Modeling and Simulation as a Service using Apache Kafka

Moritz Gütlein[a] and Anatoli Djanatliev[b]

Computer Networks and Communication Systems, Friedrich-Alexander University Erlangen-Nürnberg, Martensstr. 3, Erlangen, Germany

Keywords: Modeling and Simulation as a Service, MSaaS, SaaS, Kafka, Docker, Simulation, Traffic Simulation, Virtual Mobility World.

Abstract: Among other requirements, in the field of Modeling and Simulation there is a need to build and run (co-) simulation models on-demand. Setting up, orchestrating and executing simulations should be easy, while additional features such as interaction interfaces open up a variety of applications. Being able to connect external components seamlessly to a running simulation allows for elaborate experiments. This paper describes the architecture and design of a simulation platform that is a part of a broader platform enabling evaluations of future mobility scenarios. Apache's Kafka platform for big data stream processing is used as a communication base in order to enable all these requirements, as well as for the coupling of different simulation tools forming a co-simulation. We give an overview of existing works regarding Modeling and Simulation as a Service, before explaining our own approach. Therefore, the architecture, the interfaces, and the workflow of a simulation run is described. The approach is illustrated by a case study, which is used to measure the service's overhead.

1 INTRODUCTION

The last decade's trend for cloud services affects also the field of modeling and simulation (M&S). Reasons for that are various, one of them is the fact that many research groups do not need continuous access to an excessive computing power, but only in certain times when simulation studies are run, and costs for dedicated machines are too high. Using shared computing nodes can lower costs and enhance utilization of the resources. In addition, it can be pleasant for a researcher, when there is no need to bother with maintaining a tool's code, fix bugs, or apply updates, but instead just use Modeling and Simulation as a Service (MSaaS) components and work straight ahead on the urging questions. Furthermore, such services can provide data pools for the model input data and help to organize existing models. Using MSaaS can therefore lower the time spent in preparing and running an experiment and improve the reproducibility of experiments by (re)using provided data sets, models, standard interfaces, and predefined workflows.

The cloud thought has never been alien in simulation, for instance, when performance constraints were fulfilled by using distributed simulations, when co-simulations were formed by coupling heterogeneous tools on different systems, or when digital twins were calibrated using live data from various sources. The idea is particularly suited to further improve simulation performance and quality of results. Additionally, simulation can benefit from the advances and developments arising from the trend of cloud services.

This paper describes the architecture of a novel MSaaS approach that uses the Apache Kafka messaging platform for all types of external communication, namely orchestration, interaction, and data provisioning; and all types of internal communication, namely time synchronization and simulation state related data exchange between coupled simulators. This leads to a well performing and clearly structured architecture of a simulation service that is extendable and capable to fulfill all simulation related requirements originating from a project called Virtual Mobility World (ViM).

Information about related work, associated technology and the ViM project is given in Section 2. Based on that, the proposed simulation service is then described in detail in Section 3, taking into account its architecture, communication, and process workflow. In Section 4, the provided information is illustrated by a potential use case and performance measurements, before conclusions are drawn and future directions are given in Section 5.

[a] https://orcid.org/0000-0001-5003-3976
[b] https://orcid.org/0000-0003-1549-3621

Gütlein, M. and Djanatliev, A.
Modeling and Simulation as a Service using Apache Kafka.
DOI: 10.5220/0009780501710180
In *Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2020)*, pages 171-180
ISBN: 978-989-758-444-2

2 BACKGROUND

In this section, we will develop a common understanding of the term MSaaS and give an overview of related work. We will then explain used technologies, and finally describe the idea of the ViM platform, which integrates the proposed MSaaS component.

The idea of cloud-based simulation services is not completely new. Shen (1998) proposes the concept of a *networked-service* for Discrete Event Simulation (DES) using Java and CORBA at the end of the last century. The motivation was already the lack of portability, interoperability, and scalability of existing simulation approaches. Strassburger et al. (1998) address "internet-based simulation" based on the High Level Architecture (HLA). HLA is a standard for distributed simulation that is still under active development. Cai et al. (2002) use HLA in a Grid Computing environment in order to realize load-balancing in distributed simulations.

Over the last few years, the interest in this subject has grown: Sarjoughian et al. (2008) and Mittal et al. (2009) show up similarities between Discrete Event System Specification (DEVS) used to formalize modeling and analysis of DES and service-oriented architectures (SOA). The former combine DEVS and SOA in a framework to simulate services themselves, while the latter propose a general-purpose framework to execute DEVS models remotely.

One of the first publications that introduce the term *cloud* in the context of simulation was done by Buyya et al. (2009). However, the work is not about M&S in the cloud, but about M&S of cloud computing environments in consideration of Quality of Service (QoS) requirements. The National Institute of Standards and Technology (NIST) defines *cloud computing* as:

> "A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."(Mell and Grance, 2011)

In addition, some essential characteristics are given: On-demand self-service, broad network access, resource pooling, rapid elasticity, measured service. While the first four are self-explanatory, the last characteristic means that cloud systems should automatically control and optimize their resource use. Therefore, virtualization and containerization is closely related to cloud computing (Dua et al., 2014). Fujimoto et al. (2010) connect the cloud and parallel and distributed simulation. Due to the typical cloud architecture, effects of simulation parallelization can usually be better exploited than on traditional architectures and lead to even higher performance gains. On the downside, security and reliability are named as issues. In addition, a concept for parallel and distributed simulation in the cloud is proposed based on a master/worker paradigm.

