Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

LIBROS-II: RAILWAY MODELING WITH DEVS

Yilin Huang
Mamadou D. Seck
Alexander Verbraeck

Systems Engineering Group
Delft University of Technology
PO Box 5015, NL-2600GA Delft, The Netherlands

ABSTRACT

The increasing complexity of railway systems and the high costs incurred by design and operational errors make modeling and simulation a popular methodology in the domain of railway transportation. To successfully support detailed design and operation, a microscopic rail network model is often deemed not only suitable but also mandatory. However, the simulation of large-scale microscopic models is computationally intensive, making it unsuitable for real-time applications. In this paper, a railway simulation library, LIBROS-II, is introduced which offers high performance rail simulation at the microscopic level. The library is specified with the DEVS formalism. Its major components and their specifications are presented. Its performance is assessed through a simple example and contrasted with a typical model using a continuous modeling abstraction of train movement. The result shows that with comparable model detail and accuracy the LIBROS-II model yields a higher performance than the model using differential equations.

1 INTRODUCTION

The complex dynamics in railway systems pose many challenges in railway modeling and simulation. The model must have enough detail and accuracy to represent the sophisticated phenomena, and still be computationally efficient. Apart from these criteria, considering the reusability of the model, modularity is a concept adopted from software engineering (Baldwin and Clark 2000, Sullivan et al. 2001) that needs to be addressed during model design. In railway simulation, modeling train movement is the base element to estimate the train running time (Hansen and Pachl 2008). The train model calculates the speed-distance profile of a train traveling from one point to another (Li and Gao 2007). Other model components formulate the boundary conditions to compute the movement. Many factors influence the movement, e.g. curvatures, control signals, and speed restrictions. In the simulation of electric city railways such as trams and light-rails, modeling the interactive movement of trains is necessary as the trains do not always operate in block systems which provide automatic safe spacing control (Rudolph 2000).

Different system specification paradigms can be applied to train movement modeling. Each of them has advantages as well as disadvantages in terms of levels of model detail, accuracy, modularity, and computational efficiency. A continuous abstraction of train behavior is obtained by assuming a continuous time base and defining the rate of change of the train's state using differential equations. A numerical integrator executes the model. Some examples are discussed in Hansen and Pachl (2008). A discrete-time abstraction of train movement is obtained through the definition of a time-invariant recurrence relation between the current state of the train (position, speed, acceleration) and its future state after a predefined time interval has elapsed. Discrete-time models of train behavior have been developed with difference equations, logistic maps, and cellular automata. Some examples are Li, Gao, and Ning (2005), Li and Gao (2007), and Wainer (2007). A discrete-event abstraction of train movement is obtained through the definition of events, which represent significant state changes of the train being modeled. The definition of events is based on the

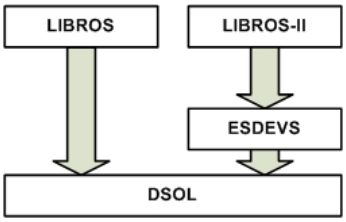


Figure 1: The library dependency.

model's purpose and/or the modelers' interest. Some examples are Middelkoop and Bouwman (2001), Lu, Dessouky, and Leachman (2004), and Li, Mao, and Gao (2009).

The choice of the time interval in both discrete-time and continuous modeling approaches is basically a trade-off between computational efficiency and simulation accuracy. The smaller the time step, the more accurate and the least computationally affordable. Owing to a longer tradition, the continuous modeling style appears to be the most intuitive approach. Well known equations, e.g. equations of motion, can be directly applied without further modeling effort. Although the discrete-event approach is often the most efficient (Zeigler and Lee 1998, Giambiasi, Escude, and Ghosh 2000, Kofman 2003) among the three, it requires a careful analysis and design of the model.

The open source java Library for Rail Operations Simulation (LIBROS) is discussed in the works of Kanacilo and Verbraeck (2005), Kanacilo and Verbraeck (2006), Kanacilo and Verbraeck (2007), and Kanacilo and Oort (2008). In LIBROS, train movement is represented by differential equations. The other components, e.g. sensors and control signals, are modeled with discrete-event abstraction. In this paper, we introduce LIBROS-II, in which the railway components are modeled using the Discrete Event System Specification (DEVS) formalism (Zeigler, Praehofer, and Kim 2000). The underlying simulator of LIBROS and LIBROS-II is DSOL, the Distributed Simulation Object Library (Jacobs, Lang, and Verbraeck 2002, Jacobs 2005). DSOL is a java suite for continuous and discrete-event simulation. It consists of components including an event-scheduler, a DEVS simulator (only the event-scheduling part), DESS and DEVDESS simulators, probability distributions, etc. As a recent development, the Event-Scheduling DEVS library (ESDEVS) (Seck and Verbraeck 2009) implements the parallel DEVS formalism on top of the DSOL library. The ESDEVS is based on the event-scheduling worldview, wherein executions of the internal transition function are scheduled according to the specified time advance function and unscheduled at the reception of external events. The confluent transition function handles the coincidence between internal and external events. Dynamic structure DEVS is also implemented in the ESDEVS library so that components and coupling relations can be added and removed dynamically during simulation runtime.

Figure 1 shows the dependency between LIBROS, LIBROS-II, ESDEVS and DSOL. Apart from statistics and animation services, LIBROS relies on DSOL's DEVDESS simulator while LIBROS-II inherits ESDEVS. In both ways, the separation of concerns between models and simulators is respected. The rest of the paper is organized as following. In the next Section, LIBROS is briefly reviewed. Section 3 presents the major components and model specifications in LIBROS-II. A simple example is given in Section 4. Test models are built with both LIBROS and LIBROS-II libraries. Their performances are compared.

