Methodology

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
2015, Vol. 91(12) 1027–1051

DOI: 10.1177/0037549715614652
sim.sagepub.com
SAGE SCS

Component-based light-rail modeling in discrete event systems specification (DEVS)

Yilin Huang[1], Mamadou D Seck[2] and Alexander Verbraeck[1]

Abstract

Rail modeling and simulation is an effective decision support instrument for planning and designing complex rail infrastructures and operations. To successfully support these activities at a large scale, the simulation model should be sufficiently detailed and yet be computationally efficient. This poses a set of challenges pertaining to the design of the models. In this paper, we propose a component-based light-rail modeling and simulation library in the discrete event system specification (DEVS) formalism. The proposed library is described in detail and is shown to be efficient and scalable. We conclude the paper by offering a set of good design principles derived from this experience, which are also relevant to other types of large-scale infrastructure system simulation model design.

Keywords

model component, model design, light-rail transportation, discrete-event modeling

1. Introduction

Complex dynamics in railway systems pose many challenges in railway *modeling and simulation* (M&S). To successfully support detailed design and operation, a microscopic rail network model is often deemed not only suitable but also mandatory.[1] A railway model must have sufficient detail and accuracy to represent the complex and often large-scale system, yet still be computationally efficient, which is often hard to achieve when a microscopic model is large-scale. Apart from these criteria, considering the life span of railway systems, the number of M&S studies demanded and the time and cost induced, reusability and extensibility of railway models is often an issue that should be addressed during model design. In this paper, we present the design of a light-rail discrete-event microscopic model component library that supports composability and reusability. The study discussed in this paper showed that with comparable model detail and accuracy, our approach outperforms a classical model implementation using differential equations from a computational point of view.

The light-rail model components are defined with *discrete event systems specification* (DEVS).[2] A variety of modeling formalisms are presented in the literature. In traditional mainstream M&S paradigms, dynamic systems are delineated such that state changes of systems are specified on a continuous or discrete time base. (In M&S, a time base is often defined as a structure $(T, <)$ where T is a set and $<$ is a transitive, irreflexive, and antisymmetric ordering relation on elements of T, $\forall t \in T$; it is conceived as flowing along independently, and all dynamic changes are ordered by this flow.[2]) There is a third class of M&S formalisms based on the discrete-event world view where states (and actions, i.e. state-to-output mapping) are determined by internal and external events. Despite the wide use in practice, modeling approaches based on this world view were not formalized until a few decades ago:[3] much later than the former two classes of formalisms. The DEVS formalism[2] represents systems as piecewise constant state trajectories over a continuous time base. It is useful for several reasons. First, DEVS supports hierarchical component-based modeling.[2,4] It supports port-based modeling which has been proposed by a number of authors for its modularity.[5–7] Second, DEVS can embed and represent many other formalisms. It is a fairly general formalism that can play the role of root

[1]Section Systems Engineering and Simulation, Faculty of Technology, Policy and Management, Delft University of Technology, the Netherlands
[2]Engineering Management and Systems Engineering, Batten College of Engineering and Technology, Old Dominion University, VA, USA

Corresponding author:
Yilin Huang, Jaffalaan 5, 2628BX Delft, the Netherlands.
Email: y.huang@tudelft.nl

formalism[4,8–10] within the class of discrete-event formalisms (known as DEVS embedding[2]), and can be used to represent or approximate models using continuous and discrete time formalisms (known as DEVS representation[9]) by means of discretization (of time) or quantization (of state variables). Moreover, a basic motivation of discrete-event modeling in general is that it is intrinsically tuned to the capabilities and limitations of digital computers.[2] If carefully designed, discrete-event models can have high computational efficiency.

Modeling rail vehicle movement is the base element in vehicle running time estimation.[1,11] Other model parts formulate the boundary conditions (such as curvatures, control signals and speed restrictions) that influence vehicle movement. Different modeling paradigms can be used. Each has pros and cons in model detail, accuracy, modularity, and computational efficiency. A continuous abstraction of vehicle behavior is obtained by assuming a continuous time base and defining the rate of state changes (e.g. location, speed and acceleration) of a vehicle using differential equations.[1] A discrete time abstraction is obtained through the definition of a time-invariant iterative relation between the current state of a vehicle and the successive state after a predefined time interval has elapsed.[11,12] A discrete-event abstraction is obtained through the definition of events that trigger significant state changes of vehicle movement.[13–15] The choice of (the length of) the time interval or integration step for continuous or discrete time models is basically a trade-off between computational efficiency and simulation accuracy. Owing to a longer tradition, the continuous modeling style of vehicle movement appears to be the most intuitive. Although the discrete-event approach is often the most efficient among the three modeling styles,[16–18] it does not per se guarantee efficient simulation. It requires careful analysis and design of the simulation model. A good design of a discrete-event model of a continuous or hybrid system can render simulation more efficient than using continuous or discrete time representations.

For the light-rail models (or model components) presented in this paper, we made a number of design choices that address the issues we just highlighted. The resulting model component library is called LIBROS (*Library for Rail Operation Simulation*). We next discuss some modeling concepts and design choices related to high-level model design in Section 2. That section lays down the foundation of more detailed model design descriptions presented in Section 3. Vehicle and infrastructure dynamics are highly interdependent. It may thus be hard to explain one without understanding the other. This has led us to adopt an incremental approach in describing the models whereby aspects of each subsystem are presented in different places with increasing levels of detail across the paper. In Section 4, we present a study with test models developed with LIBROS and another continuous model with similar modeling detail and accuracy. Their computational performance is compared. Some applications of the LIBROS library are presented in Section 5.

2. Modeling concept

According to Pachl,[19] a railway system consists of three essential elements: (1) the *infrastructure* with the trackwork, the signaling equipment, the stations, and so on; (2) the *rolling stock* with cars and locomotives; and (3) the system of *operating rules and procedures* for a safe and efficient operation. This provides a guideline for how to decompose a railway system. A basic idea in designing the vehicle model is that movement is represented by a dynamic relation of a vehicle model to the infrastructure models where the vehicle is situated in a rail network. We also designed a communication mechanism where a vehicle model communicates with its environment (including other vehicles and the rail infrastructure such as sensors and control signals) and decides on its own movement accordingly. This allows for modeling *drive-on-sight* which is an important feature in tramway and light-rail operation. Many rail simulation tools exist, for example models of stations or terminals,[20–23] and train network simulators such as Simon/TTS,[24] TOPSim,[25] RailSys,[26] UX-SIMU,[27] VirtuOS,[28] SIMONE,[13] Multi-train simulator,[29] OpenTrack[30] and SimMETRO.[31] Yet very few (RailSys[32] to our knowledge) support modeling of urban railways such as tramways or light rail. Heavy-rail vehicles generally operate in *block systems* that provide strict safe spacing control by signaling, whereas light-rail vehicles also drive on sight.[33] (Non-signal-controlled modern railway operation—i.e. the communication between the dispatcher and the train crew is made via telephone or radio—is quite rare in Europe, and can only be found on branch lines with a very low density of traffic.[19]) In the latter case, modeling vehicle interactions is necessary as the vehicles do not only drive according to signals.[32] Given the large number of urban areas with light-rail systems and the increasing acceptance of M&S as a method of inquiry,[34,35] there is a growing need for tools that allow for light-rail modeling. This is the case with HTM, our project partner, a public transport organization that provides light-rail transport (and other public transport) services in the *Haaglanden* region in the Netherlands.

Since microscopic railway models are complex, often large-scale and take long to develop, we paid particular attention to composability, reusability and extensibility in design. It is not hard to argue that for a certain application domain, simulation models can be designed to be more easily reusable and extensible by means of model components.[36–38] Functional elements in a complex model that is aimed at reuse should be loosely coupled to one another.[39,40] (Loose coupling may refer to *separate*

domain from representation,[40,41] *separate modeling from simulation*, e.g. Zeigler et al.,[2] Wainer,[4] Banks,[42] Fujimoto,[43] etc. In this paper, we only discuss loose coupling among model components.) Although components should support composition,[44] it is acknowledged by many authors that model composability is difficult to apply.[45–47] Composability requires the adoption of principles of independence and controlled explicit dependencies.[48] In this regard, the DEVS formalism[2] provides strong support for composition with coupling constraints.

2.1. Modeling vehicles

The first modeling choice concerns whether to represent the vehicles as inputs and outputs of some other models or to represent them as models that have autonomous behavior. In the first case, the vehicles are represented as messages (or events) transmitted from one model (e.g. infrastructure) to another.[49,50] In the second case, they are models that can act on their own. We choose the latter for a richer vehicle representation (which eases modeling driving-on-sight) and for modularity. When vehicles are represented as messages, vehicle behavior may be mimicked by modifying the message content, which is dependent on the behavioral units (i.e. atomic models in the case of DEVS) that hold the messages. This would raise dependency as opposed to modularity. Furthermore, when we wish to change or extend the vehicle behavior, we would have to change the behavioral units instead of the vehicle ''itself'' which is counter-intuitive. A follow-up question is whether to model the locomotives and cars separately. We choose not to, because such a separation is not required by the intended model use. The vehicle model in LIBROS thus has ''an inseparable body''. Choosing a simple but modular design is a principle we used often in component design. Some other examples are how to choose units and boundaries of infrastructures, and whether and how to separate operating rules and infrastructures, some of which we will soon discuss.

2.2. Modeling infrastructures

It is common practice to model railway (or other transport) networks as directed graphs.[1] A choice here is whether to represent the rail network as a data structure or as simulation models. A data structure representation would be an infrastructure map or graph without behavior, while a simulation model has behavior. In the first case, a behavioral unit needs to hold the infrastructure map and perform appropriate actions. We may, for example, let each vehicle know the complete network, in other words:

(1) Each vehicle holds an infrastructure map and knows its own location. It announces itself to other vehicles so that it can be ''seen''.

This option (Figure 1 part (1)) results in strongly connected (vehicle) components that need broadcast-like

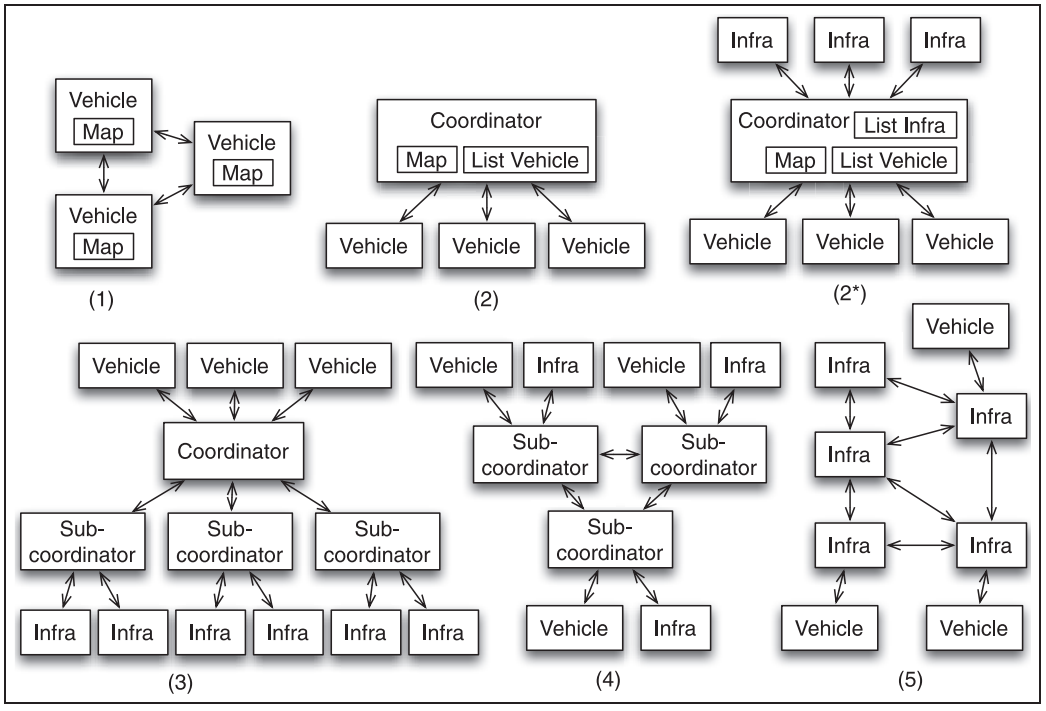

Figure 1. Modeling options for infrastructure models.

communications. The solution is not scalable with regard to map size and vehicle number. A direct improvement is to use a centralized solution:

2. A behavioral unit, let us call it a coordinator, holds the infrastructure map and maintains a list of vehicles and their locations. The coordinator communicates with the vehicles and informs them about the situation of their environment.

In this option (Figure 1 part (2)) all vehicles are connected to one coordinator instead of to each other. This reduces the communication cost (1: n as opposed to $n : n$). But the vehicle models are dependent on the coordinator and the latter is a singleton component with many duties. It needs to, for example, maintain the states and positions of the vehicles, find which vehicles may affect the others, and inform the potentially affected vehicles so that they may adapt their movement accordingly.

In both options, we have not yet mentioned that the infrastructure also has behavioral parts. Sensors, switches and signals are non-static. Sensors refer to different vehicle detection and track clear detection devices used in rail operations and controls.[19,51] Switches (also known as switchpoints or points) are movable track elements that are used to transfer rolling stocks from one track to another.[51] Signals indicate if a movement may enter the section of track behind (i.e. beyond) the signaling equipment.[19] They work together in some areas to safeguard vehicle movement. They also need to interact with vehicles. Similar to what is described in option 2, this interaction can be managed through the coordinator:

2*. A coordinator also holds a list of the behavioral infrastructure parts and their locations on the map. It manages the communications among these infrastructure parts and the vehicles.

This (Figure 1 part (2*)) results in a more complex singleton component which has to handle all communications among vehicles and dynamic infrastructure parts (and to detect communication relations among them). To reduce this complexity, we can use the principle of separation of concerns: divide and distribute the responsibilities, for example, to some sub-coordinators. A natural choice would be a geographical partition where a sub-coordinator is responsible for one partitioned area. The communications may still be handled through a coordinator or only by the sub-coordinators themselves. As such, we may have the following two options:

3. Each sub-coordinator holds a sub-map and manages a partitioned area. The sub-coordinators communicate with a higher-level coordinator who has a global view of the vehicle locations.

4. Each sub-coordinator holds a sub-map and manages a partitioned area. It has a local view of the vehicle locations. The sub-coordinators communicate with one another.