In 2013, a survey paper comes up with an early definition of MSaaS:

> "MSaaS is a model for provisioning modelling and simulation (M&S) services on demand from a cloud service provider (CSP), which keeps the underlying infrastructure, platform and software requirements/details hidden from the users. CSP is responsible for licenses, software upgrades, scaling the infrastructure according to evolving requirements, and accountable to the users for providing grade of service (GoS) and quality of service (QoS) specified in the service level agreements (SLA)." (Cayirci, 2013)

Besides MSaaS, the different "... as a Service" layers (i.e., Infrastructure, Platform, Software) that stand between the physical hardware and a user are clarified. Based on an overview of possible architectures potential threats and risks are discussed with the conclusion that MSaaS comes with potential advantages and open challenges.

Since military often drives innovation, one big player that is enrolled in MSaaS activities is NATO. The Work of NATO Modeling and Simulation Group MSG-131 ("Modelling and Simulation as a Service: New concepts and Service Oriented Architectures") sets the foundation for a twelve-year plan to implement MSaaS in their context (Siegfried et al., 2014). Based on the ITIL definition for services, they define MSaaS as follows:

> "M&S as a Service (MSaaS) is a means of delivering value to customers to enable or support modelling and simulation (M&S) user applications and capabilities as well as to provide associated data on demand without the ownership of specific costs and risks."

An extended definition of MSaaS is given later in a publication about the use of MSaaS by the NATO Modelling & Simulation Group MSG-136 ("Modelling and Simulation as a Service – Rapid deployment of interoperable and credible simulation environments"):

> "MSaaS is a new concept that combines service orientation and the provision of M&S ap-

> plications via the as-a-service model of cloud computing to enable more composable simulation environments that can be deployed and executed on-demand. The MSaaS paradigm supports stand-alone use as well as integration of multiple simulated and real systems into a unified cloud-based simulation environment whenever the need arises." (Siegfried et al., 2018)

All of the given MSaaS definitions have the "on-demand" thought in common. While the first two explicitly mention that responsibility is outsourced to the service provider, the latter adds that it should be possible to integrate multiple (real) systems when needed.

The amount of computational power that is often needed to run simulation is also the motivation for another work about a simulation service middleware by Shekhar et al. (2016). A user will provide custom parameters via an interface to start a simulation: Model name, number of simulation runs, deadline, number of CPUs, command to execute simulation, and already existing results. One use case, which is not described in detail, is about traffic simulation using the microscopic traffic simulator SUMO. As the service should be capable to start various types of simulations, there is the need to provide additional (use case specific) parameters. For the SUMO use case, the authors name the SUMO configuration file and a list of different traffic volumes for parameter variation as additional input parameters. They conclude that using a container-based solution (e.g., Docker) is better suited than a heavyweight hypervisor-based approach. More recently, Bocciarelli et al. (2019) focus on an architecture based on REST microservices in order to realize DES. Based on an existing middleware called SOASim (D'Ambrogio et al., 2016), Docker is used to deploy a simulation with containers and introduce an abstraction layer between simulation tool and operating system.

Besides from conceptual works on simulation in the cloud, there are works that address the use of MSaaS in specific application domains, e.g., the educational sector. Caglar et al. (2015) tackle the problem of a STEM (science, technology, engineering, and mathematics) crisis by enabling analysis of real-world systems in high school education by introducing simulations of such systems. In order to realize the use of simulators in the classroom, a cloud-hosted traffic simulation platform based on SUMO is proposed in the paper. Bitterman et al. (2014) also reinforce the use of MSaaS i.a. in the educational sector due to the low barriers regarding the availability of tools and data, similar to Krumnow (2013), who

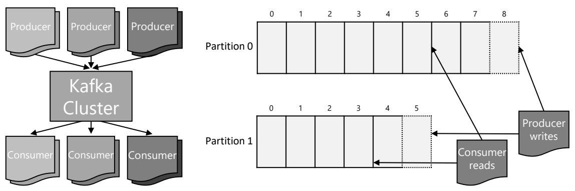

Figure 1: Kafka architecture and partitioned topic. Based on LinkedIn Corporation (2014).

introduces a web interface to SUMO as well. There are many works regarding traffic simulation, e.g., for ITS applications (Härri et al., 2010), connected vehicles (Kirchhof et al., 2019), or traffic simulation in a large scale (Hanai et al., 2014; Zehe et al., 2015). Apart from the traffic sector, MSaaS is used in other domains such as the energy sector (Preisler et al., 2015), weather simulation (Molthan et al., 2015), or for crowd modeling (Wang and Wainer, 2015).

To sum it up, there are various works about MSaaS concepts in general, as well as works related to very specific applications. Unfortunately, the implementation details of the proposed platforms or even access to them is mostly not available.

