

**Data-Centric Distributed Simulation in the
Traffic Domain**

—

**Datenzentrische Verteilte Simulation in der
Verkehrsdomäne**

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg

zur
Erlangung des Doktorgrades

DOKTOR-INGENIEUR

vorgelegt von
Moritz Gütlein
aus Hassfurt

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung: **19. Juli 2023**

Gutachter: **Prof. Dr.-Ing. habil. Reinhard German**
Prof. Dr. Oliver Rose

Abstract

Rapid technological progress offers opportunities that were unthinkable a few years ago. Sophisticated solutions may utilize a variety of domain-specific characteristics or benefit from synergy effects, when addressing cross-domain problems. As mobility is a basic need, it is an important representative for this development. For instance, technologies are subject to research that are aiming to strengthen road safety by exchanging information between traffic participants and/or the infrastructure. Hence, traffic and communication represent one example for a beneficial combination of domains. The Modeling and Simulation (M&S) of inventions in such fields can be very helpful throughout the development phase, but also while operating such systems. However, with an increasing number of involved subsystems of various kinds, the M&S is potentially also getting more and more complex.

In this thesis, we face existing challenges regarding the capabilities of current M&S approaches using a novel data-centric methodology. At first, current requirements towards simulation are identified. In that light, the potential weak spots of current methods, standards, and frameworks unveil. Based thereon, a main part of this thesis is to design a new coupling approach that bridges the depicted gap. This is done by developing a layered architecture that covers all parts that are necessary to support the heterogeneous stakeholders of modern simulation studies. In an infrastructural layer, the communication between components and the storage of simulation related data is devised. A second layer is covering all tasks that are related to the coupling of various components, while aiming for reusability and reproducibility. Finally, an application layer at the top is offering the capabilities to use the developed concepts in form of a M&S service.

In a second part, the approach is applied to the field of traffic simulation. By doing so, we show the practicability of our approach and demonstrate its capabilities to enable distributed simulations, multi-level simulations, and cross-domain simulations. Moreover, we present the feasible ingestion of external data into running simulations, the simulative enrichment of data, and the reuse of existing simulation data for new studies.

Kurzfassung

Der rasante technologische Fortschritt bietet uns Möglichkeiten, die vor einigen Jahren noch undenkbar waren. Anspruchsvolle Lösungen berücksichtigen vielfältige Zusammenhänge innerhalb einer Domäne oder profitieren von Synergieeffekten bei domänenübergreifenden Problemen. Da Mobilität ein Grundbedürfnis ist, stellt sie einen wichtigen Vertreter für diese Entwicklung dar. Beispielsweise sind Technologien Gegenstand der Forschung, die die Sicherheit im Straßenverkehr erhöhen sollen, indem Informationen zwischen Verkehrsteilnehmern und/oder der Infrastruktur ausgetauscht werden. Verkehr und Kommunikation stellen also ein Beispiel für eine fruchtbare Kombination von Domänen dar. Die Modellierung und Simulation (M&S) von Innovationen in solchen Bereichen kann sowohl während der Entwicklung als auch des Betriebs derselben hilfreich sein. Mit einer wachsenden Zahl an involvierten Subsystemen verschiedenster Art steigt potentiell jedoch auch die Komplexität jener M&S.

In dieser Arbeit begegnen wir den Herausforderungen, die die bestehenden M&S Ansätze offenbaren, indem wir eine neuartige datenzentrische Methodik entwickeln. Zunächst werden aktuelle Anforderungen an Simulation identifiziert. In diesem Kontext werden potentielle Schwachstellen von existierenden Methoden, Standards und Frameworks sichtbar. Basierend darauf besteht der Hauptteil der Arbeit im Design eines neuen Kopplungsansatzes, der vorhandene Lücken schließt. Dies geschieht indem eine Schichtenarchitektur entwickelt wird, die alle nötigen Bereiche abdeckt, um die vielfältigen Stakeholder von modernen Simulationsstudien zu unterstützen. In einer Infrastrukturschicht wird die Kommunikation zwischen Komponenten und die Speicherung von Simulationsdaten gelöst. Eine zweite Schicht ist vor dem Hintergrund von Wiederverwendbarkeit und Reproduzierbarkeit für die Kopplung verschiedenartiger Komponenten verantwortlich. Abschließend bietet eine Anwendungsschicht die Möglichkeit, die entwickelten Konzepte in Form eines M&S Dienstes zu nutzen.

In einem zweiten Teil wird der entwickelte Ansatz auf das Feld der Verkehrssimulation angewendet. Damit zeigen wir die Praxistauglichkeit unseres Ansatzes und demonstrieren die Fähigkeit, verteilte Simulationen, multi-level Simulationen und

domänenübergreifende Simulationen zu realisieren. Darüber hinaus präsentieren wir die Eignung unseres Ansatzes zur Integration von externen Daten in laufende Simulationen, zur Anreicherung von Daten durch Simulation und zur Wiederverwendung von bestehenden Simulationsdaten für neue Studien.

Contents

Abstract	iii
Kurzfassung	v
Contents	viii
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	3
1.3 Structure	4
2 Fundamentals and Related Work	5
2.1 Modeling and Simulation	7
2.2 Distributed Systems	12
2.3 Distributed Simulation	17
2.4 Data Centricism	33
2.5 Requirements Analysis	34
2.6 Research Gap	37
3 System Specification and Design	39
3.1 Requirements Specification	41
3.2 Data-Centric Architectural Concept	43
3.3 Communication Concept	47
3.4 Coupling Concept	52
3.5 MSaaS Concept	81
3.6 Summary	89
4 System Implementation and Evaluation	91
4.1 Apache Kafka and Avro	93
4.2 Implementation Details	94
4.3 Minimal Working Example	99

5	Application to the Traffic Domain	111
5.1	Modeling and Simulation of Traffic	113
5.2	Using the Approach in the Traffic Domain	125
5.3	Implementing Wrappers	137
5.4	Extension to a Further Domain	142
6	Exemplary Case Studies in the Traffic Domain	147
6.1	Study 1: Distributed Simulation	149
6.2	Study 2: Multi-level Simulation	155
6.3	Study 3: Data Enrichment	162
6.4	Study 4: Integration of External Data	167
6.5	Study 5: Cross-Domain Simulation	171
7	Conclusion and Future Directions	175
7.1	Summary and Conclusion	175
7.2	Limitations and Future Directions	178
A	General Definitions	181
B	Evaluation Resources	199
C	Definitions in the Traffic Domain	211
D	Definitions in the Communication Domain	227
E	Exemplary Case Studies	233
	List of Acronyms	249
	List of Algorithms	251
	List of Figures	253
	List of Listings	257
	List of Tables	259
	Bibliography	261
	Statement on Contribution with regard to Self-Citations	283

Chapter 1

Introduction

1.1 Motivation

Mobility is one of the core issues of our everyday lives. On average, we spend more than one hour per day on the road [160]. Hence, a reasonable mobility infrastructure is necessary for our lifestyle. Among education and work, this also includes the possibilities to go shopping, access health care, or do leisure activities. Another interesting aspect of mobility is the variety of stakeholders, apart from the users of mobility services, which reach from automotive manufacturers, future-mobility startups, and mobility service providers to environmentalists and municipal or federal decision-makers. Moreover, mobility is no isolated phenomenon. It affects (and is affected by) various other domains. Examples for these connections are manifold, for instance ecology (environmental impact of traffic), energy (e-mobility), or communication (autonomous driving). Especially the growing linkage of heterogeneous systems can be a driver for tomorrow's innovations.

In the last decades, methods from the field of Modeling and Simulation (M&S) have proven to be well suited for answering open questions. There are clear advantages of M&S when dealing with unknown scenarios. What-if-questions can be answered and investigated accurately in fully controllable and observable virtual testing grounds. Further reasons for simulative experiments arise from economical, ecological, temporal and practical motives, without having to expose someone to a safety risk. The more elaborated and cross-domain a problem and its aspired solution approach is, the more complicated might be the M&S of such scenarios. There are great opportunities nowadays to solve urging problems in an intelligent manner with new technologies. At the same time, however, these opportunities potentially introduce additional challenges because of their complexity.

In the context of simulation, these challenges consist among others of five aspects, which will be described shortly in the following. For one thing, simulation tools

and proven simulation models often do have a specific focus. It is not uncommon that this focus is narrowed down to an aspect within a single domain. Classical examples for this are traffic simulators or communication simulators. This implies that it may not be possible to model a cross-domain problem exclusively with a single tool. It might be necessary to model different parts in different tools and to connect the different models (i.e., couple the solvers) afterwards. Although the challenge regarding the coupling of simulators has been under study for a long time under the term co-simulation or parallel/distributed simulation, the realization of such co-simulations typically still requires a significant additional modeling and implementation effort.

The second aspect is about the feasibility of a simulation execution. On one hand, there are the requirements of a simulation setup towards the execution environment (e.g., required memory) and on the other hand, there are the requirements of a user towards the simulation execution (e.g., performance). It is quite conceivable that naive modeling and implementation approaches can lead to several problems. Results may not be available in time or may not be computable at all, because hardware limits would be exceeded (e.g., the main memory is not sufficient to store the simulation state). Works that are trying to tackle this problem are originating from the field of parallel/distributed simulation and as well from the field of hybrid-/multi-level simulation [134]. Again, approaches from both fields do not provide a generalizable and straight-forward solution for this aspect.

Besides the realization, there is a third aspect that addresses the issue of data. Even if the amount of available data sources is continuously growing, one cannot make the general assumption that all required data input for a certain model is accessible. Licensing modalities, privacy policies, protection of intellectual property, or technical reasons are exemplary causes for that lack. The more heterogeneous a co-simulation topology is, the more possibilities there are for a gap between accessible and required data. If one has to run a simulation model nevertheless, a method to bridge the data gap is required. Furthermore, the question of how to integrate data in a coupled simulation of such scenarios arises. One example use case could be the real-time integration of traffic counts from physical induction loops into a traffic simulation.

The fourth aspect covers the interaction of various involved stakeholders. Modern scenarios lead to new and heterogeneous partnerships. On the one hand, it must be noted here that the capabilities, the knowledge, and the aims of the different parties may differ fundamentally. On the other hand, the different partners are typically geographically dispersed, which may further complicate the collaboration. In addition, in the context of intellectual property protection it may be desirable that a certain access to a component or a simulation model is granted via an interface, while the artifact itself and its execution remains with the owner.

The fifth and last aspect is a logical consequence of the presented challenges. Without a verified and validated model, the results of a simulation study tend to be worthless. With an increasing complexity of the composed model, the required effort for this is also growing. The same happens to the endeavors of ensuring reproducibility and repeatability of results. Nevertheless, all of this is tremendously important in order to create trust in achieved results and with that enable the utilization of the generated knowledge for further decisions.

1.2 Objectives

The first two of the mentioned aspects are subject of research for decades [86]. The approach of coupling different independent models into a composed model is therefore no unknown challenge. However, there is a lack on structured methods that can be applied in practice on complex and unknown simulation scenarios (e.g., deciding which model elements can be shared between simulation instances at what conditions). Until now, only isolated sub-problems of this challenge are covered, although remarkable contributions have been achieved in form of international standards (HLA [66], FMI [145]) or well-known frameworks (Veins [206], mosaik [198]). The third and fourth aspect are less frequently subject of research. One reason for this might be that related problems are only now emerging as a result of recent opportunities. The fifth aspect has also been the target of heterogeneous research for a long time [127]. Without taking the strategies inferred from the prior aspects into account, there is no reasonable way of solving this issue. The validation challenge needs to consider the overall coupling context. The aim of this work is therefore to develop an approach that covers all five faces of the outlined problem and thus bridges the identified gaps of the current state of the art. In addition, the practicability of such an approach should be proven by implementing it in a simulation system.

Referring to a layered software architecture, this means that on lower layers the pure communication between various components has to be realized. Furthermore, exchanged messages must be delivered on time, reliable, and in the correct ordering in order to prevent causality breaks. In addition, a protocol has to be designed that defines in a generalized manner which messages need to be exchanged between which components on which occasions in order to realize a distributed simulation. Also, the question of how to translate information between different composed modeling approaches needs to be answered. On upper layers, it needs to be assured that domain experts and stakeholders can collaboratively set up, run, and evaluate such simulations without having a strong computer science background. In parallel, strategies are required addressing the verification and validation of composed models

and the persistent storage of all data that is required for reproducing results. It should be possible to execute single components on physically (and geographically) distributed computing nodes, further that the developed concepts are usable and extendable for unknown problems, and that external data sources can be integrated into a distributed simulation run.

As a main outcome of this work, a distributed simulation framework based on a new data-centric coupling approach is developed. Afterwards, the concept is applied to the field of traffic simulation. An extendable, prototypical catalog with components from the traffic domain (and one related domain) demonstrates the concept in a case study and acts as a proof of concept.

1.3 Structure

In a first part of this work, basics and related works in the fields of parallel/distributed simulation, hybrid simulation, and co-simulations are presented as a foundation in Chapter 2. Of interest are approaches that address the implementation, the modeling, and the usage of distributed systems and distributed simulations. Afterwards, the idea behind data-centric approaches is described and recent requirements towards simulation are collected. Based on the identified gap between current solutions and existing requirements, a data-centric approach is then used to design a generalized concept that bridges the gap in Chapter 3. Finally, details on the chosen technologies and implementations are given and an evaluation is done in Chapter 4. The main contribution of the first part is the development of a distributed simulation methodology that meets all identified requirements. In a second part, the developed concept will be applied to the traffic domain. Chapter 5 comprises the modeling of the data structures that are required for our developed approach for all four major modeling paradigms in the traffic domain. For each paradigm a representative simulation package is integrated into the framework using our methodology. In addition, translation rules between their modeling paradigms are modeled and implemented, and another domain is modeled and linked to the traffic domain. Several exemplary case studies follow, demonstrating the capabilities of our approach and showing potential use cases in Chapter 6. In Chapter 7, we summarize the intention and the main outcomes of the thesis and draw an outlook on future work.

Chapter 2

Fundamentals and Related Work

This chapter addresses fundamentals regarding simulation, distributed systems, and distributed simulation. Not only aspects of simulations but also of simulation modeling are covered. Apart from simulation, the basics in the context of distributed systems are presented: communication modes and architectures. Related work from the field of distributed simulation is then given. Then, data-focused architectures and trends are described before a requirements analysis is conducted. Finally, the gap between existing solutions and identified requirements is drawn. Parts of this chapter are based on previously published works in [95], [96], [97], [98], [99], and [100].

2.1	Modeling and Simulation	7
2.1.1	Modeling Paradigms	8
2.1.2	Confidence and Reliability	9
2.1.3	Input Data	11
2.2	Distributed Systems	12
2.3	Distributed Simulation	17
2.3.1	Modeling of Distributed Simulation	19
2.3.2	Synchronization Mechanisms	20
2.3.3	Distributed Simulation Standards	21
2.3.4	Implementations of Distributed Simulation	26
2.3.5	Simulation as a Service	31
2.4	Data Centricism	33
2.5	Requirements Analysis	34
2.6	Research Gap	37

2.1 Modeling and Simulation

First, a short overview about M&S in general and related terminology will be given. A model represents the attempt of a simplified description or representation of a system. The purpose of a model is to study with its help the behavior of the represented system. Creating a model is called modeling and requires to make reasonable assumptions about entities, properties and relations within the system under observation. These assumptions are usually formulated with mathematical formulas or logical expressions. [132]

The use of models is not necessarily restricted to the field of simulation. Other applications can for example be found in the field of statistics. “All models are wrong, but some are useful” [36], is a famous quote attributed to George Box. It illustrates the obvious advantages and disadvantages of a model nicely. Models are per definition no complete copies of the reality. They simplify reality, while aiming to contain as much relevant information as needed and possible for a certain purpose. However, it is only through the information reduction that the investigation of this simplified reality becomes feasible by simulation.

Despite their simplified nature, models of real systems tend to be so complex that it is not reasonable to use analytical methods in order to perform investigations. In this case, numerical methods might be better suited and thus also the method of simulation. Based on input data and the simulation model, data points are calculated, which can then be used to assess a certain behavior. [132]

Therefore, simulation is a tool that uses models to help investigating and understanding systems. The benefits of simulation are manifold. On one hand, there are economic advantages that allow for cost and time savings. Furthermore, ecological reasons can be stated as well as the potential to avoid risks to the human health when executing experiments. In addition, it might not be possible to realize physical experiments using the currently existing state of the art. Similarly, observations and measurements of relevant properties of real entities might be hard to impossible, while the conditions of the experimental environment might not be as controllable as required. It is likely that the latter circumstance prevents having the possibility of performing reasonable repetitions with a set of fixed parameters. Besides the economic perspective, the advantage of time savings, respectively the acceleration of experimentation execution, can be very important in practical terms, for example because deadlines are missed or the experiment’s run time would exceed a researcher’s lifetime. Of course, there are also disadvantages for M&S. One major drawback is that the quality of the results depends largely on the model’s quality. Also, the quality of the input data highly affects the outcome.

2.1.1 Modeling Paradigms

As stated before, the very nature of a model lies in the simplification of reality. Throughout the years, various modeling paradigms have been developed. In order to support the understanding of the following contents (e.g., multi-level simulation), a brief overview of important paradigms is given. Various works used different taxonomies for categorizing existing approaches. One well-known differentiation uses Agent-Based Simulation (ABS), Discrete Event Simulation (DES) and System Dynamics (SD) as the main modeling types [38].

In a DES, the system behavior is modeled by a state, which can change abruptly while proceeding from one discrete (simulation) point of time to another [132]. Such points of time are called events. The paradigm is suitable primarily if the system to be modeled is also characterized by discrete events. If a waiting queue is studied for example, it could be interesting to observe when a person joins the queue, when the person is served, and when the person has finished being served. As a special case of DES, the step length between the single events may be fixed. This is suitable, if the system under observation can be described with continuous relations (for example, the movement of vehicles), but only a discretized variant seems practicable for implementation. Accordingly, the DES is not only appropriate for the investigation of discrete systems. Regarding the implementation, there is typically a global event list. Each event has a (simulation) timestamp. The event list is sorted by timestamps and is processed in sequence. The first element of the list is processed and removed. The current simulation time is set to the event's timestamp. During the processing of the event, new events can be created, which are then also sorted into the event list. Usually, the timestamp of a new event must be greater than that of the current event, or at least equal to it. It is also possible to remove events from the event list without processing them. The simulation is finished when the event list is empty.

Agent-Based Modeling (ABM) is a more recent approach, which can also be realized using DES. The idea is that the overall system behavior does not have to be modeled explicitly. The system behavior results implicitly from interacting autonomous actors, the agents. This might be the only way the modeling of a complex system becomes feasible, for instance for scenarios where the mechanics of the overall system are intangible [138]. The overall model thus emerges in a decentralized bottom-up approach. An agent is an entity independent of other agents, which can perceive its environment together with other agents and make independent decisions. These decisions are based on the internal state of the agent and (adaptable) rules. Suitable applications are for example economic models or epidemic models.

Continuous models can be built up with the help of differential equations. While the numerical solution of ordinary differential equations (derivative with respect to one variable) is generally not a special challenge, it is more difficult with partial differential equations. SD is an approach to represent continuous relationships between stocks and flows. Internally, this is also realized with a system of ordinary differential equations and polynomial equations [32]. Due to its intuitive modeling, it is very popular and is used for example for behavioral models of a society. In general, SD models represent a more aggregate view on a system than ABM models do.

Hybrid models represent the combination of discrete and continuous models [38]. The hybrid simulation does not necessarily have to be carried out in the sense of a co-simulation. Sequential execution of the different models is also conceivable. Besides the fact that the combination of different paradigms already requires a certain conceptual effort, one of the challenges of a distributed execution is the identification of suitable communication points. A common strategy is to sample the state of the continuous models at the discrete event times of the discrete submodels (or a multiplicity of them). It is also conceivable to communicate every time when state variables of the model exceed or fall below certain threshold values.

2.1.2 Confidence and Reliability

Generated results might, of course, be used to better understand the system under study. Therefore, the results should be accurate to some degree. However, some voices express concern that simulation is in a crisis of confidence. In this light, simulation results can no longer be considered credible or meaningful per se [218].

Pawlikowski et al. note that many simulation studies from the field of communication networks do not establish trust [171]. The authors introduce the notion of a credibility crisis. In general, more detailed information about an experiment would be needed to identify errors in works. Dalle criticizes that exactly this information is usually not provided, which in turn is at the expense of reproducibility [67]. This is not only because information would be available but is not published by authors by choice. Another problem is that there may be hidden parameters that even the authors do not have access to (e.g., embedded in a commercial tool). Accordingly, Yilmaz et al. mention that there is growing doubt about simulation results (a credibility gap) and therefore the reproducibility of studies should definitely be aimed for [230]. It is also problematic if the origin of results cannot be traced. This can be tracked back to the problem of traceability and reproducibility of experiments. Ruscheinski et al. deal with this problem of the origin (provenance) of simulation results [191]. The authors believe that the type of modeling plays a major role in this and therefore propose a provenance model. This model should allow the

traceability of all relationships and information flows leading to a study. Taylor et al. also discuss the relevance of reproducibility and add that there are by all means also critical research fields (e.g., military or medicine) in which it is not possible to grant access to all data and techniques [212]. The same can be applied to many industrial research fields, where the protection of intellectual property is a high priority.

Closely related to reproducibility is the repeatability or replicability of simulation studies. The big difference is that repeatability is given if a repeated simulation run leads to identical results. There is some room for interpretation regarding the meaning of the result. Replicability, on the other hand, also requires that all internal states correspond to the original at all times [154]. For this, it is inevitable to guarantee deterministic co-simulations even for stochastic subparts. While for standalone simulations the use of pseudo-random number generators with known seeds contributes to this, in the coupled case the correct delivery of messages is additionally relevant.

Credibility can be strengthened with solid verification and validation. Verification can show whether and to what extent a concrete implementation correctly represents a conceptual model. It thus describes the quality of the implementation. Validation, on the other hand, makes a statement about how well the implementation and the model represent a section of reality. For this purpose, it is common to compare simulation results with real data and to determine the deviation [154].

In addition to potential benefits in terms of credibility, component reusability can save money and time. This is well known in classical software development as well as in other engineering disciplines. For example, the ideas of the object-oriented programming can be used, in particular the encapsulation of related functionality into enclosed blocks [144]. Balci et al. generally see model reusability as a difficult task, since a model is typically developed for a specific purpose [19]. Nevertheless, they see some scientific communities (e.g., electrical engineers) successfully pursuing model reuse. Accordingly, even though there is research in the field of model and simulation component reusability (e.g., [176, 194, 236]), the topic of reusability of translation logic between submodels has been unnoticed so far. This would be especially relevant for the case of hybrid or multi-level simulation.

Trust could be regained if publicly available simulation tools were used and at the same time comprehensible models were built and published together with all the parameters used. This would allow the external reproduction of the simulation results. For many works this is currently not the case. One reason for this might be the additional effort for the researcher. However, this does not necessarily need to be done manually. If using a simulation service, the platform could support the researcher by providing an automated description of an experiment.

If we assume that we would have established a good level of trust regarding traditional simulations, the question arises whether the efforts are also sufficient

for the credibility of distributed simulation. Even if there are reusable submodels and simulators, a critical aspect remains still unnoticed: the information transition between coupled submodels. Logic that translates incoming data streams into pieces of information understandable by the submodel may be scattered all over the codebase of the subcomponent and, possibly, the emitting component. The transformation(s) of a piece of information may be difficult or even impossible to trace and reproduce.

2.1.3 Input Data

A major problem arises, if necessary data is either not available or can only be acquired with considerable effort or with cutbacks [50]. Data is used for various purposes in the M&S life-cycle, for instance to build the model, for subsequent validation, or as input data for a simulation of the model. Potential data sources can be, for example, historical data collections, surveys, results of other (simulation) studies, or live data streams. Aggregated statistics can be as helpful as the recording of individual behavior. If the required data can be obtained, it may be necessary to filter it due to a poor data quality. In the worst case, this can lead to the insight that an entire data set is unusable as the quality of the whole simulation study depends on it [204].

Collected data can be used to calibrate a simulation model in order to reflect a certain situation (e.g., creating a digital twin). In this process, model parameters or inputs are adjusted with the aim of generating a congruent image of reality. A purpose for an accurate live representation of a real system could for example be the evaluation of time-critical what-if decisions (e.g., road traffic management). Calibration can be done in two ways. There is an offline approach, where the simulation is iteratively repeated with varied parameters until an acceptance level is reached. This is not always suitable, for example for synchronizing a digital twin of a live system. In that case, an online approach is necessary that adaptively adjusts the state of a running simulation to observed reference values.

An example to illustrate this is a digital twin of a road network. The simulation should run synchronized to the wall-clock time and represent the real traffic situation as accurate as possible at any time. Aggregated count data from induction loops are available as reference values of the real system. By means of a microscopic traffic simulation, it should be assessed what adjustments of the traffic light switching times would increase the traffic flow. An online calibration for such a setup would aim to schedule the creation of new simulated vehicles in a way that the error between real measured count data and corresponding simulated count data is as small as possible.

The example represents another typical data issue. It is possible that data is available in acceptable quantity and quality. At the same time, the data might be at a wrong level of detail or in a wrong format. For example, if the positions, speeds, and planned routes of all vehicles in the real system were available, the calibration of the digital twin could be simpler. Fortunately, one of the strengths of simulation is the enrichment of existing data sets. For instance, it is possible to enrich the aggregated input data with detailed simulation information and thus create added value. In the given example, the trajectories of the simulated vehicles of the digital twin could be exported for further processing, whereas previously only the aggregated count data was available.

2.2 Distributed Systems

A Distributed Simulation (DS) is implemented by a distributed system. This section will therefore provide some basic information about distributed systems in general. Steen and Tanenbaum [219] define them as collections of independently functioning programs that appear to the user as a single coherent system. Typically, the individual components are physically or even geographically distributed. By the distribution the usual challenges regarding concurrency arise, which are also playing an important role when designing programs for multi-processor architectures in general. Contrarily, in a distributed system, the interconnection of the individual components cannot be considered robust to the same extent. In addition, each involved device can fail separately and potentially cause a system failure, if there are no countermeasures. Also, a common global clock is no longer existing in the comparison to a multi-core system. The overall complexity is therefore increased compared to classical systems.

However, these limitations come also with a number of advantages. Existing computing resources can be utilized more efficiently and also the performance of the overall system might be improved. Hardware limits such as the maximum main memory available in a device can be overcome. With an appropriate system design, scaling strategies can allow almost limitless growth of the system. The robustness and fault tolerance of the overall system can be increased, for example, by the redundant design of components or dynamic strategies for intercepting faulty components.

Architectures

If different system architectures of distributed systems are considered, centralized architecture approaches are to be distinguished from decentralized approaches. In addition, there are hybrid forms of the two architecture principles. The client-server paradigm represents the most common centralized variant. Due to the two components involved, it is also called a 2-tier model. Of course, this communication

pattern can be extended by additional layers (see Figure 2.1). Typically, each layer represents thereby a set of logical tasks that are belonging together. A major advantage of a layered architecture is the possibility to develop, maintain and execute the layers independently of each other. This type of distribution of the overall system is described as vertical distribution. [219]

In contrast, the core idea of decentralized systems is horizontal distribution. In this case, there are several classic client or server components. Each component might only solve a subproblem and thus the total load is distributed. Peer-to-peer systems represent a possible implementation, in which the individual components are coupled via direct connections. A component usually acts as a client and a server at the same time. It is generally not possible for each component to have a direct connection to every other component. Instead, it may be necessary to communicate via other nodes in the existing network. Network topologies can be divided into structured and unstructured variants. Trees, rings, and cubes represent examples of structured topologies. Techniques such as Distributed Hash Tables provide an efficient organization method for data in such a network. On the contrary, in an unstructured network it is common that each node maintains a list with current neighbors. As a result, there might be no deterministic routes for exchanging data between two components. [219]

Microservices represent a special case of a distributed system. The core idea is to divide a complex system into small independently functioning programs. Ideally, each of these programs is responsible for only a single task. Typically, different microservices are only loosely coupled and connected via a network using REST, for example. The advantage of such an architecture is a fast development in small teams. Developers can create independent microservices and already roll them out without having to wait for the overall product. With regard to scalability, similar advantages are offered. If the interest in a certain service would grow, this specific

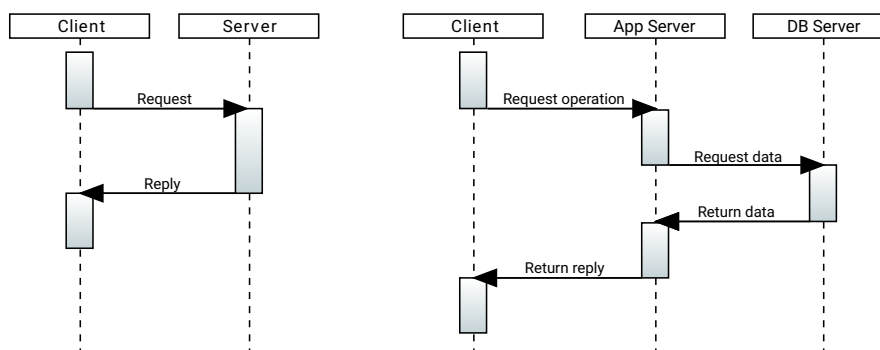


Figure 2.1 – Different layer architectures. Based on [219].

service could be scaled [105]. In addition, the development of each microservice can take place in the technology stack that is best suited to fulfill the intended task. The overall system plays only a subordinate role here. For example, it is therefore not uncommon for different interacting services to be implemented in different programming languages. As for distributed systems, the disadvantages include an increased complexity of the overall product, additional pitfalls for smooth operation, and security risks [227, 228]. The network communication in particular can have a massive impact (delay, jitter, throughput) on the successful operation of the product. Unfortunately, as simple as the “one service per task” rule may seem at first, it is generally not so easy to define what exactly a single task comprises. The more finely granular the partitioning, the more independent services are required and accordingly, it can be expected that the communication effort will increase. The more subtasks a service is allowed to cover, the more likely it is to move back towards a monolithic architecture.

Isolated services also offer the advantage that the product can be seamlessly extended by further services/functionalities. Individual services can be tested, reused and replaced in isolation. The authors in [105] provide an insight into the microservice architecture of one of Europe’s largest online trading platforms. In addition to the advantages already mentioned, they mention agility: 500 new components are rolled out per week during live operation of the system. The architecture has become indispensable for many typical online offerings. But the approach is also used, for example, in analytic pipelines [124], smart cities [130, 137], smart buildings [125], or digital factories [57]. Boucher et al. [35] think that the architecture concept of microservices is often misinterpreted in practice. There is only a very coarse-grained partitioning of tasks and many services are therefore generally too large and thus not microservices in the original sense. Regarding the field of simulation and/or modeling, there is also research ongoing. Alpers et al. [6] propose a toolkit in the context of Business Process Management (BPM). Here, individual tasks are encapsulated in microservices (e.g., the user interface or the simulation core).

Communication

The fundamental difference of non-distributed and distributed systems is that a functioning communication between sub-processes, such as via shared memory or local inter-process communication, can no longer be assumed without further ado. Additional efforts must be taken to ensure this. In general, communication in distributed systems is often based on Remote Procedure Calls (RPCs) or the exchange of messages via Message-oriented Middlewares (MoMs) that are using a publish/subscribe paradigm.

For pure client-server models, RPCs are well suited to realize the communication. However, if the components of a system do not take clearly and exclusively the role of either the client or the server, then a message-based realization of communication might be more appropriate. Depending on whether the exchanged messages are stored by the communication middleware or not, a distinction between transient and persistent communication can be made. The storing can take place until the recipient has successfully processed the message or arbitrarily longer. In addition, communication can be implemented synchronously or asynchronously. In the asynchronous case, the sender can continue with its program logic directly after the transmission process. In the synchronous case, the sender blocks until a successful delivery. A success can be defined in different ways, for example until the middleware confirms the reception of the message. RPCs represent a more direct approach to communication (at least from an application point of view), since received messages need not to be interpreted by the application layer. The sender directly calls an executable program part at the receiver and receives the return value after some time. Thus, it might seem like a local function call. [219]

Accordingly, the characteristics of the communication channel can significantly influence the performance of the overall system. If there is a high delay, for example, the performance of the overall system might appear noticeably slow.

Cloud-based Services

While cloud computing gained momentum in the public perception just in the last decade, according to Weinhardt et al. it goes back to a much older concept: grid computing. The motivation for grid computing clearly grew out of a scientific context. The challenge was to solve the problem of limited computing and storage resources by connecting infrastructures across institutes. In contrast to cluster computing, the connections and their organization are decentralized. In comparison to grid computing, virtualization is a crucial component of cloud computing in order to be able to react dynamically to the current resource requirements of a system. This is accompanied by the need for elaborate capabilities to monitor system health and the availability of a central control plane. This also allows assuring Service Level Agreements (SLAs) and thus including them in business models. Due to their dynamic character, cloud systems are also suitable for interactive applications, which should be usable as easily as possible (e.g., via web interfaces). [225]

Cloud-based services are commonly divided into three categories depending on the degree of abstraction: Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Service. Infrastructure services can be further divided into the provision of storage space and the provision of computing power. An example of this would be virtual machine access. While platform services include a more specialized

offering, such as pre-installed and configured frameworks and libraries, software services provide direct access to the desired software product. [115]

For the user, an interesting point about cloud services is that a single access device might be sufficient to use a variety of services. Possibly, it is not necessary to install any additional software locally, which offers various advantages. For example, there is no need for time-consuming configuration and maintenance. A shared cloud system has better possibilities for increasing resource utilization. The user might be only charged for resources that are actually used. As a result, everyday life can be made easier for experienced users. Also new user groups, which might have been unable to overcome the existing hurdles, can be attracted. However, there are also disadvantages. These include a certain loss of control and skill of the user, the dangers of service provider monopolies being formed, and can extend to ethical and legal issues concerning data storage, data protection, and confidentiality.

2.3 Distributed Simulation

The DS represents a special case of simulation. In this section, an overview of the motivation for DS, the resulting variants due to different motivations, and possible implementations will be provided. In addition to theoretical work, application-specific implementations will also be presented.

There are a few basic terms in the context of DS that are explained briefly. A Logical Process (LP) describes a self-contained unit that can be processed by a processor. Load balancing means the allocation of resources taking into account current and future/historical workload. Examples consist in the allocation of tasks to processors. A logical time describes a discrete counter which runs monotonously increasing completely detached from the real time. (Time) synchronization considers the coupling of logical clocks in order to prevent causality violations. Causality violations may occur, for example, when messages are delivered too late and/or in the wrong order.

Fujimoto [85] mainly distinguishes between two different aspects of DS. Putting the parallel aspect of DS in focus, he sees its advantages in the increased execution speed. As usual for parallelized processes, a speed-up by a factor corresponding to the number of used compute nodes can be achieved in the ideal case. However, the real speed-up factor is usually lower. The way the overall problem is partitioned into isolated subprocesses and their communication behavior among each other plays a crucial role here. The increase in performance can be of particular importance if complex simulations are carried out. Simulating typical real-world experiments can easily take several days, weeks, or even months. In such cases, boosting performance via parallelization can be an enormous help.

At the other end of the time horizon (e.g., considering seconds or milliseconds) a timely calculation of (intermediate) results can also be a valid motivation for increasing the speed. If a simulation interacts with real components, the simulation usually has to run synchronously to real time. This is because the execution speed of an electronic control unit, for example, typically cannot be modified and input values are expected at fixed points in time. In order to make that happen, the simulation of a certain time step is often desired to run significantly faster than the corresponding time span in real time so there is enough backup in case a calculation takes longer than expected. Before calculating the next simulation step, the simulation is paused for the rest of the backup time. Consequently, the simulation appears to run synchronous to the real-world component.

As a second aspect of the parallel nature, Fujimoto points to the ability to run larger simulations than it would be possible on a single compute node. Leaving the execution speed aside, the limiting factor for this is memory. A problem could be that there is no free space on the disk (e.g., storing resources or swapping)

or that the main memory reaches its limits and is not able to hold the necessary program state. In addition to the reasons arising from parallelizability, Fujimoto mentions the possibility of integrating several heterogeneous simulators into a composite simulation. This allows reusing existing tools and combining them with other tools to generate additional value. Furthermore, such combinations free from the limitation of having to model a system exclusively with one tool or with one paradigm and enable to individually model subsystems in the most appropriate way. The resulting possibility to connect geographically distributed components provides further benefits. Collaborating workgroups, departments, companies or institutions do not have to come together physically in order to run a joint simulation.

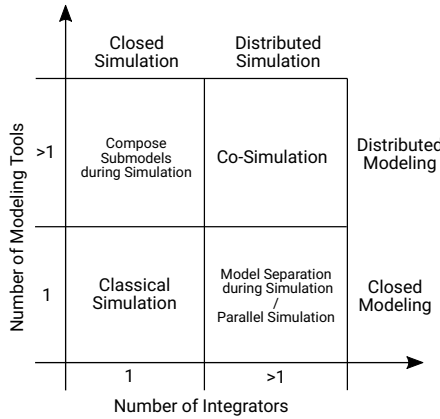


Figure 2.2 – Simulation type taxonomy. Based on [87].

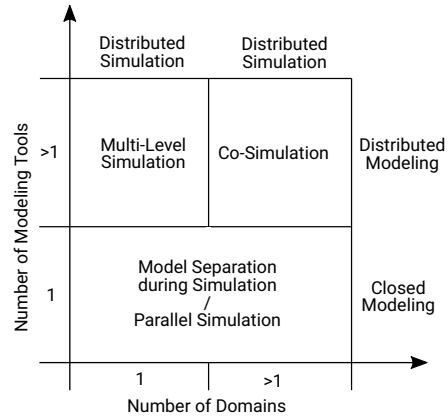


Figure 2.3 – Extended simulation type taxonomy for distributed simulations.

According to Fujimoto, the term DS was previously used primarily to describe geographically distributed simulations. In contrast, the term parallel simulation was used when a DS was executed on a multi-core system. However, since this distinction between the different computer architectures is becoming increasingly blurred in the context of clouds, grids, and clusters, he no longer sees this definition as useful and refers to all coupled simulation variants as DS [85]. Geimer et al. [87] follow this definition and present a taxonomy of different types of simulations as illustrated in Figure 2.2. Accordingly, we propose an extension of the taxonomy by the dimension of the number of modeled domains for the case of DS (i.e., number of integrators greater than one, see Figure 2.3). We will refer to this extension in the following. We also distinguish between Closed Modeling and Distributed Modeling, and further divide DS into co-simulation, multi-level simulation and parallel simulation. Relevant for the classification as a multi-level simulation is the number of domains modeled. This distinction in our terminology is worthwhile in order to be able to accurately describe and solve the challenges of the specific types.

2.3.1 Modeling of Distributed Simulation

The modeling of coupled simulations mainly considers the question on which tools or models to couple and how to partition LPs for them. Different perspectives may impact these strategic decisions. The decisions may be purely based on domain expert knowledge, utilize ontological approaches in a structured way, or simply follow practical reasons. The latter may include the protection of intellectual property by executing a certain component in a secured environment or meeting performance requirements regarding the simulation execution. There are also standardized processes for designing, executing, and analyzing distributed simulations, for instance, the *Federation Development and Execution Process* (FEDEP) or the more general *Distributed Simulation Engineering and Execution Process* (DSEEP) [215]. The standardized steps include the definition of overall environment objectives, a requirements analysis and system implementation, and the evaluation of results.

Ontological approaches allow describing the properties and relationships of entities within a domain. Accordingly, to establish an ontology means to investigate and represent the internal relationships of a concept. Benjamin et al. note that one major problem when designing an ontology is to find the right level of detail [23]. There is a danger of getting lost in the details or overlooking relevant points, as it is for the modeling of simulation models. Yilmaz [229] proposes to evaluate the conceptual congruence of simulation models by comparing a common metamodel with an associated ontology. Ontologies are also used in practice. For instance, Sarli et al. use an ontology in the supply chain research field to semi-automatically derive High Level Architecture (HLA) data models [193].

Once the question regarding which components are to be coupled is clarified, there may be another challenge. If the DS is a parallelized simulation by coupling several instances of the same tool, there is little room for interpretation regarding the required information exchange. However, if different submodels are coupled, the information exchange between the models might not be straight-forward but requires modeling decisions itself. Gomez et al. [78] pick up the previously published idea [110] of dividing that problem in a syntactic challenge and semantic challenge. The former covers the technical aspects of enabling the communication between different components. Solving this challenge can be supported by relying on existing standards, such as Functional Mock-up Interface (FMI). The latter considers the process of finding a common meaning for exchanged information. For tackling this challenge, the authors require an explicit semantic modeling, for instance, based on expert knowledge.

When considering component reuse or collaboration, the splitting of models into several submodels and the design of related interfaces is a relevant aspect in the field of M&S. Earlier works on this topic go back to the 1980s [157], but research is

still ongoing, for example, in the field of model building. Moradi et al. address the problem of an increasing complexity of simulation models [152]. As a solution, they propose to use small submodels that can then be composed again. In this context, a Base Object Model (BOM) encapsulates the data models of the HLA standard and includes other meta-information such as a conceptual description of the model. With the help of the metadata, the composability of overall models should be simplified. In addition to works that focus on the HLA standard [53, 183, 232], there are other more generalized approaches. The extensive ideas of object-orientation also have an impact on simulation (object-oriented simulation) and also on modeling (object-oriented modeling). By encapsulation, individual behavior is modeled, while the interaction of individual instances creates a global system behavior, which can then be analyzed [121, 156]. This is closely related to agent-based modeling. Component-based simulation differs from object-oriented simulation mainly by the absence of the concept of inheritance. An extended functionality is obtained therefore preferably by the coupling of different components [43]. Boer and Verbraeck note that it is generally impossible to combine arbitrary commercial simulation models if no adjustments can be made in the models [28]. Moreover, Verbraeck distinguishes between software components and simulation building blocks [221]. He describes building blocks as: self-contained, interoperable, reusable, replaceable, encapsulating their internals, providing useful services through precisely defined interfaces, and customizable to match requirements arising from their environment. In the next chapter we will adapt this concept.

Diallo et al. consider the validity of composed models. They introduce a M&S formalism based on model theory and interoperability [71]. A model is said to be interoperable with a reference model if it is a valid model for that reference model. A reference model represents the world view of the modeler. Tolk et al. [214] thereby caution that hiding parts of a model can lead to a conceptual mismatch, a lack of consistency, and different truths within connected models whose models overlap. On the contrary, we frequently notice the hiding of components in practice, either by choice or due to practical reasons. For example, physical devices that are used in a Hardware-in-the-Loop (HiL) simulation [18] are typically black boxes. In any case, we consider the issue of validity of composed models as a challenge that has to be addressed. Therefore, we will pick up on this in the next chapter.

2.3.2 Synchronization Mechanisms

One of the main challenges of a DS is avoiding causality violations. Basically, the purpose of synchronization algorithms is being able to obtain the same results from a DS as from the sequential execution of the same simulation on only one computational node. To achieve this, it is necessary to ensure the correct and

deterministic execution of a DS. This translates mainly in having the possibility of communicating in a way, where messages arrive on time and in a deterministic order. Bononi et al. [30] proclaim the necessity of a synchronization mechanism in DS with the following statement:

A simulation process is a process that incarnates the ordered events' execution, and the behaviors of at least one model entity. A synchronization mechanism is needed to order the events' executions, by following their causal order. The causal order can be informally defined as the notion of 'happens before' ordering of events. The real system evolution and the interactions between system' entities is represented by the model state variables' updates. The causal order of events can be obtained through i) a totally ordered event list in monolithic simulators, and ii) a distributed synchronization mechanism implemented through message passing communication of event notifications, in parallel and distributed simulators. It results a great importance of the communication efficiency in distributed simulation scenarios.

In practice, an error classically originates from a message that was received too late, either because the network latency was too high or the sending component was updating its simulation state slower than the receiving component. For instance, if we have DES, the result of such a situation would be that the event list is not complete or that its order is not correct. We distinguish between conservative and optimistic synchronization approaches. Conservative approaches permit the subcomponents to continue only when it is guaranteed that no errors can occur. Until then, the simulation execution of subcomponents that want to proceed is blocked. Chandy, Misra, and Bryant worked out the first and best-known solution for such a conservative approach [40, 54].

An optimistic alternative was later presented by Jefferson et al. [120] in the form of the Time Warp algorithm. As *optimistic* suggests, components are allowed to proceed before the proceeding is guaranteed to be error-free. However, in the event of an error, they must be able to detect it and return to an earlier, error-free simulation state [85].

2.3.3 Distributed Simulation Standards

Regarding the implementation of a DS, the question of generally accepted standards arises. In fact, in recent years, two international standards have been developed that cover parts of the DS: High Level Architecture (HLA) and Functional Mock-up Interface (FMI). Both will be presented in more detail in the following.

High Level Architecture

The HLA can be ascribed to the middleware-based approaches of the DS. The US Department of Defense initiated the development of HLA in the early 1990s. As a result, HLA was released in version 1.3 in 1998. By the end of the century, the project was handed over to the IEEE, which then resulted in the first international standard of the HLA rulebook (*IEEE 1516-2000*) in 2000. Ten years later, HLA Evolved (*IEEE 1516-2010*) was published and is still the current version. However, a successor is being worked on. An important aspect of HLA is that the standard solely defines a set of services that must be provided to realize DS. The underlying communication protocol is left to the implementation. This is a crucial point with respect to interoperability. As an advantage of this design decision, one can argue that this freedom leaves room for drastic performance optimizations - at least in theory and if certain framework conditions for a defined target application are known in advance. Not only the specification for the lower layers is missing. In addition, there is no complete reference implementation. Therefore, there is not *the* HLA software. Fortunately, there are several projects that implement core functionalities in what is called Runtime Infrastructure (RTI) software. They provide stubs that can be integrated and adapted into the existing simulation submodels. Obviously, the performance of the middleware implementation is crucial for the performance of the entire distributed simulation setup. The HLA standard also grants implementation freedom in the question of what information is exchanged between which components and when. These decisions are left to the Data Distribution Management (DDM). As a result, the performance of different implementing RTIs can vary widely. While one implementation may perform well in a simple use case because of its DDM approach, another implementation may perform better in terms of scalability if the scenario involves more components [93]. Actual working interoperability between different RTI implementations is highly questionable for similar reasons.

There are two important terms in the standard: *federate* and *federation*. A single component participating in a DS is called a federate. The federate's interface to the RTI is implemented via what is known as the *federate ambassador*. A federate does not necessarily have to be a simulator. The set of all federates that form the DS is called a federation. A federation exists for a specific purpose. All federates of a federation refer to the same Federation Object Model (FOM), which represents a common data model. The running federation with federates connected by an RTI is called a federation execution.

HLA is known for interoperability, model reuse, and composability of submodels [237]. Therefore, it represents a framework that can be used by developers to organize and define simulation applications. The standard identifies two main goals that lead to flexibility. The first is to create interoperability between different

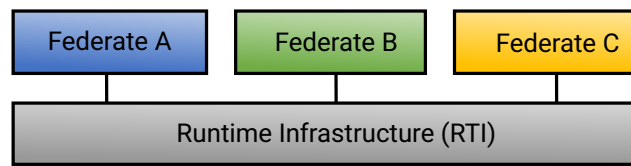


Figure 2.4 – HLA architecture.

simulations and the second is to support the reuse of models in different domains. There are ten rules that define responsibilities for federations and federates. This is to ensure the correct interaction of the different participants. There is also the Object Model Template (OMT), which defines the structure of data models used to specify a simulation. Finally, there is the federate interface specification: a set of rules for federate interfaces that ensures that submodels can exchange information. [118]

Three specific models are based on the OMT: the Simulation Object Model (SOM), the FOM, and Management Object Model (MOM). The SOM refers to a single federate and provides information about the federate's interfaces, i.e., the possible data structures that a federate can publish or consume. The FOM defines all information types that can be exchanged during the execution of a federation (e.g., objects and interactions with their parameters). Of course, it is not essential to use all the capabilities provided by different SOMs in a FOM. The MOM is used for information outside the simulation-related payload (e.g., for control and monitoring tasks).

The federates may participate in the time synchronization process, but are not forced to do so. The participation is divided into two different types. A federate can be time regulating (i.e., the federate has influence over time progress and can send Timestamp Order (TSO) messages) and/or be time restricted (i.e., the federate can proceed only when permission is granted and it can receive TSO messages). The mode is not static, but can be changed during an execution run. When a federate becomes time-regulating, a lookahead value must be defined. TSO messages are processed in a TSO buffer, otherwise there is a Receive Order (RO) buffer for RO messages. The lookahead value describes a lower bound for sent TSO messages. This means that a time-regulating federate that has a lookahead of LA and is currently at time T cannot send TSO messages with a timestamp less than $T + LA$.

A time advance can be requested using four different request types: Time Advance Request (TAR), Time Advance Request Available (TARA), Next Message Request (NMR), Next Message Request Available (NMRA), Flush Queue Request (FQR). When a Time Advance Grant (TAG) is received, a federation can continue until the requested simulation time. During the period between a logical time advance request and the corresponding grant, the connected federate is in the time advance state. In

addition to message ordering modes, there are two different transport modes: best effort and reliable transport.

Regarding interoperability between *HLA 1.3* and *HLA 1516-2000*, some differences are mentioned in [150]. With respect to the FOM, standard data types have been introduced, and the object model is in XML format and has been Unicode encoded since *HLA 1516-2000*. The FOM file is used directly by the RTI, no additional FED file is needed anymore. In contrast to these cosmetic changes, there are actual changes in the behavior of some functions. For example, the authors mention that in the *1.3* standard, existing subscriptions to attributes of an object are lost when a federate subscribes to an additional attribute. In *1516-2000*, there is no automatic replacement, but an appending of subscriptions. The ten golden rules that define responsibilities for federations and federates have not been changed.

Accordingly, the HLA standard provides a framework that allows arbitrary (simulation) components to be coupled together. This includes above all the transmission of messages from a predefined catalog, which is derived from the FOM, and the possibility for logical synchronization of the components involved. Critical voices mainly refer to the missing wire protocol. For a standard whose primary goal is interoperability, its absence is a massive limitation. On the other hand, the standard is very comprehensive and covers many aspects that might not even be necessary in many civil society applications. For one actual purpose of the HLA, to make it possible to implement huge military simulations, the complexity is of course understandable. However, the complexity can lead to serious performance degradation [30] and could be one reason why there are no mature and complete open source implementations of the standard.

Furthermore, there is no standardized way to control which federates are allowed to join which federation. While this was understandable in a local deployment in the past, nowadays the common setups have changed and communication over WAN and between different stakeholders has to be considered. Currently, a new version of the standard is being developed. Among other things, this should solve the problem of missing authentication and authorization. It is also conceivable in a new version that federates can prove via certificates that they are compliant with certain agreements before they can connect to the RTI or join a federation [149]. Other ongoing work considers the strong bond between the RTI and the simulator. The simulator connects to the RTI using the (arbitrarily complex) client libraries of the respective RTI implementation. Changing the RTI means changing the client library and adapting the simulator wrapper. Therefore, there are efforts to develop a non-blocking federate protocol to be able to reuse client implementations detached from the RTI [147]. In addition, the modular nature of FOM modeling should be further strengthened. While it is currently possible for an FOM to consist of several

parts that can also be used independently, it is still not possible to extend used FOMs in the classic sense (e.g., add an attribute) [148].

Functional Mock-up Interface

In contrast to HLA, the work on FMI was not initiated by an authority but by Daimler AG as part of the MODELISAR project. FMI is a considerably more recent standard as well. Version 1.0 of the specification was published in 2010. Since the end of the project, FMI has been further developed as one of the *Modelica Association Projects* with the help of a broad-based consortium.

The specification consists of two parts, whose goals are to support model exchange (*FMI for Model Exchange*) between tools and to enable co-simulation (*FMI for Co-Simulation*). The basic approach behind the first goal is that modeling tools generate executable C code from the model. The code can then be reused in other environments either directly or in binary form. Models are described by equations and can be executed on heterogeneous systems without further external dependencies, since any referenced code is distributed together with the model core as a Functional Mock-up Unit (FMU).

For the goal of co-simulation, the focus is on the interface. Data exchange between coupled subsystems is limited to discrete communication points. Between two communication points, the subsystems independently compute their next state. A so-called master algorithm controls the data exchange between the subsystems and thus simultaneously ensures synchronization of all FMUs involved. Information about supported functionalities is provided by each FMU via its own XML file [161]. In general, FMI's focus is on continuous simulation models, but discrete models are also supported [145].

In the current version *FMI 2.0* from 2014, the two previously separate standards were merged (*FMI for Model Exchange and Co-Simulation*). The two variants are shown in Figure 2.5. The difference between a global solver and isolated solvers of the respective submodels is relevant. By retaining the ability to maintain its own solver, FMUs can continue to present themselves completely as a black box, which allows a supplier, for example, to protect its intellectual property. The master algorithm that triggers the solver of each FMU in the co-simulation mode is explicitly not defined in the standard, nor is the wire protocol. This means, as for HLA, that the FMI standard does not provide a final solution to the problem of DS. Currently, there is ongoing work on FMI 3.0 tackling some of the disadvantages from FMI 2.0. As a consequence of the current capabilities and limitations, work has emerged that addresses the combination of HLA and FMI [15, 16, 34, 203], or that explores the combination of FMI with other technologies such as RPCs [106]. Moreover, there is

an effort by Modelica to develop a communication protocol for FMI, the so-called *Distributed Co-Simulation Protocol* [128].

2.3.4 Implementations of Distributed Simulation

With HLA and FMI, there are two mature standards that deal with the topic of DS. Both are rather theoretical in their nature and provide no (complete) reference implementations to DS. Therefore, this section will present projects that are toolkits, implementations of generalized frameworks, or middlewares for DS, either based on the two standards or using other technologies. Of course, the collection is not an exhaustive list of all existing DS projects. It rather serves to give an impression of relevant works and their mode of operation.

Generalized Frameworks and Middlewares

The first group of projects is based on the FMI standard. **Cosimate** [153] addresses the increasing complexity of systems. Whether in the automotive domain, or in other disciplines, the development of new systems requires the collaboration of heterogeneous teams with different domain knowledge. In order to test the interaction of the emerging subsystems at the earliest possible stage of development, they propose an open, bus-based system for co-simulation. As an advantage of their approach, they emphasize the open architecture, which should allow to connect an unlimited number of simulation environments. Furthermore, it should be possible to combine models of different abstraction levels. However, additional technical details are not disclosed, since the project pursues a commercial approach. **DaccoSim NG** [78] is based on JavaFMI and provides an FMI master algorithm. In addition, a graphical user interface allows a user to drag-and-drop existing FMUs together and graphically define information flows between the FMUs. In this way, the user is assisted in counteracting the problem of semantic interoperability. **FIDE** [63] represents an IDE for FMI based scenarios. It is based on the Ptolemy II framework and accordingly offers Ptolemy's functionalities, such as the graphical interface and the simulation engine. The goal is to enable the user to set up a co-simulation by connecting multiple FMUs. Accordingly, FIDE provides the implementation of an FMI master

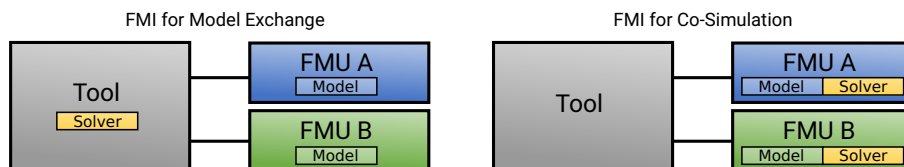


Figure 2.5 – FMI modes. Based on [146].

algorithm that deterministically couples both discrete-time and continuous-time models. The **maestro** framework [213] was created as part of the INTO-CPS project. In general, the INTO-CPS project sees itself as a toolchain for the model-based design of cyber-physical systems. INTO-CPS supports the multidisciplinary character of such systems from modeling to implementation. The co-simulation of such systems is again based on the FMI standard. Other artifacts from INTO-CPS support the modeling of FMUs, the evaluation of experiments, and also the validation of FMUs. **MECSYCO** [46] is also based on the FMI standard. It considers the simulation of complex systems and provides a master algorithm for this purpose. The approach is not limited to FMUs and integrates further components. The problem of integrating new subsystems is addressed with the help of metamodeling.

There are several implementations of the HLA standard, but many of them are rather academic and incomplete. In [95], the four most relevant implementations were identified and are briefly presented below. The French Aerospace Laboratory began developing **CERTI** [163] in 1996. Since 2002, CERTI is an open source project and comes with a C++ binding. From an architectural point of view, CERTI RTI is divided into two parts: the ambassador RTI (RTIA) and the global RTI (RTIG). Each federate has a local RTIA that enables communication between federate and RTIA via Inter-Process Communication (IPC). In addition, there is a global RTIG process that communicates with the RTIA processes via TCP and UDP sockets. Similar to CERTI, **MAK** [139] consists of a local RTI component (LRC) in each federate and a global RTI component. The LRC includes more functionality than a pure RTI ambassador implementation, which enables decentralized operation for some use cases. The connections are primarily implemented using TCP sockets. While the lightweight mode is decentralized, it is promoted to be “well suited for many real-time federations that do not use time management, DDM, MOM, reliable transport (TCP), or synchronization points”. Similar terminology is used for **Pitch** [174]. While a central RTI component (CRC) may be hosted on a separate node or next to a federate, each federate has its own local RTI component (LRC). The CRC manages the federation and delegates work between the LRCs. There are bindings for C++ and Java. Pitch uses TCP sockets for reliable and UDP sockets for best-effort communication. When the federates are running in the same process, communication is done via shared memory. In addition, multicast communication can be used for the best-effort transport mode. **Portico** [175] was started in 2005 as jaRTI and is currently available in a stable version 2.1.0, released in 2016. The Portico project was started with funding from the Australian Defense Simulation Office and as a response to the lack of open source RTI implementations. Therefore, the goal was to create an open source RTI alternative to commercial RTIs. Currently, major changes have been announced to transition from a fully decentralized approach in version 2.1.0 to a more centralized paradigm [190]. This transition could have

serious implications for Portico's performance in future releases. Because of their architecture, the developers state that it is very easy to change communication protocols, unlike in other RTI projects. In the current version, there is an option to communicate via jgroups or shared memory (if federates are running in the same process).

Besides the pure RTI implementations, there is another project which is strongly oriented towards HLA, but aims for more than just implementing the standard. In general, the goal of the Advanced RTI System (**ARTIS**) [31] is to support parallel and distributed simulation of complex systems consisting of heterogeneous and distributed submodels. The HLA standard has been extended with additional functionalities to improve scalability and execution speed. The communication is realized by shared memory, if possible, otherwise R-UDP, TCP, multicast, or Message Passing Interface is used [167]. ARTIS focuses on the message exchange between the LPs. This has a significant influence on the performance of the overall system. Therefore, ARTIS tries to optimize in an adaptive way the mapping between the LPs and their physical execution location in order to minimize the communication overhead. In addition to dynamic load balancing, the authors also refer to dynamic communication balancing. Both common types of synchronization, optimistic and conservative, are supported by implementations of the well-known Chandy-Misra-Bryant and Time Warp algorithms [30]. The lower layer of the overall architecture is IEEE 1516 compliant (e.g., Time Management). In parallel, however, additional interfaces are provided to make access easier. The Generic Adaptive Interaction Architecture (**GAIA**), by the same research group, interacts with ARTIS and enables for example the migration of entities between LPs.

There are also additional approaches that do not build on FMI or HLA. The RESTful Interoperability Simulation Environment (**RISE**) [4] takes a different approach, as the name suggests. Components communicate via XML structures over REST. A main motivation for the architecture is to enable the integration of web services in coupled simulations. The University of Toronto's **UT-Sim** framework [116] uses the University of Toronto Networking Protocol (UTNP) for communication which in turn can be used on top of TCP and UDP.

Specific Frameworks

There are other projects in the field of DS that focus on specific domains. The energy sector, for instance, is considered by a great variety of frameworks. Especially the smart grid context is of interest, because it spans across the energy and the communication domain. **Mosaik** [198] is a framework for simulating smart grid scenarios that follows a centralized architecture. It is written in Python and utilizes the SimPy library. However, there are also bindings for Java, MATLAB, and C#.

The communication is realized via JSON messages over TCP sockets. Similarly, there is **INSPIRE** [88] and **FCNS** [58]. Inspire also puts the communication of control systems in focus and presents an HLA-based framework for investigating both power systems and related intelligent control systems. Pitch pRTI is used as RTI implementation. FCNS is based on the publish/subscribe principles of HLA, but implements the message exchange between simulators via ZeroMQ. The main objective of **HELICS** [169] is to support large-scale simulations in the energy domain. This is possible by parallelizing and using models at different levels of detail. GridLAB-D, GridDyn, MATPOWER, OpenDSS, PSLF, InterPSS, and FESTIV are integrated among others using ZeroMQ. Simulating a great number of distributed energy resources is the motivation of **CyDER** [162]. Therefore, CyDER integrates several established tools based on the FMI standard. It uses PyFMI as the master algorithm.

Also, other domains are covered. **MOSAIC** [82] represents a framework for simulating connected and autonomous driving use cases. It is a further development of the **VSimRTI** project [197] and is based on HLA. For this purpose, well-known simulation tools from the field of mobility and communication are coupled with each other: SUMO, ns-3, and OMNeT++. Even if the name of the **Open Simulation Platform** [172] seems generic, the goal of the project is the realization of co-simulations in the maritime environment. It is based on FMI and provides an own FMI master algorithm.

Specific Toolkits

In addition to the previous section, we lastly present work whose purpose is the concrete coupling of usually two different tools. These projects are most likely to work out of the box. Global Event-Driven Co-Simulation (**Geco**) [136] also has a more specific purpose than its name would first suggest. It couples two simulators: the Positive Sequence Load Flow Simulator (PSLF) and ns-2. The synchronization is done via a global event list and the communication is implemented via IPC. The authors mention the theoretical possibility of extending the toolkit with further simulators, but point out that effort might be huge. **SGSim** [14] combines two well-known stand-alone tools, OMNeT++ and OpenDSS, using a COM-Interface. Based on HLA, the authors in [33] present a toolkit for coupled consideration of power and communication systems. For this purpose, OPNET is coupled with EMTP using HLA. The Transactive Energy Simulation Platform (**TESP**) [114] uses the mentioned FCNS and HELICS framework to combine GridLAB-D, TSO, OpenDSS, EnergyPlus, and ns-3.