At first glance, the difference between the two options may be only in the degree of centralization. In option 3 (Figure 1 part (3)) vehicles communicate with each other through a coordinator (as in option 2 or 2*). Communications with the dynamic infrastructure parts are through sub-coordinators. (Without a central coordinator, we may directly connect each vehicle model to each sub-coordinator. But this setting of connections may be less desirable.) In option 4 (Figure 1 part (4)) the central coordinator is left out. Each sub-coordinator will locally handle the communications among the vehicles and the dynamic infrastructure parts within its area. When two vehicles in two areas need to communicate, the two corresponding sub-coordinators will handle the communications together. An important difference between the two options, however, is that option 3 has a static model structure while option 4 would require a dynamic model structure, which means that the model is designed to change its structure during simulation. In option 4, when vehicle models ''move'' in the infrastructure network, they need to be dynamically connected to the corresponding sub-coordinators.

The vehicle models in option 4 are connected to the sub-coordinators instead of one central coordinator, which is a less centralized solution than option 3. We may take this decentralization one step further by leaving out the sub-coordinators: vehicle models can be directly connected to infrastructure models. In the aforementioned options, the infrastructure network is represented by maps or sub-maps (i.e. data structures) in which only the behavioral parts are represented by infrastructure models. In other words, the infrastructure is represented as partly static and partly dynamic (which is indeed the case in reality). We can, however, represent the infrastructure solely with (dynamic) models to achieve uniformity in infrastructure representation. In this way, the connectedness of the infrastructure model parts represents the network structure (Figure 1 part (5)). As in option 4, the vehicle models can be dynamically connected to the infrastructure models in order to model their movement:

5. The infrastructure is represented by a network of infrastructure models. A vehicle model is dynamically connected to the infrastructure models on its moving trajectory to represent vehicle movement.

We are in favor of option 5 for four reasons. First, the uniformity in infrastructure representation increases cohesion and potentially improves model composability. Second, coordinators and sub-coordinators are left out,

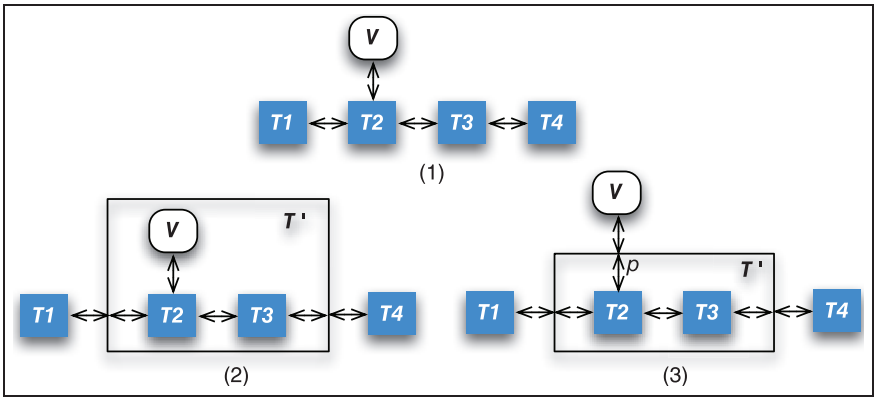

Figure 2. Vehicle and infrastructure model connections: a simple scenario of a vehicle *V* driving along a rail track composed of four successive track segments *T*1 to *T*4.

which reduces the number of artifacts that are constructed solely for the purpose of modeling. This potentially leads to simpler and more understandable morphisms. Third, decentralized communications allow for convenient modeling of autonomous behavior. Fourth, representing the infrastructure network as connected models permits flexibility in the sense that modelers may change model behavior by changing (infrastructure) model structure. Uniformity in model representation, model behavior autonomy and model structure flexibility allows for automated model generation as discussed by Huang.[52]

Infrastructure elements. The granularity of the infrastructure model is in accordance with a common sense decomposition of railway systems.[1,19,51,53] At its lowest description level, the infrastructure model is a network of *rail infrastructure elements* (RIEs) such as track segments, sensors, switches and signals (the last three are hereinafter abbreviated as 3S). RIEs can be used to recursively compose higher-level infrastructure components. (In DEVS, the former are specified in atomic models and the latter, in coupled models.) This recursive composition makes up an infrastructure network that has a (multilevel) *compositional containment hierarchy* (CCH). For example, a tram stop (or halting place) is composed of track segments and a sensor; an intersection (or crossing) is composed of track segments and 3S. The two composed models (the tram stop and the intersection) can further be used as components of larger models. In controlled areas such as an intersection, the 3S models need to communicate with one another: the signals (or signaling) need to be coordinated for safety control, during which the sensors in the area are for vehicle detection, and the switches change positions if necessary to allow vehicles to move from one track to another. This coordination (or control) is carried out by a *control unit* (CU) in the area. The use of locally centralized communications (and control) through the CU is a logical outcome of informed modeling rather than a deliberate design choice.

2.3. Modeling vehicle communications

A vehicle model needs to communicate with its ''environment'' to decide upon its actions. Option 5 in Section 2.2 presents the concept of representing the infrastructure as a network of infrastructure model parts, and connecting vehicle models dynamically to where they are located. This section addresses two issues in this design:

i how to connect a vehicle with an infrastructure model;
ii how does a vehicle communicate with another model?

Consider a simple scenario: a vehicle V drives along a rail track composed of four successive track segments $T1$ to $T4$ (Figure 2 part(1)). V can be connected to $T1$ to $T4$ at four time instances corresponding to the time needed by V to move from one segment to the next. (This time is calculated e.g. based on the length of the track segment and the vehicle's speed and acceleration.) Suppose that $T2$ and $T3$ together form a composed infrastructure model T'. Then V can be connected with $T2$ or $T3$ in two ways:

(1) place V in T' and connect V directly with $T2$ or $T3$ (Figure 2 part (2)), or
(2) place V outside of T', connect V with T' at port p, and connect port p with $T2$ or $T3$ (Figure 2 part (3)).

Which option is better? We cannot yet decide as such. Let us take a look at, in both cases, how a vehicle can communicate with another component. Suppose there is another vehicle V' connected to $T1$. Corresponding to the above two options, we can connect the two vehicles in two ways:

(1) connect V with V' at port q of T', as in Figure 3 part (1), or
(2) directly connect V with V', as Figure 3 part (2).

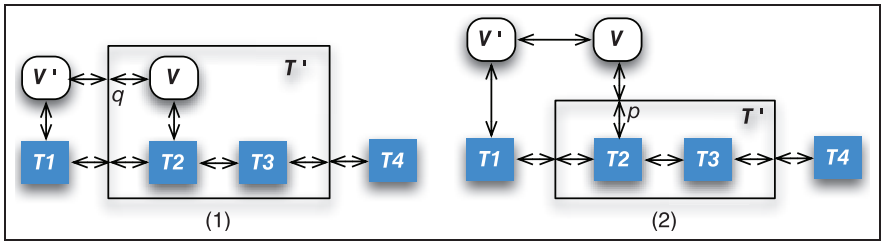

Figure 3. Vehicle and vehicle/infrastructure model connections: a simple scenario of an extra vehicle V'.

Suppose at the position of $T4$ there is a signal S instead. The connections of V and S will be similar to those of V and V'. Neither option is better in terms of ease of connection. Both have direct and indirect connections with model components depending on whether the components have the same parent model. (Two components are indirectly connected when there is more than one link connecting them. A link, or connection, directly joints two ports of two components.) As the vehicles can be anywhere in the network which has a CCH, we often need to connect (and disconnect) models that belong to different levels. These connections can be costly.

Vehicle models need to communicate with RIE models (vehicle-to-infrastructure, or V2I) and other vehicles (vehicle-to-vehicle, or V2V). The communications can focus on the most relevant, meaning that a vehicle model only needs to communicate with (1) its next closest RIE, and (2) its closest preceding and following vehicles, if any, within a certain distance. A vehicle's location is not static; neither are its connections to its next closest RIE and its preceding vehicle (PV) (the V2I and V2V connections). Our interest in reducing the cost of dynamic connections naturally raises the following question: *can we use static connections for vehicle communications*?

Setting up all possible connections beforehand is not an acceptable option. There is, nonetheless, an option worth considering. The infrastructure model is a statically connected network. When a vehicle model is connected to the network, it has in principle connections to any model that is connected to the network, despite the fact that the connections are mediated in the sense that there are RIE models in-between. As such, we can use the infrastructure network as the backbone of vehicle communications. Take Figure 3 part (1) as an example: V' is connected with V through $T1$ and $T2$. Hence, it is possible to establish communications between V' and V through the two intermediates without extra *dedicated* connections between V' and V. When V' sends a message to V or vice versa, $T1$ and $T2$ can pass on the message. By this principle, any vehicle can communicate with other vehicles and RIEs via the infrastructure network. We only need to dynamically connect each vehicle to the RIE model where the vehicle is located. Then it makes sense to directly connect a vehicle with a RIE, that is, connection option 1 (Figure 2 part (2) instead of Figure 2 part (3)). The number of dynamic connections, therefore, is equal to the number of vehicle models in the simulation. We call this mediated message-passing mechanism "message propagation" (MP).

2.4. Dedicated dynamic connections versus MP

How does MP compare to communications using *dedicated dynamic connections* (DDCs)? In the second case, point-to-point V2V and V2I connections are established when vehicles are to exchange messages directly with other vehicles or RIEs. This requires three steps: (1) given a vehicle model, search for its next closest RIE or PV; (2) if found, establish connections (or couplings) between the two models; (3) the two models can exchange messages.

In the case of MP, the existing static connections of the infrastructure network are used, and the messages are exchanged indirectly through the mediated RIE(s). New connections are not needed (step 2 above) between the communicating models. We can eliminate the search function (step 1 above) since the next closest RIE and PV can both be found during MP. The basic concept of MP can be generalized as following: (1) at a certain instant, when a given vehicle needs to know its next closest RIE and PV, it sends out a (request) message; (2) the message is forwarded by the RIE(s) along the vehicle's route; (3) once the next closest RIE or PV (if any) receives the message, it sends a (response) message back; (4) the message is forwarded by the RIE(s) back to the original vehicle. After such a round of MP, a given vehicle model receives at least one and at most two response messages. To achieve the same result, DDCs need to search for and set up connections twice. MP, however, introduces the overhead of message forwarding by the mediated RIEs: each message forwarding entails one extra model state transition.

In order to make an informed decision, the time and space complexities[54,55] of both designs are estimated (Table 1). (High-level designs can be concretized in many different ways even with the same modeling concept in mind. Our estimation is theoretical since no detailed model design is available at a high-level design stage.) The estimation of the communications cost uses one vehicle movement computation as one unit.

Table 1. Time complexity of V2I or V2V communications with DDCs or MP. The complexity denoted is per unit, which estimates how much total communication cost is required for one vehicle movement computation at one instant. The complexity in a whole simulation run raises over the total number of vehicles generated N_v and the number of RIEs n, for example the total time of the BFS in a simulation run is $O(N_v n \log n)$.

	Search	Connections		Communications	
	BFS	LCA	Setup	Message Sending	Transitions
DDCs	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log (n + N \log n))$	$O(\delta_c)$
MP	–	–	–	$O(\log n)$	$O(\delta_p \log n)$

Cost of DDCs. In DDCs, step 1 can be solved by a *breadth-first search* (BFS) in the infrastructure network starting from the RIE node where the vehicle is located. (The infrastructure network model, in terms of how it is connected, is a directed graph with RIEs as nodes. The infrastructure network model, in terms of how it is composed, has a CCH.) A BFS has linear time and space complexities; $T(f_{\text{BFS}}) = \text{O}(n)$ or $f_{\text{BFS}} \in \text{O}(n)$[55] where n is the number of nodes, that is, RIEs. (In the literature, a BFS runs in $\text{O}(n + m)$ time where n is the number of nodes and m is the number of edges in a graph. We consider it to be $\text{O}(n)$ as the infrastructure network is sparse, i.e. $n \sim m$.) Since the vehicles drive along predefined routes, we can improve the search cost to $\text{O}(\log n)$. To keep the discussion as general as possible, search improvement with bound constraints is not considered. Bound modifications are not always applicable and are different from case to case.

For step 2, setting up (or removing) a dedicated connection between two models requires knowing their parent. This can be solved by a *lowest common ancestor* (LCA) finding algorithm in the CCH of the infrastructure model. LCA has linear time and space complexities regarding the tree height.[56] The number of RIE nodes n in the infrastructure network is the leaf number in the tree of CCH (hereinafter called the *model composite tree* or MCT). The number of inner nodes in relation to the leaf number n in a full k-ary tree is $(n-1)/(k-1)$; hence the number of total nodes in a tree is $2n-1$ in the worst case, which has a tree height of $\log(2n-1)$. (A tree has a maximum node number of $2n-1$ when it is binary, i.e. $k = 2$, given a leaf number n.) The LCA, therefore, has a complexity of $\text{O}(\log n)$, that is, $T(f_{\text{LCA}}) = \text{O}(\log n)$.

The connection setup (or removal) basically has the same time and space complexities as LCA since it uses the (intermediate and final) result of the latter. To set up the connection we first need to create ports. The ports and connection themselves create extra space complexity of $\text{O}(\log n)$ for each connection. Because each vehicle always has one or two connections during its (simulation) lifetime, the total space complexity of the connections at a certain instant is $\text{O}(N \log n)$ where N is the vehicle number in a simulation at an instant. Note that N is smaller than N_v which is the total number of vehicles (generated) during a simulation run.

In step 3, message sending requires searching a list where the existing vehicle connections (or couplings) are stored. As just mentioned, the space complexity of connections is $\text{O}(N \log n)$. We hence estimate the time complexity of message sending to be $\text{O}(\log (n + N \log n))$. The space complexity of one message is constant: O(1). In DDCs, a message has only one receiver. Receiving a message triggers the state transition (δ_c; c stands for connection) of the receiver. Its cost $\text{O}(\delta_c)$ depends on the design of the receiver model (see discussions below).

Cost of MP. MP relies on indirect communications. A clear advantage is that extra dynamic connections are not required. Hence, the related search and connection costs for (re-)configuration are spared. Message sending itself is simple since one vehicle is only connected to one RIE at any instant. The time complexity of MP, however, grows with the distance, that is, the number of intermediary RIE models along the route, between the original sender and the final receiver; $\text{O}(\log n)$. For message passing, we have to impose an overhead on the intermediary RIE models. Upon receiving a message, each intermediary RIE model needs to forward the message. Once the message reaches its final receiver, be it a RIE or vehicle model, the receiver responds to the message and, if necessary, computes its state accordingly. In both cases, state transitions are required. We denote the transition function as δ_p (where p stands for propagation). An estimate of the total cost of the transitions is $\text{O}(\delta_p \log n)$ as each model along the route is involved in MP.

Costs of transitions δ_c and δ_p. From the perspective of a sender or receiver, the outcome of DDCs and MP (through functions δ_c and δ_p) should be comparable. In DDCs, a sender and a receiver communicate directly. In a modular design, they will exchange information about their own states; the situation between the two is not contained in the message per se. When the latter information is needed, extra communication is required. This entails extra an communication cost accounted for by δ_c.