In the following, we will shortly address involved technology, namely Apache Kafka for communication and Docker for containerization. Apache Kafka is a publish/subscribe platform that was initially developed by LinkedIn to handle their massive amount of information (Apache Software Foundation, 2018). The software is released under Apache License 2.0. Thus, it is an open source project. In contrast to other publish/subscribe systems, such as MQTT, the focus lies on performance and scalability. In the Kafka world, a publisher is called a producer and is writing data to a Kafka cluster (LinkedIn Corporation, 2014). A subscriber is (called a consumer) is consuming data from the cluster (Figure 1). As with other publish/subscribe platforms, data is organized by topics. The topics themselves are realized as logs in Kafka, which can be further split up into partitions in order to achieve scalability. Due to replication of these partitions, fault tolerance is possible.

In terms of using software containers, Docker (Docker Inc, 2020) is one well-known implementation. In contrast to a virtual machine, a container is in general more lightweight and still allows packaging a piece of software with all needed dependencies in order to run it on various systems (Merkel, 2014) and therefore widely used in the cloud context.

The work presented in this paper is one component of the Virtual Mobility World (ViM Project, 2020). In this context, a platform is developed that supports companies, researchers, and additional stakeholders to create tomorrow's mobility services.

Therefore, the platform consists of a core and three main layers: storage, data analytics, and simulation (see Figure 2). These layers are encapsulated from each other and can be extended by more layers when new requirements arise. Potential users have to interact with an abstraction layer to pose questions or request datasets. The platform will respond on requests by using historical data, live data, performing analytical tasks, running simulations, or a combination or cascade of multiple options. The work happens hidden from the user, who simply receives the response. The logic behind these decisions is out of this paper's scope. Since this work focuses particularly on the simulation layer, its architecture and internal parts will be described in detail in the following section.

To the best of our knowledge, this is the first approach to use Kafka for simulation orchestration, interaction, and provisioning of live simulation data, as well as using it for coupling different tools involved in a requested co-simulation. In this way, additional protocols for distributed (co-)simulation such as HLA are not needed and the architecture remains simple but powerful. Due to the open interfaces between the platform and other components, such as data analytics, our MSaaS approach is more than a pure simulation service.

3 SIMULATION SERVICE

In this section, we will describe the addressed simulation service by naming requirements, showing the architecture, and giving information about the workflow of an instantiated simulation run.

3.1 Requirements

As the name implies, the main domain of ViM is traffic. Naturally, the main focus of the simulation layer is therefore also traffic simulation, but the concept can be applied to any other domain. Based on predefined use cases the following requirements have been identified for the simulation layer:

1. The simulation platform shall be able to run a traffic simulation on-demand, based on given attributes, such as road map, time frame, traffic pattern, random seed, or execution priority. Involved tools have to be specified with i.a. their step length, precision, responsible area in road map, and outputs. Optionally, it should be possible to add additional simulators from different domains such as wireless network simulation.
2. It should provide an interface to allow third-party components, as well as the other layers, to interact with a running simulation.
3. The component is required to push live and read-only data from running simulations on accessible channels for further processing, e.g., periodic status updates of vehicles.

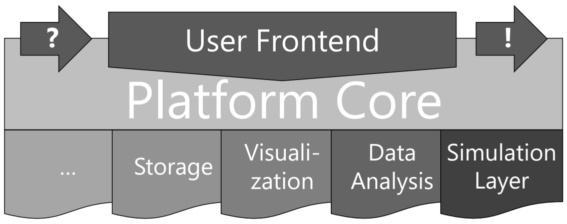

Figure 2: Architecture of ViM.

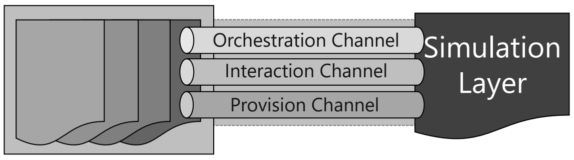

Figure 3: Three dedicated communication channels.

3.2 Communication

3.2.1 External Communication

To address these three main requirements, the usage of a publish/subscribe platform in conjunction with a channel communication model is proposed (see Figure 3). There are three dedicated communication channels, one for each requirement:

`orchestration`	(requirement #1)	(1)
`interaction`	(requirement #2)	(2)
`provision`	(requirement #3)	(3)

The use of a publish/subscribe messaging system allows for loose couplings and extensible interfaces. The communication will be realized by using the well-known Apache Kafka platform for big data streaming applications. Therefore, interaction with running simulations and information exchange can be done reliably and at high throughput rates.

Data (de)serialization is realized by using Apache Avro (Apache Software Foundation, 2020). Based on JSON schemas, data structures are defined beforehand. With a common knowledge of the defined schemas the serialization, transmission, and deserialization of data can be done efficiently and without much overhead.

`orchestration.simID` (4)

`interaction.simID` (5)

`interaction.simID.traffic.addVehicle` (6)

`provision.simID` (7)

`provision.simID.traffic.roadFoo.NOx` (8)

From the given examples, the channel design can be inferred. A request for a new simulation with the identifier `simID` is pushed to Topic 1 by the platform core, which is processed by the simulation layer and results in new base topics that are carrying the simulation id: Topic 4, Topic 5, and Topic 7. Depending on the requested simulation, topics corresponding to available API calls (e.g., Topic 6) or demanded periodic information (e.g., Topic 8) are created, watched, and populated by the simulation layer.

