Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, eds.

HIDE YOUR MODEL! LAYER ABSTRACTIONS FOR DATA-DRIVEN CO-SIMULATIONS

Moritz Gütlein
Reinhard German
Anatoli Djanatliev

Computer Networks and Communication Systems
University of Erlangen-Nuremberg
Martensstr. 3
Erlangen, 91058, GERMANY

ABSTRACT

Modeling and simulating of problems that span across multiple domains can be tricky. Often, the need for a co-simulation arises, for example because the modeling cannot be done with a single tool. Domain experts may face a barrier when it comes to the implementation of such a co-simulation. In addition, the demand for integrating data from various sources into simulation models seems to be growing. Therefore, we propose an abstraction concept that hides simulators and models behind generalized interfaces that are derived from prototypical classes. The data-driven abstraction concept facilitates having an assembly kit with predefined simulator building blocks that can be easily plugged together. Furthermore, data streams can be seamlessly ingested into such a composed model. Likewise, the co-simulation can be accessed via the resulting interfaces for further processing and interactions.

1 INTRODUCTION

Modeling and simulation can help to develop solutions for urging problems. With an increasing linkage of different sectors, rapid progress results in useful technologies that are likely more and more complex. One level of complexity is inferred from the spanning across several domains. Tight connections between domains enable new inventions, whereas connecting different domains may raise completely new problems at the same time. Luckily, simulation can also support developments for cross-domain problems. However, since simulators and models typically have a specific focus, the need for connecting different tools arises as well. This coupling is known as co-simulation, which is defined by Gomes et al. (2018) as ”the theory and techniques to enable global simulation of a coupled system via the composition of simulators”.

Regarding the realization of co-simulations, there are two major problems. First, how to connect different components? Second, how to bring data in and out of the composed model? Behind these rather simple questions stand various related problems arising from technical as well as conceptual contexts. In addition, potential users of a composed simulation model may not have the required knowledge and/or resources to implement a co-simulation on their own. Such domain experts could benefit from an easy-to-use toolkit that allows setting up and running a co-simulation via a graphical user interface. For this purpose, a user should be able to access a set of different building blocks, which can be puzzled together in order to model a problem adequately. Such a toolkit was proposed by Gütlein and Djanatliev (2020). Each building block contains a pre-assembled simulator (and potentially other tools). The composability is driven by homogeneous interfaces at the boundaries of building blocks. Typically, these interfaces shall not disclose the entire simulator, but only a reduced subset of functionality. We are aware that the intentional limitation of a building block’s functionality is not always useful. However, there are a number of cases where it is

978-1-6654-3311-2/21/$31.00 ©2021 IEEE

beneficial to disclose as little functionality as possible (e.g., simplicity when modeling the connections of components). The idea of deriving these reduced interfaces is described in this paper.

When thinking on a higher level about connecting different modules, the interfaces are one of the most obvious issues. Thoughts about the technical realization (protocols, message ordering, synchronization, quality of service guarantees, ...) are insignificant until it is clear what kinds of information should be exchanged. Therefore, there is a need to agree on a common language with a dictionary of available words. Any modeling of a composed system would not be possible without that information. Unfortunately, the design of such a dictionary is not straightforward. Every use case has its own particular requirements and circumstances, which is why the design processes may differ for each scenario. However, from our experience, one can outline two opposing modes when sketching the interfaces.

Option A): there is a clear picture of the use case. In addition, the involved components are identified. There are enough resources to modify existing software packages, write wrappers for them, or maybe even implement completely new components for that specific use case.

Option B): there is a broader vision of would should be possible, but there may be no sharp use case that can be clearly specified. There is either a very generic purpose, or a variety of use cases that should be covered. Resources may be limited and software packages should be replaceable. Tools are possibly not even known at planning time. Therefore, the coupling of the different components should be rather loose. Reusability of parts would be a great advantage for follow-on scenarios.

In the case of option A), a top-down approach seems to work fine. Interfaces can be specified in detail, and be thought of with global knowledge. This can increase effectiveness dramatically. At the same time, this encourages close couplings within the overall model. Consequently, the result is a static model that may strongly rely on internal knowledge of incorporated tools. Even software versions might matter. Option B) is more likely pointing in the direction of a bottom-up strategy. Contrarily to the previous mode, interfaces are designed use-case agnostic. To achieve that, tools that implement related paradigms/models within a domain are clustered in groups. Such a group will be called *layer* in the following. Interfaces are designed to represent the prototypical understanding of the layer. This implies that multiple tools that fit into the same layer can be addressed in the same way. For instance, a control algorithm that was written once can be used with different simulators of the same layer. Of course, this also brings some drawbacks. In particular, some special functionalities of certain tools may be hidden and may not be accessible anymore for obvious reasons. However, the focus on the real purpose of each model is encouraged as also proclaimed in the field of component-based simulation. The main novelty is that only the elements of the (layer-)prototypical class are guaranteed to be exposed by the layer interface. A model composition will be exclusively based on these elements.

By having a set of tools that are abstracted and attachable in this manner, various co-simulations can be built from scratch. Additionally, reusability of components is strengthened. In this paper, we therefore propose a concept that is based on *Option B* and encourages thinking, modeling, and building co-simulations in a more data-driven and component-based way. As a consequence, the focus is shifted from tools to their interfaces and the information flows. Related work on the topic is presented in Section 2. More details on the proposed approach will be given in Section 3. The resulting connection modes are presented in Section 4. An illustration of the concept is given in Section 5. Afterwards, a conclusion and an outlook will follow in Section 6.