In principle, the state transitions δ_p performed during MP will (a) fulfill the functionality of transitions δ_c in

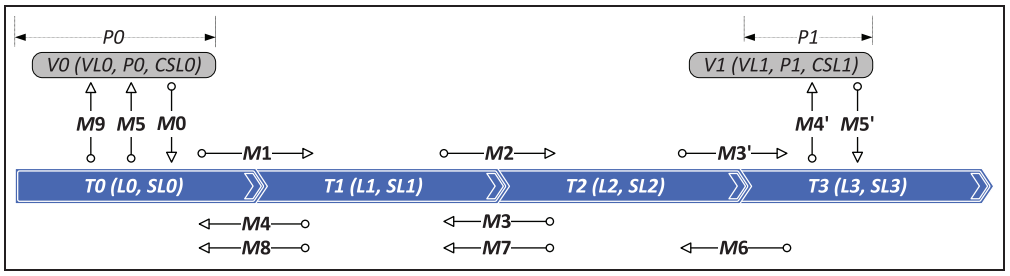

Figure 4. MP in Example 1.

DDCs, and (b) undertake the overhead Λ introduced by the use of MP. Each model along the propagation route can also participate as a communicating party rather than merely a messenger. Hence, δ_p can be assigned with tasks beyond simple message forwarding. Modelers may design rich types of behavior that intermediary models can perform so long as this behavior is conceptually consistent with the purpose of the model components. In LIBROS, the intermediary RIE models are designed to perform search functions and to enrich the information contained in a message.

MP can be a good choice when the function δ_c in DDCs can be replaced by simple tasks performed by $\delta_p{}'$ of the models in MP even after taking on the overhead Λ; that is, when $\mathrm{O}(\delta_c) \succeq \mathrm{O}(\delta_p \log n) \simeq \mathrm{O}((\delta_p{}' + \Lambda) \log n)$.

$\mathrm{O}(\delta_p)$ can be kept low by low $\mathrm{O}(\delta_p{}')$ and $\mathrm{O}(\Lambda)$, meaning that a good decomposition of function δ_c into $\delta_p{}'$ and a good design of the transition functions that handle the overhead Λ. In LIBROS, the transition function δ_p in RIE models have a constant time complexity, $\delta_p \in \mathrm{O}(1)$, for each message forwarding. This makes MP an attractive design option.

A joint consideration. We choose to use MP in LIBROS not only because it has a reasonable estimated cost in computation given our modeling context; it has advantages in terms of model design. First, indirect communications in MP result in a simpler model structure compared to that in DDCs. A direct outcome of this is that fewer couplings are needed. Second, simpler functions (without a BFS, LCA and connection setup) or the decomposition of large functions into simpler ones (in the case of δ_c and δ_p) means that fewer model components or smaller components are needed. Simpler components are easier to understand, develop, test and document. They are also easier to reuse and extend.

3. Model design

3.1. A communication mechanism: MP

MP is a mechanism for decentralized indirect communications in a connected network model.

Each RIE is capable of MP, which can be in the direction of the traffic or in the opposite direction. A vehicle model's main task in LIBROS is to correctly compute its movement based on the information about its next closest RIE and the PV it is approaching. If either type of information is absent before a movement, a vehicle sends a message forward which requests this information. (A message forward is propagated in the direction of the traffic.) The vehicle model's next closest RIE and PV shall respond to the message, which is then propagated back to the original vehicle. Obtaining the information, the vehicle is able to compute its movement trajectory to reach the next infrastructure. After reaching the RIE, a new iteration of this process starts and so on until the vehicle reaches its final destination.

MP is also used when vehicles or RIEs change their states and want to make the changes known to some other vehicles. In such cases, a model announces its state change by sending a message backward. (A message backward is propagated in the opposite direction of the traffic.) An approaching vehicle model can receive the message and change its movement accordingly when necessary.

Example 1. *Figure 4 illustrates a simple model composed of four successive track segments (T0 to T3) and two vehicles (V0 and V1). Each segment has a length (L0 to L3) and a speed limit (SL0 to SL3). Each vehicle has a length (VL0 and VL1), a position (P0 and P1) on the track segment it is coupled with, and its current speed limit (CSL0 and CSL1). (A vehicle's position on a track segment is defined as the distance between the vehicle's front end and the start node of the segment.) Suppose V0 is on T0 and V1 is on T3, in other words, V0 is behind V1 given that the direction of the traffic is from left to right; V0 does not have the information about its next infrastructure or its PV.*

3.1.1. Request and response messages. In Example 1, $V0$ will send a message forward ($\overrightarrow{M0}$). (We use an arrow to indicate forward direction ($\rightarrow$) in the case of a request message or backward direction ($\leftarrow$) in the case of a response message.) Two MP sequences are generated upon this action: (1) $\overrightarrow{M0}, \overrightarrow{M1}, \overrightarrow{M2}, \overleftarrow{M3}, \overleftarrow{M4}, \overleftarrow{M5}$, and (2) $(\overrightarrow{M0}, \overrightarrow{M1}, \overrightarrow{M2},) M3', M4', M5', \overleftarrow{M6}, \overleftarrow{M7}, \overleftarrow{M8}, \overleftarrow{M9}$.

Table 2. Distance accumulation in the response of MP sequence (2) in Example 1.

Sender	Distance Accumulation	Message	Distance Value (d) in Message	Receiver
1. $V1$	$d \leftarrow P1 - VL1$	$\overleftarrow{M5'}$	$P1 - VL1$	$T3$
2. $T3$	d	$\overleftarrow{M6}$	$P1 - VL1$	$T2$
3. $T2$	$d \leftarrow d + L2$	$\overleftarrow{M7}$	$P1 - VL1 + L2$	$T1$
4. $T1$	$d \leftarrow d + L1$	$\overleftarrow{M8}$	$P1 - VL1 + L2 + L1$	$T0$
5. $T0$	$d \leftarrow d + L0$	$\overleftarrow{M9}$	$P1 - VL1 + L2 + L1 + L0$	$V0$
6. $V0$	$d \leftarrow d - P0$	–	$P1 - VL1 + L2 + L1 + L0 - P0$	–

To reduce the frequency of MP, we defined the following rule: a RIE model responds to a request message only when the RIE requires or potentially requires a change in movement of the requesting vehicle. We call such a RIE the *next closest RIE of interest* (NCRI) of a vehicle. Some examples of NCRIs are: (a) the RIE has a defined speed limit which is different to the current speed limit of the requesting vehicle; (b) the RIE is a signal; and (c) the RIE is (or is in) a stop, a station, a terminal, and so on.

For MP sequence (1), we assume that $(SL1 = \infty \vee CSL0 = SL1) \wedge (SL2 \neq \infty \wedge CSL0 \neq SL2)$. It means that the speed limit of $T1$ is undefined or is the same as the current speed limit of $V0$, and the speed limit of $T2$ is defined and is not the same as the current speed limit of $V0$. Therefore, $T1$ does not respond to the request message but propagates it forward ($\overrightarrow{M2}$); $T2$ responds to the request by a backward message $\overleftarrow{M3}$ which is propagated back to $V0$ through $T1$ and $T0$ ($\overleftarrow{M4}$ and $\overleftarrow{M5}$).

While responding to the request from $V0$, $T2$ also propagates the request forward to $T3$ ($\overrightarrow{M3'}$) because $V0$'s PV has not yet been found. This generates MP sequence (2). For MP, two other rules are defined: (1) a RIE model forwards a request message to the vehicle model closest to its start node when there is any vehicle coupled with it; and (2) when a vehicle model receives a request message, it responds unconditionally. In Example 1, $T3$ has $V1$ coupled with it, so it forwards the request to $V1$ ($\overrightarrow{M4'}$). In response, $V1$ sends $\overleftarrow{M5'}$ addressed to $V0$, which is propagated back to $V0$ ($\overleftarrow{M6}$ to $\overleftarrow{M9}$). $T3$ then stops propagating the message because both the NCRI and the PV of $V0$ have been found. This concludes this round of MP initiated by $V0$ sending out a request message $\overrightarrow{M0}$ and ended by the two responses $\overleftarrow{M5}$ (from $T3$, $V0$'s NCRI) and $\overleftarrow{M9}$ (from $V1$, $V0$'s PV).

3.1.2. Distance accumulation in messages. The distance between the original sender and the receiver is accumulated by the vehicle and RIE models during MP. Table 2 gives the steps of distance accumulation in Example 1.

In response to $V0$'s request, message $\overleftarrow{M5'}$ is created by $V1$, addressed to $V0$, and sent to $T3$. When a vehicle creates a backward message, the distance d is set to the value of the vehicle's position in a RIE model with the vehicle's length deducted ($\overleftarrow{M5'}$ has $d = P1 - VL1$). When a RIE model receives a backward message, it does not change the value of the distance if the original sender is a vehicle that is coupled to the RIE; otherwise the RIE increases the value by its own length. Thus, $T3$ does not change d in $\overleftarrow{M6}$, while $T2$, $T1$ and $T0$ add their own lengths to d in the three successive messages ($\overleftarrow{M7}$ to $\overleftarrow{M9}$). $T0$ forwards $\overleftarrow{M9}$ to $V0$ which concludes the backward propagation. $V0$ holds the response $\overleftarrow{M9}$ with $d = P1 - VL1 + L2 + L1 + L0$, the distance from the rear end of $V1$ up to and including $T0$. When $V0$ needs to know its distance from $V1$, it deducts its own position on $T0$ from d.

3.1.3. Update messages. Suppose that, in Example 1, at a certain instant, $V1$ has a state update. In order to announce this, $V1$ creates an update message, which will be propagated backward to $V0$ which is driving behind it. An update message is created, without an addressed recipient, by a vehicle or RIE model straight after a state change, and is propagated backward to the vehicle model closest to the original message sender within a predefined distance bound.

Remark. The design of MP (and distance accumulation) supports a good level of *modularity* and *composability* of the model components in Libros. We deem the design modular in the sense that each (vehicle or RIE) model only uses its local information (viz. the state variables of the model component itself) and the information contained in the messages it receives to determine its own behavior. The direct dependency among the (vehicle and RIE) models is limited to the *couplings* designated for MP and to the *messages* being propagated. Simple MP rules are defined for cohesive model communication so that messages can be created, understood, manipulated and routed by the communicating parties. The models in Libros can be composed to work together in a modular manner without prior knowledge of the infrastructure model layout because of each model's observance of the communication protocol (or rules). Note that MP is performed in Libros with zero (simulation) time advance.

3.2. An overview of infrastructure models

The infrastructure models in Libros are characterized at a high level by whether the model is atomic or coupled, and

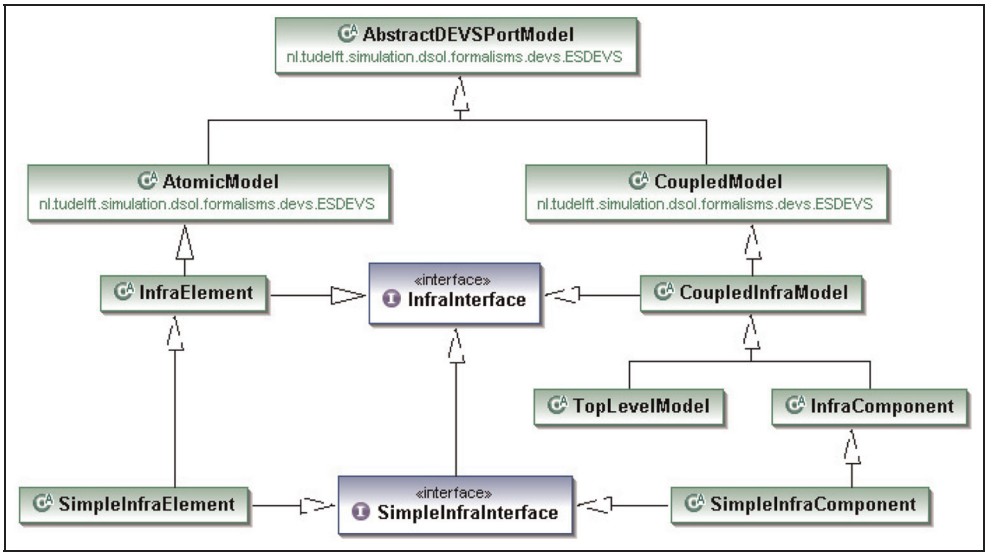

Figure 5. High-level infrastructure model classes in LIBROS.

whether it only has one inflow and outflow of traffic, or more. The former is related to the modeling formalism, and the latter to the domain specificity. This orthogonal abstraction is designed as multiple inheritance.[57,58]

3.2.1. High-level infrastructure model classes. The RIE models are defined as `InfraElement` and the composed infrastructure models are defined as `CoupledInfraModel` (Figure 5) by the respective specialization of the `AtomicModel` and `CoupledModel` classes provided by the Event Scheduling DEVS (ESDEVS) library (Section 4). Both infrastructure classes also implement `InfraInterface` which defines a number of common functions mainly concerning operations on the model ports and couplings.

`CoupledInfraModel` is used to define domain meta-models. It has two specializations. The root infrastructure model is declared as a (singleton) `TopLevelModel` with no parent model. (A parent model is a coupled model that contains a child model as a sub-component.) It serves as a container for the simulation model to be constructed (manually or automated) interfacing with the simulator. The second specialization of `CoupledInfraModel` is `InfraComponent`. It is for defining meta-models of infrastructure compositions that are meant to be *component units* for model construction. By component units we mean that the sub-components in an `InfraComponent` (as an unit) are strongly related such that their composition often reoccurs and can be reused as an assembled whole. An `InfraComponent` is defined for the convenience of assembling models and their reuse. `InfraElement` and `InfraComponent` each have a specialization that implements `Simple InfraInterface` (`SimpleInfraElement` and `SimpleInfraComponent`). The prefix Simple- indicates that the component being addressed only has one inflow and outflow of traffic. In other words, the component is not a switch or has no switch as a sub-component.

3.3. Vehicle model

In the literature, there are two approaches that support modeling of continuous and hybrid systems using the discrete-event paradigm and the DEVS formalism in particular: quantized DEVS (QDEVS) and generalized DEVS (GDEVS).[4,59] *Quantization* discretizes (or quantizes) the state space rather than the time base as in *discretization*. The principle is similar to that of an analog-to-digital converter.