In general, the interaction channel is readable and writable by components and allows for (blocking) requests and responses. In this manner, the handling of sporadic events is achieved (e.g., a client can request and await the insertion of a new vehicle).

The provision channel in contrast, is used by some party to write information depending on scenario-wide defined presets (and without the knowledge of what is going to happen with the data) and used by other parties to consume and process the available data. An example application for this could be a traffic simulation tool that periodically writes positions of all vehicles to a corresponding topic in the provisioning channel and a visualization component that consumes and renders the locations onto a road map.

`provision.simID.traffic.veh42.speed` (9)

`provision.simID.traffic.veh42.avgSpeed` (10)

Additionally, the three channels are used to implement an access management, e.g., the visualization layer will not be allowed to write on topics of the orchestration channel. More elaborated strategies can also be applied by further limiting inherited topic properties, e.g., to implement privacy or license related policies for specific clients. One exemplary application for this would be giving the owner of the input data access to all topics in the provisioning channel (e.g., Topic 9), while another client is only allowed to read contents from aggregated topics that do not allow drawing conclusions about the input data (e.g., Topic 10).

3.2.2 Internal Communication

Having a concept for external communication between platform/components and the simulation layer, one question remains open: do we additionally need to address internal communication inside the simulation layer? More precisely: what to do, when there is more than one tool/instance involved in a simulation run? If more than one tool is involved in a simulation run (see Subsection 3.3.1), it is highly likely that these tools are supposed to communicate while the simulation is running. This means that a coupling of these tools needs to be achieved. Therefore, two different types of communication can be seen from a conceptual view. First, simulation related information is exchanged between the different tools that interact within a simulation (e.g., a vehicle's position that is transferred from the traffic simulator to the communication simulator). Second, simulation metadata (most importantly time synchronization related information) is shared. The two types of internal communication will be also realized by using Kafka. The first kind of information transfer will be implemented by using the data domain and data layer model (see Section 3.3.1) and corresponding API calls. All API calls that are available during a simulation run are provided under Topic 5. Multi-level simulations are realized via detached translation units that are capable of translating data tuples between different layers as proposed by Gütlein and Djanatliev (2019).

`orchestration.simID.time` (11)

For the synchronization of logical clocks, a conservative synchronization algorithm following the HLA Fujimoto (1998) time management mechanism is implemented in the simulation layer. Similar to HLA's *time regulating* and *time constrained* concepts, simulation tool instances will or will not participate in the time management mechanism based on the scenario's description file. Therefore, the actions `join`, `request`, `grant`, and `leave` are accessible under Topic 11. For each instantiated simulation run, a timing master takes care of the participants. The master calculates and publishes the common logical time during the simulation execution. Thus, a time-based synchronization is provided for coupling different tools and components.

3.3 Simulations on-Demand

In order to fulfill the first requirement, simulations should be started on-demand by the service. Therefore, the initial question for this requirement is: how to define a simulation?

3.3.1 Scenario Description

In this context, a simulation is defined as the single execution of a scenario: one or more connected tools with given inputs (e.g., road map, traffic demand, random seed) until a given simulation end time is reached. The set of all needed parameters is called *scenario description*. A scenario description is written in JSON format and sent to the simulation layer in order to trigger the execution of the scenario simulation. To allow future extensions, the scenario description has mandatory and optional fields, which will be described in the following (Figure 4).

While the first three fields are self-explanatory, the following require some additional information. In the next field, traffic input can be specified, before parameters related to the scenario execution are set. As the ViM platform is dealing with mobility issues, every simulation on the platform is based on a traffic simulation. That means, at least one traffic simulator needs to be given in the description by a scenario-unique identifier and a simulator type, which is needed to run the right software container. Additionally, a simulator is described using the layer concept as proposed by Gütlein and Djanatliev (2019). A layer is an interface definition that corresponds to the conceptual simulation model and belongs to a single domain. As this is about traffic simulation, the domain for traffic simulator entries is fixed to traffic, but the layer of the tool's native data model (e.g., micro) needs to be set in order to scope with different paradigms and model types (e.g., to provide matching API topics). The third thing defined by the layer is a set of parameters specifically related to the layer's corresponding simulation tools (e.g., the choice of a car-following model for microscopic traffic simulation). A set of possible layer interfaces and API calls is predefined (an example is given in Figure 5) and ready to be addressed. References to required resources can be given in the next field. Related to the layer, various definitions for different sets of resulting data are provided for each layer (e.g., trajectories for micro). These data sets can be requested by populating the results field and are returned when the simulation is finished.

Participation in the time synchronization process, as described before, can be set in the next field. Responsibilities and borders are used to partition a scenario in a spatial dimension to different simulator instances in order to gain performance, combine several modeling paradigms, or fulfill further requirements. Therefore, responsibilities and (outgoing) partition borders are given as road names of the used road network. To illustrate the meaning, an excerpt of a synthetic road network for a distributed traffic simulation is given in Figure 6. There are two responsibility regions for two simulators (A; dashed and B; dotted) and border links (white), which are connecting the two regimes.