2 RELATED WORK

In this section, a brief overview of related work is given. As the layer concept has a significant impact on the implementation of co-simulations, basic information about coupled simulations needs to be clarified at first. Second, different modeling methodologies are presented.

Besides implementing custom point-to-point connections between tools, using a middleware-based approach is a widely accepted concept for connecting simulators. The High Level Architecture (HLA) is one of these middleware-based approaches and the de facto standard for distributed simulations and

co-simulations. While its development has already been started in the 1990s, the current version is still actively used (IEEE 2010). Since the HLA aims for interoperability between simulators, the description of exchangeable data packets was addressed from the beginning. While the Federation Object Model (FOM) describes a set of attributes, interactions, and objects that are available across a federation (i.e., a set of connected simulators), the Simulation Object Model (SOM) describes a similar set that is valid for a federate (i.e., a single simulator) inside a federation (Dahmann et al. 1998). A simulation and modeling as a service framework that is based on the proposed concept of this paper is described by Gütlein and Djanatliev (2020). Core ideas of the approach, such as a hierarchical data model, interactions, and the publish/subscribe communication pattern, are strongly influenced by the HLA. At the same time, these ideas do not originate exclusively out of the Modeling and Simulation (M&S) domain, as they are representing the fundamental building blocks of distributed computer systems. Technical details are not in the scope of this paper, but a remark is necessary for a full understanding of the benefits of the layer approach: communication is exclusively realized on top of a topic based publish/subscribe middleware. There is no directed communication between coupled components. In order to structure the communication, the topics are logically split into three different logical topic groups: orchestration (mainly used to start/stop a simulation and synchronize clocks), interaction (API proxy), and provision (provide information such as attribute values or final results).

While the techniques are inspired by the HLA, our proposed coupling paradigm contradicts core ideas of the HLA. Accordingly to HLA's paradigm, each federate knows the whole FOM. The FOM is seen as more important than the SOM as it enables the data flows between federates (Wenguang et al. 2009). Contrarily, we propose that each component should not know anything about its environment, in order to achieve a very loose coupling that leads to reusability, isolated testability, exchangeability, and extendability of components. The goal is to shift the focus from tools to information flows (Gütlein and Djanatliev 2019). Even more, one could argue that creation of an HLA-federated co-simulation is a top-down approach. Once you have the full picture of all involved domains, object hierarchies and interactions, you can design the federation object model and tailor your simulators to work with that FOM (Lutz, Scrudder, and Graffagnini 1998). Without any doubt, this has certain advantages especially if you work on huge federations with many involved stakeholders and have the resource to create simulators from scratch. We propose the opposite approach, thinking a federated simulation in a bottom-up manner. This fits our requirements and resources better. In this mode, we assume an existing set of simulators. Users/researchers can pick, based on their needs, what components they need and put them together in a plug and play manner.

Naturally, model (interface) development is an important topic in the field of M&S. While early publications reach back to the 1980's (Nance 1984), there is still ongoing work on the process of modeling. Moradi et al. (2006) tackle the problem of growing complexity of simulation models when simulating sophisticated problems. They propose the use of small submodels that can be composed. A Base Object Model (BOM) encapsulates the HLA FOM/SOM model i.a. together with a conceptual description of the model. The contained metadata should strengthen the composability of models by matching similar BOMs. Besides works that are related to the HLA (Yu et al. 1998; Rathnam and Paredis 2004; Çetinkaya and Oğuztüzün 2006), there are various approaches that are more general. The broad ideas of object-orientation have an impact on simulation (i.e., object-oriented simulation), as well as on modeling (i.e., object-oriented modeling). Through encapsulation, individual behavior can be modeled. The interplay of autonomous objects results in a global behavior that can be analyzed. This is tightly linked to the Agent-Based Modeling (ABM) paradigm (Joines and Roberts 1999; Nan and Eusgeld 2011).

Ontological methods allow describing entities and their relations to each other within a domain. Using an ontology approach means that the inner essence or the real nature of a concept is extracted (Benjamin et al. 2006). The authors add that one major issue is to find the right granularity of an ontology, because one can get lost in details or miss important points. Yilmaz (2007) proposes to evaluate the conceptual alignment of models by comparing a common meta model with the domain's ontology. Sarli et al. (2016) use an ontology network in order to generate FOMs semi-automatically in the field of supply chains.

These ontological methods are about identifying entities, their parameters, and their relations. We assume that ontological concepts can also be used to describe abstractions for groups of related models. In the following, such an abstraction is referred to as *layer*.

Component-based simulation differs from object-oriented simulation mainly regarding the concept of inheritance. In component-based simulation, extended functionality is preferably achieved by establishing a coupling between different components and not by inheriting and extending functionality (Buss 2000). Boer and Verbraeck (2003) describe the attempt to interface different commercial off-the-shelf (COTS) simulation models in general as impossible if it is not possible to make adjustments within the models. Verbraeck (2004) discriminates software components from simulation building blocks. The author picks up a definition for building blocks, which are accordingly: self-contained, interoperable, reusable, replaceable, encapsulating their internals, providing useful services through precisely defined interfaces, and customizable to match requirements arising from their environment. This definition almost corresponds to our understanding of a building block with the important difference that in our conception a building block is not customizable, but can only be used as it is. Originating from an arbitrary organization or company, the building block's internals might not even be disclosed. Therefore, we propose accepting the black-box character of components and assuming their validity for a certain purpose/model.