GDEVS,[17] on the other hand, is an approach that supports discrete-event modeling of continuous and hybrid systems.[4] It is more general than classic DEVS such that the state trajectory segments can be polynomial instead of constant. The coefficients of the polynomial segments are piecewise constant. As such, an event in GDEVS can be considered as a coefficient event that activates or potentially activates a change of at least one of the coefficients.[60] The choice between QDEVS and GDEVS is in principle a choice between continuous and discrete-event modeling styles. GDEVS allows modelers to decompose a modeling problem into less complex ones and express them in polynomials which otherwise have to be expressed as a whole in differential equations. This is an advantage both for modeling and for model understandability. In order to model in GDEVS, modelers have to know how a system responds to input events which are deemed significant. This can be seen as a disadvantage as the information

is not always available or complete.[4] We use GDEVS for vehicle movement modeling. Vehicle movement can be conveniently organized through piecewise polynomial segments where state changes occur when a vehicle changes its acceleration. The events that correspond to such state changes are well understood and can be rigorously defined.

3.3.1. Modeling vehicle movement. Since our model does not require kinematics or tractive force considerations, we apply equations of motion in polynomials with constant acceleration. (Relevant formulas can be found e.g. in Hansen and Pachl,[1] pp. 61–80.) Given velocity v_i and its assumed constant acceleration a_i at time t_i, a vehicle's velocity v_j and movement distance s_i after a time interval $\langle t_i, t_j \rangle_{t_i \leqslant t_j}$ are $v_j = a_i \Delta t + v_i$; $s_i = (v_i + v_j)\Delta t/2$; $(\Delta t = t_j - t_i)$, where Δt or t is the variable, and v_i and a_i are the coefficients that are constant in each polynomial segment. When a vehicle completes a certain movement in $\langle t_i, t_j \rangle$, the value of its v_j or the new v_i value for the next polynomial segment is determined. It still has to decide upon the new a_i value for the next segment as well as the anticipated time (let us call it $\Delta t'$ or t') of the next segment. In order to do so, a vehicle sends a request message as described in Section 3.1, and it will receive response messages by NCRI and PV, if any, in zero simulation time (Figure 6). (The information includes a vehicle's distance to its NCRI d_{NCRI}, the speed limit or restriction l_{NCRI} required by the NCRI, and if applicable a vehicle's distance to its PV d_{PV}, and the velocity v_{PV} and acceleration a_{PV} of the PV.) Then, a vehicle model can compute its a_i and t' (or its movement) to a position so far forward that safe driving is assured. Ideally, it *successively* computes the movement to each NCRI it approaches.

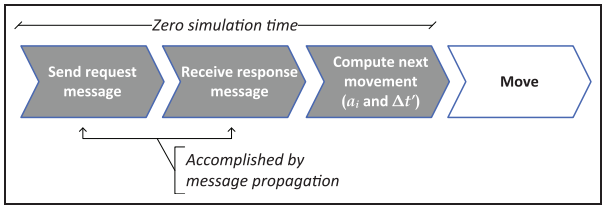

Figure 6. The action steps in one round of movement with message request.

Example 2. *At time t_i, position p_i, vehicle V* has velocity v_i, acceleration a_i *(Figure 7). After MP, V knows its NCRI is at distance d_{NCRI}, position p_j with speed limit l_{NCRI}.*

V can therefore compute its movement until the NCRI at position p_j, say at time t_j. Because the value of v_i, the current speed limit l (i.e. the previous l_{NCRI}) and the current l_{NCRI} are not necessarily the same, the movement until the NCRI (from p_i to p_j) can consist of more than one polynomial segment where acceleration a changes. In Figure 7, two movement trajectories from p_i to p_j are shown for two different cases.

(1) If $v_i < l < l_{\text{NCRI}}$, the movement would consist of two polynomial segments, that is, (1) $a > 0$, (2) $a = 0$.
(2) If $v_i < l > l_{\text{NCRI}}$, the movement would consist of three polynomial segments, that is, (1) $a > 0$, (2) $a = 0$, (3) $a < 0$.

In the first case, a changes one time from a positive value to zero so that v remains constant afterwards until p_j. In the second case (shown in gray), a changes once more and becomes negative so that v reduces to l_{NCRI} at the moment when V reaches p_j. Once V is at p_j time t_j, the vehicle is dynamically coupled to the current NCRI instead of the previous NCRI to represent its changing of position. This starts the next round of MP and movement.

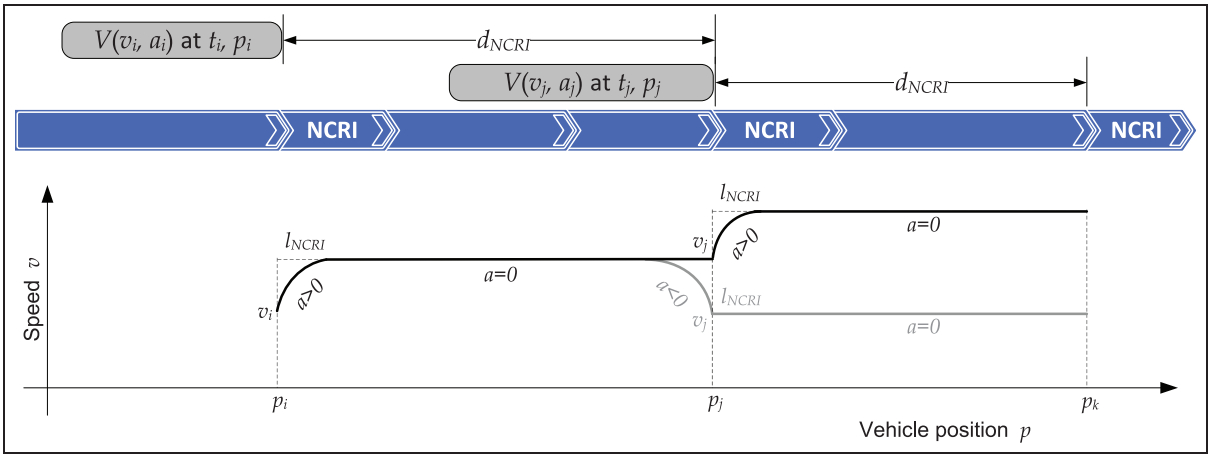

Figure 7. Movement computation with NCRI.

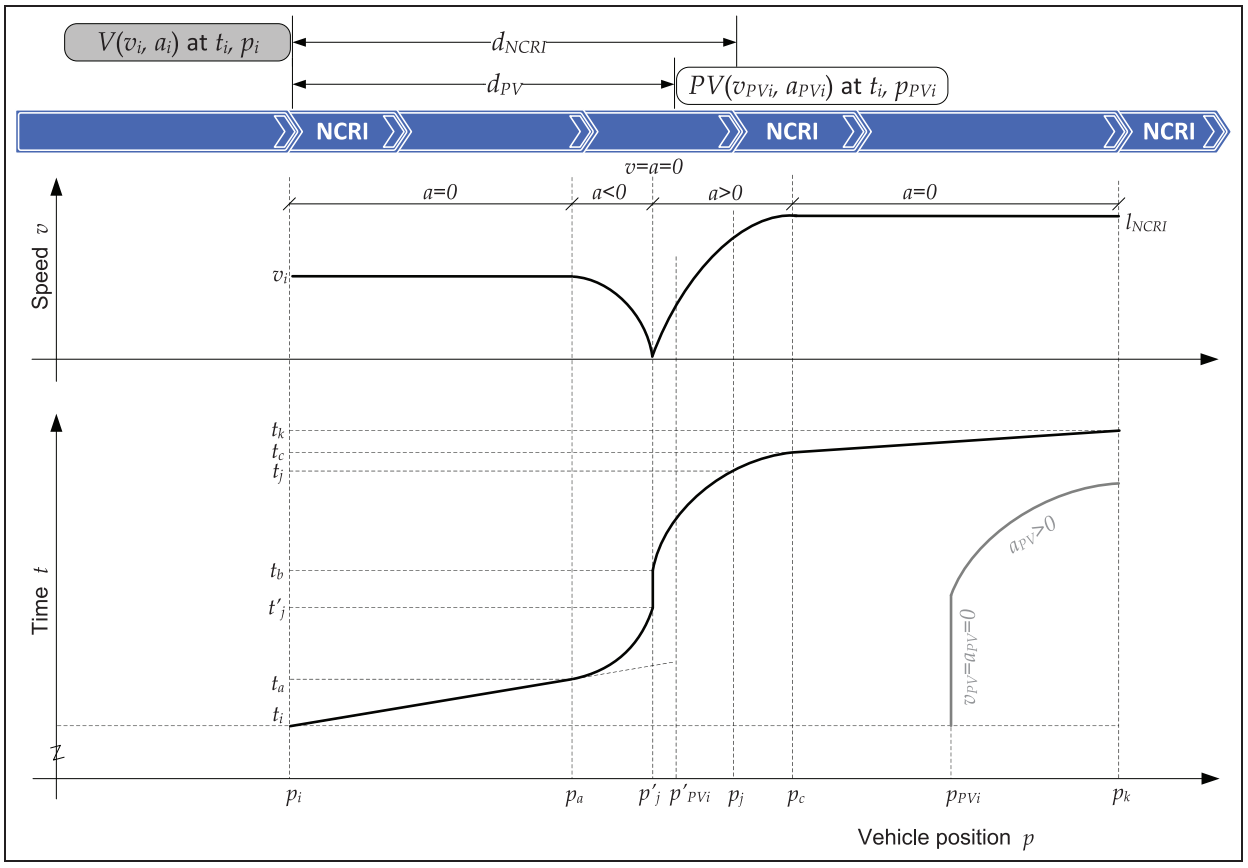

Figure 8. Movement computation with NCRI and PV.

Example 3. *Suppose V in Example 2 has a PV at distance* d_{PV} *at time* t_i *(*Figure 8*). The PV is not moving* ($v_{\text{PV}i} = a_{\text{PV}i} = 0$), *and is closer than NCRI* ($d_{PV} < d_{NCRI}$).

In this case, the computable movement to NCRI needs to be shortened. The distance d_{NCRI} has to be divided into parts. Each is to be computed separately. How V can move towards its NCRI depends on how the PV moves if the latter is close enough. In Example 3, V has to stop behind the PV: at time t_i, V can compute its movement no further than the PV's rear end position $p'_{\text{PV}i}$. The stop position is with some safety distance to $p'_{\text{PV}i}$, shown as p'_j. During the movement, V must stop in time: it starts to brake at p_a, time t_a, and after a time interval $\langle t_a, t'_j \rangle$ it stops at p'_j. Hence, the movement has two segments:

(1) in interval $\langle t_i, t_a \rangle$, V moves with a constant speed v_i ($a = 0$) from p_i to p_a;
(2) in interval $\langle t_a, t'_j \rangle$, V moves with a decreasing speed ($a < 0$) from p_a to p'_j where its end speed reaches zero.

When V can drive again, it still needs to compute its movement to the NCRI according to the new situation. Suppose that at t_b the PV starts driving and it drives fast enough that V can drive freely without considering the PV. In principle, V can accelerate ($a > 0$) up to the speed restriction l_{NCRI}. If it does so, the acceleration will last for a time interval $\langle t_b, t_c \rangle$, starting at position p'_j and ending at p_c. Suppose that the position of NCRI p_j is closer than the acceleration distance. The movement can only be computed up to p_j where V has to be coupled to the NCRI at the time when V reaches it. This movement therefore has one segment: in the interval $\langle t_b, t_j \rangle$, V moves with increasing speed ($a > 0$) from p'_j to p_j. Once V arrives at p_j, it is dynamically coupled to the NCRI. The next round of MP starts.

Remark. The above two simple examples elaborate on how a vehicle model can successively compute its movement (which can consist of a number of piecewise polynomial segments) to its NCRIs in a discrete-event manner taking into account the movement of its PV. In essence, a vehicle computes the (internal) events of two groups of actions: (i) when it will move from one NCRI to the next, and (ii) when it will change its acceleration rate.

A vehicle model locally decides on its actions according to the information at hand. As explained in Section 3.1, when a vehicle model changes acceleration, it sends out an

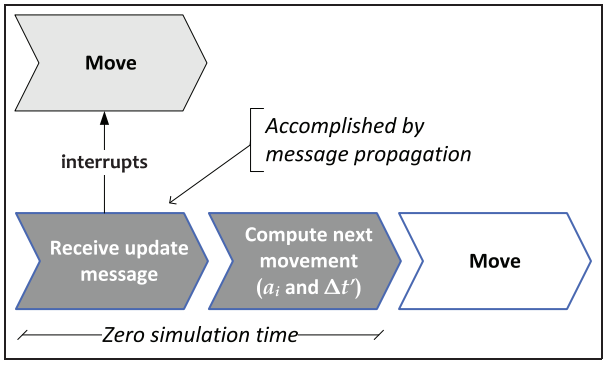

Figure 9. The action steps in one round of movement with an update message.

update message (backward) so that a following vehicle, if any, can be informed of this change. Consequently, the (external) events of (the arrival of) update messages can be expected at any simulation time. This means that the execution of an anticipated movement may be interrupted by an update message from a PV, say at time t_{ext}. If so, the movement after t_{ext} has to be computed anew (Figure 9).

In this context, ''movement'' or ''move'' is used in a general sense: it refers not only to a vehicle's moving actions but also to non-moving actions such as scheduled or unscheduled waiting. (An unscheduled wait has an undefined waiting time which can be interrupted, while a scheduled wait has a minimum waiting time which cannot be interrupted.)

Referring back to the waiting between $\langle t'_j, t_b \rangle$ in Example 3, from time t'_j onwards V is in an unscheduled wait because its PV is on halt. In this case, V is in passivity (or a passive state). When the PV starts driving some time after t'_j, V receives an update message from the PV of this event, which activates V to compute its movement according to the new situation. Time t_b, that is, when it restarts driving, is V's decision. It can, for example, wait until the PV has driven some distance.

When an external event arrives, it interrupts and can change the anticipated state variable trajectories. Such interrupts often occur in the simulation of discrete-event models and need to be treated with particular care when using piecewise polynomial abstractions.

3.3.2. Handling external events. A state variable expressed by polynomial segments is not meant to be constant. Its values are only computed for chosen (external and internal) events. When an external event (or interrupt) arrives, a state variable still holds an old value that was updated at the past event. Thus, the value must be updated for the present time.