Another important topic is the observer field. Observers are mainly introduced to populate the provision channel. An observer is a rule, defining how and when to deal with which kind of information. The main parts of an observer are a task, an attribute, a subject, a filter, a period and/or a trigger. The task answers the question of what to do with a piece of data. In a first draft, this task will essentially be publishing data to a provisioning topic for further processing. Future tasks incorporate the termination of

```
scenario description
├ id                                  sim42
├ road map                            manhattan.xml
├ time frame                          0:00 - 12:00
├ traffic
│ ├ processing                        native
│ ├ layer                             micro
│ └ file                              trips.xml
├ execution
│ ├ random seed                       194852
│ ├ constraints                       real-time
│ ├ priority                          high
│ └ synced participants               3
├ traffic simulators
│ ├ id                                sumo0
│ ├ type                              SUMO
│ ├ step length                       100
│ ├ layer                             micro
│ │ └ car-following                   krauss
│ ├ resources                         roadMap:withBikeLane.xml
│ ├ results                           fcd
│ ├ time sync                         regulating,constrained
│ ├ responsibilities                  all roads
│ ├ borders                           none
│ ├ observers
│ │ ├ task                            publish
│ │ ├ attribute                       emission
│ │ ├ subject                         road
│ │ ├ filter                          all
│ │ ├ period                          60000
│ │ └ trigger                         >100
│ └ custom params                     mode=balistic
├ translators
│ ├ id                                mesoMicroCropper
│ ├ domain                            traffic
│ ├ input layer                       meso
│ ├ output layer                      micro
│ └ custom params                     ...
└ additional simulators
  ├ id                                ns30
  ├ type                              NS3
  ├ domain                            communication
  ├ layer                             802.11p_PHY
  │ └ ...
  └ ...
```

Figure 4: Scenario description.

```
data model               interaction methods
├ string vehicleID       ├ void   addVehicle(string,...)
├ string route           ├ void   removeVehicle(string)
├ string edge            ├ double getSpeed(string)
├ int    lane            ├ void   setSpeed(string,double)
├ double speed           ├ string getRoute(string)
├ double position        ├ void   setRoute(string,string)
└ ...                    └ ...
```

Figure 5: Definition of micro layer in traffic domain.

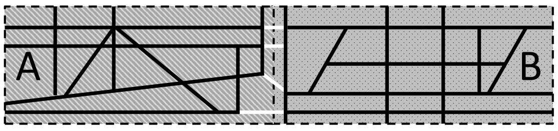

Figure 6: Road network with two traffic simulators' (A,B) responsibilities and connecting border links (white).

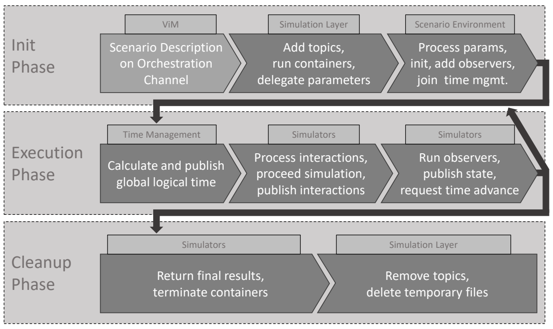
Init Phase
ViM
Scenario Description on Orchestration Channel
Simulation Layer
Add topics, run containers, delegate parameters
Scenario Environment
Process params, init, add observers, join time mgmt.
Execution Phase
Time Management
Calculate and publish global logical time
Simulators
Process interactions, proceed simulation, publish interactions
Simulators
Run observers, publish state, request time advance
Cleanup Phase
Simulators
Return final results, terminate containers
Simulation Layer
Remove topics, delete temporary files

Figure 7: Workflow for execution of a scenario.

a scenario run or the extraction of a snapshot (e.g., because a requested situation was achieved). Second, there is the attribute of interest (e.g., emission or speed) and its subject (e.g., road or vehicle). The filter field allows specifying if the attribute is queried from all subjects of the specified group or only subjects with given identifiers (e.g., road42). Lastly, the task can be triggered periodically, event-based (e.g., when a threshold is exceeded), or both at the same time. To avoid unnecessary communication and processing, the observer handling has to be done inside the simulation tool container. Since there is the need for a wrapper for every used simulation tool in order to communicate via the defined interface over the Kafka messaging platform, this is also a suitable location for observing the requested attributes via a tool's native API.

Finally, there is the option to give custom parameters that will be directly delegated to the simulator. Therefore, these parameters can be tool specific and do not need to be understood by the simulation layer. Afterwards, translators (see Subsection 3.2.2) for a multi-level simulation can be defined. The last field can be used to add additional simulation tools from different domains than traffic (e.g., for simulating wireless communications) to a scenario. Due to the variety of tools and inputs, there is also a custom field for handing over application-specific parameters.

3.3.2 Architecture and Scenario Instantiation

Based on the given description, how does the execution of a simulation scenario works in detail? There are three phases, which are depicted with clarifying tasks per step in Figure 7:

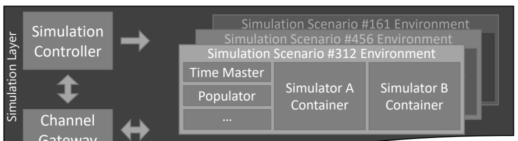

Figure 8: Architecture inside the simulation layer.