2 THE MODELS IN LIBROS

In LIBROS, each track segment is an ordered pair of nodes. The position of an infrastructure element, e.g. a sensor, a control signal or a stop, is defined in association with a track; i.e. each element has a relative position to the track segment where it is located. The train movement is modeled by differential equations solved by the Runge-Kutta integrator in DSOL. In each integration time-step Δt, given the instantaneous acceleration rate of a train at time t_n, it computes the train's speed and position/distance at time $t_{n+1} := t_n + \Delta t$. The acceleration rate is determined by whether there are any objects ahead of the train (within certain distance) that would cause the train's speed change, e.g. a speed restriction, a red traffic signal, or another train. If an object changes its state, e.g. a control signal turned from green to red or a preceding train reduced its speed, the object notifies the approaching train using the publish-subscribe (also called event notification) interaction scheme (Eugster et al. 2003). The object-to-object communication in LIBROS (and DSOL) is based on this scheme, which defines a

non static one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically (Gamma et al. 1994). For details, readers may refer to Kanacilo and Verbraeck (2005), Kanacilo and Verbraeck (2006), Kanacilo and Verbraeck (2007), Kanacilo and Oort (2008).

3 THE DEVS MODELS IN LIBROS-II

LIBROS-II is a rail simulation library in which the railway components are defined using the DEVS formalism. In the library, three basic types of the DEVS atomic model are defined. These are train (`RailVehicle`), track segment (`TrackSegment`), and control unit (`ControlUnit`). By "basic type", we mean that the atomic model may have specializations as subtypes. For example, depending on the function of a control unit, it can be a block section control unit, or an intersection control unit. A sensor (`Sensor`) or a control signal (`LinesideSignal`) is a specialized form of `TrackSegment` with length zero. The infrastructure components, e.g. a station, a block section, or an intersection, are coupled models. The LIBROS-II models have the following characteristics: (1) Each track segment has one shape (straight or arc) and one speed limit. If the speed limit of a track segment is not defined (i.e. ∞), it complies with the speed limit of its previous track segment. (2) A sensor, a control signal, a control unit, or any of their specializations can only be a sub-component in a coupled infrastructure component. (3) In a coupled infrastructure component, the speed limit of each track segment complies with the speed limit (if defined) of its direct parent component. (4) At any simulation time, a train is linked directly with a track segment. The train has a relative position counting from the starting node of the track segment.

3.1 Message Propagation

The rail infrastructure model, at the lowest description level, is a directed non-planar graph of linked track segments. Each track segment is capable of *message propagation*, which can be along the traffic current or in the opposite direction. A train determines its movement based on the information about the next infrastructure and/or the preceding train. Lacking such information, the train sends a request-message forward. The track segment that gets the message propagates the message until the next infrastructure and/or a preceding train is found. The found object sends a reply-message. The message is propagated back until it reaches the original sender of the request-message.

A simplified example is illustrated in Fig. 2. It is composed of four `TrackSegments` ($TS_0 \sim TS_3$) and two `RailVehicles` (V_0, V_1). Each `TrackSegment` has a length (L) and a speed limit (SL). A `RailVehicle` has its vehicle length (VL), position (P) relative to the track segment it is linked to, and its current speed limit (CSL). (The other attributes are not illustrated.) Supposing that V_0 doesn't have information about its next infrastructure nor about the preceding vehicle, it sends a request-message. Two message sequences will be generated upon this action: (1) $M_0, M_1, M_2, M_3, M_4, M_5$, and (2) $(M_0, M_1, M_2,)M_{3'}, M_{4'}, M_{5'}, M_6, M_7, M_8, M_9$.

In sequence (1), it is assumed that $SL_1 = CSL_0 \vee SL_1 = \infty \wedge SL_2 \neq CSL_0 \wedge SL_2 \neq \infty$. A `TrackSegment` replies to a request-message when the `TrackSegment` requires a speed (limit) change of the vehicle approaching. A `TrackSegment` forwards a request-message to a vehicle closest to its start node, if there is any vehicle linked to it. A message contains information of the sender, the contemplated receiver (if necessary), and the distance between the sender and the receiver. The distance of M_9 in

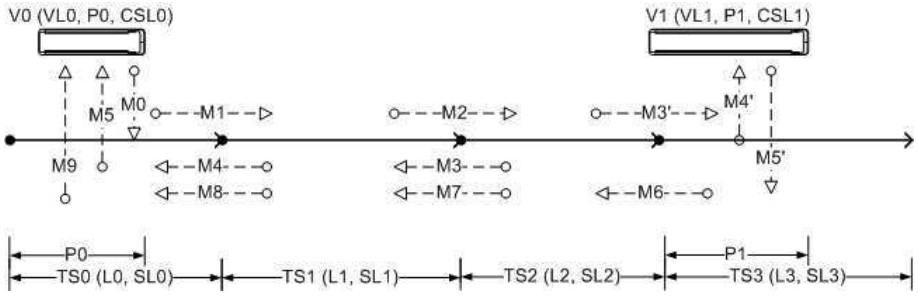


Figure 2: A message propagation example.