Example 4. *Figure 10 depicts an example of state variable update in the vehicle model. The two state variables being updated are the speed v and the position p.*

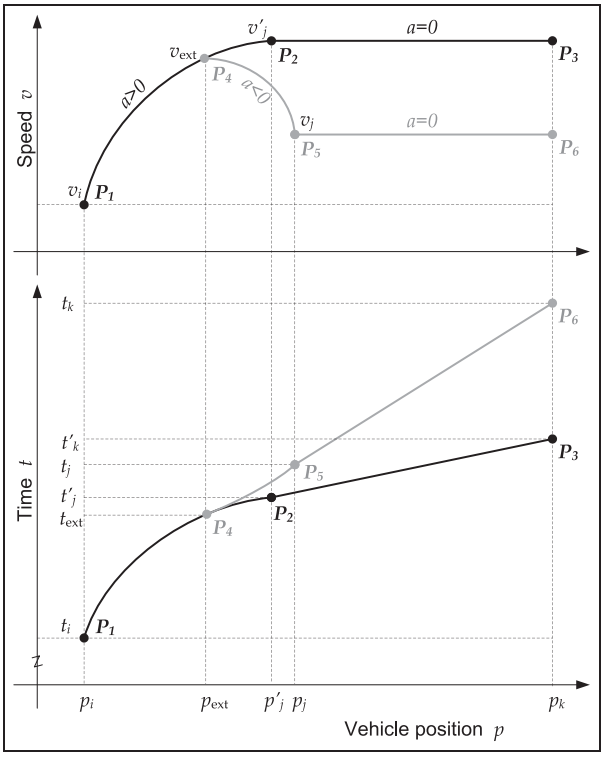

Figure 10. State variable update of speed v and position p in the vehicle model with an update message at t_{ext}.

An anticipated movement is computed from position p_i to p_k. It supposedly takes place in time $\langle t_i, t'_k \rangle$ and consists of two segments shown with black lines: (1) in interval $\langle t_i, t'_j \rangle$, V moves with an increasing speed ($a > 0$) from p_i to p'_j in which the initial speed is v_i and the final speed is v'_j; (2) in interval $\langle t'_j, t'_k \rangle$, V moves with a constant speed v'_j ($a = 0$) from p'_j to p_k. Suppose that, during the first anticipated segment of $\langle t_i, t'_j \rangle$, an update message arrives at time t_{ext}. It diverges the movement trajectory away (shown with gray lines) from the anticipated movement.

As in the previous examples, only a number of chosen points (or values) on the polynomial trajectories are computed. They are at the start (the start point is computed by the previous computation) and the end of a trajectory, and where the segments change coefficients (i.e. only the dots, not the lines). The values of the three black dots (•) are first computed. They are $P_1 = (t_i, v_i, p_i)$, $P_2 = (t'_j, v'_j, p'_j)$, and $P_3 = (t'_k, v'_j, p_k)$ and pinpoint the anticipated movement trajectory $P_1 \xrightarrow{a>0} P_2 \xrightarrow{a=0} P_3$. If no external event arrives during time $\langle t_i, t'_k \rangle$, the vehicle will move according to the plan. In this case, two internal transitions are scheduled successively at time t'_j and time t'_k. If an external event arrives during $\langle t_i, t'_k \rangle$, say at t_{ext}, the (anticipated) movement is interrupted and the vehicle's state variables must be first updated for the present time t_{ext} using the corresponding known polynomials for the time interval $\langle t_i, t_{\text{ext}} \rangle$. If the event contains an update message for the vehicle,

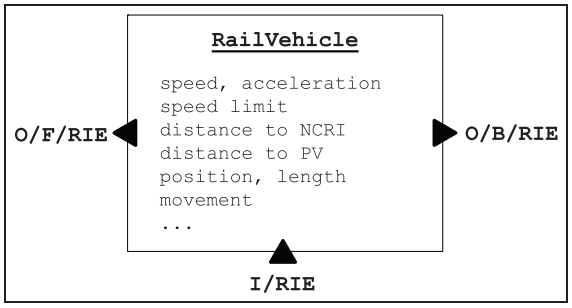

Figure 11. Rail vehicle model.

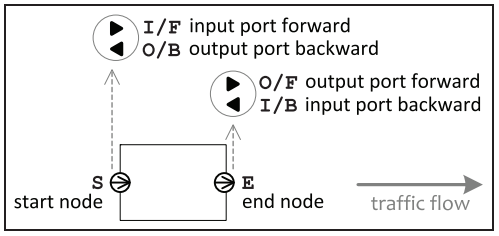

Figure 12. Start and end nodes of infrastructure models.

further movement shall be computed anew; if the event contains a request message for the vehicle, it only needs to reply to the message.

In Example 4, when the external event arrives, the state variables still hold the values of P_1 which have to be updated to those of P_4. These values are subsequently used as the initial values for the new movement computation. Suppose that given the content of the update message, further movement is computed, and we obtain discrete state variable points $P_5 = (t_j, v_j, p_j)$ and $P_6 = (t_k, v_j, p_k)$. The new movement $P_4 \overset{a<0}{\longrightarrow} P_5 \overset{a=0}{\longrightarrow} P_6$ would take place in time $\langle t_{\text{ext}}, t_k \rangle$. It is also an anticipated movement which may be interrupted by an external event. If no external event arrives during this time, the total movement trajectory from position p_i to p_k would be $P_1 \overset{a>0}{\longrightarrow} P_4 \overset{a<0}{\longrightarrow} P_5 \overset{a<0}{\longrightarrow} P_6$ where the transition of P_4 is activated by an external event, and the other three, by internal events.

3.3.3. Rail vehicle model. A `RailVehicle` model has three ports (Figure 11): (1) `I/RIE`, an input port that receives messages from a RIE; (2) `O/F/RIE`, an output port that sends messages forward to a RIE, and (3) `O/B/RIE`, an output port that sends messages backward to a RIE. All three ports are coupled to the RIE model where the vehicle is located. A vehicle model's state is composed of its own state and the information it has at hand about its NCRI and PV.

3.4. RIE models

There are mainly four types of atomic RIE models in Libros: (1) tracks are of type `TrackSegment`, (2) sensors are of type `Sensor`, (3) signals are of type `LinesideSignal`, and (4) switches are of type Point (switches are often called *points* in North American). They all participate in MP. Because MP can be in the forward and backward directions, for the convenience of model construction (and generation), we created type `Node`.

3.4.1. Nodes of infrastructure models. A `Node` is a bundle of one input port and one output port dedicated to MP. It has a direction that indicates the inflow/outflow of traffic at an (atomic or coupled) infrastructure model (Figure 12): (1) a `StartNode` is a `Node` composed of one input port for messages forward and one output port for messages backward, and (2) an `EndNode` is a `Node` composed of one output port for messages forward and one input port for messages backward.

The port design of the RIE models is illustrated in Figure 13. The labels of the ports have the following meanings:

(1) `O/V`: an output port that sends messages to a vehicle model;
(2) `O/C`: an output port that sends messages to a control unit model;
(3) `I/C`: an input port that receives messages from a control unit model;
(4) `S`: a start node (at inflow of traffic) for MP in both directions; and
(5) `E`: an end node (at outflow of traffic) for MP in both directions.

An `O/V` port is on each RIE model (a) to (f). An `O/C` is on the 3S models (b) to (f). In addition, a signal (c) and a facing point (FP) (d) each have an `I/C` port because they are instructed by a CU (Section 3.4.4). Each RIE model has at least one start node `S` and one end node `E`. The node-to-node coupling is a one-to-one relation. Note that the RIE-to-vehicle couplings allow one-to-*n* relations. Thus, when a RIE sends a message to a vehicle, the recipient is addressed. The same applies to the control-to-RIE couplings.

A RIE model has, among others, a length and a speed limit (3S models are of length zero). 3S models have roles similar to `TrackSegments` in MP, but they always respond to request messages (except when the requester is already coupled to it). Besides MP, 3S models can be triggered and released by a vehicle model.

3.4.2. Vehicle detection. The detection of occupation and clearance of tracks is fundamental to railway operation and control.[19] In real life, different technologies are used for many types of detection devices.[51] 3S models are designed to have detection capacity: (1) the Sensor for

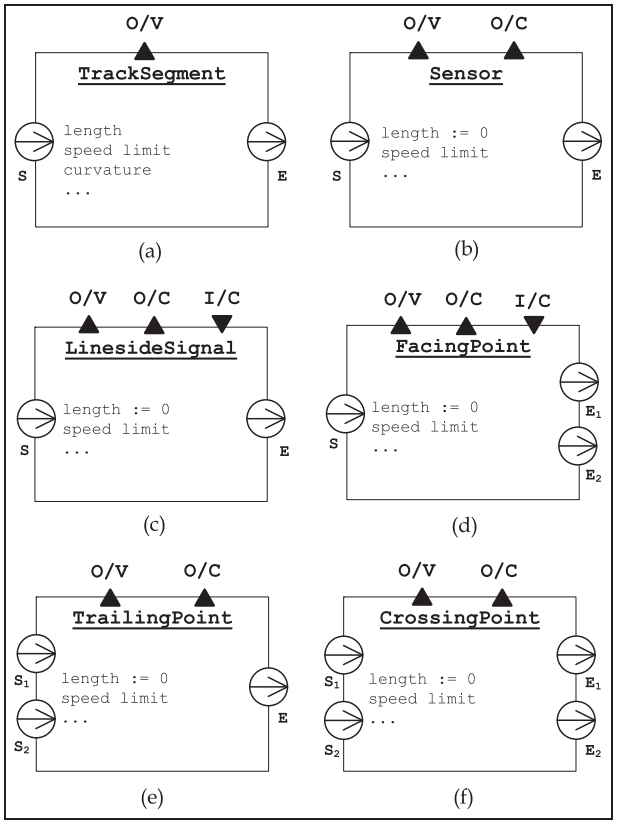

Figure 13. RIE models. With a FP, the inflow traffic at S exits at E_1 or E_2 depending on the position of the point which can be toggled by a CU. With a CP, the inflow traffic at S_1 only exits at E_2, and the inflow traffic at S_2 only exits at E_1. A CP models a location where two rail tracks cross each other.

detection, (2) the `LinesideSignal` for detection and signaling, and (3) the Point(s) for detection and switching (if applicable). (In real operation, the latter two types do not have detection capacity per se, but detection devices are placed around them.)

Three important detection purposes are: (i) detect a vehicle reaching a certain point with its *front end*, (ii) detect a vehicle passing a certain point with its *rear end*, and (iii) detect track occupancy.[51] With the 3S models, purpose (i) is fulfilled by *triggering* a detector, purpose (ii), by *releasing* a detector, and purpose (iii), by two detectors at the two ends of a track. The 3S models can be coupled with the track segment models in different combinations to form different rail infrastructure layouts.

3.4.3. Trigger and release of detectors. When a vehicle model is approaching a detector, the latter responds to the former's request message. The vehicle then computes its movement up to the detector (which is its NCRI). After the movement, the vehicle couples to the detector and initiates another request to find the next NCRI. When the detector receives the message, it propagates it. The message also

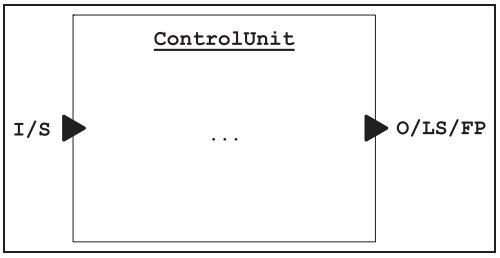

Figure 14. CU model.

triggers the detector which computes the release time (based on the vehicle's length, speed and acceleration) which schedules the detector's next internal transition. Before the release time expires, if the vehicle changes its acceleration, it will send an update message. When the detector receives the message, it recomputes the release time with the new information. If the detector receives other external events, it responds accordingly and then resumes the release event. A detector's subsequent action upon a trigger or release is typically to send out a message to inform a CU.

3.4.4. CU. A CU models the control logic of an area where the entrances are guarded by signals. When detectors obtain information about vehicle positions, they send the information to CUs, which evaluate the information and permit vehicle movement via signals.[51] `ControlUnits` in LIBROS receive input events only from detectors and send messages to `LinesideSignals` and `FacingPoints`. A `ControlUnit` model has two ports (Figure 14): (1) `I/S`, an input port that receives (sensor) messages from a sensor (i.e. detector); (2) `O/LS/FP`, an output port that sends (control) messages to a lineside signal or a facing point.

A crossing with CU. Suppose we need to model a simple tram crossing: Figure 15(a).

Example 5. *A Y-crossing is guarded by three lineside signals (LS_1 to LS_3). The crossing has three switches: a trailing point (TP), a FP, and a crossing point (CP). Four directions (or routes) of traffic are possible: (1) $A \rightarrow F$, (2) $B \rightarrow F$, (3) $C \rightarrow D$, and (4) $C \rightarrow E$. The control logic is common sense: when a track or switch is occupied, it cannot be accessed by others.*

The couplings of 3S and CU at the Y-crossing are shown in Figure 15(b). Note that the inputs or outputs to a CU are coupled with only one input or output port of the CU. An input event from a 3S model to a CU contains information about sensor trigger or release. In addition, a signal can send a request to access the crossing to a CU on behalf of an approaching vehicle. An output event from a CU to a signal takes place when the requested access is

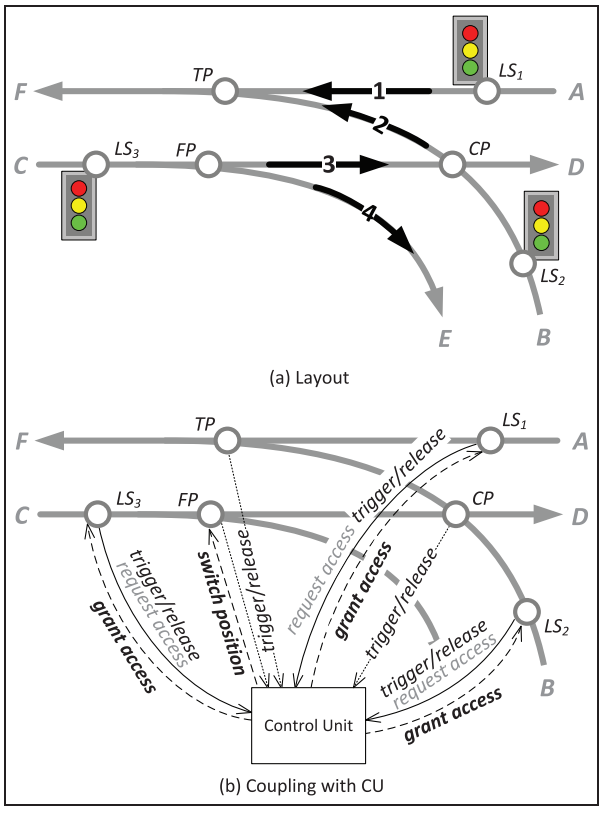

Figure 15. A Y-crossing.

granted. In addition, a CU can ask a FP to change position (viz. left or right) when necessary.

Example 6. *Suppose that a vehicle V is approaching the Y-crossing at some distance and wants to drive from C to D, that is, route 3; Figure 16.*

At some instant, *V* sends a request message (to look for its NCRI) which is received by the signal LS_3. Knowing *V* is approaching, LS_3 sends a message on behalf of *V* to request access to route 3. Upon receiving the request, the CU checks if the route is accessible. The CU does so by consulting a ''check table''.

