. Hanrahan, E. Darve: N-Body Simulations on GPUs. (2007)

[9] J.-P. Tolvanen, R. Pohjonen, S. Kelly: Advanced Tooling for Domain-Specific Modeling: MetaEdit+.

[10] J. Rossmann, B. Sommer: The Virtual Testbed: Latest Virtual Reality Technologies for Space Robotic Applications. In: *International Symposium on Artificial Intelligence, Robotics and Automation in Space* (2008), S. 133–138

[11] J. Rossmann, B. Sondermann, M. Emde: Virtual Testbeds for Planetary Exploration: The Self-Localization Aspect. In: *Symposium on Advanced Space Technologies in Robotics and Automation, ASTRA* (2011), S. 1–8

[12] J. Westra, F. Dignum, V. Dignum: Guiding User Adaptation in Serious Games. In: *Agents for Games and Simulations II* (2011), S. 117–131

[13] J. Westra, H. van Hasselt, F. Dignum: On-line Adapting Games using Agent Organizations. In: *IEEE Symposium on Computational Intelligence and Games* (2008)

[14] K. Walczak: Dynamic Database Modelling of 3D Multimedia Content. In: *Interactive 3D Multimedia Content* (2012), S. 55–102

[15] L. Pons, C. Bernon: A Multi-Agent System for Autonomous Control of Game Parameters. In: *IEEE International Conference on Systems, Man and Cybernetics (SMC)* (2013)

[16] M. Hoppen, M. Schluse, J. Rossmann, B. Weitzig: Database-Driven Distributed 3D Simulation. In: *Proceedings of the 2012 Winter Simulation Conference* (2012)

[17] 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. In: *Unmanned Systems Technology XI SPIE* (2009)

[18] M. Rohde, M. Toschlog: Toward the Fusion of Serious Simulation and Video Games. In: *Proceedings of the 2009 Spring Simulation Multiconference* (2009)

[19] P. Durst, M. Rohde, J. Crawford: A Real-Time, Interactive Simulation Environment for Unmanned Ground Vehicles: The Autonomous Navigation Virtual Environment Laboratory (ANVEL). In: *International Conference on Information and Computing Science (ICIC)* (2012), S. 7–10

[20] P. Harish, P.J. Narayanan: Accelerating large graph algorithms on the GPU using CUDA. In: *High Performance Computing HiPC 2007* 4873 (2007), S. 197–208

[21] P. Lange, A. Probst, A. Srinivas, G. Gonzales 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. In: *24th International conference on Artificial Reality and Teleexistence (ICAT-EGVE)* (2014)

[22] P. Lange, R. Weller, G. Zachmann: A Framework for Wait-Free Data Exchange in Massively Threaded VR Systems. In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)* (2014)

[23] P. Lange, R. Weller, G. Zachmann: Scalable Concurrency Control for Massively Collaborative Virtual Environments. In: *ACM Multimedia Systems - Massively Multiuser Virtual Environments (MMVE)* (2015)

[24] R. Pitkaenen, T. Mikkonen: Lightweight Domain-Specific Modeling and Model-Driven Development. In: *6th OOPSLA Workshop on Domain-Specific Modeling* (2006), S. 159–168

[25] R. Radhakrishnan, R. Radhakrishnany: Java Runtime Systems: Characterization and Architectural Implications. In: *IEEE Transactions on Computers* 50 (2001), S. 131–146

[26] S. Kelly, J.-P. Tolvanen: Domain-Specific Modelling: Enabling full code generation. In: *John Wiley and Sons* (2008)

[27] T. Hamada, T. Iitaka: The Chamomile Scheme: An Optimized Algorithm for N-body simulations on Programmable Graphics Processing Units. In: *New Astronomy* (2007)

[28] T. J. Ross: Fuzzy Logic with Engineering Applications. In: *3rd Edition* (2010)

[29] W. Liu, B. Schmidt, W. Mueller-Wittig: Molecular dynamics simulations on commodity GPUs with CUDA. In: *High Performance Computing HiPC 2007* 4873 (2007), S. 185–196