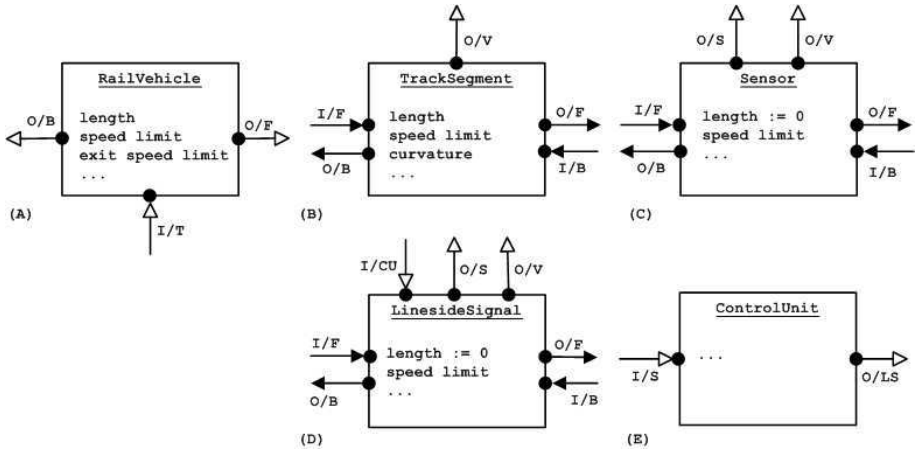


Figure 3: The atomic LIBROS-II models.

sequence (2), for example, is $M_9.D = -VL_1 + P_1 + L_2 + L_1 + L_0 - P_0$. It is accumulated (or deducted) by `TrackSegments` during the message propagation. (The behavior of a `TrackSegment` is presented in Section 3.3.) Message propagation involving `Sensors` and `LinesideSignals` functions according to the same principle. The only difference is that they always reply to request-messages.

3.2 Rail Vehicle

A `RailVehicle`'s main task is to correctly compute the vehicle's movement based on the information it has about the infrastructure and the preceding vehicle it is approaching. If either information is missing before each movement, the vehicle sends a request-message forward. After the vehicle obtains the reply-message(s), its movement is computed. If its acceleration changes as a result of the computation, the vehicle sends a message backward to inform its succeeding vehicle. (The vehicle sends a message backward every time when its acceleration changes.) A `RailVehicle` also replies to a request-message from its succeeding vehicle. The atomic `RailVehicle`'s ports are designed as shown in Fig. 3 (A). The model has one input port (I/T), and two outport ports, output forward (O/F) and output backward (O/B). All three ports are linked to the track segment the vehicle is on. The DEVS formalism with port specifications is $\texttt{RailVehicle} = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$ where

$X = \{(p,m) | p \in \{I/T\}, m \in X_p\}$ is the set of input ports and messages;

$Y = \{(p,m) | p \in \{O/F, O/B\}, m \in Y_p\}$ is the set of output ports and messages;

$S = \{\texttt{START}, \texttt{FOLLOW}, \texttt{MOVE\_TO\_NEXT\_TRACK}, \texttt{WAIT}, \texttt{DWELL}, \texttt{STOP}, \texttt{IDLE}\} \times X_p \times IS \times \mathfrak{M} \times \mathfrak{I} \times \mathfrak{V}$, $IS = \{v, a, v_{max}, p, vl, \ldots\}$ is the set of `RailVehicle`'s internal state ($v, v_{max}, p \in \mathbb{R}_0^+$, $a \in \mathbb{R}$, $vl \in \mathbb{R}^+$ are the speed, speed limit, position on the track, acceleration, and vehicle length), $\mathfrak{M} = \{M_0, M_1, \ldots, M_n\}, n \in \{\mathbb{N} \cup \varnothing\}$, is the set of future movement-trajectory with a total order ($M_n = \{t_n, v_{fn}, d_n, a_n\}$ is a movement within which the acceleration remains constant, $t_n, v_{fn}, d_n \in \mathbb{R}_0^+$, $a_n \in \mathbb{R}$ are the time (duration), final speed, total distance, and acceleration of the movement, $a_i \neq a_{i+1}$, $i < n$), $\mathfrak{I} = \{d_I, v_{ex}, \ldots\}$ is the set of information of the next infrastructure ($d_I, v_{ex} \in \mathbb{R}_0^+$ is the distance to and speed limit of the next infrastructure), $\mathfrak{V} = \{d_V, v_V, a_V\}$ is the set of information of the preceding vehicle ($d_V \in \mathbb{R}^+$, $v_V \in \mathbb{R}_0^+$, $a_V \in \mathbb{R}$ are the distance to the preceding vehicle, and its speed and acceleration);

$\delta_{ext}(phase, \sigma, IS, M', e, (p,m)) :=$

1. if $M' \neq \varnothing \wedge e \neq 0$, update current movement and internal state: $M' = \mathfrak{f}(M', e)$, $IS = \mathfrak{f}(IS, M', e)$,
2. $\sigma = \sigma - e$,
3. if $m.receiver == this$,

if $m.type == request\_message$, $send\_message\_backward = true$, $\sigma = 0$,
else,
(1) if $m.sender == vehicle$, update information of the preceding vehicle: $\mathfrak{V} = m.info$,
else, update information of the next infrastructure: $\mathfrak{I} = m.info$,
(2) if $phase \notin \{\texttt{WAIT}, \texttt{DWELL}\}$,
a. compute movement: $(\mathfrak{M}, phase) = \mathfrak{g}(\mathfrak{I}, \mathfrak{V}, IS)$,
b. if $\mathfrak{M} == \varnothing$, $M' = \varnothing$, $\sigma = \infty$,
else, $\sigma = 0$, if $a \neq a_0$, $a = a_0$, $send\_message\_backward = true$;