Diallo et al. (2014) introduce a formalism of modeling and simulation grounded in model theory. Based on this, the authors define the concept of interoperability: a model is interoperable with a reference model, if it is a valid model for that reference model. (A reference model represents a modeler's worldview.) Since we assume a building block to represent a valid model for some reference model, we also assume their composition to be valid. Hiding of model parts can lead to a conceptual misalignment, a lack of consistency, and different truths within connected models that use a shared concept (Tolk et al. 2012). Following the black box approach, we require that within each domain only one component can be responsible for a particular region (not necessarily a spatial region). In other words, we accept that neighboring models/simulators may have a different worldview. Since they are understood as services with defined inputs and outputs, the potential misalignment is not identified as a problem by us. Furthermore, it brings the advantage of being able to incorporate other components such as physical devices and form for instance a Hardware-in-the-loop simulation (Bacic 2005).

3 LAYER ABSTRACTION

In this section, the fundamental thoughts behind the layer abstraction and its application are described and the layer is defined. As the concept is applicable on multiple components (whole simulators, submodels, and other software/hardware artifacts), we will use the term *instance* in the following in order to have a common term for all different components. For the same reason, the term *entity* is used to describe an element of such an instance. However, the concept is not limited to modeling paradigms that incorporate entities. It is, for instance, also applicable on flows. There is one requirement for an instance. It must be possible to identify a single class of entities that represent the essence of the instance. Entities of this class will be referred to as *native entities*.

A *layer* is used to describe the inner essence as well as the interface of a simulator. Each layer is assigned to a single application domain (e.g., traffic or communication). Most importantly, the layer describes the layer's native entity class. The description of this entity-in-focus is referred to as *native data model (ndm)* of a layer. The ndm is a data structure that is representing the native entity (e.g., for a microscopic traffic model this could be a set of attributes that describe a vehicle: id, speed, ...). Based on the layer, the interfaces are derived. Therefore, the interfaces of all instances assigned to the same layer will be identical. The purpose of the layer is to have a tool and a methodology for describing a group of related models and their interfaces in a homogeneous way. As a consequence, all messages that will flow between instances are exclusively based on the layer.

3.1 Assumptions

A co-simulation is about establishing information exchange between the coupled components. In addition to the technical question of how to realize the communication, there is the rather methodical question of what to communicate. Related to the second question, we postulate three conditions that strongly influence the way co-simulations will be realized using the layer concept. First, neighborhood-agnostic: a single instance is not aware of its neighborhood (i.e., what other instances are involved in the co-simulation). Second, disjoint regions: within a domain, only one instance can be responsible for a certain region. Regions do not have to be formed in a spatial dimension. Third, native tongue: an instance cannot understand and process messages from other layers. This implies that each layer has its own typical message format.

While there is no awareness about the neighborhood, an instance is aware of its region of responsibilities (e.g., all entities in a specific spatial area). It shall be possible that an instance's set of responsibilities changes over time, for instance, because the region itself is modified (e.g., due to load balancing), or because entities move between regions (always considering our definition of *entity*). This requires that entities can be integrated and be sent out by instances. Since there is no knowledge about the other partitions, there can be no bidirectional transfer between the emitting and the adopting instance. Therefore, the information flow is realized by using a publish/subscribe messaging pattern. In this manner, an instance can send out information without knowing and caring about possible recipients. From a receiver's perspective, each instance will subscribe to information that is related to their responsibility region without knowing the sender. However, there is always the constraint that the regions are disjoint and not assigned to multiple instances. As a consequence, a great level of decoupling between the different partitions is realized. Without the need to be aware of the sender's identity, information cannot only flow between simulators, but also between simulators and historical data providers or even real-world live data streams.

The implications lead on to the third assumption. If information could flow between all instances, there would be a need to translate the contents of messages between different paradigms, levels, or even domains. One approach would be to let each instance try to interpret data that is received in formats that differ from the own ndm. We have chosen not do it in this way. Consequently, an instance can only process messages from the own layer. Dedicated helper components are required that are capable of translating messages between two layers. The reason for this is that we presume existing simulators/models to be valid at the first place. From the point where multiple valid models are coupled and room for interpretation of information arises, validity of the composed model is affected. In essence, this fundamental problem cannot be prevented. However, by outsourcing of the translation logic, an isolation consideration of this critical part is made possible. Additional advantages of having dedicated translators include i.a. isolated testing, enhanced reproducibility, and improved reusability. Complex simulator topologies can be realized without the need to implement interpreters for the ndms of all available layers in all simulators. A translation logic that was written once can be used by all models of the related layers. As a result, each instance will only communicate through messages in its native layer tongue (i.e., send/receive ndm tuples).

3.2 Layer Definition

Due to the wide range of different characteristics, there is no simple rule that allows a straightforward inference of a layer's values for a certain instance. Each domain, each modeling paradigm, and each purpose leads to varying situations, which cannot be considered equally. As a result, reasoned design decisions are required for each individual case. The structure of the layer, on the other hand, is defined in the following.