Check table in a CU. A *check table* in a CU maintains the notes or records about the situation in an area the CU is supervising. The table has the information about the routes in the area, the points (and positions if applicable) required by the routes, the states (whether reserved and/or occupied) of the points, whether the routes are active or have queuing requests, and so on. An example of the table is shown in Table 3. A route in a crossing is *active* when a request for the route is granted to a vehicle and when the vehicle has entered or is about to enter the route but has not yet left the route (i.e. the requesting vehicle is at the crossing). If so, when another vehicle requests the same route, the request will be granted by the CU (by the default setting).

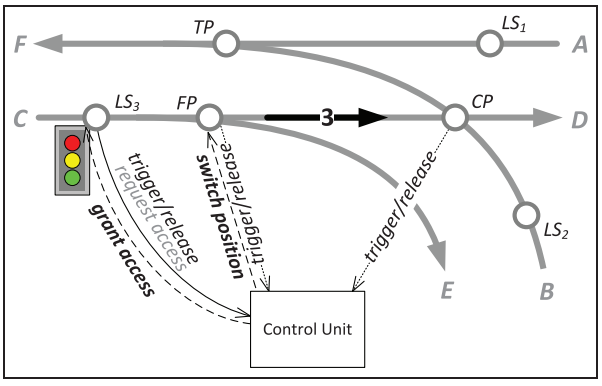

Figure 16. Route access in a Y- crossing.

In Example 6, when the CU receives a request to access route 3, it processes the request by first checking whether the route is active or has queuing requests. If both are negative, the CU then checks whether the points required by the route (i.e. FP and CP) are reserved or occupied. If they are, then the request will not be granted but is appended to the queuing requests of route 3. When a route has queuing request(s), a new request for the route will also be queued. If the points are not reserved or occupied, the request will be granted. In the latter case, the CU marks route 3 as active and the required points as reserved in its check table, and sends out a message to announce this decision. Note that the CU has one output port `O/LS/FP` that is coupled to LS_1–LS_3 and the FP. Two recipients are addressed in this message: (1) the FP whose position-to-be is noticed, and (2) the LS_3 (i.e. the entrance signal of route 3) which will give the requesting vehicle a green signal (i.e. a message).

A vehicle permitted to enter a requested route will trigger and release the detectors along the route. The CU then marks the detectors as being occupied and unoccupied respectively in its check table. Releasing a signal turns the signal back to red (which is its default state). When all detectors along the route are released, the CU marks the route as *inactive*. Once a point is released, the CU will also process the queuing requests (if any) that require this point. They are processed with vehicle priorities (if any) and on a first-come first-served basis.

3.5. Coupled infrastructure models

The RIE (`InfraElement`) models in Section 3.4 are atomic. They can be used to incrementally compose *coupled infrastructure models* (CIMs), that is, `CoupledInfraModel`. The CIMs can be used for composition as well. They are defined to represent domain meta-models that allow for a set of model compositions.

3.5.1. Node couplings and operations of infrastructure models. A CIM can be seen as a placeholder or a container

Table 3. A check table in a CU: Examples 5 and 6.

Route	Entrance Signal	Points Required	Active	Queuing requests
1	LS_1	*TP*		
2	LS_2	*CP, TP*		
3	LS_3	*FP*/L, *CP*		
4	LS_3	*FP*/R		

for a set of infrastructure models. It holds a list of coupling relations of its infrastructure sub-components. The coupling relations are specified by nodes. A node coupling is comprised of two port couplings. Figure 17 shows an example of a CIM M with two (infrastructure) sub-components M_1 and M_2. The rules for the external input coupling (EIC), external output coupling (EOC) and internal coupling (IC) in DEVS models also apply to node couplings.

A node-to-node coupling relation is strictly one-to-one. For example, the outflow node of an infrastructure model can only connect to one inflow node of another infrastructure model (which is not the former's parent model). An infrastructure layout that has only one inflow and one outflow of traffic is hence represented by a chain of successively connected infrastructure models, each of which has only one inflow node and one outflow node. We call this type of (atomic or coupled) infrastructure model `SimpleInfra-` (Section 3.2.1).

Note that the layout of ''a sensor or a signal on or along a track'' has to be represented by a chain of successive connections, similar to Figure 18(a) and (b). Connecting, for example, M_1 through one outflow node with Sensor (or `LinesideSignal`) and with M_2 at the same time will cause a non-deterministic execution sequence of the external transition functions δ_{ext} in the two latter models when there is an output event at M_1. Both δ_{ext} shall be and will be executed at the same simulation time, but which one is executed first depends on the implementation of the simulator. Uncontrolled non-determinism is in general undesirable in M&S. A non-deterministic execution sequence of different external transition functions (which are supposed to be executed at the same simulation time) may cause non-deterministic simulation results. Because non-determinism would have a negative effect on our simulation, a non-one-to-one node-to-node coupling is defined as impermissible in Libros. A sequential connection is a measure that forces a sequential activation of the aforementioned external transition functions.

An infrastructure layout with non-one-to-one traffic is represented through the use of Points, which may have one-to-two, two-to-one, or two-to-two inflow and outflow of traffic (Figure 18(c) to (e)). Combinations of them can represent an *n*-to-*n* infrastructure layout.

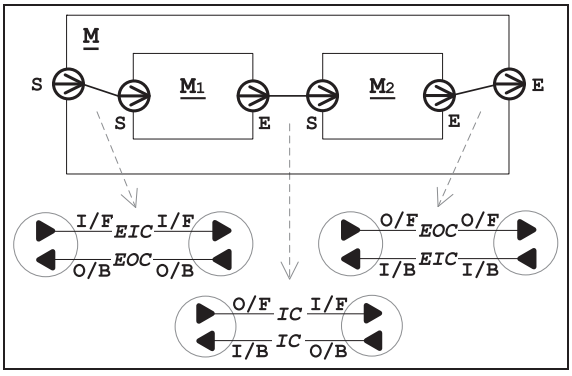

Figure 17. Couplings of nodes of infrastructure models.

A `CoupledInfraModel` needs a number of functions that can operate on model nodes, on couplings and on its sub-components. For the convenience of model construction, we defined operations to allow for, for example, adding and removing sub-components, and adding, removing and managing nodes and the couplings of nodes dynamically. Since node operations are also needed by the RIE models (`InfraElement`), both classes implement `InfraInterface`, in order to reuse the definition of these operations among the infrastructure models. These operations are indispensable for the automated model generation discussed by Huang.[52]

3.5.2. Infrastructure components. An `InfraComponent` model defines a specific layout or setting which is meant to be reusable as a whole. It can be seen as a component unit, whose sub-components are strongly related in the sense that they have common operations and/or need common control logic for the component unit to be well-functioning. The Y-crossing discussed in Example 5 is defined as an `InfraComponent` because it is a common infrastructure layout (pattern), and the 3S models in the crossing need to be supervised by a CU. Each `InfraComponent` defines a meta-model of a certain infrastructure composition whose compositional feature is described by a corresponding graph pattern.

3.5.3. Vehicle coupling. A vehicle model is generated in a source model (Section 3.5.4) during simulation, and then dynamically connected to a sequence of NCRIs one at a time representing where the vehicle is located. When it reaches its destination, it is removed from the simulation by a sink model. Dynamic structure DEVS (DSDEVS) implemented in ESDEVS allows for the change of model structure at simulation run time. Figure 19 gives an example of vehicle couplings. A vehicle model *V* is successively coupled with the RIE models at time intervals corresponding to the time that is needed by *V* moving from one NCRI to the next. It always has three couplings with a RIE:

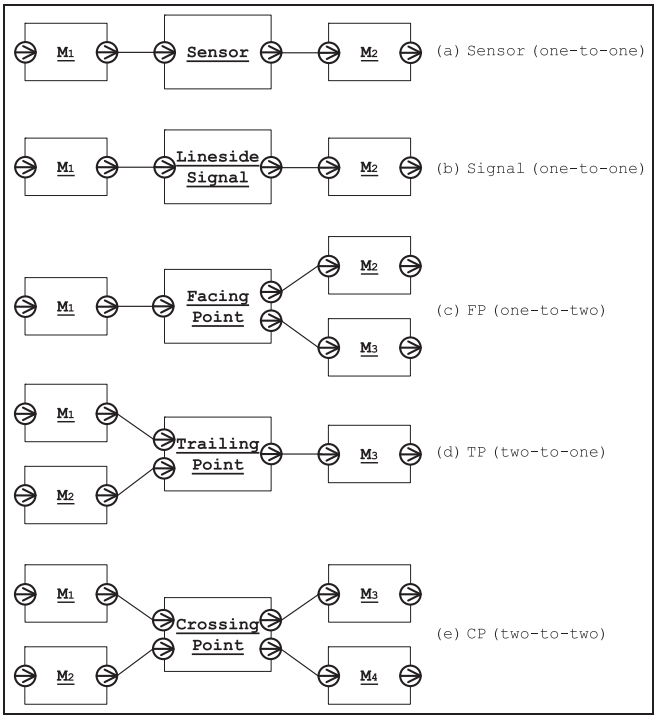

Figure 18. 3S model couplings.

(1) one coupling for input messages: from RIE output port `O/V` to *V* input port `I/RIE`;
(2) one coupling for output messages forward: from *V* output port `O/F/RIE` to RIE start node `S` input port `I/F`; and
(3) one coupling for output messages backward: from *V* output port `O/B/RIE` to RIE end node `E` input port I/B.

Since all *V*–RIE are coupled in this manner, the configuration function for dynamic *V*–RIE coupling has cohesive operations. (When a RIE model has more than one start or end node, a coupling with either will give correct results. The default setting is to connect with the first start or end node.) When a vehicle model reaches its NCRI, it invokes the configuration function `CoupleToNCRI` where the couplings with the previous NCRI are removed and new couplings with the current NCRI are established. As discussed in Section 3.3.1, a vehicle model always reaches its NCRI at an internal event. The `CoupleToNCRI` function is then invoked after which a new round of MP is initiated.

3.5.4. Source and sink. A LIBROS model contains at least one source and one sink model (Figure 20). They are of a special type of infrastructure such that no traffic flows through: only one of the two nodes of a source or sink is connected.

A `Source` is a coupled model that contains one virtual track and at least one (atomic) vehicle generator. A `VehicleGenerator` generates vehicles according to one timetable or one time interval distribution. Once generated, a vehicle is coupled to the `TrackSegment` in the source and assigned to a position as close as possible to the end of the track. The vehicle model then sends a request message at initialization (initialization refers to the very first internal event of the DEVS atomic model) after which it decides on its movement. We use a track in a source model for two reasons: (1) when there is more than one vehicle in the source, the track is a means to queue the vehicles; (2) the track is also a means that allows a vehicle to send out a request message and to drive out from inside the source, which is cohesive with the other infrastructure models. To couple a newly generated vehicle directly with a track outside a source would break the modularity of the source model. A track inside a source model is virtual as it does not model any part of the real infrastructure.

A `Sink` responds to a request message in the same manner as RIE models. But once a vehicle reaches a `Sink`, it is removed from the simulation: the vehicle's couplings are removed and the vehicle model is removed from the component list of the parent model. Because the vehicle's succeeding vehicle (if any) still holds the information about its PV, which would cause false movement computation of the

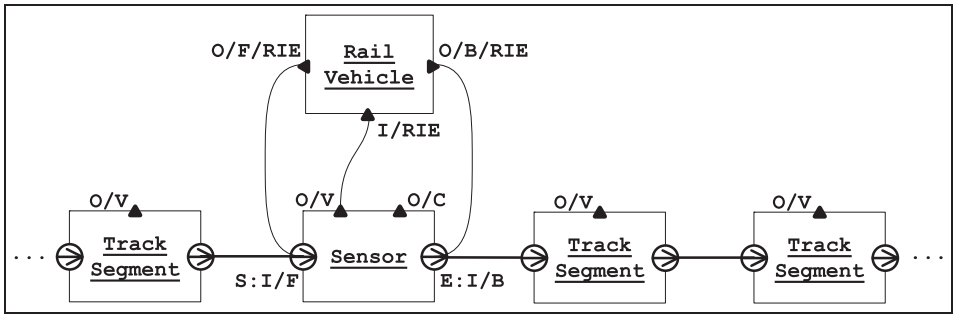

Figure 19. Rail vehicle model coupling with RIE models.

latter, the `Sink` sends a backward message to inform the latter which then clears its PV information.

4. A study on LIBROS performance

This section presents a study that compares a LIBROS model with a model that uses continuous abstractions.[61] The work reported by Kanacilo and Verbraeck[62–64] developed a rail simulation library where vehicle movement is represented by differential equations. We hereinafter call this library LIBODE. We deem LIBODE and LIBROS comparable because they were developed with the same modeling assumptions and purposes, and both present light-rail systems at the same abstraction level. LIBROS and LIBODE can both model light-rail geolocation and shape based on light-rail computer-aided design (CAD) data. Model components such as vehicles, track segments, switches and signals in LIBODE can also be found in LIBROS. Both libraries model vehicle movement with segmented constant acceleration. The real-world events modeled in one library are also modeled in the other. In addition, both libraries have the same underlying (event-scheduling-based) simulator, which is called Distributed Simulation Object Library (DSOL).[65] As depicted in Figure 21, LIBODE uses DSOL directly while LIBROS uses DSOL indirectly through ESDEVS, an event-scheduling-based DEVS simulator.[66]

4.1. Experimental setup

Using the LIBODE and LIBROS libraries respectively, we created two models representing the light-rail operation in The Hague city center tunnel in the Netherlands. This location is chosen for its simplicity but suffices to show examples of LIBODE and LIBROS models, and to compare their performance.

4.1.1. LIBROS model. The LIBROS model of the light-rail operation in the tunnel is shown in Figure 22. Only one direction (from left to right) of traffic is illustrated; the other direction is modeled similarly. In the tunnel, there is a stop with two halting places (T_3 and T_4). The stop is guarded

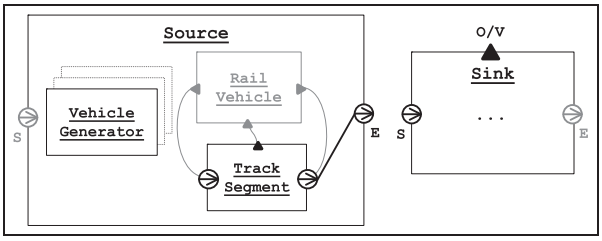

Figure 20. Source and sink models

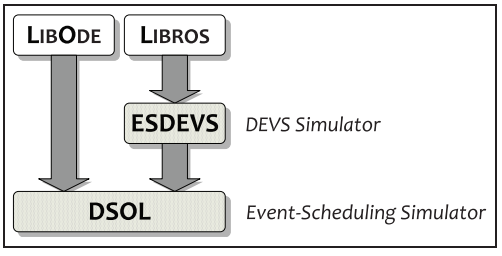

Figure 21. LIBODE and LIBROS.

by a signal (*LS*) which signals vehicles to wait (i.e. red) when either T_2 or the second halting place (T_3) is occupied by a vehicle. When the first halting place (T_4) is occupied and both T_2 and T_3 are not, the signal shows amber. The signal turns green when the block section is completely cleared. The speed limit in this tunnel is 45 kmph and it is reduced to 35 kmph when the signal is amber.