RoboNetSim [131] combines ARGoS and ns-2 or ns-3. The robot simulator uses a DES with fixed step length, while the network simulator uses DES without time constraints. After each step of the robot simulator, the updated positions of the robots

are sent to the network simulator via TCP or UDP sockets. In addition, other more general information can be exchanged so that the robot simulator can, for example, incorporate the network load into its positioning strategy. **Veins** [205] combines SUMO and OMNeT++ via TCP enabling Vehicle-to-Everything (V2X) simulations. Analogous to RoboNetSim, the primary purpose of the information exchange is to transfer the positions of the vehicles simulated by SUMO to OMNeT++ in order to update the positions of the network nodes. This perspective might not be hardware-aware enough to validate electrical/electronic architectures. Therefore, the authors in [189] extend Veins using Ptolemy II to include the ability to model electronic control units. The overall model is thus extended at the hardware level to include aspects of sensors and actuators. **iTETRIS** [188] considers a possible problem of Veins. Both the V2X application to be evaluated and the management of the co-simulation are implemented within OMNeT++. Accordingly, it may be necessary to react to changes in the architecture of OMNeT++. Above all, however, users have to familiarize themselves with OMNeT++ before modeling the algorithms to be tested. They propose a different architecture for the iTETRIS framework. Here, a central management and control instance has IP socket connections to the traffic simulator (SUMO), the network simulator (ns-3), and possibly others. There is also an interface to the applications under test. **Artery** [185] extends the 802.11p WAVE protocol implemented in Veins to the European ETSI ITS-G5 protocol. **MOBATSim** [192] provides a toolkit for MATLAB Simulink. Scenarios from the field of autonomous driving can be modeled and be simulated with a focus on the investigation of error causes. The framework is not tool-independent, but within MATLAB Simulink it provides the possibility to integrate different isolated models and to investigate their interaction on the different levels of the component. There are also approaches that couple tools via shared files. The Aerial Vehicle Network Simulator (**AVENS**) [141] represents a coupling between OMNeT++ and the XPlane simulator. The goal is to provide an environment to study mobile ad hoc networks spanned by unmanned drones for communication. OMNeT++ is used for the communication layer, while XPlane models the movement of the flying objects. Information exchange between the two simulators is provided via shared access to a local file system. Specifically, after each simulation step XPlane writes the current positions of the previously defined flying objects into an XML file, which is continuously read by OMNeT++. The coupling of Gazebo and ns-3 is called **Cornet** [2] and is aiming for a similar use case. The movement of the objects is handled by a continuous model of Gazebo, the communication part is modeled by ns-3. The data exchange between the two tools is done via ZeroMQ.

2.3.5 Simulation as a Service

Simulation-as-a-Service (SaaS) seems almost as a logical consequence of distributed systems, distributed simulation, and the trend towards cloudization of entire domains. The concept of Modeling-and-Simulation-as-a-Service (MSaaS) is either used synonymously or in order to explicitly emphasize the modeling capabilities of a service. An early definition of MSaaS comes from a survey paper published in 2013, which also discusses possible architectures and risks:

“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).” [52]

The Modeling and Simulation Group MSG-131 (“Modeling and Simulation as a Service: New concepts and Service Oriented Architectures”) of the NATO also seems committed to put resources into the development of MSaaS concepts and implementations. Within a twelve-year plan, defined use cases are to be realized [202]. Here, the ITIL definition for services will be used to define MSaaS:

“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.”

This definition is expanded in a more recent publication of the Modeling & Simulation Group MSG-136 (“Modeling and Simulation as a Service - Rapid deployment of interoperable and credible simulation environments”) by shifting the focus to MSaaS:

“MSaaS is a new concept that combines service orientation and the provision of M&S applications 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.” [201]

All of the above definitions use the concept of *on-demand* as their common theme. Furthermore, the first two explicitly place the responsibility for errors with the service operator. The third definition adds that the integration of different virtual and real systems should be possible.

From a more application-centric perspective, there is also research on MSaaS. The authors in [200] see the main challenge of modern simulation in the potentially huge demand for computing power. They recognize opportunities in computing on clusters and present a simulation service. This enables the user to adjust certain parameters of a simulation via an interface before it is run. In general, modifiable parameters include the name of the model, the number of repetitions, or the command to run the simulation. For illustration purposes, a traffic simulation is performed as a use case. It is also possible to adjust model-specific parameters, such as the traffic volume. This then makes it possible to perform a parameter variation study. Regarding the simulation execution, the authors note that they see an advantage in a container-based solution (such as Docker) compared to heavyweight hypervisor-based approaches. Docker also forms the technical basis for simulators of the work in [27]. A DES is implemented based on an existing middleware called SOASim [68] with a microservice architecture. The communication is done via REST.

Further work addresses the use of MSaaS in concrete application areas such as the energy domain [178], weather models [151], crowd modeling [224], or the education sector. For the latter, criticism of current STEM schooling forms the motivation for the work of Caglar et al. [44]. The authors would like to enable the analysis of real systems in high school by working with simulations of such systems in a playful way. One condition for this is undoubtedly the availability of the simulators in the classroom. Therefore, the use of a cloud-based simulation platform is proposed to ensure direct usability. As a use case, the concept is implemented for a traffic simulation. Bitterman et al. [25] address the education sector as well. They extend the requirement for availability of simulation even further to include availability of data and see the potential of MSaaS primarily in providing this with low barriers to entry. Krumnow expresses similar ideas and proposes a web interface through which a simulator can be used [129]. There is a lot of research in this area that involves the simulation of mobility. This ranges from ITS applications [104], over networked mobility [126], to very large-scale traffic simulations [103, 233].

In conclusion, there is already a variety of work investigating, developing, and applying MSaaS concepts in general and also related to specific application purposes. However, the implementation details of the proposed platforms, their interface definitions, detailed descriptions of the internal processes, or even access to them, is mostly not available.

2.4 Data Centrism

In today's world, data is available in unprecedented quantity and variety. As a result, under the umbrella of *Big Data*, procedures for storage and evaluation have developed, whose purpose lies primarily in controlling the floods of data. At the same time, *data science* has established itself as a separate profession in this field with the goal of gaining knowledge on the basis of (automated) analyses of existing data streams [180].

Data-Driven Approaches

Data-drivenness has slightly different definitions depending on the field of application. As simple as the concept of data-driven approaches may seem on an abstract level, there is no generally accepted definition beyond the meaning of the word. However, methods described with the term can often be reduced to the ideas of *Data-Driven Decision Making*. In this context, data-drivenness describes deriving decisions based on data analysis rather than pure intuition or experience [180]. This corresponds to the core of the data-driven idea regardless of the target domain. Data is used to derive actions or strategies. Among others, such strategies can be reflected in business models, design decisions, or goal formations. The authors in [211], for example, examine the advantages and a possible implementation of data-driven strategies in an intelligent manufacturing plant for wafers.

Data-Centric Approaches

In contrast, data-centric approaches also take the system architecture into account. Accordingly, data-driven and data-centric approaches do not oppose each other, but are suitable for complementing each other. The core of the data-centric idea is to break down isolated data silos within an organization. In a historically grown corporation, there are typically several applications that may be interconnected to a certain degree. Nevertheless, it is conceivable that each application is responsible for its own data management with its own data models and databases.

Mainly two problems can arise from this. One challenge is keeping redundant data sets consistent across databases. Furthermore, innovation may be lost as separate data management may prevent opportunities from being identified. By that, the full exploitation of existing potential is limited. With increasing digitization and new data sources and applications, this problem is becoming more apparent.

The data-centric approach attempts to integrate (mostly) all existing data in a common data model on which all applications then work together (see Figure 2.6). While the architectural concept may appear simple, the possibilities that become

feasible as a result are extensive. In the corporate environment, there are efforts to make a paradigm shift from an application-centric approach to a data-centric approach [7].

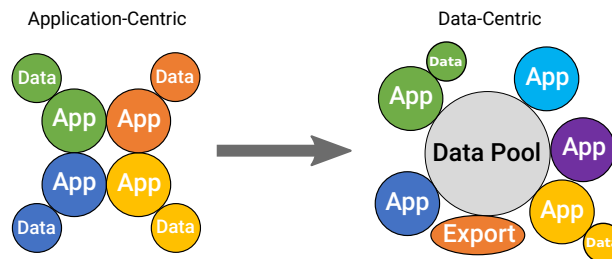


Figure 2.6 – Application-centric and data-centric architectures. Based on [79].

Data Reuse

In the context of M&S, the term *reuse* is mostly used with regard to the reuse of components. However, there are also works that are considering the reuse of data. Blondet et al. [26] see the main advantage of simulation data reuse in shorter simulation processes. The authors distinguish two kinds of data: long-term data and short-term data. Long-term data includes the simulation results, but also further analyses and interpretations of the results. In addition, long-term data covers the initial modeling choices and parameters. Data that is required for computing the next simulation steps (e.g., intermediate results) is called short-term data and is typically not available anymore when a simulation study is finished. In the field of product engineering, product life cycle management sometimes offers Simulation Data Management (SDM). Such tools cover the provision of historic simulation data on various levels, reaching from a local database to a collaborative cloud storage. The latter may in particular require to use structured data models to be able to benefit from a large amount of heterogeneous data sets. Also, the use of ontologies can help in this context. For some fields, ontologies already have been designed therefore, for instance, for physics simulation [56] or building energy simulation [164].

2.5 Requirements Analysis

Based on the overview on the historical foundation and the current state of research, the question arises which requirements exist towards distributed simulation (and frameworks for its realization).

The authors in [91] answer this from their point of view with a detailed collection. They distinguish between three types of requirements: non-functional requirements,

simulator-related requirements and framework-related requirements. As a non-functional requirement **fault tolerance** is mentioned, which can be ensured, for example, if components are designed redundantly and can be replaced dynamically in case of a fault. By **configuration reusability** they mean the ability of a framework to have a scenario configuration file that is abstracted from the used tools. Creating a new scenario by adapting such a file and without having to make changes to the tools should be possible. Other issues represent **performance**, **scalability**, and the ability to **parallelize** and **distribute**. Potentially, **protection of intellectual property** can be supported, for example, in the form of black box component integration. Combining coupled subsystems into individual (black box) elements of a larger **hierarchy** can simplify the modeling of complex systems. If **extensibility** is given, a framework can be extended with additional functionality by new simulators with relatively little effort. Furthermore, they also list the requirement of **accuracy**. The last points mentioned are **open source** licensing models and **platform independence** of the framework.

With respect to simulators, five additional requirements are named. A simulator might provide **insight** into its internal state at a wide variety of levels. Examples include intermediate results, formal model equations, or gradients for extrapolating intermediate results. Simulators should emphasize **causal correctness** and **availability**. In addition, it may be necessary to adhere to certain **time constraints** and have the ability to perform a **rollback**. In that case, a previous state is restored, for example, because a causality violation has occurred due to an optimistic synchronization procedure. The addressed third group of framework-related requirements presents itself rather as listing of possible characteristics (e.g., communication paradigm), which serves for the classification in a taxonomy. We will therefore skip these points.

In general, a composed model is expected to satisfy the classical requirements towards a simulation model: **validity**, **credibility**, **usefulness**, and **feasibility** [187]. Apart from the presented compilation, additional (and also already mentioned) requirements can be found in other literature sources. In many cases, an **easy usability** is mentioned as a requirement for a simulation framework [80, 143, 173, 208, 210]. This includes tool integration, scenario design, as well as simulation execution. Intuitive usability may be particularly relevant if the framework should contribute to decision making in critical situations such as environmental disasters. A **graphical user interface** plays a crucial role in this regard [173, 195] and can usually help to keep the **effort** of the user for configuration and execution of a coupled simulation low [109]. The possibility of a **centralized orchestration capability** is also mentioned [166].

Although they generally appear to be desirable, the importance for **stability** [109, 173], **error tolerance** [109, 173], **reliability**, and **availability** [140] is weighted

differently depending on the application area. If the infrastructure for running distributed simulations is not under the user's own control or publicly accessible, **security**, **trustworthiness**, and **encryption** play a major role [140]. More specific, there are requirements of **access control** [173] or **secure communication** [80]. Other non-functional requirements include **performance** [208], a **resource-saving framework core** [80], mechanisms of **load balancing** [69], and **scalability** [173, 187].

To ensure correctness, it is necessary to enable **synchronization** of the subcomponents [209], potentially also with methods that support **variable step lengths** [213]. Apart from that, it might be necessary to integrate components that are **time independent** [3]. Further work presents communication under **real-time guarantees** [143, 210] as a possible requirement. Soft real-time guarantees may also be necessary to ensure **interactivity** [140]. A special case of real-time capability is the ability to run **synchronous to wall-clock time**, which may especially be required if real-world components are integrated into the scenario (HiL) [3, 209]. Aside from physical devices, there might be other external components, for instance, external control or evaluation scripts. Accordingly, the requirement for **external interfaces** is expressed [168].

The framework implementation should be based on **open standards** [80, 109], be **platform and architecture independent** [213], and ensure a **homogeneous and programming language independent interface** [213]. Potential advantages of a general framework are **extensibility** [80, 213], **flexibility** [143], **backward compatibility** [109, 173], and the possibility of **reuse** of individual components [3, 173]. For a variety of applications, the possibility of a framework for **logging** [168] is desirable. Another requirement is to support **black box**-based coupling approaches to protect intellectual property [3]. However, so-called information hiding can also address the **hiding of sensitive data** that is integrated into a model. Such requirements might apply, for example, in the context of medicine, military, or mobility [96]. As a contrary position, though, there is also a requirement for the possibility of the **disclosure of all data** and internal states of the subcomponents [168]. It should not only be possible to locally connect similar components. This should also be possible if they implement **different paradigms** (DES, CT) [195], if they are **geographically distributed** [3], or represent pure (external) **data source** [173] providers. In general, it is desirable to have functionalities that facilitate **collaboration** [187] of users. Low cost or **free licenses** [143] naturally play an equally important role in acceptance.

2.6 Research Gap

In summary, distributed simulation is a mature concept that builds among others on preliminary works from the fields of M&S and distributed systems. At the same time, it still holds open challenges. In particular, the fairly recent trend towards simulation services offers further potential. There are rather theoretical works that address information transfer of submodels in general and aim for the reuse of simulation models. The information exchange between submodels usually provides room for interpretation, because the conceptual connection of heterogeneous submodels could be anything but unambiguous. As the conceptual connection is often the result of a modeling process itself, we consider the model connections of coupled models as the critical point for reusing, reproducing, and finally adapting coupled simulation models. To the best of our knowledge, there are no comparable works that explicitly consider these challenges of model translations. Thus, the conceptual coupling model or the implementation of the coupling model (e.g., a translation between different levels of detail) should be disclosed, be provided, and be reusable like a regular simulator component.

There are also more practical works, often with a particular target domain or a certain application-focus. Some of them are very tool-specific connections of two or more software artifacts. Regarding published results, one may search in vain for relevant information (e.g., used parameters, version numbers, or compile flags) or fail to obtain the used software artifacts. That can result in problems regarding the reusability of single parts and the reproducibility of findings of the composed simulation study - even if the involved tools themselves might offer all features needed. As for the first point, this has the potential to damage the credibility of simulation. Consequently, a composed simulation scenario should be described in a comprehensive and structured manner.

A seamless extendability to support other topologies by incorporating arbitrary simulators and models is in most cases not possible with the existing toolkits, especially not in a plug-and-play manner. Proposed couplings are often static and designed around the inner logic of certain tools. More general approaches, incorporating HLA or FMI, cannot fully compensate the drawbacks of their middlewares. Beside the mentioned problems, potential users might fail at the high entry barrier, even if they might just want to access some data points. Thus, our system should be built on top of a general-purpose big data messaging platform that is well-known, has an active user community, and provides simple data access methods. Accordingly, the couplings should be realized in a loose and extendable way.

Apart from the functional aspects, non-functional requirements such as licensing modes are a problem, when addressing accessibility. Accordingly, a simulation system should be available under open source terms.

Considering data-reuse, the reuse of long-term data seems useful without any doubt, for instance for static input resources. In addition, we especially see potential in storing and reusing short-term data (i.e., intermediate results). This once again strengthens the aspect of reproducibility, but also enables adaptations of historic simulation runs (e.g., reusing simulated vehicle positions for a new V2X communication simulation). Therefore, it should be possible to store simulation data in a structured way and ingest existing data into new simulation runs.

To conclude, there is a gap between the collected requirements and the functionality that is currently provided by existing distributed/co-simulation projects. Different projects address various problem areas, but no single approach is covering all identified problem fields. At this point, the present thesis assumes relevance. The main objective is to identify and solve related issues to the presented problems and requirements. The developed solutions for all sub-problems will be put together under the hood of a self-contained data-centric distributed simulation system. Involved layers cover the technical communication, the coupling methodologies, and a high level M&S service that provides applications for designing, running, and evaluating simulation scenarios via a graphical interface. Such a system will enable the user to focus on the real tasks and not having to bother with technical details or the modeling of problems that has already been done once for previous scenarios. Prepared tools and models will be plugged together like building blocks. If an elaborate conversion of data flows between coupled models is required, existing connector components will be (re-)used. Aside from that, a secondary objective will be an exemplary case study in which the developed concept is applied to the traffic domain and potential use cases are demonstrated.

Chapter 3

System Specification and Design

The previous chapter led to the conclusion that there is no existing solution that covers all presented requirements nor the objectives stated in Section 1.2. A new methodology that fills the current gap will be developed conceptually in this chapter. Consequently, the designed concepts will be combined into a service-based simulation system.

The specific requirements that can be derived from the objectives are identified and formulated as a first step. Subsequently, the core idea of a data-centric approach is described, which will provide the architectural foundation for all further steps. After that, information on the realization of communication on a lower layer will be given. Then, the coupling concept, which operates on top of that is developed. Finally, an MSaaS application layer for the proposed methodology is designed and a summary is given. Large parts of this chapter are based on previously published works in [96], [97], [98], [99], and [100].

3.1	Requirements Specification	41
3.2	Data-Centric Architectural Concept	43
3.3	Communication Concept	47
3.3.1	Middleware-based Approach	47
3.3.2	Publish/Subscribe Messaging Systems	48
3.3.3	Apache Kafka	50
3.3.4	Data Serialization	51
3.4	Coupling Concept	52
3.4.1	Core Idea	53
3.4.2	Domain & Layer Taxonomy	56
3.4.3	Deterministic Information Exchange	61
3.4.4	Coupling Protocol	72

3.4.5	Composing Building Blocks	74
3.5	MSaaS Concept	81
3.5.1	Definition and Design of Scenarios	81
3.5.2	Execution of Scenarios	85
3.5.3	Evaluation of Scenarios	86
3.5.4	Management	87
3.5.5	Graphical User Interface	88
3.5.6	Service Architecture	88
3.6	Summary	89

3.1 Requirements Specification

Some of the general DS requirements that were collected in Section 2.5 may be redundant, address similar points under different terms, contradict each other, or are not applicable under the given motivation of this thesis. In the following, a subset of relevant requirements is therefore filtered out of the total set of requirements. In case of conflicts, the requirement that appears to be more useful is chosen. The given motivation for the desired simulation system builds the groundwork for all filtering decisions and also for organizing the different remaining requirements within five categories. These categories are coupling, correctness, resources, data, and usability. If a requirement fits into multiple categories, it is just given once. Consequently, the resulting set of requirements is a consistent collection that allows the design of the desired system.

In the course of the following sections, a methodology, respectively a system, is developed which satisfies these requirements. Since many of the upcoming design decisions are based on more than just one requirement, there will be no reference to the related requirements for each and every single design choice. Instead, the summarizing Section 3.6 depicts the links between requirements and considerations. Furthermore, it is shown that all identified requirements are met.

Coupling

Within this category, all requirements that are directly addressing the coupling are collected.

- REQ 1.1** The main requirement is to be able to combine different tools. This should involve simulators and other components in general.
- REQ 1.2** It should be possible that the different components are executed on different computing nodes and may also be run on different operating systems and hardware architectures. Implementations in different programming languages should be no problem.
- REQ 1.3** The combination of different modeling paradigms should work as well.
- REQ 1.4** Every component should be able to run on its own and a failure in one component should not lead to the failure of a connected one. Therefore, the coupling should be loose.
- REQ 1.5** Existing couplings should be easily extendable with new components.
- REQ 1.6** An external interface should be provided to enable the interaction with a running DS in order to support a wide variety of use cases.

REQ 1.7 Encrypted communication should be supported to protect data.

Correctness

This category covers requirements related to the efforts of achieving correct results. Correctness is the prerequisite for repeatable and reproducible simulations that lead to credibility.

REQ 2.1 There is the possibility to synchronize logical clocks of coupled components.

REQ 2.2 Based on that, it must be possible to exchange messages between components in a deterministic order and on time.

REQ 2.3 A DS should be repeatable. The same input should always lead to the exact same outcome.

REQ 2.4 (Intermediate) results during a simulation should be stored and be traceable.

Resources

This category deals with topics related to resources, which includes hardware resources and software artifacts.

REQ 3.1 A DS should provide the potential for an increased performance compared to the sequential alternative.

REQ 3.2 Large-scale simulation scenarios should be feasible by a scalable architectural concept.

REQ 3.3 Components should be reusable.

REQ 3.4 The system should be open source.

Data

The gathering, integration, storage, and access of data is covered in the following.

REQ 4.1 It should be possible to bridge the gap between accessible and required data.

REQ 4.2 Information should be represented in a structured way.

REQ 4.3 Exchanged messages should be stored persistently.

REQ 4.4 A common data pool should support the integration of heterogeneous components and data sources.

REQ 4.5 An external interface should be provided to enable access to the state of a running DS.

REQ 4.6 It should be possible to realize access management strategies if required.

Usability

The usability is in the focus of the following requirements.

REQ 5.1 A MSaaS functionality should be provided. The user should not have to provide any specific tool on his local machine.

REQ 5.2 A structured format for describing simulation scenarios is required.

REQ 5.3 The design, the execution, and the evaluation of scenarios should be possible via a graphical user interface.

REQ 5.4 There should be a low entry barrier.

3.2 Data-Centric Architectural Concept

In order to meet the previously specified requirements, a data-centric architectural concept is designed. The core contribution of this work is the development of a coupling methodology for DS on the basis of a data-centric paradigm. Fundamental information about the data-centric approach was presented in Section 2.4. The data-centric idea will be adapted to the use case of DS in the following.

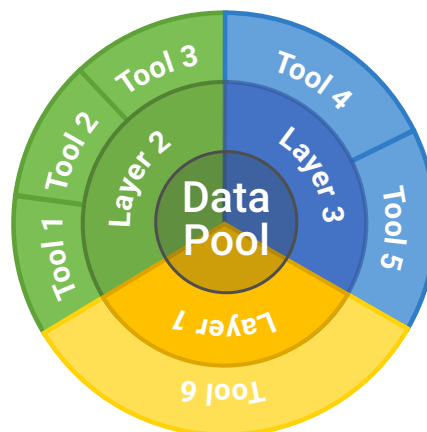


Figure 3.1 – Data-centric architecture.

According to the data-centric thought, relevant (simulation) data will not only remain inside subcomponents of a composed simulation, but is integrated in a shared data pool (see Figure 3.1). Consequently, there will be no isolated data silo per component, but rather one common storage that builds the heart of the overall system. By reading and writing information from and to this data pool various kinds of DS are to be realized, ranging from simple parallel simulations to complex co-simulations. In the same way, external data sources can be integrated into composed simulations and analysis/control scripts can run in parallel.

In addition, this means that information does not need to be exchanged via transient messages, which disappear after delivery, but can rather be stored persistently by suitable technologies. As a consequence, it is possible to repeat (parts of) a simulation and compare or reproduce results once a simulation run is finished. A post-processing or visualization of interesting events is possible in the same way. Information flows can be traced across components. By keeping the data available, additional questions can be answered at a later stage that may not have been known before when designing or running the simulation. This retrospective analysis is not restricted to a certain simulation run. For instance, the impact of a parameter variation can be investigated across several simulation runs. Another use case might be to use a subset of the generated messages as an input for a new simulation study.

Regarding the coupling approach, this implies that the traditional perspective is changed. The question is no longer how to couple specific models or tools (e.g., does it offer an API, what is the programming language, which software version is used). It is rather about how to couple different data models in a more abstracted way. Consequently, the focus is shifted from the tool implementations to their data. This lays the foundation for a very loose coupling and, potentially, a great level of reusability and extendability.

Based on the question of what data tuples are produced and consumed by a certain component, stereotypical domain-specific data models need to be designed. When using an appropriate degree of abstraction, these data models are reusable and cover a whole group of homogeneous models and components. Such a group of

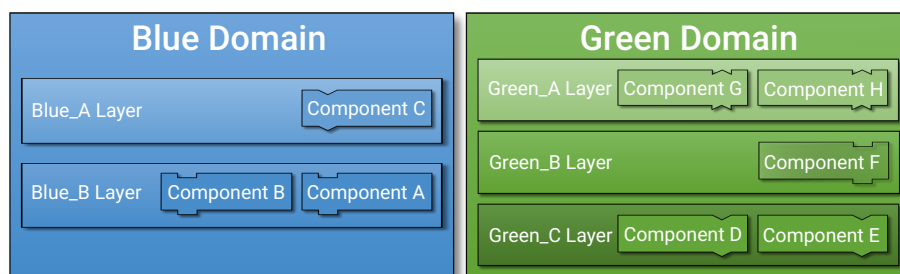


Figure 3.2 – Similar components can be hidden behind the same layer.

similar components can then be integrated in the identical manner (see Figure 3.2). In order to have stereotypical models that are able to represent a whole class of different components, it might be necessary to ignore individual tool- or model-details. Assessing the trade-off between specific functionality and generalizability is thereby a crucial step. The resulting data model will be referred to as *layer* in the following. Eventually, the layer is representing the abstraction level at which information is exchanged in order to realize a coupling between components. Apart from the interfaces that can be inferred from the layer, a single component is seen as a black box in the simulation system. Therefore, the predefined data models of the layers are essential for the data-centric approach. Without the data models, stored messages would lack their syntactic and semantic meaning and could not be used without room for interpretation.

Another aspect that should be covered by the coupling approach is the problem of verification and validation of coupled models. One might argue that it is not feasible to have a global insight into every detail of a large-scale model. This might especially be the case for HiL simulations, but also due to the protection of intellectual property, proprietary components, or other technical reasons. In our approach, the integrated black boxes are assumed to be verified and valid for their specific purpose. The potential weak spot of the validity of the composed model thus remains in the translation of information flows between the different black boxes. Therefore, we suggested to detach this translation process to dedicated components (see Figure 3.3). By doing so, we avoid to spread converter logic across the internals of the black boxes themselves or across the wrappers around them. Regarding other requirements such as reusability and scalability, this seems reasonable as well, because the translations are exclusively converting information between predefined layer data models. The

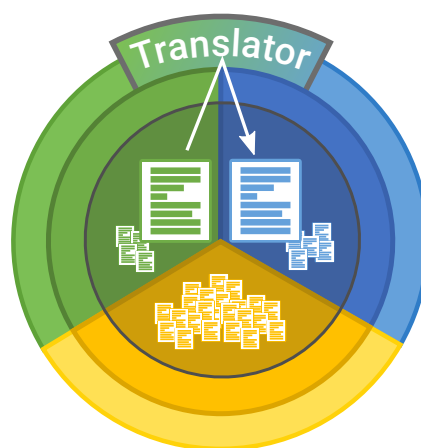


Figure 3.3 – The translation of information flows between submodels is detached into separate reusable components.

descriptions of the translation models can be organized in a central place, where also the layer definitions and data models are provided. With that, the process of translation is fully disclosed and reproducible. As all exchanged messages are stored, the input and the output of translations also remain in the persistent data pool. Eventually, the verification and validity of complex composed models can be assessed under the given assumptions.

Regarding the construction of a composed model, this implies that there is nothing wrong with composing building blocks in a bottom-up plug-and-play manner when modeling a certain scenario. No global knowledge of all involved components or even about their internals is required. As for the way companies apply the data-centric approach, each component can in theory access every piece of exchanged information - given that there are no access restrictions. In this manner, it is feasible to realize a loosely coupled, large-scale distributed simulation with heterogeneous components, while providing extendability.

The data-centric approach does solve another problem: the integration of historic and live data sources and other real-world components (see Figure 3.4). By using the black box approach and the loosely coupling of components, the connection between different components is limited to the exchange of information pieces that comply to the layer definitions. Because of this, it does not matter anymore if a simulator is interacting with another simulator, with a data source, or with a physical component. In each case, information is produced to and consumed from the shared data pool. Of course, the exchange is not restricted to a 1 : 1 pattern, but can also happen in an $m : n$ pattern.

Based on this concept, the overall objective is achieved by distributing the open challenges across a three tier architecture. Starting from the bottom, the following sections cover the infrastructural tier, the coupling tier, and finally the application tier.

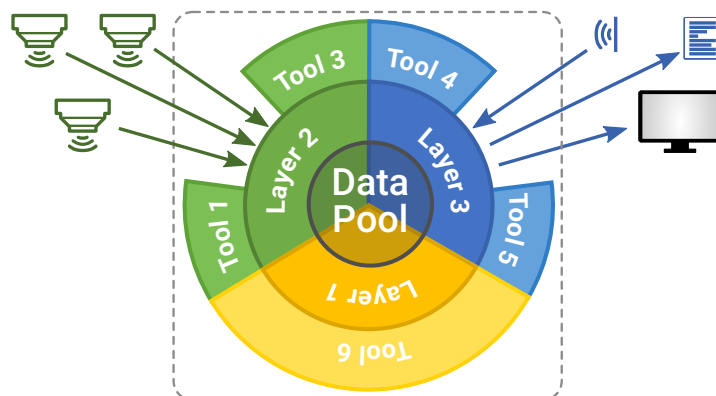


Figure 3.4 – Integration of external components.

3.3 Communication Concept

In this section, the lower layer of the system will be designed. Its main purpose is the realization of data exchange between multiple components and the storage of data.

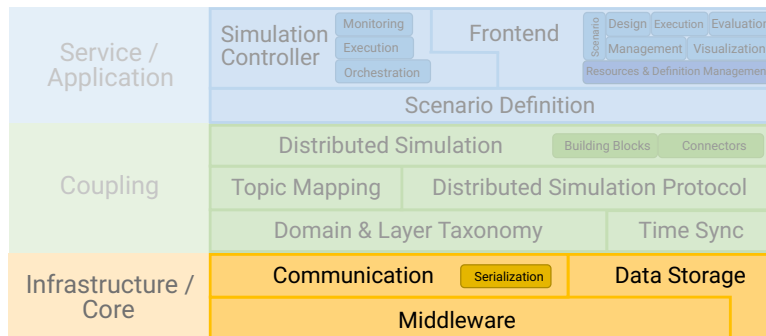


Figure 3.5 – The lower layer of the system’s architecture is developed.

3.3.1 Middleware-based Approach

As for distributed systems in general, the main architectural question is if the communication between components should be realized via direct point-to-point connections or in a middleware-based fashion (see Section 2.2). In this work, communication is primarily required to couple different components such as simulators. A middleware-based approach, more precisely using a publish/subscribe MoM, seems to be suitable for our use case. One of the most important reasons for that decision is to meet the requirement of extendability. In general, a point-to-point approach would require $n - 1$ new connections (one between each of the existing components and the new component), when adding an n -th component to a setup. That leads in total to $\frac{n*(n-1)}{2}$ bidirectional connections if a fully meshed topology between n components is required. In contrast, the middleware-based approach works similar as a communication bus. For n components there are n bidirectional links. Figure 3.6 depicts exemplary topologies for both approaches when connecting up to five nodes. The middleware-based approach would require two links for connecting two components, while the direct approach requires only one. For the case of three nodes, there are three links necessary for both approaches. It can be seen that with increasing numbers, the number of required links for the direct approach grows rapidly. The number of connections alone is not the only problem of a direct approach. If there is no common interface, each existing component might need to be adapted in order

to be able to interact with the interface of a new component. This is not feasible if the collection of composable components reaches a certain size.

From a performance perspective it is likely that the direct approach performs better for obvious reasons. If only two or three components are connected, there is a clear overhead introduced by the middleware. However, if there are many components involved, a scalable message broker could add certain features leading to a more performant message distribution. In addition, for this work the focus on extendability and usability outweighs potential performance impairments. Another advantage of the middleware-based approach is that with typical publish/subscribe mechanisms it is possible to implement a message-based communication protocol instead of an address-based protocol. That brings the benefit that a sender does not need to know its recipients (and where and how to reach them). This feature will become important when designing the coupling concept in the following. Producing information to multiple receivers at the same time is supported by many publish/subscribe concepts. Furthermore, based on the requirements, every exchanged message should be stored. In case of direct point-to-point connections that would require an additional connection per component for forwarding duplicates of each message to a storage component. Aside from the general advantage of easing the extendability, the middleware-based approach implicitly allows for dynamic changes in the topology of communication partners. This can be helpful if additional instances are joining a distributed simulation during the simulation execution, for instance, triggered by a load balancing mechanism.

3.3.2 Publish/Subscribe Messaging Systems

There are multiple well-known publish/subscribe MoMs in the field, which are widely used both in academics and as well in industry. Therefore, it is assumed that

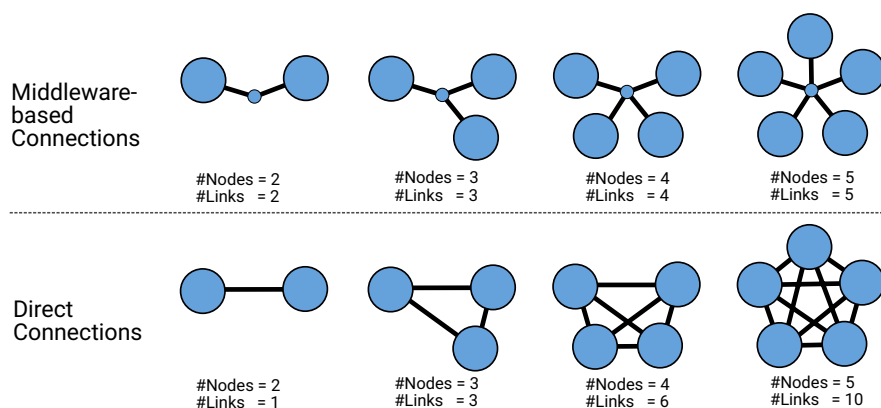


Figure 3.6 – Different link topologies.

there is at least one that is suitable for our approach and that there is no need for developing a proper MoM from scratch. A brief comparison of some of the most popular ones will provide more insights in the following. Based on the collected information, a reasonable decision on the most suitable MoM is made.

A literature review was conducted in order to get an impression on relevant projects/protocols. For this purpose, only recent survey papers were taken into account. Some works did also include non-MoM technologies (e.g., gRPC) because of varying intentions or use cases. Such projects will be ignored. Collected technologies are presented in Table 3.1 along with their count of appearance. The identified set of relevant technologies matches with our personal experience. Based on that, we will have a look on AMQP, DDS, Kafka, MQTT and ZeroMQ in the following.

One well known **AMQP** implementation is **RabbitMQ** [222]. A broker consists of exchanges that redirect messages to queues where consumers can pick them up. This is not only done with topics, but, for instance, also in terms of broadcast exchanges. **DDS** [170] follows a decentralized approach without data brokers. Per default data is held in memory, but could also be stored in a database. Apache **Kafka** was invented at LinkedIn before it was handed over to the Apache foundation. It is an open source messaging system that allows persistent storage of messages. The implementation uses an own protocol that is operating on TCP/IP connections. Messages are published to and consumed from a broker that can be replicated. Topics are used to organize messages. They are split into partitions in order to speed up read and write events. There is a guaranteed ordering within partitions and there are options for the common delivery semantics [9]. The focus of **MQTT** is providing a lightweight publish/subscribe communication protocol that works on TCP/IP. It is widely used for IoT communication especially when low-power devices are involved. There are different delivery semantics. Some implementations such as HiveMQ also support clustering [112]. Persistent data storage is not envisioned for MQTT broker implementations. **ZeroMQ** [234] is primarily a library, which is

	[5]	[231]	[207]	[29]	[179]	[216,217]	[107]	[84]	Count
AMQP	•	•	•	•		•	•	•	7
ActiveMQ		•						•	2
DDS	•				•	•			3
Kafka		•	•	•		•	•	•	6
MQTT		•	•		•	•			4
OPC UA					•	•			2
Redis							•		1
RocketMQ		•						•	2
ROS					•				1
Pulsar								•	1
XMPP	•								1
ZeroMQ		•	•	•					3

Table 3.1 – Recent considerations of MoMs in the literature.

available in many programming languages. As a consequence, there is a need for custom implementations. There is no intermediate broker, hence communication partners are in direct exchange. At the same time, this makes it a good choice when performance is top priority.

From our selection, only Kafka and RabbitMQ come with the desired message storage capabilities. RabbitMQ does not write the messages to disk by default. According to Dobbelaere et al. [73] both perform comparably regarding latency and throughput, although Kafka's latency may double when using an at-least-once semantic. Contrarily, Kafka shows better scaling abilities. Nannoni [158] notes that Kafka performs better than RabbitMQ when configuring RabbitMQ to store messages on disk. As Kafka satisfies all requirements regarding popularity, security, performance, scalability, and persistence it will build the foundation of our system. At the same time, the following design decisions for upper layers will be made MoM-agnostic if reasonable, so that the underlying MoM can be exchanged if the priorities or requirements change in the future.

3.3.3 Apache Kafka

Nevertheless, it cannot be avoided that some higher-level aspects are influenced by the capabilities of the underlying MoM. The former presentation of different MoMs was rather briefly in order to get an overview of the different systems. As a reference for the upper layers, we will describe relevant characteristics of Kafka in more depth in the following based on the official documentation [9].

Kafka is a topic-based publish/subscribe message streaming system that is using brokers as mediators. One or more brokers are placed in a cluster along with additional components such as data integrators (i.e., Kafka Connect). A cluster can be spread over different locations, which enables scalability and fault-tolerance. That means multiple brokers can be replications and/or split their work.

A message is called event in the Kafka terminology. An event contains a timestamp, a key, and a value. Publishers of events are called **producers** and subscribers are called **consumers**. An event is published to a specific topic. Topics are not exclusively assigned to one or more producers or consumers. An important feature is the event persistence. An event is not removed after its consumption but can be re-consumed. Size- or time-policies allow configuring when events are not needed anymore and can eventually be deleted from disk.

A topic can consist of multiple partitions that can exist on different brokers. Different partitions of a topic are a core element for the scalability and are therefore no plain copies of each other but hold different events of the topic. There is a guarantee that within a partition consumers will see events in exactly the order they were published, but this applies not topic wide. For fault-tolerance, the number

of replicated topics/partitions can be configured by a replication factor. Message persistence is achieved by storing messages directly to the file system. However, there are not always flush operations so the data might still remain in the OS page cache. The producer can not only specify the topic where an event should be published to but also the partition. In addition, it is possible to specify parameters regarding the trade-off of throughput and latency (e.g., batching).

The consumer keeps track of its current offset on a topic (the consumer position). With each request, the current offset is sent to the broker, which then delivers data from that position. In this way, a re-consumption of old events is no problem. At the same time, each consumer is assigned to a consumer group and there will only be one consumer per partition per consumer group. Kafka follows a push-pull strategy, meaning that publishers push their message to the broker, while consumers pull for new messages. A reason for this decision is that a push-mechanism towards the consumer could easily lead to an overload. In contrast, using the pull-mechanism the client has full control of the pull frequency and the consumption rate can be adapted easily.

Messages can contain several records that are embedded in a record batch. Kafka is operating a binary protocol on TCP. Hence, a reliable communication can be assumed on lower layers. Starting from Kafka 0.11, there is an idempotence flag, that lets the broker identify and eliminate duplicates by tracking the producer identifier and a sequence number that is sent along with each message. Transactions are supported and useful, for instance, when sending related data to multiple partitions. Based on that, all three common delivery semantics can be implemented.

It is possible to enable the auto-creation of topics in case a publisher is sending a message to a non-existing topic. A topic name may consist of alphanumeric characters and “.”, “-”, and “_”. It is suggested not to mix “.” and “_”, because they could interfere due to internal mechanisms. As a consequence, the upper layer is required to escape other special chars within topic names. In addition, there is no predefined hierarchy (delimiter) for topic names.

Various SASL authentication mechanisms are supported and as well authentication via TLS/SSL. Similarly, TLS encryption of communication between publishers/-subscribers and brokers can be enabled. Authorization of read and write operations by publishers/subscribers can be realized with Access Control Lists (ACLs) (e.g., “*userA* has access to all topics that start with *simulation.scenario-userA-**”).

3.3.4 Data Serialization

As a communication technology choice is made, the question of how to serialize data arises. The purpose of this is mainly to have a common exchange format that is understood by all communication partners. The most common formats are

either text-based and human-readable such as JSON [39] and XML [223] or binary formats such as Avro [12] and ProtoBuf [92]. While XML is user-friendly, it leads to a huge overhead due to verbose and redundant structure-related elements (e.g., start- and end-tags). JSON is more lightweight and supports basic data typing, whereas in XML typeless data strings need to be parsed. In contrast, the binary formats are more efficient. This does not only concern the size of encoded data but usually also the effort for (de)serialization. Avro and ProtoBuf achieve this efficiency by having schema definitions that describe all types and lengths of fields in byte arrays [220]. What makes Avro and ProtoBuf particularly interesting is a project called *Schema Registry* that provides a management solution for the schema definitions [62]. Typically, an Avro container consists of a header and (multiple) data blocks. The header holds aside from other metadata (e.g., compression codec) the schema definition for the data blocks. If using a schema registry, it is possible to replace the former header by a simple schema identifier, which further reduces the message size. With the identifier reference, all schema related information can be looked up in the registry. In addition, the registry component allows for a central storage of all data structure definitions. This helps to avoid schema mismatches between different components. There are no reasons apparent that would justify a custom tailored serialization protocol over the existing ones. Furthermore, there is good Avro support by Kafka. That is why we will use Avro for (de)serializing data and the schema registry to manage schema definitions.

3.4 Coupling Concept

Having a sound concept for the communication plane, our loose coupling mechanism can be developed on top. Obviously, this mainly addresses the requirements related to the category of coupling, but also of correctness (see Section 3.1).

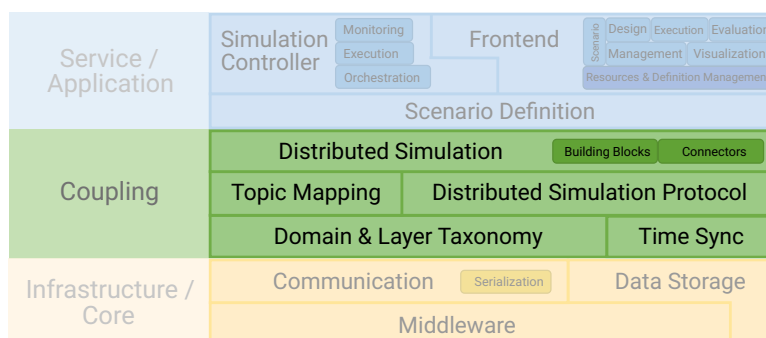


Figure 3.7 – The second layer of the system’s architecture is developed.

3.4.1 Core Idea

Aiming for an approach that works not only for predefined use cases but also for unknown topologies and unknown components represents a real challenge. Solutions that aim for the coupling of a small and static set of known simulators can be realized in a much simpler way. However, that would result in a close coupling. When looking for example at Veins (see Section 2.3.4), the coupling is realized by assigning a master role to one component (OMNeT++). The master calls the API of the second component (SUMO) when necessary (e.g., to proceed to the next time step or to gather information). Such an approach works efficiently for their specific use case of vehicular communication. However, if there is a need for an adjustment of the topology (e.g., splitting the work of the traffic simulation to two SUMO instances) the codebase needs to be modified significantly. Obviously, when aiming not only for distribution features but also for the possibility to extend a setup with additional tools, such an approach clearly reaches its limits.

A more flexible and extendable coupling attempt is therefore necessary. Involved *components* (i.e., primarily simulators, but also physical devices, general software packages, or data sources/sinks) of a composed simulation scenario should be capable of operating on their own and not be depending on external control or orchestration commands to operate. Moreover, it is assumed that a component is a self-contained and executable artifact. A component's instance will be called *Building Block (BB)* in the following.

In addition, a BB should not need to know how many and what other BBs are also involved in a composed simulation run. Given that, it is no longer feasible to have a master component that is orchestrating everything by calling the custom APIs of involved BBs. Instead, the coupling will be solely realized by exchanging predefined data tuples in accordance to a structured rule-based process. Such a decentralized coupling approach does neither require for a global knowledge of all involved data flows nor a central orchestration component.

If the problems of numerous links between the involved BBs of a simulation run is solved by using a middleware-based messaging paradigm, there remains the issue of interfacing between them. There are two basic approaches when interfacing more than two components.

Approach A: It is in the responsibility of the receiver to understand the incoming data. Figure 3.8 depicts that approach. Each component has a module that is capable of receiving and understanding received messages from other languages. The advantage of this approach is that the sender does not need detailed information about receivers.

Approach B: The sender is responsible for using interfaces that are supported by its receivers (see Figure 3.9). In reality, also a mixture might be a suitable solution. This might especially be required if components are involved that cannot be modified (e.g., software that is not under own control, existing data feeds, or hardware components).



Figure 3.8 – The receiver is responsible for decoding.



Figure 3.9 – The sender is responsible for encoding.

For well-known use-cases, creating another common interface (i.e., applied to the example: a common language) that is provided by every component might be a good solution. Since unknown use-cases and extendability with unknown components should explicitly be supported, the latter is no option. As a consequence, when introducing a new component, one has to modify all other components (see Figure 3.10). For the first approach, where the receiver is responsible, all existing components need a new module that is capable of understanding the new component (dotted lines). In addition, the new component needs modules for understanding all other existing components. For the second approach, where the sender is responsible, all existing components need a new module that is capable of using the new component interface (solid lines). In addition, the new component needs modules for using the interfaces of all existing components when sending something.

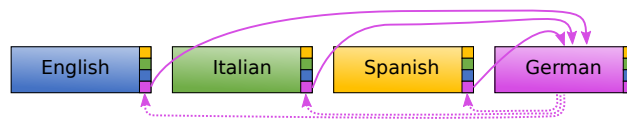


Figure 3.10 – An additional component is causing modifications.

Furthermore, it would be desirable to have an extensive catalog of integrated simulators that allow to cover a vast variety of simulation scenarios. However, this ambition is contradicted if such tremendous integration efforts are required at the same time. Another issue arises from the context of correctness and credibility. As already pointed out in Section 3.2, the connection between submodels is a crucial part of the composed model, especially if information has to be translated for example in terms of filtering or aggregation. If this logic is spread across the code of all components, reproducibility and traceability might get very hard. Therefore, we propose a different design idea: each component can only send and receive information that corresponds to its native data model. Referring to the language example, this

idea can be denoted as *monolinguisism*. Consequently, such an architecture requires some additional mechanism for converting information between heterogeneous components (see Figure 3.11). Modules that provide such a mechanism will be called *connectors*. Connectors are detached modules with no other purpose than transferring data between two different components. Additionally, this eases the reproducibility and traceability of information flows. Following that concept, it is no longer necessary to modify existing components, once a previously unknown component is integrated. Instead, it might be necessary to add new connectors that are capable of mapping the data model of the new component with the existing ones.

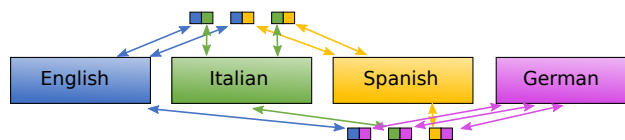


Figure 3.11 – The translation is detached.

In order to further simplify this, a certain degree of abstraction is introduced. Groups of similar components are using a common data model that works stereotypical for the whole group. Such a data model will be called *layer* in the following. As a consequence, there is no need for translating information between such similar components. At the same time, connectors between heterogeneous components are designed at the group-level. This abstraction allows a reasonable trade-off between the total number of different data models (and required connectors) and preserving the main functionalities of individual components.

The decision of having self-centered components does not only affect their communication language. Likewise, the addressing of recipients is also simplified. There is neither a need for managing a knowledge base of interdependencies globally nor a need for local neighbor-discovery-protocols within the components. Exploiting the spatial decoupling that is provided by the topic-based MoM, a BB is producing messages to the data pool and consuming messages from the data pool by following a structured procedure that is described in the following sections. From a BB's perspective, the publishing of messages is not affected by the question if and how many other BBs are consuming its messages. Similarly, a BB does not care about the source of messages that are consumed, which greatly simplifies the ingestion of non-simulation data (such as real-world live data) in a scenario run.

Besides the information flows themselves, another challenge is the partitioning of the scenario, respectively the design of the BB topology. In order to have a common term, we will use the word *entity* to describe elements that are represented by BBs. However, we have to clarify that the concept is by no means limited to

entity-based modeling paradigms, as “entity” is already linked to a certain meaning in the simulation vocabulary. The concept is, for example, also applicable on flow-based models. Our understanding of a component is similar to Verbraeck’s module definition [221] (see Section 2.3.1), with the important difference that in our concept an existing component is not modifiable. While a component can be created by any stakeholder, the internal details do not have to be published. Regarding credibility, we accept the black-box nature of components. If we assume that components are valid models of reference models, we presume that their composition is also valid, if only one instance is responsible for a certain region within a single domain. A region does not necessarily need to be a spatial area, but is rather a set of intra-domain elements that are understood by all components of the domain. Thus, the sets of responsibilities of BBs within a domain are disjoint and therefore it is accepted that neighboring models/components may have a different underlying view of reality. BBs are understood as individual microservices with defined inputs and outputs. Consequently, a single instance of an entity within a scenario will only be simulated by one BB at a time in order to ensure the consistency of the overall model.

The introduced *layer* is not only a plain data model. The elements of the data model are tagged with certain keywords that allow for instance to infer which mutator methods need to be provided by the layer interface. Moreover, the simulation distribution logic (i.e., when to publish what information, what information to consume) is also included. As a result, the coupling mechanisms are defined on a data level and per layer.

3.4.2 Domain & Layer Taxonomy

The following ideas aim to allow for such a coupling by giving the tool for finding abstraction categories. Therefore, we will distinguish components by two levels, their domain and their layer. Based on that it is possible to provide a well-defined and structured description for each component that is used for the coupling. The purpose of the domain-layer taxonomy is to achieve a trade-off between addressing individual (heterogeneous) components within a domain on one side and being able to couple them in a structured way on the other side. In addition, it solves the problem of finding data structures for storing messages in the data pool. While the domain defines the context, the layer is supposed to provide a data model that allows the representation of its stereotypical perception of its domain. If multiple components are assigned to the same layer, the layer data model is typically an intersection of their natural data models that would perfectly represent the perceptions of these components.

Domain

When using the term *domain* in this work, it is not about the mathematical or technical meaning. It is about what the Cambridge Dictionary describes with “an area of interest” [45]. While this still leaves plenty of room for interpretation, some domain examples will do the trick: communication, economy, energy, healthcare, traffic, society, warfare. Each domain addresses a typical group of certain aspects of our lives. A domain is a logical group of related entities and systems that are typically investigated and modeled using domain-specific knowledge. A person that has deep knowledge of entities and their relations within a domain is called a domain expert. That could be a traffic engineer in the traffic domain or a doctor in the healthcare domain. In general, different domains deal with a different set of entities, properties, and relations. In other words, the chances of having similar ontologies for two diverse domains is not very high.

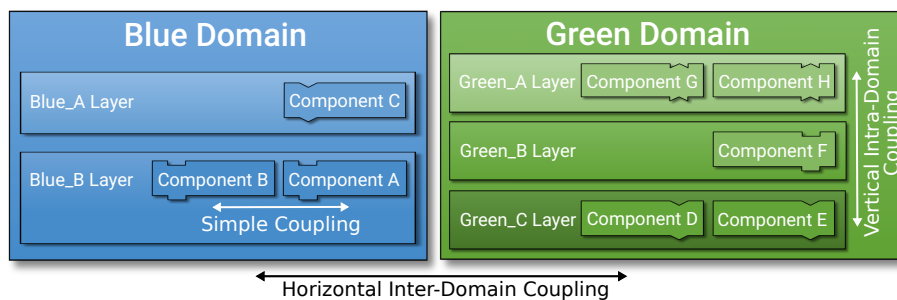


Figure 3.12 – There are different types of couplings.

On the contrary, within a domain there might be a variety of different approaches for describing, investigating, and representing the same phenomena. These approaches might differ in their understanding of the domain, their level of detail, their way of modeling the assumed structures, the coverage, or the desired outcome. If the underlying world-view of each approach would be described by a separate ontology, there is a high probability of having common elements within a single domain. Due to the aim of this work, such a describing approach will mainly be a simulation model but could also be a piece of software, a data source, a data sink, a hardware component, or another composed system.

This aspect has a major impact on the design of the coupling concept. It justifies the assumption that it is reasonable to differentiate between inter-domain and intra-domain couplings. Possible compositions are depicted in Figure 3.12. When exchanging data between different modeling approaches within a domain it makes sense to intend preserving as much information as possible (*vertical intra-domain coupling*). It is conceivable to transfer data information between models without any information loss. This is in particular the case in a special variant of the intra-domain

coupling, where two BBs with identical layers are coupled (*simple coupling*), for example, in order to realize a parallel simulation. In practice, vertically connected models might address different levels of detail and an aggregation or a disaggregation of information is required. But in every case the underlying domain is the same and acts as a common reference. When exchanging data between different models across domains this shared reference is missing. In general, there is no point in trying to comprehensively preserve received information in a cross-domain coupling (*horizontal inter-domain coupling*). In most cases it will simply not be possible. In our system, a domain is defined by a name, a version, and said reference element (see Figure 3.13 and A.1 in the appendix). There are optional fields for a description and further arbitrary elements.

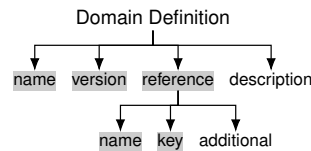


Figure 3.13 – Definition of the domain structure.

Layer

A *layer* belongs exclusively to a single domain. Within that domain, it describes a certain perception of the system in focus. This perception is reflected in a data model that allows representing the specific world-view. As already proclaimed for the relation between layer and domain, each component that is integrated in our simulation system (e.g., a simulator) has to be exclusively assigned to one single layer. Because it is possible that several components have a very similar purpose and perception, multiple different components can be assigned to the same layer.

The stereotypical data model of a layer contains different kinds of information. Typically, there will be rather passive elements that form the system’s environment and rather active entities that are the subject of the domain. The subset of data elements that are holding the representations of the active entities will be called *Native Data Model (NDM)*. Therefore, it represents the heart of each layer. Exclusively these NDM-tuples will be exchanged between BBs to realize simple and vertical couplings (i.e., intra-domain couplings). For the inter-domain couplings, the options are more manifold and allow also connecting other parts of the data models. This is possible as we don’t aspire for inner-domain consistency in that case.

As each domain has its own particularities, it is not possible to apply a simple rule in order to infer a layer’s data model. Even the purpose and the modeling paradigm might influence this design process. Therefore, each new layer needs to be modeled by domain knowledge. The structure of the layer data model, on

the other hand, is defined generally and given below. Typically, a new layer is not designed on a hypothetical level. It is more likely that a new component is going to be integrated in the toolbox and no existing layer is a suitable abstraction. It might be helpful to take a high-level perspective and ask what the model is all about from a more methodological, engineering, or philosophical perspective. Techniques used in the areas of semantics and ontology can be useful here. Questions of the following kind can support the user in this modeling process (see also [100]): what is this layer about, what is the intended purpose, what is the main element type of a typical model. If there are no answers to such questions, the component might cover too much aspects of a system (for complying with the proposed approach). If the component is a simulation model, splitting the model into several submodels might help to get rid of this problem. In contrast, if there is a clear idea of the purpose and the main entity type, an NDM can be created that represents the main entities.

A layer definition that was purely based on the NDM would be very coherent to the underlying idea. However, our layer description is not inevitably restricted to the NDM because that would limit applications. In a second step, the layer data model can therefore be extended by adding data structures for additional relevant entities next to the NDM. Information that is hold by these secondary fields will not be used for the intra-domain coupling, but can be used for inter-domain couplings and further tasks (e.g., interactions or analytics).

If there are multiple new similar components that should be covered, the modeling can be easier. Related components will have a common subset of classes, agents, objects, attributes, or methods. In a bottom-up manner, the components' common subset can serve well in order to establish a baseline for the layer definition. However, developing a layer definition is always a trade-off between generalization and specialization. This applies to the NDM, as well as to the additional fields. Obviously, the absence of an element in a single component cannot be a compelling reason for the exclusion from the layer definition if it clearly fits the layer's stereotypical idea.

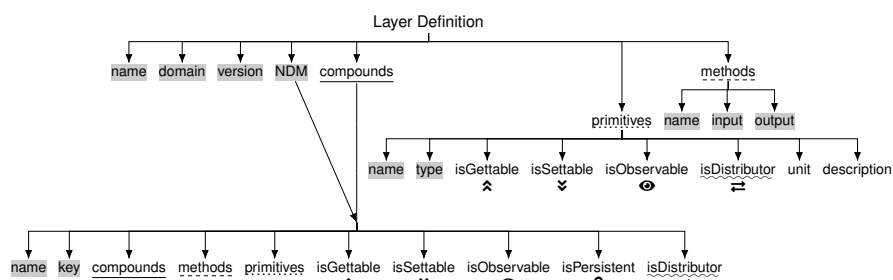


Figure 3.14 – Definition of the layer structure. Gray fields are mandatory.

Figure 3.14 depicts the recursive structure that is used to define layers (see also Appendix A.2). The fields with a gray background are mandatory. Most importantly,

there needs to be a domain, a domain-wide unique name as an identifier, a version number, and finally the NDM. There are three base structures that allow the construction of arbitrarily complex composed structures. It can be seen that the layer is no plain data model, but does also provide information about mutators (i.e., *isGettable*, *isSettable*). The *isPersistent* property specifies if the represented entity is existing during the whole duration of a scenario run or if it can be dynamically created and removed during a run. The *isObservable* property indicates that the element is a candidate for providing status updates during the scenario run, which can for instance be used for analytic tasks. Fundamental for the coupling concept is the *isDistributor* tag, which has to be assigned to exactly one element of a layer definition. Based on the element that is tagged as distributor, a BB can detect if an NDM-tuple has to be published (i.e., information is leaving the responsibility region of the BB) and can consume NDM-tuples (i.e., information is entering the BB's responsibility region). Such a distributor element could either directly represent the responsibility set or have a rather implicit link to the responsibility set. In addition, the layer structure allows the declaration of methods that are more elaborate than simple setter and getter methods.

- **Primitive** attributes consist of a name, of a data type, the *isGettable* property, the *isSettable* property, the *isObservable* property, and the *isDistributor* property. Optionally, there might be information about the unit and a detailed description of the attribute.
- **Methods** are defined by a name and a set of input and output parameters.
- **Compounds** hold a key, potentially primitives, methods, and other compounds, the *isGettable* property, the *isSettable* property, the *isObservable* property, the *isPersistent* property, and the *isDistributor* property. To provide a key is mandatory.

Components and Building Blocks

A *component* is assigned to a single layer and therefore also to a single domain. Instances of a component can participate in a distributed simulation as *Building Blocks (BBs)*. Each component definition consists of a list of mandatory or optional attributes that are used by the instantiating BBs (see Figure 3.15 and A.3 in the appendix). In addition, the definition provides information about required and optional resource types. There is also a set of results, which can be requested to be returned after the scenario run is finished. The BB definition is embedded in the scenario definition (see Section 3.5.1). As it is an instance of a component, it is using the possibilities that are provided by its component definition.

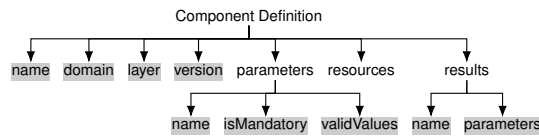


Figure 3.15 – Definition of the component structure.

3.4.3 Deterministic Information Exchange

The workload of the overall simulation scenario is split into several LPs for various reasons (e.g., parallelization or model combination). It might also be possible that an external component is involved. As for most distributed applications, there are certain events that require a communication between the different LPs. The domain-layer taxonomy provides syntactic and semantic definitions for data tuples that can be exchanged. We will now further explain how data is exchanged by using the presented communication concept (see Section 3.3). The exchange of information is exclusively based on the elements of the layer definition and interfaces that are derived from that. At the current point, it is clear how communication works on a technical level and what to communicate. However, in the context of DS, there is not just the requirement that communication between different BBs is possible at all. The communication is required to be reliable and deterministic. In particular, it has to be guaranteed that messages arrive at their consumers at all, that messages arrive on time, and that messages arrive in the correct order. Otherwise, the communication could cause causality violations and with that affect the correctness of the DS. The combination of the underlying TCP protocol and the Kafka acknowledge mechanism will take care of the first problem. This subsection will cover the message ordering and afterwards the issue of timeliness. Finally, the format of messages and the mapping of messages to topics will be clarified. We will follow the common MoM terminology and generously use the term *message*. Such a message is representing an exchanged data tuple, a simulation event, or an API-call between components.

Message Ordering

Regarding the ordering of exchanged messages, one has to take the messaging system into account. In the case of Kafka, there is a global ordering guarantee across brokers and replications, but only within topic partitions. As a consequence, it is not guaranteed that a consumer receives messages in the same order they were published as soon as we are using multiple topics or more than one partition. Moreover, in these cases we can also not guarantee a deterministic consumption order (i.e., that all consumers receive the messages in an identical order - even if this order would not be the same as the publishing order). Obviously, the resulting communication is not deterministic. The following consideration addresses in particular the ordering of

messages that are created simultaneously (in logical time). The problem is illustrated in Figure 3.16. A red message and a black message are published concurrently to two different topics. Consumers can either receive the black message and then the red message or the other way around.

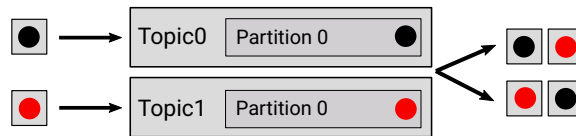


Figure 3.16 – No global ordering across multiple topics. There are two possible receive orders.

Another example is depicted in Figure 3.17. A red message and then a black message is published to a topic that is consisting of two partitions. The sequence of publishing is dismissed at the time the messages are stored in the partitions. Consumers may now first consume the black and then the red message or the other way around.

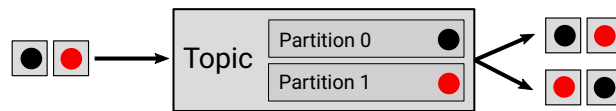


Figure 3.17 – No global ordering across multiple partitions. There are two possible receive orders.

Having only one topic and only one partition as in Figure 3.18, the consumption order would match the publishing order if there is only one publisher.

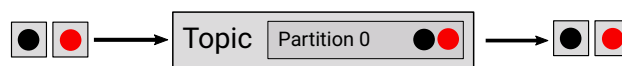


Figure 3.18 – Global ordering within a topic partition.

However, being limited to single topic would contradict the whole idea of the topic-based MoM. Moreover, introducing another concurrent publisher could again bring ambiguity (Figure 3.19). This approach allows at least that all consumers would consume the messages in the same order. Such a behavior would be sufficient in order to reproduce results. Repeatability, on the other hand, is not realizable because the partition-order might differ due to the concurrent publishing.