1. Initialization Phase

   The first phase is triggered by pushing a scenario description onto the orchestration channel. Based on the execution parameters (e.g., execute as soon as possible) and current system state further actions are immediately taken or scheduled for later. When action is taken, the requested tool containers are instantiated by the controller, topics are created and connected via a gateway component, and tool related parameters from the scenario description file are forwarded to the containers. Additionally, a time master is spawned for time management (see Figure 8). The running containers process the parameters, run their initialization methods, possibly start observer tasks, and may register for the time management mechanism. Due to the messaging and containerization, additional computing nodes could be integrated easily, as well as load balancing strategies.

2. Execution Phase

   When every component has registered at the time master, phase two is entered and an initial time step is granted. While simulation end has not been reached, the three steps of the execution phase are executed repeatedly. After a new global logical time is published, tools proceed their simulations to this time (if they are time constrained) while processing and creating interactions. Afterwards, scheduled observer tasks are run, scheduled state feedback is published, and finally the proceeding to the next simulation step is requested from the time master (if they are time regulating).

3. Cleanup Phase

   The third phase is reached, when the simulation of the scenario is finished. Summaries and results are published before the tool containers are terminated. Finally, the simulation layer cleans up the left over topics.

In summary, the proposed MSaaS architecture provides abilities that allow users and components to

- define and model simulation scenarios,
- (re)simulate custom simulation scenarios deterministically and on-demand,
- interact with a running simulation (e.g., control a specific EGO-car inside a traffic simulation),

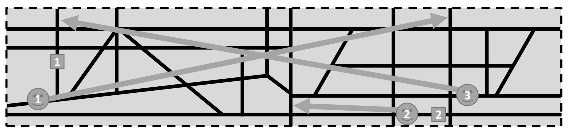

Figure 9: Task assignment in fleet management.

- gather data layer specific information via forwarded API calls,
- process simulation state online and trigger actions (e.g., watch for accidents and export current simulation state for further analysis),
- feed back the results from data analytic jobs into the same running simulation (e.g., evaluation of multistage traffic management strategies).

4 CASE STUDY

In order to demonstrate and illustrate the abilities of the presented approach, one exemplary use case will be described in the following. Afterwards, related performance measurements are given and compared to a traditional approach.

Fleet management is used to plan and control the use of a set of vehicles. Depending on the business case, priorities, tasks, and open questions vary. These may include monitoring, assigning of maintenance jobs, relocating vehicles prospectively, or assigning vehicles to a customer request. Obviously, these decisions are usually not made by hand, but by algorithms. In this exemplary use case, two fleet management algorithms should be compared. Therefore, a microscopic traffic simulation based on the road map of a real city is run in conjunction with the different fleet management algorithms. The microscopic simulation is demanded, in order to be able to model finely granular relocation processes in a realistic traffic situation.

A user provides a population model for generating customer request and the fleet management algorithms, which trigger actions in the traffic simulation. As presented in Subsection 3.2, the third-party components can talk with the simulation via the interaction channel. When the experiment is started, a handle to the corresponding channels is returned and the components can connect remotely. Due to the design, the intellectual property of the algorithms are protected and preserved by hiding the internal logic from the rest of the platform in a black-box manner. Obviously, a malicious component could try to reverse-engineer the outsourced logic, but that is out of scope of this work.

A very simple but striking example is constructed: there are two different strategies (A and B) for assigning vehicles to user requests. First, there is a greedy one, where the closest available vehicle is simply assigned to a customer. Strategy B tries to maximize total utilization. There are three customers (circles) and two vehicles (squares) visualized in Figure 9. Probably, the greedy version would assign vehicle 1 to customer 1 and vehicle 2 to customer 2, which results in declining customer 3. Contrary, Algorithm B would assign vehicle 1 to customer 1, but vehicle 2 to customer 3, and decline customer 2's request. The analytics component monitors user requests and depending on chosen key performance indicators, the second strategy might be evaluated as the better algorithm. Having results for one set of algorithms and a single scenario is not yet useful, but multiple replications with varied random seeds are quickly realized with the presented approach. This leads finally to meaningful results. Different metrics for the customers' satisfaction (e.g., waiting times, travel times, walking distance) and system metrics (such as fleet size, fleet utilization, and driven kilometers) are calculated by an analytics component based on the information provided on the provision channel. By this, an affiliated user can explore the outcome of the various metrics and rate the quality of the individual fleet management algorithms that were under evaluation.

Controversially, the question of which algorithm performs better is not relevant in this paper. The interesting question is: at which costs do all the offered possibilities come? As a basis, we estimated the overhead in this work with a simple case, without exploiting performance gains by distributing the simulation. With this setup, no performance enhancement is possible, but an evaluation of the pure overhead is.

Therefore, we conducted two different experiments: First, we let a potential fleet management algorithm make direct API calls to a SUMO simulation (Baseline Run). Leaving out all the presented abstraction and communication, this is the fastest solution possible and results in a good baseline to compare with. The second experiment will be a running SUMO container and a dedicated fleet management algorithm, both synchronized to a time master, and loosely coupled via Kafka as previously presented (SaaS Run). In both of the experiments, the algorithm queries a list of every active vehicle once every minute to have some communication load beside the time synchronization.

