Journal of Cellular Automata, Vol. X, pp. 1–53
Reprints available directly from the publisher
Photocopying permitted by license only

Defining a Traffic Modeling language Using Cellular Discrete-Event abstractions

GABRIEL WAINER*, ALEJANDRA DAVIDSON

Carleton University Department of Systems and Computer Engineering 1125 Colonel By Drive Ottawa, ON. K1S 5B6. CANADA.
Universidad de Buenos Aires Computer Science Department Planta Baja, Pabellón I - Ciudad Universitaria Buenos Aires (1428). ARGENTINA

Received: September 18, 2006. Accepted: October 31, 2006.

We present the definition of a specification language to outline sections of cities as discrete-event cell spaces. Streets are characterized by their traffic direction, number of lanes, etc. Specialized behavior is included to model traffic lights, trucks, traffic signs, railways, etc. The models are formally specified as DEVS and Cell-DEVS models, avoiding a high number of errors in the developed application. As the modelers can focus in the problem to solve, development times for the simulators can be highly reduced. We present the formal definitions for the language, and its translation into discrete-event models.

Keywords: traffic simulation, DEVS models, Cellular models, Cell-DEVS models, Modeling methodologies, Simulation support systems: environments.

1 INTRODUCTION

The use of computer simulation for urban traffic analysis and control is gaining acceptance due to the complexity of the phenomena involved, which cannot be studied analytically. Simulation models have been widely applied to solve these problems, allowing to improve traffic control, to measure the consequences of collisions, avoid pollution, congestion, etc. Nowadays, most existing techniques are based on *microscopic* models, which describe both the system entities and their interactions at a high level of detail (for example, a lane change could consider nearby cars, as well as detailed driver decisions).

*Corresponding author: Email:gwainer@sce.carleton.ca, ad3n@dc.uba.ar

Different modeling techniques have been used to create traffic simulations, including queuing networks (Schmidt 2000), Cellular Automata (Treiber et al. 2000, Rodríguez Zamora 2004), DEVS (Lee and Chi 2005), software agents (Balmer et al. 2004), object-oriented programming (Sadoun 2003) and other approaches, including Game Theory (Chen and Ben-Akiva 1998), Petri Nets (Tolba et al. 2005), up to fluid or electrical flow models. Following these ideas, we created a microsimulation language to precisely describe the behavior of traffic in closed sections. The goal is to let a modeler to analyze behavior in cities with complex urban design, or in closed traffic conditions (parking, roads in shopping malls, amusement parks, etc).

Cellular Automata (CA) is a popular technique widely used for defining these kinds of models (Maniezzo 2004, Nagel et al. 2000, Nagel 2002, Esser and Schreckenberg 1997, Marinosson 2002, Rickert et al. 1996). CA define a grid of cells using discrete variables for time, space and system states (Chopard and Droz 1998, Wolfram 2002). Cells are updated synchronously and in parallel for every cell in the space according with a local rule using a finite set of nearby cells (the neighborhood). Cellular models represent a quite intuitive way of analyzing the traffic flow in detail, and they enable good visualization of the results. Nonetheless, CA are synchronous, a fact that poses precision constraints and extra compute time. The Cell-DEVS formalism (Wainer and Giambiasi 2002) was proposed to solve these problems by defining cell spaces as DEVS (Discrete Events systems Specifications) models (Zeigler et al. 2000). Using Cell-DEVS, a cell space is described as a discrete event model in which explicit delays can be used to accurately model the cell timing properties.

ATLAS (Advanced Traffic LAnguage Specifications) allows one to represent city sections, and its constructions are translated into discrete event models represented using DEVS and Cell-DEVS. DEVS and Cell-DEVS are discrete event formalisms, providing precision and speedups in the simulations. Likewise, these formal approaches allowed us to guarantee the correctness and completeness of the simulation models, as DEVS and Cell-DEVS simulators have been formally proved to ensure correct execution. In this way, errors in the simulation can be detected by analyzing the specification, without considering the underlying software system. The streets can be defined, analyzing the traffic direction, number of lanes, traffic lights, etc. The language is focused in the description of precise microsimulations, allowing analyzing different behavior according with the shape of the city and different traffic characteristics. In Wainer (2006) we presented the general ideas about ATLAS, focusing on the language definition, the creation of the ATLAS Traffic Simulation Compiler, and showing the simulation results obtaining in applying the tool to a real city section, allowing us to validate the proper behavior of the simulation tool. Here, we focus on ATLAS constructions, and present a detailed mapping of the language constructions into discrete event and cellular models.

