A DEVS library for rail operations simulation

Huang, Y; Seck, MD; Verbraeck, A

Publication date
2011

Document Version
Accepted author manuscript

Published in
Proceedings of the 2011 Spring simulation Multiconference, Emerging M&S Applications in Industry and Academia Symposium

Citation (APA)
Huang, Y., Seck, MD., & Verbraeck, A. (2011). A DEVS library for rail operations simulation. In A. Tolk (Ed.), *Proceedings of the 2011 Spring simulation Multiconference, Emerging M&S Applications in Industry and Academia Symposium* (pp. 76-83). Society for Computer Simulation/ACM.

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

A DEVS Library for Rail Operations Simulation

Yilin Huang, Mamadou D. Seck and Alexander Verbraeck
Systems Engineering Group
Faculty of Technology, Policy and Management
Delft University of Technology
PO Box 5015, NL-2600GA Delft
The Netherlands
{y.huang,m.d.seck,a.verbraeck}@tudelft.nl

Keywords: simulation library, DEVS, rail transportation

Abstract

Detailed yet computationally efficient simulation models are needed to support the design and operation of modern rail infrastructure systems. LIBROS-II is a model component library for microscopic rail operations simulation. Basic rail elements are modeled as atomic DEVS models which are further aggregated into more elaborate rail component models. The latter can in turn be used modularly for the composition of rail network systems of arbitrary complexity in a detailed, efficient, and rigorous way. This paper explains the communication principles in the model and gives an overview of vehicle detection and control system simulation. To enhance its usability, LIBROS-II is augmented with CRMB, a model generator capable of inferring structural and behavioral features of a rail network from standard CAD data.

1. INTRODUCTION

Modeling and Simulation (M&S) of transport systems have had important developments since the mid 1970s and are now better recognized by transport designers as an effective decision support instrument [Ortúzar and Willumsen, 2001]. As working with complex infrastructure networks increasingly becomes a standard approach, in order to successfully support design and operation, a microscopic rail network model is often deemed not only suitable but also mandatory [Hansen and Pachl, 2008]. Due to the long life span of the infrastructure and services, changes often take place, leading to new issues to study, which can require the construction or alteration of simulation models. Therefore, model construction of rail-based networks particularly requires malleable model composition and configuration to enhance flexibility and reusability. A powerful approach is to exploit model modularity, and to hierarchically construct large and complex models by means of simpler model components. In this respect, Hu et al. [2005] discussed the suitability of the DEVS (Discrete Event Systems Specification) formalism [Zeigler et al., 2000] for component-based M&S.

LIBROS (LIBrary for Rail Operations Simulation) is a rail model component library. Its models describe the railway operation at a microscopic level [Hansen and Pachl, 2008]. Following the DEVS formalism, LIBROS-II defines *Rail Element*s, such as rail vehicles, sensors, signals and tracks, as atomic models. They can constitute more complex (coupled) *Rail Component*s, such as stations and block sections, which in turn can be composed, and so forth recursively. The motivation of developing a rail simulation library based on the DEVS formalism is discussed in Huang et al. [2010b]. Huang et al. [2010a] compared microscopic rail simulation models in which train movements are represented by differential equations and by discrete-event abstraction. The results show that, with comparable model detail and accuracy, the LIBROS-II model yields a higher performance than the model using differential equations. In this paper, the design of LIBROS-II is discussed. The next section briefly reviews the underlying simulation environment of the LIBROS-II library. Section 3 presents the communication mechanism used in the library. Vehicle detection and control are very important for railway safety. The modeling of this part is discussed in section 4. Section 5 shows the model generation process.

2. DSOL, ESDEVS AND LIBROS-II

The underlying simulator of LIBROS-II is DSOL, the Distributed Simulation Object Library [Jacobs et al., 2002; Jacobs, 2005]. LIBROS-II defines rail model behaviors, which are executed by the DSOL simulator. As such, the separation of concerns between models and simulators is respected. DSOL is a simulation environment that supports continuous and discrete-event simulation. It consists of components including an event-scheduler, numerical integrators, and probability distributions. The work described in Seck and Verbraeck [2009], i.e., the Event-Scheduling DEVS (ESDEVS) library, implements the parallel DEVS formalism on top of the DSOL library. 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 (except for confluent transition situations). Dynamic structure DEVS [Barros, 1995] is also implemented in the ESDEVS library so that components and coupling relations can

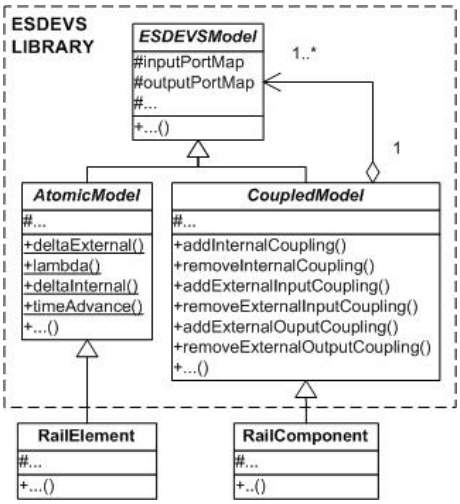

Figure 1. Atomic and coupled models in ESDEVS.