Consequently, we must add additional logic in order to introduce deterministic behavior. Following the data-centric approach, we want to not only to store information in a central data pool. Furthermore, we want to directly organize data within a hierarchical topic structure that matches the domain-layer taxonomy and

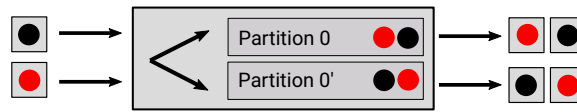


Figure 3.19 – No deterministic appending order. After the messages are appended to the partition, there is a deterministic consumption order for all consumers.

therefore there will be multiple topics. In order to guarantee a deterministic order of processing, a consumer needs to pre-process incoming messages in a buffer (Figure 3.20). As soon as a certain barrier is reached, all received messages are ordered in an identical way across all consumers and are finally ready to be processed in that order. More details on these barriers will be given in the next section. This solves primarily the problem of deterministic consumption. We can guarantee that all consumers have the same view on the ordering of data. When replaying the data from the existing topics, the messages will be again processed in the exact same order.

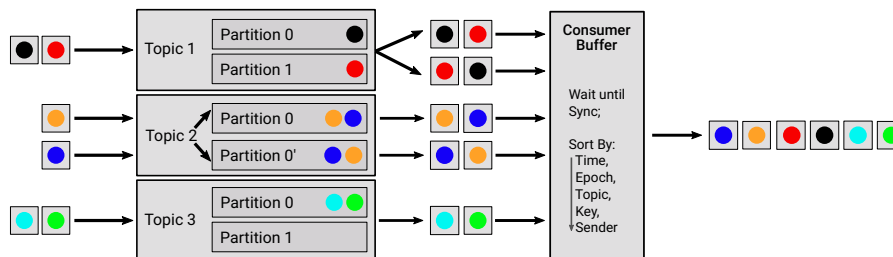


Figure 3.20 – Pre-processing in a buffer.

Additionally, this mechanism does also allow for a deterministic repetition (i.e., re-simulation) of a scenario. In case of concurrent publishing tasks, it does not matter which message is appended first. For illustration, see *Partition 0* and its variant *Partition 0'* of *Topic 2* in Figure 3.20. Both possible outcomes of the partition will lead to the same final processing order on the client side. The drawback of this approach is that a producer has no control of the ordering of instantaneous events. Methodological, this is not seen as a problem, because if two producers are creating a concurrent message there is no correct order per se. In addition to the timestamp, an epoch is also considered. Let us assume, that one producer creates a sequence of messages that are bound to the same timestamp but are nevertheless expected to be in order. If they are published to the same partition they would remain in order, but it is also thinkable that such a message sequence will cover multiple partitions or topics. Therefore, epochs are introduced in the next section, which address the

problem with sub-timestamps. In conclusion, the sorting mechanism has to meet the following conditions to solve the described problems:

- Wait until it is guaranteed that no new messages will arrive for the current logical time (i.e., a synchronization barrier is reached) before releasing the corresponding sorted message list.
- The already existing partition-wide message ordering for individual publishers has to be contained during further sorting.
- With descending importance, messages are sorted by:
 1. Logical timestamp¹
 2. Epoch
 3. Topic
 4. Key²
 5. Sender³

Synchronization

What remains as an open issue is the late receiving of messages, which can cause causality breaks in the overall simulation. Further details on the necessity of a synchronization mechanism in a DS system were given in Section 2.3.2. In this section, we will therefore develop a synchronization mechanism that is suitable for a topic-based coupling based on Apache Kafka. As the overall coupling concept does not rely on any central orchestration component, the synchronization mechanism should also work in a decentralized way and without a dedicated time master. Admittedly, one could argue that the Kafka brokers are a central element per se, but as the brokers are just storing and delivering messages and not actively participating, the term decentralized describes the synchronization mechanism properly.

Although optimistic approaches offer opportunities for great performance gains, an optimistic concept requires a lot of fine tuning and suitable topologies and scenarios for practical benefits. A generalizable conservative mechanism seems therefore more feasible for our flexible and extensible coupling approach. That means that we will divide our coupled simulation run into several episodes⁴. After each episode, we will wait until every BB has received all messages of interest and is ready to proceed with the next simulation step.

Having a message broker with a persistent storage of messages and their ordering makes a significant difference to direct and transient communication between

¹In particular, when BBs with different step lengths are coupled.

²Sorting by keys is more powerful than just sorting by partition ids. However, this requires that partition ids are not set manually, but derived by hashes of the keys as intended by Kafka.

³Required, if there are multiple publishers that use the same message keys on a single topic.

⁴In case of a DES with fixed step lengths, an episode is typically corresponding to a simulation step.

components. Using the aforementioned ordering approach, we can already assure that all components have received all messages of an episode in the same order. However, we want to guarantee this repeatedly during the whole simulation run and not only for one episode. Therefore, we need to provide a mechanism that helps to synchronize all involved BBs after each simulation step. Unless they are in sync, the BBs are not allowed to proceed to a later point in (simulation) time in order to avoid a causality violation. Each synchronization episode acts as a barrier that blocks until all relevant messages of the last step are received and ready to be sorted and processed.

The designed algorithm will be illustrated by an example in the following. As a first step, a BB announces its wish to proceed to a certain time T by publishing to a specific synchronization topic. As soon as all involved BBs have also announced that they want at least proceed to T , the so-called Lower Bound Time Stamp (LBTS) of the composed simulation is T . With that it is possible for the BB to proceed its simulation to T . This straight-forward procedure is depicted for two BBs A and B with different time steps in Figure 3.21. Both start at time t . A announces the wish to proceed to $t + 1$ and afterwards B makes the request to proceed to $t + 2$. This means that all involved BBs want at least to proceed to $t + 1$, which allows A to proceed to that point in time. As a next step, A wants to proceed further and now the LBTS is $t + 2$, which enables both A and B to proceed to $t + 2$.

Synchronizing a virtual clock between multiple BBs without any other information exchange is highly unlikely as we do not synchronize for the sake of synchronization. We need the synchronization to assure that simulation related messages are processed on time. The described behavior would be sufficient for this purpose if the sending

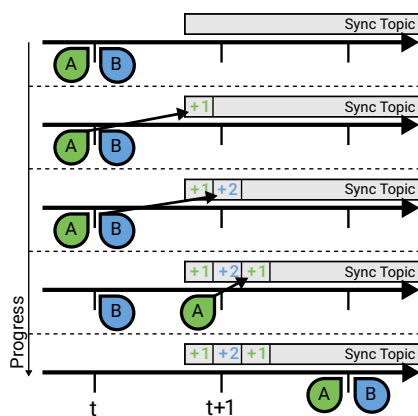


Figure 3.21 – Basic synchronization using a dedicated synchronization topic.

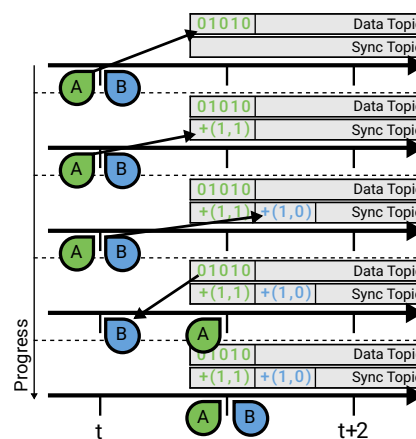


Figure 3.22 – Synchronization with the consideration of message delivery.

components would know if and when sent messages are received by all involved recipients. However, such a design would put additional logic into the components and would require additional communication. Besides, it would contradict the different types of decoupling of MoMs and simply would be impossible with our current design, where the BBs do not even know their recipients.

A solution that matches our chosen design extends the synchronization message. A BB will not only announce its desired future point in simulation time, but also how many messages the BB published to which topics. Following the decentralized idea, a BB makes a request to time T and has to wait for two conditions before being able to process received messages in a sorted way and advance its local clock. First, the LBTS has to become $\geq T$. Second, all announced relevant messages with a timestamp $< T$ have to be consumed. The second condition is crucial because of the lack of a global message ordering across all topics. In addition, another constraint is introduced. By announcing an advance request to time T , no further messages are allowed to be published until the BB has proceeded to T . In other words, it is not possible to publish messages with a simulation timestamp lower than the most recently announced own advance request. This is illustrated in Figure 3.22. The notation is (x, y) , where x symbolizes the time and y the sent messages. Both components A and B start at time t . A publishes a simulation related message on an arbitrary topic with the timestamp of t . Afterwards, A is ready to proceed and announces the newly published message with the wish to advance to $t + 1$. B has no data to publish and wants also to proceed to $t + 1$. Now, A can already proceed because the LBTS condition is met and there are no other announced messages. Because of A 's announcement of one new message, B is aware that it has to wait until receiving and processing A 's messages before being able to proceed to $t + 1$.

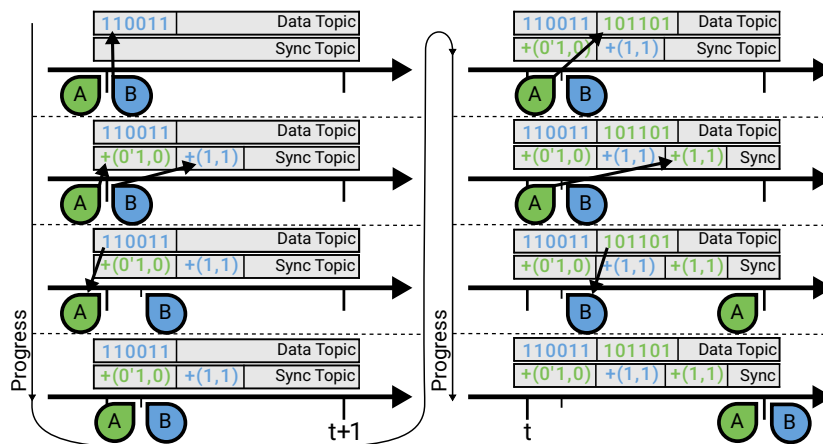


Figure 3.23 – Epochs allow for quasi-simultaneous events.

The mechanism assures that messages are received on time and in a deterministic order by incorporating the previous sorting mechanism. However, in the current state a BB can only respond (i.e., publish another message) to a received message in the next time step. This might be sufficient for many use cases, but there are also cases where an *instantaneous* reaction is necessary. An example is the transformation of a message by a dedicated translation component. Therefore, we will divide a time step into further sub-units, which we call *epochs*. Using epochs, it is possible to create deterministic sequences of messages that are quasi-simultaneous. Regarding the process of time advances, this implies that it is not only possible to announce the wish to proceed to a certain point in logical time, but optionally also to a certain epoch within a point in time. Figure 3.23 is illustrating this extension. The notation is $(x'y)$, where x symbolizes the time and y the epoch. A and B start at time t . B sends a message to a data topic and announces the wish to proceed to $+(1'0, 1)$. A requests to proceed to $+(0'1, 0)$. After processing B 's message, both conditions are met for A and it is possible to proceed. A publishes a response and then announces the sent message within the request to advance to $+(1'0, 1)$. With that, the LBTS is $t + 1$ and A can proceed (B 's message is received and B is not allowed to create new messages before reaching $t + 1$). B has to wait for processing A 's message and is then finally also able to proceed to $t + 1$.

In conclusion, each BB is responsible for consuming a timing topic, announcing the desire to proceed and the publication of messages. Before proceeding, it has to wait for the LBTS and the consumption of all announced relevant messages. The pseudo code for this procedure is given in Algorithm 3.1. The example considered two BBs for simplicity, but the approach works for an arbitrary number of BBs.

```

1: procedure SIMULATIONLOOP
2:   while  $T \neq END$  do
3:     Simulate Step  $T$ 
4:     Publish Simulation Data
5:      $T \leftarrow T + 1$ 
6:     Request Time Advance to  $T$  and Announce Published Messages
7:     while  $(T \neq LBTS) \ \& \ (receivedMsgs \neq announcedMsgs)$  do
8:       Consume Messages and Store in Buffer
9:     end while
10:     $Msgs \leftarrow receivedMsgs$  with Timestamp < LBTS
11:    Sort  $Msgs$ 
12:    Process  $Msgs$ 
13:  end while
14: end procedure

```

Algorithm 3.1 – Synchronization algorithm.

Message Structure

A message consists of the predefined Kafka message header, a custom header, and the payload (see Figure 3.24). The most relevant element in the Kafka header is the key, which is describing the payload and is relevant for assigning a partition. Depending on the Kafka configuration, the wall-clock timestamp is either the *Create Time* or the *Log Append Time*. The custom header provides the logical timestamp, the epoch of the message, and the sender identifier. The payload will be serialized via Avro and is therefore preceded by an Avro magic byte and the schema identifier of the used Avro schema. The schema is registered under that identifier in the schema registry. For debugging, the serialization of the payload as a plain JSON UTF-8 string will also be supported by our system for most message types.

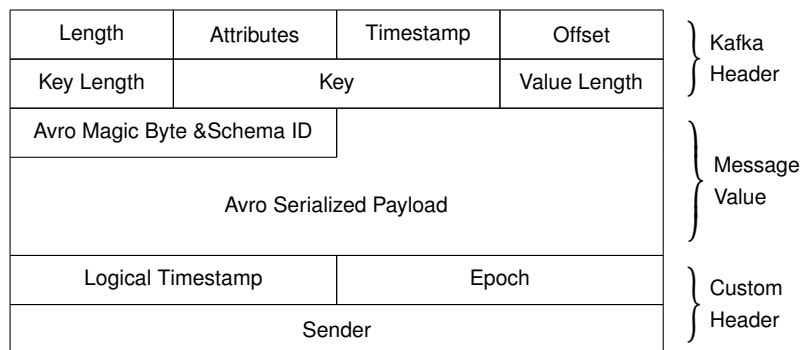


Figure 3.24 – The structure of messages.

Communication Channels and Data Mapping

A topic usually represents a group of coherent data, which suits message-based communication protocols. Likewise, there is typically no address-based communication protocol when using a topic-based publish/subscribe system. The topic can be for example a description of the information itself, the location where the information was gathered, or the purpose that the information is used for. In this manner, multiple consumers can consume information from a single topic after announcing their interest by subscribing to it. Similarly, multiple producers can usually produce information on a single topic and a single consumer can subscribe to multiple topics. In our distributed simulation system, there will be three different kinds of information flows. They will be referred to as *channels*.

1. Orchestration data:

Exchanged information is used to manage the execution of a simulation scenario. That involves the initial simulation request by the user, which contains

all data that define the scenario, control commands, but also the previously designed synchronization messages by the BBs.

2. Provision of the data model:

The composition of all layer definitions of involved components represents the data model of the overall scenario. According to the properties, data from this model is provided in a rule-based way (cyclic or event-triggered). While event-triggered publications of NDMs build the main mechanism for intra-domain couplings, it is possible to define additional scenario rules. An example is the cyclic publication of a certain attribute for an analytic task. In addition, this channel covers also the exchange of resources and final results.

3. Interaction data:

Interactions are used to realize more sophisticated scenarios by enabling modifications to a running base scenario such as creating, modifying, or removing entities. The set of available interaction methods are again exclusively based on the mutators and methods that are provided by the layer definitions.

Every kind of information flow has its individual purpose, characteristics, and requirements. There will be several topics related to each kind. Therefore, the three logical groups represent three different channels, which will form the first hierarchy level for topic names (Topic 3.1). The dot character will serve as a level separator. The logical distinction on the channel level allows the use of separate consumers and producers for different kinds of problems. In addition, the channels are not only used internally, but also provide the interface for connecting external components (see Figure 3.25).

In the orchestration channel, there will be a main topic that is used to trigger new scenario runs (Topic 3.2). More information on the initiation of scenarios will follow in Section 3.5. For each scenario, there will be a topic to transfer control commands (e.g., “pause”, “terminate”), to provide status updates (e.g., “waiting for resources”), and to exchange synchronization messages.



Figure 3.25 – There are three different logical channels that are also accessible for external stakeholders.

[orchestration|provision|interaction].[...] (3.1)

orchestration (3.2)

orchestration.[sceID].ctrl (3.3)

orchestration.[sceID].status (3.4)

orchestration.[sceID].sync (3.5)

The purpose of the provision channel is to reflect the data model of a distributed simulation scenario. Besides, there are several scenario-related topics that are always available. They are used to provide initial resources (Topic 3.6), final results (Topic 3.7), and the definition of the scenario (Topic 3.8). During the scenario execution, the provision channel is populated with simulation data that complies to the composed scenario data model. As already mentioned, this involves messages of the distribution logic and optionally also additional messages that are created based on the scenario configuration. Such additional messages will be produced at fixed cycles (e.g., “publish the speeds of all vehicles every 1000 ms”) or on events (e.g., “publish the vehicle id if its speed exceeds 50 km/h”). These rules will be implemented by so called *observers* within the components. The hierarchical scenario data model is a combination of all involved layer definitions. Its contents are directly mapped to topic names by appending the element names to the channel name and a scenario id (Topic 3.9). The third level describes the domain and the fourth level represents the layer. Starting from the fifth level, the heterogeneous structures of the different layers begin (Topic 3.10). If an entity is tagged as *persistent*, the entity’s key will also be included in the topic (e.g., Topic 3.11). From a performance perspective, only topics for elements that will be published during the scenario execution will be created (e.g., Topic 3.12). Access to data on the provision channel is read-only and limited to the pre-defined element properties. With that, there is little overhead for the producing components and it is in line with the decoupled design. However, as a drawback, there is no possibility of spontaneously exchanging data at unpredictable events in a request-reply pattern.

provision.[sceID].resource (3.6)

provision.[sceID].result (3.7)

provision.[sceID].scenario (3.8)

provision.[sceID].[domain].[layer].[layer data model] (3.9)

provision.[sceID].[domain].[layer].[compound.[...] |primitive] (3.10)

provision.sce1.traffic.micro.edge.edge1.speed (3.11)

provision.sce1.traffic.micro.vehicle.speed (3.12)

That issue is covered by the interaction channel. It allows for modifications of the simulation state by triggering API calls within BBs on received messages. Not all communication in the interaction channel will match the request-reply pattern. It is also possible to invoke methods without any return message (e.g., “turn off the traffic light”). Because it is possible to declare custom methods within the layer definition, the availability of methods is depending on the layers that are involved in a scenario run. Apart from the custom methods, there are also four kinds of methods that are automatically inferred from the compound and primitive elements of the layer definitions. The first two are representing setter and getter methods, while the other two are related to the *isPersistent* flag. If an entity class is not marked as persistent, there are methods to add or remove instances of said kind (e.g., “add a new vehicle”).

```
interaction.[sceID].[domain].[layer].[BB].[compound.[...]|method] (3.13)
```

```
interaction.[sceID].[domain].[layer].[compound.[...]|method] (3.14)
```

```
interaction (3.15)
```

```
interaction.[sceID].[domain].[layer] (3.16)
```

```
interaction.[sceID].[domain].[layer].request (3.17)
```

```
interaction.[sceID].[domain].[layer].response (3.18)
```

In contrast to the provision channel, the scopes that embrace a method will not be completely mapped to the topic name. Incorporating the full path and the instance identifier in the topic (Topic 3.13) would be efficient for consumers. However, a caller would need to know the instance that is responsible for invoking a certain method, which contradicts our main idea. Topic 3.14 solves the aforementioned problem of resolving responsibilities. The downside is that a caller would need to synchronize after each call to different topics, if the ordering of its actions matters (e.g., “add entity x and then modify entity x” vs. “modify entity and then add entity x”). This is caused by the ordering guarantees that are restricted to topic partitions. A single topic for all interactions on the other hand would help regarding the overall ordering (Topic 3.15), but would increase the effort for all consumers tremendously. A lot of consumed messages would be filtered out and dropped on consumer-side. As the ratio of relevant messages would drop, so would the communication efficiency. Therefore, we chose the trade-off between a single (ordered) interaction topic and more specific topics and use the schema of Topic 3.16. Consequently, this is realized with Topic 3.17 and Topic 3.18 in order to further increase efficiency. As a result, the messages in the interaction channel will also contain a specific method identifier and a call id, besides parameters or return values.

3.4.4 Coupling Protocol

Based on the previous sections, we develop the coupling protocol itself. First, the Building Block is characterized and afterwards, the logic of a coupled simulation is described.

Properties of Building Blocks

A BB is an instance of a component that can be flexibly coupled with other BBs. In order to make this work, there are a couple of properties that each BB has to satisfy. A BB is the instance of one single component and is therefore assigned to a single layer. The provided interface of the BB will exclusively reflect its layer. Components need to support the mechanism for providing data on the provision channel and filter and delegate incoming interaction messages to native methods that are implementing the corresponding functionality (e.g., returning a certain value). The major impact of the domain-layer abstraction is that it is very likely that there are certain elements (e.g., entities, attributes, functionalities) within the embraced model of a specific component that are not covered by the layer definition. These uncovered elements will be completely hidden and are not accessible via the abstracted interfaces anymore. BBs comply to the following three conditions in order to realize an extendable coupled simulation.

1. Each BB is assigned a set of responsibilities. The responsibility assignments of a scenario are **disjoint** within a domain. Each domain has a predefined reference element class that is used for the responsibility sets. These classes will typically be of a general spatial nature (e.g., coordinates), domain specific elements (e.g., ids), or a mixture.
2. Each BB acts **neighborhood agnostic**. As there is no known recipient of outgoing information flows, a direct connection between sending and receiving instance is not possible. Based on the responsibility set and the layer definition (mainly the NDM and the distributor element), the corresponding topics and rules are inferred for exchanging simulation data. According to custom scenario-wide defined observers, additional information might be published.
3. Each BB is **monolingualistic**, meaning only able to understand and speak its own native language. This mother tongue will correspond to the elements of the own layer definition.

A user can add an arbitrary number of observers to a scenario definition. An observer is characterized by several parameters.

- **Element:** The element to be observed by its absolute path in the layer definition. This is only applicable on elements that are tagged as *isObservable*.

- **Filter:** An optional filter that is applied on the elements' keys.
- **Period:** The cycle duration after which the observation is repeated. This will be typically the simulation step size or a multiple of that.
- **Task:** An action that is performed on the element when it is time to act (typically “publish”).
- **Trigger:** A condition that is evaluated and decisive for taking action.
- **Type:** The serialization mode that should be used.

Simulation Loop

In addition to all structural definitions, it is necessary to define the behavior of components. By that, the behavior of the wrapper that forms a building block by embracing a model (or similar) is meant and not the behavior of the model itself. While the simulation scenario is running, each BB is iterating over the activity loop that is presented in Figure 3.26. The coupling in our system is realized by exchanging data at discrete communication points in simulation time. However, that does not exclude components that for instance embrace a time-continuous model. The wrappers of affected components need to provide a mapping between the communication points and their internals (e.g., by sampling or aggregating over time windows).

Before the simulation loop is entered, the BB subscribes to all topics from the provision channel that are matching its own responsibility set and represent the distributor element (e.g., Topic 3.19). Each synchronized BB subscribes to the synchronization topic to get notified about pending messages and the progress of other participants that are involved in the synchronization process. In order to announce its presence to others a join message is published to said topic.

```
provision.sce2.traffic.micro.edge.edge23.vehicles (3.19)
```

Each iteration begins with the computation of the next simulation state. The new state is processed. Using the element that is tagged as *isDistributor* (see Section 3.4.2) it is checked whether an entity of the NDM is leaving the region of responsibility. Therefore, every component has to implement an appropriate strategy that checks if entities are outside of the responsibility region. In such a case, the corresponding NDM is published and the entity is either instantly removed or marked as a ghost (i.e., converted to a passive representation that will be removed at a later point in time). After all messages are published, their quantities are announced together with the logical time of the next desired synchronization/communication point on the timing topic in the orchestration channel.

Then, the consuming part of the iteration begins. (However, messages are constantly consumed and buffered for obvious reasons.) Based on incoming synchronization messages, the LBTS and the number of expected messages is continuously updated. As soon as the LBTS equals the desired new simulation time and all announced messages are received, the BB identifies that it is safe to proceed. Buffered messages are filtered, sorted (as previously described), and are processed. A typical result will be the integration of an entity that moved in the BB's region of responsibility or the invocation of an interaction. Finally, the updated simulation state is considered valid and the configured observers are executed.

Afterwards, the loop continues at the beginning unless the scenario end time is reached or the termination is triggered. Finally, the timing process is left by publishing a quit message and other post-simulation tasks (e.g., provide final results) are performed.

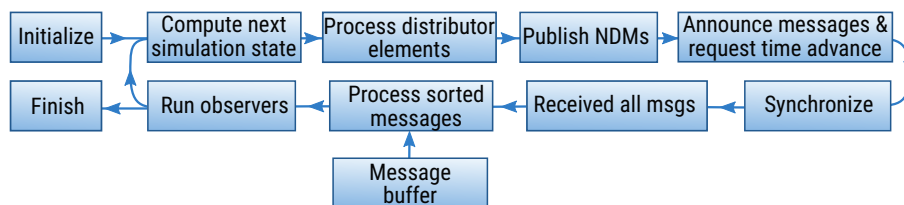


Figure 3.26 – The main simulation loop.

3.4.5 Composing Building Blocks

The described protocol is already sufficient to distribute a simulation scenario among BBs of the same layer. In this subsection, we illustrate this and further explain the mechanisms that enable the coupling of BBs of different layers within the same domain and also of BBs of different domains (see Figure 3.27). From a modeling perspective, these three cases address different kinds of conceptual connections. The first aims to connect consistent views and therefore creates consistent representations of transferred entities. The second is not so strict. However, the goal is still to preserve as much information as possible and minimize the error between the representations of connected components, although there is awareness that this is only possible to a certain extent. For the third case, it is clear that there will be no consistent representation in general. In addition, bidirectional information transfer is not the usual case. As a preliminary step, we will address the partitioning.

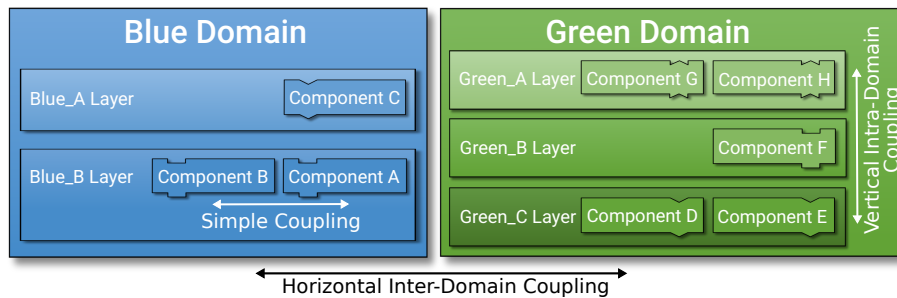


Figure 3.27 – The different types of coupling.

Connecting Similar Building Blocks

For the purpose of illustration, Figure 3.28 shows two different models of an exemplary scenario in an artificial domain called *DomainBlue*. Entities move around the world. The *Blue_A* layer is a more detailed approach, while the *Blue_B* layer uses an aggregated representation. The *Blue_A* NDM represents single entities and the *Blue_B* NDM groups of entities. The domain reference will be a cell as the *blue* world is divided into several cells.

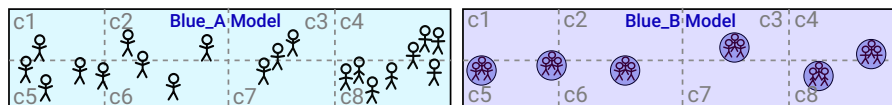


Figure 3.28 – Blue_A and Blue_B model of the base scenario.

As described, the coupling approach is primarily based on exchanging NDM tuples that represent a subset of the simulation state. In the simplest case that means either connecting multiple instances of the same component (i.e., classical parallel distributed simulation) or connecting instances of different components that are still of the same layer. Both senders and receivers are using the same NDM structure. When transferring an entity, the published NDM can be processed natively by the receiver. Therefore, there is no room for interpretation or the need for a conversion in order to process the incoming data. The purpose of coupling BBs of the same layer is usually done in order to split the total workload over multiple instances, for instance to reduce the overall execution time.

The example scenario is partitioned into two spatial regions (Figure 3.29) that are assigned to two BBs (*InstA* and *InstB*) of a *Blue_A* component by setting their responsibilities to $\{c1, c2, c5, c6\}$ and $\{c3, c4, c7, c8\}$, respectively. In case an entity leaves one region (e.g., the red entity is moving from cell *c2* to cell *c3*), the entity is transferred simply by publishing and consuming a *Blue_A* NDM tuple that reflects all necessary information for representing the entity. That implies in particular that

entity identifiers are preserved across BBs, so that a user can track and analyze an entity scenario-wide. If there are more than two BBs of the same layer involved, the distribution mechanism can also be applied without any modification.

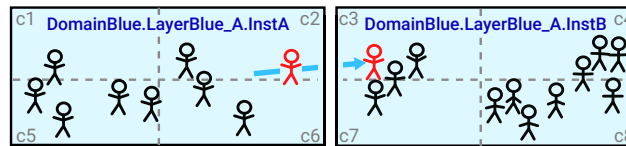


Figure 3.29 – Two BBs of the same layer share the work.

The information flow would look like the following. Let us assume that the *persons* attribute of a cell is tagged as *isDistributor*. As the responsibility set of *InstB* consists of the grid cells {c3, c4, c7, c8}, the BB subscribes to the following topics, in order to receive all necessary information.

```
provision.simpleExample.blue.blueA.cell.(c3|c4|c7|c8).persons (3.20)
```

Accordingly, *InstA*'s responsibility region consists of cells c1, c2, c5, and c6. As soon as the red entity is moving outside the responsibility set (i.e., entering cell c3), *InstA* identifies that the entity is leaving its own scope. This is realized by detecting the entity's appearance in a distributor element (*cell.c3.persons*) that is not linked to the own responsibilities. As a consequence, *InstA* is publishing the entity's current state in form of the NDM before removing it from the local simulation state. Using the entity's new cell, the NDM is published to Topic 3.21. From there it will be consumed by *InstB*, as the topic matches *InstB*'s subscriptions.

```
provision.simpleExample.blue.blueA.cell.c3.persons (3.21)
```

This basic logic can be extended by further conditions and feedback loops. As arbitrary adaptations are thinkable, an example for an extension is used in Chapter 5.

Connecting Layers by Translation

However, it is obviously not sufficient to couple only similar components in order to meet our requirements of a flexible and extendable simulation system. If a composed model consists of BBs that belong to different layers of a domain, further actions have to be taken as their information flows are not connected at the current point. The NDMs of the involved layers will differ and therefore a conversion between the data models of the sending and the receiving BB is required. If both layers belong to the same domain, this conversion will be called a *translation* that aims for preserving as much information as possible and minimizing the error between the different

representations. Again, preserving entity identifiers is aspired if possible. Typically, the various layers of a domain will have a different level of detail. As a consequence, the complexity of a translation will not be symmetrical. One direction of a translation process represents an aggregation, while in the other direction the incoming data might need to be disaggregated (i.e., enriched with information that was acquired or sampled in parallel). Obviously, the aggregation is usually easier and provides less room for interpretations and modeling decisions.

As pointed out in Section 3.4.1, the core idea of having a flexible simulation system consists in having detached translators, so that a component does not have to bother about any other interface and data model than its own. Existing components do not need to be modified once a new component is integrated in the simulation system.

A translation component will connect the data models of two different layers L_A (i.e., aggregated layer, the layer with a lower level of detail) and L_D (i.e., disaggregated layer, the layer with a higher level of detail) of the same domain. The translation itself will convert the data models of these two by providing an aggregation function f_a and a disaggregation function f_d . The input tuple of f_a has to be the output of f_d and vice versa.

$$f_a : NDM_{L_D} \rightarrow NDM_{L_A}, \quad f_d : NDM_{L_A} \rightarrow NDM_{L_D} \quad (3.22)$$

In other words, the NDM tuple of one layer can be converted to an NDM tuple of the other layer. Usually, the aggregation function f_a is not invertible. In algebraic terms, the functions would need to provide a surjective mapping (i.e., elements of the input set can be ignored, but all elements of the output set need to be mapped). Therefore, a translation component T is defined by two layers and the two related translation functions as depicted in Figure 3.30.

$$T = (L_A, L_D, f_a, f_d) \quad (3.23)$$

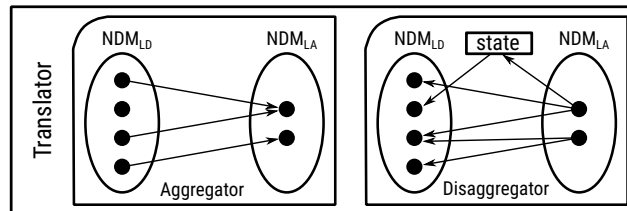


Figure 3.30 – A translator provides two individual conversion functions.

The implemented functions can be rather simple or use an internal state for sophisticated sampling. Generating an output does not need to happen for every input tuple (input-triggered), but can also happen state-triggered or time-triggered. The first mode indicates that the multiplicity of NDMs is similar in both layers (i.e., for each NDM input tuple there will be exactly one NDM output tuple). The second implies the opposite. The destination layer might model groups of entities. As soon as a certain amount of NDMs is received an output is generated. The last one is suitable if the input NDM is describing continuous processes, which need to be sampled or aggregated until a communication point is reached.

There is no exclusive association between a layer and a single translator, which means that it is possible to define several varying translators for the same two layers (Equation 3.24). In addition, the connection of one layer is not limited to only one other layer. There can be multiple translators that connect the layer with different other layers (Equation 3.25). Lastly, it is feasible to stack multiple translators (Equation 3.26) in order to compose new layer connections based on existing translators.

$$T_1 = (L_1, L_2, f_1, f_2), \quad T_2 = (L_1, L_2, f_3, f_4) \quad (3.24)$$

$$T_3 = (L_2, L_3, f_5, f_6), \quad T_4 = (L_1, L_3, f_7, f_8) \quad (3.25)$$

$$T_5 = T_1 \times T_3 = (L_1, L_3, f_1 \times f_5, f_6 \times f_2) \quad (3.26)$$

As an illustration, the base scenario is partitioned into a region for a BB of the Blue_A layer and another region for a BB of the Blue_B layer (see Figure 3.31). Such a setup could be desirable to achieve a performance gain by only representing certain areas of interest with a higher detailed model. The layers of both BBs differ and so do their NDMs. The presented translation is applied in order to establish the information flow between the two instances. We assume that the responsibility set for the right region consists of $\{c3, c4, c7, c8\}$ and that the distributor element of the Blue_B layer is *cell.groups*. *InstB* will subscribe to Topic 3.27. The left part is identical to the example from the previous example in Figure 3.29, resulting in the identical publication of a Blue_A NDM by *InstA* to Topic 3.28.

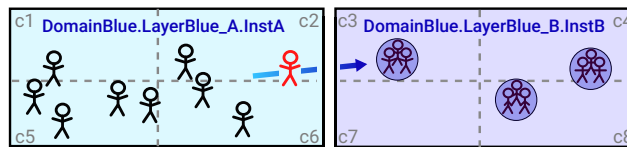


Figure 3.31 – Two instances of different layers share the work.

In contrast to the previous example, Topic 3.28 does not match the subscriptions of *InstB* (by intention, as their NDMs differ). However, Topic 3.28 does match the subscriptions of the translator's aggregation function (Topic 3.29). Consequently, the tuple is consumed by the translator and then translated and published on Topic 3.30, where it is finally consumed by *InstB*.

```
provision.transExample.blue.blueB.cell.(c3|c4|c7|c8).groups (3.27)
```

```
provision.transExample.blue.blueA.cell.c3.persons (3.28)
```

```
provision.transExample.blue.blueA.cell.(c3|c4|c7|c8).persons (3.29)
```

```
provision.transExample.blue.blueB.cell.c3.groups (3.30)
```

To conclude, in the case of connecting different layers within a domain, the coupling is again realized by exchanging (translated) NDM tuples. This is possible, because all layers within a domain are models of the same system, which is partitioned in disjunctive regions. The translation functionality is not limited to simulators, but can also be used to bridge data gaps in general (e.g., an available data set from another layer is translated and then consumed by a certain BB).

Connecting Domains by Projection

The translation opens up for a variety of new applications, but a typical cross-domain co-simulation is still not covered. The main motivation for composed simulation topologies, which involve instances of different domains, is by no means to split work for enhancing performance. The connection type of a *projection* does fundamentally differ from the simple connection and the translation. The primary purpose is clearly the generation of new functionality by composing different models. With that the consideration of responsibility regions is not necessary, if coupling two instances from different domains. Therefore, also the transfer of NDMs between BBs will not be utilized for solving this problem.

In contrast to the two previous cases, there is an awareness that it might be neither useful nor possible to have uniform and consistent representations of the simulation scenario in BBs of different domains. Based on that, connected BBs of different domains are permitted to cover identical regions, as the exclusive partitioning applies only within a domain. A projection will not preserve as much information as possible, but may use a subset of a layer's elements by purpose. Different domains will in general be so heterogeneous that a more specific rule regarding the mapping between data models is not useful. Consequently, the inputs for a projection do not have to originate from the NDMs, but can rather consist of all layer elements that can be published by an observer. Establishing the coupling will happen by a projector component that is consuming such a data tuple. In a following step, the

projector is either providing tuples that correspond to the receiving layer's NDM or initiating method calls via the interaction channel (e.g., incorporating the projected data into the receiving layer's environment). Exemplary projectors are depicted in Figure 3.32. As for the translation, the actions can be input-triggered, state-triggered or time-triggered.

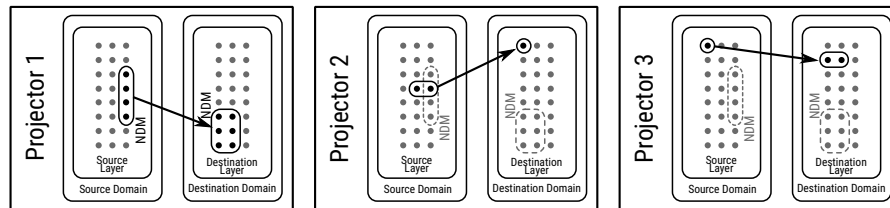


Figure 3.32 – Three examples for projectors that are connecting different layers.

In comparison to the translation, a projection is a unidirectional function, since a unidirectional information flow between certain layer elements is rather typical between domains. One could argue that the projection loosens up the data-centric thought and the point of restricting each layer to a single class of stereotypical entities (i.e., the NDM). However, opening up for these cross-domain functionalities is necessary in order to provide a certain amount of flexibility regarding the coupling topologies and a variety of use cases. As the projectors are still operating on the layer definition, the key features still apply, such as plug-and-play capabilities, reusability, and a modular design. Therefore, projections are consistent with the overall idea.

As an example, contents of the *DomainBlue* are projected into another domain (see Figure 3.33). The projector component is consuming (observable) information about the entities of all *Blue_A* cells (Topic 3.31). In order to ingest the data into the environment of *DomainAnotherExample*, API calls are invoked via messages that are published to Topic 3.32.

```
provision.projExample.blue.blueA.cell.*.persons (3.31)
```

```
interaction.projExample.anotherExample.foo.request (3.32)
```

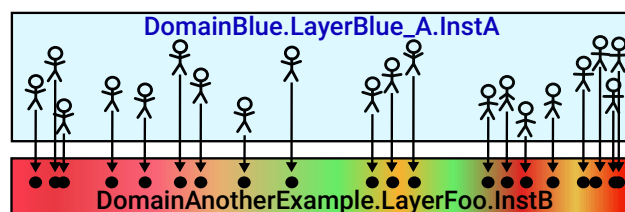


Figure 3.33 – A projector example is ingesting data into another domain.

3.5 MSaaS Concept

The concepts from Section 3.3 and Section 3.4 do already satisfy many of the specified requirements and lay a good foundation for further applications. However, using the concept for realizing a DS does require manual implementation work at this point, although a ready-to-use simulation system with a low entry barrier is aspired. Therefore, this section deals with the application layer at the top of the overall system (see Figure 3.34).

Fundamental information on MSaaS was given in Section 2.3.5. A service character is mainly characterized by the ability to compose models or components (i.e., design scenarios), trigger the simulation of scenarios, and evaluate results from the simulation. All of this should be covered by our service, without the need for the user to have the involved simulators and other components locally installed. The execution should be performed on one or more nodes that can be geographically separated from each other and from the user. Moreover, the service should provide possibilities to manage domain/layer definitions, components, and resources. In the following, the different aspects are described in more detail. Finally, the system architecture that integrates all developed blocks into a single MSaaS solution is presented.

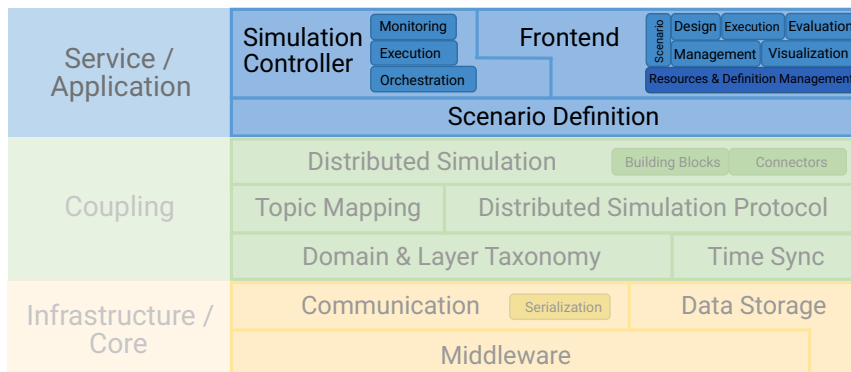


Figure 3.34 – The application layer is addressed in this section.

3.5.1 Definition and Design of Scenarios

The MSaaS layer should support the design of simulation scenarios by composing different BBs. In addition, there should be the ability to adjust the parameters of BBs. This involves computation related parameters (such as simulation step length), execution related parameters (observers), as well as customized input resources and desired result outputs. Apart from that, there are scenario wide parameters (e.g., simulation end time) and connectors that realize the information flow between

BBs. Thus, a scenario is characterized by global parameters and a topology of BBs, connectors, and their parameter sets.

The design of a scenario should be possible in two ways. First, by providing a text-based definition of the scenario. This allows automatizing the creation and a script-based execution of scenarios, reusing (potentially modified) existing scenarios, or simply creating new scenarios in a fast way without any overhead. Second, a graphical user interface should guide and support a user to create new scenarios from scratch (i.a., by providing drag and drop functionalities for adding BBs and connectors from a component catalog). In order to store and transfer a scenario, a defining text-based structure is designed and specified in the following. Corresponding files will be called Scenario Definition Files (SCEs), which can be created and modified via the graphical user interface as well as directly by the user.

Scenario Definition File

All information that is necessary to define and (re)run a scenario is embedded in the scenario definition file. Its structure is depicted in Figure 3.35 and provided in A.4 in the appendix. A scenario is defined as a set of BBs and parameters that describe a simulation experiment. Each scenario has a unique identifier and will be executed once, which will be referred to as scenario execution. The scenario identifier allows for differentiation between different runs.

There are four different sections in a SCE. Scenario-wide parameters are set in a general section. The *scenarioID* is very important as it is incorporated in every scenario related topic. This allows for running multiple scenarios in parallel and the retrospective access to historical runs. The simulation time frame, provides a start and end time, that is used by the instances. The *domainReference* files provide information that is used to organize the topologies of the involved domains. With the *execution* block it is possible to define a global random seed in order to achieve repeatable results and set the number of components that take part in the time sync process. In addition, it is possible to define the priority of the execution and other constraints such as that the execution should run in real-time mode (i.e., synced to wall-clock time).

The second part consists of the involved BBs and their instance-specific parameters. Each BB is defined by providing an *instanceID* and information that is related to its component (i.e., the *type* of the component, the *layer*, and the *domain*). In addition, component-specific *parameters* are given, as well as references to required *resources* and desired *results*. Related to the scenario topology, there is a list of *responsibilities*. The *synchronized* field allows defining the participation in the synchronization mechanism (e.g., for a typical simulator). It is also possible to ignore the time synchronization mechanism, which can be useful if, for example, just plotting

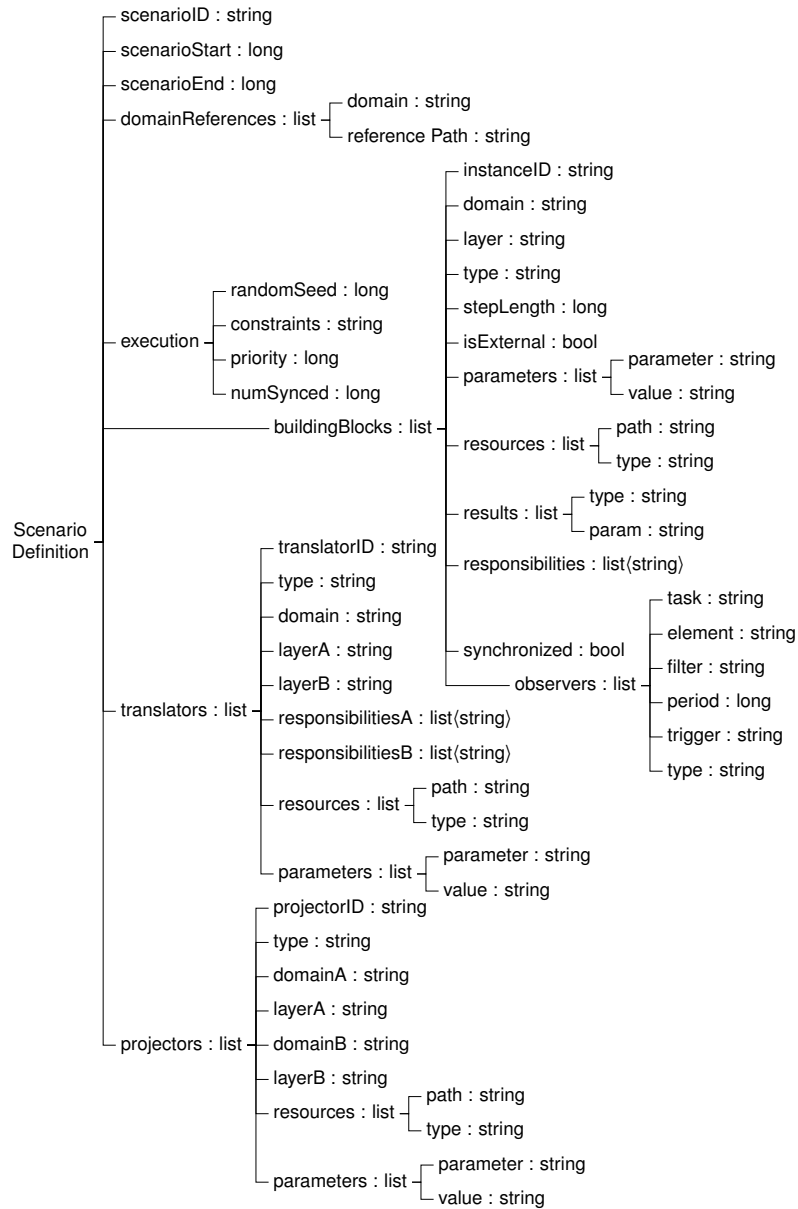


Figure 3.35 – The structure of the scenario definition file.

recent updates. Moreover, the instance can be tagged to be *internal or external*. In the external case, the instance is not instantiated and orchestrated by the simulation controller although being a part of the scenario. As a consequence, external instances have to be executed by some external party (e.g., a component that will not be disclosed by some stakeholder and is operated on their site). Finally, it is possible to add entries to the *observer* list in order to let the instance publish certain elements of the layer data model on topics in the provision channel. Observers can be cyclic (i.e., “publish a certain attribute every 100 ms of simulation time”) or event-based (i.e., “publish a certain attribute when threshold x is reached”).

The third and fourth part is about *translators* and *projectors*. Their parameterizations and responsibilities are given in separate blocks. Again ids, types, domains, and layers are provided. They are complemented with resources and custom parameters.

Scenario Design Rules

In order to be able to execute designed scenarios, there are some rules that need to be followed when populating a scenario description file.

1. The scenario identifier must be either empty or unique (i.e., not already present in the data pool). If empty, the simulation service will assign a unique scenario identifier.
2. The time frame needs to consist of non-negative numbers, provided in seconds.
3. There needs to be a domainReference file for each involved domain.
4. There needs to be at least one BB.
 - (a) The BB’s instance id must be unique within the scenario.
 - (b) Domain, layer, and type of the BB must be known definitions.
 - (c) All mandatory parameters for the component have to be provided.
 - (d) All given parameter values must lie within the valid value ranges.
 - (e) All mandatory resources for the component have to be provided.
 - (f) All given resources must have valid types and exist.
 - (g) All desired results have to be valid.
 - (h) The responsibility set is a subset of the elements of the reference domain file. The responsibility sets of all BBs of the same domain are distinct within the scenario definition.
5. Translators are optional, even if instances of different layers are involved.
 - (a) The translator id must be unique within the scenario.
 - (b) The domains and the translator type have to be registered definitions.

- (c) The input and output layer must differ, be registered, and be assigned to the given domain.
- 6. Projectors are optional, even if instances of different domains are involved.
 - (a) The projector id must be unique within the scenario.
 - (b) The projector type has to be registered.
 - (c) The input and output layer must be registered and be assigned to the given input and output domain, respectively.

3.5.2 Execution of Scenarios

In addition to the design of scenarios, the MSaaS layer is responsible for executing a designed scenario. Therefore, there will be a **simulation controller** component, which is responsible for instantiating involved BBs and orchestrating the scenario run. The developed scenario definition is used as an input for the simulation controller. Following the data-centric idea and aiming for a slim architecture, all communication is still realized via Kafka messages. Messages related to the execution are using the orchestration channel and the provision channel.

The simulation controller builds the heart of the service and fulfills several tasks. It will continuously wait for incoming requests on Topic 3.33. A validation of incoming requests is performed, before a scenario execution is triggered. In that case, internal BBs are instantiated in various ways (locally or on remote nodes, using native binaries or docker containers). The acceptance, the revocation, and the termination of a scenario run are published by the controller on Topic 3.34 along with other general status updates. Such information can, for instance, be used by third parties in order to instantiate external BBs and let them join the scenario run. The simulation controller and all BBs listen to control commands on Topic 3.35 (e.g., “pause scenario run”). The BBs publish status updates (e.g., “waiting for resources”) on Topic 3.36 and use Topic 3.37 to exchange synchronization related information. Announced resources are consumed by the BBs on Topic 3.39. After finishing, they publish requested results on Topic 3.40.

<code>orchestration</code>	(3.33)
<code>orchestration.status</code>	(3.34)
<code>orchestration.[sceID].ctrl</code>	(3.35)
<code>orchestration.[sceID].status</code>	(3.36)
<code>orchestration.[sceID].sync</code>	(3.37)
<code>provision.[sceID].scenario</code>	(3.38)
<code>provision.[sceID].resources</code>	(3.39)
<code>provision.[sceID].results</code>	(3.40)

More precisely, an execution life-cycle has three sequential stages concerning initialization, running the simulation, and cleaning up. The first stage is launched as soon as a scenario description file is posted on the orchestration channel. The simulation controller runs several checks (e.g., if there are sufficient hardware resources). In case an execution of the posted scenario is possible, the instantiation of involved BBs is triggered. The results of the check is posted on the status channel. Scenario wide topics are created (Topics 3.35-3.40). The simulation controller publishes a copy of the initial scenario description on Topic 3.38 after successfully instantiating all internal BBs. This copy acts as an acknowledgment for the client that triggered the scenario run and provides the scenario information to involved external parties. Moreover, it can be consumed by the involved BBs in order to receive all necessary scenario information without having to consume and filter messages from the general Topic 3.33. BBs register observers that are requested within the scenario definition and create topics that will be populated during the scenario run. Potentially, they may join the synchronization mechanism.

The second stage is entered when every designated BB has joined the synchronization procedure. The simulation coupling protocol (see Section 3.4.4) is applied until the scenario run is over. Finally, the last stage is used to clean all temporary files, collect (and upload) final results and publish them (via a reference) to the results topic. Running instances such as containers are terminated. Apart from that, the simulation controller periodically kills all spawned processes that are exceeding a pre-configured run time in order to clean up zombie processes and crashed BBs.

3.5.3 Evaluation of Scenarios

For most cases, the plain execution of a simulation scenario is not enough and an evaluating of the results will follow. Due to the data-centric nature, the proposed system already brings a variety of possibilities that do not require additional functionality from a dedicated service component.

- The data pool stores messages persistently. All exchanged information can be reproduced by (re-)consuming the messages from the topics. This includes among other things:
 - Scenario definition and status messages
 - Input resources
 - Messages that are related to the distribution of a simulation
 - Information flows between BBs (simulators, data sources, data sinks, physical devices, ...), translators, and projectors
 - Simulation state updates provided by the observers
 - Final results

- As Kafka is widely-used, there are many compatible technologies that can be utilized. For instance, *ksqlDB* can be used to run SQL-like queries on the existing topics [60, 61].
- Furthermore, the (post-)processing is not restricted to a single scenario. Data from multiple scenarios can be integrated in the analysis (e.g., conducting a parameter study).
- As one major requirement is usability, the graphical user interface provides access to the topics in an easy way.

3.5.4 Management

Besides these three scenario related task areas, it should be possible to manage the catalog of domains, layers, components, connectors, and resources. The simulation controller is supposed to check the validity of received scenario run requests. At first sight, it may seem reasonable to store and manage the catalog within the controller component. However, this would contradict the data-centric thought of having a common data pool and no isolated silos. As a consequence, definitions and resources will also be stored on topics. Adding, removing, or modifying definitions is done by publishing an updated configuration that contains all valid contents to the according topic. Hence, the most recent message on Topic 3.41 - Topic 3.44 represents the current definitions. Resources are added similarly, with the difference that each resource is published in a single message in order to avoid redundancy. A resource can contain a reference to a file or the file itself. Deletion of old resources happens by using Kafka's retention policies.

`orchestration.definitions.domains` (3.41)

`orchestration.definitions.layers` (3.42)

`orchestration.definitions.components` (3.43)

`orchestration.definitions.connectors` (3.44)

`orchestration.definitions.resources` (3.45)

3.5.5 Graphical User Interface

The Graphical User Interface (GUI) should support the user in all four addressed task areas. We decided to use a web-based technology to implement the GUI in order to minimize the system requirements for the user. Therefore, a recent web browser is all that is required to use the simulation system (impressions are given in Section 4.2.1). There is a logical separation between general management, scenario-related tasks, and a historical scenario catalog. The management view provides functionality for uploading resources and for creating, modifying, and deleting domain, layer, component, translator, and projector definitions. The scenario view is split into two parts:

- Design
 - Create instances of components in form of BBs
 - Design topologies for various domains
 - Add connecting translators and projects
 - Parameterize all involved BBs (observers, resources, ...) and connectors
 - Export a scenario into a SCE
- Execution
 - Trigger a new simulation run by sending a SCE to the orchestration channel
 - Provide referenced resources on the corresponding topic
 - Display simulation state updates and BB status updates during a scenario run

The catalog view is designed to list historical runs and according SCEs and provide raw access to the scenario topics.

3.5.6 Service Architecture

As a result, the elaborations and design decisions of this section lead to the system architecture that is depicted in Figure 3.36. The central component is the simulation controller that is responsible for instantiating BBs and managing scenario runs. Almost any connection is realized via Kafka. One exception is the connection between web frontend and web backend because there is no client side web technology library for Kafka. The other exception is the connection between BBs and the resource storage, which can be used if large resource files should not be stored persistently in the data pool⁵. For such use cases, it is suitable to put a file reference in the Kafka

⁵Kafka's default size limit for messages is 1 MB. While this parameter can be adjusted, one could assume that this is the intended order of magnitude for smooth operations.

message. More details regarding the implementation and other technical aspects are given in Section 4.2.

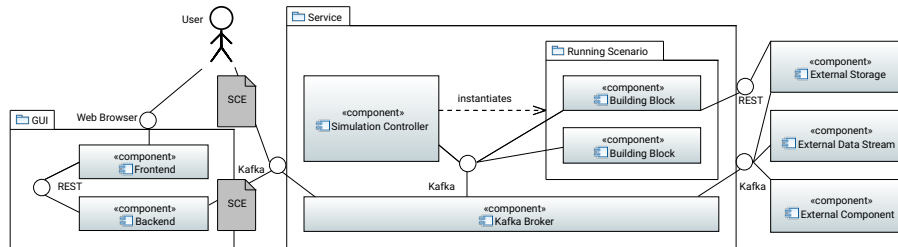


Figure 3.36 – A component diagram of the developed service architecture.

3.6 Summary

In summary, all specified requirements from Section 3.1 are met by the developed system design. Decisions were based on the requirements and were explained when there were multiple reasonable options. In addition, this section will provide an overview on which requirement is impacting which part of the system. In order to do so, all developed concepts are labeled with a letter (see Figure 3.37). These labels are used in Table 3.2 to visualize the relations between system elements and requirements. **REQ 1.3**, for instance, is satisfied by block **d** and **h** and so forth.

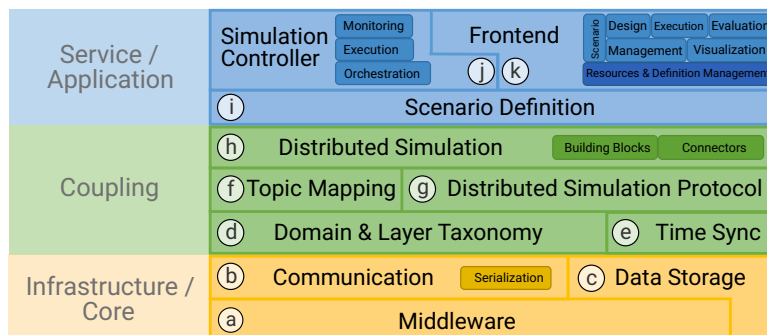


Figure 3.37 – Developed architecture with labeled components.

	Architectural Element										
	a	b	c	d	e	f	g	h	i	j	k
REQ 1.1	↗	↗		↗	↗	↗	↗	↗		↗	
REQ 1.2	↗	↗		↗	↗					↗	
REQ 1.3				↗				↗			
REQ 1.4	↗	↗					↗	↗			
REQ 1.5	↗	↗		↗	↗			↗			
REQ 1.6	↗	↗				↗	↗				
REQ 1.7	↗	↗									
REQ 2.1					↗						
REQ 2.2					↗	↗	↗				
REQ 2.3					↗			↗	↗		
REQ 2.4	↗		↗								
REQ 3.1								↗			
REQ 3.2	↗	↗					↗	↗			
REQ 3.3				↗			↗	↗			
REQ 3.4	↗									↗	↗
REQ 4.1				↗				↗			
REQ 4.2				↗		↗					
REQ 4.3	↗		↗			↗					
REQ 4.4			↗	↗		↗					
REQ 4.5	↗					↗	↗				
REQ 4.6	↗		↗			↗					
REQ 5.1									↗	↗	↗
REQ 5.2									↗		
REQ 5.3											↗
REQ 5.4	↗					↗					↗

Table 3.2 – Relations between requirements and architectural elements.

Chapter 4

System Implementation and Evaluation

In this chapter, we provide implementation related details of the described concept. That covers Apache Kafka and Avro, the simulation controller component, the graphical user interface, and base wrapper libraries that can be used for building own components. Afterwards, demo components are developed and used to evaluate the concept in an application-agnostic way. Parts of this chapter are based on previously published works in [97] and [98].

4.1	Apache Kafka and Avro	93
4.2	Implementation Details	94
4.2.1	Simulation Controller and Graphical User Interface	94
4.2.2	Base Wrapper Libraries	95
4.3	Minimal Working Example	99
4.3.1	Exemplary Data Models and Components	100
4.3.2	Evaluation of Test Applications	103

4.1 Apache Kafka and Avro

As stated in Section 3.3.3, Kafka is using the TCP/IP protocol. In contrast to UDP, this enables a reliable communication, which simplifies the logic on upper layers. Regarding the delivery semantics, Kafka offers several acknowledge modes. There is the “0”-mode, where a publisher does not wait for any acknowledgments by the server. In the “1”-mode, a publisher does wait until the leading broker has acknowledged the reception of a message. Lastly, there is the “all”-mode, where a publisher waits until all replicated brokers have acknowledged the message. Although using TCP, we will set the acknowledge option to “1” in order to prevent message loss that is for instance caused by violated deadlines or overflowing buffers. In order to keep the latency low, we use the default value for “linger.ms” and set it to 0 ms. If efficiency is more important, one could increase this value to enforce batching. “max.in.flight.requests.per.connection” is set to 1, because we want to assure a deterministic publication order of messages per publisher. For the implementations, we are using the native Java library [59], librdkafka for C++ [76], kafka-python for Python [177], and KafkaJS for Node.js [122].

According to the specification, a Kafka topic name does not contain any hierarchies. The “.” is therefore used by us as a level delimiter. Any other non-alphanumeric character in the topic name is escaped by replacing it with two dashes followed by the character’s ASCII code.

Per default, Avro serialization will be used in conjunction with a schema registry for managing the Avro schema definitions. We configure Kafka to use the schema registry with the topic naming strategy, which requires that all messages on a specific topic comply to the same schema. Apart from the already presented structure for representing the scenario definition (including the building blocks, the translators, and the projectors), we designed additional Avro schema definitions that are used domain-agnostic by the components (Appendix A.5):

Control Message (CtrlMsg):

Exchange control commands. For example, pause or resume a simulation.

Structure: `string sender`, `string receiver`, `string command`

Resource File (ResourceFile):

Transfer resources. Depending on the MoM’s characteristics and custom settings, the maximum message size that is acceptable might vary. Therefore, one has to decide if resources can be embedded directly as a byte stream, or if it is more suitable to hand over a plain reference to an external source.

Structure: `string id`, `string type`, `bytes file`, `string fileref`

Synchronization Message (SyncMsg):

Exchange the synchronization data. That incorporates the desired time and epoch advance and a list of all (relevant) messages that were published since the last synchronization message. Each component that is participating in the synchronization mechanism will use the contained info to locally compute the current LBTS.

Structure: long time, int epoch, string sender, string action, map(string,long) messages

4.2 Implementation Details

4.2.1 Simulation Controller and Graphical User Interface

The simulation controller is responsible for the orchestration of scenario runs. The execution of a scenario is requested by publishing the corresponding scenario description on the *orchestration* topic. A user can do this manually (e.g., by using a Kafka CLI tool) or by using the developed GUI. The *SimulationService* is continuously polling for new messages on said topic. The *ScenarioManager* checks for each received SCE tuple if the request is valid (e.g., outdated) and if there are enough resources available. In the event of success, a *ScenarioInstanceExecutor* thread is spawned and the core thread returns to the main loop in the *SimulationService* class. Before the next polling is performed, the statistics on currently running scenarios are updated and occasionally a keep alive message is sent to the broker. For each referenced instance in the scenario description that is tagged as *isInternal*, the *ScenarioInstanceExecutor* is spawning threads that are executing the respective instances. This happens by calling customized run scripts that can either execute native binaries, instantiate docker containers, or trigger similar actions on a computing cluster.

For the graphical user interface, we decided to develop a browser-based solution that does not require the installation of any software artifacts from the user (apart from an internet browser). The application is classically split into a frontend and a backend part. The frontend is a JavaScript application that uses the Vue.js framework and is communicating via HTTP/REST with the backend that is based on Node.js. The user can access three different logical areas: general management, scenario, and catalog. In the general tab, it is possible to manage resources, data models, components, and connectors (see Figure 4.1). Based on this, the user can create and trigger a new simulation scenario in the scenario tab and also observe live data (see Figure 4.2 and Figure 4.3). Results and topics of historic runs can be browsed in the catalog tab (see Figure 4.4).