3.2.1 Native Data Model

The first questions that need to be asked are: What is this layer about? What is the purpose? What is the main element type of the model? If these questions cannot be answered, it might be necessary to split the model into submodels. When there is a clear picture of such a main entity, a representing data model can be constructed that encloses its typical parameters and methods. This native data model (ndm) can then

be used to access an instance's native entities from other components. In addition, it is used to transport information about entities that enter or leave an instance's region.

3.2.2 Structure

While the ndm is essential for the coupling, the interfaces could also be used by third party components to interact, control, or analyze a simulation run. The usability would be very limited, if the layer interface would exclusively depend on the ndm. For this reason, it is possible to extend the layer (and thus also the interface). A visual representation is given in Figure 1. Each layer has a domain-unique name, a reference to a domain, a version, and the ndm. The ndm follows the type of a compound. A compound consists of primitive attributes, methods, and again compound structures. In this manner, arbitrary complex structures can be constructed. The layer itself is a compound. It can optionally contain additional compound attributes, primitive attributes, and methods itself. From a practical perspective, these values are specified within a JSON document, which must be compliant with a corresponding JSON schema definition of the layer structure (Gütlein 2021). Therefore, a layer can be understood as a class in the object-oriented world.

Compound and primitive attributes do not only provide information about (composite) parameter names and their types, but also if getter and setter methods are accessible for each item and if they can be subscribed to by observers (which are populating provision topics). In addition, a compound has an *isPersistent* flag, which indicates whether represented entities are persistent during a simulation run, or can be dynamically added and removed. Each compound needs to have a key parameter. In Addition, further methods can be added that are not directly inferred from the data models' attributes (i.e., they are no simple getter/setter methods). A method has a name, a set of inputs and an output data type. A primitive holds the optional fields unit and description.

With each given compound, a corresponding scope is introduced. Based on the given information, the inferring of an interface definition that is valid for all different simulators of the same layer is possible. Finally, the scope, the compounds, the attributes, and the methods are used to generate mappings, which provide access to the different elements. If using a topic based publish/subscribe middleware, the hierarchical structure can directly be mapped on corresponding topics.

3.2.3 Design

Whether considering the ndm or the additional attributes: specifying a layer for some instance comes down to a compromise between generalization and specialization. For example, if there are a number of established tools that can be assigned to a layer, one starting point is to begin with their common subset. (Of course, the absence of an attribute/function in a single tool cannot be a criterion for exclusion.) Another approach is to ask from a more methodological, engineering, or even philosophical perspective what the nature of the model is all about. Procedures from the fields of semantics and ontology will help here.

3.2.4 Layer Interface

The ndm is the heart of each layer definition as it gives information about the core of the simulation model. Within a layer description, the ndm is not inevitably given on its own, but rather embedded in a broader description of the layer. When the layer design is finished, its description has to be provided in a machine-

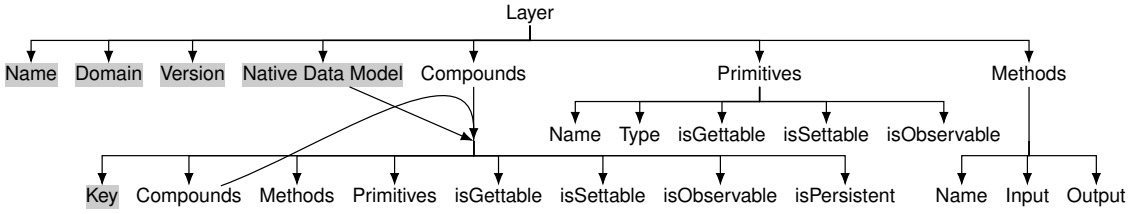

Figure 1: Definition of the layer structure. Gray fields are mandatory.

readable format (i.e., in this case via the referenced JSON schema for the layer structure). Based on that, further actions can be triggered automatically. For instance, a human readable interface documentation can be generated, as well as serialization definitions for the compounds (e.g., Avro/JSON/ProtoBuf schemes). Code generation can be used to produce program stubs that incorporate the interface and give developers a starting point for attaching their tool wrappers. Of course, all attribute meta information has to be considered at this stage (e.g., an *isGettable* flag results in a corresponding getter method stub for that specific attribute).

As described in Section 2, there is a co-simulation service that is based on the layer concept (Gütlein and Djanatliev 2020). Regarding the used interactions and provision topic groups, the layer description is used to infer the set of available attributes and the interface description (in terms of a set of publicly available methods that provide access to these attributes).

4 CONNECTING LAYERS

One of the main purposes that motivated the layer abstraction is the ability to establish information exchange between multiple components (i.e., couple tools) in a uniform manner. Therefore, couplings will be formed by exchanging ndm tuples. Depending on the topology, different circumstances have to be taken into account. All of them can be broken down into three basic problems, which results in three different kinds of connectors. Specific implementations for them are highly use case specific and not in the scope of this paper. However, the nature of the three mechanism will be described in the following. For illustration purposes, Figure 2 depicts two different models of an exemplary scenario, where entities (still considering the entity definition from Section 3) move on a map. While the blue model may represent single persons, the green model may use an aggregated representation.