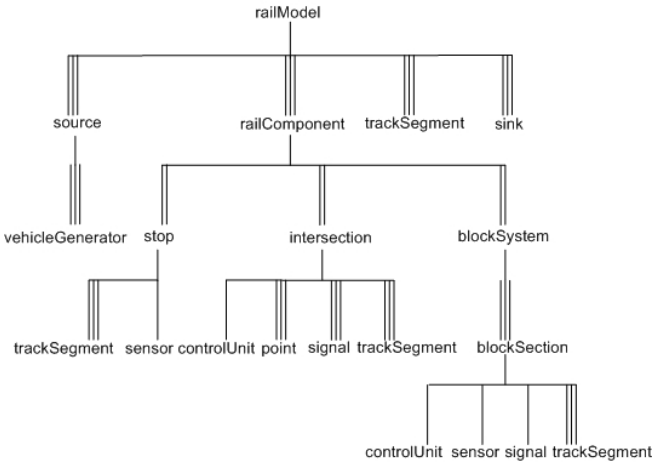

Figure 2. SES of the rail infrastructure model.

be added and removed dynamically during simulation runtime. On the whole, the library specifies the meta structure of atomic and coupled DEVS models, and handles the couplings, output function, transition functions at a high level, so that ESDEVS (together with DSOL) serves as a DEVS simulator. By subclassing *AtomicModel* and *CoupledModel* in the ESDEVS library, users can specify model behaviors and coupling relations[1]. Figure 1 shows a simplified example.

LIBROS-II is built on the basis of ESDEVS. Rail elements are irreducible atomic models, e.g., a vehicle, a sensor, a signal or a piece of track segment. Rail components are resultants of composition. In Figure 2, a rail infrastructure model is schematized in System Entity Structure (SES) [Zeigler and Hammonds, 2007]. A top level rail model contains one or more sources and sinks where vehicles are generated and removed from the simulation. The rail model composition may contain stops, intersections, block systems, etc. For instance, an intersection has signals to control the accessibility of the track sections; points allow vehicles to move from one track to another; a control unit computes the signalling logic depending on the occupancy of the tracks and points. Each rail element models one functionality of the rail infrastructure, and the aggregation of rail elements forms a rail component that performs more complex tasks.

The rail network model, at its lowest description level, is a directed cyclic graph of linked rail infrastructure elements (track segments, sensors, points and signals). Many DEVS traffic simulation models, e.g. Lee et al. [2004]; Wainer [2006], represent vehicles as messages (or objects) being transferred from one infrastructure model to another. We choose to represent the vehicles as atomic models that have autonomous behaviors. As such, the individual vehicle's profiles (driving behavior, capacity, energy consumption, etc.) could be modeled, which is an interesting asset for detailed transportation studies.

In LIBROS-II, a vehicle model is linked (or coupled) directly with an infrastructure element. Each infrastructure element is capable of conditional message propagation, the main object-to-object communication mechanism used by the LIBROS-II models. Its basic concept is message-passing through paired I/O ports in DEVS. A vehicle moves from one place to another while driving in the rail network; this behavior is modeled by successively coupling the vehicle model from one rail infrastructure element to another using dynamic structure DEVS. A model design question is then when to change the couplings and how the vehicles' internal states change accordingly. These computations depend on the vehicle's circumstances, e.g., if there is a preceding vehicle or where the next traffic light is located. In order to model such detailed interactions, the communication mechanism among the rail elements is the essence of obtaining the necessary information.

3. COMMUNICATION MECHANISM

There are two main types of communications in a LIBROS-II model, vehicle to vehicle (V2V) and vehicle to infrastructure (V2I). As the vehicles' positions are dynamic, the establishment of communication channels between the rail elements at one instant generally requires two steps: (A) a vehicle finds its preceding vehicle/infrastructure; (B) then the coupling between the two rail elements can be created. These two tasks may be accomplished in different ways, depending on how the model (and its data structure) is designed.

One possibility is to set up point-to-point couplings. For each vehicle model, step (A) can be reduced to the breadth-first search (BFS) in the rail infrastructure graph. Constrained by the DEVS formalism, step (B) requires first the knowledge

[1]For more information about DSOL and ESDEVS: `http://simulation.tudelft.nl`.