$\delta_{int}(\mathfrak{M}) :=$
if $\mathfrak{M} \neq \varnothing$,
1. if $a \neq a_0$ $(a_0 \in M_0 \in \mathfrak{M})$,
(1) $a = a_0$, update internal state: $IS = \mathfrak{f}(IS, M', e)$,
(2) update information of the preceding vehicle and next infrastructure:
$\mathfrak{I} = \mathfrak{f}(\mathfrak{I}, M', e)$, $\mathfrak{V} = \mathfrak{f}(\mathfrak{V}, M', e)$,
2. $send\_message\_backward = true$, $\sigma = t_0$ $(t_0 \in M_0 \in \mathfrak{M})$,
3. $M' = M_0$, remove M_0 from $\mathfrak{M}$: if $\#\mathfrak{M} == 1$, $\mathfrak{M} = \varnothing$, else, $M_{n-1} = M_n$, $M_n = \varnothing$,
else,
if $phase \in \{\texttt{START}, \texttt{MOVE\_TO\_NEXT\_TRACK}, \texttt{FOLLOW}\}$,
1. if $phase == \texttt{MOVE\_TO\_NEXT\_TRACK}$,
(1) link vehicle to next track: $\mathfrak{h}(\mathfrak{I})$, $p = 0$,
(2) if $not\ \mathfrak{I}.entered\_stopping\_place$, clear information of the next infrastructure: $\mathfrak{I} = \varnothing$,
2. if $\mathfrak{I} == \varnothing$, $send\_message\_foreward = true$, $\sigma = 0, phase = \texttt{IDLE}$,
else,
a. compute movement: $(\mathfrak{M}, phase) = \mathfrak{g}(\mathfrak{I}, \mathfrak{V}, IS)$,
b. if $\mathfrak{M} == \varnothing$, $M' = \varnothing$, $\sigma = \infty$,
else, $\sigma = 0$, if $a \neq a_0$, $a = a_0$, $send\_message\_backward = true$;
else if $phase == \texttt{STOP}$,
if $\mathfrak{I}.at\_stopping\_place$, $\sigma = t_{dwell}(\, t_{dwell} \in \mathfrak{I})$, $phase = \texttt{DWELL}$,
clear information of the station: $\mathfrak{I} = \varnothing$,
else if $\mathfrak{I}.scheduled\_waiting$, $\sigma = t_{wait}(\, t_{wait} \in \mathfrak{I})$, $phase = \texttt{WAIT}$,
else, $\sigma = \infty$,
else if $phase \in \{\texttt{DWELL}, \texttt{WAIT}\}$, $\sigma = 0$, $phase = \texttt{START}$;
else $\sigma = \infty$;

$\lambda(send\_message\_forward,\ send\_message\_backward) :=$
1. if $send\_message\_forward$, send message forward: $(p, m) = (O/F, m(d = -p))$,
2. if $send\_message\_backward$, send message backward: $(p, m) = (O/B, m(d = p - vl))$;

$ta = \sigma$.

In *compute movement* $(\mathfrak{M}, phase) = \mathfrak{g}(\mathfrak{I}, \mathfrak{V}, IS)$, the `RailVehicle`'s future movement-trajectory is computed based on the information of the next infrastructure and the preceding vehicle (if it exists). A movement-trajectory may consist of several movements. Within each movement the acceleration remains constant. As each track segment has only one speed limit, one movement-trajectory is rather simple. Let's first consider the situation without a preceding vehicle as shown in Fig. 4. A vehicle is at track segment TS_i, and there is a track segment TS_j at distance d which requires a speed change. The speed limit of TS_j, is called *maximum exit speed* v_{ex} (Hansen and Pachl 2008). The current speed limit is v_{max}. The general form of the movement-trajectory consists of three movements, i.e.

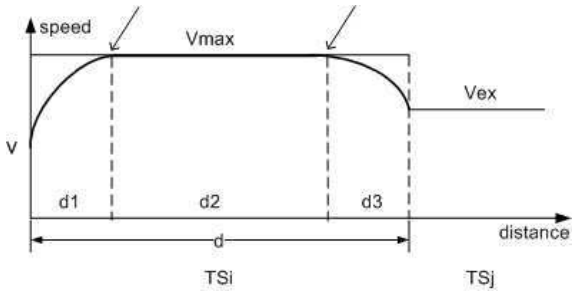


Figure 4: The general form of the movement-trajectory.

acceleration, cruising, and braking. If the current speed v of the vehicle equals to v_{max}, or v_{ex} is not less than v_{max}, the part of d_1 or d_3 falls out. In other cases, if the sum of d_1 and d_2 is greater than d, the vehicle is unable to accelerate until v_{max}. Thus the movement-trajectory may have only one acceleration or braking curve, or nothing at all if the vehicle is unable to drive (e.g. v_{ex} is zero in case of a red traffic signal). The next *phase* can be set to `MOVE_TO_NEXT_TRACK` or `STOP` after the computation. When the vehicle has an acceleration change, it sends a message backward to inform the succeeding vehicle (if any). Before completing one movement-trajectory, if the vehicle receives a message from the next infrastructure or a preceding vehicle (which indicates their state change), the movement-trajectory will be recomputed.