The experiments were executed on an i7-7600U machine with SUMO 1.2 and Kafka 2.3.0. The well-known LuST scenario (Codeca et al., 2017) was simulated from 5 am to 6 am with a step length of 100 ms. Ten repetitions were captured for both trials. The time synchronization between the fleet management

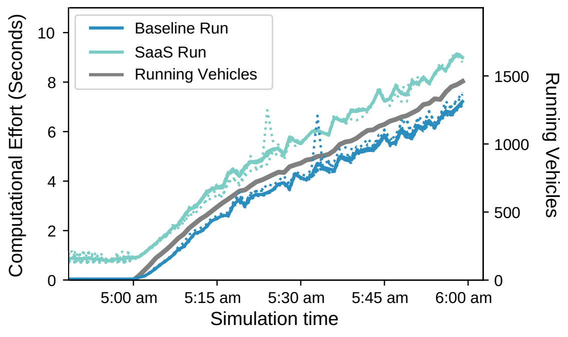

Figure 10: Performance comparison.

algorithm and SUMO was done with a step length of one second. Since the list of vehicles is queried once a minute, we measured the computational time for every 60 seconds of simulation time (Figure 10). While some outliers can be observed in the individual runs (dotted lines), the solid lines show the averaged effort over the ten repetitions.

Shortly before 5 am, when no vehicles are in the simulation yet, the pure synchronization overhead can be identified and is about one second per minute of simulation time and thus ~17 ms per synchronization step. With increasing traffic load, the computational effort in both experiments grows. Naturally, the baseline run will always be faster as this also the base for the SaaS run, where additional layers are added. The difference between the two trials increases slightly to ~1.9 s at 6 am and can be explained with the grown volume of exchanged data due to the number of running vehicles. The overhead can clearly be measured, but it does not exceed the simulation effort once some vehicles are inserted. With increasing computational costs the measurable effects become minor. Thus, enabling MSaaS opportunities, the additional effort is considered reasonable.

5 CONCLUSIONS

In this paper, we described an MSaaS component that is based on Apache Kafka. With the presented architecture, it is possible to fulfill the simulation related requirements that result out of the ViM platform related to new mobility services. Furthermore, the design allows future extensions, the integration of new use cases, and its use in different environments. All communication types - whether for orchestration, tool coupling, or external interfaces - have been mapped to the three-channel approach on top of the Kafka publish/subscribe messaging platform. After the architecture, the scenario description, and the workflow was explained, we gave an exemplary application and estimated the resulting overhead on a standard computer. Once some traffic load was injected, the SaaS overhead became secondary - even though only a single SUMO instance was used. By using the proposed service, the distribution over several nodes can be realized and thereby performance gains can be achieved.

Future work includes the measurement of simulation performance compared to classical distributed simulation approaches, such as HLA. Kafka's scalability raises curiosity regarding the performance of large-scale scenarios in a dedicated cloud environment. Due to Kafka's popularity, various extensions can be added. One of these provides, for instance, the ability to run live queries on Kafka topics in an SQL manner. This allows for an easy event processing within a running simulation, e.g., to detect collisions or traffic jams and trigger further actions. As the concept of data dimensions and layers is not limited to the traffic simulation domain, the extension of the service to other areas seems promising.

ACKNOWLEDGEMENTS

This work is part of the Virtual Mobility World (ViM) project and has been funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy (StMWi) through the Centre Digitisation.Bavaria, an initiative of the Bavarian State Government.

REFERENCES

Apache Software Foundation (2018). Apache Kafka Documentation. https://kafka.apache.org/0101/documentation.html (accessed April 3, 2020).

Apache Software Foundation (2020). Apache Avro. https://avro.apache.org (accessed April 13, 2020).

Bitterman, T., Calyam, P., Berryman, A., Hudak, D., Li, L., Chalker, A., Gordon, S., Zhang, D., Cai, D., Lee, C., and Ramnath, R. (2014). Simulation as a service (SMaaS): a cloud-based framework to support the educational use of scientific software. *Int. J. of Cloud Computing*, 3:177 – 190.

Bocciarelli, P., D'Ambrogio, A., Giglio, A., and Paglia, E. (2019). A microservice-based approach for fine-grained simulation in MSaaS platforms. In *Proc. of the 2019 Summer Simulation Conf.*, SummerSim '19, pages 1–12. Society for Computer Simulation Int.

Buyya, R., Ranjan, R., and Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In *2009 Int. Conf. on High Performance Computing & Simulation*, pages 1–11. IEEE.

Caglar, F., Shekhar, S., Gokhale, A., Basu, S., Rafi, T., Kinnebrew, J., and Biswas, G. (2015). Cloud-hosted simulation-as-a-service for high school STEM education. *Simulation Modelling Practice and Theory*, 58:255–273.

Cai, W., Turner, S. J., and Zhao, H. (2002). A load management system for running HLA-based distributed simulations over the grid. In *Proc. of the 6th IEEE Int. Workshop on Distributed Simulation and Real-Time Applications*, pages 7–14. IEEE.