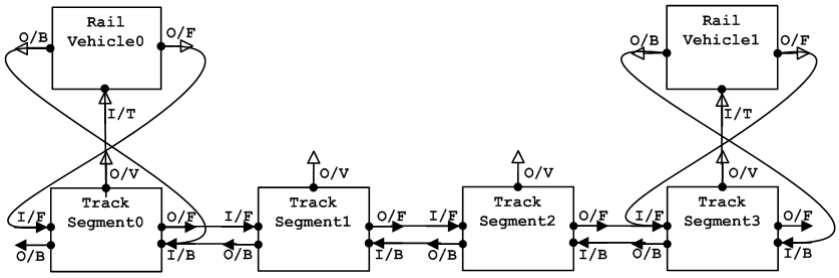

Figure 3. A coupling example.

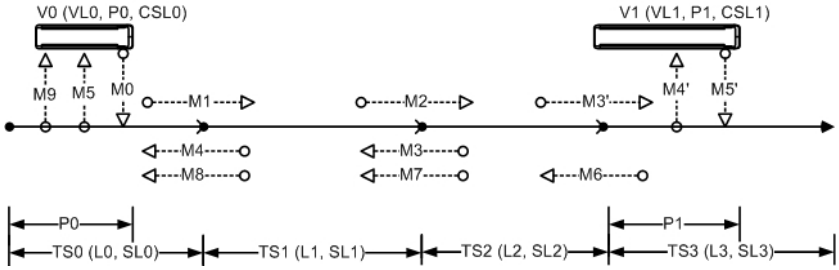

Figure 4. A message propagation example.

of the common parent (coupled) model of the two atomic models. This can be solved by a lowest common ancestor (LCA) finding algorithm in the DEVS model structure tree. It is well known that both problems have linear time and space complexities [e.g. Cormen et al., 2001; Alstrup et al., 2004][2]. The removal and setup of couplings require the same time complexity. Therefore, setting up (and removing) couplings of vehicles and infrastructures in the rail network has a complexity of $(BFS + LCA + Coupling) \times number\_of\_vehicles \times (number\_of\_infrastructures + state\_change\_frequency) = O(n+n+n) \times n \times n \sim O(n^3)$.

Another possibility is to set up indirect coupling relations. If each vehicle is coupled with an infrastructure element, and the infrastructure elements have been coupled corresponding to the infrastructure layout, then the V2V and V2I communications could be achieved by conditional message propagation through the intermediary infrastructure elements. With such an approach, finding the nearby vehicle and infrastructure could be naturally solved without search functions. Only the direct couplings between each vehicle and the infrastructure element are dynamic. Such coupling has a complexity of $coupling\_infrastructures + number\_of\_vehicles \times number\_of\_infrastructures = O(n+n^2) \sim O(n^2)$.

Therefore, we choose to use indirect coupling with conditional message propagation as the communication mechanism for LIBROS-II models. Each infrastructure element is capable of message propagation, which can be along the traffic current (forward) or in the opposite direction (backward). A vehicle determines its movement (acceleration, deceleration or cruising) based on the information about the next infrastructure and/or the preceding vehicle. Lacking such information, the vehicle sends a request-message forward. The infrastructure element that receives the message propagates the message until the next infrastructure of interest and a preceding vehicle is found. If no preceding vehicle is found, the message propagation stops after a predefined distance. The found vehicle, if any, or infrastructure sends a response-message, which is propagated backward until reaching the original sender of the request-message. On receiving the message, a vehicle calculates its movement. At the end of the movement, the outdated information is cleared and a new request-message is sent out.

A simplified coupling example is illustrated in Figure 3, and its message propagation is in Figure 4. The example 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). It has dedicated ports for input (I) and output (O) forward (F) as well as backward (B), and a port for output to vehicles (V). A `RailVehicle` has its vehicle length (VL), position (P) relative to the track segment it is coupled to, and its current speed limit (CSL). (The other attributes are not illustrated.) It also has ports for sending and receiving messages. Suppose that V_0 doesn't have information about its next infrastructure nor about the preceding vehicle; it sends a request-message forward. Two message sequences will be triggered 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 $SL_1 = CSL_0$ and $SL_2 \neq$

[2]There exist several fast algorithms; however the complexity can be said to be of order $O(n)$.