ID	Domain	Layer	Type	Path	Actions
Manhattan.net.xml	Traffic	Micro	RoadMap	data/resources/uploads/undefined_Manhattan.net_165243653814.xml	
Manhattan.net.reference.xml	Traffic	Domain	DomainReference	data/resources/uploads/undefined_Manhattan.net.reference_1652436555615.xml	
Manhattan.traffic.xml	Traffic	Micro	Traffic	data/resources/uploads/undefined_Manhattan.traffic_1652436571752.xml	
Manhattan.types.xml	Traffic	Micro	Additional	data/resources/uploads/undefined_Manhattan.types_1652436592843.xml	

Figure 4.1 – General management tab.

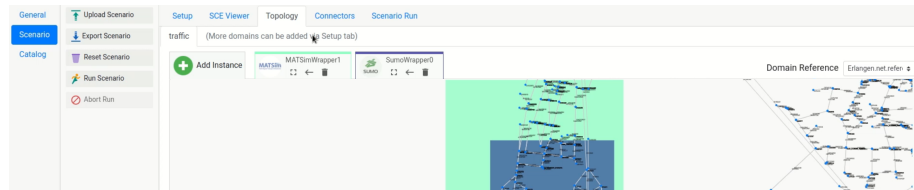


Figure 4.2 – Designing a scenario.

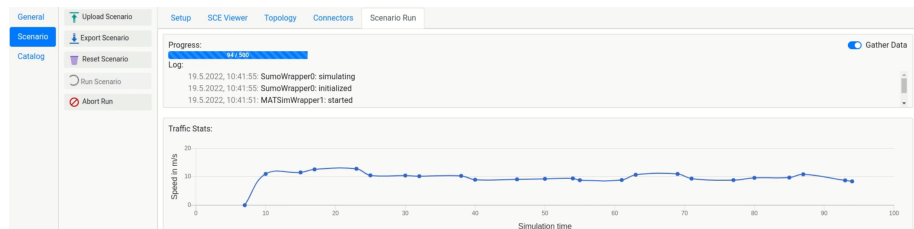
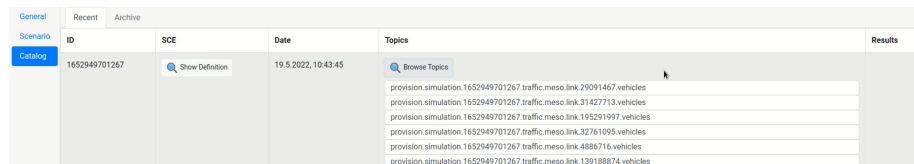


Figure 4.3 – Running a scenario.



ID	SCE	Date	Topics	Results
1652949701267	Show Definition	19.5.2022, 10:43:45	Browse Topics	<ul style="list-style-type: none"> provision.simulation.1652949701267.traffic.meso.link.20091467.vehicles provision.simulation.1652949701267.traffic.meso.link.21427712.vehicles provision.simulation.1652949701267.traffic.meso.link.19291997.vehicles provision.simulation.1652949701267.traffic.meso.link.32761095.vehicles provision.simulation.1652949701267.traffic.meso.link.4886716.vehicles provision.simulation.1652949701267.traffic.meso.link.139188874.vehicles

Figure 4.4 – Catalog view.

4.2.2 Base Wrapper Libraries

In order to have a starting point that allows fast prototyping, we implemented wrapper libraries for C++, Java, and Python, which realize the described coupling approach. Due to certain limitations and capabilities of the respective programming languages and the involved technologies the architectures, the implementations, and the extents will differ for the respective languages. However, the resulting libraries are still so similar that we will describe only the Java base library in more detail. The main components of the *JavaBaseWrapper's* system architecture are given in Figure 4.5 and are described along with other included classes textually in the following.

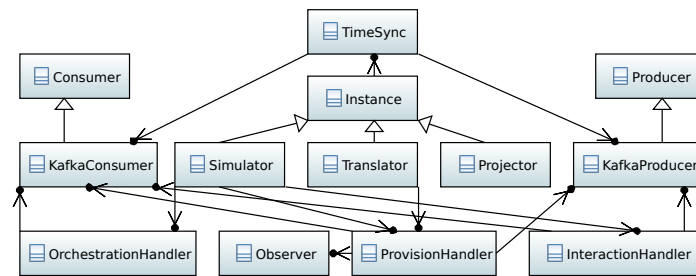


Figure 4.5 – Main architecture of the JavaBaseWrapper.

Config:

A Config instance imports and provides the parameters of the textual configuration file and offers several helper functions for instance regarding topic name inference.

Consumer:

The Consumer class represents a general Pub/Sub-Consumer. It serves as an additional abstraction in order to be able to add other messaging technologies easily. Basically, a Consumer can either subscribe to a list of topics or a list of regular expressions. Moreover, it can be *counting*, which defines if a received message is logged for synchronization purposes. A Consumer can be used in a manual or an automatic polling mode. When in auto-mode, a thread is spawned that continuously checks for new messages. In the manual mode, a user has to call the `poll()` function every time new messages should be received. For both modes, a received message will be delegated to an instance that implements the ConsumerCallback interface. By using Java Generics, a single message type that can be fetched by the consumer is parameterized for each Consumer instance.

KafkaConsumer:

The KafkaConsumer class inherits from the general Consumer class and implements the intended functionality in accordance to the Kafka stack.

Producer:

As for the Consumer, the Producer is a general Pub/Sub-Producer class. A Producer can be *counting*, which results in the appending of sent messages to a log that is used for the synchronization process. The generic parameter of the Producer specifies the type of a message that can be published with a certain instance. The `publish()` method returns a boolean value indicating

the success of a single publications. In addition, there is a method for creating topics in case explicit topic creation is required.

KafkaProducer:

The KafkaProducer implements the general Producer in compliance with the Kafka technology.

TimeSync:

In order to read and write to the synchronization topic, the TimeSync class instantiates one Producer and one Consumer object. Accordingly, both are typed to the *SyncMsg* definition and the TimeSync class is implementing the ConsumerCallback interface (i.e., a method for processing received messages). Received *SyncMsgs* have an impact on the current LBTS and the list of expected messages. The main purpose of the class is the implementation of the synchronization procedure that was developed in Section 3.4.3. Most importantly, there is the `timeAdvance()` method that is used in order to participate in the proceeding of the overall simulation. Its structure is depicted in Algorithm 4.1. In addition, there are methods for joining and leaving the collaborative synchronization progress.

```

1: procedure TIMEADVANCE(time, epoch)
2:   MSGs ← getSentMsgsSinceLastAdvance()
3:   SMSG ← createSyncMsg(time, epoch, MSGs)
4:   publish(SMSG)
5:   TOK ← timeOK()                                ▷ Checks if LBTS is ≥ time.epoch
6:   MOK ← msgOK()                                  ▷ Checks if all expected messages are received
7:   while !(TOK & MOK) do
8:     sleep()
9:     poll()
10:    TOK ← timeOK()
11:    MOK ← msgOK()
12:  end while
13:  localTime ← time
14:  localEpoch ← epoch
15: end procedure

```

Algorithm 4.1 – Time advance algorithm.

OrchestrationHandler:

The OrchestrationHandler is responsible for all communication related to the orchestration channel. That involves mainly the gathering of the initial scenario definitions, logging capabilities, and control commands.

InteractionHandler:

In order to minimize the overhead of processing messages that are not of

interest for a certain component, a single `InteractionHandler` instance is either in request mode (i.e., subscribing to a `interaction.[...].reply` topic and producing to a `interaction.[...].request` topic) or in reply mode (i.e., subscribing to a `interaction.[...].request` topic and producing to a `interaction.[...].reply` topic). When messages are received from subscribed topics, they are stored in a sorted buffer until an Instance component calls the `processBuffer` function, which delegates messages in a deterministic order to the `processInteraction` method of the calling Instance.

DomainHelper:

The `DomainHelper` is used to store the intra-domain responsibilities. It provides a mapping between domain references and layer entities and thus provides knowledge about responsibilities of entities.

ProvisionHandler:

The `ProvisionHandler` is mainly subscribing and producing to topics that are used to realize the scenario distribution. Therefore, there is a `Producer` and a `Consumer` object that is typed to the component's NDM. The subscription pattern is constructed, based on the layer-internal responsibilities that are provided by the `DomainHelper`. It provides the same buffer and process mechanism as the `InteractionHandler` for incoming messages. In addition, observers that were requested in the scenario definition are managed. They are executed systematically, when `runObservers()` is called.

Instance:

The instance is the abstract main wrapper class. It provides all attributes and methods that are used by interacting components (i.e., BBs and connectors). That includes instantiating a logger, a `TimeSync` instance and a scenario run loop. The loop is implementing the loop concept from Section 3.4.4. In order to provide flexibility regarding inheritance, the structure of the loop looks as depicted in Algorithm 4.2. In addition, it provides a `processMessage()` and `processInteraction()` method stub that can be used by `InteractionHandlers` and `ProvisionHandlers`, respectively. In addition, each instance provides a pseudo random number generator that is initialized with the scenario's random seed.

Simulator:

The generic `Simulator` class is inherited from the `Instance` class. It holds a `ProvisionHandler` instance that is typed to the addressed component's NDM. As a consequence, it has to implement a `processMessage()` method that is called when the `processBuffer()` method is called (which happens af-

```

1: procedure SCENARIOLOOP
2:   preLoopEvent()                                ▷ Initial preparations
3:   while  $T \neq SCENARIO\_END$  do
4:     preStepEvent( $T$ )                            ▷ Preparations
5:     stepEvent( $T$ )                                ▷ E.g., perform a simulation step
6:     processStepEvent( $T$ )                          ▷ Process new state, e.g., transfer entities
7:     synchronizeEvent( $T$ )                          ▷ Block until proceeding is safe
8:     postStepEvent( $T$ )                             ▷ Run observers
9:      $T \leftarrow T + Step$ 
10:  end while
11:  postLoopEvent()                                ▷ Clean up
12: end procedure

```

Algorithm 4.2 – Scenario loop.

ter each synchronize event) and if there are unprocessed messages in the ProvisionHandler buffer.

Projector:

As a Projector has no predefined behavior, its generic class will inherit from the Instance class and provides the projector description from the scenario description file. Any additional functionality (ProvisionHandlers, Interaction-Handlers) has to be implemented by a specific projector implementation.

Translator:

The generic Translator class is initialized with the classes of the two data models A and B that should be connected. The translator instantiates two Provision-Handlers, one for each of the two layers to be connected. Depending on the incoming tuple, the `processMessage()` method calls either `translateA2B()` or `translateB2A()`. For typical use cases, a specific translator implementation is only required to implement the latter two methods. However, it might be necessary to extend the existing logic for more sophisticated conversions (e.g., adding a sampling logic if bridging between a continuous and a discrete regime).

4.3 Minimal Working Example

Finally, we design exemplary data models and create a minimal working example that covers all fundamental aspects. The exemplary definitions are also provided in Appendix B.1. This serves as a demonstration, but as well for a subsequent performance evaluation. In addition, the demo applications might be a good starting point for developing own applications.

4.3.1 Exemplary Data Models and Components

The first demo domain is related to numbers. The domain reference consists of intervals that are described by an *ID* and a lower and upper bound (see Figure 4.6). The number of intervals and their bounds can be specified individually for each scenario. Moreover, we define two layers that use a different representation for numbers: integers and doubles. The integer layer consists of entities that will be called *demoInteger* and entities that will be called *interval*. The *demoInteger* represents the NDM. The NDM's key is an *ID* string. In addition, it holds one primitive attribute called *value* with the data type integer. The *interval* is a compound, with an *ID* string as key, a list of *demoIntegers*, and a counter for the current amount of contained *demoIntegers*. The list holds the representations of all *demoIntegers* that are currently assigned to an *interval* and is tagged as the *distributor* element (see Figure 4.7). The double layer will be designed likewise. The only difference is the name of the NDM (*demoDouble*) and its value's data type (see Figure 4.8).

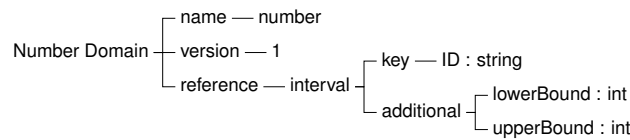


Figure 4.6 – Definition of the number domain.

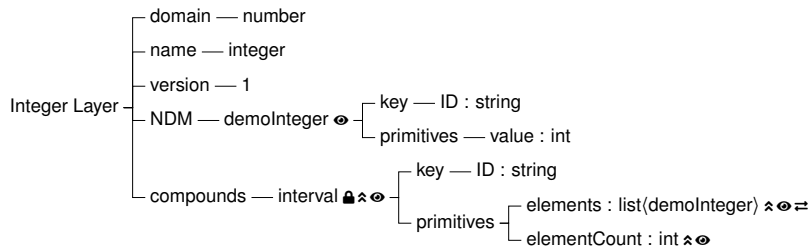


Figure 4.7 – Integer layer definition in the number domain.

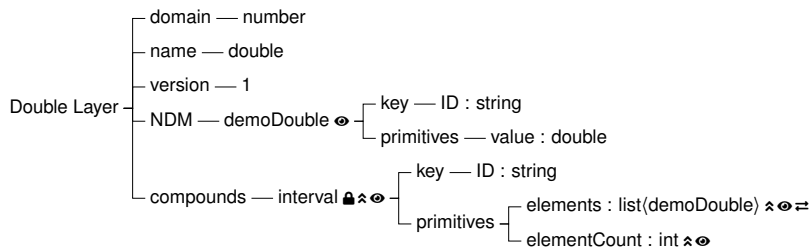


Figure 4.8 – Double layer definition in the number domain.

The second exemplary domain is about colors. The domain reference is a *cell* with an *ID* and coordinates (see Figure 4.9). We define two layers that have a different representation for colors: 3-bit RGB and 24-bit RGB. The RGB3 layer holds an NDM called *pixel* with an *ID* integer. It has three primitive values *red*, *green*, and *blue* of the datatype boolean (i.e., one bit for each color). In addition, there is a compound called *cell* with an *ID* string as key and a list of *pixels* (see Figure 4.10) that acts as the distributor element. The RGB24 layer is defined in the same way, but the data type of *red*, *green*, and *blue* is a char (i.e., 8 bit per color, see Figure 4.11). Both NDMs provide methods to increase and decrease the color temperature.

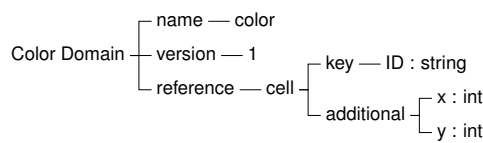


Figure 4.9 – Definition of the color domain.

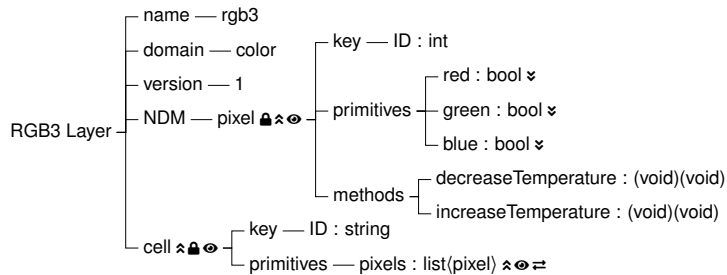


Figure 4.10 – 3-bit RGB layer definition in the color domain.

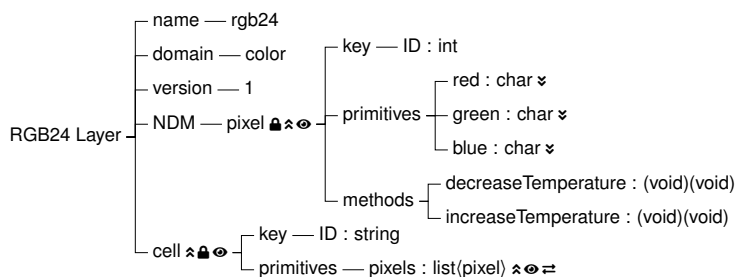


Figure 4.11 – 24-bit RGB layer definition in the color domain.

Simulator Components

A simulator S_I of the integer layer in the number domain is constructed in the following way. The scenario-wide *domainReference* file will provide available intervals.

A set of input parameters contains a number of integers N , a list with initial values V , and a pseudo random number generation function F . The number of iterations I and the responsibilities are derived from the scenario definition. The simulator will create N *demoInteger* variables and preinitialize them with the given values V . Afterwards, the simulation loop is repeated for I iterations. In each iteration, each variable is modified using F and might be moved to another interval container, depending on the value. After the simulation step, the updated simulation state is processed.

The distribution logic checks the content of each distributor element of the layer that is not in the responsibility set of the BB. In this example, this is the `interval.elements` container. The NDMs of variables that no longer belong to an interval container of own responsibility are published to the topic that is also derived by the distributor element (i.e., `provision.number.integer.interval.[key].elements`). The published messages and the wish to proceed to the next iteration are announced by a synchronization message. When all expected messages are consumed, they are sorted, and processed. Finally, custom observers are processed. In the same way a simulator S_D is implemented for the double layer.

For the second domain, there will be two color simulators: S_3 will operate in the 3-bit RGB layer and S_{24} in the 24-bit RGB layer. As for the number simulators, they will have no real practical use, as they are designed to have minimal complexity in order to be able to understand the executable's code easily. Their exemplary purpose is to modify an input picture (i.e., a set of RGB tuples) incrementally over the simulation duration. The S_3 simulator is capable of rendering the picture with a 3-bit RGB color model (i.e., with 8 possible colors). In contrast, S_{24} can differentiate between 16,777,216 different colors.

Connectors

A translator T_0 is designed in the following way. The translation between the integer and the double layer clearly shows a (dis-) aggregation pattern. The more detailed double layer is translated to the integer layer by rounding the *value* element of the double NDM, while the *ID* is not modified. The disaggregation function, on the other hand, will incorporate some randomness to sample missing information. In particular, the output NDM's value is constructed by adding a uniformly distributed random offset within ± 0.5 to the value of the *demoInteger* element of the input NDM. The *ID* is preserved. The translator is trigger-based, hence every incoming NDM will trigger the creation and publication of another NDM.

A projector P_0 is implementing the following logic in order to connect the number and the color domain. P_0 is subscribing to all *demoIntegers* on `provision.number.integer.demoInteger`. If the values of all consumed integers of one

simulation step are negative, the temperature of all pixels in the color domain is decreased. Similarly, the temperature is increased if all received values are positive. The modification is triggered by sending according interaction messages to `interaction.color.[rgb3|rgb24].request`.

4.3.2 Evaluation of Test Applications

In this section, we use the given example layers, simulators, and connectors to demonstrate and evaluate the functionalities on an exemplary level. As for the definitions and components, the experiments will be as simple as possible in order to focus on the core mechanics.

Experiment 1: Baseline

For a first experiment, we use a single BB of S_I that should perform a simulation in the number domain. This use case is for testing the workflow and generating a baseline for further experiments. The whole scenario definition is given along with the input file in B.2 in the appendix. The scenario-wide *domainReference* file defines two intervals (*negative*: $[-10000, 0)$, *positive*: $[0, 10000)$). The BB is responsible for both intervals. There are two `demoInteger` in the scenario, whose behavior (i.e., values) are simulated by using a function called *growingSine*. We want to let the BB publish information about all `demoInteger` after each simulation step.

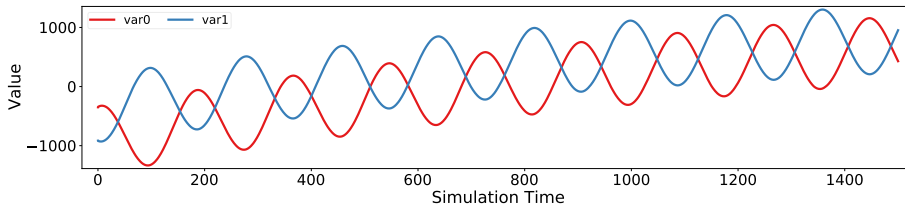


Figure 4.12 – Generated baseline data.

The generated simulation data is depicted in Figure 4.12. It is also published on `provision.experiment1.number.integer.demoInteger`, according to the specified observer. In each simulation step, the `demoInteger` entities' values are modified by adding a static and an oscillating part to the old value by the *growingSine* function. This function has four input parameters (the random seed s , the current simulation time t , the step length l , and the variables current value v) and implements the following formula that allows deriving a dynamic but deterministic behavior:

$$growingSine(s, t, l, v) = v + l * (1 + \sin((s + t) * \alpha) * \beta)$$

Experiment 2: Distribution

As a second application, we use two BBs of S_I that shall split the workload (i.e., the responsibility list of the first BB S_{I0} will contain the *negative* interval, the second BB S_{I1} will cover the *positive* interval). Apart from that, we will use the same parameters as in the first experiment. As intended, we get the exact same results as before, although two independent BBs shared the work and repeatedly exchanged integer entities. Figure 4.13 shows the combined results and gives information about the origin of data points. With this example, we can demonstrate that our approach provides the features for a correct distribution of a simulation scenario. The results of the distributed run show no deviation to the non-distributed baseline.

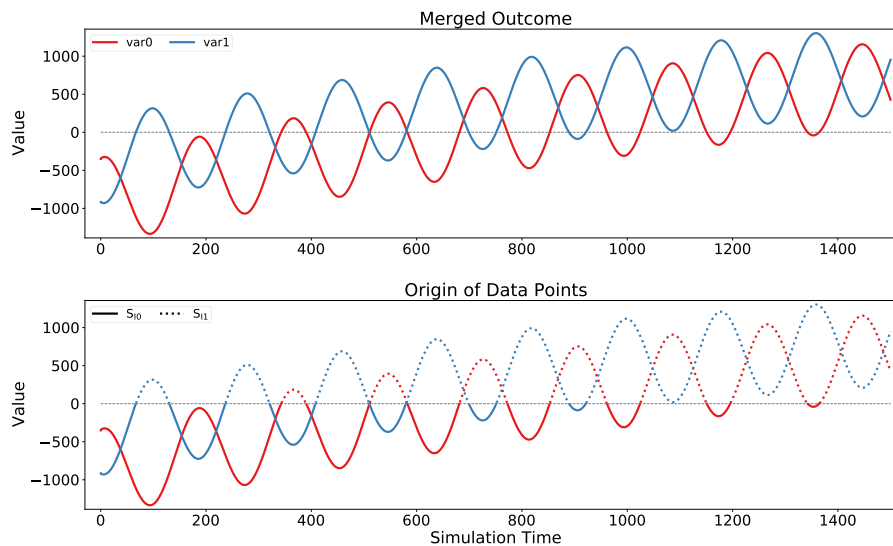


Figure 4.13 – Merged result and origin of data points for experiment 2.1.

Based on our fully functional implementation of a distributed simulation system, we can illustrate the importance of the synchronization for correctness. A BB has a *synchronized* flag, which can be disabled individually for each BB involved in a scenario (see Section 3.5.1). If disabled, the BB will not take part in the developed synchronization mechanism. This makes sense for certain use cases (e.g., for a BB that is just rendering the current simulation state, or if performance is top priority, while correctness is not so important). However, if such non-synchronized BBs are actively participating in the simulation of the scenario, this may lead to causality violations. In order to emphasize the importance of synchronization for deterministic results, we will disable the synchronization for both BBs in the scenario definition of experiment 2.2. The results are depicted in Figure 4.14. As a consequence from the missing synchronization, the results differ from the baseline. Each entity transfer

between BBs bears the risk of receiving and processing data too late, too early, or in the wrong order. Hence, we can observe a growing error over the simulation time.

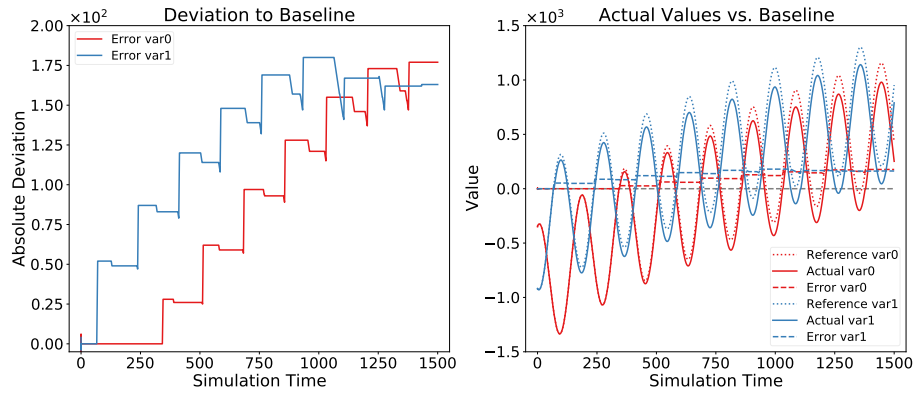


Figure 4.14 – If synchronization is disabled, the results of the distributed run differ from the baseline for experiment 2.2.

Experiment 3: Translation

In order to demonstrate the composition of two BBs of different layers within a domain, the second BB S_{D0} is an instance of SD (see Appendix B.3). Additionally, the step length of S_{I0} is set to 20 to show the possibility of having differing step lengths (e.g., for the purpose of speeding up the simulation). Besides these modifications, the other parameters match experiment 2.1. As both BBs operate in different layers, we need to add a translator to the scenario in order to establish a data flow between the BBs. The already specified translator T_0 is used for this purpose. The outcome is depicted in Figure 4.15. We can see smooth changes for the modifications that are done by S_{D0} . For S_{I0} , the bigger step size is leading to clearly notable edges. In addition, the simulation state of S_{D0} is also impacted as NDMs are sent and processed by S_{I0} only every 20th-step. Errors of such kind are typically accepted when considering a trade-off between accuracy and computational effort. Moreover, introducing errors by inaccuracies are inherent for multi-level modeling. However, the resulting deviation to the baseline is deterministic and not random as in experiment 2.2 when synchronization was missing. Experiment 3 is therefore still reproducible and repeatable.

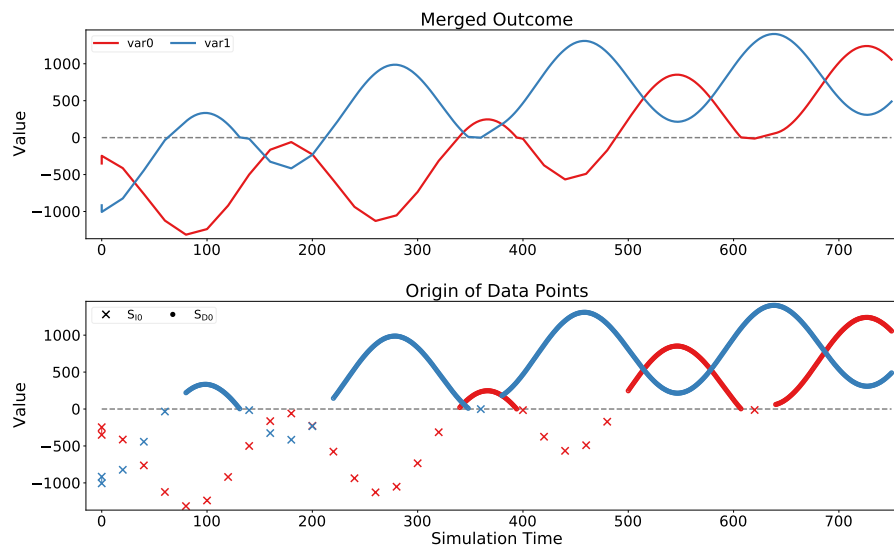


Figure 4.15 – Merged result and origin of data points using two BBs with different levels of detail and step lengths.

Experiment 4: Projection

The composition of BBs of different domains will be illustrated by coupling simulators from the color domain with an integer simulator. As already stated, a color simulator takes an input picture and displays a simulated representation that is according to its color model. Moreover, it is possible to modify single pixels by setting the values of the different color components or trigger a change in the color temperature of the whole picture. For an exemplary application, we might find it useful to modify the temperature based on the intermediate results of a simulator from the number domain. To establish the information flow between the BBs from different domains, the projector P_0 is used.

The input picture will be a random set of pixels with a size of 8×1 that is shown at the top of Figure 4.16. At the bottom left and at bottom center the development of the simulation state is illustrated for a simulator from the 24-bit layer and from the 3-bit layer, respectively. The resulting output picture is given in the last rows of both plots (i.e., for a simulation time of 1500). At the bottom right, the incorporated simulation state from the integer simulator is shown. Red and blue colored areas depict situations where P_0 's conditions for triggering interactions in the color domain are met. At the beginning of the simulation, there is a phase where the temperature is decreased, followed by a steady phase, and finally the temperature is increased as the integer values are mostly in the positive interval. Consequently, we demonstrated the modeling and simulation of a cross-domain problem by simply connecting data models.

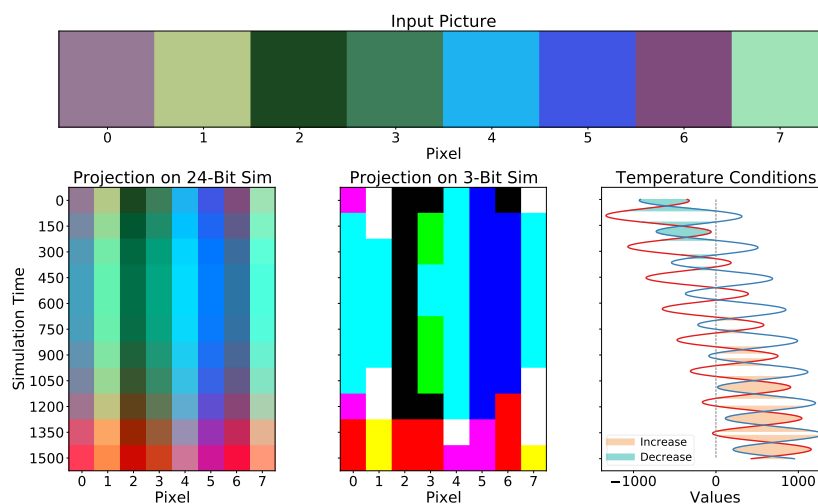


Figure 4.16 – Cross-domain coupling using a projector. Data from the number domain is affecting the modeled picture.

Experiment 5: Performance

Finally, we assess the overhead of the coupling protocol by measuring the duration of the synchronization in each iteration. For this purpose, we will consider a local setup, where all components including the data pool are operated on a single physical multi-core system, and a physically distributed setup that involves several desktop PCs. The local setup is executed on a HP ProLiant DL380 G7 with 12 physical Intel Xeon X5690 cores operated by Ubuntu 20.04.4 LTS. For the distributed run, the broker will remain on the HP server. Each synchronized participant is executed on a dedicated Ubuntu 20.04.4 LTS machine with four Intel i7-2600 cores. All devices are connected via a Gigabit Ethernet switch.

For both setups, we considered two, four, six, and eight involved BBs, which had to synchronize for 1000 steps, respectively. We repeated the experiments for ten times. The measures for the synchronization efforts are given in Figure 4.17 and Table 4.1. As expected, the local run for only two synchronized instances requires the least amount of time. With an increasing number of participants the duration seems to grow linearly. While the mean and average duration is below two milliseconds for the topology with two instances, it takes around 5 milliseconds for a synchronization step when eight instances are involved. Interestingly, the physically distributed run shows a better performance for almost all topologies. The local run is only faster when just two instances are connected. As the server provides 12 physical cores this indicates a bottleneck regarding the communication. Moreover, one can see a nearly static behavior for all four physically distributed topologies. For the median values, all of them require around 2.4 milliseconds. With that we could demonstrate that our approach is introducing an acceptable low overhead. Furthermore, the performance is comparable to the synchronization performance of current HLA implementations [95].

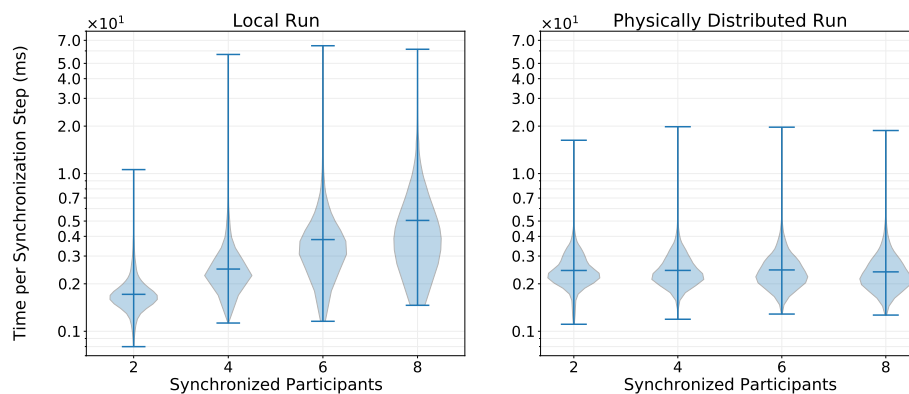


Figure 4.17 – Measurements for time synchronization steps in ms.

Participants	Local Run				Physically Distributed Run			
	2	4	6	8	2	4	6	8
Median	1.72	2.48	3.82	5.05	2.43	2.43	2.45	2.38
Mean	1.84	2.81	4.51	6.14	2.60	2.63	2.65	2.62
Std.	0.53	1.35	3.00	4.25	0.79	0.90	1.00	1.11
Max	10.60	56.96	64.70	61.45	16.27	19.83	19.69	18.76
Min	0.80	1.13	1.16	1.46	1.11	1.19	1.29	1.27

Table 4.1 – Measurements for time synchronization steps in ms.

Chapter 5

Application to the Traffic Domain

In the previous chapters, we developed a generalized data-centric distributed simulation concept and implemented it in form of a MSaaS system. As a second part of this thesis, we will now apply the general and domain-agnostic concept to a specific application domain, the road traffic domain. In order to do so, we first provide fundamentals on traffic and its modeling. This underlines the assumption that from the application’s perspective, collaborating with domain-experts or having domain knowledge is important for the modeling. Based on the given theory, we then develop the domain definition and the layer definitions for all major modeling paradigms within the traffic domain. Finally, we design translators, implement wrappers for well-known traffic simulators, and integrate another domain. Parts of this chapter have been previously published in [96], [97], [98], and [100].

5.1	Modeling and Simulation of Traffic	113
5.1.1	Established Modeling Paradigms and Stereotypical Tools . . .	114
5.1.2	Distributed Traffic Simulation Approaches	121
5.1.3	Mobility Data	122
5.2	Using the Approach in the Traffic Domain	125
5.2.1	Modeling of the Domain Definition	125
5.2.2	Modeling of the Layer Definitions	128
5.2.3	Modeling of the Translators	133
5.3	Implementing Wrappers	137
5.4	Extension to a Further Domain	142

5.1 Modeling and Simulation of Traffic

As for many other domains, methods from the M&S field are precious tools in the domain of road traffic. Barcelo [20] describes the causes for mobility as a phenomenon of social and economic aspects that are caused by human nature. Our lifestyle and behavior creates a demand for moving people as well as cargo (i.e., mobility). The transportation system provides the infrastructure that has the potential to enable one or multiple ways to realize these movements in form of trips. Regarding the process of transportation, it is not only about the origin and destination of a trip. It is also relevant when a journey happens, how long it takes, how much it costs, what mode is used, and how subordinate modalities look like (e.g., reliability or ecological impact). Barcelo further describes the transportation system as a system consisting of two main elements: the infrastructure and the user. While the user has needs, the infrastructure offers possibilities. From a modeling perspective, this leads to the question of how users make choices about trips. Making choices implies that a user has certain preferences and an own individual understanding of the principles and the functioning of the transportation system. Both preferences and understanding are used in conjunction with the user's plans to make decisions.

Modeling of demand can happen on various levels. Typically, there are two different ways [20]. An aggregated method uses so-called origin-destination matrices to represent mobility demand. Each origin-destination pair holds information about the demand traveling from source to destination. In the simplest case, this could be a single number of total trips. However, the level of detail can still be manifold, for instance the matrix can hold time-varying values or information about mode choices. Both origin and destination nodes are usually not representing a precise geo-location or a street address, but rather cover a whole area (also called a traffic zone). The input for such models does not necessarily come from knowledge about behavior of individuals. Moreover, it is common to use traffic count data to estimate origin/destination matrices [1, 51, 186].

On a more detailed level, the activities that cause mobility can be used to represent demand. By using an activity-based approach, there might be additional information for each trip such as planned stopovers. However, this implies that there is some model about the emergence of activities (i.e., activity demand modeling). Ben-Akiva et al. [22] are stating assumptions used for activity modeling by referring to Chapin [55] and Hagerstand [101]. The assumptions include that the household has an impact on decisions. For instance, the income, the size, the availability of cars, the age, and the number of children can play an important role. Besides, there are more general assumptions that are useful. A person typically rests at the same place every night or a human being can only be present in one certain location at a certain time. In addition, there might be other constraints that do not arise from the personal

circumstances, such as office hours. Activity-based modeling does typically consider a whole day and categorizes activities in certain groups such as work, leisure, or shopping.

The modeling of the transportation network on the other hand, can be modeled in a more straight-forward way. Most of the necessary information is accessible at a certain fidelity and there is little room for interpretation compared to the demand modeling. Depending on the questions that should be answered, different modeling paradigms and levels of details are suitable. More information on that will follow in the subsequent sections. While the network offers possibilities, it has a certain capacity that has to be taken into account for most analyses in order to get useful results. The process of mapping the mobility demand to the network is called traffic assignment. As a result, it is often the case that routes are sampled for each trip [22].

Finally, the simulation of the developed model will consider how the traffic system model behaves over time. As for simulations in general, the simulation time can be continuous or advanced in discrete steps (see Section 2.1.1). Usually, discrete simulations are used and although it is thinkable to have event-based traffic simulations with varying step lengths, it is more common to have a fixed step length (e.g., SUMO or MATSim).

5.1.1 Established Modeling Paradigms and Stereotypical Tools

Different objectives lead to different models with varying purposes. The most mature and popular approaches are macroscopic and microscopic models [108]. Later, the category of mesoscopic models appeared. While there are numerous definitions of mesoscopic, which are partially contradictory, there is a consensus about mesoscopic models filling the gap between macroscopic and microscopic models. Lastly, there is the group of submicroscopic models that is sometimes also referred to as nanoscopic. Figure 5.1 depicts their relation to each other. In the following, the approaches of all four groups will be described briefly in order to have an elementary understanding of them and their differences. In addition, one well-known off-the shelf implementation out of each category is presented as it will be integrated as a stereotypical simulator into the simulation system later on. The decisions for or against tools are based on our own experience in the traffic field and backed by other scientific publications [64, 77, 81]. We tried to stick to open source projects if possible.

Macroscopic Traffic Simulation

Macroscopic traffic modeling is the most aggregated approach. Within this category, it is more common to have time-continuous models. Many approaches use equation-based attempts for modeling traffic flows. Established time-continuous models use first or higher order differential equations. They allow the modeling of the traffic

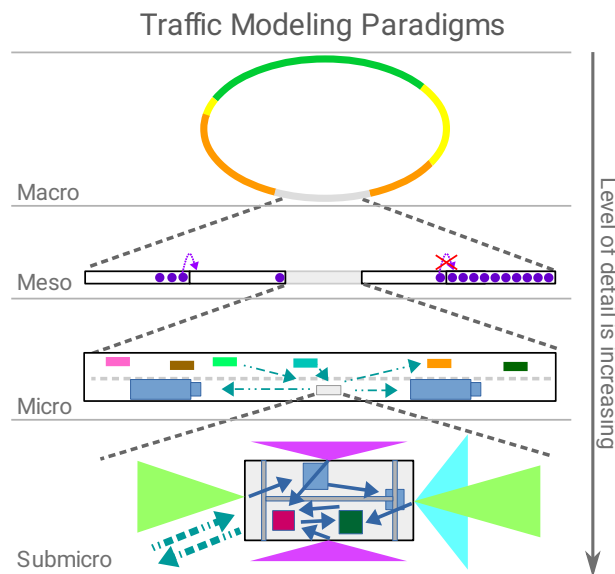


Figure 5.1 – Different traffic modeling paradigms based on [99].

evolution over time. Well-known ones are the models of Aw-Rascle-Zhang [13, 235] and Lighthill-Whitham-Richards [135, 184].

The models are inspired by hydrodynamics. As traffic is seen as a flow, there is no representation of single traffic participants such as vehicles. Helbing [108] sees the purpose of macroscopic models mainly in short-term traffic predictions, as a foundation for traffic management systems, and for evaluating aggregate numbers such as average vehicle speeds or emissions. The lack of details brings advantages regarding the simulation. There are relatively low input data requirements and the computational effort is small. In general, there is no proportional relation between traffic load and computational effort as in microscopic models. Basic variables of interest are the traffic density, the traffic flow, and the speed of a certain road stretch at a certain time.

- The density (k) describes the coverage of a stretch by vehicles. In order to normalize values, its unit is commonly given in the number of vehicles per kilometer or mile.
- The flow (q) describes the throughput of a stretch by vehicles per hour. Sometimes the flow is also referred to as volume.
- The speed (u) describes the average speed on a stretch by kilometers per hour or miles per hour.

Matching the hydrodynamic idea, the traffic system can also be imagined as a pipeline system. Similarly, there is a conservation of traffic flow on a stretch that has

no additional entries and exits (i.e., the outgoing traffic equals the incoming traffic). Obviously, density, flow, and speed are correlated. The relation of these three basic variables is well studied under the term *Macroscopic Fundamental Diagram (MFD)* (see Figure 5.2). Backed by a sound mathematical foundation [89], one can spot phenomena that match one's individual experience on the road. When there is a stationary state, a simplified relation between the variables can be expressed as in Equation 5.1.

$$q = ku \quad (5.1)$$

There is typically a turning point regarding the density (k_c). For a density between 0 and k_c the traffic situation is described as *free flow* (see Figure 5.2). There is (practically) no restriction caused by other traffic participants. If the density is higher than k_c , then this changes and the behavior of vehicles is impacted, which can for example lead to traffic jams. Within the free flow regime, there is another turning point k_s . While the density lies between 0 and k_s , the traffic state is stable. A nearly linear relation between q and k can be seen (see Equation 5.1). This relation justifies, for example, the meaningfulness of variable speed limit signs, which reduce the allowed maximum speed when the density is growing in order to prevent traffic jams and optimize the traffic flow.

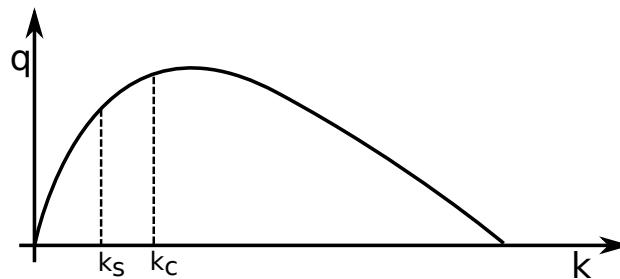


Figure 5.2 – Macroscopic fundamental diagram based on [65].

PTV Visum is a well-known macroscopic tool that primarily addresses traffic engineers and municipal decision makers. PTV GmbH describes it as a traffic planning software that is suitable to satisfy the various needs of people without losing sight of economic aspects. By having the opportunity to model multi-modal traffic, a digital representation of a whole transportation system can be generated. The model can be used to analyze current problems or identify future possibilities [181]. This implies to be capable of modeling large-scale scenarios. Currently, PTV Visum runs only on Windows although a Linux version is announced. There is a COM-API that provides access to most functions and attributes. It is possible to use the common macroscopic origin-destination matrices to specify travel demand between zones. This can be done for each mode of transport or in a combined way. Also, adding a time dimension

is no problem. The traffic network is modeled with nodes and links. A node is a representation of a junction. It has a certain position and holds information about connected links. Each link object is bidirectional, but has two directed sub-elements. The sub-elements have certain attributes such as allowed modes, start and end node, and a capacity. Zones are connected to (multiple) nodes [111]. Typically, a PTV Visum user wants to analyze the resulting traffic by providing the travel demand and using the integrated assignment procedures. Results of interest include, for example, the traffic volumes, speeds, and mode choices. The available assignment procedures use different algorithms for estimating the traffic situation. In general, this is realized by finding all possible ways to realize the travel demand. Then, initial choices are made and travel costs are evaluated. In an iterative procedure, some choices are then consecutively modified until the overall costs are converging (e.g., if all cars take the shortest route, there might be a traffic jam, while it is less bad to take a little detour than to be stuck in traffic).

Microscopic Traffic Simulation

In contrast to macroscopic traffic models, microscopic traffic models are more detailed. By modeling individual traffic participants and their behavior, the overall traffic conditions of a system can be inferred bottom-up. Moreover, these approaches do also allow for a detailed investigation of the individual behavior. Burghout et al. [41] mention that microscopic models offer the required level of detail to evaluate intelligent traffic system technologies. At the same time, they admit that the effort for input modeling and calibration is not neglectable and that the potential error grows with the level of detail. The computational effort is also increased with each loaded vehicle. Scalability is therefore an issue.

The most common modeling approach is using car-following models [90, 123]. Hence, it is not only about modeling individual vehicles but also individual behavior of vehicles. The general idea behind these models is that each vehicle has a desired speed. It follows a certain route in order to accomplish a trip and is affected by the transportation infrastructure (e.g., road topology and speed limits) and other traffic participants. The behavior of each vehicle is computed by taking into account the speed and distance of the vehicle that is driving in front of it, the speed limit, and the desired speed.



Figure 5.3 – A car-following model is mainly based on the distance between vehicles.

Typically, this decision process leads to actions such as accelerate or decelerate, which in turn can be used to calculate the current acceleration and the speed in the next simulation step. Besides the longitudinal control, it is also common to model lane change maneuvers, for instance, in order to overtake. The basis for this is a simple physical vehicle dynamics model (e.g., it is not possible to stop a vehicle immediately). This does not only lead to a more realistic model of the traffic dynamics. It also helps to estimate certain variables within a vehicle more precisely. With the slope of the road, even the third dimension could be considered. This might be required for a good accuracy when calculating the consumption (fuel or electricity) or emissions in hilly areas. Considering discrete calculation steps, the car-following models are suitable to be implemented in a DES. Smaller steps can obviously lead to a higher precision, but will cause an increased computational effort.

Besides traffic infrastructure related information, simulating microscopic models requires basically a set of vehicles, their parameters, and their routes as an input. Some simulators provide the possibility to sample parameters from random distributions or calculate routes based on origin-destination pairs. Possible parameters of a vehicle describe the vehicle itself (e.g., type, geometrical dimensions, physical limits) and the driver (e.g., perception time, error rate, mood).

The **SUMO** (Simulation of Urban MObility) package is a microscopic traffic simulation tool. In contrast to PTV Visum, it is an open source project, which is implemented in C++ and runs on Linux, macOS, and Windows. Multiple modalities are supported as well as large networks. However, it is obvious that the computational effort for a large-scale scenario is bigger than with PTV Visum. Typical use cases are the evaluation of traffic light programs or the analysis of route choices. Providing speed values and absolute positions for vehicles, it is popular to be coupled with simulators of other domains (e.g., for V2X applications). SUMO implements various car-following models. Per default, an adaption of the well-known Krauss model is used. A SUMO user needs to provide at least a set of vehicles with departure times, starting point, and destination. Usually, one introduces multiple types of vehicles with different characteristics (e.g., slow car, fast car, truck, bus, ...) in order to enhance the degree of realism. Also, the explicit definition of routes for involved vehicles is common. Besides vehicles, it is also possible to model pedestrians and additional entities such as traffic lights or parking places. [8, 75]

Mesoscopic Traffic Simulation

As already pointed out, the definition of mesoscopic models is not as clear as it was for the prior ones. Burghout [42] lists several exemplary approaches that fill the gap between microscopic and macroscopic models:

- Multiple vehicles are grouped into one representation. The group is moving as a single entity. Each vehicle has still an individual representation, but its speed is derived from the MFD-relation (i.e., using the link density) [133].
- Vehicles can enter and leave cells that are also moving (called cell-based aggregation). A vehicle infers its speed from its cell's speed [21].
- Roads are modeled as queues. The vehicles can have individual speeds that are derived from a MFD relation. When arriving at the end of a road, a queue-server is moving the vehicle to a connected road (i.e., the next queue). This concept also allows for a more elaborate intersection modeling (e.g., traffic lights) [119].
- Roads are partitioned into stationary cells that have capacity for a single vehicle. Based on a cellular automaton, a vehicle is moving in discrete steps from cell to cell [155].

In the following, we call a model mesoscopic, if traffic is modeled by discrete (groups of) passive traffic participants that are moved through a system. As for any modeling approach other than macroscopic, the focus is clearly shifted from the road to the road user. Burghout et al. [41] see the big advantages over microscopic models in the fact that there are typically fewer parameters that need to be calibrated. Also, mesoscopic models are less sensitive to faulty inputs (e.g., small errors in the road network, demand mismatch). Obviously, the computational effort is lower than for microscopic approaches. The cost of the more aggregated nature is a lack of information (e.g., the current acceleration or the precise position of a vehicle). There is no driver state and behavior. Nevertheless, it might be sufficient for evaluating use cases that require individual representations of vehicles, but not in a very detailed way (e.g., fleet management algorithms or novel taxi services). Possible outputs range from the number of customers and the assigned routes to the travel times. As for microscopic models, one has to provide the trip data as input.

The **MATSim** (Multi-Agent Transport Simulation) software aims for estimating the resulting mobility of a synthetic population [142]. It is an open source project that is implemented in Java and can therefore be used on various platforms. The maintainers do also aim for supporting large-scale scenarios. There is no dedicated API. However, the code is implemented in a very modular and extendable way. For demand modeling, MATSim follows an activity-based approach, where the population consists of agents that try to realize their individual plans of activities. A plan usually covers a whole day in an agent's life, which is sufficient for many traffic analyses. Agents do not only try to realize their plans, they try also to optimize their route, mode, and time choices. In order to do this, the traffic simulation is iteratively repeated, while allowing some agents to modify their choices. A key to this is having

an efficient traffic simulation algorithm. One supported algorithm is implementing a queue-based model, where each link is representing a queue [17]. Moreover, for an already calibrated set of agent plans (incorporating mode choices and routes), a user can disable additional iterations and use only the traffic simulation feature of MATSim. Partly because of the individual agents, MATSim is also considered to be representing a microscopical approach. However, in our understanding, and in comparison with SUMO and PTV Visum, MATSim seems like a good fit for the gap-filling category of mesoscopic models.

Submicroscopic Traffic Simulation

The most recent modeling approach is called submicroscopic and is characterized by having an even more detailed model of the vehicle and also of the environment compared to the microscopic traffic modeling. That means that there is a realistic physics engine combined with a three-dimensional graphics engine. It is common to use state-of-the-art game engines as they provide both components (e.g., Unity, Unreal Engine). As a consequence, it is possible to model the environment in a very realistic way (e.g., road surface, weather, lighting conditions). The presence of a realistic 3D environment allows for the reasonable modeling of additional components inside a vehicle (e.g., cameras or lidar sensors). Eventually, detailed sensor models can be used to evaluate advanced driving functions such as Adaptive Cruise Control (ACC) or Automatic Emergency Braking (AEB).

A difference to prior approaches is that a submicroscopic experiment is often about one (or a few) ego vehicle that is under study and not about the system behavior. Other vehicles may be modeled with less detail, as they are seen as parts of the environment. Obviously, this is related to a major increase in terms of computational effort and input modeling effort. Although the quality of the input and the validity of the models are crucial, it is clear that such a desired degree of detail makes only sense if one can provide inputs at an adequate level (e.g., a realistic vehicle dynamics model, if investigating the impact of different road surfaces).

The **CARLA** software is also an open source project and represents a submicroscopic tool. It is mainly aiming to provide support for problems in the field of autonomous driving. That can for instance include the training of perception algorithms or the evaluation of driving functions. It is built on top of the Unreal Engine and will therefore be assigned to the submicroscopic category. CARLA runs on Linux, macOS, and Windows. The architecture follows a client-server structure, where the simulation states are computed in the server component, while modifications (e.g., spawn a new vehicle) are triggered via scripts running on a client. For this purpose, CARLA offers a Python and C++ API [48, 49, 74]. The offered degree of detail obviously comes at the cost of performance. Moreover, the modeling effort for

a realistic scenario can be tremendous at this level. Nevertheless, it can be necessary for certain applications, for instance, when developing sensor models or algorithms that are related to autonomous driving. Regarding resources, CARLA supports the OpenDRIVE standard as input for the road topology. Contrary to the microscopic and mesoscopic tools, there is no initial input file that holds involved vehicles and their plans. Therefore, a user has to trigger the spawning and the parameterization of each vehicle explicitly during the scenario run. This can be done using the API.

5.1.2 Distributed Traffic Simulation Approaches

As for many domains, DS has mainly two manifestations within the field of traffic. On one hand, there is the plain distribution of logical partitions to multiple instances of the same simulator for the sake of overcoming hardware limitations (e.g., increase performance or overcome memory limits). This results, for example, in having one instance simulating the northern part of a city and another instance simulating the southern part of that city. Ideally, this would boost the performance by a factor of two. The challenges are mainly having a working synchronization, a partitioning algorithm, and a concept for the data exchange at the connecting border regions between instances. All in all, this manifestation requires hardly any further modeling. The main issue is to assure a sound implementation of the parallelization on a technical level. One issue regarding the connection of neighboring regions is how to communicate the state of connecting links. If dealing with microscopic models one would exchange vehicles on a connecting road between neighboring instances. However, if there is no additional knowledge, the sending instance would not know if the target link in the receiving instance is congested and cannot host any new vehicles. This specific problem was already addressed by a few works using the term *ghosting* to describe a solution approach [41, 196]. Passive representations of vehicles remain on the connection road in the out-sending instance. These passive vehicles are referred to as ghosts. The internal states of ghost entities are not computed by the old instance anymore, but are rather ingested from an external source (e.g., from the new receiving instance). Using such concepts, it is possible to propagate environment conditions such as traffic jams across partition borders. A practical example for the distribution of a traffic scenario to multiple SUMO instances is given in [37].

On the other hand, there is the idea of multi-level simulation, where traffic models with a varying degree of detail are combined. This can be motivated once more by computational aspects (e.g., aiming for the perfect trade-off between required level of detail and performance). However, it could also be necessary from an input modeling perspective (e.g., available inputs and desired outputs do not match a single model). In order to realize a multi-level simulation, the previous distribution

challenges have to be tackled. In addition, it requires a modeling effort regarding the information transfer between different submodels. In a simple example, there is aggregated traffic volume data for a specific region. The data would be suitable to be used in a macroscopic model. We may want to determine good locations for adaptive traffic lights in this region by conducting a simulation study. However, the adaptive algorithm might need information of individual vehicles and therefore the need for a microscopic traffic model arises. As a result, one option would be to have a macroscopic model of the whole region that is based on the available input data. In the same scenario, interesting junctions could be modeled in a microscopic way. The challenge would then be to realize the information transfer between the different models that are running side by side (e.g., sample discrete vehicles from continuous traffic flows). Practical examples are given in [99, 199].

5.1.3 Mobility Data

In the following, there will be a non-exhaustive collection of mobility data, its availability, data requirements, and on how inputs find their way into models and simulations. We distinguish between two different kinds of information related to mobility: static and dynamic data. The first category of static data is not necessarily understood as mobility data in the first place as it is mostly describing infrastructural elements. Nevertheless, input from this group is absolutely necessary for almost all traffic studies. The most important piece of static information is the transportation network itself. Data requirements of existing traffic models differ greatly, but there is almost always a need to know which locations are connected in which way and at which costs (e.g., distance). A basic representation of a transportation network can be given as a graph with vertices and edges. The OpenStreetMap project is a precious source for real-world mobility network topologies [102, 165]. Embedded or related information covers:

- Roads, train lines, bus lines, bicycle lanes
- Modalities of the links (e.g., one-way restrictions)
- Junctions, turning rules, traffic lights
- Geo-locations, elevation information
- Additional information such as crosswalks or buildings

Comparable to the Wikipedia project [226], volunteers play an important role in maintaining the data in the OpenStreetMap project. Therefore, the amount of information is huge. However, it must also be taken into account that the level of accuracy can differ significantly for certain maintainers and regions. Other information that can be queried from heterogeneous sources include:

- Public transport schedules and prices
- Reasonable general estimations (maximum acceleration of an average car, average speed of a bicycle)
- Tolls, (varying) speed limits

On the other hand, there is dynamic information, which is obviously more likely to change. This could cover gas prices, the presence of a public event that would attract a lot of people, or the availability of limited resources such as a rental car or bicycle. Mobility patterns are one of the most interesting pieces of information in this context. They can be estimated based on historical or live sources:

- Meta models on individual choices or region-wide mobility distributions
- Aggregated measures (e.g., loop detector data, overhead radars)
- Aggregated processed measures (e.g., pseudo-anonymized data sets from telecommunication providers, taxi and company fleet data)
- (Commercial) traffic information (e.g., Here, TomTom, Google)
- Detailed trajectories (e.g., floating car data from fleets, test drives)
- Surveys

Depending on the objective of a study and the type of the model, the suitability of the various sources differs. Moreover, technical and regulatory reasons may negatively affect the availability and quality of available data. From a privacy perspective, an individual's traffic data might be highly sensitive. Macroscopic models typically require a road network with capacities and OD-matrices as input. Outputs are, for example, the modal split, the route distributions, and the flow values. For mesoscopic and microscopic models, there is a need for an input set of trips or even a synthetic population with day plans. Possible outputs are the driven trajectories, the occurring traffic jams, or the duration of trips. Submicroscopic models may need the driving dynamics model and sensor models as input. The road network might need to be modeled in a very detailed way (surface, weather). Desired outputs can aim for a specific ego vehicle and involve, for instance, its generated sensor data or the behavior of integrated assistance systems.

Integration of mobility data into models can happen straight-forward (e.g., importing the network topology into an according data structure). However, more sophisticated processes might require the use of calibration methods. Calibration happens on different levels. If building a general mobility model, one can take empirical data and use it to adjust model parameters for all simulations. In a microscopic car-following model, such fixed parameters include perception times or the probability of speed limit violations. If the model is assumed to be ready and validated, it might be desirable to reproduce a specific real-world scenario based on mobility measurements. For this purpose, there are two approaches. A scenario can

be iteratively repeated with modified inputs (e.g., estimated route choices) until the error between simulated measures and real measures are below a certain threshold (i.e., offline calibration). Another approach is to perform online calibration, where adaptations are done during a single simulation run. This could be necessary, for example, when simulating a live digital twin.

5.2 Using the Approach in the Traffic Domain

The proposed data-centric coupling approach is now being applied to the traffic domain. The required domain and layer definitions are designed by using the provided information on the traffic domain.

5.2.1 Modeling of the Domain Definition

Typical entities in the traffic domain (i.e., traffic participants) are moving over time in a spatial dimension. While spatial relocation of objects happens in many domains, these movements are the inner essence of the traffic domain. Furthermore, for some applications in this domain it is only about these movements. A LP will therefore cover a certain spatial region (see Section 2.3 for fundamentals on DS). Communication is primarily needed when an entity is leaving the region of one LP and entering the region of a neighboring LP. Considering that movements are often modeled (pseudo) space-continuously and are bound to physical laws, the estimated communication overhead between LPs seems to be satisfactory if regions are chosen in a reasonable way. As **spatial partitioning** is a well-accepted partitioning strategy in general [70], it is in particular suitable in this context.

Figure 5.4 depicts examples for different strategies for partitioning a road network (gray) into two partitions. Assuming that the traffic pattern is distributed uniformly, it might be considered to use the topology on the left (A). An advantage of this choice is that there are only two links that connect the separate regimes. In order to split work fairer, the second choice could be made, as both regions cover an equal part of the road network. However, in this setup there are more connecting links that have to be monitored and handled. For a uniform traffic pattern, the third option might be a bad choice. It is even likely that a single entity has to be transferred multiple times between the LPs. Obviously, the partitioning strategy has a significant impact on the overall performance.

The first thing to be modeled is the domain *reference* (see Section 3.4.2). It is tightly connected to the partitioning strategy. We will use a location related element

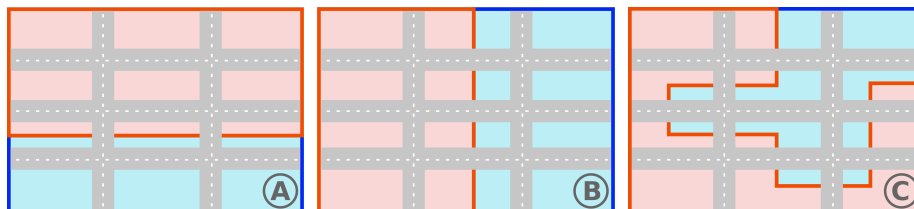


Figure 5.4 – Different topologies as a result of exemplary partitioning strategies.

as a reference object for the traffic domain. It has been discussed that a street map is one of the most elementary data sources of a transport system (see Section 5.1.3). Furthermore, all presented modeling approaches rely on some representation of the road topology. When designing a scenario of a real-world system, it is convenient to convert a real map section into such a representation of the network. Basically, all of the presented off-the-shelf tools model the road topology as a set of vertices and a set of connecting edges. The same graph concept applies to common digital street map formats (e.g., OpenStreetMaps, OpenDrive). As the different elements of road network models are positioned at certain locations, it seems like a reasonable choice to use one of them as the domain reference instead of raw geo-spatial locations.

As mentioned, all of the four presented stereotypical simulation software packages use graph networks. However, when it comes to the details, their concepts for representing the transportation network differs as illustrated in Figure 5.5:

- Visum works with pairs of directed link objects that are grouped by a common id. Each link holds parameters such as the id, the type, the length, the capacity, the number of lanes, the permitted speed, the allowed modes, and the connected nodes (see E_1).
- MATSim has unidirectional links that have an id, a length, a capacity, a number of lanes, a permitted speed, allowed modes, and the connected nodes (see E_5 and E_6). There is no direct grouping of links that are connecting the same nodes.
- SUMO has unidirectional edges that connect two nodes (see E_2 and E_3). An edge contains at least one lane that is defined by an index, a length, a maximum speed value, and optionally a list of three-dimensional vectors that describes the lane's shape [72].
- CARLA uses the OpenDrive standard for representing the road network. A network consists of roads that may have several sections with several lanes in both directions. The geometry is given for the whole road, while each lane has certain offset values [11]. In addition, there are waypoints that have a three-dimensional location and rotation vector. Each waypoint is assigned to a position on a specific street lane (see E_4). Sets of connected waypoints represent a route for navigating vehicles [47].

Consequently, there is no common road concept that is directly usable by all approaches and tools. Neither a link, an edge, a lane, nor a road is well suited to represent a common reference for the whole traffic domain because the available and required parameters differ between paradigms. On the contrary, all of the different road representations do have the understanding that a road is connecting

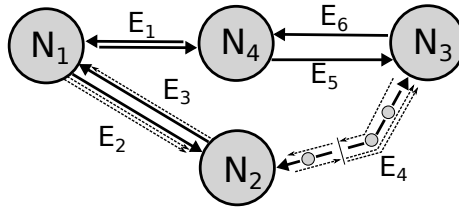


Figure 5.5 – Different possibilities for road representations.

two elements (i.e., vertices). Therefore, we will use the second elementary type of graph structures and thus the domain reference will be a node. A node can for instance be thought of as a junction or a simple connection between two roads. The resulting definition of the traffic domain is given in Figure 5.6 and in Appendix C.1. A node will have a unique id and a spatial position. As a consequence, the responsibility set for all layers in the traffic domain will be a list of node ids.