Cayirci, E. (2013). Modeling and simulation as a cloud service: a survey. In *Proc. of the 2013 Winter Simulation Conf. (WSC)*, pages 389–400. IEEE Press.

Codeca, L., Frank, R., Faye, S., and Engel, T. (2017). Luxembourg SUMO traffic (LuST) scenario: Traffic demand evaluation. *IEEE Intelligent Transportation Systems Magazine*, 9(2):52–63.

D'Ambrogio, A., Bocciarelli, P., and Mastromattei, A. (2016). A PaaS-based framework for automated performance analysis of service-oriented systems. In *Proc. of the 2016 Winter Simulation Conf. (WSC)*, pages 931–942. IEEE.

Docker Inc (2020). Docker Website. https://www.docker.com (accessed April 3, 2020).

Dua, R., Raja, A. R., and Kakadia, D. (2014). Virtualization vs containerization to support PaaS. In *2014 IEEE Int. Conf. on Cloud Engineering*, pages 610–614.

Fujimoto, R. M. (1998). Time management in the High Level Architecture. *Simulation*, 71(6):388–400.

Fujimoto, R. M., Malik, A. W., Park, A., et al. (2010). Parallel and distributed simulation in the cloud. *SCS M&S Magazine*, 3:1–10.

Gütlein, M. and Djanatliev, A. (2019). Coupled traffic simulation by detached translation federates: An HLA-based approach. In *Proc. of the 2019 Winter Simulation Conf. (WSC)*, pages 1–12. IEEE.

Hanai, M., Suzumura, T., Ventresque, A., and Shudo, K. (2014). An adaptive VM provisioning method for large-scale agent-based traffic simulations on the cloud. In *2014 IEEE 6th Int. Conf. on Cloud Computing Tech. and Science*, pages 130–137. IEEE.

Härri, J., Killat, M., Tielert, T., Mittag, J., and Hartenstein, H. (2010). Demo: Simulation-as-a-service for ITS applications. In *2010 IEEE 71st Vehicular Tech. Conf.*, pages 1–2. IEEE.

Kirchhof, J. C., Kusmenko, E., Rumpe, B., and Zhang, H. (2019). Simulation as a service for cooperative vehicles. In *2019 ACM/IEEE 22nd Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 28–37. IEEE.

Krumnow, M. (2013). SUMO as a service–building up a web service to interact with SUMO. In *Simulation of Urban MObility User Conf.*, pages 62–70. Springer.

LinkedIn Corporation (2014). Apache Kafka Introduction. https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines (accessed April 3, 2020).

Mell, P. M. and Grance, T. (2011). The NIST definition of cloud computing. Technical report, Gaithersburg, MD, USA.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Mittal, S., Risco-Martín, J. L., and Zeigler, B. P. (2009). DEVS/SOA: A cross-platform framework for net-centric modeling and simulation in DEVS unified process. *Simulation*, 85(7):419–450.

Molthan, A. L., Case, J. L., Venner, J., Schroeder, R., Checchi, M. R., Zavodsky, B. T., Limaye, A., and O'Brien, R. G. (2015). Clouds in the cloud: weather forecasts and applications within cloud computing environments. *Bulletin of the American Meteorological Society*, 96(8):1369–1379.

Preisler, T., Dethlefs, T., and Renz, W. (2015). Simulation as a service: A design approach for large-scale energy network simulations. In *2015 Federated Conf. on Computer Science and Information Systems (FedCSIS)*, pages 1765–1772. IEEE.

Sarjoughian, H., Kim, S., Ramaswamy, M., and Yau, S. (2008). A simulation framework for service-oriented computing systems. In *Proc. of the 2008 Winter Simulation Conf. (WSC)*, pages 845–853. IEEE.

Shekhar, S., Abdel-Aziz, H., Walker, M., Caglar, F., Gokhale, A., and Koutsoukos, X. (2016). A simulation as a service cloud middleware. *Annals of Telecommunications*, 71(3):93–108.

Shen, C.-C. (1998). Discrete-event simulation on the internet and the web. *Simulation*, 12(3):6.

Siegfried, R., Lloyd, J., and Berg, T. (2018). A new reality: Modelling & simulation as a service. *J. of Cyber Security and Information Systems*, 6(3):18–29.

Siegfried, R., van den Berg, T., Cramp, A., and Huiskamp, W. (2014). M&s as a service: Expectations and challenges. In *2014 Fall Simulation Interoperability Workshops (SIW), Orlando, Florida*, pages 248–257. SISO.

Strassburger, S., Schulze, T., Klein, U., and Henriksen, J. O. (1998). Internet-based simulation using off-the-shelf simulation tools and HLA. In *Proc. of the 1998 Winter Simulation Conf. (WSC)*, pages 1669–1676. IEEE.

ViM Project (2020). Virtual mobility world. https://www.vim-project.org (accessed April 3, 2020).

Wang, S. and Wainer, G. A. (2015). A simulation as a service methodology with application for crowd modeling, simulation and visualization. *SIMULATION*, 91:71 – 95.

Zehe, D., Knoll, A., Cai, W., and Aydt, H. (2015). Semsim cloud service: Large-scale urban systems simulation in the cloud. *Simulation Modelling Practice and Theory*, 58:157 – 171. Special issue on Cloud Simulation.