2 THE DEVS FORMALISM

In this section we introduce the formal techniques employed to define the ATLAS modeling language. We will present classic DEVS models, and then we introduce Cell-DEVS.

2.1 Classic DEVS

A real system modeled using DEVS (Zeigler et al. 2000), can be described as composed of several submodels. Each of them can be behavioral (atomic) or structural (coupled). A DEVS atomic model can be formally described as:

$$M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

$\boldsymbol{X}$ is the input events set;
S is the state set;
$\boldsymbol{Y}$ is the output events set;
$\boldsymbol{\delta}_{int}$: S $\rightarrow$ S, is the internal transition function;
$\boldsymbol{\delta}_{ext}$: Q x X $\rightarrow$ S, is the external transition function; where Q = { (s, e) / s $\in$ S, and e $\in$ [0, D(s)]};
λ: S $\rightarrow$ Y, is the output function; and
D: S $\rightarrow \boldsymbol{R}_0^+ \cup \infty$, is the elapsed time function.

Each state in a model has a given lifetime, defined by the elapsed time function. Once the lifetime of a given state is consumed, the output and the internal transition functions are activated. At any moment, a model can receive input external events. When an external event arrives, the external transition function is activated, which computes a new state for the model using the present state, the input values, and the elapsed time for the model.

An atomic model can be integrated with other DEVS models to build a structural model. These models are called coupled, and are integrated by base models, that is, atomic or other coupled ones. DEVS coupled models are formally defined as:

$$CM =< X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, \text{select} >$$

$\boldsymbol{X}$ is the set of input events;
$\boldsymbol{Y}$ is the set of output events;
D $\in N$, D $< \infty$ is an index for the components of the coupled model, and $\forall$ i $\in$ D, $\mathbf{M}_i$ is a basic DEVS model,
$\mathbf{I}_i$ is the set of influencees of model i, and $\forall$ j $\in \mathrm{I}_i$, and
$\mathbf{Z}_{ij}$: $\mathrm{Y}_i \rightarrow \mathrm{X}_j$ is the i to j translation function.

Finally, **select** is the tie-breaking selector.

Each coupled model consists of a set of basic models (atomic or coupled) interconnected through the model's interfaces. Each component is identified by an index number. The influencees of each model define other models

where output values must be sent. The translation function uses an index of influencees, created for each model (I_i). The function defines which outputs of model M_i are connected to inputs in model M_j. When two submodels have simultaneous events, the *select* function defines which of them should be activated first.

2.2 Cell-DEVS

Cell-DEVS allows defining complex cellular models that can be integrated with other DEVS. Here, each cell of a space is defined as an atomic DEVS. Transport and inertial delays allow defining timing behavior of each cell in an explicit and simple fashion. A *transport* delay allows us to model a variable response time for each cell. Instead, *inertial* delays are preemptive: a scheduled event is executed only if the delay is consumed. Cell-DEVS atomic models are formally specified as:

$$\text{TDC} =< X, Y, I, \text{S}, \theta, \text{N}, \text{d}, \delta_{\text{int}}, \delta_{\text{ext}}, \tau, \lambda, \text{D} >$$

where for # $\mathbf{T} < \infty \wedge \text{T} \in \{\boldsymbol{N}, \boldsymbol{Z}, \Re, \{0, 1\}\} \cup \{\phi\}$

$\boldsymbol{X} \subseteq$ T is the set of external input events;
$\boldsymbol{Y} \subseteq$ T is the set of external output events;