With a preceding vehicle, both vehicles' speed, acceleration, and their distance are needed to first compute if the succeeding vehicle needs to brake (to avoid a collision with the preceding vehicle) before it reaches the next infrastructure. If not, the preceding vehicle is momentarily irrelevant so that the movement-trajectory is computed as described in the previous paragraph. Otherwise, if the anticipated collision point is not farther than the next infrastructure, the distance to the collision point that deducts a safety distance is used as d and the preceding vehicle's final speed is used as v_{ex} to compute the movement-trajectory. The next *phase* may be set to `FOLLOW` or `STOP`. If the collision point is farther than the next infrastructure, the speed at which the preceding vehicle enters the next infrastructure plus a safety distance is used as v_{ex} of the next infrastructure to compute the movement-trajectory. In this case, the *phase* may be set to `MOVE_TO_NEXT_TRACK` or `STOP`.

When a vehicle is at *phase* `STOP`, it can have a transition to *phase* `DWELL` or `WAIT` (a timed waiting), or enter a passive mode waiting for the arrival of a backward message that indicates the vehicle can drive again, e.g. the preceding vehicle starts to drive or the traffic signal turns green. At the end of a movement-trajectory, if the vehicle reaches the next infrastructure, it links itself to the next track $\mathfrak{h}(\mathfrak{I})$. The couplings to the current track will be disconnected, the atomic `RailVehicle` model is removed from its parent model and added to the parent model of the next track, new couplings to the next track are set up, and the position on the new track is set to zero. At this moment, v_{ex} becomes v_{max}, and the vehicle sends a request-message (forward) in order to get new information, i.e. among others, the new v_{ex} of a next track. Its *phase* is set to `IDLE`, which implies that it will get a reply-message in zero (simulation) time.

3.3 Track Segment

The task of a `TrackSegment` is partly introduced in Section 3.1. A track replies to a request-message when its speed limit is unequal to the current speed limit of the message sender; if there is any vehicle linked to it, the track forwards the request-message to the vehicle closest to its start node. Otherwise the message is propagated to the next track. The message propagation stops when a track and a vehicle replied to the request-message, or when a track replied to the message and no vehicle is found within a predefined extra range. For a backward message, if the sender is a vehicle, the message is forwarded back to the vehicle closest to the sender; if the sender is a track, the message is forwarded back to the vehicle which is the original sender of the request-message. The `TrackSegment`'s ports are shown in Fig. 3 (B). The model has four ports dedicated for message propagation (shown with black headed arrows), input forward and backward (I/F, I/B), output forward and backward (O/F, O/B) which are linked to the previous and next track segments; and it has one extra output port (O/V) that sends messages to the vehicles linked to it. A vehicle sends messages to the track through I/F or I/B. The DEVS formalism with port specifications is $\texttt{TrackSegment} = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$ where

$X = \{(p,m) | p \in \{I/F, I/B\}, m \in X_p\}$ is the set of input ports and messages;

$Y = \{(p,m) | p \in \{O/F, O/B, O/V\}, m \in Y_p\}$ is the set of output ports and messages;

$S = X_p \times L$ ($L = \{V_0, V_1, \ldots, V_n\}, n \in \{\mathbb{N} \cup \varnothing\}$, is the set of vehicles on the track with a total order);

$\delta_{ext}(L, (p,m)) :=$

1. $\sigma = 0$,
2. if $p == I/F$,

if $m.sender \in L$,
  if $\exists$ *predecessor of m.sender* $\in L$,
    (1) if *m.sender's_preceding_vehicle_not_found*,
      a. $m(receiver = predecessor, d = m.d + predecessor.position - predecessor.length)$,
      b. $send\_message\_to\_vehicle = true$,
    (2) if *m.sender's_next_track_not_found*, $send\_message\_forward = true$,
  else, $send\_message\_forward = true$,
else,
  (1) if *m.sender's_next_track_not_found*,
    if $this.v_{max} \neq m.sender.v_{max} \wedge this.v_{max} \neq \infty$,
      (1) $m(sender = this, contemplated\_receiver = m.sender, d = 0)$,
      (2) $send\_message\_backward = true$,
    else, $send\_message\_forward = true$,
  (2) if *m.sender's_preceding_vehicle_not_found*,
    if $L \neq \varnothing$,
      (1) $m(receiver = V_n, d = m.d + V_n.position - V_n.length)$,
      (2) $send\_message\_to\_vehicle = true$,
    else if *m.sender's_next_track_found* $\wedge$
    $m.d - m.sender.d_l > m.sender.d_{braking} + m.sender.d_{safety}$,
      //do nothing (stop message propagation),
    else, $send\_message\_forward = true$,
else if $p == I/B$,
  if $L == \varnothing$, $m(d = m.d + this.length)$, $send\_message\_backward = true$,
  else,
    if $m.sender.type == vehicle$,
      if $m.sender \in L$,
        if $\exists$ *successor of m.sender* $\in L$,
          $m(receiver = successor, d = m.d - successor.position)$,
        $send\_message\_to\_vehicle = true$,
        else, $send\_message\_backward = true$,
      else, send message to $V_0 \in L$,
    else,
      if $m.contemplated\_receiver == \varnothing$, send message to $V_0 \in L$,
      else if $m.contemplated\_receiver \in L$,
        (1) $m(receiver = m.contemplated\_receiver, d = m.d + this.length - receiver.length)$,
        (2) $send\_message\_to\_vehicle = true$,
      else, $m(d = m.d + this.length)$, $send\_message\_backward = true$;

$\delta_{int} := \sigma = \infty$;

$\lambda(send\_message\_forward, send\_message\_backward, send\_message\_to\_vehicle) :=$

1. if $send\_message\_forward$, send message forward: $(p,m) = (O/F, m(d = m.d + this.length))$,
2. if $send\_message\_backward$, send message backward: $(p,m) = (O/B, m)$;
3. if $send\_message\_to\_vehicle$, send message to vehicle: $(p,m) = (O/V, m)$;

$ta = \sigma$.