The MCT of the LIBROS tunnel model is illustrated in Figure 23. The four nodes in the MCT are `CoupledInfraModels`, among which the root is a `TopLevelModel`. The tunnel model is composed of a *block section*, two track segments T_1 and T_5, a *source* and a *sink*. The block section is composed of a lineside signal *LS*, a CU, and other components as shown. Besides vehicle detection, the sensor S_1 in *Stop* is also used by a vehicle to identify the position of the stop. Another sensor S_2 is placed at the end of the block section to detect clearance.

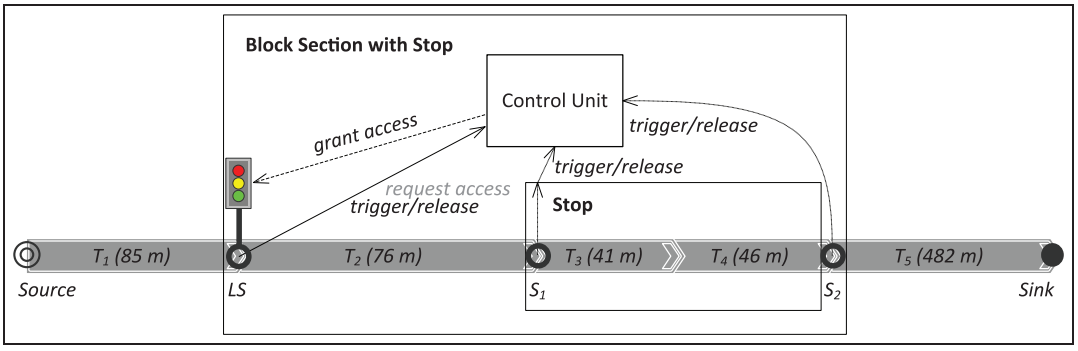

Figure 22. Light-rail operation in The Hague city center tunnel modeled with LIBROS.

4.1.2. LIBODE model. The same infrastructure and control logic are modeled with LIBODE. The LIBODE tunnel model has one track T that comprises the total length of T_1 to T_5 in the LIBROS model. The positions of the block section and the stop are defined by positions associated with the track; for example, the block section is on T, offset 85 m, length 163 m. The positions of the signal and sensors are defined in the same manner.

In LIBODE, communications are mainly accomplished through *publish–subscribe* which is implemented by DSOL. The publish–subscribe (or event notification) interaction mechanism defines an asynchronous non-static one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated.[57] More specifically, subscribers register their interests in an event (or an event pattern) with publishers, and are subsequently notified each time such an event is generated by the publishers.[67] For example, when a vehicle is approaching a signal, the state changes of the signal are of the interest to the vehicle. The vehicle model therefore registers itself to the signal's *subscribers list* and becomes a listener of its state changes. Once the vehicle passes the signal, it deregisters itself from the list.

The vehicle movement is represented by differential equations solved by the Runge–Kutta integrator contained in DSOL. In principle, at each integration step a vehicle model decides if it will accelerate, cruise or brake based on the states of the objects (i.e. vehicles, signals, sensors, etc.) associated with the track(s) ahead of it. If a new object is in sight, the vehicle subscribes to the model so that it can be notified when the model changes states.

4.1.3. Simulation experiments. For each model, we ran simulation experiments with different vehicle generation frequencies (VGFs), namely 15, 22, 30, 40, 50 and 60 vehicles per hour (vph). Two measures are taken to ensure that some vehicles do interact with another. First, the vehicle generation interval is not uniformly distributed but has time variations. When, for example, the VGF is 15 vph, generating one vehicle every four minutes would likely yield no interactions as the total driving time through the tunnel is short (~1 min in the best case without halting) and only one stop exists which can delay this time. Second, the halting time at the stop is configured such that it is long enough (i.e. more than the average vehicle generation interval) to yield vehicle interactions. When, for example, the VGF is 30 vph, the halting time would be more than two minutes.

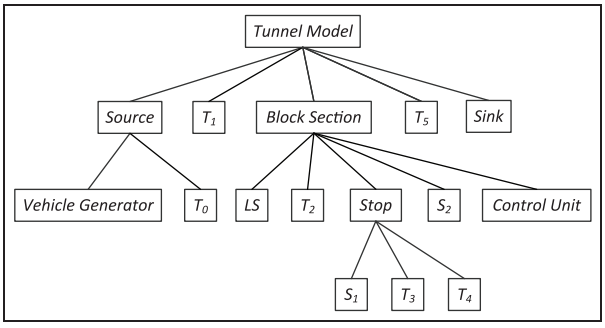

Figure 23. Composite tree of the LIBROS tunnel model.

Each pair of experiments is set with the same configurations for both models so that the results are comparable. The LIBODE model is simulated with an integration step of 1/20 s which in the worst case has a vehicle position accuracy of 0.625 m, taking into account the speed limit of 45 kmph. The vehicle position in the LIBROS mode is computed as described in Section 3.3.1 (with floating point precision). The simulation run length of each experiment is two hours. Each simulation run is profiled with the TPTP tool (Eclipse Test & Performance Tools Platform, www.eclipse.org/tptp). It is expected that the VGF and the computational cost of the simulation have a positive correlation, since more vehicles create more communications in both models: more need for integration in the case of the LIBODE model, and more state transitions in the case of the LIBROS model. The degree of difference is what we want to find out through the experiments.

VGF (vph)	15	22	30	40	50	60
LibOde model	1094	1563	1953	2797	3782	4266
Libros model	344	469	563	688	828	953

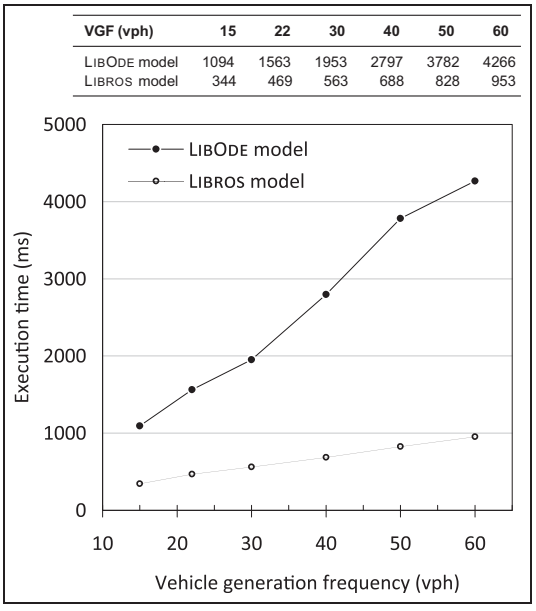

Figure 24. Execution time of the LibOde and Libros tunnel models in milliseconds.

4.2. Experiment results and discussion

The execution time measurement of the LibOde and Libros tunnel models is presented in Figure 24. In both cases, the results show that the model execution time increases nearly linearly with the increase of VGF. However, the execution time of the latter is consistently lower and increases more slowly compared to that of the former. Although the model execution time in the experiments is not large, performance becomes important for large-scale microscopic M&S. We profiled the experiments with the VGFs of 15, 30 and 60 vph, and then categorized the *execution base time* using the major functionality (with the highest base time) used by the two models. (According to the Eclipse TPTP glossary, base time is the time spent executing a particular method, not including the time spent on other methods that this method calls.) The results are shown in Figure 25.

As mentioned earlier, both LibOde and Libros libraries are based on DSOL. LibOde uses DSOL for integration and model communications (i.e. publish–subscribe) which in turn use the event scheduler in DSOL. Libros uses ESDEVS for model state transitions and communications (i.e. port-based message passing). The events are scheduled by ESDEVS using the event scheduler in DSOL as well. Because the base time is taken as the time measure for the experiments, the time spent on integration by the LibOde model is ''pushed'' from the *DSOL integrator* category to the *LibOde vehicle* where the demand for integration is located. The data shows the high computational demand of the integration as expected. The integration (or

LibOde model

VGF (vph)	15	30	60
DSOL event-scheduling	140	237	505
DSOL integrator	175	256	444
LibOde vehicle	539	1048	2383
Others	188	285	544
LibOde track	53	126	389
Others	188	285	544

Libros model

VGF (vph)	15	30	60
DSOL event-scheduling	172	275	442
ESDEVS	16	43	88
Libros vehicle/ track	6	15	31
Others	149	230	392

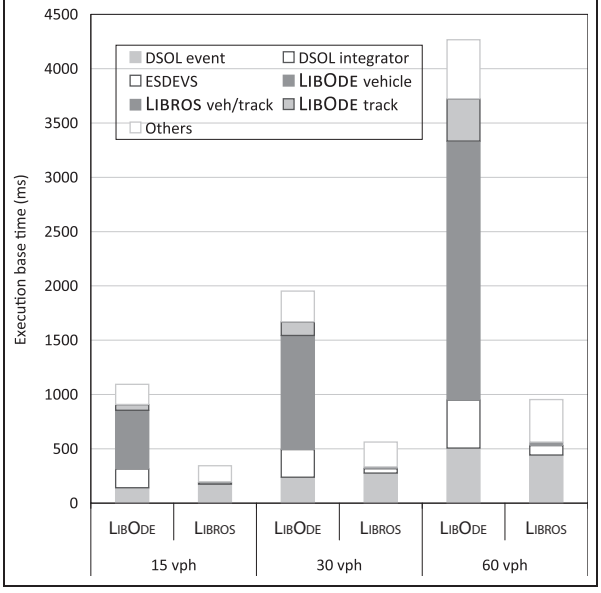

Figure 25. Execution base time per category of the LibOde and Libros tunnel models.

the vehicle model as the primary component in the LibOde library) needs a large portion of the total execution time.

Compared with the *LibOde vehicle* and *LibOde track* categories, the *Libros vehicle/track* category is lightweight. But the *DSOL event-scheduling* in the Libros model is relatively heavy and the most demanding because all DEVS model transitions and communications managed by the ESDEVS library eventually use the event scheduler in DSOL. Nevertheless, the data shows that the computational cost of this part is comparable to the cost of DSOL event-scheduling in LibOde. Note that model communications in both libraries are handled at the cost of DSOL event-scheduling. The communication mechanism in LibOde is based on publish–subscribe. In Libros it is MP. This means that, for the simulation experiments, despite extra message sending and state transitions, the design of MP in Libros does not yield significantly more computational cost than the publish–subscribe communications in LibOde. (In the case of 60 vph in the DSOL event-scheduling category, the total cost of transitions and

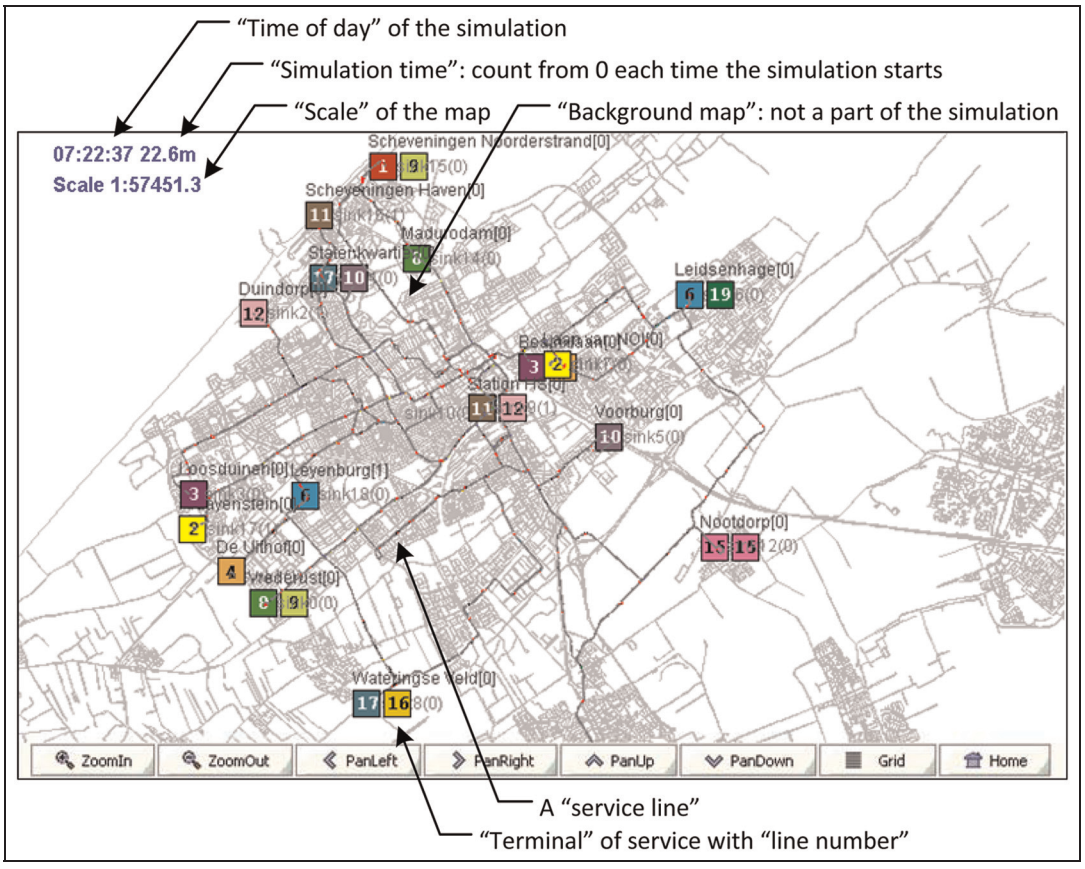

Figure 26. Animation window of The Hague light-rail network simulation model with legend.

communications in LIBROS is even lower than the cost of communications alone in LIBODE.) This result is consistent with the cost estimation of MP presented in Section 2.3.

In general, the integrator and the use of it comprise the primary component of the continuous model. The event handler (e.g. ESDEVS in combination with the DSOL event scheduler in the case of LIBROS) and its use have comparable roles for a discrete-event model. Given the same integrator, the design choices of a continuous model, however, often have little influence on the efficiency of the numerical solution, since the integrator solves the differential equations with a defined time step. In contrast, with a discrete-event model, the design choices (i.e. the choice of events and the state transitions on the events) often have a strong influence on the time advances and hence affect model performance. This opens up possibilities for, but does not guarantee, improvement of model performance, and requires modelers to make careful choices regarding model design. The results of the experiments show that, with comparable modeling details and accuracy, the LIBROS model yields higher performance than the LIBODE model. This is an example that careful design of discrete-event models can yield high performance without losing model accuracy.