Table 1. Message initiators

Message Initiator	Message Type	Direction	Trigger Event/Condition
vehicle	request-message	forward	when the vehicle has no information about its next infrastructure and/or the preceding vehicle
vehicle	response-message	backward	when the vehicle receives a request-message
vehicle	update-message	backward	when the vehicle changes its acceleration/deceleration rate
track segment	response-message	backward	(1) when the track segment receives a request-message and the vehicle's next infrastructure of interest is not yet found (2) when the track segment has a different speed limit or the message propagation distance exceeds a predefined value
sensor/point/signal	response-message	backward	when the sensor/point/signal receives a request-message and the vehicle's next infrastructure of interest is not yet found
signal	update-message	backward	when the signal changes its state (e.g. a traffic light turns from red to green)

CSL_0. A `TrackSegment` replies to a request-message when there is a speed limit change, and it also delivers a request-message to a vehicle closest to its start node if there is any vehicle linked to it, otherwise it propagates a message to the next infrastructure element. Thus in the example, TS_0 and TS_1 propagates the message and TS_2 replies to the message. TS_2 also propagates the message to TS_3, because V_0's preceding vehicle is not yet found. TS_3 has V_1 linked to it, so it delivers the message to V_1. The message propagation at the forward direction ends at this moment. V_1 receives a request-message, and sends a response-message backward to V_0. 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 sequence (2), e.g., 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. Message propagation involving sensors, points and signals follows the same principle. The only difference is that they always reply to request-messages.

As mentioned earlier, a vehicle can be a message initiator, i.e. it can initiate message propagations according to the circumstances. The infrastructure elements can play the similar roles. Table 1 summarizes the possible message types, directions and the triggering events and conditions. The infrastructure elements are the message propagator and deliverers. The propagation and delivering rules (ordered with priority) are as following. **Request-Message**: (1) deliver to the closest vehicle, if the infrastructure element has any vehicle coupled to it and the preceding vehicle was not found; (2) propagate when an infrastructure of interest is not found; (3) propagate when an preceding vehicle was not found and the propagation distance is less than a predefined value; (4) stop propagation when the message propagation distance exceeds a predefined value. **Response-Message**: deliver when the infrastructure element has the contemplated receiver vehicle coupled to it, otherwise propagate. **Update-Message**: (1) stop propagation when the message propagation distance exceeds a predefined value; (2) deliver to the closest vehicle when the infrastructure element has any vehicle coupled to it, otherwise propagate. The conditional message propagation (and delivery) are performed with zero time advance, i.e. when a vehicle sends a request-message, it receives one or two response-messages in zero simulation time.

4. VEHICLE DETECTION AND CONTROL

In this section, the (atomic) sensor, point, signal, and control unit models are presented. Compared to the track segment models, the sensor, point and signal models have more interaction with the vehicle models. Besides the message propagation capability, they also can be triggered and released by the vehicle models. In combination with control units, they can specify sophisticated railway operation and control rules.

The occupation and clearance detection of tracks is fundamental to railway block control and interlocking [Pachl, 2002]. There are different technologies for many types of detection devices [Theeg and Vlasenko, 2009]. To generalize detectors' functionalities, three types of atomic models are designed that have detection capacity (referred to hereinafter as the detector models or detectors): the sensor model (detection), the point model (detection and switching), and the signal model (detection and signalling). The three most important detection purposes are (1) detecting vehicles reaching a certain point with its front end, (2) detecting vehicles passing a certain point with its rear end, and (3) track occupancy detection [Theeg and Vlasenko, 2009]. In the detector models, purpose (1) is fulfilled by triggering the detectors, purpose (2) by releasing the detectors, and purpose (3) by placing two detectors at the two ends of a track.

When a vehicle is approaching a detector, the latter replies the former's request-message. The vehicle then calculates its movement from its current location until reaching the detector. Upon ending the movement, the vehicle's couplings to the former infrastructure element are removed, and the new

couplings to the detector are created. At this moment, the vehicle initiates another request-message in order to calculate its movement to the next infrastructure element. The detector will receive the message and propagates it. Additionally the detector is triggered and it will schedule the release time (based on the vehicle's length, speed and acceleration) for the next internal transition. Before the scheduled release time expires, if the vehicle changes its acceleration, it sends an update-message backward to inform the succeeding vehicle (if any). The triggered detector will get the (backward) message during message propagation, and it reschedules the release time for the next internal transition.

Block control and interlocking safeguard the railway by protecting the following and opposing movements. The detectors gain the information about vehicle positions, and transmit it to the control unit. The latter evaluates the information and permits vehicle movements via the signals [Theeg and Vlasenko, 2009]. The control unit model represents the control logic of block systems and interlocking, each of whose entrances is guarded by the signals.

Control unit models are not coupled with track segments, i.e. they do not participate in message propagation. They receive input events only from detectors and send outputs to signals. A simple example is shown in Figure 5 (the traffic current is from left to right). Such an arrangement in railway operation is called a block section. The permission of entering the block section is granted by the control unit and indicated by the signal. A vehicle that is moving towards the block section would send a request-message forward. The signal will receive the message eventually and send back a response-message by which the vehicle would know if it is allowed to enter. If not the vehicle has to stop. Otherwise by entering the block section the vehicle will trigger the signal (as a detector) which transmits this information to the control unit. The signal then turns red to block the entrance. When the vehicle leaves the block section, it releases the sensor who notifies the control unit about this event. On receiving the information, the control unit clears the block section by instructing the signal to turn green, so that the next vehicle could enter.