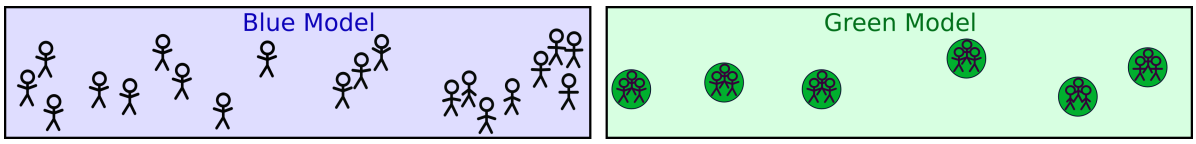

Figure 2: Blue and green model of the base scenario.

4.1 Simple

The simplest case involves the composition of two instances of the same layer. Following the layer logic, this implies that there is a common language and therefore there is no need for a translation, when information flows between instances. When transferring an entity is necessary, the ndm data tuple can be sent and processed natively. For demonstration, the base scenario is partitioned spatially into two regions (see Figure 3). These regions are simulated by *Instance A* and *Instance B*, respectively. If an entity leaves the scope of one partition (e.g., the red one moving to the right), the entity can be transferred solely using the ndm tuple that contains all necessary information to fully represent the entity. Of course, the procedure can also be applied on topologies that consist of more than two instances of the same layer.

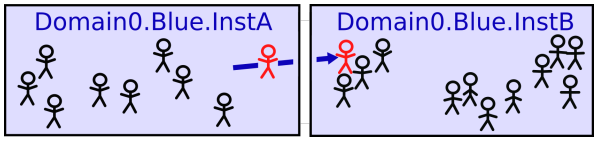

Figure 3: Partitioned scenario with two instances of the same layer.

4.2 Translation

In the second case, there is a need for a translation. The base scenario is again partitioned spatially into two regions (see Figure 4). However, the modeling paradigm of instances *A* and *B* is not the same anymore. The reasons for this setup can be manifold (e.g., meeting performance constraints). Consequently, the layers of *A* and *B* differ. As a result, information cannot flow straight away between instances and a translation

is required when transferring an entity. A translator component compromises functions that map the ndm tuple of the blue layer to the ndm tuple of the green layer and vice versa. The given example is a typical one, where the two layers represent different levels of detail. This leads to (dis)aggregation of information when translating.

4.3 Projection

A conventional co-simulation may also combine different domains. This represents the third case: coupling two instances of different domains (i.e., also different layers). When connecting two different layers of the same domain, a translation aims at conserving as much information as possible. In contrast, a projection might use only a subset of a layer's ndm intentionally. Information contained within that subset is typically incorporated into the environment of another layer. For example, the micro model could be extended with an instance from another domain. That instance may use the density of instance *A*'s entities in its model.

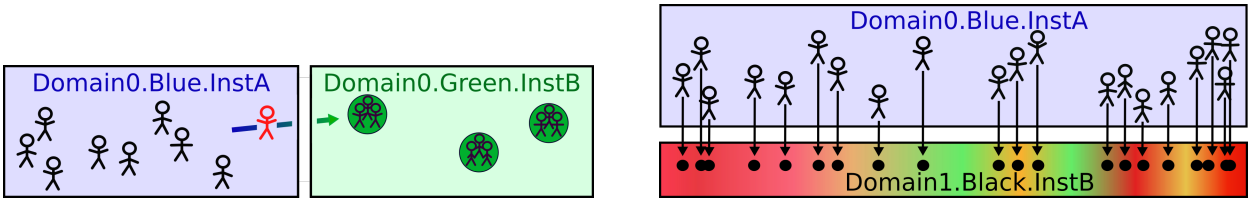

Figure 4: A partitioned scenario with two instances of different layers on the left (requires a translation) and a partitioned scenario with two instances of different domains on the right (requires a projection).

5 EXEMPLARY APPLICATION ON MICROSCOPIC TRAFFIC SIMULATION

In the following, the proposed concept is applied to the field of traffic. A layer for microscopic traffic simulation models is drafted. Microscopic traffic simulation describes the overall traffic flow by simulating the movements of individual vehicles (Barceló 2010). A vehicle's actions are influenced by the vehicle's surroundings. We will start with the mandatory fields of the layer definition: name, domain, version, and ndm (see Figure 1). The name will be *micro* and the domain *traffic*. Following the explanations in Subsection 3.2.1, the next step is to identify the essential entity in the ontology of microscopic traffic models. Entities within these models might be vehicles, edges, traffic lights, or even parking areas. The key point is that the set of vehicles allows drawing further conclusions and calculating additional indicators. For example, it is possible to estimate the average speed and flow rate on a certain road based on the vehicles' actions. The layer should represent the heart of the model. Since the most representing object class for this example is the vehicle, it is clear that the layer should be about vehicles. Roads, for example, are also important, but their attributes and states become interesting only through the effects of the vehicles' behavior. Furthermore, when we think about realizing a distributed simulation, one would typically want to transfer vehicles between the logical partitions and not roads. Therefore, a data structure that describes a vehicle will become the so-called ndm of the *traffic.micro* layer.