$\boldsymbol{I} = < \eta, \mu, \text{P}^x, \text{P}^y >$ represents the definition of the model's modular interface. Here, $\eta \in \boldsymbol{N}$ is the neighborhood's size, $\mu \in \boldsymbol{N}$ is the number of other input/output ports, and $\forall j \in [1,\eta]$, $i \in \{\text{X}, \text{Y}\}$, P_j^i is a definition of a port (input/output), $\text{P}_j^i = \{(\text{N}_j^i, \text{T}_j^i) / \forall\ j \in [1, \eta + \mu], \text{N}_j^i \in [i_1, i_{\eta+\mu}]$ (port name), $\text{T}_j^i \in \text{I}_i$ (port type)$\}$, $\text{I}_i = \{\ x / x \in \boldsymbol{X}$ if X $\}$ or $\text{I}_i = \{\ x / x \in \boldsymbol{Y}$ if i = Y $\}$;

S $\subseteq$ T is the set of sequential states for the cell;
θ is the definition of the cell's state, $\theta = \{$ (s, phase, σqueue, σ)/s $\in$ S is the status value for the cell, phase $\in \{$active, passive$\}$, σqueue = $\{$ $((\text{v}_1, \sigma_1),...,(\text{v}_\text{m},\sigma_\text{m}))/\text{m} \in \boldsymbol{N} \wedge \text{m} < \infty) \wedge \forall(\text{i} \in \boldsymbol{N}, \text{i} \in [1,\text{m}])$, $\text{v}_\text{i} \in$ S $\wedge \sigma_\text{i} \in \boldsymbol{R}_0^+ \cup \infty\}$; and $\sigma \in \boldsymbol{R}_0^+ \cup \infty\}$;
N $\in \text{S}^{\eta+\mu}$ is the set of input events;
d $\in \boldsymbol{R}_0{}^+$, d $< \infty$ is the transport delay for the cell;
$\boldsymbol{\delta}_{\text{int}}$: $\theta \rightarrow \theta$ is the internal transition function;
$\boldsymbol{\delta}_{\text{ext}}$: Qx$X \rightarrow \theta$is the external transition function, where Q = $\{$ (s, e) / s $\in \theta$ x N x d; e $\in$ [0, D(s)]$\}$;
$\boldsymbol{\tau}$: N $\rightarrow$ S is the local computation function;
$\boldsymbol{\lambda}$: S $\rightarrow Y$ is the output function; and
D: θ x N x d $\rightarrow \boldsymbol{R}_0^+ \cup \infty$, is the state's duration function.

The state for each cell is defined as a set composed by its present value and phase. A queue is also used to keep the next events values and their scheduled simulated time. The **N** set is used to represent the input values

for the cell and it is composed by an $\eta + \mu$-tuple (s_1,..., $s_{\eta+\mu}$), where $s_i \in S$. This set is used to record the present values used to compute the future value for the cell through the local computation function τ. The duration function **D** manages the cell's lifetime. Here, **D**(s, phase, σqueue, σ, N, d) = t represents the time during which a cell keeps the present status if no external events are detected. The transition and output functions ($\boldsymbol{\lambda}$, $\boldsymbol{\delta}_{int}$, $\boldsymbol{\delta}_{ext}$) are used to activate the model's local and delay functions. Each cell can have an associated delay (**d**), allowing to delay the execution of the internal transition function.

A Cell-DEVS coupled model is defined by:

$$GCC = < X_{list}, Y_{list}, I, X, Y, n, \{t_1, ..., t_n\}, N, C, B, Z >$$

Here, $\mathbf{Y}_{list}$ is an output coupling list, $\mathbf{X}_{list}$ is an input coupling list and $\boldsymbol{I}$ represents the interface of the model. $\boldsymbol{X}$ are the external input events and $\boldsymbol{Y}$ the external outputs. The **n** value defines the dimension of the cell space, $\{\mathbf{t}_1,...,\mathbf{t}_n\}$ is the number of cells in each dimension, and **N** is the neighborhood set. **C** is the cell space, **B** is the set of border cells and **Z** the translation function. The cell space defined by this specification is a coupled model composed of an array of atomic cells. Each of them is connected to the cells defined by the neighborhood. As the cell space is finite, the borders should have a different behavior than the remaining cells. Otherwise, the space is wrapped, meaning that cells in a border are connected with those in the opposite one. Finally, the Z function allows one to define the internal and external coupling of cells in the model. This function translates the outputs of m-eth output port in cell *Cij* into values for the *m*-eth input port of cell *Ckl*. The input/output coupling lists can be used to transfer data with other models. This informally presented in Figure 1.