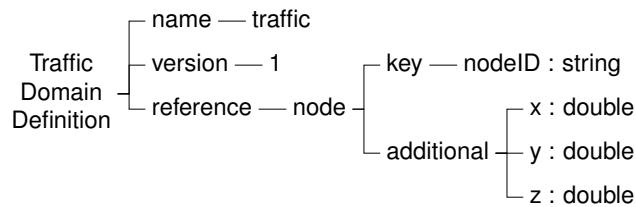


Figure 5.6 – Traffic domain definition.

5.2.2 Modeling of the Layer Definitions

In the following, the layer definitions for the four modeling paradigms are given and described. Our choices regarding the definitions were made based on typical tool capabilities from each field and domain expert knowledge.

Macroscopic Layer

The macroscopic layer will embrace macroscopic components. As described before, this modeling paradigm is about traffic flows. Individual traffic participants are not in focus of the modeling, but the road network. Therefore, the NDM of the macroscopic layer will represent a link between two nodes. The key is a linkID string and in addition there will be eight primitive attributes that describe the status of a link: the flow (in vehicles per hour), the density (in vehicles per kilometer), the average speed on that link (in kilometers per hour), the allowed modes of transportation on that link, the maximum capacity (in vehicles per kilometer), the length (in meters), the contained paths, and the turning probabilities. As many macroscopic procedures are based on differential equations that are solved once per simulation run, we will introduce a method for triggering the recalculation of values. A link is tagged as persistent, observable, and as the distributor element itself. Also, all dynamic link attributes are observable. It is possible to retrieve a full description of specific links, as well as query every single link primitive. On the contrary, we will only permit to set the allowed modes on a link and modify its maximum capacity. The full definition can be found in Appendix C.2.

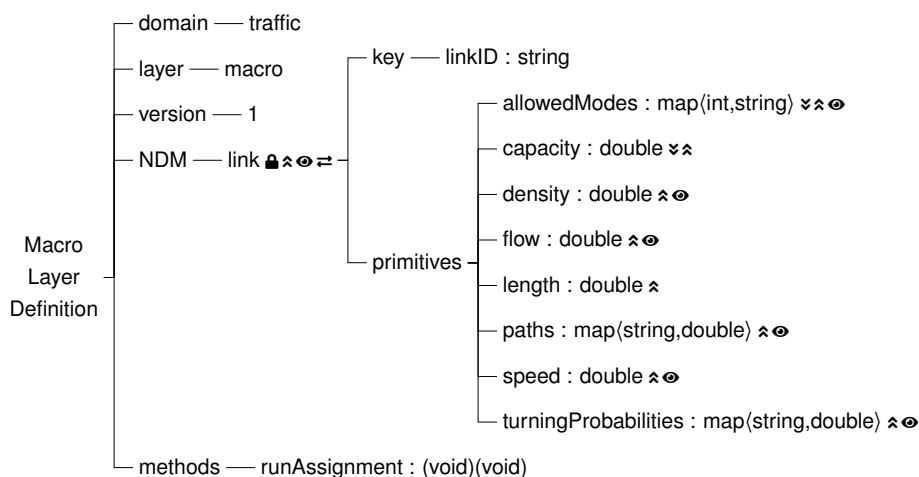


Figure 5.7 – Definition of the macroscopic layer.

Mesoscopic Layer

From the mesoscopic layer on, there is a focus-shift from the road network towards the traffic participant. Therefore, the mesoscopic NDM will describe vehicles. A mesoscopic vehicle contains a vehicleID, is located on a certain link, has a route, and is described further by a vehicle type. One could approximate further attributes (e.g., the vehicle's speed) incorporating link statistics, but as these attributes do not originate from the vehicle model itself, that does not seem reasonable. As the focus is shifted to the vehicle, the simulation distribution mechanism will not work by publishing link attributes, but by transferring individual vehicles. Besides the NDM, we add a link compound to the layer definition. A link entity has an identifier and can host multiple vehicles. Furthermore, a link is described by the current occupancy (in vehicles per kilometer), the flow (in vehicles per hour), and the average speed of passing vehicles (in kilometers per hour). While a link will exist persistently during the scenario run, a vehicle is tagged as non-persistent. A link, its primitives, a vehicle, and the current link of a vehicle can be observed. Each primitive attribute is gettable, while only the current link and route of a vehicle is settable. The list of vehicles on a link is tagged as the distributor element. The full definition can be found in Appendix C.3.

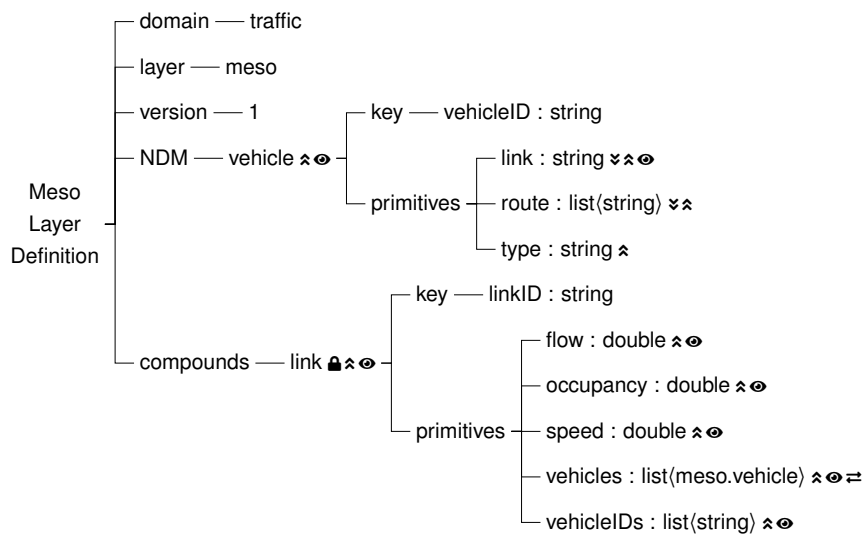


Figure 5.8 – Definition of the mesoscopic layer.

Microscopic Layer

The microscopic layer definition will be similar to the mesoscopic layer, while adding more detail. Therefore, the logic regarding the transfer of NDMs will also be comparable. However, as a microscopic vehicle will hold information about its position on a certain edge, ghosting can be implemented reasonably (see Section 5.1.2). Besides the position on an edge, a vehicle is identified by a vehicleID, and has an acceleration (in meters per square second), an angle (in degrees), an edge, a lane, an absolute position vector, a route, a slope (in degrees), a speed (in meters per second), and a type value.

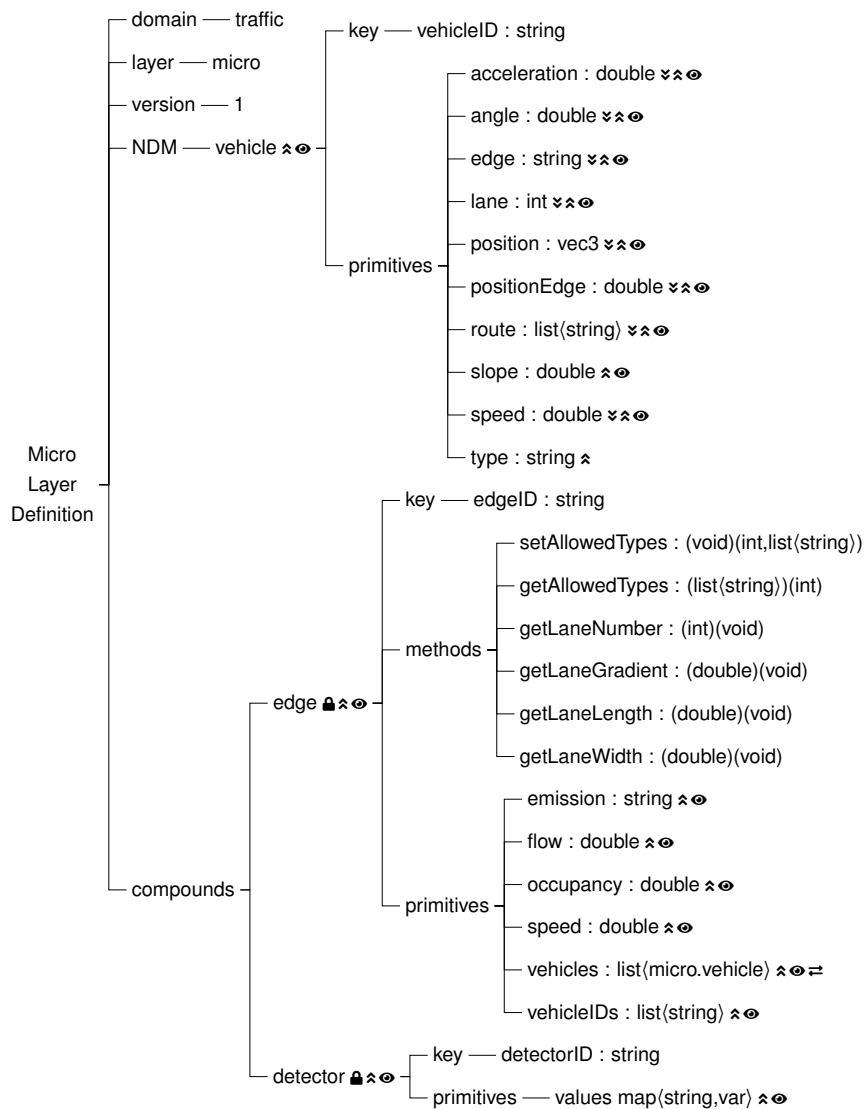


Figure 5.9 – Definition of the microscopic layer.

Furthermore, there will be an edge compound that consists of an emission string, a flow value (in vehicles per hour), and a speed value (in meters per second). It also holds a list of contained vehicles and a list of contained vehicle ids. Another compound will represent detectors, which are common in microscopic models. As their nature differs greatly from model to model, the detector compound will be very general and hold only an identifier string and a generic value map.

One can query all primitive attributes and also the full vehicle, edge, and detector tuples. There are many possibilities to modify a vehicle. It is possible to set the acceleration value, the angle, the edge, the lane, the position, the route, and the speed. No primitive attributes of the edge or the detector compound are settable, because these attributes are only derived from the microscopic vehicles and not modeled explicitly. As for the mesoscopic layer, the distributor element will be the vehicle list of an edge. The full definition can be found in Appendix C.4.

Submicroscopic Layer

In the submicroscopic layer, the focus lies still on the vehicle. Again, we add additional details to the vehicle compared to the microscopic and the mesoscopic layer. Once more, the transfer logic will be based on the information of vehicles on lanes that are (not) in the responsibility set. As the physical vehicle models are assumed to be far more detailed than in the microscopic case, the NDM holds relative values for throttling, the steering angle, braking, and the current gear. Also, already known values such as acceleration are now represented as three-dimensional vectors (e.g., in meters per square second). The reason for this is that multiple forces might be exerted on a vehicle and that a vehicle does by no means have to follow the road track. In contrast to the microscopic definition, the laneID is a scenario-wide descriptor string. A lane compound mainly provides information about vehicles that are currently present, but also about their average speed (in meters per second). In addition, its vehicle list is tagged as distributor element.

The full NDM and all primitive attributes can be queried. Besides the static vehicle type, the laneID, and the contained sensors, the vehicle attributes can be modified by setters. The vehicle and the lane tuple can be observed. Also, all vehicle and lane children are observable except for the static type and the route. The full definition can be found in Appendix C.5.

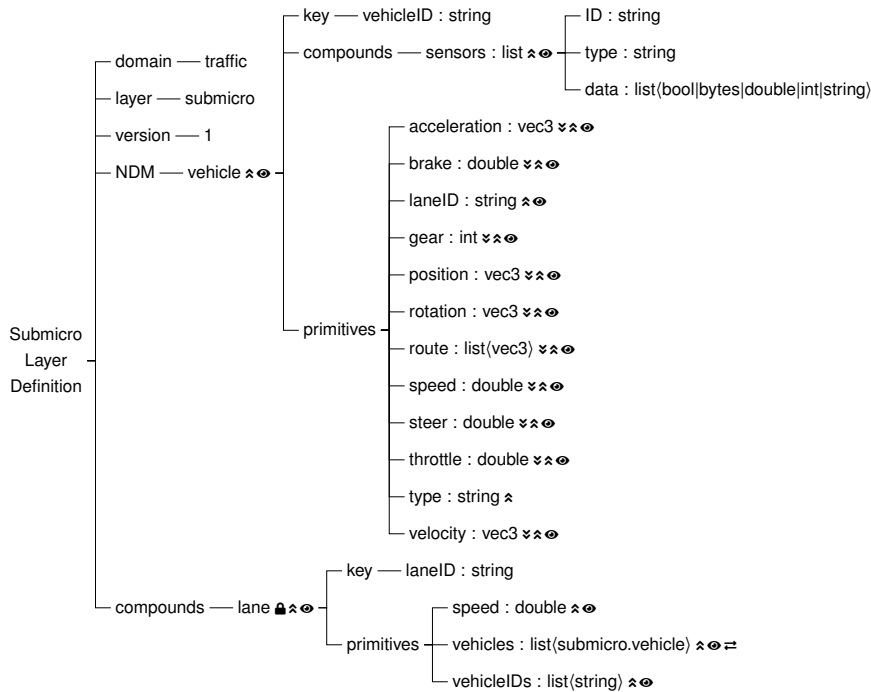


Figure 5.10 – Definition of the submicroscopic layer.

5.2.3 Modeling of the Translators

The combination of multiple layers requires not only the modeling of layer definitions, but especially the modeling of translation rules between those layers. The purpose of a translator is always to create a mapping between the NDM definitions of two layers in order to translate NDMs back and forth. As a proof-of-concept, in the following, we describe the modeling of translators for all neighbored layers (i.e., macro-meso, meso-micro, micro-submicro). All of them are implemented using the presented `JavaBaseWrapper`.

Macroscopic \leftrightarrow Mesoscopic

One of the most interesting challenges of the translator between the macroscopic and the mesoscopic layer is the connection between a time-continuous and a time-discrete modeling paradigm. Therefore, its logic is not representable by a classic function with a 1:1 relation between input NDM tuple and output NDM tuple. Hence, it will not be reasonable to create a dedicated output message for each received input message by solely converting the input using a translation rule as it will be possible for the following two translators. Another cause of this peculiarity is that a mesoscopic NDM is representing a single vehicle, while a macroscopic NDM is representing a road. Therefore, also a shifted focus for the NDMs has to be taken into account. This is also accompanied by a different modeling perspective. While the system is described top-down in the macroscopic case, the system behavior arises bottom-up in the mesoscopic case.

For the aggregation (i.e., meso \rightarrow macro), information about each mesoscopic vehicle is stored until a macroscopic sample window is over. Afterwards, speeds, amount, and routes of all collected vehicles are averaged per transfer link and thereby the generation of a macroscopic representation of each relevant link is straight-forward. The mesoscopic *vehicle.link* attribute is therefore linked to the key of a macroscopic tuple (*link.linkID*) and vice versa.

For the disaggregation (i.e., macro \rightarrow meso), it is required that the existence and the state of vehicles is inferred from macroscopic data describing roads. While the number of vehicles to be created is easily accessible, the most basic question is how to sample the spawn times (i.e., arrival times) of vehicles that enter the mesoscopic regime. Equidistantly distributed inter-arrival times could lead to very artificially appearing mobility patterns. Another approach for sampling arrival events in a discrete system is to incorporate a Poisson Arrival Process (PAP). This requires that the arrivals are assumed to be independent of each other. A well-known application example for this is the modeling of incoming calls in a phone network [117]. Therefore, we will utilize a PAP to sample discrete vehicle arrivals based on a link's traffic flow, which results in exponentially distributed interarrival

times. Using Equation 5.2, we can calculate the probability that n arrivals happen during a certain period t . The arrival rate is given as λ , which is directly representing the traffic flow. If there would be for instance an incoming flow of 2 vehicles per minute, λ would be $2 \text{ veh}/\text{min}$. If we are interested in sampling at which point in simulation time the next arrival is likely to happen, we estimate the next exponentially distributed interarrival time. In order to do so, the Cumulative Distribution Function (CDF) is used (Figure 5.11).

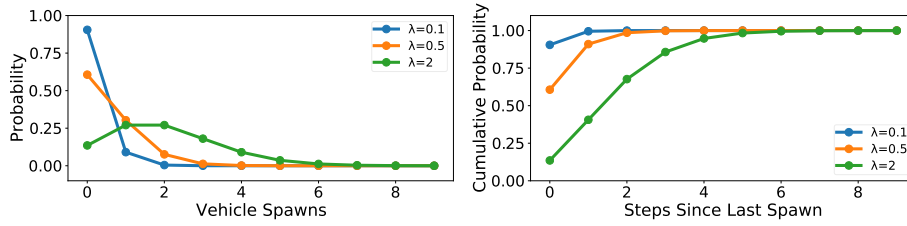


Figure 5.11 – Spawn probabilities and cumulative representation.

In the translation process, we would then draw a random number $r \in (0, 1)$ from a uniform distribution for each arrival. We map this random number r to an interarrival time i by using the inverse CDF (Equation 5.3) as depicted in Figure 5.12 and are able to schedule a new spawn event. This makes sense if we face a relatively low traffic flow. If we only have sparsely scheduled spawn events, we can benefit from not having to synchronize on every mesoscopic simulation step, but only on our next scheduled spawn event in order to send out the corresponding mesoscopic NDM. However, this only works flawlessly with a stationary arrival rate. When considering the translation of traffic, a varying λ might be very likely. For a changing λ , our PAP is called non-homogeneous. An exemplary sampling result with a changing arrival rate is depicted in Figure 5.12. In order to consider such varying conditions, it requires additional logic to interpolate between an old and an updated arrival rate for an already scheduled spawn event. Otherwise, we would most likely never recover from a phase with an arrival rate of (almost) zero, for example.

$$P\{N(t) = n\} = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad (5.2)$$

$$CDF(t) = 1 - e^{-\lambda t}, \quad CDF^{-1}(r) = \frac{-\ln(r)}{\lambda} = i \quad (5.3)$$

Therefore, we also implement a more straight-forward approach that is especially suitable for higher arrival rates. The translator will use the smallest simulation step length that is found in a scenario definition as an own synchronization step length. With that, we do not have to schedule future arrivals, but simply can use the current arrival rate and Equation 5.2 in each step to figure out the amount of NDM tuples that should be published.

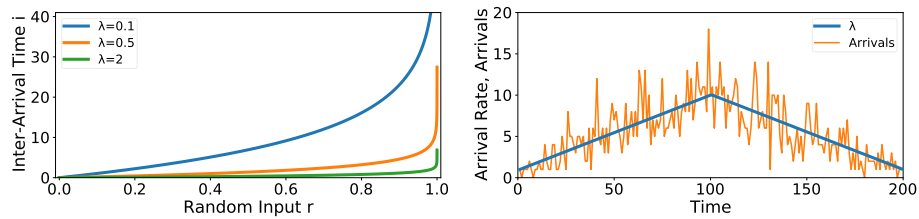


Figure 5.12 – The mapping of an input value r to an inter-arrival time i and the outcome of a non-homogeneous PAR

In addition to the spawn time, a mesoscopic vehicle requires a route. In order to provide some flexibility, we incorporate two mechanisms to sample a route from a macroscopic NDM. If the NDM contains path information, we use the provided information to compute the probabilities of following a certain path and simply draw a random path for each vehicle. In case that there is no path information, we have no direct access to global system knowledge about traffic patterns. Therefore, we have to subscribe to the turning probabilities of all links. Based on this information we build a map of turning probabilities that allows rolling a dice at every intersection and thus assemble a random route piece by piece. As we already discussed in Section 5.2.1, the common element of all layers is not the edge, but the node as there are various concepts for modeling the links of a road network topology. Therefore, another important task of the translator is to convert macroscopic link ids to mesoscopic link ids and vice versa. If reasonable, a vehicle type can be sampled from a mode probability table (e.g., 5 % trucks, 30 % aggressive cars, ...). Such a table is specified as a scenario resource, as the probability distribution is highly scenario dependent.

Mesoscopic \leftrightarrow Microscopic \leftrightarrow Submicroscopic

Compared to the macro-meso translator, the meso-micro translator is more straightforward as we have a 1:1 relation between incoming and outgoing NDMs in both directions. The translation happens mainly by adding or dropping information pieces to already existing discrete vehicle representations.

For the aggregation (i.e., micro \rightarrow meso), all attributes that originate in the microscopic car-following model (i.e., acceleration, angle, lane, edge, position, positionEdge, slope, speed) are dropped. As for the macro-meso translator, the microscopic edge identifier is converted to its matching counterpart in the mesoscopic regime. Also, the route is translated in the same way.

For the disaggregation (i.e., meso \rightarrow micro), we are populating attributes that will be updated frequently in a microscopic regime such as acceleration, position, or slope with default values. The mesoscopic linkID is converted to the according microscopic edgeID. The same applies to the route, while the vehicle type is kept.

Regarding the target lane, for instance, the highway capacity manual states that it is not possible to provide general lane distribution patterns [159]. Being aware that the microscopic lane change models will move a vehicle anyway (if necessary and possible) we use a fair circular pattern for choosing an initial lane in order to utilize the existing space on the road well.

The micro-submicro translator works similarly. A mapping between submicroscopic laneIDs and microscopic edgeIDs is used for the conversion. NDM elements that cannot be represented in the microscopic NDM (e.g., the gear) are dropped. In addition, basic math is used to convert attributes such as acceleration. They are represented as a three-dimensional vector in the submicroscopic regime, while we mostly consider the vectors' lengths in the microscopic regime:

$$micro.acc = \sqrt{submicro.acc.x^2 + submicro.acc.y^2 + submicro.acc.z^2}$$

5.3 Implementing Wrappers

In Section 5.1.1, we described a typical traffic software package for each modeling paradigm. All four will be integrated in the developed simulation system. In the following section, we give a brief overview on the implementation of their wrappers. These insights are useful if the existing wrappers should be modified. Also, if a new simulator should be integrated, such an impression of potential workflows might be helpful.

PTV Visum

PTV Visum [181] is a commercial closed-source application that provides an API over a COM-Interface. Although it is not very convenient to use other languages, a COM-interface does not necessarily restrict the developer to use languages from the Microsoft platform (e.g., C#). It can also be used via suitable Python or Java libraries such as JACOB [83]. As there is currently no Linux version of Visum, we are using the Windows Version of Visum 2022 and build the Visum wrapper on top of our JavaBaseWrapper. Hence, we are using JACOB 1.20 in order to utilize the COM-Interface. As the officially provided JACOB jar is built with a Java 8 compiler, one has to build it manually for Java 11 in order to be compliant with our JavaBaseWrapper.

In order to be able to represent time dynamics, we considered two non-static traffic assignment procedures of Visum: Dynamic User Equilibrium (DUE) and Simulation-based Dynamic Assignment (SBA) [182]. We decided to use the DUE procedure, because it allows modifying the maximum link capacities. This will be important to ingest traffic information from other layers back into the macroscopic simulation. However, setting the capacity is only possible over the whole scenario duration and not for specific time periods. Therefore, recalculations of the whole assignment procedures might be necessary during the overall simulation.

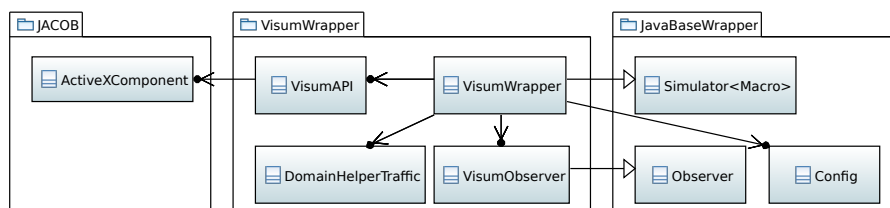


Figure 5.13 – Class diagram of the VisumWrapper.

Although the DUE is based on using individual travel paths, the current Visum implementation allows only querying the total flow rates over the whole scenario and not for specific time steps. As we face the same issue for the capacity, we will

deal with this constraint. With the SBA, we could retrieve path flows per time step, but as a drawback, there is no way to manipulate the capacities of roads.

VisumWrapper is the main class and extends the *Simulator* class of the *JavaBaseWrapper* package (see Figure 5.13). Via the API we can specify traffic assignment methods with their parameters and trigger the execution of the traffic assignment calculations. Afterwards, we can access the computed macroscopic numbers by jumping to predefined discrete sampling intervals. On a *processStepEvent(t)* call, for each outgoing link a macroscopic NDM containing the link's state at time *t* is queried using the *VisumAPI* class. The NDM tuples are published to corresponding topics using the *provisionHandler* that is provided by the simulator base class. On *postStepEvent(t)*, custom observers are triggered. The *VisumAPI* class provides the bridge between the wrapper and Visum via the COM-interface. An impression on the usage of the offered COM-interface is shown in Listing 5.1.

```

ActiveXComponent ax, oNetLinks;
void init(){
    ax = new ActiveXComponent("Visum.Visum.22");
    oNetLinks =
        ↪ ax.getPropertyAsComponent("Net").getPropertyAsComponent("Links");
}
Dispatch getLink(long from, long to) {
    return Dispatch.call(oNetLinks, "ItemByKey",String.valueOf(from),
        ↪ String.valueOf(to)).toDispatch();
}
double getFlow(MacroLink link){
    Dispatch link = getLink(link.getFromNode(), link.getToNode());
    Variant v = Dispatch.call(d, "AttValue",
        ↪ "VOLVEHPRT("+time+)");
    double flow = v.getDouble() * (60 / stepLengthInMinutes);
    return flow;
}
void setCapacity(MacroLink link, double flow) {
    Dispatch link = getLink(link.getFromNode(), link.getToNode());
    Variant variant = new Variant();
    variant.putDouble(flow);
    setValue(link,"Capprt",variant);
}

```

Listing 5.1 – Exemplary Visum COM-commands via JACOB.

MATSim

MATSim [113] is published under an open source license and is developed in Java. Therefore, we build the *MatsimWrapper* as well on top of the *JavaBaseWrapper*. We use MATSim in version 12.0. In contrast to Visum, MATSim offers no dedicated

API. Instead, a comprehensive event listener approach [17] was designed by the developers that can be utilized when adding custom extensions.

Using the event listeners, we cannot stick to the general simulation workflow that is provided by the *Simulator* class of the *JavaBaseWrapper*. For instance, the *MatsimWrapper* has to run its final initialization routines when `notifyMobsimInitialized()` is called by MATSim. In addition, it is not possible to proceed the simulation by triggering a next simulation step. On the contrary, we only can delay the proceeding of the simulation that is carried out by the MATSim logic. Thus, we replace the scenario loop logic of the *Simulator* by executing the required tasks for one simulation step every time `notifyMobsimAfterSimStep()` is called as depicted in Listing 5.2. In addition, we subscribe to the *LinkEnterEventHandler*. Thus, MATSim calls `handleEvent(LinkEnterEvent)` each time a vehicle enters a new link. In this handler function, we check for each entering vehicle if the new link is not in the BBs responsibility region. In such case, we generate a mesoscopic NDM using the *MatsimAPI* class and publish the tuple. In the local MATSim state, we let the vehicle end its trip on the current link. When a mesoscopic NDM is received, the *MatsimWrapper* is calling `addAgentToMobSim()` of the *MatsimAPI* class in order to adopt the received vehicle in the local simulation state. The mentioned components can be found in Figure 5.14.

```
public void notifyMobsimAfterSimStep(MobsimAfterSimStepEvent e) {
    long t = TimeSync.getLocaltime();
    simulator.preStepEvent(t);
    simulator.stepEvent(t);
    simulator.processStepEvent(t);
    simulator.synchronizeEvent(t);
    simulator.postStepEvent(t);
}
```

Listing 5.2 – Main simulation loop of the *MatsimWrapper*.

SUMO

SUMO [8] is also an open source project and is written in C++. It provides an API called TraCI, which will be used by our *SumoWrapper*. Originally, interaction was only possible via network sockets. Since SUMO 1.0, there is also a C++ API library called *Libsumo* that supports similar function calls, but comes without the network overhead. We use SUMO 1.13 and *Libsumo* for our wrapper.

As we are using the C++ API library, we will build the wrapper on top of our *CppBaseWrapper* library. With the offered API functionality, we are in full control over the simulation and can trigger the next simulation on our own. The simulation loop in the *SumoSimulationControl* class is proceeding SUMO via the *SumoAPILibsumo*

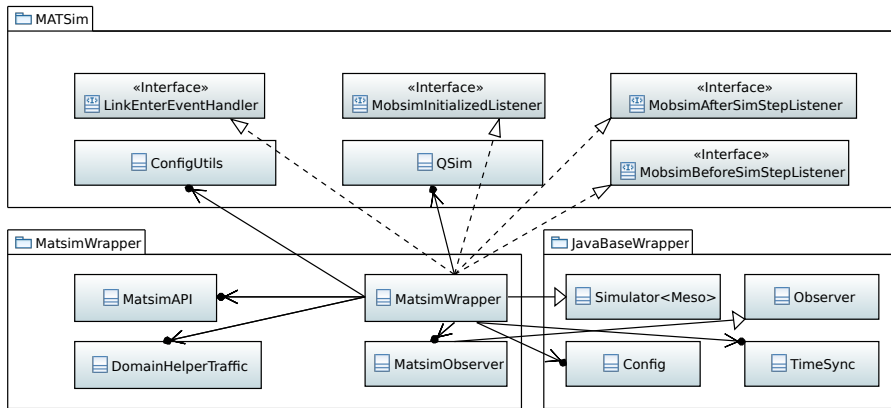


Figure 5.14 – Class diagram of the MatsimWrapper.

class. Afterwards, for each vehicle in the local simulation state that is not marked as a ghost, we check if its current edge is still in the local responsibility set. If not, we tag the vehicle as a ghost, create a microscopic NDM representation of it, publish the tuple, and let the vehicle finish its existence in the local simulation world after passing the current edge. With the tagging as a ghost, a vehicle has officially left the simulation scope of the BB and will not appear in observer outputs.

If ghosting is enabled per scenario parameter, and there is another BB that is covering the outgoing edge, we use the attribute updates of the receiving BB to update the attributes of their ghost representations in the out-sending BB. Therefore, a ghost represents a perfect copy of an active vehicle in another instance, as long as that vehicle is still on the transfer link. As a next step, we request a time advance and announce all messages that were published by us. When the synchronization is finished, we process all incoming messages (e.g., position updates for ghosts or incoming vehicles) and finally run custom observers before starting with the next iteration. Figure 5.15 illustrates the main components.

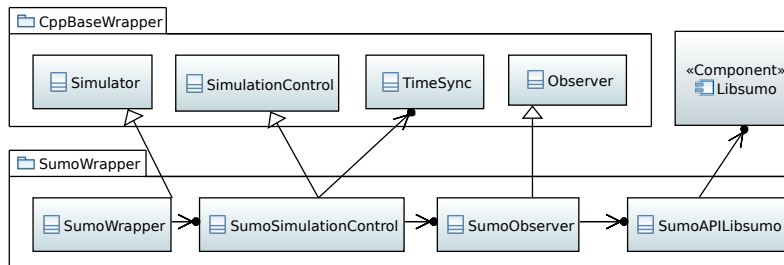


Figure 5.15 – Class diagram of the SumoWrapper.

CARLA

Similarly, CARLA [74] is an open source project that is written in C++. Besides a C++ API, there is also a Python API for interacting with the simulation core. In order to demonstrate the flexibility of our approach, we will use the Python API and build a wrapper on top of our PythonBaseWrapper library.

The architectural design of CARLA is mainly split in a server and a client component. For the wrapper, we create a client script that connects to the CARLA server and runs initialization routines (e.g., loading the scenario map). The following procedure resembles that of the SumoWrapper. Iteratively, we trigger a new simulation step on the CARLA server component and process the updated state. If there are vehicles on outgoing roads, we publish representing submicroscopic NDMs and end their journey in the local world. After synchronizing, we adopt incoming submicroscopic NDMs into the local simulation state and run observers. A brief orientation is presented in Figure 5.16.

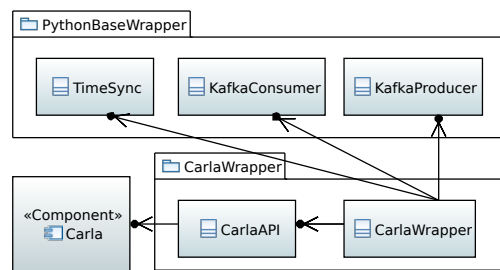


Figure 5.16 – Class diagram of the CarlaWrapper.

5.4 Extension to a Further Domain

As a last section in this chapter, we want to demonstrate the extendability of the approach by expanding to another domain using the domain of communication networks. Such a domain might address the question if and under which conditions communication happens between several communicating nodes. Different layers in the domain might consider various levels of detail (e.g., simple probabilistic approaches or detailed packet-level models) and different technologies (e.g., 802.11p or 5G). Developed definitions can be found in Appendix D.

Definitions for the Communication Domain

Regarding the domain definition, a common reference element is required that can be understood by all layers of the domain. It does not seem reasonable to use a very spatial-related element such as grid cells, as most communication flows will travel nearly with light speed. A partitioning based on spatial cells could make sense, if only isolated radio cells are considered, for instance. However, implementing a useful partitioning on top of such regions might not work well in general, as communication messages might propagate almost instantaneous within a whole system. In contrast to the traffic domain, we therefore use a logical network as the domain *reference* for all communication layers, which will basically be a set of nodes that might be communicating with each other. With that, several networks may co-exist and can be used to form logical partitions. A network may be further described with a textual description and a technology identifier.

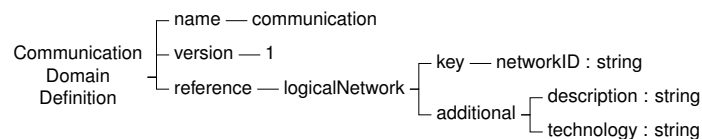


Figure 5.17 – Communication domain definition.

We will design a layer for the wireless communication standard IEEE 802.11p. The reason for choosing this layer is that it was designed for vehicular communication. Thereby, we clearly recognize the link to the traffic domain and are in the position to demonstrate a cross-domain study in the following chapter. The *80211p* layer focuses on radio messages. The NDM is represented by a message tuple that contains information about the networkID, the sender, the receiver, the type, the stage, the (radio) channel, the priority, and the payload. The message stage is used to distinguish between events, where messages were sent or were received. As a radio message is highly transient, it will not be possible to get or set message attributes nor to observe specific attributes of a message. The only possibility is to observe all

generated messages in general. Besides the NDM, there is also information about the environment. There is a compound related to existing logical networks that gives information about sent and received messages within the logical network. It builds the connection to the domain reference element. The *received* field is used for the coupling mechanism and is therefore tagged as the distributor element. All network child attributes are gettable and observable. A logical network is supposed to be persistent during a scenario run, which does not mean that it has to host nodes all the time. In addition, there is a compound that is observable and represents all nodes in the environment. The position of each node is settable, gettable and observable.

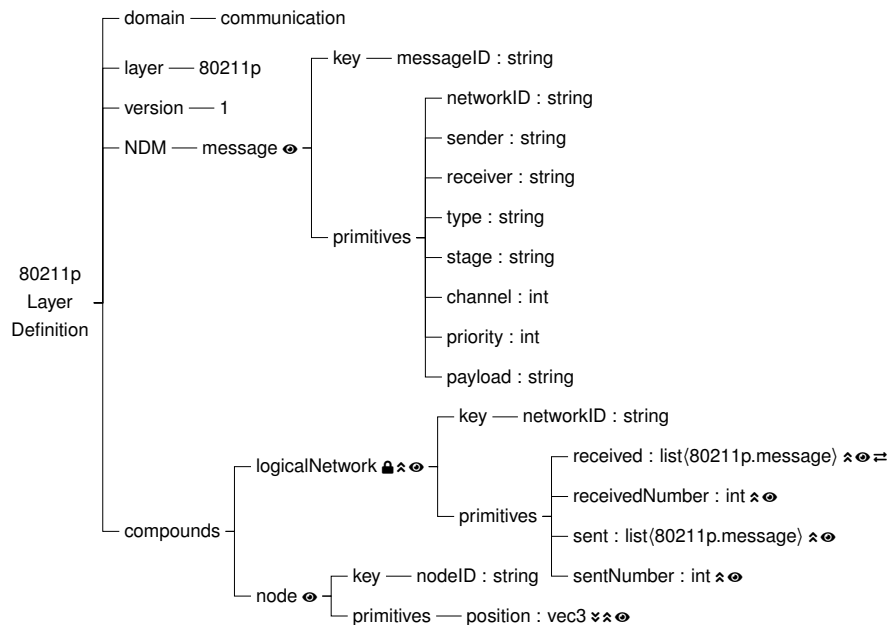


Figure 5.18 – Definition of the 80211p layer.

We model a projector that links the communication layer to a layer from the traffic domain. A detailed wireless communication model for vehicular ad-hoc networks depends strongly on the positions of the sending and receiving nodes, which would be offered by both the microscopic and the submicroscopic layer. Considering simulation performance, we will focus on the microscopic layer and create a projector between the microscopic layer and the 80211p layer. The main task of the projector is to create representations of new vehicles in the environment of the communication model and remove them when vehicles leave the simulation. In addition, the projector continuously has to retrieve the positions of vehicles and trigger position updates of nodes in the communication model. In order to get the needed information, the projector will subscribe to Topic 5.4. Based on an internal log, the projector can infer

the required actions: add a new node, remove a node that is not present anymore, or update the position of a node. Each action will result in an interaction request by publishing a corresponding message to Topic 5.5.

```
provision.[sceID].traffic.micro.vehicle.position (5.4)
```

```
interaction.[sceID].communication.80211p.request (5.5)
```

OMNeT++ Wrapper Implementation

Finally, we implement a component for the developed layer. OMNeT++ is a well-known DES framework in the context of communication simulations. Many protocols were already modeled as extension to the framework. There is for instance an IEEE 802.11p model that was added as part of the Veins project [206]. We use parts of the Veins library and the OMNeT++ simulator in version 5.6.1 in order to provide an 802.11p communication simulator in our system. The OMNeT++ environment is using C++. That is why we are building the wrapper on top of our CppBaseWrapper. There is no dedicated API provided by OMNeT++. An OMNeT++ user is implementing the desired features within the OMNeT++ framework and is able to modify existing parts of the already provided DES functionality. According to the existing architecture, we modify the default discrete event scheduler of OMNeT++ and ingest our synchronization logic. Inspired by the Veins workflow, there is a *ScenarioManager* that takes care of updating the communication nodes based on messages that are provided by the *OppProvisionImpl* class. Another important part is the *TriggerMsgApp* class that is inherited from the *BaseAppLayer* class of the Veins

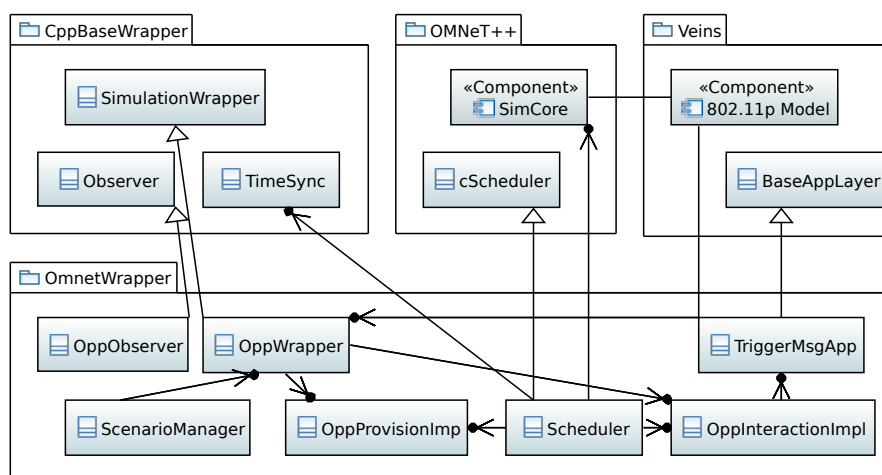


Figure 5.19 – Class diagram of the OmnetWrapper.

library. It represents a proxy application that runs on each simulated network node and has two functions. First, it is able to notify the OmnetWrapper that a radio message was successfully received by a node. This information can then be published on the provision channel if corresponding observers are specified. Second, it is used to trigger the transmission of a radio message if such an action was requested via the interaction channel. The main components are depicted in Figure 5.19.

Chapter 6

Exemplary Case Studies in the Traffic Domain

In the following, we demonstrate the capabilities of the developed system and its application to the traffic domain. Five exemplary simulation studies cover and make use of the system's main features: distributed simulation, multi-level simulation, data enrichment, integration of real-world data, data reuse, and cross-domain simulation. In order to illustrate the different aspects, we give examples of situations in which a certain feature might be desirable.

6.1	Study 1: Distributed Simulation	149
6.1.1	Validation	149
6.1.2	Performance Gains	152
6.2	Study 2: Multi-level Simulation	155
6.3	Study 3: Data Enrichment	162
6.4	Study 4: Integration of External Data	167
6.5	Study 5: Cross-Domain Simulation	171

6.1 Study 1: Distributed Simulation

In the first study, we cover the simulation distribution abilities. We consider two different aspects in this light. In a first step, the correctness of a distributed traffic simulation is assessed by comparing a naive and error-prone approach with a valid traffic distribution approach. After that, possible speed-ups using the correct version are evaluated. The scenario definitions can be found in Appendix E.1.

6.1.1 Validation

The general correctness of the coupling approach (i.e., mainly the synchronization mechanism) was demonstrated in Section 4.3.2. However, depending on the application we also mentioned that additional measures might have to be taken if the exchange of NDMs also depends on additional external conditions. In the road traffic context, such a condition is the state of the transfer link. If there are for example slow vehicles at the beginning of the transfer link, approaching vehicles typically already brake on the preceding link. Furthermore, the receiving link could also be completely congested. In such a case, it will not be realistic to transfer the NDM at all as a vehicle cannot move forward. This specific problem was already addressed in Section 5.1.2 using the term *ghosting*.

Using the observer concept, the integration of a ghosting approach with our system works flawlessly. In the following, we will use a simple scenario to illustrate the importance of such a feedback mechanism. The road network consists of five 100 m long road stretches that form a straight road line of 500 m (see Figure 6.1). Fast vehicles will drive from the start to the end of the road (i.e., starting at 0 m and leaving at 500 m). In addition, there will be one slower vehicle that spawns on the centered road stretch, pauses for 20 seconds, and drives to the end of the road. The fast vehicles are illustrated as a red rectangle, and the slower vehicle as a blue rectangle. The scenario is equally partitioned into two regions (red and blue), which means that the central road is the transition link. Both LPs will be simulated by two microscopic SUMO instances (*Instance0*, *Instance1*), respectively. As vehicles drive from the left to the right, the red instance is responsible for the first two road stretches, while the blue instance is covering the three others. For the opposite direction, the blue instance would be responsible for the roads between 300m and 500m, while the red instance would cover the three other stretches.

As a result, the blue vehicle is spawned in the blue regime and the red instance would never know about its existence if there would be no ghosting concept. This is not necessarily a problem. For instance, if the traffic situation is relaxed enough, there is a high probability that no inconsistencies would arise. However, without a feedback mechanism, we cannot guarantee that distributing the traffic simulation

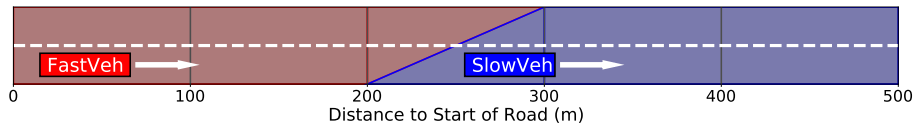


Figure 6.1 – Line scenario with five connected road stretches.

will lead to results that match the non-distributed runs. A typical problem is a traffic jam that is propagating over the partition boundaries. One example that illustrates this is shown in Figure 6.2. The distance of the vehicles to the start of the road is given on the y-axis, while the simulation time is progressing on the x-axis. The reference trajectories are produced by simulating the same scenario with a single instance of the same microscopic traffic simulator.

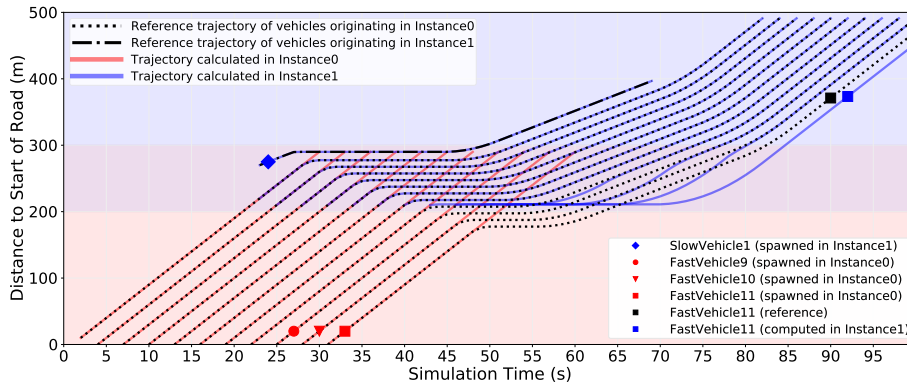


Figure 6.2 – Distributed micro simulation that produces wrong trajectories due to missing ghosting features.

Because there is no awareness of the slow-driving blue vehicle in the scope of the red instance, the red instance is sending out vehicles as there would be no congestion on the third road stretch. In the non-distributed case, *FastVehicle9*, *FastVehicle10*, and *FastVehicle11* would already start to slow down at the end of the second road stretch. Consequently, this leads to inconsistent results with *FastVehicle11* being delayed for two seconds. Obviously, this can lead to arbitrarily large errors, for instance if the transfer link is completely congested for a significant amount of time.

In contrast, one can see the perfect match between both instances and therefore also between the non-distributed reference and the distributed simulation in Figure 6.3 if the ghosting feature is activated. Due to the ability of integrating external status updates on the border regions, the propagation and representation of traffic situations across system borders is now possible. Using this feature, the overall results are consistent with the non-distributed baseline run as desired.

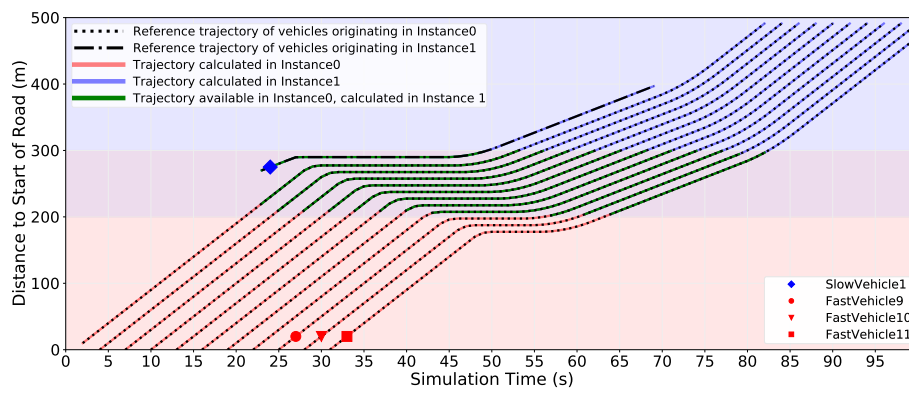


Figure 6.3 – Distributed micro simulation that produces correct trajectories by using ghosting features.

6.1.2 Performance Gains

One reason for using DS is to split the work across multiple computing nodes so that hardware constraints (e.g., main memory) of a single node can be overcome. For example, the required main memory of a single node could decrease accordingly if the executed simulator instance has to represent and simulate half as many vehicles as present in the overall scenario. Another reason for using DS is to speed up the overall simulation by taking advantage of the parallelization effects that arise. As the involved communication and synchronization tasks of the developed approach require a non-negligible effort, we will assess exemplary speed-ups that are resulting from the offered distribution possibilities in this study.

Assumption Let us assume that we are interested in the performance impact of using different numbers of instances and partitioning strategies when running a microscopic traffic simulation of a Manhattan grid with a random traffic pattern. The used topologies are shown in Figure 6.4. Each LP is covered by a single SUMO instance.

Requirements We need a microscopic traffic model, the traffic input, and the road resources.

The main goal of this study is to demonstrate that speed-ups are possible, but that the overall performance is at the same time highly dependent on the scenario and the partitioning strategy. Furthermore, we are using the Manhattan grid scenario with uniformly distributed traffic flows because we are considering it as a challenging scenario for such an application. Due to the traffic pattern and the road topology, there are no inherent regions that are forming largely self-contained traffic clusters. When facing traffic clusters with a reduced amount of inter-vehicle traffic, an even

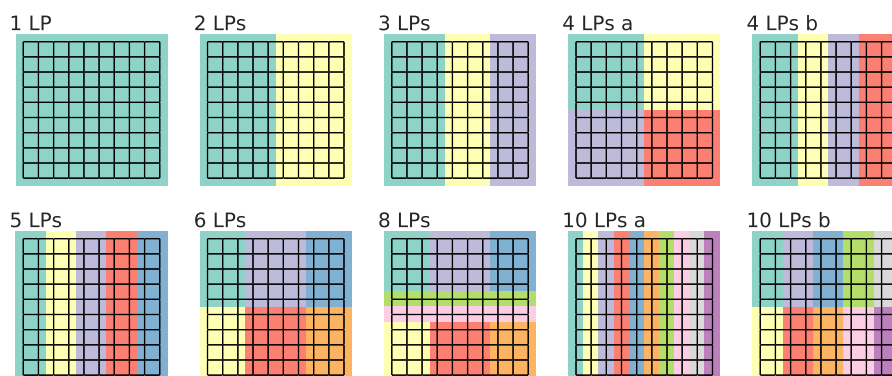


Figure 6.4 – Different topologies for the distributed simulations.

better performance is conceivable. Such clusters might be found in real-world maps as they can contain for instance rivers, self-contained districts, or freeways.

Topology	Total	Synchronize	Process Outgoing	Simulate	Process Incoming
Baseline	46.23	0.00	0.00	0.00	0.00
1 LP	50.45	0.01	0.00	50.44	0.00
2 LPs	40.01	9.18	0.54	28.30	2.12
3 LPs	45.54	19.53	0.56	23.39	2.06
4 LPs a	29.14	10.40	0.64	15.58	2.78
4 LPs b	40.33	19.33	0.63	17.92	2.30
5 LPs	43.83	28.40	0.55	12.05	2.46
6 LPs	33.26	20.59	0.56	10.02	2.32
8 LPs	58.22	46.60	0.55	8.49	2.03
10 LPs a	96.00	80.64	1.28	9.20	5.41
10 LPs b	54.30	44.60	0.47	7.44	2.00

Table 6.1 – Median values of scenario run effort in seconds.

The simulations are executed on an HP ProLiant DL380 G7 machine with 12 physical Intel Xeon X5690 cores that is operated by Ubuntu 20.04.4 LTS. For each topology we are conducting 25 simulation runs and measure the required amount of wall-clock time for each run. The results are given in Table 6.1 and illustrated in Figure 6.5. A classical SUMO simulation that is executed without using any of the concepts that were developed in this work is building the baseline. As expected, the 1 LP runs are slower than the baseline, because no parallelization can happen. At the same time, the framework's overhead is already added. However, using such a topology might still be useful, as the execution is orchestrated easily by the developed service and the observer concept can be used for simple generation and processing of

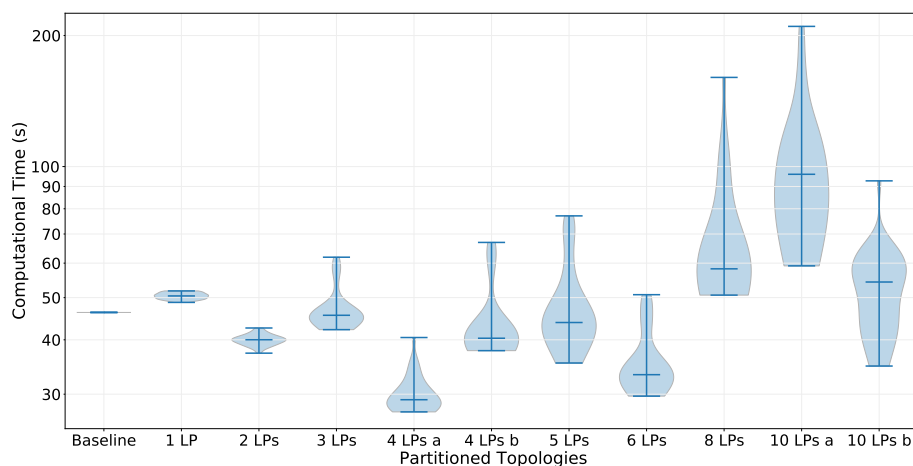


Figure 6.5 – Scenario runtimes of different topologies.

simulation results. Starting from the 2 LPs topology until the 6 LPs topology a benefit regarding the overall execution duration is observable, either for all repetitions or at least for the average values. The best suited topology for increasing the performance seems to be 4 LPs a. Topologies 8 LPs, 10 LPs a, and 10 LPs b do not make any sense from a performance point of view as most runs are even slower than the 1 LP topology.

In addition, we tracked the respective shares of computing the next simulation state, synchronizing instances, processing states for publications, and ingesting incoming entities in the total amount of time. The results are depicted in Figure 6.6. Nearly all time is spent for pure simulation for the runs with one LP. Starting from the runs with two LPs to the runs with ten LPs, we see a trend of an increasing share of time that is needed for synchronization. That can be explained with an increased message volume that needs to be delivered reliably between an increased number of communication partners. Even more than 50 % of the overall time is spent on synchronization if at least five partitions are used. The bad performance of the topologies 8 LPs, 10 LPs a, and 10 LPs b seems reasonable as more than 75 % of the total time is spent on synchronization.

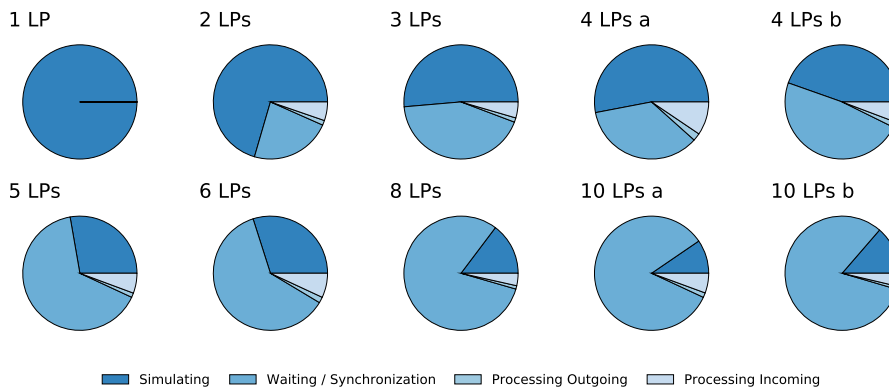


Figure 6.6 – Time consumption shares per topology.

Besides the possibility of achieving speed-ups, an interesting finding is that not only the number of involved instances appears to be important but also the specific partitioning strategy. When looking at the two different topologies involving four instances, the median run time of the 4 LPs a approach is below 30 seconds, while the median run time of the 4 LPs b topology is above 40 seconds. 4 LPs b is therefore performing significantly poorer, even if the same number of instances is used. Reasons for these differences may include the pure number of transferred vehicles and the distribution of the communication patterns over time (e.g., transmissions during each synchronization period vs. communication breaks) that are related to the partitioning.

6.2 Study 2: Multi-level Simulation

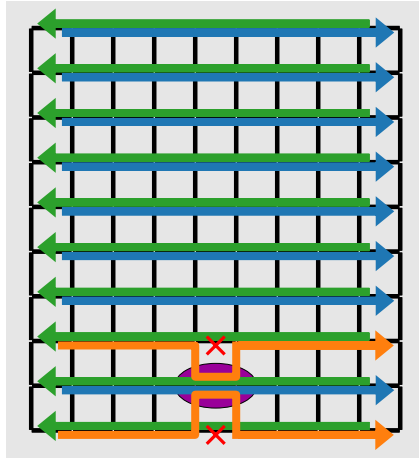


Figure 6.7 – Used traffic pattern. The purple ellipse shows the bottleneck.

27	65	103	141	179	217	255	293	331
25	62	100	138	176	214	252	290	328
22	58	96	134	172	210	248	286	324
19	54	92	130	168	206	244	282	320
16	50	88	126	164	202	240	278	316
13	46	84	122	160	198	236	274	312
10	42	80	118	156	194	232	270	308
7	38	76	114	152	190	228	266	304
4	34	72	110	148	186	224	262	300
1	30	68	106	144	182	220	258	296

Figure 6.8 – Topology for the multi-level run as defined in Appendix E.2.

When aspiring to increase the overall performance, the use of a coarser model is also a reasonable approach besides parallelization through distribution. However, performance benefits often come at cost of fidelity. A lack of detail is not limited to rounding errors. If the behavior of entities or systems are modeled in a simpler way, it is possible that certain situations cannot be represented at all. This can introduce errors, which eventually lead to major deviations when comparing final results of different models. Given that, it is obvious that using the fastest model is not always reasonable and using the highest detail is not always necessary. The question on if a modeling approach is suitable for a certain scenario is highly dependent on the use case (i.e., the input data, the questions to be answered, and the arising conditions within the simulation state). In this study, we want to showcase the potential of the combination of different modeling paradigms. As possible benefits depend completely on the use case and we are only considering exemplary questions in exemplary studies in this chapter, a specific investigation on which speed-up is possible in a certain case is not purposeful. In contrast, we demonstrate that the appropriateness of using certain paradigms is changing. Therefore, we simulate different variations of a common scenario with different paradigms and examine the respective errors and the computational efforts.

Assumption Let us assume that we are interested in a Manhattan grid road map that has a peculiar characteristic in the south. The traffic pattern consists of horizontal traffic flows (see Figure 6.7). By

declaring two roads as one-way roads (red crosses), we reroute the two affected flows (orange arrows) over the same neighbored road, which results in a bottleneck (purple ellipse). As a result, we are creating a potential congestion at the road that is now exposed to the vehicles of three flows in east-direction. We are interested in the impact of using a mesoscopic, a microscopic, and a combined model in order to assess the vehicle density on several roads.

Requirements We need a mesoscopic and a microscopic traffic model, the traffic input, road resources, and a mesoscopic-microscopic translator.

As we mainly expect an error of the mesoscopic model around the area of the bottleneck, we are using the topology that is depicted in Figure 6.8 for the multi-level simulation. The yellow area that is assumed to be critical is modeled microscopically, while the surroundings are modeled mesoscopically. We consider three variations of the described scenario with a low, medium, and high overall traffic load. The traffic volume of the medium traffic variant is 50 % higher than of the low traffic variant and the high traffic variant is 50 % higher than the medium one. In contrast to the number of vehicles, the qualitative development of the traffic pattern over the time and the used routes remain the same for all three variants. As the microscopic model is the most detailed approach, we will use its results as a baseline.

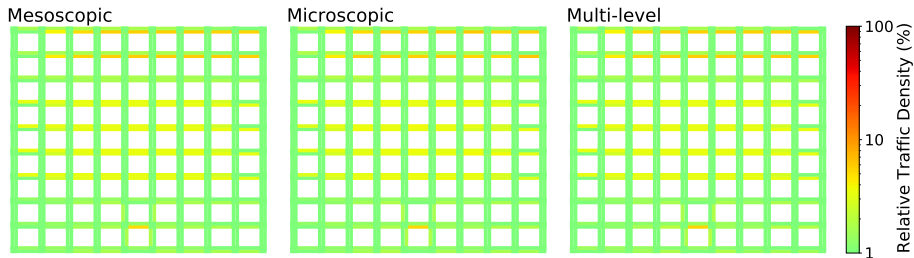


Figure 6.9 – Simulated densities with a low traffic volume at 21 min.

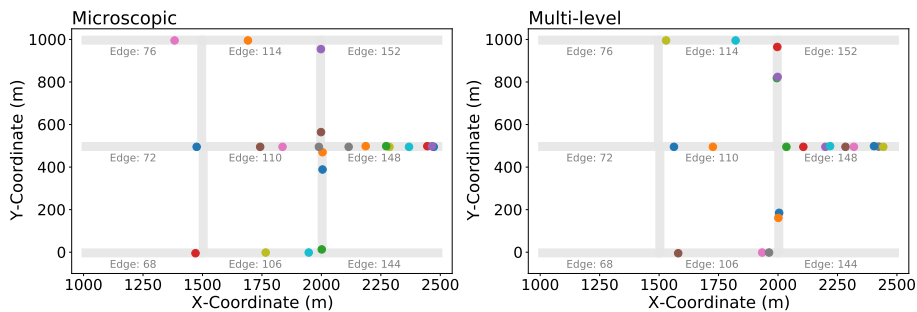


Figure 6.10 – Simulated distribution with a low traffic volume at 21 min.

For the low traffic variant, two things in particular can be learned from the results, when considering the situation at 21 minutes of simulation time (see Figure 6.9). First, the bottleneck is not leading to a congestion. Second, all three approaches lead to quite similar results. At the bottleneck there is a relative vehicle density⁶ of 5.05 % in the mesoscopic model, a density of 5.25 % in the microscopic model, and a density of 5.48 % in the combined model. On the preceding link there is a relative density of 1.69 % in the mesoscopic model, a density of 1.72 % in the microscopic model, and a density of 1.97 % in the combined model. When looking at the distribution of the single vehicles on the roads in the critical area (see Figure 6.10), the similarity between the baseline and the combined approach can also be seen. Edge 148 represents the bottleneck. There is no illustration for the mesoscopic model as positions of individual vehicles are explicitly not modeled in that paradigm.

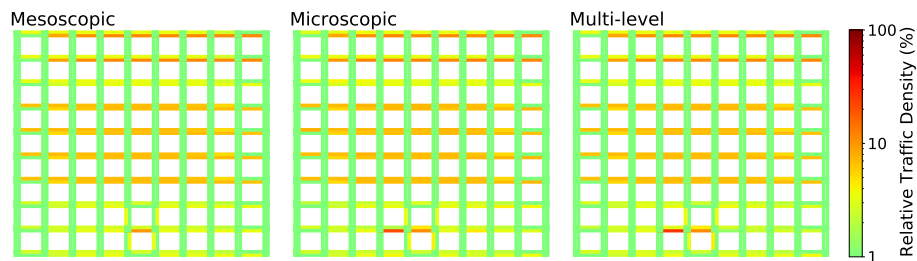


Figure 6.11 – Simulated densities with a medium traffic volume at 21 min.