Safety issues render the interlocking control very complex. The information evaluation involves routes, points, sensors, signals, vehicle priorities, etc. Its principles are extensively discussed in the literature, e.g. Pachl [2002]; Hansen and Pachl [2008]; Theeg and Vlasenko [2009], and many works deal with algorithms and methods of interlocking control and

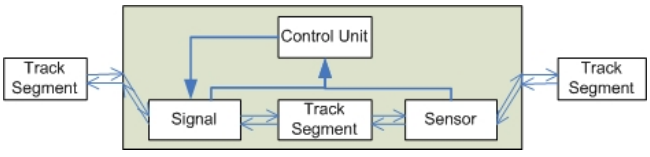

Figure 5. A coupled model with control unit.

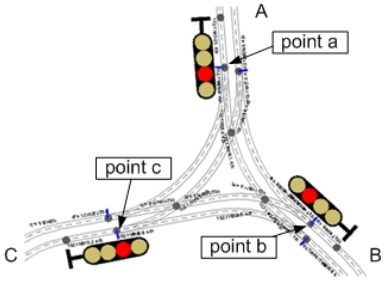

Figure 6. The light-rail infrastructure at a T-intersection.

route searching, e.g. She et al. [2007]; Mirabadi and Yazdi [2008]; Roanes-Lozano et al. [2010]. Therefore we will not delve into the matter. However, we will demonstrate how the control logic can be modeled in DEVS with an example.

Figure 6 shows the infrastructure arrangement for light-rails at a T-intersection with directions A, B and C. The dots on the tracks are the detectors that are coupled to the control unit. The control unit is coupled to the signals and the points (to instruct how they shall switch, i.e. left or right). A route starts always at a signal (the entrance signal of the route) [Pachl, 2002]. It is permitted for access when there is a route request and if all tracks on the route are cleared and all points are properly set. For example, at direction B in Figure 6, B→C and B→A are the two possible routes; route B→C can be used if point b is set to left and the three succeeding detectors are unoccupied. The point position and track clearance are checked upon route request. The requests are processed according to vehicle priorities (if any) and on a first-come first-served (FCFS) basis. Based on these rules, the model of interlocking control unit is specified. The model has one input port and one output port. In principle, it processes and buffers incoming (detector) messages, sends out (control) messages if necessary, and then stays in passivity until receiving another message. $\texttt{InterlockingControlUnit} = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$ where

$X = M_i$ is the set of input messages;

$Y = M_o$ is the set of output messages;

$S = \mathcal{R} \times \mathfrak{P} \times \mathfrak{R} \times \overline{\mathfrak{R}} \times \hat{R}$; $\mathcal{R}(v.id, \mathfrak{s}) = R$ is the map of routes[3], $R = \{\mathfrak{s}, \mathcal{P}, S\}$, $\mathfrak{s}$ is the entrance signal of the route, $\mathcal{P} = \{(p, \bar{p}) | p \in P, \bar{p} \in \{\texttt{LEFT}, \texttt{RIGHT}\}\}$, P and S are the sets of points and sensors in the route; $\mathfrak{P}(v.id, \mathfrak{s}) = \mathfrak{p}$ is the map of priorities; the waiting request (or requester) map $\mathfrak{R}$ is indexed by entrance signals: $\mathfrak{R}(\mathfrak{s}) = (v_0, v_1, \cdots, v_k) \cup \varnothing, k \in \mathbb{N}$, so that $\mathfrak{s} = \mathcal{R}(v_j.id, \mathfrak{s}).\mathfrak{s}$, v_j is sorted by request time, $j \in \mathbb{N}, j \leq k$; $\overline{\mathfrak{R}} = (v_0, v_1, \cdots, v_n) \cup \varnothing, n \in \mathbb{N}$ is the list of waiting requesters ordered so that $\mathfrak{P}(v_{j-1}.id, \mathfrak{s}) \geq \mathfrak{P}(v_j.id, \mathfrak{s}')$, v_j is sorted by request time, $j \in \mathbb{N}, j \leq n$; $\hat{R} \in \mathcal{R}$ is the active route;

$\delta_{ext}(\mathcal{R}, \mathfrak{P}, \mathfrak{R}, \overline{\mathfrak{R}}, \hat{R}, m_i \in M_i, m_o \in M_o) :=$

[3] v stands for *vehicle*.