3 THE ATLAS MODELING LANGUAGE

ATLAS allows the definition of a city section with detail. The language constructions allow to build the structure of a model that represents the topology of a city section. The basic structure is the *segment,* (a portion of a street) connected by *crossings*. These constructions define a static view of the model, representing a standard city map. According to the kind of construction used, an implicit dynamic behavior is associated. The dynamic will depend on how the traffic is injected. This is done by an experimental framework used to generate traffic and to analyze traffic metrics. Different decorations can be added, including railways, traffic signs, parking sections, traffic lights, etc.

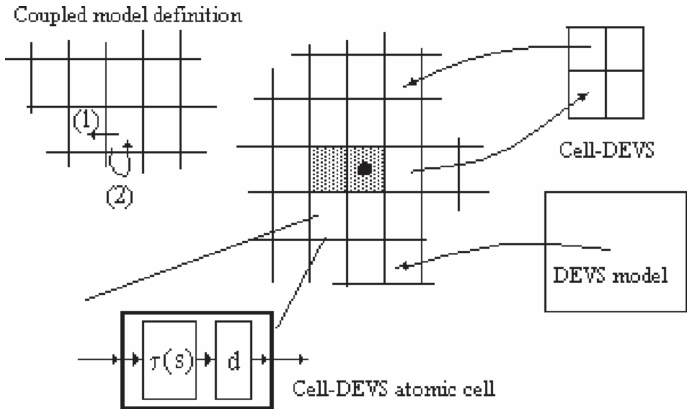

FIGURE 1
Informal definition of a Cell-DEVS model.

Figure 2 shows a part of a city section in detail, labeling the segments and crossings. We will use this example in the following sections to show how a construction is translated into DEVS models.

The different components in Figure 2 can be defined in ATLAS as follows:

Segments = { rA, rB, rC, rD1, rD2, rE, rF, rG1, rG2, rH1, rH2, rI1, rI2 }, where:

rA=[(0,0), (0,130), 1, 40, 0, 1]
rB=[(0,130), (0,200), 1, 40, 0, 1]
rC=[(0,200), (0,300), 1,40, 0, 1]
rD1=[(0,130),(100,200),2,60,0, 1]
rF=[(0,300), (100,300), 1, 40, 0, 1]
rG1=[(100,0),(100,200),4,60,0, 1]
rG2=[(100,0),(100,200),4,60,0, 0]
rH1=[(100,200),(100,300),2,60,0,1]

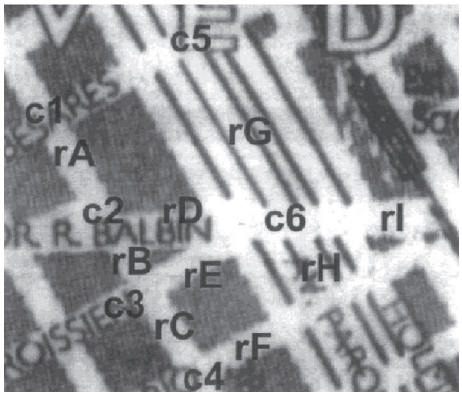

FIGURE 2
Specification of the segments and crossings in the city section.

rD2=[(0,130),(100,200),2, 60, 0, 0]
rE=[(0,200), (100,200), 1, 40, 0, 0]
rH2=[(100,200),(100,300),2,60,0,0]
rI1=[(100,200),(200,280),2,60,0,1]
rI2=[(100,200),(200,280), 2, 60, 0, 0]