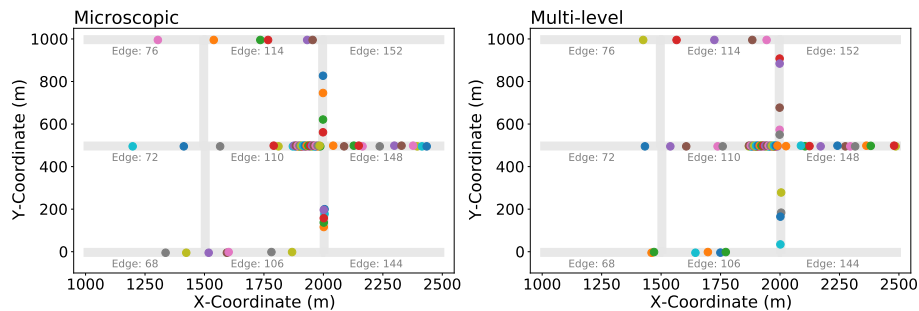


Figure 6.12 – Simulated distribution with a medium traffic volume at 21 min.

For the medium traffic variant, a deviation between the microscopic baseline and the mesoscopic outcome is recognizable in Figure 6.11. In contrast, the multi-level approach appears quite accurate. The latter two create a dense traffic situation at the link that is preceding the bottleneck, while there is a relaxed state on said link in the mesoscopic model. The reason for this difference might probably be that the fast mesoscopic model does not model individual junction behavior per default and therefore a traffic jam might not arise.

⁶Sum of estimated vehicle lengths and safety gaps divided by the link length.

On the bottleneck road, there is a relative density of 10.12 % in the mesoscopic model, a density of 10.97 % in the microscopic model, and a density of 11.13 % in the combined model. On the preceding link there is a density of 3.38 % in the mesoscopic model, a density of 20.93 % in the microscopic model, and a density of 26.65 % in the combined model. Similarly to the densities, the distribution patterns of vehicles in the microscopic and the multi-level approach are comparable (see Figure 6.12).

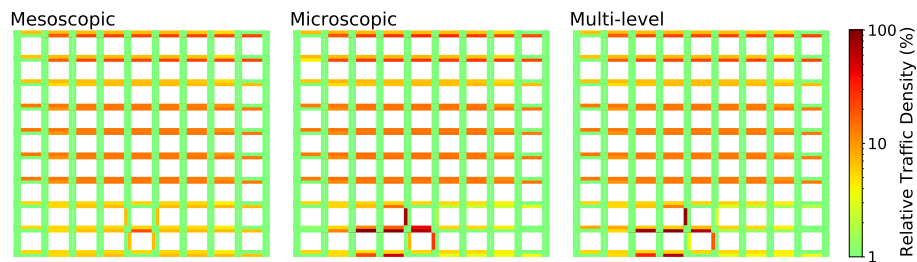


Figure 6.13 – Simulated densities with a high traffic volume at 21 min.

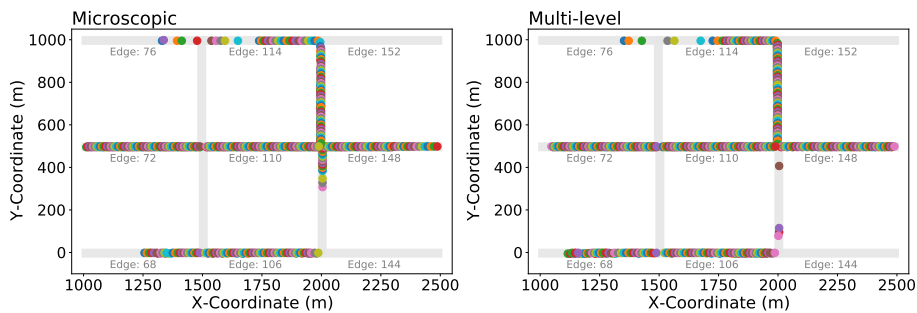


Figure 6.14 – Simulated distribution with a high traffic volume at 21 min.

The biggest differences can be observed in the high traffic scenario (see Figure 6.13). In the baseline results, there is an exhaustive congestion on two of the links preceding the bottleneck. The same situation can be found in the combined model. However, the mesoscopic model is not leading to any serious congestion. The bottleneck road has a density of 20.63 % in the mesoscopic model, a density of 67.16 % in the microscopic model, and a density of 66.13 % in the combined model. On the preceding link there is a density of 6.75 % in the mesoscopic model, a density of 93.38 % in the microscopic model, and a density of 91.50 % in the combined model. The congestion can also clearly be seen for the latter two approaches in Figure 6.14. A difference to the baseline can be observed, when looking at the path starting from edge 68 to edge 148 via edge 106. However, this deviation is not a result of a systematic difference, but rather expression of temporal dynamics. The volume that is missing on the vertical link can be found on edge 68. This might be

happening because of a dense burst of vehicles flowing from edge 144 to edge 68 at that time. Consequently, left turn maneuvers of vehicles on edge 106 are temporarily delayed.

As a first conclusion, the simulated outcome is comparable between the mesoscopic and the microscopic model for the used scenario, as long as there is no serious

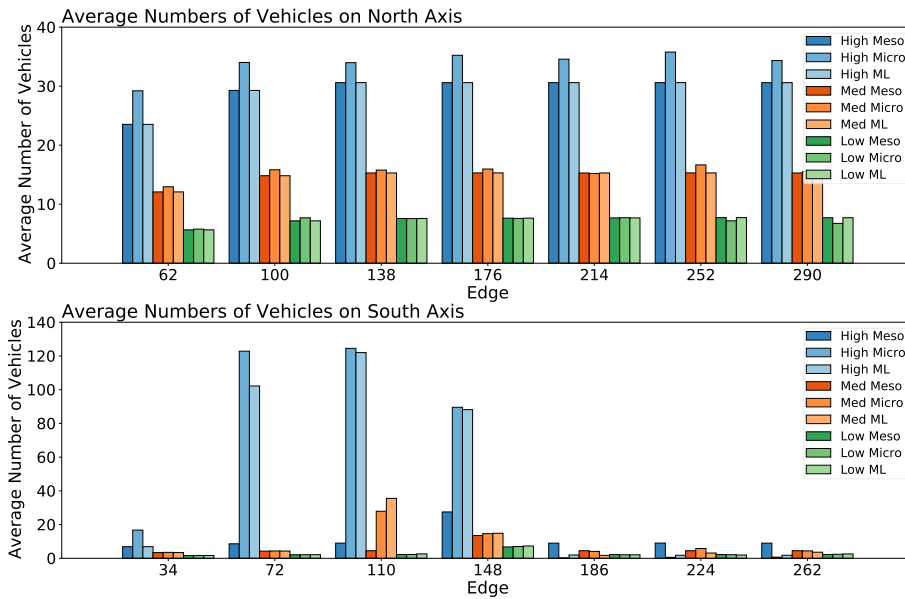


Figure 6.15 – Traffic load at 21 minutes.

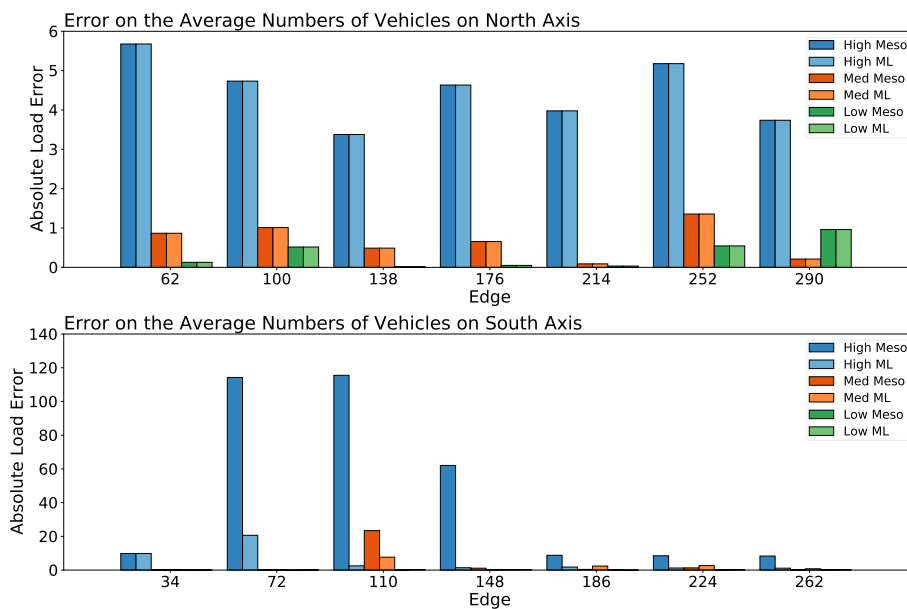


Figure 6.16 – Errors of traffic load at 21 minutes.

jam situation. Compared to the deviation of the mesoscopic model, the multi-level model works quite well. This is as expected, because the critical areas are modeled microscopically while only the surroundings are modeled mesoscopically.

The traffic load and error of two interesting traffic axes are illustrated in detail for the discussed simulation time in Figure 6.15 and Figure 6.16. Both axes are directed from west to east. The south axis consists of the roads that are forming the straight line through the bottleneck, and the north axis is the counterpart in the north. For orientation, the map in Figure 6.8 contains labels of the used edge names. The different densities are plotted over the simulation time for all paradigms and variations in Figure 6.17. For the illustration, we averaged the densities of the two described axes. Further illustrations on found absolute and relative errors compared to the simulated vehicle loads are given in Appendix E.3 for each scenario variation.

At this point, we only considered the error compared to the baseline. The question arises why an inaccurate approach should be considered at all, even if the error is relatively low. Such a decision seems more reasonable, when looking at the runtimes of the different models (see Figure 6.18). Of course, the pure mesoscopic model has the highest potential regarding speed-ups. For the high traffic volume scenario a speed-up factor of 125.9 compared to the baseline run was measured. For the medium and the low traffic variations of the scenarios the speed-up was still 47.2 and 22.2. However, this comes at the cost of relative errors of 28.4 %, 7.8 %, and 5.8 %, respectively. In case of the combined model, we face exemplary speed-ups of 2.3 for the high traffic load, 3.5 for the medium traffic load, and 2.6 for the low traffic load. The performance gains of the combined model come with an error of

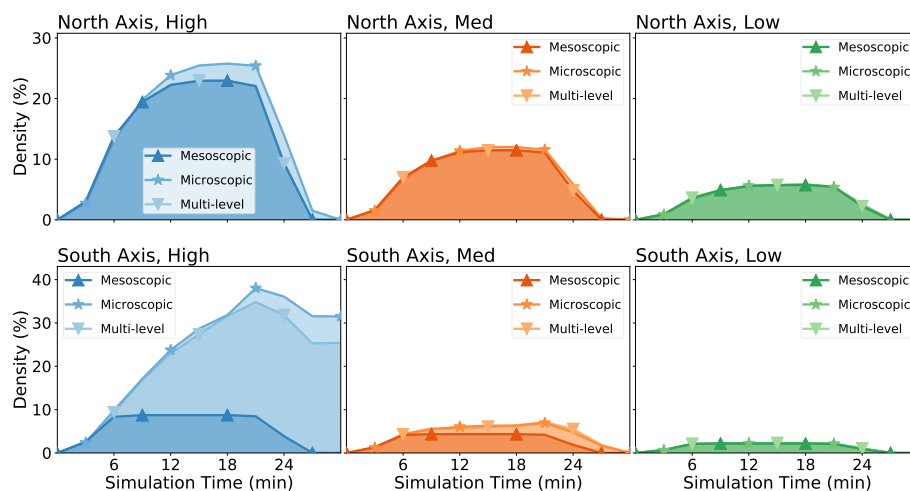


Figure 6.17 – Relative traffic densities on two main axes over the simulation time for the different traffic loads.

12.0 %, 6.2 %, and 5.9 %, respectively. The corresponding absolute computational times are depicted in Figure 6.19.

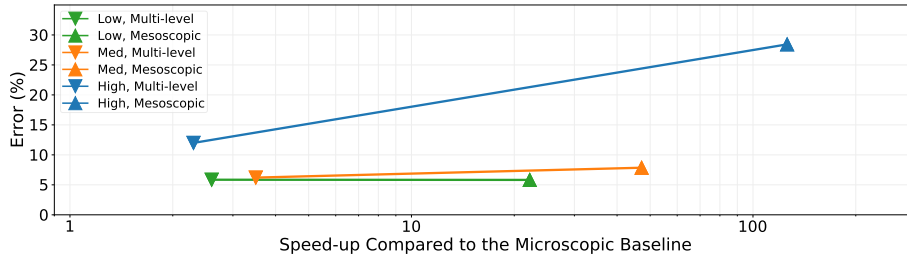


Figure 6.18 – Error of the whole road network vs. speed-up.

Given that, the typical dilemma of choosing an adequate model (or a combination) by estimating the trade-off between expected error and speed-up is outlined. If we say, for instance, that an error of 15 % is acceptable and performance is top priority, the pure mesoscopic model would be the best fit for the scenarios with a low and a medium traffic load. However, in the case of the scenario with a high traffic load the multi-level approach would be the better fit as the error is below 15 % and we are still able to accelerate the computational effort by a factor of 2.3. The poorer speed-up performance of the combined approach for the high traffic scenario compared to the medium traffic scenario might be attributed to the excessive exchange of NDMs due to the high traffic volume. In summary, we demonstrated that a speed-up at cost of accuracy is possible when using coarser models and that the combination of models of different detail level can also be beneficial when aiming for performance gains.

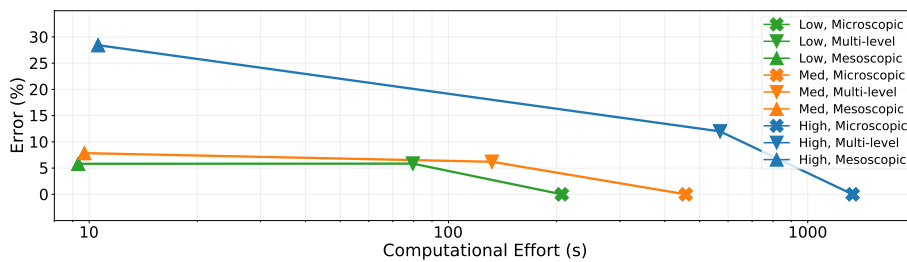


Figure 6.19 – Error of the whole road network vs. computational time.

6.3 Study 3: Data Enrichment

One of the most crucial benefits of multi-level models is the ability to generate knowledge on detailed levels, while the input data is provided on coarser levels. We want to showcase the ability to use multi-level traffic simulations in order to enrich existing macroscopic data sets in this study. In addition, we use this scenario as a proof-of-concept for demonstrating that all four developed layers are interoperable.

Assumption Let us assume that we developed an AEB system. Before we conduct real tests, we want to evaluate the AEB system virtually by feeding in virtual sensor data. Therefore, we want to gather sensor data in detailed three-dimensional virtual environments. At the same time, we want to be able to define the traffic pattern of a city in a simple way (i.e., in a macroscopic format). Hence, we are aiming for a multi-level approach, where we define the traffic patterns in a macroscopic regime and generate sensor data in a submicroscopic regime that is embedded in the macroscopic scenario.

Requirements We need a macroscopic traffic model, a mesoscopic traffic model, a microscopic traffic model, a submicroscopic traffic model, the macroscopic traffic input, road resources, and translators between all different layers.

The quality of the resulting synthetic sensor data of the environment highly depends on the quality of the model. If there is, for example, no information about road signs, pavements, or buildings, it is obviously not possible to find such objects in simulated lidar scans. As a consequence, the generation of detailed road maps and shape files is very time-consuming. While most of the static scenario resources for macroscopic, mesoscopic, and microscopic models can be generated automatically from available sources such as the OpenStreetMap project [165], typically, a lot of manual work is required to create resources for submicroscopic models.

Therefore, we use an existing three-dimensional model that is provided by the CARLA project. It represents a part of an artificial town and is called *Town01*. The original road topology is shown with a red background in Figure 6.20. As the map size is rather small, we added incoming and outgoing axes at the north and at the south borders of the original map, which is also depicted in the figure. In order to showcase the translation between all traffic layers, we will use an onion like topology, where the most northern and southern nodes are in the responsibility of a macroscopic BB that is neighbored to a mesoscopic BB. A microscopic region is then surrounding the central area, which falls in a submicroscopic regime (see Figure 6.20 for a graphical representation of the topology). We will use a traffic

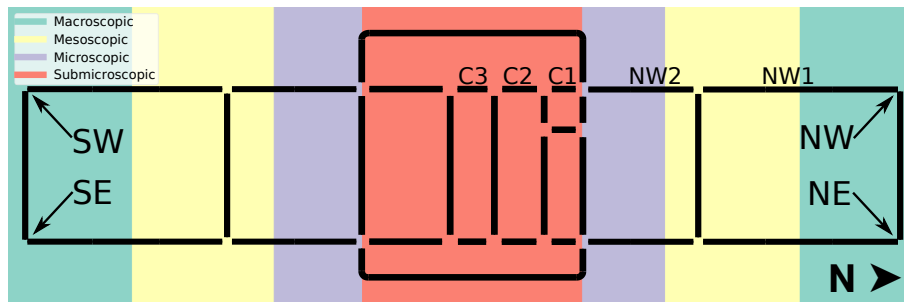


Figure 6.20 – The extended road network of Town01 with the used responsibility topology.

pattern with flows between north and south, so that the traffic has to pass all different modeling paradigms. Thus, all modeled translation processes are actively used. The used exemplary traffic pattern is shown as an origin-destination matrix in Table 6.2. Apart from the conducted multi-level simulation, we also simulate the same scenario exclusively by a macroscopic component to generate a baseline.

```
provision.study3.traffic.macro.road.NW1 (6.1)
```

```
provision.study3.traffic.meso.link.NW1.vehicles (6.2)
```

```
provision.study3.traffic.meso.link.NW2.vehicles (6.3)
```

```
provision.study3.traffic.micro.edge.NW2.vehicles (6.4)
```

```
provision.study3.traffic.micro.edge.C1.vehicles (6.5)
```

```
provision.study3.traffic.submicro.lane.C11.vehicles (6.6)
```

Data is, for instance, published by the macroscopic BB to Topic 6.1, where it is consumed by the macroscopic-mesoscopic translator, converted, and again published to Topic 6.2, where it is consumed by the mesoscopic BB. If vehicles arrive on NW2, the mesoscopic BB is publishing the NDMs on Topic 6.3, where it is consumed by the mesoscopic-microscopic translator, converted, and published to Topic 6.4. From there it is consumed by the microscopic BB. As soon as vehicles arrive on C1, the microscopic BB is publishing according NDMs on Topic 6.5, where data is consumed by the microscopic-submicroscopic translator, converted, and published to Topic 6.6. The data is finally consumed and integrated by the central submicroscopic BB. Similarly, the traffic is propagated further south and also back north in the opposite direction.

We simulate a period of 30 minutes. First, we compare the resulting traffic on link C3 between the multi-level simulation and the baseline simulation. In total, a minimal deviation regarding the flow can be observed. If the observation window is small enough, we can also see the impact of the PAP that is powering the macroscopic-

	NW	SW	SE	NE
NW	0	900	0	0
SW	600	0	0	300
SE	0	0	0	0
NE	0	0	0	0

Table 6.2 – Origin-destination matrix. Traffic demand per hour.

mesoscopic translator. For a window of 1 minute the average error is around 7.7 %, for a window of 5 minutes we see an average error of 2.4 %, for 10 minutes 1.5 %, and for a window of 30 minutes the error is below 1 %. The fluctuations for different observation windows are depicted in Figure 6.21. For the 10 minute observation window and especially for the 30 minute observation window, the near match can clearly be seen, which allows us to approve the validity of the translators. Regarding observed average speeds on road C3, we see values close around the allowed 30 km/h on said road.

The fluctuations regarding the traffic flow, which are especially visible when looking at the 1 minute window, are not there by accident. On the contrary, we intended to create such behavior and therefore used the PAP in the translator as uniformly distributed spawn events (i.e., interarrival times) would not be very realistic. The underlying sampled spawn events are depicted in Figure 6.22. One can clearly see regions of different densities (i.e., different numbers of spawned vehicles), while the macroscopic traffic flow values remained static.

Besides considerations regarding the validity, we can now benefit from a much more detailed world that is populated according to the macroscopic input. The exemplary goal was to generate sensor data that can be further used. The following

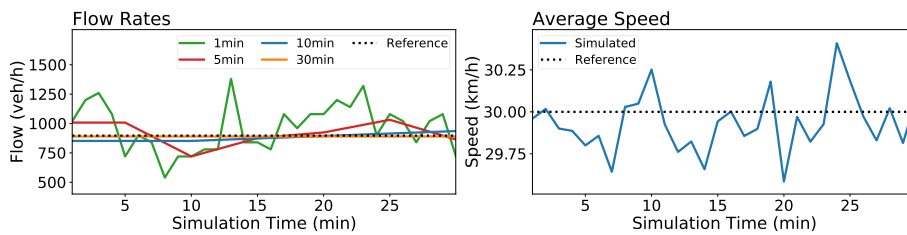


Figure 6.21 – Traffic flows and speeds on road C3.



Figure 6.22 – Each black line represents a discrete spawn event in the submicroscopic regime.



Figure 6.23 – Snapshots of the evaluation scenario taken in the three-dimensional world of CARLA.

examples are produced by using sensors that are available out of the box by CARLA and using only the macroscopic traffic input. In total, we illustrate six different views on the same scene. In Figure 6.23, the RGB camera sensor is used to create an impression of the scene to let the researcher understand the current context. It is done by using a view into the distance and a bird's eye view on an ego vehicle. Of course, the camera could also be positioned behind the windshield to act as a typical in-vehicle camera sensor. The rendered scene also contains lightning conditions (e.g., there are shadows on the road). Such level of detail could be useful when generated pictures are used to train neural networks that should recognize objects purely based on camera input.

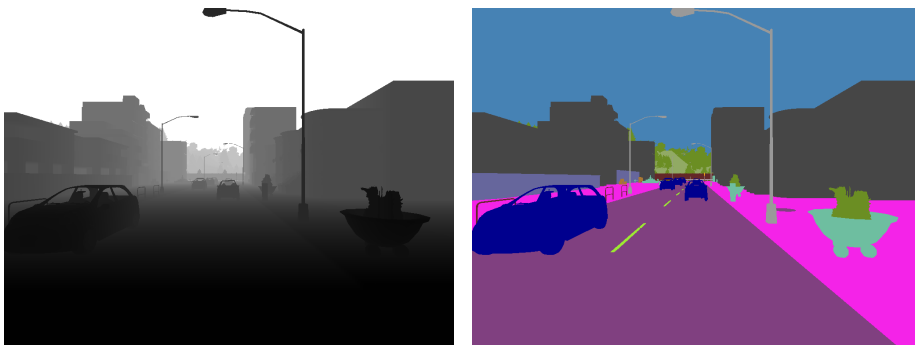


Figure 6.24 – Exemplary sensors: distance estimation and semantic segmentation.

However, it might also be desirable to not only work on raw camera data, but on data from other sensors, or already existing preprocessed camera data. Two examples are shown in Figure 6.24. On the left side, there is a depth image that represents the distance between the ego vehicle and external objects. For the picture on the right side, we use the semantic segmentation camera of CARLA that maps the type of an object to a certain color.

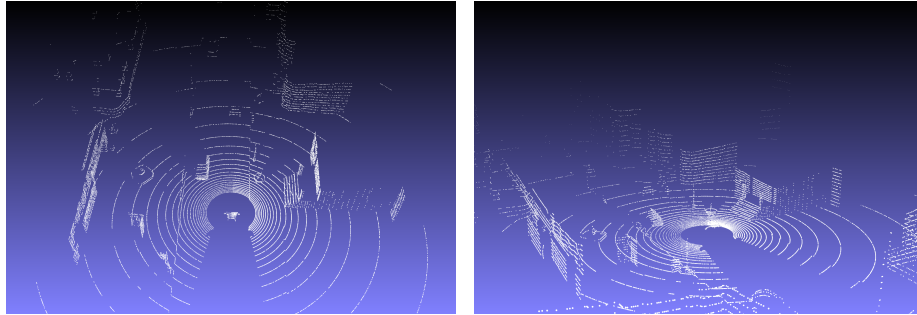


Figure 6.25 – Lidar scan point clouds from above and from the side.

As a last example, we assume that the vehicle is additionally equipped with a lidar sensor and creates lidar scans of the surroundings. Exemplary point clouds of the given scene are depicted in Figure 6.25. For instance, the stairs that are also shown in Figure 6.23 are clearly visible. Many functions from the field of autonomous driving and other recent assistance systems (e.g., this study’s exemplary AEB) are relying on lidar input. All of them can benefit from the opportunity to create an unlimited amount of input data and use it for tests, evaluations, and enhancements of the application under test.

We demonstrated the possibility to enrich existing data sets with additional, potentially far more detailed, information pieces. In addition, we showed our approaches’ capabilities regarding tool and platform interoperability: PTV Visum was executed under Windows, while the other components were run in Linux Docker containers. Also, three different programming languages are involved. The translators, the VisumWrapper, and the MatsimWrapper are implemented in Java. The SumoWrapper is implemented in C++ and the CarlaWrapper is implemented in Python.

6.4 Study 4: Integration of External Data

In the fourth case study, we want to demonstrate two additional features of the developed approach: integrating real-world data and creating a new virtual translator by pipelining already existing translators.

Assumption Let us assume that we want to consider the traffic impact of closing one or two lanes due to construction work on a road stretch of the motorway A9 in Germany. In order to assess this question appropriately, we might have decided that the fidelity of a microscopic model is required as well as a realistic traffic pattern. In addition, access to real-world traffic data of the region of interest is available. However, the accessible data is generated by physical road detectors and is already aggregated. It is therefore in a macroscopic format (i.e., traffic volumes and speeds in a minute resolution).

Requirements We need a microscopic traffic model, the macroscopic traffic input, and a possibility to use the macroscopic data in the microscopic model.

We create a digital twin of a section of the A9 motorway in the south of Nuremberg (see Figure 6.26) and run what-if experiments with it. In order to bridge the information gap between the macroscopic and the microscopic world, we use translators as in the use case before. In contrast to study 3, we do not necessarily have to import the input data in a macroscopic BB such as Visum for publishing it to the data pool. As the whole coupling process is focused on data, we will use the external data source directly as a macroscopic BB that provides traffic information input on corresponding topics.

More precisely, we received a dump of historical detector data from a whole weekday in April 2018 from the German *Die Autobahn GmbH des Bundes*. The data was gathered on the A9 in northbound direction at a detector near Allersberg and

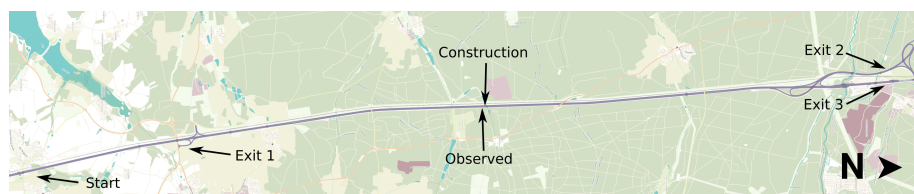


Figure 6.26 – The road section to be considered is located in the south of Nuremberg, Germany and stretches from (49.2441, 11.2176) to (49.3634, 11.2024). The traffic is ingested near the *start* label. The map is based on contents from ©OpenStreetMap [165].

contains i.a. traffic volumes that are already aggregated to one minute slices. As we use historic data, we implemented a simple data provider using the `JavaBaseWrapper` library. The provider acts as a normal BB and takes part in the synchronization process. It is therefore capable of publishing the input data at the corresponding points in simulation time. A live-data source could be used similarly. In this case, an external component could directly publish the data to the scenario topics in the data pool. The only drawback of such a live version of a digital twin would be that the synchronization mechanism obviously cannot influence the wall-clock time (i.e., the real-world detector device will continuously publish data at a fixed period). Involved BBs would need to make sure to run at least with a real-time factor of one, if a live representation of the real-world is desired.

Regarding the translation between the different layers, there are multiple options. We prefer not to include an intermediate mesoscopic model as it was done in study 3, because we are only interested in the outcome of the microscopic simulation model. Consequently, there are two options to connect the macroscopic and the microscopic regime. We could either model a new translator that is capable of translating directly between macroscopic and microscopic NDMs. Or we could map a chain between the two existing macroscopic-mesoscopic and mesoscopic-microscopic translators, which results in a new virtual translator that is also capable of transforming macroscopic NDMs to microscopic NDMs. In the following, we will use the latter strategy and demonstrate the strengths of the developed approach regarding flexibility and reusability.

```
provision.study4.traffic.macro.road.start (6.7)
```

```
provision.study4.traffic.meso.link.start (6.8)
```

```
provision.study4.traffic.micro.edge.start (6.9)
```

```
provision.study4.traffic.micro.edge.observed (6.10)
```

As the macroscopic BB acts as a pure data source, we can observe a single data flow in one direction. The BB provides macroscopic NDMs for every minute of the scenario for the most southern road stretch on Topic 6.7. The macroscopic-mesoscopic translator subscribes to said topic and uses the macroscopic information to sample mesoscopic NDMs at exponentially distributed spawn times, as described in Section 5.2.3, which are published to Topic 6.8. The mesoscopic-microscopic translator subscribes to that topic and uses its disaggregation logic to sample and publish microscopic NDMs to Topic 6.9. Finally, the microscopic BB subscribes to Topic 6.9 and integrates received microscopic NDMs into its simulation state. Because we are interested in the traffic condition on the road section directly before the potential construction site, an observer for that edge is specified in the definition

of the microscopic BB. All described parameters can be found in the used scenario definition in Appendix E.4.

The simulated traffic pattern of the baseline scenario appears as expected. The resulting relative vehicle densities per lane are depicted in Figure 6.27 as stackplots (i.e., the total relative road density is the sum of all colored parts). It gives not only insights about the traffic pattern but also about potential congestions. While there is minimal traffic between 2 am and 4 am there is a peak in the afternoon. From 12 am to 5 am, the traffic condition allows using a single lane without any limitations for the vehicles. For the “no construction work”-scenario, the density of the observed edge stays below 20 % during the whole 24 hours of simulation time. If the right lane is closed, two peaks in the afternoon slightly exceed the density of 20 %. However, the impact on the overall traffic condition is limited. On the contrary, if two lanes are closed, there are multiple peaks that almost reach the level of 40 %. We can infer that the traffic condition is highly impacted by the decision of closing two lanes.

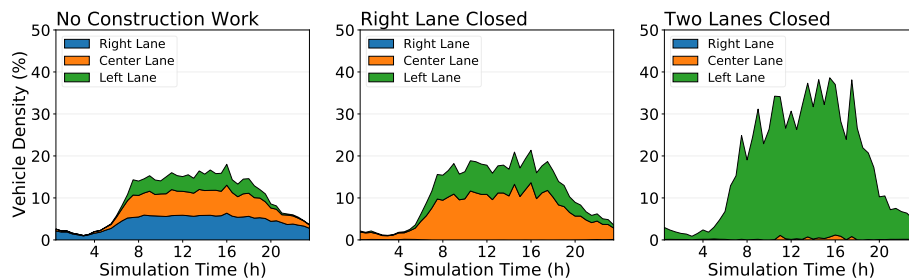


Figure 6.27 – Relative vehicle densities of the three different scenarios.

The impact can also be identified when looking at the average speeds. Over the whole 24 hour period, the speeds differ significantly for the different scenarios. If all lanes are open, the average speed is 113 km/h. If the right lane is closed, the average speed drops to 105 km/h. If two lanes are closed, the speed drops further to 80 km/h. The distribution over the whole 24 hour period is depicted in Figure 6.28. Between 12 am and 4 am, there is so little traffic flow that there is no impact of the closing decisions on the average speeds. Similarly, only one lane is used by all vehicles when looking at the vehicle densities. On the contrary, the average speed partly drops below 50 km/h between 1 pm and 6 pm for the scenario where both lanes are closed.

As an exemplary conclusion, we could state that the closing of one lane is feasible. Carrying out construction works on two lanes at the same time is no reasonable decision and would regularly lead to strong congestions. However, if works on two lanes at the same time are necessary, a possible time window could be 8 pm to 6

am. We demonstrated the integration of a real-world data set and the reusability capabilities of our approach.

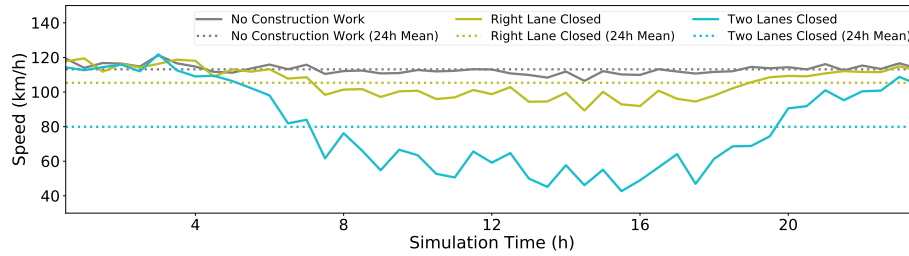


Figure 6.28 – Average speeds of the three different scenarios.

6.5 Study 5: Cross-Domain Simulation

As a final study, we demonstrate the abilities to combine BBs from different domains in order to model cross-domain problems. In addition, we will show the advantages of being able to directly reuse simulation data from the data pool. A domain that is closely related to the traffic domain is the communication domain. Vehicular communication is an innovation driver and will enable a lot of new business models such as smart parking and enhance safety, for instance, by providing blind spot warnings [10]. Naturally, if a certain technology is relying on communication, the performance of the communication medium has a great impact on the overall quality. Thus, it is of interest to assess the quality of a communication system. Such investigations can be carried out as real-world experiments. However, if a whole system of vehicles is under observation, the effort increases immensely and conducting real experiments is often no longer feasible. Instead, simulation can be used - even if large-scale scenarios are intended. One of the most interesting aspects of vehicular communication is the underlying dynamic network topology that originates from the movement of the vehicles. Updating the positions of the communication nodes is therefore crucial and often an adequate mobility model is required rather than simple linear movement models. That means nodes representing discrete vehicles should not float randomly in free space, but move in a realistic way according to a road network and complying to mobility patterns. By doing so, the characteristics of communication can also be examined for typical corner cases such as traffic jams, sparse traffic, or dense traffic. The evaluation of the quality of a vehicular communication system is not straight-forward as different use-cases have different requirements. Comfort applications, for instance, might require a high bandwidth as a lot of data is exchanged (e.g., video streaming), while latency is not that important. On the contrary, safety applications might need to exchange low volumes of data, while latency or reliable delivery is top priority. In a cooperative safety application, each participating vehicle might be broadcasting status updates (e.g., its position) and might be gathering the same messages from surrounding vehicles. The received information can then be used to build a virtual representation of neighboring road participants and display warnings to the driver.

Assumption Let us assume that we are interested in the time the radio channel is busy in order to assess whether such a cooperative safety application can be realized in a robust way. Further constraints are that the used communication technology will be IEEE 802.11p, and that we are using a realistic traffic pattern on a stretch of the south-north axis of the motorway A9 in Germany that is depicted in Figure 6.29.

Requirements We need a traffic pattern as input data, a microscopic traffic model, and an 802.11p model.

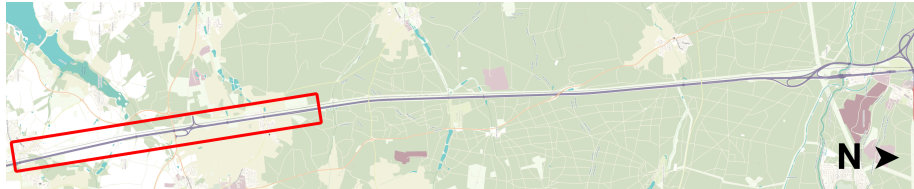


Figure 6.29 – We use a part of the map from study 4. The area of interest is marked by the red rectangle. The map is based on contents from ©OpenStreetMap [165].

As we constructed a similar example as for the previous study, we can reuse the generated data from study 4. We do so by creating a scenario definition with similar global parameters, a BB from the 802.11p layer in the communication domain, and the projector that was developed in Section 5.4. The projector will subscribe to the historic vehicle data from study 4 and trigger API-calls to the 802.11p communication layer in order to create, delete, or move nodes that represent modeled vehicles. Because we want to equip all vehicles with a radio and have a common beaconing rate of 1 Hz, we are adding these parameters in the parameter section of the communication BB. Finally, we are interested in the average channel busy time that is sensed by each vehicle, lost packets, and required backoff slots. Thus, we specify desired results. The resulting scenario description can be found in Appendix E.5.

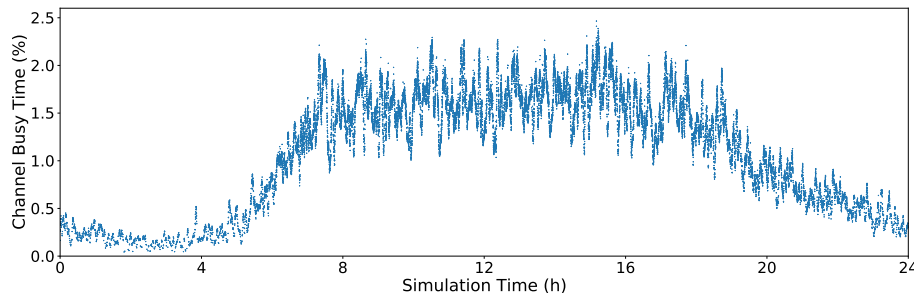


Figure 6.30 – Busy time of the channel as perceived by each vehicle.

Due to the packet-level simulation of the wireless communication, the overall simulation took almost 2.5 h (8898 s), which is far longer than the time needed in study 4. Over the whole 24 hours, we are facing an average channel busy time of 1.40 % and a maximum value of 2.47 %. The distribution over the simulated day is depicted in Figure 6.30. If compared to the traffic pattern that was already depicted in study 4 (see Figure 6.27), one can clearly see the relation between the two curves. As the traffic volume grows, the number of required backoff slots is also

increasing. This can especially be seen during the hours with a high traffic volume (see Figure 6.31). As a result of the study, the simulated busy times might be seen as low enough to establish such a system in practice.

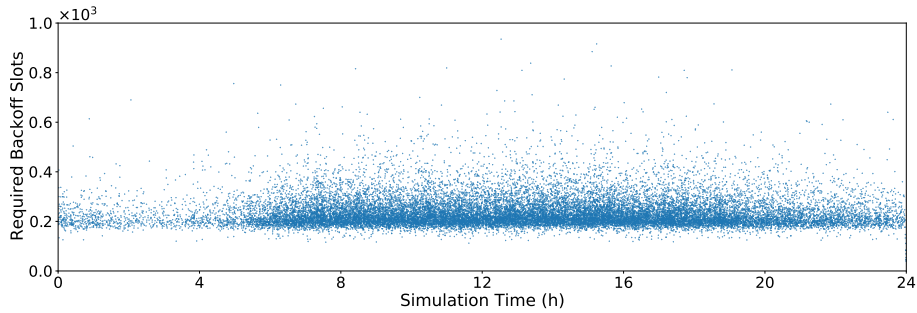


Figure 6.31 – Diced backoff slots per vehicle.

However, despite the relaxed busy times, one could also argue that the communication system is not good enough. We have a look at packets that were theoretically received with a sufficient signal-to-noise ratio, but were lost nevertheless, because an own transmission was ongoing. These lost packets are plotted in Figure 6.32. We see that a few vehicles lose many messages. Consequently, we infer that some information of surrounding vehicles was not delivered reliably, which could increase the risk of an accident.

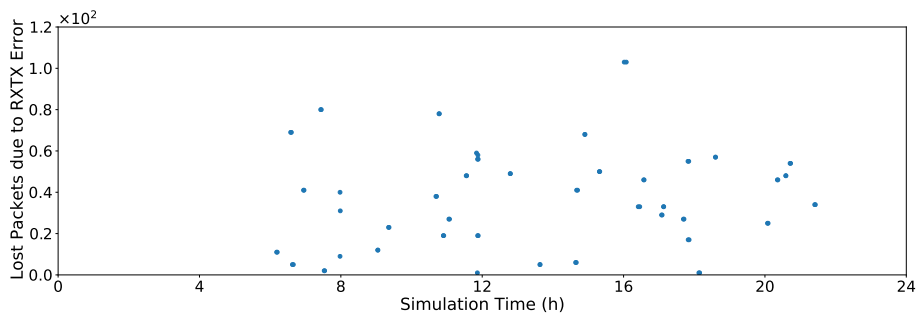


Figure 6.32 – Lost packets due to ongoing transmission on signal reception.

With this fifth study, we demonstrated how easy it is to realize cross-domain simulations with our approach. In addition, we showed the potential that arises from simply reusing already existing simulation data. However, if the reuse of existing simulation data should not be appropriate, it is of course also possible to simulate the vehicle movements in parallel. For such cases, one could simply add the SUMO BB description from the scenario definition file of study 4 to the current scenario definition and let the projector consume this scenario's vehicle data.

Chapter 7

Conclusion and Future Directions

In this chapter, we summarize the work that was presented within this thesis and draw a conclusion. Finally, we talk about limitations of the developed approach and give an outlook on possible future directions and research topics.

7.1 Summary and Conclusion

The purpose of this thesis is to provide a sound distributed simulation methodology that satisfies the described requirements as well as a framework that implements the developed approach. By doing so, the gaps between existing solutions and identified requirements are bridged. Furthermore, the utility of the concept is demonstrated by applying it to the field of road traffic simulation.

The intention for this was to tackle five urging simulation aspects. The growing linkage of domains increasingly brings up problems that span across domains. Modeling and simulation of such problems might require the connection of multiple components, which can also be utilized to overcome hardware limitations of single nodes. Input data may have to be retrieved or generated, converted, and ingested into a coupled simulation ecosystem. Finally, diversity of stakeholders has to be taken into account as well as supporting validation of coupled models and reproducibility of simulation runs.

In order to do so, we described theoretical and practical foundations, identified current solution approaches and existing requirements. Based on the gained insights, we developed a data-centric methodology that is capable of solving the drawn issues. The main idea was to move away from considering specific tools, but rather put data models in the focus of a co-simulation protocol. Following our approach, there will be no dedicated and custom-tailored data model for each tool or application. Contrarily, we aim for having stereotypical data models that cover whole groups of tools, for instance all components of a certain modeling paradigm within a domain.

We called such a data model layer and introduced several tags that can be added to the elements of a layer data model. Most importantly, the layer contains the information on what data structure (*NDM*) is communicated in which occasions. Therefore, the layer abstraction provides the foundation of the co-simulation protocol and defines the component coupling logic from a pure data perspective.

Each component that is integrated in the simulation system is assigned to a single layer and infers its interface exclusively from the layer definition. Consequently, there is a high abstraction level between integrated (simulation) components and the co-simulation protocol. Regarding the communication between components, we chose an approach where we abandon all transient communication, but rather persistently store each exchanged message. This happens in order to strengthen reproducibility and traceability of simulation runs and allows providing generated data for further analysis. In the course of this, we picked a topic-based communication system as a foundation that allows a direct mapping of the used data models on topics. This enables a structured access to the simulation data (even for external components), while we can implement a flexible communication schema that operates without global knowledge of other involved participants. Consequently, components are coupled in a loose and very extendible way. Regarding correctness, we designed a synchronization mechanism, which assures that messages are delivered on time and in a deterministic order, while communicating via multiple topics without a global ordering.

From a practical point of view, we implemented the methodology in client libraries for C++ (*CppBaseWrapper*), Java (*JavaBaseWrapper*), and Python (*PythonBaseWrapper*). The simulation service functionality was implemented in a Java component (*SimulationController*). In order to support heterogeneous users, a graphical user interface is provided via a web application (*Frontend*) allowing to manage resources and definitions, design scenarios via drag and drop functionalities and execute them, and access historic runs in a user-friendly way. Afterwards, we evaluated the approach in a domain-agnostic way demonstrating the developed synchronization mechanism and showing its performance. We also illustrated the problem that arises if we would not provide an appropriate mechanism for assuring deterministic information exchange.

In a second part, we then demonstrated the practicability of the approach by applying it to the traffic domain. We created layer definitions for the macroscopic, the mesoscopic, the microscopic, and the submicroscopic traffic modeling paradigms. We chose a representative simulation package for each layer and implemented integrating wrappers for every tool. Therefore, PTV Visum (*VisumWrapper*), MATSim (*Mat-simWrapper*), SUMO (*SumoWrapper*), and CARLA (*CarlaWrapper*) were integrated in our system. To enable multi-level simulations and bridge data gaps, we modeled and implemented translators between each neighboring layer (macroscopic ↔ meso-

scopic, mesoscopic \leftrightarrow microscopic, and microscopic \leftrightarrow submicroscopic). Also, we defined a related domain (communication) with one exemplary layer (80211p) in order to demonstrate the cross-domain capabilities of the approach. Thus, we implemented a wrapper for OMNeT++ and the Veins IEEE 802.11p model for simulating wireless communication (*OmnetWrapper*) and a corresponding projector (traffic.micro \rightarrow communication.80211p). The implementations are accessible in a public repository [94]. Several exemplary case studies concluded the work by illustrating different capabilities of the developed approach.

- First, we showed that performance gains are possible when using several instances of the same tool to distribute and parallelize a simulation run.
- In second study, we illustrated the relation between error and speed-up when combining models of varying level of detail.
- Third, we demonstrated the coupling of all four relevant traffic modeling paradigms in a single scenario. It was used to generate sensor data from a submicroscopic environment, while only providing input at a macroscopic level.
- In the fourth study, we showcased the benefits of the abstraction by stereotypical data models. We used a real-world traffic data stream and ingested it via a macroscopic component, translated it, and generated a microscopic digital twin of a German motorway stretch.
- Lastly, we reused simulation data by ingesting vehicle positions into a component from the communication domain, and evaluated a vehicular communication scenario on the motorway, which resulted in a cross-domain simulation.

In conclusion, we fulfilled the specified requirements that were motivated by five simulation challenges given in Chapter 1. The developed ideas help to create reusable co-simulation scenarios by following a data-centric approach. The key features of the developed methodology and the simulation framework comprise the support of distributed, multi-level, and cross-domain simulations using a data-centric protocol. Furthermore, reproducibility of results is ensured by storing all exchanged messages. A black-box approach that is powered by the stereotypical data models enables interoperability, reusability, and the individual validation of components and connectors. The concept allows the plug-and-play design of sophisticated simulation scenarios, which is further facilitated by providing a simulation service with a user-friendly frontend.

7.2 Limitations and Future Directions

Due to its nature, this thesis covers a broad area and is by no means presenting a closed solution to all issues related to future simulations. On the contrary, there are potential extension points in various directions, from which some are pointed out in the following.

The flexible coupling protocol is mainly driven by the concealing of a component's peculiarities behind the stereotypical layer interface. At the same time, reducing the externally available functionality of a component by purpose might be a limitation for the user. It would be interesting to investigate the acceptance of having building blocks with deliberately limited API functions, if the user benefits from the features of the developed system in return (e.g., plug-and-play design of new scenario topologies). Similarly, regarding the extension to additional domains, another open question is if the constraint of having one single NDM per layer is always suitable. Therefore, a study on the usefulness of the presented concept in different fields of application would be worthwhile.

As a result, an extensive catalog with domain and layer definitions could be provided, shared, and collaboratively enhanced. The same applies to developed tool wrappers, translators, and projectors. Another interesting step in this direction could be the integration of existing reference data models, especially ontologies. Currently, the domain and layer definitions are based on individual domain expert knowledge and are represented in a custom JSON structure. Following the ideas of the semantic web [24], there are already standardized definitions of structures of concepts found in the world called *schemas*. Incorporating these schemas in our system could bring the benefit of having access to a big amount of potential layer definitions that were already designed by other domain experts. Moreover, as it is the core idea of the semantic web, there might be links between different schemas, which could be used to infer connectors automatically. As schemas are actively used in other applications, for instance when providing public data with a semantic model, one could directly integrate these data sets in simulations.

From an implementation perspective, one could add code generation features to automatically create skeleton wrappers in all main programming languages directly based on a layer definition. In addition, the wrappers use a local deterministic pseudo number generator. As a consequence, the resimulation of a scenario will lead to the same results if the same seed is used and the topology is not changed. If repeatability would be also desired for repetitions with varying topologies, a scenario-wide pseudo random number generator provider would be a possible extension. However, this would add a lot overhead and will only work for suitable scenarios (e.g., distribution of a scenario over several instances of the same component).

At the moment, the design of a scenario and especially the partitioning is done by hand. By providing the graphical user interface, this is easily possible. However, it would be interesting to add a functionality that generates the simulation topology (e.g., components and their responsibility sets) automatically based on predefined indicators (e.g., average density on road X should be simulated, while performance is important). Such functionality could also be extended in order to adapt the simulation topology dynamically during a simulation run. As we have placed special emphasis on an expandable system, we are curious to see whether these or new ideas will be contributed in the future and what research our approach may enable.

Appendix A

General Definitions

A.1 Domain Definition

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "title": "Domain",
4   "description": "Domain Sefinition",
5   "type": "object",
6   "definitions": {
7     "simplePrimitive": {
8       "type": "object",
9       "required": [
10        "name",
11        "type"
12      ],
13      "properties": {
14        "name": {
15          "description": "The attribute's name",
16          "type": "string"
17        },
18        "type": {
19          "description": "The attribute's type, e.g. 'double'. ↘
20             This has nothing to do with json types",
21          "type": "string"
22        },
23        "unit": {
24          "description": "The attribute's unit, e.g., ms",
25          "type": "string"
26        },
27        "description": {
28          "description": "A further description",
29          "type": "string"
30        }
31      }
32    }
33  }
```

```
31     },
32     "reference": {
33         "type": "object",
34         "required": [
35             "name",
36             "key"
37         ],
38         "properties": {
39             "name": {
40                 "description": "The compound's name",
41                 "type": "string"
42             },
43             "key": {
44                 "description": "The compound's key",
45                 "$ref": "#/definitions/simplePrimitive"
46             },
47             "additional": {
48                 "description": "Additional elements",
49                 "type": "array",
50                 "items": {
51                     "anyOf": [
52                         { "type": "array" },
53                         {
54                             "$ref": "#/definitions/simplePrimitive"
55                         }
56                     ]
57                 }
58             }
59         }
60     },
61 },
62 "required": [
63     "name",
64     "version",
65     "reference"
66 ],
67 "properties": {
68     "name": {
69         "description": "The domain's name",
70         "type": "string"
71     },
72     "version": {
73         "description": "Version of the domain definition",
74         "type": "integer"
75     },
76     "reference": {
77         "description": "The domain reference",
78         "$ref": "#/definitions/reference"
79     },
80     "description": {
```

```
81     "description": "A description",
82     "type": "string"
83   }
84 }
85 }
```

Listing A.1 – Domain definition.

A.2 Layer Definition

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "title": "Layer",
4   "description": "Layer Definition",
5   "type": "object",
6   "definitions": {
7     "compound": {
8       "type": "object",
9       "required": [
10        "name",
11        "key"
12      ],
13      "properties": {
14        "name": {
15          "description": "The compound's name",
16          "type": "string"
17        },
18        "key": {
19          "description": "The compound's key",
20          "$ref": "#/definitions/primitive"
21        },
22        "compounds": {
23          "description": "Embedded compound structures",
24          "type": "array",
25          "items": {
26            "$ref": "#/definitions/compound"
27          }
28        },
29        "methods": {
30          "description": "Interface extensions",
31          "type": "array",
32          "items": {
33            "$ref": "#/definitions/method"
34          }
35        },
36        "primitives": {
37          "description": "Embedded primitive attributes",
```

```
38     "type": "array",
39     "items": {
40         "$ref": "#/definitions/primitive"
41     }
42 },
43 "isDistributor": {
44     "description": "Is this related to the distribution ↘
45         process?",
46     "type": "boolean",
47     "default": false
48 },
49 "isGettable": {
50     "description": "Is there a getter?",
51     "type": "boolean",
52     "default": false
53 },
54 "isObservable": {
55     "description": "Is there an observer?",
56     "type": "boolean",
57     "default": false
58 },
59 "isPersistent": {
60     "description": "Are contents persistent?",
61     "type": "boolean",
62     "default": false
63 },
64 "isSettable": {
65     "description": "Is there a setter?",
66     "type": "boolean",
67     "default": false
68 }
69 },
70 "method": {
71     "type": "object",
72     "required": [
73         "name",
74         "input",
75         "output"
76     ],
77     "properties": {
78         "name": {
79             "description": "The method's name",
80             "type": "string"
81         },
82         "input": {
83             "description": "A list of parameter types",
84             "type": "array",
85             "items": {
86                 "type": "string"
```

```
87     }
88   },
89   "output": {
90     "description": "Return type of method",
91     "type": "string"
92   },
93   "description": {
94     "description": "A further description",
95     "type": "string"
96   }
97 }
98 },
99 "primitive": {
100   "type": "object",
101   "required": [
102     "name",
103     "type"
104   ],
105   "properties": {
106     "name": {
107       "description": "The attribute's name",
108       "type": "string"
109     },
110     "type": {
111       "description": "The attribute's type, e.g. 'double'. ↘
112         This has nothing to do with json types",
113       "type": "string"
114     },
115     "isDistributor": {
116       "description": "Is this related to the distribution ↘
117         process?",
118       "type": "boolean",
119       "default": false
120     },
121     "isGettable": {
122       "description": "Will there be a getter?",
123       "type": "boolean",
124       "default": false
125     },
126     "isObservable": {
127       "description": "Will there be an observer?",
128       "type": "boolean",
129       "default": false
130     },
131     "isSettable": {
132       "description": "Will there be a setter?",
133       "type": "boolean",
134       "default": false
135     },
136     "unit": {
```

```
135         "description": "The attribute's unit, e.g., ms",
136         "type": "string"
137     },
138     "description": {
139         "description": "A further description",
140         "type": "string"
141     }
142 }
143 },
144 },
145 "required": [
146     "name",
147     "domain",
148     "NDM",
149     "version"
150 ],
151 "properties": {
152     "name": {
153         "description": "The layer's name",
154         "type": "string"
155     },
156     "domain": {
157         "description": "The layer's domain",
158         "type": "string"
159     },
160     "version": {
161         "description": "Version of the layer definition",
162         "type": "number"
163     },
164     "NDM": {
165         "description": "The layer's native data model",
166         "$ref": "#/definitions/compound"
167     },
168     "compounds": {
169         "description": "The layer's compound structures",
170         "type": "array",
171         "items": {
172             "$ref": "#/definitions/compound"
173         }
174     },
175     "methods": {
176         "description": "Interface extensions",
177         "type": "array",
178         "items": {
179             "$ref": "#/definitions/method"
180         }
181     },
182     "primitives": {
183         "description": "The layer's primitive attributes",
184         "type": "array",
```

```
185     "items": {
186         "$ref": "#/definitions/primitive"
187     }
188 }
189 }
190 }
```

Listing A.2 – Layer definition.

A.3 Component Definition

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "title": "Component",
4   "description": "Component definition",
5   "type": "object",
6   "definitions": {
7     "result": {
8       "type": "object",
9       "required": [
10        "name",
11        "parameters"
12      ],
13      "properties": {
14        "name": {
15          "description": "The result's name",
16          "type": "string"
17        },
18        "parameters": {
19          "description": "The result's parameters",
20          "type": "array",
21          "items": {
22            "type": "object"
23          }
24        }
25      }
26    },
27    "parameter": {
28      "type": "object",
29      "required": [
30        "name",
31        "isMandatory",
32        "values"
33      ],
34      "properties": {
35        "name": {
36          "description": "The parameter's name",
```

```
37         "type": "string"
38     },
39     "isMandatory": {
40         "type": "boolean"
41     },
42     "values": {
43         "description": "A list of possible values",
44         "type": "array",
45         "items": {
46             "type": "string"
47         }
48     }
49 }
50 }
51 },
52 "required": [
53     "name",
54     "domain",
55     "layer",
56     "version"
57 ],
58 "properties": {
59     "name": {
60         "description": "The component's name",
61         "type": "string"
62     },
63     "version": {
64         "description": "The component's version",
65         "type": "integer"
66     },
67     "domain": {
68         "description": "The component's domain",
69         "type": "string"
70     },
71     "layer": {
72         "description": "The component's layer",
73         "type": "string"
74     },
75     "parameters": {
76         "description": "The component's parameters",
77         "type": "array",
78         "items": {
79             "$ref": "#/definitions/parameter"
80         }
81     },
82     "resources": {
83         "description": "The component's resources",
84         "type": "array",
85         "items": {
86             "type": "string"
```



```
87     }
88   },
89   "results": {
90     "description": "The component's results",
91     "type": "array",
92     "items": {
93       "$ref": "#/definitions/result"
94     }
95   }
96 }
97 }
```

Listing A.3 – Component definition.

A.4 Scenario Definition

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "title": "Scenario",
4   "description": "Scenario Definition",
5   "type": "object",
6   "definitions": {
7     "domainReference": {
8       "type": "object",
9       "properties": {
10        "domain": {
11          "description": "The domain's name",
12          "type": "string"
13        },
14        "referencePath": {
15          "description": "The path to the referenceFile",
16          "type": "string"
17        }
18      }
19    },
20    "observer": {
21      "type": "object",
22      "properties": {
23        "task": {
24          "description": "The observer's task",
25          "type": "string"
26        },
27        "element": {
28          "element": "The element under observation",
29          "type": "string"
30        },
31        "filter": {
```

```
32     "element": "Filter for certain elements?",
33     "type": "string"
34 },
35 "period": {
36     "element": "Observation cycle",
37     "type": "number"
38 },
39 "trigger": {
40     "element": "Additional conditions",
41     "type": "string"
42 },
43 "type": {
44     "element": "The serialization method",
45     "type": "string"
46 }
47 }
48 },
49 "buildingBlock": {
50     "type": "object",
51     "properties": {
52         "instanceID": {
53             "description": "The bb's id",
54             "type": "string"
55         },
56         "domain": {
57             "description": "The bb's domain",
58             "type": "string"
59         },
60         "layer": {
61             "description": "The bb's layer",
62             "type": "string"
63         },
64         "type": {
65             "description": "The bb's component",
66             "type": "string"
67         },
68         "stepLength": {
69             "description": "The bb's step length",
70             "type": "number"
71         },
72         "isExternal": {
73             "description": "Is the bb external?",
74             "type": "boolean"
75         },
76         "parameters": {
77             "description": "The bb's parameters",
78             "type": "object",
79             "additionalProperties": {
80                 "type": "string"
81             }
82         }
83     }
84 }
```

```
82     },
83     "resources": {
84         "description": "The bb's resources",
85         "type": "object",
86         "additionalProperties": {
87             "type": "string"
88         }
89     },
90     "results": {
91         "description": "The bb's results",
92         "type": "object",
93         "additionalProperties": {
94             "type": "string"
95         }
96     },
97     "responsibilities": {
98         "description": "The bb's responsibilities",
99         "type": "array",
100        "items": {
101            "type": "string"
102        }
103    },
104    "synchronized": {
105        "description": "Synchronization modes",
106        "type": "boolean"
107    },
108    "observers": {
109        "description": "The bb's observers",
110        "type": "array",
111        "items": {
112            "$ref": "#/definitions/observer"
113        }
114    }
115 }
116 },
117 "translator": {
118     "type": "object",
119     "properties": {
120         "translatorID": {
121             "description": "The translator's instance id",
122             "type": "string"
123         },
124         "type": {
125             "description": "The translator's type",
126             "type": "string"
127         },
128         "domain": {
129             "description": "The translator's domain",
130             "type": "string"
131         },

```

```
132     "layerA": {
133         "description": "The first layer",
134         "type": "string"
135     },
136     "layerB": {
137         "description": "The second layer",
138         "type": "string"
139     },
140     "resources": {
141         "description": "The translator's resources",
142         "type": "object",
143         "additionalProperties": {
144             "type": "string"
145         }
146     },
147     "responsibilitiesA": {
148         "description": "The first layer's responsibilities",
149         "type": "array",
150         "items": {
151             "type": "string"
152         }
153     },
154     "responsibilitiesB": {
155         "description": "The second layer's responsibilities",
156         "type": "array",
157         "items": {
158             "type": "string"
159         }
160     },
161     "parameters": {
162         "description": "Custom parameters",
163         "type": "object",
164         "additionalProperties": {
165             "type": "string"
166         }
167     }
168 },
169 "projector": {
170     "type": "object",
171     "properties": {
172         "projectorID": {
173             "description": "The projector's instance id",
174             "type": "string"
175         },
176     },
177     "type": {
178         "description": "The projector's type",
179         "type": "string"
180     },
181     "domainA": {
```

```
182         "description": "The first domain",
183         "type": "string"
184     },
185     "layerA": {
186         "description": "The first layer",
187         "type": "string"
188     },
189     "domainB": {
190         "description": "The second domain",
191         "type": "string"
192     },
193     "layerB": {
194         "description": "The second layer",
195         "type": "string"
196     },
197     "resources": {
198         "description": "The projector's resources",
199         "type": "object",
200         "additionalProperties": {
201             "type": "string"
202         }
203     },
204     "parameters": {
205         "description": "Custom parameters",
206         "type": "object",
207         "additionalProperties": {
208             "type": "string"
209         }
210     }
211 }
212 }
213 },
214 "required": [
215     "name",
216     "domain",
217     "NDM",
218     "version"
219 ],
220 "properties": {
221     "scenarioID": {
222         "description": "The scenario's id",
223         "type": "string"
224     },
225     "scenarioStart": {
226         "description": "The scenario's start time",
227         "type": "number"
228     },
229     "scenarioEnd": {
230         "description": "The scenario's end time",
231         "type": "number"
```

```
232     },
233     "domainReferences": {
234         "description": "The scenario's domain references",
235         "type": "array",
236         "items": {
237             "$ref": "#/definitions/domainReference"
238         }
239     },
240     "execution": {
241         "description": "Execution related parameters",
242         "type": "object",
243         "properties": {
244             "constraints": {
245                 "type": "string"
246             },
247             "numSynced": {
248                 "type": "number"
249             },
250             "priority": {
251                 "type": "number"
252             },
253             "randomSeed": {
254                 "type": "number"
255             }
256         }
257     },
258     "buildingBlocks": {
259         "description": "The scenario's building blocks",
260         "type": "array",
261         "items": {
262             "$ref": "#/definitions/buildingBlock"
263         }
264     },
265     "translators": {
266         "description": "The scenario's translators",
267         "type": "array",
268         "items": {
269             "$ref": "#/definitions/translator"
270         }
271     },
272     "projectors": {
273         "description": "The scenario's projectors",
274         "type": "array",
275         "items": {
276             "$ref": "#/definitions/projector"
277         }
278     }
279 }
280 }
```

Listing A.4 – Scenario definition.