We denote a map and its mapping function with the same symbol, e.g. $\mathcal{R}(v.id, \mathfrak{s}) = R$ also means $R \in \mathcal{R}$.

if $m_i.approachingVehicle \neq \varnothing$ **then**
  if $m_i.approachingVehicle \in \overline{\mathfrak{R}}$ **then**
    <request already saved, do nothing>
  else
    $v \leftarrow m_i.approachingVehicle$
    $\mathfrak{s} \leftarrow m_i.sender$
    requested route $R \leftarrow \mathcal{R}(v.id, \mathfrak{s})$
    if $\mathfrak{R}(R.\mathfrak{s}) = \varnothing \wedge R.isCleared$ **then**
      $\hat{R} \leftarrow R$
      $\hat{R}.reserve$
      $\sigma \leftarrow 0$
      createMessage
    end if
    saveRequest(v, $\mathfrak{s}$)
  end if
else if $m_i.sender$ is `Signal` **then**
  if $m_i.detectorAction = TRIGGER$ **then**
    $\mathfrak{s} \leftarrow m_i.sender$
    $v \leftarrow \mathfrak{R}(\mathfrak{s}).getFirst$
    $\mathfrak{R}(\mathfrak{s}).remove(v)$
    $\overline{\mathfrak{R}}.remove(v)$
  end if
else if $m_i.detectorAction = RELEASE$ **then**
  point $p \leftarrow m_i.sender$
  $p.releaseReservation$
  if $p.notReserved$ **then**
    processWaitingRequests
    if $requestGranted$ **then**
      $\hat{R}.reserve$
      $\sigma \leftarrow 0$
      createMessage
    end if
  end if
end if;
$\delta_{int}(\sigma) := \sigma \leftarrow \infty$;
$\lambda(m_o) := send(m_o), m_o.receivers \leftarrow \varnothing$;
$ta = \sigma$;
createMessage :=
$m_o.receivers.add(\hat{R}.\mathfrak{s})$
for $\forall e \in \hat{R}.elements$ **do**
  if $e \in \hat{R}.\mathcal{P}.P \wedge e.position \neq \bar{p}$ **then**
    $m_o.receivers.add(e)$
  end if
end for;
saveRequest(v, s) :=
$\mathfrak{R}(\mathfrak{s}).add(v)$
if $\overline{\mathfrak{R}} = \varnothing$ **then**
  $\overline{\mathfrak{R}}.add(v)$
else
  $\mathfrak{p} \leftarrow \mathfrak{P}(v.id, \mathfrak{s})$
  for $i = 0$ **to** $\overline{\mathfrak{R}}.size$ **do**
    $v' \leftarrow \overline{\mathfrak{R}}.get(i)$
    if $\mathfrak{p} > \mathfrak{P}(v'.id, \mathfrak{s}')$ **then**
      $\overline{\mathfrak{R}}.insert(v)$ at position i
    end if
  end for
end if;
processWaitingRequests :=
if $\overline{\mathfrak{R}} \neq \varnothing$ **then**
  requestOrderBuffer $ROB = \varnothing$
  for $\forall \mathfrak{R}(\mathfrak{s}) \in \mathfrak{R}$ **do**
    if $\mathfrak{R}(\mathfrak{s}) \neq \varnothing$ **then**
      $ROB.add(position\_of\_\mathfrak{R}(\mathfrak{s}).v_o\_in\_\overline{\mathfrak{R}})$
      <the first request at each route entrance>
    end if
  end for
  $ROB.sort$ <the smaller the position of $\mathfrak{R}(\mathfrak{s}).v_o$ in $\overline{\mathfrak{R}}$, the higher the priority and the earlier the request time>
  for $i = 0$ **to** $ROB.size$ **do**
    $v \leftarrow \overline{\mathfrak{R}}.get(ROB.get(i))$
    $R \leftarrow \mathcal{R}(v.id, \mathfrak{s})$
    if $R.isCleared$ **then**
      $\hat{R} \leftarrow R$
      $requestGranted \leftarrow TRUE$
      return
    end if
  end for
end if.

An interlocking block is a closed block. This means its entrance signal is by default red. When a vehicle is approaching the entrance signal, the latter replies the former's request-message and sends the route request to the interlocking control unit. If the requested route is not cleared, the request will be queued by priority and request time. The requests are also queued per entrance signal, because only the first request at each entrance signal can be processed. If the requested route is cleared, the route will be reserved. Consequently, the control unit sends a message to the points that are not on the correct position and to the entrance signal. On receiving the message, the points toggle their positions and the signal turns from red to green, so that the vehicle can enter the route safely. When the vehicle enters the route, it triggers the signal which turns back to red. The control unit will receive the trigger message from the signal and removes the request from the queue. Each time when the vehicle releases a detector while moving along the route, the control unit processes the waiting requests and send out a message if a route is cleared.