3.4 Sensor, Signal, Control Unit, and Coupled Models

A `Sensor`'s ports are shown as Fig. 3 (C). Compared to a `TrackSegment`, it has one extra output port, the sensor output (O/S). A `Sensor` is a specialization of a `TrackSegment` with one additional function: when a vehicle is linked to the sensor, it triggers the sensor, and a message is sent at the output port O/S to inform, e.g. a control unit, about this activity. At the same time, an internal transition of the sensor is scheduled, based on the vehicle's speed, acceleration, and length (plus an offset), when the vehicle will release the sensor. If meanwhile the vehicle changes its movement (i.e. acceleration), the sensor will get the vehicle's backward message and correspondingly reschedule the internal transition of sensor release. At the occurrence of the internal transition, a notify message is sent out at port O/S.

A signal (`LinesideSignal`) is a specialized `Sensor` with an extra input port from a control unit (I/CU); see Fig. 3 (D). Through this port, the signal receives messages which inform the signal how to change its state, e.g. from red to green or vice versa. Once its state is changed, the `LinesideSignal` sends a message backward to inform the closest vehicle (approaching). A signal has the functions of a sensor, since the information about a vehicle passing the signal is often needed. A `ControlUnit`

has its input port (I/S) coupled to one or more `Sensors`, and its output port (O/LS) coupled to one or more `LinesideSignals`; see Fig. 3 (E). Based on the sensor information, the `ControlUnit` computes the state changes of the signals. A control unit is typically placed in a block section or an intersection.

Sensors, signals, and control units are sub-components of coupled infrastructure components such as stops, stations, intersections, and block sections. An infrastructure component is constructed by organizing the coupling relation of its sub-components, and defining the control logic in the control unit. A block section, for example, is composed of one signal at the entrance of the section, and one or more track segments that constitute the section area. Its control unit is defined differently if the section applies one-block or multi-block signaling (Pachl 2002).

4 MODEL EXAMPLE AND PERFORMANCE ASSESSMENT

In this section, an example of The Hague city center tunnel in The Netherlands (Fig. 5) is chosen to illustrate the use of LIBROS-II components and to compare the model performance with LIBROS. Although simple, the example is sufficient to show the typical structure of a LIBROS-II model. Only one direction of the traffic current is shown (from left to right). In the tunnel, there is a stop with two stopping places (TS_2 and TS_3). The stop is guarded by a signal, which is red when either TS_1 or the second stopping place (TS_2) is occupied by a tram. If the first stopping place (TS_3) is occupied and both the TS_1 and TS_2 are cleared, the signal is yellow. If the block section is completely cleared, the signal turns green. The speed limit in this tunnel is 45 km/h, and it is reduced to 35 km/h if the signal is yellow.

The model structure specified in LIBROS-II is illustrated in Fig. 6. The boxes with bold lines are coupled models. The *Top Level Model* contains a *Block Section*, two *Track Segments*, and a *Source* and a *Sink* which respectively generates and removes vehicles. The *Block Section* has a *Signal*, a *Control Unit*, etc. Besides vehicle position detection, the *Sensor* in the *Stop* is used by the vehicles to identify the position of the stopping places before the vehicles reach the *Stop*. An extra *Sensor* is placed at the end of the block section to detect clearance.

The example is also modeled with the LIBROS library in order to compare their performance. The model with LIBROS has one *Track*. The positions of the other components are relative to the track using offsets. The block section is a component in the physical layer of the library, and its control logic is defined in the control layer; see e.g. Kanacilo and Verbraeck (2005).

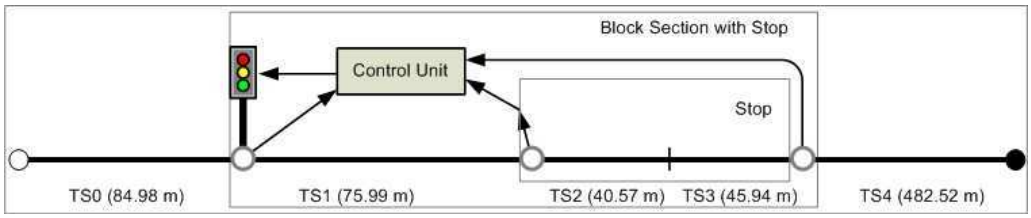


Figure 5: A model example - The Hague city center tunnel.