A.5 Avro Definitions

```
1 {
2   "type": "record",
3   "namespace": "eu.fau.cs7.daceDS.datamodel",
4   "name": "CtrlMsg",
5   "fields": [
6     {
7       "name": "sender",
8       "type": "string"
9     },
10    {
11      "name": "receiver",
12      "type": "string"
13    },
14    {
15      "name": "command",
16      "type": "string"
17    }
18  ]
19 }
```

Listing A.5 – CtrlMsg.avro definition.

```
1 {
2   "type": "record",
3   "name": "ResourceFile",
4   "namespace": "eu.fau.cs7.daceDS.datamodel",
5   "fields": [
6     {
7       "name": "id",
8       "type": "string"
9     },
10    {
11      "name": "type",
12      "type": "string"
13    },
14    {
15      "name": "file",
16      "type": [
17        "null",
18        "bytes"
19      ],

```

```
20     "default": null
21   },
22   {
23     "name": "fileReference",
24     "type": [
25       "null",
26       "string"
27     ],
28     "default": null
29   }
30 ]
31 }
```

Listing A.6 – ResourceFile.avro definition.

```
1 {
2   "type": "record",
3   "namespace": "eu.fau.cs7.daceDS.datamodel",
4   "name": "SyncMsg",
5   "fields": [
6     {
7       "name": "sender",
8       "type": "string"
9     },
10    {
11      "name": "action",
12      "type": "string",
13      "default": "request"
14    },
15    {
16      "name": "time",
17      "type": "long"
18    },
19    {
20      "name": "epoch",
21      "type": "int",
22      "default": 0
23    },
24    {
25      "name": "messages",
26      "type": {
27        "type": "map",
28        "values": "long",
29        "default": {}
30      }
31    }
32  ]
33 }
```

Listing A.7 – SyncMsg.avro definition.

Appendix B

Evaluation Resources

B.1 Demo Definitions

```
1 {
2   "name": "number",
3   "version": 1,
4   "description": "this domain covers numbers",
5   "reference": {
6     "name": "interval",
7     "key": {
8       "name": "ID",
9       "type": "string"
10    },
11    "additional": [
12      {
13        "name": "lowerBound",
14        "type": "int"
15      },
16      {
17        "name": "upperBound",
18        "type": "int"
19      }
20    ]
21  }
22 }
```

Listing B.1 – Number domain definition.

```
1 {
2   "name": "integer",
3   "domain": "number",
4   "version": 1,
5   "NDM": {
```

```
6     "name": "demoInteger",
7     "isPersistent": false,
8     "isGettable": false,
9     "isSettable": false,
10    "isObservable": true,
11    "compounds": [],
12    "key": {
13      "name": "ID",
14      "type": "string",
15      "isGettable": false,
16      "isSettable": false,
17      "isObservable": false
18    },
19    "methods": [],
20    "primitives": [
21      {
22        "name": "value",
23        "type": "int",
24        "isGettable": false,
25        "isSettable": false,
26        "isObservable": false
27      }
28    ]
29  },
30  "compounds": [
31    {
32      "name": "interval",
33      "isPersistent": true,
34      "isGettable": true,
35      "isSettable": false,
36      "isObservable": true,
37      "compounds": [],
38      "key": {
39        "name": "ID",
40        "type": "string",
41        "isGettable": false,
42        "isSettable": false,
43        "isObservable": false
44      },
45      "methods": [],
46      "primitives": [
47        {
48          "name": "elements",
49          "type": "list<demoInteger>",
50          "isGettable": true,
51          "isSettable": false,
52          "isObservable": true,
53          "isDistributor": true
54        },
55        {
```

```
56         "name": "elementCount",
57         "type": "int",
58         "isGettable": true,
59         "isSettable": false,
60         "isObservable": true
61     }
62 ]
63 }
64 ],
65 "methods": [],
66 "primitives": []
67 }
```

Listing B.2 – Integer layer definition.

```
1 {
2   "name": "double",
3   "domain": "number",
4   "version": 1,
5   "NDM": {
6     "name": "demoDouble",
7     "isPersistent": false,
8     "isGettable": false,
9     "isSettable": false,
10    "isObservable": true,
11    "compounds": [],
12    "key": {
13      "name": "ID",
14      "type": "string",
15      "isGettable": false,
16      "isSettable": false,
17      "isObservable": false
18    },
19    "methods": [],
20    "primitives": [
21      {
22        "name": "value",
23        "type": "double",
24        "isGettable": false,
25        "isSettable": false,
26        "isObservable": false
27      }
28    ]
29  },
30  "compounds": [
31    {
32      "name": "interval",
33      "isPersistent": true,
34      "isGettable": true,
```

```
35     "isSettable": false,
36     "isObservable": true,
37     "compounds": [],
38     "key": {
39         "name": "ID",
40         "type": "string",
41         "isGettable": false,
42         "isSettable": false,
43         "isObservable": false
44     },
45     "methods": [],
46     "primitives": [
47         {
48             "name": "elements",
49             "type": "list<demoDouble>",
50             "isGettable": true,
51             "isSettable": false,
52             "isObservable": true,
53             "isDistributor": true
54         },
55         {
56             "name": "elementCount",
57             "type": "int",
58             "isGettable": true,
59             "isSettable": false,
60             "isObservable": true
61         }
62     ]
63 },
64 ],
65 "methods": [],
66 "primitives": []
67 }
```

Listing B.3 – Double layer definition.

```
1 {
2     "name": "color",
3     "version": 1,
4     "description": "this domain covers colors",
5     "reference": {
6         "name": "cell",
7         "key": {
8             "name": "ID",
9             "type": "string"
10        },
11        "additional": [
12            {
13                "name": "x",
```

```
14         "type": "int"
15     },
16     {
17         "name": "y",
18         "type": "int"
19     }
20 ]
21 }
22 }
```

Listing B.4 – Color domain definition.

```
1 {
2     "name": "rgb3",
3     "domain": "color",
4     "version": 1,
5     "NDM": {
6         "name": "pixel",
7         "isPersistent": true,
8         "isGettable": true,
9         "isSettable": false,
10        "isObservable": true,
11        "compounds": [],
12        "key": {
13            "name": "ID",
14            "type": "int",
15            "isGettable": false,
16            "isSettable": false,
17            "isObservable": false
18        },
19        "methods": [
20            {
21                "name": "increaseTemperature",
22                "input": [],
23                "output": "void"
24            },
25            {
26                "name": "dereaseTemperature",
27                "input": [],
28                "output": "void"
29            }
30        ],
31        "primitives": [
32            {
33                "name": "red",
34                "type": "bool",
35                "isGettable": false,
36                "isSettable": true,
37                "isObservable": false
```

```
38     },
39     {
40         "name": "green",
41         "type": "bool",
42         "isGettable": false,
43         "isSettable": true,
44         "isObservable": false
45     },
46     {
47         "name": "blue",
48         "type": "bool",
49         "isGettable": false,
50         "isSettable": true,
51         "isObservable": false
52     }
53 ]
54 },
55 "compounds": [
56     {
57         "name": "cell",
58         "isPersistent": true,
59         "isGettable": true,
60         "isSettable": false,
61         "isObservable": true,
62         "compounds": [],
63         "key": {
64             "name": "ID",
65             "type": "string",
66             "isGettable": false,
67             "isSettable": false,
68             "isObservable": false
69         },
70         "methods": [],
71         "primitives": [
72             {
73                 "name": "pixels",
74                 "type": "list<pixel>",
75                 "isGettable": true,
76                 "isSettable": false,
77                 "isObservable": true,
78                 "isDistributor": true
79             }
80         ]
81     }
82 ],
83 "methods": [],
84 "primitives": []
85 }
```

Listing B.5 – RGB3 layer definition.

```
1 {
2   "name": "rgb24",
3   "domain": "color",
4   "version": 1,
5   "NDM": {
6     "name": "pixel",
7     "isPersistent": true,
8     "isGettable": true,
9     "isSettable": false,
10    "isObservable": true,
11    "compounds": [],
12    "key": {
13      "name": "ID",
14      "type": "int",
15      "isGettable": false,
16      "isSettable": false,
17      "isObservable": false
18    },
19    "methods": [
20      {
21        "name": "increaseTemperature",
22        "input": [],
23        "output": "void"
24      },
25      {
26        "name": "dereaseTemperature",
27        "input": [],
28        "output": "void"
29      }
30    ],
31    "primitives": [
32      {
33        "name": "red",
34        "type": "char",
35        "isGettable": false,
36        "isSettable": true,
37        "isObservable": false
38      },
39      {
40        "name": "green",
41        "type": "char",
42        "isGettable": false,
43        "isSettable": true,
44        "isObservable": false
45      },
46      {
47        "name": "blue",
48        "type": "char",
49        "isGettable": false,
```

```
50     "isSettable": true,
51     "isObservable": false
52   }
53 ]
54 },
55 "compounds": [
56   {
57     "name": "cell",
58     "isPersistent": true,
59     "isGettable": true,
60     "isSettable": false,
61     "isObservable": true,
62     "compounds": [],
63     "key": {
64       "name": "ID",
65       "type": "string",
66       "isGettable": false,
67       "isSettable": false,
68       "isObservable": false
69     },
70     "methods": [],
71     "primitives": [
72       {
73         "name": "pixels",
74         "type": "list<pixel>",
75         "isGettable": true,
76         "isSettable": false,
77         "isObservable": true,
78         "isDistributor": true
79       }
80     ]
81   }
82 ],
83 "methods": [],
84 "primitives": []
85 }
```

Listing B.6 – RGB24 layer definition.

B.2 Scenario Definition Experiment 1

```
1 {
2   "scenarioID": "experiment1",
3   "domainReferences": {
4     "number": "NumberIntervals.xml"
5   },
6   "simulationStart": 0,
```

```
7     "simulationEnd": 1500,
8     "execution": {
9         "randomSeed": 87,
10        "constraints": "",
11        "priority": 0,
12        "syncedParticipants": 1
13    },
14    "buildingBlocks": [
15        {
16            "instanceID": "negSim",
17            "type": "IntegerSimulator",
18            "stepLength": 1,
19            "layer": "integers",
20            "domain": "numbers",
21            "parameters": {
22                "F": "growingSine",
23                "N": "2",
24                "V": "-350,-916"
25            },
26            "resources": {},
27            "responsibilities": [ "negative", "positive"
28        ],
29            "results": {},
30            "synchronized": false,
31            "isExternal": false,
32            "observers": [ {
33                "task": "publish",
34                "element": "demoInteger",
35                "filter": "",
36                "period": 1,
37                "trigger": "",
38                "type": "avro"
39            }
40        ]
41    }
42 }
43 ],
44 "translators": [],
45 "projectors": []
46 }
```

Listing B.7 – Scenario definition of experiment 1.

B.3 Scenario Definition Experiment 3

```
1 {
2     "scenarioID": "experiment3",
```

```
3     "domainReferences": {
4         "number": "NumberIntervals.xml"
5     },
6     "simulationStart": 0,
7     "simulationEnd": 1500,
8     "execution": {
9         "randomSeed": 87,
10        "constraints": "",
11        "priority": 0,
12        "syncedParticipants": 3
13    },
14    "buildingBlocks": [
15        {
16            "instanceID": "negSim",
17            "type": "IntegerSimulator",
18            "stepLength": 20,
19            "layer": "integers",
20            "domain": "numbers",
21            "parameters": {
22                "F": "growingSine",
23                "N": "2",
24                "V": "-350,-916"
25            },
26            "resources": {},
27            "responsibilities": [ "negative"
28            ],
29            "results": {},
30            "synchronized": true,
31            "isExternal": false,
32            "observers": [ {
33                "task": "publish",
34                "element": "demoInteger",
35                "filter": "",
36                "period": 1,
37                "trigger": "",
38                "type": "avro"
39            }
40        ]
41    },
42    {
43        "instanceID": "posSim",
44        "type": "DoubleSimulator",
45        "stepLength": 1,
46        "layer": "doubles",
47        "domain": "numbers",
48        "parameters": {
49            "F": "growingSine",
50            "N": "0",
51            "V": ""
52        },
```

```
53     "resources": {},
54     "responsibilities": [ "positive"
55 ],
56     "results": {},
57     "synchronized": true,
58     "isExternal": false,
59     "observers": [ {
60         "task": "publish",
61         "element": "demoDouble",
62         "filter": "",
63         "period": 1,
64         "trigger": "",
65         "type": "avro"
66     }
67 ]
68 }
69 ],
70 "translators": [
71     {
72         "translatorID": "translatorID0",
73         "type": "intDbl",
74         "domain": "numbers",
75         "layerA": "integers",
76         "responsibilitiesA": [ "positive"
77 ],
78         "layerB": "doubles",
79         "responsibilitiesB": [ "negative"
80 ],
81         "resources": {},
82         "parameters": {}
83     }
84 ],
85 "projectors": []
86 }
```

Listing B.8 – Scenario definition of experiment 3.

Appendix C

Definitions in the Traffic Domain

C.1 Traffic Domain Definition

```
1 {
2   "name": "traffic",
3   "version": 1,
4   "description": "this domain covers traffic",
5   "reference": {
6     "name": "node",
7     "key": {
8       "name": "nodeID",
9       "type": "string"
10    },
11    "additional": [
12      {
13        "name": "x",
14        "type": "double"
15      },
16      {
17        "name": "y",
18        "type": "double"
19      },
20      {
21        "name": "z",
22        "type": "double"
23      }
24    ]
25  }
26 }
```

Listing C.1 – Traffic domain definition.

C.2 Macroscopic Layer Definition

```
1 {
2   "name": "macro",
3   "domain": "traffic",
4   "version": 1,
5   "NDM": {
6     "name": "link",
7     "isPersistent": true,
8     "isGettable": true,
9     "isSettable": false,
10    "isObservable": true,
11    "isDistributor": true,
12    "compounds": [],
13    "key": {
14      "name": "linkID",
15      "type": "string",
16      "isGettable": false,
17      "isSettable": false,
18      "isObservable": false
19    },
20    "methods": [],
21    "primitives": [
22      {
23        "name": "allowedModes",
24        "type": "map<integer,string>",
25        "isGettable": true,
26        "isSettable": true,
27        "isObservable": true
28      },
29      {
30        "name": "capacity",
31        "type": "double",
32        "isGettable": true,
33        "isSettable": true,
34        "isObservable": false,
35        "unit": "veh/km"
36      },
37      {
38        "name": "density",
39        "type": "double",
40        "isGettable": true,
41        "isSettable": false,
42        "isObservable": true,
43        "unit": "veh/km"
44      },
45      {
46        "name": "flow",
47        "type": "double",
```



```
48     "isGettable": true ,
49     "isSettable": false ,
50     "isObservable": true ,
51     "unit": "veh/h"
52 },
53 {
54     "name": "length",
55     "type": "double",
56     "isGettable": true ,
57     "isSettable": false ,
58     "isObservable": false ,
59     "unit": "m"
60 },
61 {
62     "name": "paths",
63     "type": "map<string,double>",
64     "isGettable": true ,
65     "isSettable": false ,
66     "isObservable": true
67 },
68 {
69     "name": "speed",
70     "type": "double",
71     "isGettable": true ,
72     "isSettable": false ,
73     "isObservable": true ,
74     "unit": "km/h"
75 },
76 {
77     "name": "turningProbabilities",
78     "type": "map<string,double>",
79     "isGettable": true ,
80     "isSettable": false ,
81     "isObservable": true
82 }
83 ]
84 },
85 "compounds": [],
86 "methods": [
87     {
88         "name": "runAssignment",
89         "input": [],
90         "output": "void"
91     }
92 ],
93 "primitives": []
94 }
```

Listing C.2 – Macroscopic layer definition.

C.3 Mesoscopic Layer Definition

```
1 {
2   "name": "meso",
3   "domain": "traffic",
4   "version": 1,
5   "NDM": {
6     "name": "vehicle",
7     "isPersistent": false,
8     "isGettable": true,
9     "isSettable": false,
10    "isObservable": true,
11    "compounds": [],
12    "key": {
13      "name": "vehicleID",
14      "type": "string",
15      "isGettable": false,
16      "isSettable": false,
17      "isObservable": false
18    },
19    "methods": [],
20    "primitives": [
21      {
22        "name": "link",
23        "type": "string",
24        "isGettable": true,
25        "isSettable": true,
26        "isObservable": true
27      },
28      {
29        "name": "route",
30        "type": "list<string>",
31        "isGettable": true,
32        "isSettable": true,
33        "isObservable": false
34      },
35      {
36        "name": "type",
37        "type": "string",
38        "isGettable": true,
39        "isSettable": false,
40        "isObservable": false
41      }
42    ]
43  },
44  "compounds": [
45    {
46      "name": "link",
47      "isPersistent": true,
```

```
48     "isGettable": true,
49     "isSettable": false,
50     "isObservable": true,
51     "compounds": [],
52     "key": {
53       "name": "linkID",
54       "type": "string",
55       "isGettable": false,
56       "isSettable": false,
57       "isObservable": false
58     },
59     "methods": [],
60     "primitives": [
61       {
62         "name": "flow",
63         "type": "double",
64         "isGettable": true,
65         "isSettable": false,
66         "isObservable": true,
67         "unit": "veh/h"
68       },
69       {
70         "name": "occupancy",
71         "type": "double",
72         "isGettable": true,
73         "isSettable": false,
74         "isObservable": true,
75         "unit": "veh/km"
76       },
77       {
78         "name": "speed",
79         "type": "double",
80         "isGettable": true,
81         "isSettable": false,
82         "isObservable": true,
83         "unit": "m/s"
84       },
85       {
86         "name": "vehicles",
87         "type": "list<vehicle>",
88         "isGettable": true,
89         "isSettable": false,
90         "isObservable": true,
91         "isDistributor": true
92       },
93       {
94         "name": "vehicleIDs",
95         "type": "list<string>",
96         "isGettable": true,
97         "isSettable": false,
```

```

98         "isObservable": true
99     }
100 ]
101 }
102 ],
103 "methods": [],
104 "primitives": []
105 }

```

Listing C.3 – Mesoscopic layer definition.

C.4 Microscopic Layer Definition

```

1 {
2   "name": "micro",
3   "domain": "traffic",
4   "version": 1,
5   "NDM": {
6     "name": "vehicle",
7     "isPersistent": false,
8     "isGettable": true,
9     "isSettable": false,
10    "isObservable": true,
11    "compounds": [],
12    "key": {
13      "name": "vehicleID",
14      "type": "string",
15      "isGettable": false,
16      "isSettable": false,
17      "isObservable": false
18    },
19    "methods": [],
20    "primitives": [
21      {
22        "name": "acceleration",
23        "type": "double",
24        "isGettable": true,
25        "isSettable": true,
26        "isObservable": true,
27        "unit": "m/s2",
28        "description": "will lead to: ↘
                interaction[...]micro.vehicle.acceleration.get, ↘
                interaction[...]micro.vehicle.acceleration.ret, ↘
                provision[...]micro.vehicle.acceleration"
29      },
30      {
31        "name": "angle",

```

```
32     "type": "double",
33     "isGettable": true,
34     "isSettable": true,
35     "isObservable": true,
36     "unit": "deg"
37 },
38 {
39     "name": "edge",
40     "type": "string",
41     "isGettable": true,
42     "isSettable": true,
43     "isObservable": true
44 },
45 {
46     "name": "lane",
47     "type": "int",
48     "isGettable": true,
49     "isSettable": true,
50     "isObservable": true
51 },
52 {
53     "name": "position",
54     "type": "vec3",
55     "isGettable": true,
56     "isSettable": true,
57     "isObservable": true
58 },
59 {
60     "name": "positionEdge",
61     "type": "double",
62     "isGettable": true,
63     "isSettable": true,
64     "isObservable": true,
65     "unit": "m",
66     "description": "distance to beginning of current edge"
67 },
68 {
69     "name": "route",
70     "type": "list<string>",
71     "isGettable": true,
72     "isSettable": true,
73     "isObservable": false
74 },
75 {
76     "name": "slope",
77     "type": "double",
78     "isGettable": true,
79     "isSettable": false,
80     "isObservable": true,
81     "unit": "deg"
```

```
82     },
83     {
84         "name": "speed",
85         "type": "double",
86         "isGettable": true,
87         "isSettable": true,
88         "isObservable": true,
89         "unit": "m/s"
90     },
91     {
92         "name": "type",
93         "type": "string",
94         "isGettable": true,
95         "isSettable": false,
96         "isObservable": false
97     }
98 ]
99 },
100 "compounds": [
101     {
102         "name": "edge",
103         "isPersistent": true,
104         "isGettable": true,
105         "isSettable": false,
106         "isObservable": true,
107         "compounds": [],
108         "key": {
109             "name": "edgeID",
110             "type": "string",
111             "isGettable": false,
112             "isSettable": false,
113             "isObservable": false
114         },
115         "methods": [
116             {
117                 "name": "setAllowedTypes",
118                 "input": [
119                     "integer",
120                     "list_string"
121                 ],
122                 "output": "void"
123             },
124             {
125                 "name": "getAllowedTypes",
126                 "input": [
127                     "integer"
128                 ],
129                 "output": "list<string>"
130             },
131             {
```

```
132     "name": "getLaneNumber",
133     "input": [],
134     "output": "integer"
135 },
136 {
137     "name": "getLaneGradient",
138     "input": [
139         "integer"
140     ],
141     "output": "double"
142 },
143 {
144     "name": "getLaneLength",
145     "input": [
146         "integer"
147     ],
148     "output": "double"
149 },
150 {
151     "name": "getLaneWidth",
152     "input": [
153         "integer"
154     ],
155     "output": "double"
156 }
157 ],
158 "primitives": [
159     {
160         "name": "emission",
161         "type": "string",
162         "isGettable": true,
163         "isSettable": false,
164         "isObservable": true,
165         "description": "some calculated emission information"
166     },
167     {
168         "name": "flow",
169         "type": "double",
170         "isGettable": true,
171         "isSettable": false,
172         "isObservable": true,
173         "unit": "veh/h"
174     },
175     {
176         "name": "occupancy",
177         "type": "double",
178         "isGettable": true,
179         "isSettable": false,
180         "isObservable": true,
181         "unit": "veh/km"
```

```
182     },
183     {
184         "name": "speed",
185         "type": "double",
186         "isGettable": true,
187         "isSettable": false,
188         "isObservable": true,
189         "unit": "m/s"
190     },
191     {
192         "name": "vehicles",
193         "type": "list<vehicle>",
194         "isGettable": true,
195         "isSettable": false,
196         "isObservable": true,
197         "isDistributor": true
198     },
199     {
200         "name": "vehicleIDs",
201         "type": "list<string>",
202         "isGettable": true,
203         "isSettable": false,
204         "isObservable": true
205     }
206 ]
207 },
208 {
209     "name": "detector",
210     "isPersistent": true,
211     "isGettable": true,
212     "isSettable": false,
213     "isObservable": true,
214     "compounds": [],
215     "key": {
216         "name": "detectorID",
217         "type": "string",
218         "isGettable": false,
219         "isSettable": false,
220         "isObservable": false
221     },
222     "methods": [],
223     "primitives": [
224         {
225             "name": "values",
226             "type": "map<boolean, double, integer, string>",
227             "isGettable": true,
228             "isSettable": false,
229             "isObservable": true,
```



```
230         "description": "detector's values, very generic ↘  
                because no further knowledge, key is ↘  
                det.attribute name, value is det.value"  
231     }  
232 ]  
233 }  
234 ],  
235 "methods": [],  
236 "primitives": []  
237 }
```

Listing C.4 – Microscopic layer definition.

C.5 Submicroscopic Layer Definition

```
1 {  
2   "name": "submicro",  
3   "domain": "traffic",  
4   "version": 1,  
5   "NDM": {  
6     "name": "vehicle",  
7     "isPersistent": false,  
8     "isGettable": true,  
9     "isSettable": false,  
10    "isObservable": true,  
11    "compounds": [  
12      {  
13        "name": "sensor",  
14        "isPersistent": false,  
15        "isGettable": true,  
16        "isSettable": false,  
17        "isObservable": true,  
18        "compounds": [],  
19        "key": {  
20          "name": "sensorID",  
21          "type": "string",  
22          "isGettable": false,  
23          "isSettable": false,  
24          "isObservable": false  
25        },  
26        "methods": [],  
27        "primitives": [  
28          {  
29            "name": "type",  
30            "type": "string",  
31            "isGettable": false,  
32            "isSettable": false,
```

```
33         "isObservable": false
34     },
35     {
36         "name": "data",
37         "type": "list<bool,bytes,double,int,string>",
38         "isGettable": false,
39         "isSettable": false,
40         "isObservable": false,
41         "isDistributor": false
42     }
43 ]
44 }
45 ],
46 "key": {
47     "name": "vehicleID",
48     "type": "string",
49     "isGettable": false,
50     "isSettable": false,
51     "isObservable": false
52 },
53 "methods": [],
54 "primitives": [
55     {
56         "name": "acceleration",
57         "type": "vec3",
58         "isGettable": true,
59         "isSettable": true,
60         "isObservable": true,
61         "unit": "m/s2,m/s2,m/s2"
62     },
63     {
64         "name": "brake",
65         "type": "double",
66         "isGettable": true,
67         "isSettable": true,
68         "isObservable": true
69     },
70     {
71         "name": "laneID",
72         "type": "string",
73         "isGettable": true,
74         "isObservable": true,
75         "isSettable": false,
76         "description": "contains road and lane id"
77     },
78     {
79         "name": "gear",
80         "type": "int",
81         "isGettable": true,
82         "isSettable": true,
```

```
83         "isObservable": true
84     },
85     {
86         "name": "position",
87         "type": "Vec3",
88         "isGettable": true,
89         "isSettable": true,
90         "isObservable": true
91     },
92     {
93         "name": "rotation",
94         "type": "Vec3",
95         "isGettable": true,
96         "isSettable": true,
97         "isObservable": true,
98         "unit": "deg,deg,deg"
99     },
100    {
101        "name": "route",
102        "type": "list<vec3>",
103        "isGettable": true,
104        "isSettable": true,
105        "isObservable": false
106    },
107    {
108        "name": "speed",
109        "type": "double",
110        "isGettable": true,
111        "isSettable": true,
112        "isObservable": true,
113        "unit": "m/s"
114    },
115    {
116        "name": "steer",
117        "type": "double",
118        "isGettable": true,
119        "isSettable": true,
120        "isObservable": true,
121        "unit": "deg"
122    },
123    {
124        "name": "throttle",
125        "type": "double",
126        "isGettable": true,
127        "isSettable": true,
128        "isObservable": true
129    },
130    {
131        "name": "type",
132        "type": "string",
```

```
133     "isGettable": true,
134     "isSettable": false,
135     "isObservable": false
136   },
137   {
138     "name": "velocity",
139     "type": "list<vec3>",
140     "isGettable": true,
141     "isSettable": true,
142     "isObservable": true,
143     "unit": "m/s,m/s,m/s"
144   }
145 ]
146 },
147 "compounds": [
148   {
149     "name": "lane",
150     "isPersistent": true,
151     "isGettable": true,
152     "isSettable": false,
153     "isObservable": true,
154     "compounds": [],
155     "key": {
156       "name": "laneID",
157       "type": "string",
158       "isGettable": false,
159       "isSettable": false,
160       "isObservable": false,
161       "description": "contains road and lane id"
162     },
163     "methods": [],
164     "primitives": [
165       {
166         "name": "speed",
167         "type": "double",
168         "isGettable": true,
169         "isSettable": false,
170         "isObservable": true,
171         "unit": "m/s"
172       },
173       {
174         "name": "vehicles",
175         "type": "list<vehicle>",
176         "isGettable": true,
177         "isSettable": false,
178         "isObservable": true,
179         "isDistributor": true
180       },
181       {
182         "name": "vehicleIDs",
```

```
183         "type": "list<string>",
184         "isGettable": true,
185         "isSettable": false,
186         "isObservable": true
187     }
188 ]
189 }
190 ],
191 "methods": [],
192 "primitives": []
193 }
```

Listing C.5 – Submicroscopic layer definition.

Appendix D

Definitions in the Communication Domain

```
1 {
2   "name": "communication",
3   "version": 1,
4   "description": "this definition covers the communication \
domain",
5   "reference": {
6     "name": "logicalNetwork",
7     "key": {
8       "name": "networkID",
9       "type": "string"
10    },
11    "additional": [
12      {
13        "name": "description",
14        "type": "string"
15      },
16      {
17        "name": "technology",
18        "type": "string"
19      }
20    ]
21  }
22 }
```

Listing D.1 – Communication domain definition.

```
1 {
2   "name": "80211p",
3   "domain": "communication",
4   "version": 1,
```

```
5  "NDM": {
6    "name": "message",
7    "isPersistent": false,
8    "isGettable": false,
9    "isSettable": false,
10   "isObservable": true,
11   "compounds": [],
12   "key": {
13     "name": "messageID",
14     "type": "string",
15     "isGettable": false,
16     "isSettable": false,
17     "isObservable": false
18   },
19   "methods": [],
20   "primitives": [
21     {
22       "name": "transmissionPower",
23       "type": "double",
24       "isGettable": true,
25       "isSettable": false,
26       "isObservable": false,
27       "unit": "mW"
28     },
29     {
30       "name": "origin",
31       "type": "vec3",
32       "isGettable": true,
33       "isSettable": false,
34       "isObservable": false,
35       "unit": "m"
36     },
37     {
38       "name": "networkID",
39       "type": "string",
40       "isGettable": false,
41       "isSettable": false,
42       "isObservable": false
43     },
44     {
45       "name": "sender",
46       "type": "string",
47       "isGettable": false,
48       "isSettable": false,
49       "isObservable": false
50     },
51     {
52       "name": "receiver",
53       "type": "string",
54       "isGettable": false,
```



```
55     "isSettable": false ,
56     "isObservable": false
57   },
58   {
59     "name": "type",
60     "type": "string",
61     "isGettable": false ,
62     "isSettable": false ,
63     "isObservable": false
64   },
65   {
66     "name": "stage",
67     "type": "string",
68     "isGettable": false ,
69     "isSettable": false ,
70     "isObservable": false
71   },
72   {
73     "name": "channel",
74     "type": "int",
75     "isGettable": false ,
76     "isSettable": false ,
77     "isObservable": false
78   },
79   {
80     "name": "priority",
81     "type": "int",
82     "isGettable": false ,
83     "isSettable": false ,
84     "isObservable": false
85   },
86   {
87     "name": "payload",
88     "type": "string",
89     "isGettable": false ,
90     "isSettable": false ,
91     "isObservable": false
92   }
93 ]
94 },
95 "compounds": [
96   {
97     "name": "logicalNetwork",
98     "isPersistent": true ,
99     "isGettable": true ,
100    "isSettable": false ,
101    "isObservable": true ,
102    "compounds": [],
103    "key": {
104      "name": "networkID",
```

```
105     "type": "string",
106     "isGettable": false,
107     "isSettable": false,
108     "isObservable": false
109 },
110 "methods": [],
111 "primitives": [
112   {
113     "name": "received",
114     "type": "list<message>",
115     "isGettable": true,
116     "isSettable": false,
117     "isObservable": true,
118     "isDistributor": true
119   },
120   {
121     "name": "receivedNumber",
122     "type": "int",
123     "isGettable": true,
124     "isSettable": false,
125     "isObservable": true
126   },
127   {
128     "name": "sent",
129     "type": "list<message>",
130     "isGettable": true,
131     "isSettable": false,
132     "isObservable": true
133   },
134   {
135     "name": "sentNumber",
136     "type": "int",
137     "isGettable": true,
138     "isSettable": false,
139     "isObservable": true
140   }
141 ]
142 },
143 {
144   "name": "node",
145   "isPersistent": false,
146   "isGettable": false,
147   "isSettable": false,
148   "isObservable": true,
149   "compounds": [],
150   "key": {
151     "name": "nodeID",
152     "type": "string",
153     "isGettable": false,
154     "isSettable": false,
```

```
155     "isObservable": false
156   },
157   "methods": [],
158   "primitives": [
159     {
160       "name": "position",
161       "type": "vec3",
162       "isGettable": true,
163       "isSettable": true,
164       "isObservable": true
165     }
166   ]
167 }
168 ],
169 "methods": [],
170 "primitives": []
171 }
```

Listing D.2 – 80211p layer definition.

Appendix E

Exemplary Case Studies

E.1 Study 1

```
1 {
2   "scenarioID": "uc11",
3   "domainReferences": {
4     "traffic": "MiniLine.net.reference.xml"
5   },
6   "simulationStart": 0,
7   "simulationEnd": 100,
8   "execution": {
9     "randomSeed": 23,
10    "constraints": "",
11    "priority": 0,
12    "syncedParticipants": 2
13  },
14  "buildingBlocks": [
15    {
16      "instanceID": "SumoWrapper0",
17      "type": "SumoWrapper",
18      "layer": "micro",
19      "domain": "traffic",
20      "stepLength": 1000,
21      "parameters": {},
22      "resources": { "MiniLine.net.xml": "RoadMap", ↵
23                   "2lps.MiniLine.traffic.SumoWrapper0.xml": ↵
24                   "Traffic", "vtypes.xml": "Additional"
25    },
26    "results": {},
27    "synchronized": true,
28    "isExternal": false,
29    "responsibilities": [ "0", "1", "2"
30  ],
31  "observers": []
32 }
```

```

30     },
31     {
32         "instanceID": "SumoWrapper1",
33         "type": "SumoWrapper",
34         "layer": "micro",
35         "domain": "traffic",
36         "stepLength": 1000,
37         "parameters": {},
38         "resources": { "MiniLine.net.xml": "RoadMap", ↘
39                       "2lps.MiniLine.traffic.SumoWrapper1.xml": ↘
40                       "Traffic", "vtypes.xml": "Additional"
41         },
42         "results": {},
43         "synchronized": true,
44         "isExternal": false,
45         "responsibilities": [ "3", "4", "5"
46         ],
47         "observers": []
48     }
49 ],
50 "translators": [],
51 "projectors": []
52 }
53 {
54     "scenarioID": "uc12",
55     "domainReferences": {
56         "traffic": "MiniLine.net.reference.xml"
57     },
58     "simulationStart": 0,
59     "simulationEnd": 100,
60     "execution": {
61         "randomSeed": 23,
62         "constraints": "",
63         "priority": 0,
64         "syncedParticipants": 2
65     },
66     "buildingBlocks": [
67         {
68             "instanceID": "NorthWest",
69             "type": "SumoWrapper",
70             "layer": "micro",
71             "domain": "traffic",
72             "stepLength": 1000,
73             "parameters": {},
74             "resources": { "Manhattan.net.xml": "RoadMap", ↘
75                       "4lps.Manhattan.traffic.NorthWest.xml": ↘
76                       "Traffic", "Manhattan.types.xml": "Additional"
77             },
78             "results": {},
79             "synchronized": true,

```

```
76         "isExternal": false,
77         "responsibilities": [ "15", "16", "17", "18", ↵
           "19", "25", "26", "27", "28", "29", "35", ↵
           "36", "37", "38", "39", "45", "46", "47", ↵
           "48", "49", "5", "6", "7", "8", "9"
78     ],
79     "observers": []
80 },
81 {
82     "instanceID": "NorthEast",
83     "type": "SumoWrapper",
84     "layer": "micro",
85     "domain": "traffic",
86     "stepLength": 1000,
87     "parameters": {},
88     "resources": { "Manhattan.net.xml": "RoadMap", ↵
           "4lps.Manhattan.traffic.NorthEast.xml": ↵
           "Traffic", "Manhattan.types.xml": "Additional"
89     },
90     "results": {},
91     "synchronized": true,
92     "isExternal": false,
93     "responsibilities": [ "55", "56", "57", "58", ↵
           "59", "65", "66", "67", "68", "69", "75", ↵
           "76", "77", "78", "79", "85", "86", "87", ↵
           "88", "89", "95", "96", "97", "98", "99"
94     ],
95     "observers": []
96 },
97 {
98     "instanceID": "SouthWest",
99     "type": "SumoWrapper",
100    "layer": "micro",
101    "domain": "traffic",
102    "stepLength": 1000,
103    "parameters": {},
104    "resources": { "Manhattan.net.xml": "RoadMap", ↵
           "4lps.Manhattan.traffic.SouthWest.xml": ↵
           "Traffic", "Manhattan.types.xml": "Additional"
105    },
106    "results": {},
107    "synchronized": true,
108    "isExternal": false,
109    "responsibilities": [ "0", "1", "10", "11", "12", ↵
           "13", "14", "2", "20", "21", "22", "23", "24", ↵
           "3", "30", "31", "32", "33", "34", "4", "40", ↵
           "41", "42", "43", "44"
110    ],
111    "observers": []
112 },
```

```

113     {
114         "instanceID": "SouthEast",
115         "type": "SumoWrapper",
116         "layer": "micro",
117         "domain": "traffic",
118         "stepLength": 1000,
119         "parameters": {},
120         "resources": { "Manhattan.net.xml": "RoadMap", ↘
                       "4lps.Manhattan.traffic.SouthEast.xml": ↘
                       "Traffic", "Manhattan.types.xml": "Additional"
121     },
122     "results": {},
123     "synchronized": true,
124     "isExternal": false,
125     "responsibilities": [ "50", "51", "52", "53", ↘
                           "54", "60", "61", "62", "63", "64", "70", ↘
                           "71", "72", "73", "74", "80", "81", "82", ↘
                           "83", "84", "90", "91", "92", "93", "94"
126     ],
127     "observers": []
128 }
129 ],
130 "translators": [],
131 "projectors": []
132 }

```

Listing E.1 – Scenario definition of study 1.1 and 1.2.

E.2 Study 2

```

1 {
2     "scenarioID": "uc2",
3     "domainReferences": {
4         "traffic": "Manhattan.net.reference.xml"
5     },
6     "simulationStart": 0,
7     "simulationEnd": 1800,
8     "execution": {
9         "randomSeed": 23,
10        "constraints": "",
11        "priority": 0,
12        "syncedParticipants": 3
13    },
14    "buildingBlocks": [
15        {
16            "instanceID": "MatsimWrapper0",
17            "type": "MATSimWrapper",

```



```

18     "layer": "meso",
19     "domain": "traffic",
20     "stepLength": 1000,
21     "parameters": {},
22     "resources": { "Manhattan.matsim.network.xml": ↘
23         "RoadMap", "routes.high.plans.xml": "Traffic"
24     },
25     "results": {},
26     "synchronized": true,
27     "isExternal": false,
28     "responsibilities": [ "0", "1", "10", "11", "12", ↘
29         "13", "14", "15", "16", "17", "18", "19", "2", ↘
30         "20", "21", "22", "23", "24", "25", "26", ↘
31         "27", "28", "29", "3", "33", "34", "35", "36", ↘
32         "37", "38", "39", "4", "43", "44", "45", "46", ↘
33         "47", "48", "49", "5", "6", "7", "8", "9", ↘
34         "53", "54", "55", "56", "57", "58", "59", ↘
35         "63", "64", "65", "66", "67", "68", "69", ↘
36         "70", "71", "72", "73", "74", "75", "76", ↘
37         "77", "78", "79", "80", "81", "82", "83", ↘
38         "84", "85", "86", "87", "88", "89", "90", ↘
39         "91", "92", "93", "94", "95", "96", "97", ↘
40         "98", "99"
41     ],
42     "observers": [ { "task": "publish", ↘
43         "element": "link", "filter": "2 28 0 31 5 ↘
44         1 32 66 29 36 4 69 33 40 7 73 10 37 44 77 13 ↘
45         41 48 81 16 45 52 85 19 49 56 89 22 53 60 93 ↘
46         25 57 64 97 101 27 61 3 35 8 104 30 70 107 34 ↘
47         67 74 111 38 71 78 115 42 75 82 119 46 79 86 ↘
48         123 50 83 90 127 54 87 94 131 58 91 98 102 135 ↘
49         62 95 139 65 99 11 39 6 113 120 153 80 117 124 ↘
50         157 84 121 128 161 88 125 132 165 92 129 136 ↘
51         169 96 100 133 140 173 103 137 177 14 43 9 118 ↘
52         151 158 191 122 155 162 195 126 159 166 199 ↘
53         130 163 170 203 134 167 174 207 138 171 178 ↘
54         211 141 175 215 12 17 47 15 20 51 18 23 55 21 ↘
55         26 59 24 63 156 189 196 229 160 193 200 233 ↘
56         164 197 204 237 168 201 208 241 172 205 212 ↘
57         245 176 209 216 249 179 213 253 222 219 226 ↘
58         223 230 194 234 267 198 231 238 271 202 235 ↘
59         242 275 206 239 246 279 210 243 250 283 214 ↘
60         247 254 287 217 251 291 220 260 294 224 257 ↘
61         264 297 228 261 268 301 232 265 272 305 236 ↘
62         269 276 309 240 273 280 313 244 277 284 317 ↘
63         248 281 288 321 252 285 292 325 255 289 329 ↘
64         258 298 332 262 295 302 334 266 299 306 337 ↘
65         270 303 310 340 274 307 314 343 278 311 318 ↘
66         346 282 315 322 349 286 319 326 352 290 323 ↘
67         330 355 293 327 358 296 335 300 333 338 304 ↘

```

```

336 341 308 339 344 312 342 347 316 345 350 ↘
320 348 353 324 351 356 328 354 359 331 357", ↘
    "period": 1000,      "trigger": "",      ↘
    "type": "avro" }
30     ]
31   },
32   {
33     "instanceID": "SumoWrapper1",
34     "type": "SumoWrapper",
35     "layer": "micro",
36     "domain": "traffic",
37     "stepLength": 100,
38     "parameters": { "ghosting": "false"
39   },
40     "resources": { "Manhattan.net.xml": "RoadMap", ↘
41                   "Manhattan.types.xml": "Additional"
42   },
43     "results": {},
44     "synchronized": true,
45     "isExternal": false,
46     "responsibilities": [ "30", "40", "50", "31", ↘
47                           "41", "51", "32", "42", "52", "60", "61", "62"
48   ],
49     "observers": [ { "task": "publish", ↘
50                     "element": "edge",      "filter": "108 142 68 ↘
51                     106 146 180 144 184 218 105 112 145 72 110 143 ↘
52                     150 183 148 181 188 221 109 116 149 76 114 147 ↘
53                     154 187 152 185 192 225 190 186 182 263 259 ↘
54                     256 227", "period": 1000, "trigger": ↘
55                     "", "type": "json" }
56   ]
57   }
58 ],
59 "translators": [
60   {
61     "translatorID": "MicroMesoJava0",
62     "type": "MicroMesoJava",
63     "domain": "traffic",
64     "layerA": "micro",
65     "responsibilitiesA": [ "0", "1", "10", "11", "12", ↘
66                           "13", "14", "15", "16", "17", "18", "19", "2", ↘
67                           "20", "21", "22", "23", "24", "25", "26", ↘
68                           "27", "28", "29", "3", "33", "34", "35", "36", ↘
69                           "37", "38", "39", "4", "43", "44", "45", "46", ↘
70                           "47", "48", "49", "5", "6", "7", "8", "9", ↘
71                           "53", "54", "55", "56", "57", "58", "59", ↘
72                           "63", "64", "65", "66", "67", "68", "69", ↘
73                           "70", "71", "72", "73", "74", "75", "76", ↘
74                           "77", "78", "79", "80", "81", "82", "83", ↘
75                           "84", "85", "86", "87", "88", "89", "90", ↘

```

```

        "91", "92", "93", "94", "95", "96", "97", ↵
        "98", "99"
58     ],
59     "layerB": "meso",
60     "responsibilitiesB": [ "30", "40", "50", "31", ↵
        "41", "51", "32", "42", "52", "60", "61", "62"
61     ],
62     "parameters": {},
63     "resources": { "Manhattan.matsim.network.xml": ↵
        "MesoNetwork"
64     }
65 }
66 ],
67 "projectors": []
68 }

```

Listing E.2 – Scenario definition of study 2 (multi-level topology with high traffic).

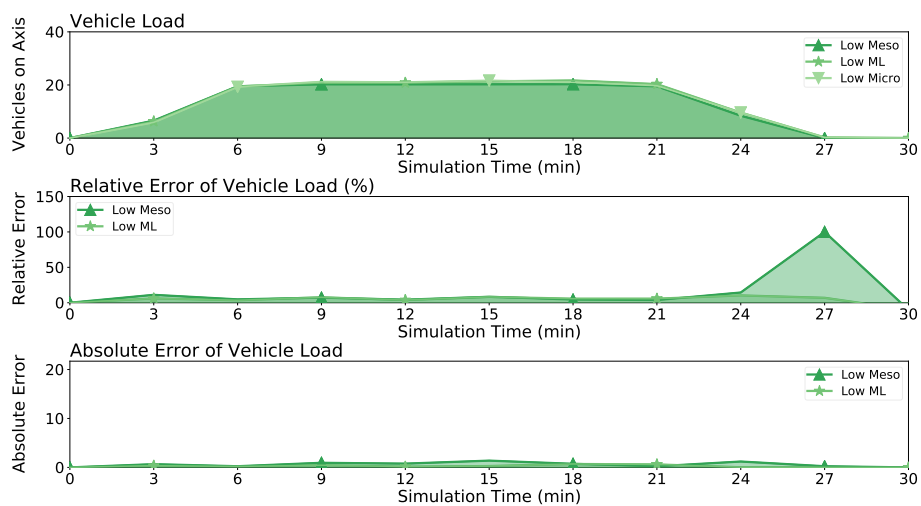


Figure E.1 – Errors of traffic load over time on the south axis for low traffic.

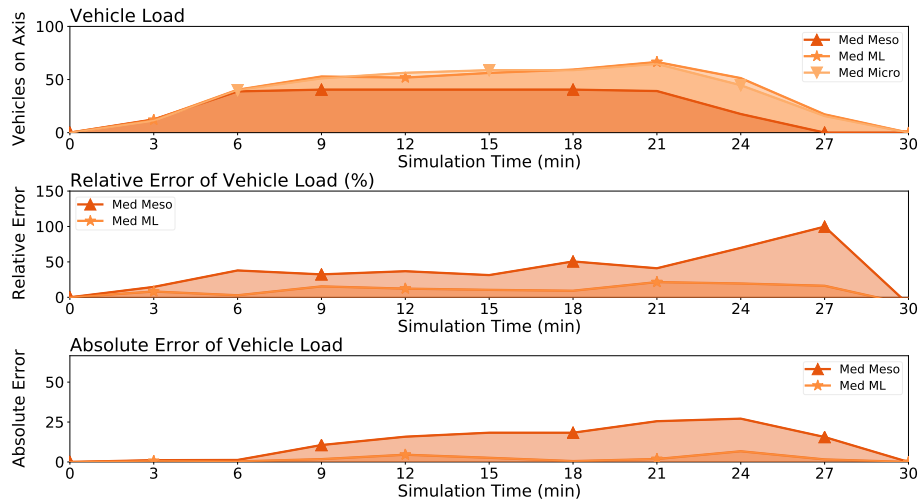


Figure E.2 – Errors of traffic load over time on the south axis for medium traffic.

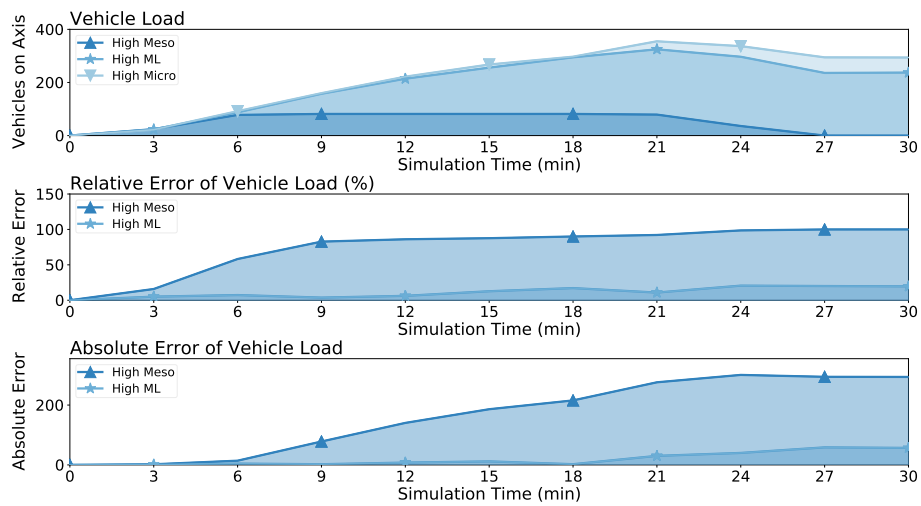


Figure E.3 – Errors of traffic load over time on the south axis for high traffic.

E.3 Study 3

```
1 {
2   "scenarioID": "uc3",
3   "domainReferences": {
4     "traffic": "Town01.reference.xml"
5   },
6   "simulationStart": 0,
7   "simulationEnd": 2100,
8   "execution": {
9     "randomSeed": 23,
10    "constraints": "",
11    "priority": 0,
12    "syncedParticipants": 7
13  },
14  "buildingBlocks": [
15    {
16      "instanceID": "VisumWrapper3",
17      "type": "VisumWrapper",
18      "layer": "macro",
19      "domain": "traffic",
20      "stepLength": 3600000,
21      "parameters": {},
22      "resources": { "Town01_extended_NSLoad_DUE.ver": ↘
23        "Version"
24      },
25      "results": {},
26      "synchronized": true,
27      "isExternal": false,
28      "responsibilities": [ "10014", "10015", "10024", ↘
29        "10025", "10034", "10035", "10044", "10045"
30      ],
31      "observers": []
32    },
33    {
34      "instanceID": "MATSimWrapper2",
35      "type": "MATSimWrapper",
36      "layer": "meso",
37      "domain": "traffic",
38      "stepLength": 1000,
39      "parameters": {},
40      "resources": { "Town01_extended.network.xml": ↘
41        "RoadMap", "Town01.noplans.xml": "Traffic"
42      },
43      "results": {},
44      "synchronized": true,
45      "isExternal": false,
```

```

43         "responsibilities": [ "10011", "10012", "10013", ↵
           "10021", "10022", "10023", "10032", "10033", ↵
           "10042", "10043"
44     ],
45     "observers": []
46 },
47 {
48     "instanceID": "SumoWrapper0",
49     "type": "SumoWrapper",
50     "layer": "micro",
51     "domain": "traffic",
52     "stepLength": 1000,
53     "parameters": { "ghosting": "false"
54 },
55     "resources": { "Town01.vtypes.xml": "Additional", ↵
           "Town01_extended.net.xml": "RoadMap"
56 },
57     "results": {},
58     "synchronized": true,
59     "isExternal": false,
60     "responsibilities": [ "10031", "10041", "110", ↵
           "111", "139", "1513", "195", "2015", "26", ↵
           "313", "43", "520", "60", "714", "77", "811", ↵
           "814", "94"
61 ],
62     "observers": []
63 },
64 {
65     "instanceID": "CarlaWrapper1",
66     "type": "CarlaWrapper",
67     "layer": "submicro",
68     "domain": "traffic",
69     "stepLength": 100,
70     "parameters": {},
71     "resources": { "Town01.xodr": "RoadMap"
72 },
73     "results": {},
74     "synchronized": true,
75     "isExternal": false,
76     "responsibilities": [ "128", "156", "167", "184"
77 ],
78     "observers": []
79 }
80 ],
81 "translators": [
82     {
83         "translatorID": "SubmicroMicroJava",
84         "type": "SubmicroMicroJava",
85         "domain": "traffic",
86         "layerA": "submicro",

```

```

87     "layerB": "micro",
88     "responsibilitiesA": [ "10031", "10041", "110", ↘
      "111", "139", "1513", "195", "2015", "26", ↘
      "313", "43", "520", "60", "714", "77", "811", ↘
      "814", "94"
89   ],
90   "responsibilitiesB": [ "128", "156", "167", "184"
91   ],
92   "parameters": {},
93   "resources": { "Town01_extended.net.xml": ↘
      "MicroNetwork"
94   }
95 },
96 {
97   "translatorID": "MicroMesoJava",
98   "type": "MicroMesoJava",
99   "domain": "traffic",
100  "layerA": "micro",
101  "layerB": "meso",
102  "responsibilitiesA": [ "10011", "10012", "10013", ↘
      "10021", "10022", "10023", "10032", "10033", ↘
      "10042", "10043"
103  ],
104  "responsibilitiesB": [ "10031", "10041", "110", ↘
      "111", "139", "1513", "195", "2015", "26", ↘
      "313", "43", "520", "60", "714", "77", "811", ↘
      "814", "94"
105  ],
106  "parameters": {},
107  "resources": { "Town01_extended.network.xml": ↘
      "MesoNetwork"
108  }
109 },
110 {
111   "translatorID": "MesoMacroJava",
112   "type": "MesoMacroJava",
113   "domain": "traffic",
114   "layerA": "meso",
115   "layerB": "macro",
116   "responsibilitiesA": [ "10014", "10015", "10024", ↘
      "10025", "10034", "10035", "10044", "10045"
117  ],
118  "responsibilitiesB": [ "10011", "10012", "10013", ↘
      "10021", "10022", "10023", "10032", "10033", ↘
      "10042", "10043"
119  ],
120  "parameters": {},
121  "resources": { "Town01_extended.network.xml": ↘
      "MesoNetwork"
122  }

```

```
123     }
124   ],
125   "projectors": []
126 }
```

Listing E.3 – Scenario definition of study 3.

E.4 Study 4

```
1 {
2   "scenarioID": "uc4",
3   "simulationStart": 0,
4   "simulationEnd": 86399,
5   "domainReferences": {
6     "traffic": "A9.reference.xml"
7   },
8   "execution": {
9     "randomSeed": 123,
10    "constraints": "",
11    "priority": 0,
12    "syncedParticipants": 4
13  },
14  "buildingBlocks": [
15    {
16      "instanceID": "provider0",
17      "type": "DataProviderTrafficMacro",
18      "stepLength": 60000,
19      "layer": "macro",
20      "domain": "traffic",
21      "parameters": {},
22      "resources": {},
23      "results": {},
24      "synchronized": true,
25      "isExternal": false,
26      "responsibilities": [],
27      "observers": []
28    },
29    {
30      "instanceID": "SumoWrapper1",
31      "type": "SumoWrapper",
32      "layer": "micro",
33      "domain": "traffic",
34      "stepLength": 1000,
35      "parameters": { "ghosting": "false"
36    },
37    "resources": { "A9.open.net.xml": "RoadMap", ↵
      "add.xml": "Additional"
```



```

38     },
39     "results": {},
40     "synchronized": true,
41     "isExternal": false,
42     "responsibilities": [ "31219920", "31219921", ↵
        "31219922", "1475350055", "1475350057", ↵
        "1475350058", "15848360", "1618588414", ↵
        "1618588415", "2139163501", "21607269", ↵
        "21607745", "2219600162", "2309384888", ↵
        "2309384890", "2309384908", "2309384913", ↵
        "250966173", "250966178", "250966183", ↵
        "254482149", "254482188", "2557520658", ↵
        "2557520659", "26609271", "267532588", ↵
        "267532682", "26869417", "26869418", ↵
        "2800347206", "29747074", "29747075", ↵
        "29747082", "3321250461", "3347605792", ↵
        "3347606294", "3347606296", "3347606302", ↵
        "3347606305", "3347606307", "344222949", ↵
        "350380302", "3779207414", "3779207419", ↵
        "4971737525", "5913529", "5913535", ↵
        "679686945", "747044287", "8562052108"
43     ],
44     "observers": [ { "task": "publish", ↵
        "element": "edge", "filter": ↵
        "observableedge", "period": 1000, ↵
        "trigger": "", "type": "json" }, { ↵
        "task": "publish", "element": ↵
        "edge.vehicles", "filter": "26869418 ↵
        2557520658 3347606302 3347606296 29747082 ↵
        250966183 250966178 3347606305 29747074 ↵
        1618588415 3347606307", "period": 1000, ↵
        "trigger": "", "type": "avro" }
45     ]
46     }
47 ],
48 "translators": [
49     {
50         "translatorID": "MicroMesoJava",
51         "type": "MicroMesoJava",
52         "domain": "traffic",
53         "layerA": "micro",
54         "layerB": "meso",
55         "responsibilitiesA": [],
56         "responsibilitiesB": [ "31219920", "31219921", ↵
            "31219922", "1475350055", "1475350057", ↵
            "1475350058", "15848360", "1618588414", ↵
            "1618588415", "2139163501", "21607269", ↵
            "21607745", "2219600162", "2309384888", ↵
            "2309384890", "2309384908", "2309384913", ↵
            "250966173", "250966178", "250966183", ↵

```

```

        "254482149", "254482188", "2557520658", ↘
        "2557520659", "26609271", "267532588", ↘
        "267532682", "26869417", "26869418", ↘
        "2800347206", "29747074", "29747075", ↘
        "29747082", "3321250461", "3347605792", ↘
        "3347606294", "3347606296", "3347606302", ↘
        "3347606305", "3347606307", "344222949", ↘
        "350380302", "3779207414", "3779207419", ↘
        "4971737525", "5913529", "5913535", ↘
        "679686945", "747044287", "8562052108"
57     ],
58     "resources": { "A9.network.xml": "MesoNetwork"
59     },
60     "parameters": {}
61 },
62 {
63     "translatorID": "MesoMacroJava",
64     "type": "MesoMacroJava",
65     "domain": "traffic",
66     "layerA": "meso",
67     "layerB": "macro",
68     "responsibilitiesA": [],
69     "responsibilitiesB": [ "31219920", "31219921", ↘
        "31219922", "1475350055", "1475350057", ↘
        "1475350058", "15848360", "1618588414", ↘
        "1618588415", "2139163501", "21607269", ↘
        "21607745", "2219600162", "2309384888", ↘
        "2309384890", "2309384908", "2309384913", ↘
        "250966173", "250966178", "250966183", ↘
        "254482149", "254482188", "2557520658", ↘
        "2557520659", "26609271", "267532588", ↘
        "267532682", "26869417", "26869418", ↘
        "2800347206", "29747074", "29747075", ↘
        "29747082", "3321250461", "3347605792", ↘
        "3347606294", "3347606296", "3347606302", ↘
        "3347606305", "3347606307", "344222949", ↘
        "350380302", "3779207414", "3779207419", ↘
        "4971737525", "5913529", "5913535", ↘
        "679686945", "747044287", "8562052108"
70     ],
71     "resources": { "A9.network.xml": "MesoNetwork"
72     },
73     "parameters": {}
74 }
75 ],
76 "projectors": []
77 }

```

Listing E.4 – Scenario definition of study 4 (all lanes open).

E.5 Study 5

```
1 {
2   "scenarioID": "uc5",
3   "simulationStart": 0,
4   "simulationEnd": 86399,
5   "domainReferences": {
6     "traffic": "A9.reference.xml",
7     "communication": "net.reference.xml"
8   },
9   "execution": {
10    "randomSeed": 123,
11    "constraints": "",
12    "priority": 0,
13    "syncedParticipants": 2
14  },
15  "buildingBlocks": [
16    {
17      "instanceID": "OMNeTWrapper2",
18      "type": "OMNeTWrapper",
19      "layer": "80211p",
20      "domain": "communication",
21      "stepLength": 1000,
22      "parameters": {
23        "equipped" : "1.0",
24        "beaconing" : "1.0"
25      },
26      "resources": {},
27      "results": {
28        "full" : "results.zip"
29      },
30      "synchronized": true,
31      "isExternal": false,
32      "responsibilities": ["*"],
33      "observers": []
34    }
35  ],
36  "translators": [
37  ],
38  "projectors": [
39    {
40      "projectorID": "proj0",
41      "type": "v2x",
42      "domainA": "traffic",
43      "layerA": "micro",
44      "domainB": "communication",
45      "layerB": "80211p",
46      "resources": {},
47      "parameters": { "sceID": "uc4_all_lanes_open"
```

```
48         }  
49     }  
50 ]  
51 }
```

Listing E.5 – Scenario definition of study 5.