5. Applications

There are a number practical uses for the LIBROS simulation models. For example, key performance indicators such as deadhead kilometers and average delays were evaluated using LIBROS models to inform the choice of depot capacity and vehicle schedules.[68] To reduce waiting times at the entrance of the tram tunnel *Grote Markt* in The Hague (where the setup is similar to that discussed in Section 4), simulation experiments were run with the models using different timetables and safety control measures. The models were also used to study the impact of, for example, merging two stations and the new infrastructure construction at *Station HS* in the light-rail network of The Hague. An automated model generation method is developed using different light-rail infrastructure (CAD) data as input and the LIBROS model components as building blocks.[52] Figure 26 shows the generated light-rail network simulation model of The Hague. For a whole-day operation (from around 6 a.m. to 1 a.m.) of the network, the model's simulation run time is about 8 min on an ordinary laptop. HTM is interested in further uses of the model generator and the model component library developed. A preliminary version of a graphical user interface was developed[69] to help

end users changing model parameter settings. The organization is also in the process of developing a web application to access the simulation models. The use of the model components in these cases showed their composability and reusability.

6. Conclusions

We have presented a scalable component-based light-rail simulation model library developed in the DEVS formalism. The proposed library enables a detailed microscopic modeling of the vehicle and infrastructure dynamics and offers an innovative scheme for MP and vehicle-to-vehicle/infrastructure communication. This paper has given a detailed account of the design choices adopted in developing the library. The experimental study has demonstrated that with a similar level of model detail and accuracy, the modeling approach that we propose results in a significantly lower computational cost than a differential equation model does. Furthermore, the library has been applied to different cases, thereby giving an indication of its composability and reusability.

From this experience, a number of good design practices were derived, which may be applicable to the development of large-scale transport infrastructure models beyond the particular case of light-rail systems. First, one of the main design guidelines is to sparsely use model artifacts which are constructed purely for the purpose of modeling. By this we mean that artifacts that do not represent an aspect of the real system should be used with parsimony. Using a formal model specification that separates abstraction from simulation execution is helpful in this respect. Second, when presented with a choice among competing modeling alternatives, the alternative that offers most modularity and simplicity should be favored. Third, when modeling communications, if a decentralized design leads to strongly connected model components and a broadcast-like communication, centralization can be a beneficial alternative in terms of communication costs. The latter, however, can be less convenient in modeling the autonomy of the aforementioned components. Fourth, the configuration (and reconfiguration) cost is an important criterion in choosing a communication mechanism between model components that have dynamic structures. Finally, vehicle movement can be represented without loss in accuracy with piecewise polynomial segments, each of which has a constant acceleration rate as a coefficient.

Future directions of this research include expanding the library to enable modeling disturbances, rerouting, depots, and passenger origin and destination. Some of these components can be linked with real data such as that from electronic ticketing systems and automatic passenger counting systems.

Acknowledgements

The authors thank Martijn Warnier from TU Delft for his feedback. This article is based on the PhD thesis *Automated simulation model generation*, Delft University of Technology, 2013, by Yilin Huang.

Funding

This work was partly supported by HTM, a public transport company in The Hague, the Netherlands.

References

1. Hansen IA and Pachl J. *Railway timetable & traffic: Analysis – modelling – simulation*. Hamburg, Germany: Eurailpress, 2008.
2. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*. 2nd ed. New York, NY: Elsevier/Academic Press, 2000.
3. Ho YC. *Discrete event dynamic systems: Analyzing complexity and performance in the modern world*. New York, NY: IEEE, 1994.
4. Wainer GA. *Discrete-event modeling and simulation: A practitioner's approach* (*Computational Analysis, Synthesis, and Design of Dynamic Systems*). Boca Raton, FL: CRC Press, 2009.
5. Paredis CJJ, Diaz-Calderon A, Sinha R, et al. Composable models for simulation-based design. *Eng Comput* 2001; 17(2): 112–128.
6. Liang VC and Paredis CJJ. A port ontology for conceptual design of systems. *J Comput Inf Sci Eng* 2004; 4(3): 206–217.
7. Breedveld PC. Port-based modeling of dynamic systems. In: Duindam V, Macchelli A, Stramigioli S, et al. (eds) *Modeling and control of complex physical systems: The Port-Hamiltonian approach*. Berlin: Springer-Verlag, 2009, pp. 1–52.
8. De Lara J, Vangheluwe H and Alfonseca M. Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3. *Softw Syst Model* 2004; 3(3): 194–209.
9. Vangheluwe H. DEVS as a common denominator for multi-formalism hybrid systems modelling. In: *IEEE international symposium on computer-aided control system design*, 2000, pp. 129–134.
10. Vangheluwe H. Foundations of modelling and simulation of complex systems. In: *Electronic communications of the EASST – proceedings of the 7th international workshop on graph transformation and visual modeling techniques*, 2008.
11. Li KP and Gao ZY. An improved equation model for the train movement. *Simul Modell Pract Theor* 2007; 15(9): 1156–1162.
12. Li KP, Gao ZY and Ning B. Cellular automaton model for railway traffic. *J Comput Phys* 2005; 209(1): 179–192.
13. Middelkoop D and Bouwman M. SIMONE: Large scale train network simulations. In: *Proceedings of the 2001 winter simulation conference*, 2001, pp. 1042–1047.
14. Lu Q, Dessouky M and Leachman RC. Modeling train movements through complex rail networks. *ACM Trans Model Comput Simul* 2004; 14(1): 48–75.

15. Li KP, Mao BH and Gao ZY. An improved walk model for train movement on railway network. *Commun Theor Phys* 2009; 51(6): 979–984.
16. Zeigler BP and Lee JS. Theory of quantized systems: Formal basis for DEVS/HLA distributed simulation environment. In: *Proceedings of SPIE – The International Society for Optical Engineering*, 1998, pp. 49–58.
17. Giambiasi N, Escude B and Ghosh S. GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. *Trans SCS* 2000; 17(3): 120–134.
18. Kofman E. Quantization-based simulation of differential algebraic equation systems. *Simul* 2003; 79(7): 363–376.
19. Pachl J. *Railway operation and control*. Mountlake Terrace, WA, USA: VTD Rail Publishing, 2002.
20. Carey M and Lockwood D. A model, algorithms and strategy for train pathing. *J Oper Res Soc* 1995; 46(8): 988–1005.
21. Rizzoli AE, Fornara N and Gambardella LM. A simulation tool for combined rail/road transport in intermodal terminals. *J Math Comput Simul* 2002; 59(1–3): 57–71.
22. Carey M and Carville S. Testing schedule performance and reliability for train stations. *J Oper Res Soc* 2002; 51(6): 666–682.
23. Carey M and Carville S. Scheduling and platforming trains at busy complex stations. *Transp Res A Pol Prac* 2003; 37(3): 195–224.
24. Wahlborg M. Simulation models: Important aids for Banverket's planning process. In: Brebbia CA, Allan J, Hill J, et al. (eds) *Computers in railways*, vol. 5. Southampton, UK: WIT Press, 1996, pp. 175–181.
25. Sandblad B, Andersson A, Jonsson KE, et al. A train traffic operation and planning simulator. *Adv Transp* 2000; 7: 241–248.
26. Bendfeldt JP, Mohr U and Muller L. RailSys, a system to plan future railway needs. *Adv Transp* 2000; 7: 249–255.
27. Kaas AH. Punctuality model for railways. *Adv Transp* 2000; 7: 853–860.
28. Kavicka A and Klima V. Simulation support for railway infrastructure design and planning processes. *Adv Transp* 2000; 7: 447–456.
29. Ho TK, Mao BH, Yuan ZZ, et al. Computer simulation and modeling in railway applications. *Comput Phys Commun* 2002; 143(1): 1–10.
30. Nash A and Huerlimann D. Railroad simulation using OpenTrack. *Adv Transp* 2004; 15: 45–54.
31. Koutsopoulos HN and Wang Z. Simulation of urban rail operations: Application framework. *Transp Res Rec J Transp Res Board* 2007; 2006: 84–91.
32. Rudolph R. Operational simulation of light rail systems. In: *Proceedings of the European transport conference*, 2000, pp. 167–178.
33. Overton DT. Traffic signal control of LRVs. In: *IEE colloquium on light rapid transit on-street*, 1989, pp. 9/1–9/3.
34. Sol HG. *Simulation in information systems development*. PhD Thesis, University of Groningen, the Netherlands, 1982.
35. Keen PG and Sol HG. *Decision enhancement services: Rehearsing the future for decisions that matter*. Amsterdam, the Netherlands: IOS Press, 2008.
36. Hofmann M. Criteria for decomposing systems into components in modeling and simulation: Lessons learned with military simulations. *Simul* 2004; 80(7–8): 357–365.
37. Petty MD and Weisel EW. A formal basis for a theory of semantic composability. In: *Proceedings of the spring 2003 simulation interoperability workshop*, 2003.
38. Verbraeck A, Saanen Y, Stojanovic Z, et al. What are building blocks? In: Verbraeck A and Dahanayak ANW (eds) *Building blocks for effective telematics application development and evaluation*. Delft, the Netherlands: TU Delft, 2002, pp. 8–21.
39. Sommerville I. *Software engineering*. 5th ed. Reading, MA: Addison-Wesley, 1996.
40. Braude EJ and Bernstein ME. *Software engineering: Modern approaches*. 2nd ed. New York, NY: John Wiley & Sons, 2010.
41. Fowler M. *Refactoring: Improving the design of existing code*. Reading, MA: Addison-Wesley, 1999.
42. Banks J. *Handbook of simulation: Principles, methodology, advances, applications, and practice*. New York, NY: Wiley Interscience, 1998.
43. Fujimoto RM. *Parallel and distributed simulation systems* (*Wiley Series on Parallel and Distributed Computing*). New York, NY: John Wiley & Sons, 2000.
44. Crnkovic I, Stafford J and Szyperski C. Software components beyond programming: From routines to services. *IEEE Softw* 2011; 28(3): 22–26.
45. Yilmaz L. On the need for contextualized introspective models to improve reuse and composability of defense simulations. *J Defen Model Simul Appl Method Tech* 2004; 1(3): 141–151.
46. Szabo C and Teo YM. On syntactic composability and model reuse. In: *Proceedings of the international conference on modeling and simulation*, 2007, pp. 230–237.
47. Tolk A, Diallo SY, King RD, et al. Conceptual modeling for composition of model-based complex systems. In: Robinson S, Brooks R, Kotiadis K, et al. (eds) *Conceptual modeling for discrete-event simulation*. Boca Raton, FL: CRC Press, 2010, pp. 355–381.
48. Szyperski C. *Component software: Beyond object-oriented programming* (*Addison-Wesley Component Software Series*). 2nd ed. Reading, MA: Addison-Wesley, 2011.
49. Lee JK, Lim YH and Chi SD. Hierarchical modeling and simulation environment for intelligent transportation systems. *Simul* 2004; 80(2): 61–76.
50. Wainer G. ATLAS: A language to specify traffic models using Cell-DEVS. *Simul Modell Prac Theor* 2006; 14(3): 313–337.
51. Theeg G and Vlasenko S. *Railway signalling & interlocking: International compendium*. Hamburg, Germany: Eurailpress, 2009.
52. Huang Y. *Automated simulation model generation*. PhD Thesis, Delft University of Technology, Delft, the Netherlands, 2013.
53. Vuchic VR. *Urban transit: Operations, planning, and economics*. New York, NY: John Wiley & Sons, 2005.
54. Knuth DE. *The art of computer programming, volume 1: Fundamental algorithms*. Reading, MA: Addison-Wesley Professional, 1997.
55. Cormen TH, Leiserson CE, Rivest RL, et al. *Introduction to algorithms*. 3rd ed. New York, NY: MIT Press and McGraw-Hill, 2001.

56. Alstrup S, Gavoille C, Kaplan H, et al. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theor Comput Syst* 2004; 37: 441–456.
57. Gamma E, Helm R, Johnson R, et al. *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley, 1994.
58. Viega J, Tutt B and Behrends R. Automated delegation is a viable alternative to multiple inheritance in class based languages. Charlottesville, VA: Microsoft Corporation, 1998.
59. Kofman E, Lee JS and Zeigler BP. DEVS representation of differential equation systems: Review of recent advances. In: *Proceedings of the 2001 European simulation symposium*, 2001.
60. Giambiasi N and Carmona JC. Generalized discrete event abstraction of continuous systems: GDEVS formalism. *Simul Modell Prac Theor* 2006; 14(1): 47–70.
61. Huang Y, Seck MD and Verbraeck A. LIBROS-II: Railway modelling with DEVS. In: *Proceedings of the 2010 winter simulation conference*, Baltimore, MD, 2010.
62. Kanacilo EM and Verbraeck A. A distributed multi-formalism simulation to support rail infrastructure control design. In: *Proceedings of the 2005 winter simulation conference*, 2005, pp. 2546–2553.
63. Kanacilo EM and Verbraeck A. Simulation services to support the control design of rail infrastructures. In: *Proceedings of the 2006 winter simulation conference*, 2006, pp. 1372–1379.
64. Kanacilo EM and Verbraeck A. Assessing tram schedules using a library of simulation components. In: *Proceedings of the 2007 winter simulation conference*, 2007, pp. 1878–1886.
65. Jacobs PHM. *The DSOL simulation suite – Enabling multi-formalism simulation in a distributed context*. PhD Thesis, Delft University of Technology, Delft, the Netherlands, 2005.
66. Seck MD and Verbraeck A. DEVS in DSOL: Adding DEVS operational semantics to a generic event-scheduling simulation environment. In: *Proceedings of the 2009 summer computer simulation conference*, 2009.
67. Eugster PT, Felber PA, Guerraoui R, et al. The many faces of publish/subscribe. *ACM Comput Surv* 2003; 35(2): 114–131.
68. Cai J. *Assessing the impact of capacity of depots and vehicle schedule in transportation systems*. Master's Thesis, Delft University of Technology, Delft, the Netherlands, 2011.
69. Van Antwerpen H. *Rail simulation suite development*. BA Thesis, De Haagse Hogeschool, The Hague, the Netherlands, 2011.

Author biographies

Yilin Huang is a researcher at the Department of Multi-Actor Systems, Faculty of Technology, Policy and Management, Delft University of Technology, the Netherlands.

Mamadou D Seck is an assistant professor of engineering management and systems engineering at Old Dominion University in Norfolk, VA.

Alexander Verbraeck is a professor at the Department of Multi-Actor Systems, Faculty of Technology, Policy and Management, Delft University of Technology, the Netherlands.