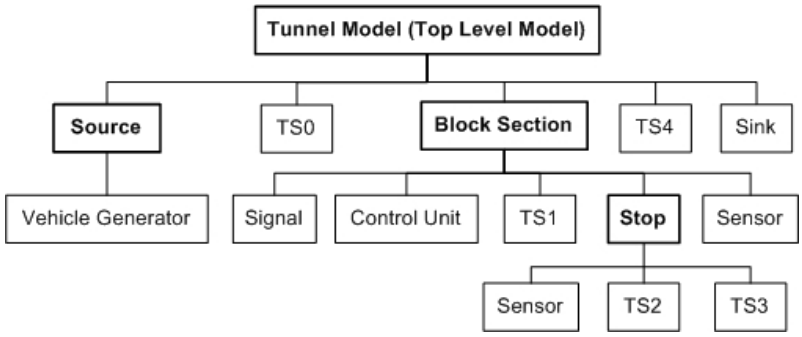
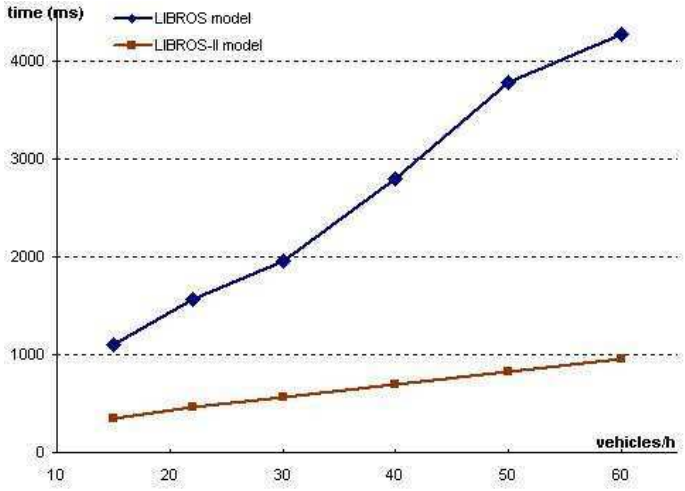
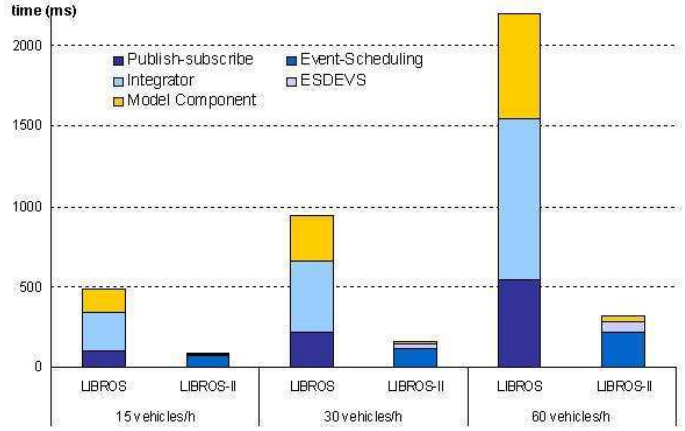


Figure 6: The tunnel model structure with LIBROS-II.



(a) Model execution time.



(b) Model execution time per category.

Figure 7: Performance comparison of the LIBROS and LIBROS-II model examples.

Experiments with different vehicle generation frequencies (15 to 60 vehicles/h) are run with both models. These frequencies represent the lower and higher bounds of the traffic intensity of tram operation in the city of The Hague. The simulation run length is 2 hours. The vehicles in the LIBROS model are simulated with an integration step of 0.05 seconds to obtain a train positioning accuracy of 0.625 meters in the worst case, considering the maximum speed of 45 km/h. The train positioning in LIBROS-II is computed as described in Section 3.2 with floating point precision. As shown in Fig. 7 (a), within the boundary of the experimentation, the execution times of both the LIBROS and LIBROS-II models evolve linearly as a function of the vehicle generation frequency. However, the latter is consistently lower compared to the former with a factor of 5.

The execution times (for the frequencies 15, 30, and 60 vehicles/h) are categorized by library package as shown in Fig. 7 (b). (The three highest categories are shown.) The underlying simulation mechanisms (namely the integrator and the event-scheduler), communication schemes, and the model components in both models are compared. In the LIBROS model, the general discrete behavior is handled by *model component*s; specific vehicle movement computation depends on the *integrator*; and the object-to-object communication is carried out by *publish-subscribe*. The latter two are contained in DSOL. In the LIBROS-II model, the general behavior is handled by *model component*s; object-to-object communication is carried out by ESDEVS; and *event-scheduling* is performed by DSOL. In Fig. 7 (b), the LIBROS model components hold a higher percentage of the total execution time than those of LIBROS-II. The communication in the LIBROS-II models (by message exchange through ports) produces less overhead than using publish-subscribe. As expected, the integrator is computationally

more demanding than the event-scheduler. It is the primary component in the LIBROS model that occupies the highest portion of the execution; so is the event-scheduler in LIBROS-II. Although both components play a comparable role in each model, the design choice of a continuous model such as in LIBROS has little influence on the efficiency of the numerical solution. On the contrary, the efficiency of event-scheduling as that used in LIBROS-II is a result of the model design and the event-triggering strategy. This opens up possibilities for further research to improve model design. Although the execution time in the example counts in seconds, in large-scale railway simulation e.g. of a city or a country, efficiency becomes important when the model is microscopic and high quality data visualization of the simulation is required.

5 CONCLUSIONS AND FUTURE RESEARCH

In this paper, we reviewed the LIBROS library, in which the train movement is represented by differential equations and the other railway components are modeled with discrete-event abstraction. The LIBROS-II library is introduced, where the DEVS formalism is used for the railway model specification. An example is modeled with both libraries, and their performances are compared. We conclude that, with comparable model detail and accuracy, the LIBROS-II models specified by the DEVS formalism yield a higher performance than the LIBROS models. Modularity of the model component can be obtained through a careful object-oriented analysis and design; however, by following the DEVS formalism, modularity is guaranteed by allowing message based communication between components. Future research will focus on assessing the scalability of the approach by implementing a large-scale railway network. With its high performance, one can envision an integration of the simulation library within advanced real-time applications such as traffic control systems and interactive training games.