The algorithm for the interlocking control logic described is applicable for different interlocking topologies. The routes can be obtained by automated route searching. However, the mapping of routes $\mathcal{R}(v.id, \mathfrak{s}) = R$, i.e. which vehicle takes which route, needs to be configured according to the routing plan. In the following section, the model generation process is presented.

5. MODEL GENERATION

Rail infrastructure networks are often very complex. To facilitate the model construction and configuration for infrastructure designers and operators, a CAD Rail Model Builder (CRMB) is developed as one of the possible approaches to interface with the LIBROS-II library. Figure 7 shows the rail model generation process. Many railway companies and authorities use CAD and GIS applications for infrastructure design. The resulting design data can be used to generate the infrastructure model. However, CAD data only contain a list of geometric primitives describing each object [Flynn and Jain, 1991]. Therefore, geometric inference is needed to identify the objects and represent them in a relational graph which contains orientation and connectivity relations among others. Additionally, these relations are transformed into a hierarchical structure in order to generate DEVS models. For example, the track segments shall be oriented in accord with the traffic current (bi-directional is also possible); the point locations are detected where more than two track segments connect; then the point type is determined (whether it is converging or diverging). After these steps pattern recognition techniques are applied to identify the rail (infrastructure) components, e.g. stops, blocks, etc. The identified track segments and points that belong to different rail components are grouped for model generation. The rail infrastructure model is generated and coupled according to the components groups and the relational graph. Other data sources than the CAD data provide the routing information and the timetables. Based on the routing data, the route mapping of intersections and junctions in the rail infrastructure model is configured. And the timetable data are used to configure the vehicle generators.

Figure 8 shows the light rail network model of The Hague and the surrounding area generated by CRMB. The area covers over 150 km^2 with 14 lines, 135 km tracks, and 539 stops. Figure 6 is a representation of the amount of model details which are automatically generated. According to our experience, the manual construction of a microscopic rail model

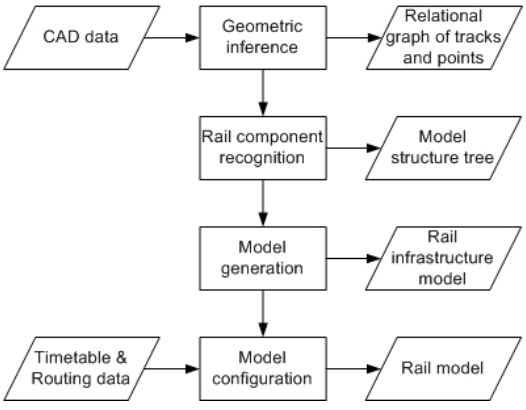

Figure 7. The rail model generation process of CRMB.

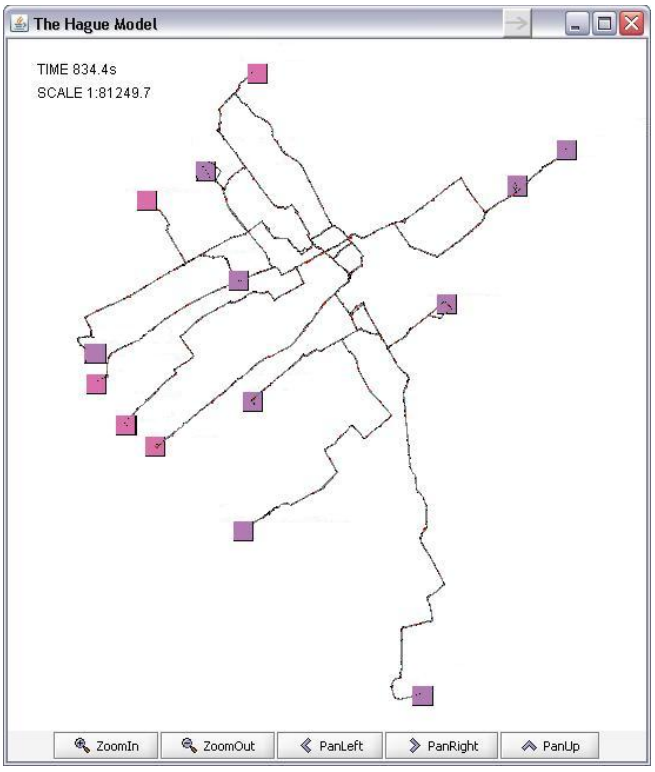

Figure 8. The light rail network model of The Hague.

of 10 km can take 5∼6 weeks. The generation of The Hague model by CRMB takes a few seconds.