Next, the question of how to represent such a vehicle adequately for the entire group of microscopic models must be answered. Typical microscopic traffic simulators are SUMO (Lopez et al. 2018) and Vissim (Fellendorf and Vortisch 2010). An excerpt of a possible representation of a vehicle by SUMO and Vissim is given in Table 1. It is not straightforward to provide a complete representation, since both of the vehicle data models could be further extended by retrieving data from other objects (e.g., the current edge is directly available and could be used to identify surrounding vehicles by querying all vehicles on that edge). The provided sets of attributes are solely based on the contents of the tools' main documentation artifacts (DLR 2021; PTV AG 2021). However, this does not imply that all attributes are mapped to a unique internal attribute (e.g., SUMO allows querying a *position* and a *position 3d* attribute from a vehicle). The collection of existing representations of microscopic vehicles gives a good starting point for constructing

a generalized data model for a vehicle. The following thoughts lead to the native data model given in the right column of Table 1:

- A key item is mandatory. That will be *vehicle id*.
- *Acceleration*, *(angle,) position*, *(slope,)* and *speed* are important figures when calculating the movement for the next simulation step.
- Vehicles do not move in free space, but on *edges* and *lanes* along a *route*.
- Related to practical aspects, there is no need to continuously provide attributes that are static. A vehicle's *width* is for example referenced by its *type*.
- Observable attributes are expected to change over time. Otherwise, there is no use in providing observers. Therefore, *vehicle id*, *route*, and *type* are not observable.
- Because *position edge* and *slope* are inferred from *position* and *lane*, there will be no setter. The *vehicle id* and the *type* of a vehicle can also not be changed during the simulation.

In this specific case, there is no need for further branched compounds and additional methods inside the ndm besides the listed *primitive* attributes. Because a vehicle has no persistent nature during a typical simulation run, the *isPersistent* attribute of the ndm compound is set to false. As a consequence, methods for instantiating and destroying entities will be enabled when inferring API methods from the layer description.

The result is a generalized representation of a microscopic vehicle. It contains enough information to reproduce vehicles. This allows for instance realizing a distributed traffic simulation solely based on exchanging the ndm tuples. Another possible use case could be to provide real floating car data (FCD) in this format and ingest it seamlessly into a micro simulation. As the layer definition will not change frequently, a data scientist or a traffic engineer can write algorithms that access the layer interface. Even if the underlying tools are exchanged, these scripts can be reused without changing a single line of code due to the layer abstraction.

As explained in Subsection 3.2.4, an interface for all tools that hide behind the micro layer can be inferred based on the elements of the layer description and their *isGettable, isSettable, isObservable* and *isPersistent* flags. Table 2 shows the attributes that are available for subscription/provision (first column) and the methods that are accessible for interaction calls (second column). Potential analytics tasks stretch further than just processing individual vehicles. Therefore, we will make use of an optional field and extend the layer definition. Besides the ndm, there will be an *edge* compound representing a piece of a road. Similarly to the vehicle, one can compare the modeling of edges in established tools and design a

Table 1: Established vehicle models and the inferred micro layer ndm.

Established vehicle models		**Inferred native data model**				
SUMO	**PTV Vissim**	Parameter	Type	get	set	obs.
Vehicle id	Name / No	Vehicle id	string	–	–	–
Acceleration	Acceleration	Acceleration	double	✓	✓	✓
Angle	Orientation angle	Angle	double	✓	✓	✓
Road id	Lane → edge	Edge	string	✓	✓	✓
Lane id	Lane	Lane	int	✓	✓	✓
Position 3d	Coordinates (8 different)	Position	vec3	✓	✓	✓
Lane position	Pos	Position edge	double	✓	–	✓
Route id	Route number	Route	list⟨string⟩	✓	✓	–
Slope	Edge → slope	Slope	double	✓	–	✓
Speed	Speed	Speed	double	✓	✓	✓
Type id	Vehicle type	Type	string	✓	–	–
Distance	Desired speed	–	–	–	–	–
Edges	Speed difference	–	–	–	–	–
Electricity consumption	Emissions (35 different)	–	–	–	–	–
Emissions (6 different)	Fuel consumption	–	–	–	–	–
... (68 in total)	... (157 in total)					

Table 2: Inferred access.

Provision	**Interaction**	
provision.vehicle	interaction.vehicle.add	interaction.vehicle.remove
–	interaction.vehicle.get	–
provision.vehicle.acceleration	interaction.vehicle.acceleration.get	interaction.vehicle.acceleration.set
provision.vehicle.angle	interaction.vehicle.angle.get	interaction.vehicle.angle.set
provision.vehicle.edge	interaction.vehicle.edge.get	interaction.vehicle.edge.set
provision.vehicle.lane	interaction.vehicle.lane.get	interaction.vehicle.lane.set
provision.vehicle.position	interaction.vehicle.position.get	interaction.vehicle.position.set
provision.vehicle.positionEdge	interaction.vehicle.positionEdge.get	–
–	interaction.vehicle.route.get	interaction.vehicle.route.set
provision.vehicle.slope	interaction.vehicle.slope.get	–
provision.vehicle.speed	interaction.vehicle.speed.get	interaction.vehicle.speed.set
–	interaction.vehicle.type.get	–

generalized edge structure. In contrast to the ndm, there will also be a definition for two additional methods that can be used for opening and closing lanes during a simulation run. Following the structure of Figure 1, the top hierarchy level can also host primitives (e.g., current *simulation time*) or methods that have a global scope. The structure of the final layer definition is given in Figure 5. As a result, microscopic simulators that are hidden behind the micro layer can still be used for various applications. Some of them are:

- Distributed microscopic traffic simulation: each instance subscribes to vehicles on all edges in its region, while publishing all vehicles that leave the region via ndm tuples (see Subsection 4.1).
- Multilevel traffic simulation: the prior case is extended with an additional layer (e.g., macroscopic traffic) and a corresponding translator (e.g., macro $\leftrightarrow$ micro). Responsible regions can again be realized with a set of edges. (see Subsection 4.2).
- Cross-domain co-simulation: a subset of the ndm (e.g., the position of each vehicle) is projected on nodes inside a wireless communication simulator to realize a V2X simulation (see Subsection 4.3)
- Digital twin: the first example is extended by incorporating data not from a neighboring simulator anymore, but rather from a real-world data source. Therefore, information of real cars in a region of interest is provided as ndm tuples and can processed without extra effort (see Subsection 4.1).

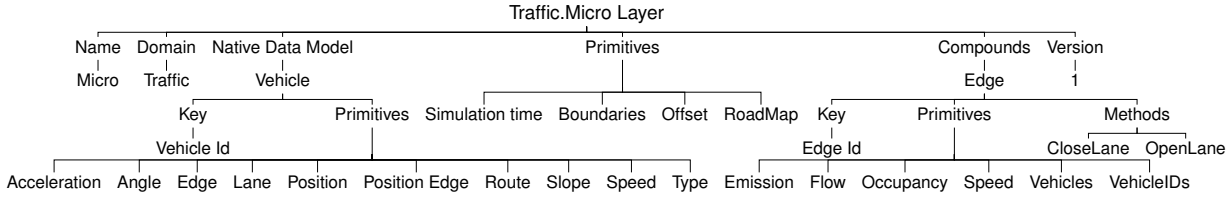

Figure 5: Traffic.Micro layer definition.

6 CONCLUSION & FUTURE WORK

In this paper, we tackled the problem of interfacing components in coupled simulations. This also addresses the question of how to feed such a simulation with input data and how to provide access to it. Therefore, we presented an abstraction approach for simulation models (and other components). We gave a brief review on related work, before we explained the concept of a layer abstraction by naming the involved assumptions and giving the definition of such a layer. Afterwards, we pointed out the three potential types of connecting instances hidden behind a layer. Finally, we illustrated the concept by applying it to the field of microscopic traffic simulation. The layer abstraction is intended to strengthen reusability, exchangeability of underlying simulators, reproducibility, and first of all provide the ability to enable new simulator couplings in a simple

way. The idea behind this approach is that it should be possible in many cases to identify a core element type within a model. Splitting up a model into several submodels can help if there is a problem in finding a single core element for a model. A prototypical data model is then constructed, which is representing this native entity type. The resulting data model is called native data model (ndm) of the layer. The ndm builds the foundation for inferring interfaces and transferring entities between instances. Besides the ndm, further secondary data structures and methods can be added to the layer definition. The interfaces of involved instances are solely inferred from the layer definition. All other parts of an instance's model are hidden and not accessible anymore. Various types of simulator couplings (e.g., co-simulation, distributed simulation, multi-level simulation) can be implemented based on this approach. Through the hiding of tool specific functionalities behind the layer interface, a user is obviously confronted with a reduced range of functionalities at first. However, the added value is derived from the manifold possibilities of data-driven coupled simulations. In addition, they can be set up in a plug-and-play manner.

Future work includes a toolkit that supports the design process of layer definitions and allows defining and managing translator and projector connectors. This will be based on an existing GUI tool already used to sketch coupled topologies based on spatial regions. In addition to the resulting advantages, the encapsulation caused by the layer interface can be a limitation for the user. Therefore, a study on the acceptance of using building blocks with intentionally restricted functionality would be interesting. However, this would require pushing ahead the application in practice first. In this sense, a study on the suitability of the presented approach in different application domains would be appealing. A catalog of predefined reference layer definitions and building blocks that can be used straight away could be the result.

ACKNOWLEDGMENTS

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

REFERENCES

Bacic, M. 2005. "On hardware-in-the-loop simulation". In *Proceedings of the 44th IEEE Conference on Decision and Control*, 3194–3198. Seville, Spain: Institute of Electrical and Electronics Engineers, Inc.

Barceló, J. 2010. "Models, traffic models, simulation, and traffic simulation". In *Fundamentals of traffic simulation*, edited by J. Barceló, 1–62. New York: Springer.

Benjamin, P., M. Patki, and R. Mayer. 2006. "Using ontologies for simulation modeling". In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland, B. G. L. J. Liu, D. M. Nicol, and R. M. Fujimoto, 1151–1159. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Boer, and Verbraeck. 2003. "Distributed simulation with COTS simulation packages". In *Proceedings of the 2003 Winter Simulation Conference*, edited by S. E. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, 829–837. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Buss, A. H. 2000. "Component-based simulation modeling". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 964–971. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Çetinkaya, D., and H. Oğuztüzün. 2006. "A metamodel for the HLA object model". In *Proceedings of the 20th European Conference on Modeling and Simulation*, edited by W. Borutzky, A. Orsoni, and R. Zobel, 207–213. Germany: ECMS.