List of Acronyms

ABM	Agent-Based Modeling
ABS	Agent-Based Simulation
ACC	Adaptive Cruise Control
ACL	Access Control List
AEB	Automatic Emergency Braking
BB	Building Block
BOM	Base Object Model
CDF	Cumulative Distribution Function
DDM	Data Distribution Management
DES	Discrete Event Simulation
DS	Distributed Simulation
DUE	Dynamic User Equilibrium
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FOM	Federation Object Model
FQR	Flush Queue Request
GUI	Graphical User Interface
HiL	Hardware-in-the-Loop
HLA	High Level Architecture
IPC	Inter-Process Communication
LBTS	Lower Bound Time Stamp
LP	Logical Process
MFD	Macroscopic Fundamental Diagram
MOM	Management Object Model
MoM	Message-oriented Middleware
MSaaS	Modeling-and-Simulation-as-a-Service
M&S	Modeling and Simulation
NDM	Native Data Model
NMR	Next Message Request

NMRA	Next Message Request Available
OMT	Object Model Template
PAP	Poisson Arrival Process
RO	Receive Order
RPC	Remote Procedure Call
RTI	Runtime Infrastructure
SaaS	Simulation-as-a-Service
SBA	Simulation-based Dynamic Assignment
SCE	Scenario Definition File
SD	System Dynamics
SDM	Simulation Data Management
SLA	Service Level Agreement
SOM	Simulation Object Model
TAG	Time Advance Grant
TAR	Time Advance Request
TARA	Time Advance Request Available
TSO	Timestamp Order
V2X	Vehicle-to-Everything

List of Algorithms

3.1 Synchronization algorithm.	67
4.1 Time advance algorithm.	97
4.2 Scenario loop.	99

List of Figures

2.1	Different layer architectures. Based on [219].	13
2.2	Simulation type taxonomy. Based on [87].	18
2.3	Extended simulation type taxonomy for distributed simulations.	18
2.4	HLA architecture.	23
2.5	FMI modes. Based on [146].	26
2.6	Application-centric and data-centric architectures. Based on [79].	34
3.1	Data-centric architecture.	43
3.2	Similar components can be hidden behind the same layer.	44
3.3	The translation of information flows between submodels is detached into separate reusable components.	45
3.4	Integration of external components.	46
3.5	The lower layer of the system's architecture is developed.	47
3.6	Different link topologies.	48
3.7	The second layer of the system's architecture is developed.	52
3.8	The receiver is responsible for decoding.	54
3.9	The sender is responsible for encoding.	54
3.10	An additional component is causing modifications.	54
3.11	The translation is detached.	55
3.12	There are different types of couplings.	57
3.13	Definition of the domain structure.	58
3.14	Definition of the layer structure. Gray fields are mandatory.	59
3.15	Definition of the component structure.	61
3.16	No global ordering across multiple topics. There are two possible receive orders.	62
3.17	No global ordering across multiple partitions. There are two possible receive orders.	62
3.18	Global ordering within a topic partition.	62

3.19 No deterministic appending order. After the messages are appended to the partition, there is a deterministic consumption order for all consumers.	63
3.20 Pre-processing in a buffer.	63
3.21 Basic synchronization using a dedicated synchronization topic.	65
3.22 Synchronization with the consideration of message delivery.	65
3.23 Epochs allow for quasi-simultaneous events.	66
3.24 The structure of messages.	68
3.25 There are three different logical channels that are also accessible for external stakeholders.	69
3.26 The main simulation loop.	74
3.27 The different types of coupling.	75
3.28 Blue_A and Blue_B model of the base scenario.	75
3.29 Two BBs of the same layer share the work.	76
3.30 A translator provides two individual conversion functions.	77
3.31 Two instances of different layers share the work.	78
3.32 Three examples for projectors that are connecting different layers.	80
3.33 A projector example is ingesting data into another domain.	80
3.34 The application layer is addressed in this section.	81
3.35 The structure of the scenario definition file.	83
3.36 A component diagram of the developed service architecture.	89
3.37 Developed architecture with labeled components.	89
4.1 General management tab.	95
4.2 Designing a scenario.	95
4.3 Running a scenario.	95
4.4 Catalog view.	95
4.5 Main architecture of the JavaBaseWrapper.	96
4.6 Definition of the number domain.	100
4.7 Integer layer definition in the number domain.	100
4.8 Double layer definition in the number domain.	100
4.9 Definition of the color domain.	101
4.10 3-bit RGB layer definition in the color domain.	101
4.11 24-bit RGB layer definition in the color domain.	101
4.12 Generated baseline data.	103
4.13 Merged result and origin of data points for experiment 2.1.	104
4.14 If synchronization is disabled, the results of the distributed run differ from the baseline for experiment 2.2.	105
4.15 Merged result and origin of data points using two BBs with different levels of detail and step lengths.	106

4.16	Cross-domain coupling using a projector. Data from the number domain is affecting the modeled picture.	107
4.17	Measurements for time synchronization steps in ms.	108
5.1	Different traffic modeling paradigms based on [99].	115
5.2	Macroscopic fundamental diagram based on [65].	116
5.3	A car-following model is mainly based on the distance between vehicles.	117
5.4	Different topologies as a result of exemplary partitioning strategies.	125
5.5	Different possibilities for road representations.	127
5.6	Traffic domain definition.	127
5.7	Definition of the macroscopic layer.	128
5.8	Definition of the mesoscopic layer.	129
5.9	Definition of the microscopic layer.	130
5.10	Definition of the submicroscopic layer.	132
5.11	Spawn probabilities and cumulative representation.	134
5.12	The mapping of an input value r to an inter-arrival time i and the outcome of a non-homogeneous PAP.	135
5.13	Class diagram of the VisumWrapper.	137
5.14	Class diagram of the MatsimWrapper.	140
5.15	Class diagram of the SumoWrapper.	140
5.16	Class diagram of the CarlaWrapper.	141
5.17	Communication domain definition.	142
5.18	Definition of the 80211p layer.	143
5.19	Class diagram of the OmnetWrapper.	144
6.1	Line scenario with five connected road stretches.	150
6.2	Distributed micro simulation that produces wrong trajectories due to missing ghosting features.	150
6.3	Distributed micro simulation that produces correct trajectories by using ghosting features.	151
6.4	Different topologies for the distributed simulations.	152
6.5	Scenario runtimes of different topologies.	153
6.6	Time consumption shares per topology.	154
6.7	Used traffic pattern. The purple ellipse shows the bottleneck.	155
6.8	Topology for the multi-level run as defined in Appendix E.2.	155
6.9	Simulated densities with a low traffic volume at 21 min.	156
6.10	Simulated distribution with a low traffic volume at 21 min.	156
6.11	Simulated densities with a medium traffic volume at 21 min.	157
6.12	Simulated distribution with a medium traffic volume at 21 min.	157
6.13	Simulated densities with a high traffic volume at 21 min.	158

6.14	Simulated distribution with a high traffic volume at 21 min.	158
6.15	Traffic load at 21 minutes.	159
6.16	Errors of traffic load at 21 minutes.	159
6.17	Relative traffic densities on two main axes over the simulation time for the different traffic loads.	160
6.18	Error of the whole road network vs. speed-up.	161
6.19	Error of the whole road network vs. computational time.	161
6.20	The extended road network of Town01 with the used responsibility topology.	163
6.21	Traffic flows and speeds on road C3.	164
6.22	Each black line represents a discrete spawn event in the submicro- scopic regime.	164
6.23	Snapshots of the evaluation scenario taken in the three-dimensional world of CARLA.	165
6.24	Exemplary sensors: distance estimation and semantic segmentation.	165
6.25	Lidar scan point clouds from above and from the side.	166
6.26	The road section to be considered is located in the south of Nurem- berg, Germany and stretches from (49.2441, 11.2176) to (49.3634, 11.2024). The traffic is ingested near the <i>start</i> label. The map is based on contents from ©OpenStreetMap [165].	167
6.27	Relative vehicle densities of the three different scenarios.	169
6.28	Average speeds of the three different scenarios.	170
6.29	We use a part of the map from study 4. The area of interest is marked by the red rectangle. The map is based on contents from ©OpenStreetMap [165].	172
6.30	Busy time of the channel as perceived by each vehicle.	172
6.31	Diced backoff slots per vehicle.	173
6.32	Lost packets due to ongoing transmission on signal reception.	173
E.1	Errors of traffic load over time on the south axis for low traffic.	239
E.2	Errors of traffic load over time on the south axis for medium traffic.	240
E.3	Errors of traffic load over time on the south axis for high traffic.	240

List of Listings

5.1	Exemplary Visum COM-commands via JACOB.	138
5.2	Main simulation loop of the MatsimWrapper.	139
A.1	Domain definition.	181
A.2	Layer definition.	183
A.3	Component definition.	187
A.4	Scenario definition.	189
A.5	CtrlMsg.avro definition.	195
A.6	ResourceFile.avro definition.	195
A.7	SyncMsg.avro definition.	196
B.1	Number domain definition.	199
B.2	Integer layer definition.	199
B.3	Double layer definition.	201
B.4	Color domain definition.	202
B.5	RGB3 layer definition.	203
B.6	RGB24 layer definition.	205
B.7	Scenario definition of experiment 1.	206
B.8	Scenario definition of experiment 3.	207
C.1	Traffic domain definition.	211
C.2	Macroscopic layer definition.	212
C.3	Mesosopic layer definition.	214
C.4	Microscopic layer definition.	216
C.5	Submicroscopic layer definition.	221
D.1	Communication domain definition.	227
D.2	80211p layer definition.	227
E.1	Scenario definition of study 1.1 and 1.2.	233
E.2	Scenario definition of study 2 (multi-level topology with high traffic).	236

E.3 Scenario definition of study 3.	241
E.4 Scenario definition of study 4 (all lanes open).	244
E.5 Scenario definition of study 5.	247

List of Tables

3.1	Recent considerations of MoMs in the literature.	49
3.2	Relations between requirements and architectural elements.	90
4.1	Measurements for time synchronization steps in ms.	109
6.1	Median values of scenario run effort in seconds.	153
6.2	Origin-destination matrix. Traffic demand per hour.	164

Bibliography

- [1] T. ABRAHAMSSON, “Estimation of Origin-Destination Matrices Using Traffic Counts - A Literature Survey,” International Institute for Applied Systems Analysis, Austria, IIASA, Laxenburg, Austria, Interim Report, May 1998.
- [2] S. ACHARYA, A. BHARADWAJ, Y. SIMMHAN, A. GOPALAN, P. PARAG, and H. TYAGI, “CORNET: A Co-Simulation Middleware for Robot Networks,” in *2020 International Conference on COMMunication Systems NETWORKS (COMSNETS)*. IEEE, Jan. 2020, pp. 245–251.
- [3] A. AKKERMANN and B. Å. HJØLLO, “Scenario-Based V&V in a Maritime Co-Simulation Framework,” in *2019 Spring Simulation Conference (SpringSim)*. IEEE, Apr. 2019, pp. 1–12.
- [4] K. AL-ZOUBI and G. WAINER, “RISE: A general simulation interoperability middleware container,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 5, pp. 580–594, May 2013.
- [5] M. ALBANO, L. L. FERREIRA, L. M. PINHO, and A. R. ALKHAWAJA, “Message-oriented middleware for smart grids,” *Computer Standards & Interfaces*, vol. 38, pp. 133–143, Feb. 2015.
- [6] S. ALPERS, C. BECKER, A. OBERWEIS, and T. SCHUSTER, “Microservice Based Tool Support for Business Process Modelling,” in *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*. IEEE, Sep. 2015, pp. 71–78.
- [7] D. ALVAREZ-COELLO, D. WILMS, A. BEKAN, and J. MARX GÓMEZ, “Towards a Data-Centric Architecture in the Automotive Industry,” *Procedia Computer Science*, vol. 181, pp. 658–663, 2021.
- [8] P. ALVAREZ LOPEZ, M. BEHRISCH, L. BIEKER-WALZ, J. ERDMANN, Y.-P. FLÖTTERÖD, R. HILBRICH, L. LÜCKEN, J. RUMMEL, P. WAGNER, and E. WIESSNER, “Microscopic Traffic Simulation using SUMO,” in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, Nov. 2018, pp. 2575–2582.

- [9] APACHE, “Apache Kafka,” <https://kafka.apache.org/31/documentation.html>, Mar. 2022.
- [10] F. ARENA and G. PAU, “An Overview of Vehicular Communications,” *Future Internet*, vol. 11, no. 2:27, pp. 1–12, Jan. 2019.
- [11] ASAM, “ASAM OpenDRIVE®,” <https://www.asam.net/standards/detail/opendrive/>, Feb. 2022.
- [12] AVRO, “Welcome to Apache Avro!” <https://avro.apache.org/>, Mar. 2022.
- [13] A. AW and M. RASCLE, “Resurrection of “Second Order” Models of Traffic Flow,” *SIAM Journal on Applied Mathematics*, vol. 60, no. 3, pp. 916–938, Jan. 2000.
- [14] A. AWAD, P. BAZAN, and R. GERMAN, “SGsim: A simulation framework for smart grid applications,” in *2014 IEEE International Energy Conference (ENERGYCON)*. IEEE, May 2014, pp. 730–736.
- [15] M. U. AWAIS, M. CVETKOVIC, and P. PALENSKY, “Hybrid simulation using implicit solver coupling with HLA and FMI,” *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 08, no. 04, pp. 1–21, Dec. 2017.
- [16] M. U. AWAIS, P. PALENSKY, A. ELSHEIKH, E. WIDL, and S. MATTHIAS, “The high level architecture RTI as a master to the functional mock-up interface components,” in *2013 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, Jan. 2013, pp. 315–320.
- [17] K. W. AXHAUSEN and ETH ZÜRICH, *The Multi-Agent Transport Simulation MATSim*, ETH ZÜRICH, A. HORNI, K. NAGEL, and TU BERLIN, Eds. Ubiquity Press, Aug. 2016.
- [18] M. BACIC, “On hardware-in-the-loop simulation,” in *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, Dec. 2005, pp. 3194–3198.
- [19] O. BALCI, G. L. BALL, K. L. MORSE, E. PAGE, M. D. PETTY, A. TOLK, and S. N. VEAUTOUR, “Model Reuse, Composition, and Adaptation,” in *Research Challenges in Modeling and Simulation for Engineering Complex Systems*, ser. Simulation Foundations, Methods and Applications, R. FUJIMOTO, C. BOCK, W. CHEN, E. PAGE, and J. H. PANCHAL, Eds. Cham: Springer International Publishing, 2017, pp. 87–115.
- [20] J. BARCELÓ, Ed., *Fundamentals of Traffic Simulation*, ser. International Series in Operations Research & Management Science. New York, NY: Springer New York, 2010, vol. 145.

- [21] M. BEN-AKIVA, M. BIERLAIRE, H. KOUTSOPOULOS, and R. MISHALANI, "DynaMIT: A simulation-based system for traffic prediction," in *DACCORD Short Term Forecasting Workshop*, 1998, pp. 1–12.
- [22] M. E. BEN-AKIVA and J. L. BOWMAN, "Activity Based Travel Demand Model Systems," in *Equilibrium and Advanced Transportation Modelling*, P. MARCOTTE and S. NGUYEN, Eds. Boston, MA: Springer US, 1998, pp. 27–46.
- [23] P. BENJAMIN, M. PATKI, and R. MAYER, "Using ontologies for simulation modeling," in *Proceedings of the 2006 Winter Simulation Conference*, L. F. PERRONE, F. P. WIELAND, B. G. L. J. LIU, D. M. NICOL, and R. M. FUJIMOTO, Eds. Piscataway, New Jersey: IEEE, 2006, pp. 1151–1159.
- [24] T. BERNERS-LEE, J. HENDLER, and O. LASSILA, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.
- [25] T. BITTERMAN, P. CALYAM, A. BERRYMAN, D. HUDAK, L. LI, A. CHALKER, S. GORDON, D. ZHANG, D. CAI, C. LEE, and R. RAMNATH, "Simulation as a service (SMaaS): A cloud-based framework to support the educational use of scientific software," *Int. J. of Cloud Computing*, vol. 3, pp. 177–190, Jan. 2014.
- [26] G. BLONDET, J. LE DUIGOU, N. BOUDAOU, and B. EYNARD, "Simulation data management for adaptive design of experiments: A litterature review," *Mechanics & Industry*, vol. 16, no. 6, p. 611, 2015.
- [27] P. BOCCIARELLI, A. D'AMBROGIO, A. GIGLIO, and E. PAGLIA, "A microservice-based approach for fine-grained simulation in MSaaS platforms," in *Proc. of the 2019 Summer Simulation Conf.*, ser. SummerSim '19. Society for Computer Simulation Int., Jul. 2019, pp. 1–12.
- [28] BOER and VERBRAECK, "Distributed simulation with COTS simulation packages," in *Proceedings of the 2003 Winter Simulation Conference*, S. E. CHICK, P. J. SANCHEZ, D. FERRIN, and D. J. MORRICE, Eds. Piscataway, New Jersey: IEEE, Dec. 2003, pp. 829–837.
- [29] A. BONDARENKO and K. ZAYTSEV, "Studying systems of open source messaging," *Journal of Theoretical and Applied Information Technology*, vol. 97, no. 19, p. 11, 2019.
- [30] L. BONONI, M. BRACUTO, G. D'ANGELO, and L. DONATIello, "Artis Scalable and Efficient Parallel and Distributed Simulation of Complex, Dynamic and Mobile Systems," in *2005 Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems (FIRB-PERF'05)*. IEEE, Sep. 2005, pp. 136–145.

- [31] L. BONONI, M. BRACUTO, G. D'ANGELO, and L. DONATIELLO, "Artis: Analysis of High Performance Communication and Computation Solutions for Parallel and Distributed Simulation," in *High Performance Computing and Communications*, ser. Lecture Notes in Computer Science, D. HUTCHISON, T. KANADE, J. KITTLER, J. M. KLEINBERG, F. MATTERN, J. C. MITCHELL, M. NAOR, O. NIERSTRASZ, C. PANDU RANGAN, B. STEFFEN, M. SUDAN, D. TERZOPOULOS, D. TYGAR, M. Y. VARDI, G. WEIKUM, L. T. YANG, O. F. RANA, B. DI MARTINO, and J. DONGARRA, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3726, pp. 640–651.
- [32] A. BORSHCHEV and A. FILIPPOV, "From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools," in *Proceedings of the 22nd International Conference of The System Dynamics Society*, Jul. 2004, pp. 1–23.
- [33] R. BOTTURA, D. BABAZADEH, K. ZHU, A. BORGHETTI, L. NORDSTRÖM, and C. A. NUCCI, "SITL and HLA co-simulation platforms: Tools for analysis of the integrated ICT and electric power system," in *Eurocon 2013*. IEEE, Jul. 2013, pp. 918–925.
- [34] Y. BOUANAN, S. GORECKI, J. RIBAUT, G. ZACHAREWICZ, and N. PERRY, "Including in HLA federation functional mockup units for supporting interoperability and reusability in Distributed Simulation," in *Summer Simulation Conference 2018*. SCSi, Jul. 2018.
- [35] S. BOUCHER, A. KALIA, D. G. ANDERSEN, and M. KAMINSKY, "Putting the "Micro" Back in Microservice," in *2018 USENIX Annual Technical Conference*, 2018, pp. 645–650.
- [36] G. E. BOX, "Robustness in the strategy of scientific model building," in *Robustness in Statistics*. Elsevier, 1979, pp. 201–236.
- [37] Q. BRAGARD, A. VENTRESQUE, and L. MURPHY, "dSUMO: Towards a distributed SUMO," in *1st SUMO User Conference 2013*, May 2013, pp. 132–146.
- [38] S. C. BRAILSFORD, T. EL DABI, M. KUNC, N. MUSTAFEE, and A. F. OSORIO, "Hybrid simulation modelling in operational research: A state-of-the-art review," *European Journal of Operational Research*, vol. 278, no. 3, pp. 721–737, Nov. 2019.
- [39] T. BRAY, "The JavaScript Object Notation (JSON) Data Interchange Format," Internet Engineering Task Force, Request for Comments RFC 8259, Dec. 2017.
- [40] R. E. BRYANT, "Simulation of Packet Communication Architecture Computer Systems." Massachusetts Institute of Technology, Tech. Rep., Nov. 1977.

- [41] W. BURGHOUT, H. N. KOUTSOPOULOS, and I. ANDRÉASSON, “Hybrid Mesoscopic–Microscopic Traffic Simulation,” *Transportation Research Record*, vol. 1934, no. 1, pp. 218–225, Jan. 2005.
- [42] W. BURGHOUT, TEKNISKA HÖGSKOLAN I STOCKHOLM, and INSTITUTIONEN FÖR INFRASTRUKTUR, “Hybrid microscopic-mesoscopic traffic simulation,” Ph.D. dissertation, Dept. of Infrastructure, Royal Institute of Technology, Stockholm, 2004.
- [43] A. H. BUSS, “Component-based simulation modeling,” in *Proceedings of the 2000 Winter Simulation Conference*, J. A. JOINES, R. R. BARTON, K. KANG, and P. A. FISHWICK, Eds. Piscataway, New Jersey: IEEE, Dec. 2000, pp. 964–971.
- [44] F. CAGLAR, S. SHEKHAR, A. GOKHALE, S. BASU, T. RAFI, J. KINNEBREW, and G. BISWAS, “Cloud-hosted simulation-as-a-service for high school STEM education,” *Simulation Modelling Practice and Theory*, vol. 58, pp. 255–273, Nov. 2015.
- [45] CAMBRIDGE UNIVERSITY PRESS, “Domain,” <https://dictionary.cambridge.org/us/dictionary/english/domain>, Aug. 2022.
- [46] B. CAMUS, T. PARIS, J. VAUBOURG, Y. PRESSE, C. BOURJOT, L. CIARLETTA, and V. CHEVRIER, “MECSYCO: A Multi-agent DEVS Wrapping Platform for the Co-simulation of Complex Systems,” Université de Lorraine, Tech. Rep., 2016.
- [47] CARLA, “3rd- Maps and navigation - CARLA Simulator,” https://carla.readthedocs.io/en/latest/core_map/, Feb. 2022.
- [48] CARLA, “CARLA,” <http://carla.org/>, Feb. 2022.
- [49] CARLA, “Introduction - CARLA Simulator,” https://carla.readthedocs.io/en/latest/start_introduction/, Feb. 2022.
- [50] J. CARSON, “Introduction to modeling and simulation,” in *Proceedings of the Winter Simulation Conference, 2005*. IEEE, Dec. 2005, pp. 1–8.
- [51] E. CASCETTA, “Estimation of trip matrices from traffic counts and survey data: A generalized least squares estimator,” *Transportation Research Part B: Methodological*, vol. 18, no. 4, pp. 289–299, Aug. 1984.
- [52] E. CAYIRCI, “Modeling and simulation as a cloud service: A survey,” in *2013 Winter Simulations Conference (WSC)*. IEEE, Dec. 2013, pp. 389–400.

- [53] D. ÇETINKAYA and H. OĞUZTÜZÜN, “A metamodel for the HLA object model,” in *Proceedings of the 20th European Conference on Modeling and Simulation*, W. BORUTZKY, A. ORSONI, and R. ZOBEL, Eds., Germany, 2006, pp. 207–213.
- [54] K. M. CHANDY and J. MISRA, “Asynchronous distributed simulation via a sequence of parallel computations,” *Communications of the ACM*, vol. 24, no. 4, pp. 198–206, Apr. 1981.
- [55] F. S. CHAPIN, *Human Activity Patterns in the City: Things People Do in Time and in Space*. New York: Wiley, 1974.
- [56] H. CHEONG and A. BUTSCHER, “Physics-based simulation ontology: An ontology to support modelling and reuse of data for physics-based simulation,” *Journal of Engineering Design*, vol. 30, no. 10-12, pp. 655–687, Dec. 2019.
- [57] M. CIAVOTTA, M. ALGE, S. MENATO, D. ROVERE, and P. PEDRAZZOLI, “A Microservice-based Middleware for the Digital Factory,” *Procedia Manufacturing*, vol. 11, pp. 931–938, Jan. 2017.
- [58] S. CIRACI, J. DAILY, J. FULLER, A. FISHER, L. MARINOVICI, and K. AGARWAL, “FNCS: A framework for power system and communication networks co-simulation,” in *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, ser. DEVS ’14. Society for Computer Simulation International, Apr. 2014, pp. 1–8.
- [59] CONFLUENT, INC., “Kafka Java Client | Confluent Documentation,” <https://docs.confluent.io/clients-kafka-java/current/overview.html>, Apr. 2022.
- [60] CONFLUENT, INC., “ksqlDB,” <https://github.com/confluentinc/ksql>, Mar. 2022.
- [61] CONFLUENT, INC., “ksqlDB: The database purpose-built for stream processing applications.” <https://ksqldb.io/distributions.html>, Mar. 2022.
- [62] CONFLUENT, INC., “Schema Registry,” <https://github.com/confluentinc/schema-registry>, Mar. 2022.
- [63] F. CREMONA, M. LOHSTROH, S. TRIPAKIS, C. BROOKS, and E. A. LEE, “FIDE: An FMI integrated development environment,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, Apr. 2016, pp. 1759–1766.
- [64] H. G. CRIŞAN and N. FILIP, “Traffic Modeling Aspects Using Visum Software and Effects on the Traffic Optimization,” in *Proceedings of the European Automotive Congress EAEC-ESFA 2015*, C. ANDREESCU and A. CLENCI, Eds. Cham: Springer International Publishing, 2016, pp. 495–506.

- [65] C. F. DAGANZO and N. GEROLIMINIS, “An analytical approximation for the macroscopic fundamental diagram of urban traffic,” *Transportation Research Part B: Methodological*, vol. 42, no. 9, pp. 771–781, Nov. 2008.
- [66] J. S. DAHMANN, R. M. FUJIMOTO, and R. M. WEATHERLY, “The Department of Defense High Level Architecture,” in *Proceedings of the 29th Conference on Winter Simulation - WSC '97*. ACM Press, 1997, pp. 142–149.
- [67] O. DALLE, “On reproducibility and traceability of simulations,” in *Proceedings of the 2012 Winter Simulation Conference (WSC)*. IEEE, Dec. 2012, pp. 1–12.
- [68] A. D’AMBROGIO, P. BOCCIARELLI, and A. MASTROMATTEI, “A PaaS-based framework for automated performance analysis of service-oriented systems,” in *Proc. of the 2016 Winter Simulation Conference*. IEEE, Dec. 2016, pp. 931–942.
- [69] G. D’ANGELO and M. BRACUTO, “Distributed simulation of large-scale and detailed models,” *International Journal of Simulation and Process Modelling*, vol. 5, no. 2, p. 120, 2009.
- [70] M. DEVECI, S. RAJAMANICKAM, K. D. DEVINE, and Ü. V. ÇATALYÜREK, “Multi-Jagged: A Scalable Parallel Spatial Partitioning Algorithm,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 803–817, Mar. 2016.
- [71] S. Y. DIALLO, J. J. PADILLA, R. GORE, H. HERENCIA-ZAPANA, and A. TOLK, “Toward a formalism of modeling and simulation using model theory,” *Complexity*, vol. 19, no. 3, pp. 56–63, 2014.
- [72] DLR, “SUMO Road Networks - SUMO Documentation,” https://sumo.dlr.de/docs/Networks/SUMO_Road_Networks.html, Feb. 2022.
- [73] P. DOBBELAERE and K. S. ESMALI, “Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper,” in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 227–238.
- [74] A. DOSOVITSKIY, G. ROS, F. CODEVILLA, A. LOPEZ, and V. KOLTUN, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, Nov. 2017, pp. 1–16.
- [75] ECLIPSE, “Eclipse SUMO - Simulation of Urban MObility,” <https://www.eclipse.org/sumo/>, Feb. 2022.
- [76] M. EDENHILL, “Librdkafka - the Apache Kafka C/C++ client library,” <https://github.com/edenhill/librdkafka>, Apr. 2022.

- [77] P. M. EJERCITO, K. G. E. NEBRIJA, R. P. FERIA, and L. L. LARA-FIGUEROA, “Traffic simulation software review,” in *2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)*, Aug. 2017, pp. 1–4.
- [78] J. ÉVORA GÓMEZ, J. J. HERNÁNDEZ CABRERA, J.-P. TAVELLA, S. VIALLE, E. KREMER, and L. FRAYSSINET, “Daccosim NG: Co-simulation made simpler and faster,” in *The 13th International Modelica Conference*, Mar. 2019, pp. 785–794.
- [79] P. FELDMANN, J. WISE, and D. FORRESTER, “Enterprise Applications Program Review,” <https://slideplayer.com/slide/12748063/>, Aug. 2022.
- [80] J. FITZGIBBONS, R. FUJIMOTO, D. FELLIG, S. KLEBAN, and A. SCHOLAND, “IDSim: An extensible framework for Interoperable Distributed Simulation,” in *Proceedings. IEEE International Conference on Web Services, 2004.* IEEE, Jul. 2004, pp. 532–539.
- [81] N. FORMOSA, M. QUDDUS, A. PAPADOULIS, and A. TIMMIS, “Validating a Traffic Conflict Prediction Technique for Motorways Using a Simulation Approach,” *Sensors*, vol. 22, no. 2, p. 566, Jan. 2022.
- [82] FRAUNHOFER FOKUS, “Eclipse MOSAIC,” <https://www.eclipse.org/mosaic/about/>, Jan. 2019.
- [83] J. FREEMAN, “JACOB Release Notes,” <https://github.com/freemansoft/jacob-project>, Mar. 2022.
- [84] G. FU, Y. ZHANG, and G. YU, “A Fair Comparison of Message Queuing Systems,” *IEEE Access*, vol. 9, pp. 421–432, 2021.
- [85] R. M. FUJIMOTO, “Distributed simulation systems,” in *Proceedings of the 2003 Winter Simulation Conference.* IEEE, Dec. 2003, pp. 124–134.
- [86] R. M. FUJIMOTO, *Parallel and Distributed Simulation Systems.* Wiley, 2000.
- [87] M. GEIMER, T. KRÜGER, and LINSEL. P, “Co-Simulation, gekoppelte Simulation oder Simulatorkopplung? Ein Versuch der Begriffsvereinheitlichung,” *O+P Ölhydraulik und Pneumatik*, vol. 50, no. 11-12, pp. 572–576, 2006.
- [88] H. GEORG, S. C. MÜLLER, N. DORSCH, C. REHTANZ, and C. WIETFELD, “INSPIRE: Integrated co-simulation of power and ICT systems for real-time evaluation,” in *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm).* IEEE, Oct. 2013, pp. 576–581.
- [89] D. L. GERLOUGH and M. J. HUBER, *Traffic Flow Theory: A Monograph*, ser. Special Report. Washington: Transportation Research Board, National Research Council, 1975, no. 165.

- [90] P. G. GIPPS, “A behavioural car-following model for computer simulation,” *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, Apr. 1981.
- [91] C. GOMES, C. THULE, D. BROMAN, P. G. LARSEN, and H. VANGHELuwe, “Co-Simulation: A Survey,” *ACM Computing Surveys*, vol. 51, no. 3, pp. 1–156, May 2018.
- [92] GOOGLE DEVELOPERS, “Protocol Buffers,” <https://developers.google.com/protocol-buffers>, Mar. 2022.
- [93] L. GRANOWETTER, “RTI Interoperability Issues – API Standards, Wire Standards, and RTI Bridges,” in *Proceedings of the 2003 European Simulation Interoperability Workshop*. SISO, Jun. 2003, pp. 1–7.
- [94] M. GÜTLEIN, “daceDS GitHub Repository,” <https://github.com/guetlein/daceDS>, Aug. 2022.
- [95] M. GÜTLEIN, W. BARON, C. RENNER, and A. DJANATLIEV, “Performance Evaluation of HLA RTI Implementations,” in *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, Sep. 2020, pp. 1–8.
- [96] M. GÜTLEIN and A. DJANATLIEV, “Coupled Traffic Simulation by Detached Translation Federates: An HLA-Based Approach,” in *2019 Winter Simulation Conference (WSC)*. IEEE, Dec. 2019, pp. 1378–1389.
- [97] M. GÜTLEIN and A. DJANATLIEV, “Modeling and Simulation as a Service using Apache Kafka;,” in *Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SCITEPRESS - Science and Technology Publications, Jul. 2020, pp. 171–180.
- [98] M. GÜTLEIN and A. DJANATLIEV, “On-demand Simulation of Future Mobility Based on Apache Kafka,” in *Simulation and Modeling Methodologies, Technologies and Applications*, ser. Lecture Notes in Networks and Systems, M. S. OBAIDAT, T. OREN, and F. D. RANGO, Eds. Cham: Springer International Publishing, 2022, pp. 18–41.
- [99] M. GÜTLEIN, R. GERMAN, and A. DJANATLIEV, “Towards a Hybrid Co-simulation Framework: HLA-Based coupling of MATSim and SUMO,” in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, Oct. 2018, pp. 1–9.
- [100] M. GÜTLEIN, R. GERMAN, and A. DJANATLIEV, “Hide Your Model! Layer Abstractions for Data-Driven Co-Simulations,” in *2021 Winter Simulation Conference (WSC)*. IEEE, Dec. 2021, pp. 1–12.

- [101] T. HAGERSTAND, “What about people in spatial science?” *Regional Science Association*, vol. 24, pp. 7–21, 1970.
- [102] M. HAKLAY and P. WEBER, “OpenStreetMap: User-Generated Street Maps,” *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct. 2008.
- [103] M. HANAI, T. SUZUMURA, A. VENTRESQUE, and K. SHUDO, “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*, 2014, pp. 130–137.
- [104] J. HÄRRI, M. KILLAT, T. TIELERT, J. MITTAG, and H. HARTENSTEIN, “DEMO: Simulation-as-a-service for ITS applications,” in *2010 IEEE 71st Vehicular Tech. Conf.*, 2010, pp. 1–2.
- [105] W. HASSELBRING and G. STEINACKER, “Microservice Architectures for Scalability, Agility and Reliability in E-Commerce,” in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Apr. 2017, pp. 243–246.
- [106] L. I. HATLEDAL, H. ZHANG, A. STYVE, and G. HOVLAND, “FMU-proxy : A Framework for Distributed Access to Functional Mock-up Units,” in *Proceedings of the 13 Th International Modelica Conference*. Linköping University Electronic Press, Mar. 2019, pp. 79–86.
- [107] R. G. HEGDE, “Low Latency Message Brokers,” *International Research Journal of Engineering and Technology*, vol. 07, no. 05, pp. 2731–2738, 2020.
- [108] D. HELBING, “Theoretical foundation of macroscopic traffic models,” *Physica A: Statistical Mechanics and its Applications*, vol. 219, no. 3-4, pp. 375–390, Oct. 1995.
- [109] S. HENSEL, M. GRAUBE, L. URBAS, T. HEINZERLING, and M. OPPELT, “Co-simulation with OPC UA,” in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. IEEE, Jul. 2016, pp. 20–25.
- [110] J. J. HERNANDEZ, J. EVORA, J.-P. TAVELLA, and B. G. MONGE, “Semantic interoperability in co-simulation: Use cases and requirements,” in *30th European Simulation and Modelling Conference*. Las Palmas de Gran Canaria, Spain: EUROSIS, Oct. 2016, pp. 5–9.
- [111] P. HEYKEN SOARES, L. AHMED, Y. MAO, and C. L. MUMFORD, “Public transport network optimisation in PTV Visum using selection hyper-heuristics,” *Public Transport*, vol. 13, no. 1, pp. 163–196, Mar. 2021.

- [112] HIVEMQ GMBH, “HiveMQ - Enterprise ready MQTT to move your IoT data,” <https://www.hivemq.com/>, Aug. 2022.
- [113] A. HORNI, K. NAGEL, and K. W. AXHAUSEN, “Introducing matsim,” in *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press, 2016.
- [114] Q. HUANG, T. E. MCDERMOTT, Y. TANG, A. MAKHMALBAF, D. J. HAMMERSTROM, A. R. FISHER, L. D. MARINOVICI, and T. HARDY, “TESP: Simulation-Based Valuation of Transactive Energy Systems,” *IEEE Transactions on Power Systems*, vol. 34, no. 5, pp. 4138–4147, Sep. 2019.
- [115] W. HUANG, A. GANJALI, B. H. KIM, S. OH, and D. LIE, “The State of Public Infrastructure-as-a-Service Cloud Security,” *ACM Computing Surveys*, vol. 47, no. 4, pp. 1–31, Jul. 2015.
- [116] X. HUANG and O.-S. KWON, “UT-SIM: A Generalized Numerical/Experimental Distributed Simulation Framework,” *Journal of Earthquake Engineering*, vol. 24, no. 4, pp. 682–703, 2018.
- [117] R. IBRAHIM, N. REGNARD, P. L’ECUYER, and H. SHEN, “On the modeling and forecasting of call center arrivals,” in *Proceedings of the 2012 Winter Simulation Conference (WSC)*. IEEE, Dec. 2012, pp. 1–12.
- [118] IEEE, “IEEE standard for modeling and simulation (m/s) high level architecture (HLA)– framework and rules - redline,” *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000) - Redline*, pp. 1–38, 2010.
- [119] R. JAYAKRISHNAN, H. S. MAHMASSANI, and T.-Y. HU, “An evaluation tool for advanced traffic information and management systems in urban networks,” *Transportation Research Part C: Emerging Technologies*, vol. 2, no. 3, pp. 129–147, Sep. 1994.
- [120] D. JEFFERSON, B. BECKMAN, F. WIELAND, L. BLUME, and M. DILORETO, “Time warp operating system,” in *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, ser. SOSP ’87. New York, NY, USA: Association for Computing Machinery, Nov. 1987, pp. 77–93.
- [121] J. A. JOINES and S. D. ROBERTS, “Simulation in an object-oriented world,” in *Proceedings of the 1999 Winter Simulation Conference*, P. A. FARRINGTON, H. B. NEMBHARD, D. T. STURROCK, and G. W. EVANS, Eds., vol. 1. Piscataway, New Jersey: IEEE, 1999, pp. 132–140.
- [122] KAFKAJS, “KafkaJS · KafkaJS, a modern Apache Kafka client for Node.js,” <https://kafka.js.org/>, Apr. 2022.

- [123] A. KESTING, M. TREIBER, and D. HELBING, “Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4585–4605, Oct. 2010.
- [124] A. A. KHALEQ and I. RA, “Cloud-Based Disaster Management as a Service: A Microservice Approach for Hurricane Twitter Data Analysis,” in *2018 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE, Oct. 2018, pp. 1–8.
- [125] K. KHANDA, D. SALIKHOV, K. GUSMANOV, M. MAZZARA, and N. MAVRIDIS, “Microservice-Based IoT for Smart Buildings,” in *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, Mar. 2017, pp. 302–308.
- [126] J. C. KIRCHHOF, E. KUSMENKO, B. RUMPE, and H. ZHANG, “Simulation as a service for cooperative vehicles,” in *2019 ACM/IEEE 22nd Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2019, pp. 28–37.
- [127] J. P. KLEIJNEN, “Verification and validation of simulation models,” *European journal of operational research*, vol. 82, no. 1, pp. 145–162, 1995.
- [128] M. KRAMMER, M. BENEDIKT, T. BLOCHWITZ, K. ALEKEISH, N. AMRINGER, C. KATER, S. MATERNE, R. RUVALCABA, K. SCHUCH, J. ZEHETNER, M. DAMM-NORWIG, V. SCHREIBER, N. NAGARAJAN, I. CORRAL, T. SPARBER, S. KLEIN, and J. ANDERT, “The Distributed Co-simulation Protocol for the integration of real-time systems and simulation environments,” in *SummerSim '18: Proceedings of the 50th Computer Simulation Conference*. Society for Computer Simulation International, Jul. 2018, pp. 1–14.
- [129] M. KRUMNOW, “SUMO as a Service—Building up a web service to interact with SUMO,” in *Simulation of Urban MObility User Conf.*, 2013, pp. 62–70.
- [130] A. KRYLOVSKIY, M. JAHN, and E. PATTI, “Designing a Smart City Internet of Things Platform with Microservice Architecture,” in *2015 3rd International Conference on Future Internet of Things and Cloud*. IEEE, Aug. 2015, pp. 25–30.
- [131] M. KUDELSKI, L. M. GAMBARDELLA, and G. DI CARO, “RoboNetSim: An integrated framework for multi-robot and network simulation,” *Robotics and Autonomous Systems*, vol. 61, pp. 483–496, May 2013.

- [132] A. M. LAW, *Simulation Modeling and Analysis*, 5th ed., ser. McGraw-Hill Series in Industrial Engineering and Management Science. Dubuque: McGraw-Hill Education, 2013.
- [133] D. R. LEONARD, P. GOWER, and N. B. TAYLOR, "CONTRAM: Structure of the Model," *TRRL RESEARCH REPORT*, no. 78, 1989.
- [134] C. LESTER, C. A. YATES, M. B. GILES, and R. E. BAKER, "An adaptive multi-level simulation algorithm for stochastic biological systems," *The Journal of Chemical Physics*, vol. 142, no. 2:024113, pp. 1–23, Jan. 2015.
- [135] M. J. LIGHTHILL and G. B. WHITHAM, "On Kinematic Waves. I. Flood Movement in Long Rivers," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 229, no. 1178, pp. 281–316, 1955.
- [136] H. LIN, S. S. VEDA, S. S. SHUKLA, L. MILI, and J. THORP, "GECO: Global Event-Driven Co-Simulation Framework for Interconnected Power System and Communication Network," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1444–1456, Sep. 2012.
- [137] A. M. DEL ESPOSTE, F. KON, F. M. COSTA, and N. LAGO, "InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities:," in *Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems*. SCITEPRESS - Science and Technology Publications, 2017, pp. 35–46.
- [138] C. MACAL and M. NORTH, "Tutorial on agent-based modeling and simulation," in *Proceedings of the Winter Simulation Conference, 2005*. IEEE, Dec. 2005, pp. 1–14.
- [139] MÄK, "MÄK RTI," <https://www.mak.com/products/link/mak-rti>, Aug. 2022.
- [140] A. MALIK, A. PARK, and R. FUJIMOTO, "Optimistic Synchronization of Parallel Simulations in Cloud Computing Environments," in *2009 IEEE International Conference on Cloud Computing*. IEEE, 2009, pp. 49–56.
- [141] E. A. MARCONATO, M. RODRIGUES, R. D. M. PIRES, D. F. PIGATTO, L. C. Q. FILHO, A. R. PINTO, and K. R. L. J. C. BRANCO, "AVENS - A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System," in *Hawaii International Conference on System Sciences*. ScholarSpace, Jan. 2017.
- [142] MATSIM, "MATSim.org," <https://matsim.org/>, Feb. 2022.
- [143] D. MCGRATH, A. HUNT, and M. BATES, "A simple distributed simulation architecture for emergency response exercises," in *Ninth IEEE International*

- Symposium on Distributed Simulation and Real-Time Applications*. IEEE, Oct. 2005, pp. 221–226.
- [144] B. MEYER, “Reusability: The case for object-oriented design,” *IEEE Software*, vol. 4, no. 2, p. 50, 1987.
- [145] MODELICA, “FMI 2.0.1 specification,” Oct. 2019.
- [146] MODELICA ASSOCIATION, “Modelica Newsletter 2014-3 — Modelica Association,” <https://modelica.org/publications/newsletters/2014-3>, Aug. 2022.
- [147] B. MÖLLER, F. ANTELIUS, and M. KARLSSON, “Towards a Standardized Federate Protocol for HLA 4,” in *Proceedings of 2018 Winter Simulation Interoperability Workshop*. SISO, Jan. 2018, pp. 1–9.
- [148] B. MÖLLER and M. KARLSSON, “New Object Modeling Opportunities in HLA 4,” in *Proceedings of 2019 Winter Simulation Innovation Workshop*. SISO, Feb. 2019, pp. 1–9.
- [149] B. MÖLLER, M. KARLSSON, R. HERZOG, and D. WOOD, “Security in Simulation – New Authorization Opportunities in HLA 4,” in *Proceedings of 2021 Virtual Simulation Innovation Workshop*. SISO, 2021, pp. 1–10.
- [150] B. MÖLLER and L. OLSSON, “Practical Experiences from HLA 1.3 to HLA IEEE 1516 Interoperability,” in *2004 European Simulation Interoperability Workshop*. SISO, 2004, pp. 1–7.
- [151] A. L. MOLTHAN, J. L. CASE, J. VENNER, R. SCHROEDER, M. R. CHECCHI, B. T. ZAVODSKY, A. LIMAYE, and R. G. O’BRIEN, “Clouds in the cloud: Weather forecasts and applications within cloud computing environments,” *Bulletin of the American Meteorological Society*, vol. 96, no. 8, pp. 1369–1379, 2015.
- [152] F. MORADI, P. NORDVALLER, and R. AYANI, “Simulation model composition using BOMs,” in *Proceedings of the 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, E. ALBA, S. J. TURNER, D. ROBERTS, and S. J. E. TAYLOR, Eds. IEEE, 2006, pp. 242–252.
- [153] Y. MOTIE, A. NKETSA, and P. TRUILLET, “Cosimate: A co-simulation framework interoperability for Neo-campus project,” in *31st European Simulation and Modelling Conference*. EUROSIS, 2017, pp. 1–7.
- [154] L. MULUGETA, A. DRACH, A. ERDEMIR, C. A. HUNT, M. HORNER, J. P. KU, J. G. MYERS JR., R. VADIGEPALLI, and W. W. LYTTON, “Credibility, Replicability, and Reproducibility in Simulation for Biomedicine and Clinical Applications in Neuroscience,” *Frontiers in Neuroinformatics*, vol. 12, no. 18, pp. 1–16, 2018.

- [155] K. NAGEL and M. SCHRECKENBERG, "A cellular automaton model for freeway traffic," *Journal de Physique I*, vol. 2, no. 12, pp. 2221–2229, Dec. 1992.
- [156] C. NAN and I. EUSGELD, "Adopting HLA standard for interdependency study," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 149–159, 2011.
- [157] R. E. NANCE, "A tutorial view of simulation model development," *SIGSIM Simul. Dig.*, vol. 15, no. 2, pp. 16–22, Apr. 1984.
- [158] N. NANNONI, "Message-oriented Middleware for Scalable Data Analytics Architectures," KTH Stockholm, Master's Thesis, 2015.
- [159] NATIONAL RESEARCH COUNCIL (U.S.), Ed., *Highway Capacity Manual*. Washington, D.C: Transportation Research Board, National Research Council, 2000.
- [160] C. NOBIS and T. KUHNIMHOF, "Mobilität in Deutschland (MiD): Ergebnisbericht 2018," http://www.mobilitaet-indeutschland.de/pdf/MiD2017_-_Ergebnisbericht.pdf, Aug. 2022.
- [161] T. NOUIDUI, M. WETTER, and W. ZUO, "Functional mock-up unit for co-simulation import in EnergyPlus," *Journal of Building Performance Simulation*, vol. 7, no. 3, pp. 192–202, May 2014.
- [162] T. S. NOUIDUI, J. COIGNARD, C. GEHBAUER, M. WETTER, J.-Y. JOO, and E. VRETTOS, "CyDER – an FMI-based co-simulation platform for distributed energy resources," *Journal of Building Performance Simulation*, vol. 12, no. 5, pp. 566–579, Sep. 2019.
- [163] E. NOULARD, J.-Y. ROUSSELOT, and P. SIRON, "CERTI, an open source RTI, why and how," in *Spring Simulation Interoperability Workshop*. SISO, Mar. 2009, pp. 1–11.
- [164] J. O'DONNELL, R. SEE, C. ROSE, T. MAILE, V. BAZJANAC, and P. HAVES, "Sim-Model: A domain data model for whole building energy simulation," in *IBPSA Building Simulation 2011*, Jan. 2011, pp. 1–8.
- [165] OPENSTREETMAP, "OpenStreetMap," <https://www.openstreetmap.org/>, Feb. 2022.
- [166] M. OPPELT, G. WOLF, and L. URBAS, "Capability-analysis of co-simulation approaches for process industries," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, Sep. 2014, pp. 1–4.
- [167] PADS BOLOGNA, "ARTIS," <https://pads.cs.unibo.it/doku.php?id=pads:artis>, Dec. 2021.

- [168] F. PALLONETTO, E. MANGINA, F. MILANO, and D. P. FINN, “SimApi, a smartgrid co-simulation software platform for benchmarking building control algorithms,” *SoftwareX*, vol. 9, pp. 271–281, Jan. 2019.
- [169] B. PALMINTIER, D. KRISHNAMURTHY, P. TOP, S. SMITH, J. DAILY, and J. FULLER, “Helics: Design of the HELICS high-performance transmission-distribution-communication-market co-simulation framework,” in *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, Apr. 2017, pp. 1–6.
- [170] G. PARDO-CASTELLOTE, B. FARABAUGH, and R. WARREN, “An Introduction to DDS and Data-Centric Communications,” *Real-Time Innovations*, pp. 1–16, 2005.
- [171] K. PAWLIKOWSKI, H.-D. JEONG, and J.-S. LEE, “On credibility of simulation studies of telecommunication networks,” *IEEE Communications Magazine*, vol. 40, no. 1, pp. 132–139, Jan. 2002.
- [172] F. PERABO, D. PARK, M. K. ZADEH, Ø. SMOGELI, and L. JAMT, “OSP: Digital Twin Modelling of Ship Power and Propulsion Systems: Application of the Open Simulation Platform (OSP),” in *2020 IEEE 29th International Symposium on Industrial Electronics (ISIE)*. IEEE, Jun. 2020, pp. 1265–1270.
- [173] A. PIATER, T. B. IONESCU, and W. SCHEUERMANN, “A Distributed Simulation Framework for Mission Critical Systems in Nuclear Engineering and Radiological Protection,” *International Journal of Computers, Communications & Control*, vol. 3, pp. 448–453, 2008.
- [174] PITCH, “Pitch pRTI USER’S GUIDEv 5.4,” 2019.
- [175] PORTICO, “poRTIco project,” <http://porticoproject.org/>, Aug. 2022.
- [176] A. POS, P. BORST, J. TOP, and H. AKKERMANS, “Reusability of simulation models,” *Knowledge-Based Systems*, vol. 9, no. 2, pp. 119–125, Apr. 1996.
- [177] D. POWERS and A. DAVID, “Kafka-python — kafka-python 2.0.2-dev documentation,” <https://kafka-python.readthedocs.io/en/master/>, Apr. 2022.
- [178] T. PREISLER, T. DETHLEFS, and W. RENZ, “Simulation as a service: A design approach for large-scale energy network simulations,” in *2015 Federated Conf. on Computer Science and Information Systems (FedCSIS)*. IEEE, 2015, pp. 1765–1772.
- [179] S. PROFANTER, A. TEKAT, K. DOROFEEV, M. RICKERT, and A. KNOLL, “OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0

- Protocols,” in *2019 IEEE International Conference on Industrial Technology (ICIT)*. Melbourne, Australia: IEEE, Feb. 2019, pp. 955–962.
- [180] F. PROVOST and T. FAWCETT, “Data Science and its Relationship to Big Data and Data-Driven Decision Making,” *Big Data*, vol. 1, no. 1, pp. 51–59, Mar. 2013.
- [181] PTV GMBH, “PTV Visum,” <https://www.ptvgroup.com/de/loesungen/produkte/ptv-visum/>, Nov. 2021.
- [182] PTV GMBH, “PTV Visum 2022 Manual,” 2022.
- [183] T. RATHNAM and C. J. PAREDIS, “Developing federation object models using ontologies,” in *Proceedings of the 2004 Winter Simulation Conference*, R. G. INGALLS, M. D. ROSSETTI, J. S. SMITH, and B. A. PETERS, Eds. Piscataway, New Jersey: IEEE, 2004, pp. 1054–1062.
- [184] P. I. RICHARDS, “Shock Waves on the Highway,” *Operations Research*, vol. 4, no. 1, pp. 42–51, Feb. 1956.
- [185] R. RIEBL, H.-J. GÜNTHER, C. FACCHI, and L. WOLF, “Artery: Extending Veins for VANET applications,” in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, Jun. 2015, pp. 450–456.
- [186] P. ROBILLARD, “Estimating the O-D matrix from observed link volumes,” *Transportation Research*, vol. 9, no. 2, pp. 123–128, Jul. 1975.
- [187] S. ROBINSON, “Distributed Simulation and Simulation Practice,” *SIMULATION*, vol. 81, no. 1, pp. 5–13, Jan. 2005.
- [188] M. RONDINONE, J. MANEROS, D. KRAJZEWICZ, R. BAUZA, P. CATALDI, F. HRIZI, J. GOZALVEZ, V. KUMAR, M. RÖCKL, L. LIN, O. LAZARO, J. LEGUAY, J. HÄRRI, S. VAZ, Y. LOPEZ, M. SEPULCRE, M. WETTERWALD, R. BLOKPOEL, and F. CAR-TOLANO, “ITETRIS: A modular simulation platform for the large scale evaluation of cooperative ITS applications,” *Simulation Modelling Practice and Theory*, vol. 34, pp. 99–125, May 2013.
- [189] C. ROTH, H. BUCHER, A. BRITO, O. SANDER, and J. BECKER, “A Simulation Tool Chain for Investigating Future V2X-based Automotive E/E Architectures,” in *Proceedings of the 7th European Congress on Embedded Real Time Software and Systems (ERTS²)*, Feb. 2014, pp. 1–10.
- [190] T. ROTH, M. BURNS, and T. POKORNY, “Extending Portico HLA to Federations of Federations with Transport Layer Security,” in *2018 Fall Simulation Innovation Workshop*. SISO, Sep. 2018, pp. 1–10.

- [191] A. RUSCHEINSKI and A. UHRMACHER, "Provenance in modeling and simulation studies — Bridging gaps," in *2017 Winter Simulation Conference (WSC)*. IEEE, Dec. 2017, pp. 872–883.
- [192] M. SARAOLU, A. MOROZOV, and K. JANSCHKEK, "MOBATSim: MOdel-Based Autonomous Traffic Simulation Framework for Fault-Error-Failure Chain Analysis," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 239–244, Jan. 2019.
- [193] J. L. SARLI, H. P. LEONE, and M. DE LOS MILAGROS GUTIÉRREZ, "Ontology-based semantic model of supply chains for modeling and simulation in distributed environment," in *Proceedings of the 2016 Winter Simulation Conference*, T. M. K. ROEDER, P. I. FRAZIER, R. SZECHTMAN, E. ZHOU, T. HUSCHKA, and S. E. CHICK, Eds. Piscataway, New Jersey: IEEE, 2016, pp. 1182–1193.
- [194] E. SAULNIER and B. BORTSCHELLER, "Simulation model reusability," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 64–69, Mar. 1994.
- [195] F. SCHLOEGL, S. ROHJANS, S. LEHNHOFF, J. VELASQUEZ, C. STEINBRINK, and P. PALENSKY, "Towards a classification scheme for co-simulation approaches in energy systems," in *2015 International Symposium on Smart Electric Distribution Systems and Technologies (EDST)*. IEEE, Sep. 2015, pp. 516–521.
- [196] T. SCHULZE, S. STRASSBURGER, and U. KLEIN, "Migration of HLA into Civil Domains: Solutions and Prototypes for Transportation Applications," *SIMULATION*, vol. 73, no. 5, pp. 296–303, Nov. 1999.
- [197] B. SCHÜNEMANN, "V2X simulation runtime infrastructure VSimRTI: An assessment tool to design smart traffic management systems," *Computer Networks*, vol. 55, no. 14, pp. 3189–3198, Oct. 2011.
- [198] S. SCHÜTTE, S. SCHERFKE, and M. TRÖSCHEL, "Mosaik: A framework for modular simulation of active components in Smart Grids," in *2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS)*. IEEE, Oct. 2011, pp. 55–60.
- [199] J. SEWALL, D. WILKIE, and M. C. LIN, "Interactive hybrid simulation of large-scale traffic," in *Proceedings of the 2011 SIGGRAPH Asia Conference*. New York, NY, USA: Association for Computing Machinery, Dec. 2011, pp. 1–12.
- [200] S. SHEKHAR, H. ABDEL-AZIZ, M. A. WALKER, F. CAGLAR, A. GOKHALE, and X. KOUTSOUKOS, "A simulation as a service cloud middleware," *Annals of Telecommunications*, vol. 71, pp. 93–108, 2016.
- [201] R. SIEGFRIED, J. LLOYD, and T. BERG, "A new reality: Modelling & simulation as a service," *J. of Cyber Security and Information Systems*, vol. 6, no. 3, pp. 18–29, 2018.

- [202] R. SIEGFRIED, T. VAN DEN BERG, A. CRAMP, and W. HUISKAMP, “M&S as a service: Expectations and challenges,” in *2014 Fall Simulation Interoperability Workshop*, SISO. SISO, 2014, pp. 248–257.
- [203] N. SIEVERT, “Modelica Models in a Distributed Environment Using FMI and HLA,” Linköping University, Software and Systems, Master’s Thesis, 2016.
- [204] A. SKOOGH and B. JOHANSSON, “Mapping of Time-Consumption During Input Data Management Activities,” *SNE Simulation Notes Europe*, vol. 19, no. 2, pp. 39–46, Aug. 2009.
- [205] C. SOMMER, D. ECKHOFF, A. BRUMMER, D. S. BUSE, F. HAGENAUER, S. JOERER, and M. SEGATA, “Veins: The Open Source Vehicular Network Simulation Framework,” in *Recent Advances in Network Simulation*, A. VIRDIS and M. KIRSCHKE, Eds. Cham: Springer International Publishing, 2019, pp. 215–252.
- [206] C. SOMMER, R. GERMAN, and F. DRESSLER, “Bidirectionally coupled network and road traffic simulation for improved IVC analysis,” *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, Jan. 2011.
- [207] P. SOMMER, F. SCHELLROTH, M. FISCHER, and J. SCHLECHTENDAHL, “Message-oriented Middleware for Industrial Production Systems,” in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, Aug. 2018, pp. 1217–1223.
- [208] C. STEINBRINK, F. SCHLÖGL, D. BABAZADEH, S. LEHNHOFF, S. ROHJANS, and A. NARAJAN, “Future Perspectives of Co-Simulation in the Smart Grid Domain,” in *2018 IEEE International Energy Conference (ENERGYCON)*. IEEE, Nov. 2018, pp. 1–6.
- [209] G. STETTINGER, M. BENEDIKT, N. THEK, and J. ZEHETNER, “On the difficulties of real-time co-simulation,” in *International Conference on Computational Methods for Coupled Problems in Science and Engineering*, 2013, pp. 1–11.
- [210] S. SÜSS, A. STRAHILOV, and C. DIEDRICH, “Behaviour simulation for virtual commissioning using co-simulation,” in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*. IEEE, Sep. 2015, pp. 1–8.
- [211] F. TAO, Q. QI, A. LIU, and A. KUSIAK, “Data-driven smart manufacturing,” *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, Jul. 2018.
- [212] S. J. E. TAYLOR, T. ELDABI, T. MONKS, M. RABE, and A. M. UHRMACHER, “Crisis, what crisis: Does reproducibility in modeling & simulation really matter?” in *2018 Winter Simulation Conference (WSC)*. IEEE, Dec. 2018, pp. 749–762.

- [213] C. THULE, K. LAUSDAHL, C. GOMES, G. MEISL, and P. G. LARSEN, “Maestro: The INTO-CPS co-simulation framework,” *Simulation Modelling Practice and Theory*, vol. 92, pp. 45–61, Apr. 2019.
- [214] A. TOLK, S. Y. DIALLO, J. J. PADILLA, and C. D. TURNITSA, “How is M&S interoperability different from other interoperability domains?” *M&S Journal*, vol. 7, no. 3, pp. 5–14, 2012.
- [215] O. TOPÇU, U. DURAK, H. OĞUZTÜZÜN, and L. YILMAZ, *Distributed Simulation: A Model Driven Engineering Approach*, ser. Simulation Foundations, Methods and Applications. Cham: Springer International Publishing, 2016.
- [216] E. TRUNZER, P. PRATA, S. VIEIRA, and B. VOGEL-HEUSER, “Concept and Evaluation of a Technology-independent Data Collection Architecture for Industrial Automation,” in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2019, pp. 2830–2836.
- [217] E. TRUNZER, T. SCHILLING, M. MÜLLER, and B. VOGEL-HEUSER, “Comparison of Communication Technologies for Industrial Middlewares and DDS-based Realization,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 10 935–10 942, 2020.
- [218] A. M. UHRMACHER, S. BRAILSFORD, J. LIU, M. RABE, and A. TOLK, “Panel — Reproducible research in discrete event simulation — A must or rather a maybe?” in *2016 Winter Simulation Conference (WSC)*. IEEE, Dec. 2016, pp. 1301–1315.
- [219] M. VAN STEEN and A. S. TANENBAUM, *Distributed Systems*, 3rd ed. Maarten van Steen, 2017.
- [220] J. VANURA and P. KRIZ, “Performance Evaluation of Java, JavaScript and PHP Serialization Libraries for XML, JSON and Binary Formats,” in *Services Computing – SCC 2018*, ser. Lecture Notes in Computer Science, J. E. FERREIRA, G. SPANOUDAKIS, Y. MA, and L.-J. ZHANG, Eds. Cham: Springer International Publishing, 2018, no. 10969, pp. 166–175.
- [221] A. VERBRAECK, “Component-based distributed simulations: The way forward?” in *Proceedings of the 18th Workshop on Parallel and Distributed Simulation*. New York, NY, USA: Association for Computing Machinery, 2004, pp. 141–148.
- [222] VMWARE, INC., “Messaging that just works — RabbitMQ,” <https://www.rabbitmq.com/>, Aug. 2022.
- [223] W3C, “Extensible Markup Language (XML),” <https://www.w3.org/XML/>, Mar. 2022.

- [224] S. WANG and G. A. WAINER, “A simulation as a service methodology with application for crowd modeling, simulation and visualization,” *SIMULATION*, vol. 91, pp. 71–95, 2015.
- [225] C. WEINHARDT, A. ANANDASIVAM, B. BLAU, N. BORISSOV, T. MEINL, W. MICHALK, and J. STÖSSER, “Cloud Computing – A Classification, Business Models, and Research Directions,” *Business & Information Systems Engineering*, vol. 1, no. 5, pp. 391–399, Oct. 2009.
- [226] WIKIMEDIA FOUNDATION, INC., “Wikipedia Main Page,” https://en.wikipedia.org/w/index.php?title=Main_Page, Jun. 2022.
- [227] E. WOLFF, *Microservices: Grundlagen Flexibler Softwarearchitekturen*. dpunkt.verlag, 2018.
- [228] T. YARYGINA and A. H. BAGGE, “Overcoming Security Challenges in Microservice Architectures,” in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, Mar. 2018, pp. 11–20.
- [229] L. YILMAZ, “Using meta-level ontology relations to measure conceptual alignment and interoperability of simulation models,” in *Proceedings of the Winter Simulation Conference*, S. G. HENDERSON, B. BILLER, M.-H. HSIEH, J. SHORTLE, J. D. TEW, and R. R. BARTON, Eds. Piscataway, New Jersey: IEEE, 2007, pp. 1090–1099.
- [230] L. YILMAZ, S. J. E. TAYLOR, R. FUJIMOTO, and F. DAREMA, “Panel: The future of research in modeling and simulation,” in *Proceedings of the Winter Simulation Conference 2014*. IEEE, Dec. 2014, pp. 2797–2811.
- [231] J. YONGGUO, L. QIANG, Q. CHANGSHUAI, S. JIAN, and L. QIANQIAN, “Message-oriented Middleware: A Review,” in *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, Aug. 2019, pp. 88–97.
- [232] L. C. YU, J. S. STEINMAN, and G. E. BLANK, “Adapting your simulation for HLA,” *SIMULATION*, vol. 71, no. 6, pp. 410–420, 1998.
- [233] D. ZEHE, A. KNOLL, W. CAI, and H. AYDT, “SEMSim Cloud Service: Large-scale urban systems simulation in the cloud,” *Simulation Modelling Practice and Theory*, vol. 58, pp. 157–171, 2015.
- [234] ZEROMQ, “ZeroMQ,” <https://zeromq.org/>, Aug. 2022.
- [235] H. M. ZHANG, “A non-equilibrium traffic model devoid of gas-like behavior,” *Transportation Research Part B: Methodological*, vol. 36, no. 3, pp. 275–290, 2002.

- [236] F. ZHU, Y. YAO, H. CHEN, and F. YAO, “Reusable Component Model Development Approach for Parallel and Distributed Simulation,” *The Scientific World Journal*, vol. 2014, pp. 1–12, Mar. 2014.
- [237] F. ZHU, Y. YAO, J. LI, and W. TANG, “Reusability and composability analysis for an agent-based hierarchical modelling and simulation framework,” *Simulation Modelling Practice and Theory*, vol. 90, pp. 81–97, 2019.

Statement on Contribution with regard to Self-Citations

Statement on Contribution to Citation [95]

M. GÜTLEIN, W. BARON, C. RENNER, and A. DJANATLIEV, "Performance Evaluation of HLA RTI Implementations," in 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT). IEEE, Sep. 2020, pp. 1–8.

- Idea and concept
- Formulation and solution of the problem
- Preponderant implementation and evaluation
- Preponderant preparation of the paper

Statement on Contribution to Citation [96]

M. GÜTLEIN and A. DJANATLIEV, "Coupled Traffic Simulation by Detached Translation Federates: An HLA-Based Approach," in 2019 Winter Simulation Conference (WSC). IEEE, Dec. 2019, pp. 1378–1389.

- Idea and concept
- Formulation and solution of the problem
- Preponderant implementation and evaluation
- Preponderant preparation of the paper

Statement on Contribution to Citation [97]

M. GÜTLEIN and A. DJANATLIEV, "Modeling and Simulation as a Service using Apache Kafka," in Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. SCITEPRESS - Science and Technology Publications, 2020, pp. 171–180.

- Idea and concept

- Formulation and solution of the problem
- Preponderant implementation
- Preponderant preparation of the paper

Statement on Contribution to Citation [98]

M. GÜTLEIN and A. DJANATLIEV, "On-demand Simulation of Future Mobility Based on Apache Kafka," in Simulation and Modeling Methodologies, Technologies and Applications, ser. Lecture Notes in Networks and Systems, M. S. OBAIDAT, T. OREN, and F. D. RANGO, Eds. Cham: Springer International Publishing, 2022, pp. 18–41.

- Idea and concept
- Formulation and solution of the problem
- Preponderant implementation and evaluation
- Preponderant preparation of the paper

Statement on Contribution to Citation [99]

M. GÜTLEIN, R. GERMAN, and A. DJANATLIEV, "Towards a Hybrid Co-simulation Framework: HLA-Based Coupling of MATSim and SUMO," in 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT). IEEE, 2018, pp. 1–9.

- Idea and concept
- Formulation and solution of the problem
- Preponderant implementation and evaluation
- Preponderant preparation of the paper

Statement on Contribution to Citation [100]

M. GÜTLEIN, R. GERMAN, and A. DJANATLIEV, "Hide Your Model! Layer Abstractions for Data-Driven Co-Simulations," in 2021 Winter Simulation Conference (WSC). IEEE, Dec. 2021, pp. 1–12.

- Idea and concept
- Formulation and solution of the problem
- Preponderant implementation of the model
- Preponderant preparation of the paper