6. CONCLUSIONS AND FUTURE WORK

This paper discussed the design of LIBROS-II library, where the DEVS formalism is used for the railway model specification. The models in the library describe the railway operation at a microscopic level. The rail vehicles and infrastructure elements (e.g. track segments and signals) are defined as atomic models, and the composites of rail infrastructure elements form rail components, which in turn can compose more complex rail components, and so recursively. The vehicle movement is modeled by dynamically coupling the vehicle model from one infrastructure element to another. Instead of direct transmission, V2V and V2I communications use conditional message propagation along the infrastructure elements. The sensor, signal and point models are coupled with the control units, so that signaling can be simulated based on the route request and track clearance. Additionally, a rail model generator is developed where the rail infrastructure definition is obtained through inferencing the geometric primitives in the CAD data. Future research will focus on automation of model calibration using measurement data from rail operations. Different data analysis methods and techniques will be investigated to estimate the model parameters.

REFERENCES

Alstrup, S., Gavoille, C., Kaplan, H., and Rauhe, T. 2004. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theory of Computing Systems*, 37:441–456.

Barros, F. J. 1995. Dynamic structure discrete event system specification: A new formalism for dynamic structure modeling and simulation. In *Proceedings of The 1995 Winter Simulation Conference*, pages 781–785.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. 2001. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 3rd edition.

Flynn, P. J. and Jain, A. K. 1991. CAD-Based Computer Vision: From CAD Models to Relational Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):114–132.

Hansen, I. A. and Pachl, J., editors 2008. *Railway Timetable & Traffic: Analysis-Modelling-Simulation*. Eurailpress.

Hu, X., Zeigler, B., and Mittal, S. 2005. Variable structure in DEVS component-based modeling and simulation. *Simulation*, 81(2):91–102.

Huang, Y., Seck, M. D., and Verbraeck, A. 2010a. LIBROS-II: Railway Modelling with DEVS. In Johansson, B., Jain, S., Montoya-Torres, J., Hugan, J., and Yücesan, E., editors, *Proceedings of The 2010 Winter Simulation Conference*, Baltimore, MD, USA. IEEE.

Huang, Y., Seck, M. D., and Verbraeck, A. 2010b. The Architecture and Components of LIBROS: Strengths, Limitations, and Plans. In Janssen, G. K., Ramaekers, K., and Caris, A., editors, *Proceedings of The 2010 European Simulation and Modelling Conference*, pages 80–87, Hasselt, Belgium. Eurosis-ETI.

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

Jacobs, P. H. M., Lang, N. A., and Verbraeck, A. 2002. D-SOL: A distributed java based discrete event simulation architecture. In *Proceedings of the 2002 Winter Simulation Conference*, pages 793–800. IEEE.

Lee, J.-K., Lim, Y.-H., and Chi, S.-D. 2004. Hierarchical modeling and simulation environment for intelligent transportation systems. *Simulation*, 80(2):61–76.

Mirabadi, A. and Yazdi, M. 2008. Automatic generation and verification of railway interlocking control tables using FSM and NuSMV. *International Journal for Engineering Modelling*, 21(1-4):57–63.

Ortúzar, J. and Willumsen, L. 2001. *Modelling Transport*. John Wiley & Sons, 3rd edition.

Pachl, J. 2002. *Railway Operation and Control*. VTD Rail Publishing.

Roanes-Lozano, E., Hernando, A., Alonso, J. A., and Laita, L. M. 2010. A logic approach to decision taking in a railway interlocking system using maple. *Mathematics and Computers in Simulation*, In Press.

Seck, M. D. and Verbraeck, A. 2009. DEVS in DSOL: Adding DEVS operational semantics to a generic event-scheduling simulation environment. In *Proceedings of the 2009 Summer Computer Simulation Conference*.

She, X., Sha, Y., Chen, Q., and Yang, J. 2007. The application of graphic theory on railway yard interlocking control system. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 883–887. IEEE.

Theeg, G. and Vlasenko, S., editors 2009. *Railway Signalling & Interlocking: International Compendium*. Eurailpress.

Wainer, G. 2006. ATLAS: A language to specify traffic models using Cell-DEVS. *Simulation Modelling Practice and Theory*, 14(3):313–337.

Zeigler, B. and Hammonds, P. 2007. *Modeling and Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange*. Elsevier/Academic Press.

Zeigler, B. P., Praehofer, H., and Kim, T. G. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Elsevier/Academic Press, 2nd edition.

BIOGRAPHY

YILIN HUANG is a Ph.D. Candidate at Delft University of Technology. Her research interests include dynamic data-driven simulation, automation of model calibration, and transportation systems simulation.

MAMADOU D. SECK is an Assistant Professor at Delft University of Technology. His research interests include M&S formalisms, dynamic data-driven simulation, human behavior simulation, and agent directed simulation.

ALEXANDER VERBRAECK is a Full Professor at Delft University of Technology. His current research focuses on development of object-oriented simulation building blocks, participative modeling, serious gaming using virtual reality, and agent technology in simulation.