Dahmann, J., R. M. Fujimoto, and R. M. Weatherly. 1998. "The DoD High Level Architecture: An Update". In *Proceedings of the 1998 Winter Simulation Conference*, edited by D. Medeiros, E. Watson, J. Carson, and M. Manivannan, 797–804. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Diallo, S. Y., J. J. Padilla, R. Gore, H. Herencia-Zapana, and A. Tolk. 2014. "Toward a formalism of modeling and simulation using model theory". *Complexity* 19(3):56–63.

DLR 2021. "Vehicle Value Retrieval". https://sumo.dlr.de/docs/TraCI/Vehicle_Value_Retrieval.html, accessed 6th April.

Fellendorf, M., and P. Vortisch. 2010. "Microscopic traffic flow simulator VISSIM". In *Fundamentals of traffic simulation*, edited by J. Barceló, 63–93. New York: Springer.

Gomes, C., C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe. 2018. "Co-Simulation: A Survey". *ACM Computing Surveys* 51(3):1–63.

Gütlein, M., and A. Djanatliev. 2020. "Modeling and Simulation as a Service using Apache Kafka.". In *Proceedings of the 10th International Conference on Simulation and Modeling Methodologies*, edited by F. D. Rango, T. Ören, and M. Obaidat, 171–180. Setúbal, Portugal: Scitepress.

Gütlein, M. 2021. "JSON Schema for Layer". https://github.com/cs7org/LayerDefinition, accessed 17th June.

Gütlein, M., and A. Djanatliev. 2019. "Coupled Traffic Simulation by Detached Translation Federates: An HLA-Based Approach". In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K.-H. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, 1378–1389. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

IEEE 2010. "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*:1–38.

Joines, J. A., and S. D. Roberts. 1999. "Simulation in an object-oriented world". In *Proceedings of the 1999 Winter Simulation Conference*, edited by P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, Volume 1, 132–140. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lopez, P. A., M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. 2018. "Microscopic Traffic Simulation using SUMO". In *The 21st IEEE International Conference on Intelligent Transportation Systems*. Maui, Hawaii: Institute of Electrical and Electronics Engineers, Inc.

Lutz, R., R. Scrudder, and J. Graffagnini. 1998. "High level architecture object model development and supporting tools". *Simulation* 71(6):401–409.

Moradi, F., P. Nordvaller, and R. Ayani. 2006. "Simulation Model Composition Using BOMs". In *Proceedings of the 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, edited by E. Alba, S. J. Turner, D. Roberts, and S. J. E. Taylor, 242–252. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Nan, C., and I. Eusgeld. 2011. "Adopting HLA standard for interdependency study". *Reliability Engineering & System Safety* 96(1):149–159.

Nance, R. E. 1984, April. "A Tutorial View of Simulation Model Development". *SIGSIM Simul. Dig.* 15(2):16–22.

PTV AG 2021. "Vissim COM". Vissim 2021. [Embedded Documentation in Vissim 2021].

Rathnam, T., and C. J. Paredis. 2004. "Developing federation object models using ontologies". In *Proceedings of the 2004 Winter Simulation Conference*, edited by R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 1054–1062. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Sarli, J. L., H. P. Leone, and M. De los Milagros Gutiérrez. 2016. "Ontology-based semantic model of supply chains for modeling and simulation in distributed environment". In *Proceedings of the 2016 Winter Simulation Conference*, edited by T. M. K. Roeder, P. I.Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, 1182–1193. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Tolk, A., S. Y. Diallo, J. J. Padilla, and C. D. Turnitsa. 2012. "How is M&S Interoperability different from other Interoperability Domains?". *M&S Journal* 7(3):5–14.

Verbraeck, A. 2004. "Component-Based Distributed Simulations: The Way Forward?". In *Proceedings of the Eighteenth Workshop on Parallel and Distributed Simulation*, 141–148. New York, NY, USA: Association for Computing Machinery.

Wenguang, W., X. Yongpinq, C. Xin, L. Qun, and W. Weiping. 2009. "High level architecture evolved modular federation object model". *Journal of Systems Engineering and Electronics* 20(3):625–635.

Yilmaz, L. 2007. "Using meta-level ontology relations to measure conceptual alignment and interoperability of simulation models". In *Proceedings of the Winter Simulation Conference*, edited by S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 1090–1099. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Yu, L. C., J. S. Steinman, and G. E. Blank. 1998. "Adapting Your Simulation for HLA". *SIMULATION* 71(6):410–420.

AUTHOR BIOGRAPHIES

MORITZ GÜTLEIN received a diploma in computer science in 2017. He is a Research Assistant and PhD student at the Department of Computer Science 7, Friedrich-Alexander University Erlangen-Nürnberg. Besides mobility patterns, his research interests include multi-level simulation, and distributed simulation. His email address is moritz.guetlein@fau.de.

REINHARD GERMAN is a Full Professor at the Department of Computer Science, University Erlangen-Nürnberg, Germany. His research interests include analysis of interconnected systems based on numerical analysis, discrete-event simulation and network calculus as well as vehicular communications and smart energy application domains. His email is reinhard.german@fau.de.

ANATOLI DJANATLIEV is an Assistant Professor (AkadR) at University of Erlangen-Nürnberg. His research interests include topics on simulation and modeling using SD, DES, ABS, hybrid simulation, and methods for healthcare decision-support. Recently, he focuses on topics from the area of connected mobility and autonomous driving. His email is anatoli.djanatliev@fau.de.