ACKNOWLEDGMENTS

The authors would like to acknowledge the support of HTM Urban Public Transport, The Hague, the Netherlands.

REFERENCES

Baldwin, C. Y., and K. B. Clark. 2000. *The power of modularity*, Volume 1 of *Design Rules*. MIT Press.

Eugster, P. T., P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. 2003. The many faces of publish/subscribe. *ACM Computing Surveys* 35 (2): 114–131.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1994. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Giambiasi, N., B. Escude, and S. Ghosh. 2000. Gdevs: A generalized discrete event specification for accurate modeling of dynamic systems. *Transactions of the Society for Computer Simulation* 17 (3): 120–134.

Hansen, I. A., and J. Pachl. (Eds.) 2008. *Railway timetable & traffic: Analysis-modelling-simulation*. Eurailpress.

Jacobs, P. H. M. 2005. *The DSOL simulation suite - enabling multi-formalism simulation in a distributed context*. Ph. D. thesis, Delft University of Technology, the Netherlands.

Jacobs, P. H. M., N. A. Lang, and A. Verbraeck. 2002. D-SOL: A distributed java based discrete event simulation architecture. In *Proceedings of the 2002 Winter Simulation Conference*, ed. J. L. S. E. Yücesan, C.-H. Chen and J. M. Charnes, 793–800: IEEE.

Kanacilo, E. M., and N. v. Oort. 2008. Using a rail simulation library to assess impacts of transit network planning on operational quality. In *WIT Transactions on the Built Environment*, Number 103, 35–43. WIT Press.

Kanacilo, E. M., and A. Verbraeck. 2005. A distributed multi-formalism simulation to support rail infrastructure control design. In *Proceedings of the 2005 Winter Simulation Conference*, 2546–2553: IEEE.

Kanacilo, E. M., and A. Verbraeck. 2006. Simulation services to support the control design of rail infrastructures. In *Proceedings of the 2006 Winter Simulation Conference*, 1372–1379: IEEE.

Kanacilo, E. M., and A. Verbraeck. 2007. Assessing tram schedules using a library of simulation components. In *Proceedings of the 2007 Winter Simulation Conference*, 1878–1886: IEEE.

Kofman, E. 2003. Quantization-based simulation of differential algebraic equation systems. *Simulation* 79 (7): 363–376. Cited By (since 1996): 4.

Li, K.-P., and Z.-Y. Gao. 2007. An improved equation model for the train movement. *Simulation Modelling Practice and Theory* 15 (9): 1156 –1162.
Li, K.-P., Z.-Y. Gao, and B. Ning. 2005. Cellular automaton model for railway traffic. *Journal of Computational Physics* 209 (1): 179 –192.
Li, K.-P., B.-H. Mao, and Z.-Y. Gao. 2009. An improved walk model for train movement on railway network. *Communications in Theoretical Physics* 51 (6): 979–984.
Lu, Q., M. Dessouky, and R. C. Leachman. 2004. Modeling train movements through complex rail networks. *ACM Transactions on Modeling and Computer Simulation* 14 (1): 48–75.
Middelkoop, D., and M. Bouwman. 2001. Simone: Large scale train network simulations. In *Proceedings of the 2001 Winter Simulation Conference*, 1042–1047: IEEE.
Pachl, J. 2002. *Railway operation and control*. VTD Rail Publishing.
Rudolph, R. 2000. Operational simulation of light rail systems. In *Proceedings of the European Transport Conference*, Number 167-178.
Seck, M. D., and A. Verbraeck. 2009. DEVS in DSOL: Adding DEVS operational semantics to a generic event-scheduling simulation environment. In *Proceedings of the 2009 Summer Computer Simulation Conference*.
Sullivan, K. J., W. G. Griswold, Y. Cai, and B. Hallen. 2001. The structure and value of modularity in software design. *ACM SIGSOFT Software Engineering Notes* 26 (5): 99–108.
Wainer, G. 2007. Developing a software toolkit for urban traffic modeling. *Software - Practice and Experience* 37 (13): 1377–1404.
Zeigler, B., and J. Lee. 1998. Theory of quantized systems: Formal basis for DEVS/HLA distributed simulation environment. In *Proceedings of SPIE - The International Society for Optical Engineering*, Volume 3369, 49–58.
Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*. 2nd ed. Elsevier/Academic Press.

AUTHOR BIOGRAPHIES

YILIN HUANG is a Ph.D. Candidate in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology. Her research interests include dynamic data-driven simulation, on-line data analysis, and transportation systems simulation. Her email address is <y.huang@tudelft.nl>.

MAMADOU D. SECK is an Assistant Professor in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology. He received his Ph.D. from the Paul Cézanne University of Marseille, France. His research interests include modeling and simulation formalisms, dynamic data-driven simulation, human behavior simulation, and agent directed simulation. His email address is <m.seck@tudelft.nl>.

ALEXANDER VERBRAECK is a Full Professor in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology, and a part-time Full Professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete-event simulation for real-time control of complex transportation systems and for modeling business systems. His current research focuses on development of object-oriented simulation building blocks, participative modeling, serious gaming using virtual reality, and agent technology in simulation. His email address is <a.verbraeck@tudelft.nl>.