Crossings = { c1, c2, c3, c4, c5, c6 }, where c1 = ((0,0); 20), c2 = ((0,130); 30), c3 =((0,200); 20), c4 = ((0,300); 30), c5 = ((100,0); 30), c6 = ((100,200); 30)
Railnets = { (Station1, Rail1) }, where Rail1 = { (rI1, 90, 1), (rI2, 10, 1), ... }

TLCrossings = { c2, c5, c6 }
TrafficSigns = { (rA, school,20), (rC, stop, 95), (rE, PedXing, 95) }
Pothole = { (rA, 0, 10) }
TruckSegments = { rD1, rD2, rG1, rG2, rH1, rH2, rI1, rI2 }
TruckXings = {c2, c6, ... }
Parking = {(rG1,1)}
InputSegments = { rH1, rI1 }
OutputSegments = { rH2, rI2 }

The specification considers the plane starting in the crossing *c1*. The segment *rA* is a one-way/one-lane segment. Its maximum speed is 40 km /h, and it is a straight line. Segments *rB* and *rC* are continuations of this segment. Segment *rD* is a two-way segment (therefore, *rD1* and *rD2* are defined). The maximum speed allowed is 60 km /h. Segment *rI* is the continuation of this segment. Finally, *rG* is a two-way segment with 4 lanes in each way. The maximum speed is also 60 km /h. As we can see, trucks are allowed in segments *rD*, *rI*, *rG* and *rH* (both ways). The crossing specifications show the position and maximum speed allowed for each of them. The railway construction shows that a crossing level intersects the segment *rI*. Segment *rI1* is cut 10 m from the crossing, and *rI2*, 90 m from the next crossing. Finally, we show the definition of a pothole, several traffic signs, and a parking lane.

3.1 Segments

Each street in the city is represented as a sequence of **segments** that represent the section between two corners. Every lane in a given segment has the same direction (one way segments) and a maximum speed (two-way streets are built using one segment on each direction, informally presented in Figure 3). Segments are specified by:

$$Segments = \{(p1, p2, n, max, a, dir)/p1, p2 \in City \wedge n, max \in \Re \wedge a, dir \in \{0, 1\}\}.$$

Here, **p1** and **p2** represent the boundaries of each segment, which belong to City = { (x,y) / x, y $\in \Re$ }. Then, **n** defines the number of lanes in the segment, and **max** is used to define the maximum speed allowed in the

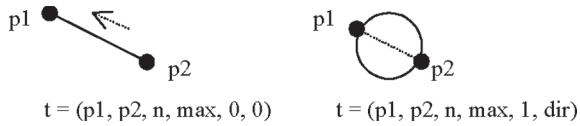

FIGURE 3
Specification of a segment.

segment. The **a** parameter defines the shape of the segment. Here, $a = 0$ is used when the segment is a straight, and $a = 1$ when the segment is curve. Finally, **dir** represents the vehicle direction (*dir* = 1 means that vehicles move towards *p2*, otherwise they go to *p1*).

Each segment is defined as a bidimensional Cell-DEVS. The behavior defined for each cell is different according to the number of lanes, because in each case we must define different border conditions. The most simple mapping allows us to define a one lane segment *s* = (*p1, p2, 1, a, dir, max*) as a one-dimensional Cell-DEVS with transport delays, showed in Figure 4.

Each cell is defined as:

$$S1 =< I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

I = $<P^x, P^y>$, P^x = { (X_1, binary), (X_2, binary), (X_3, binary) }; P^y = { (Y_1, binary), (Y_2, binary), (Y_3, binary) }.
$\boldsymbol{X} = \boldsymbol{Y} = \{0, 1\}$; **N** = { (0,-1), (0,0), (0,1) };

$$\boldsymbol{S} = \begin{cases} 1 \text{ if there is a vehicle in the cell;} \\ 0 \qquad\qquad \text{otherwise.} \end{cases}$$

delay = transport; **d** = speed(max);
λ, δ_{int} and δ_{ext} behave as defined in the Cell-DEVS formalism with transport delays.
τ: S x N $\rightarrow$ S is defined as follows:

τ(N)	N
1	(0,-1) = 1 and (0,0) = 0
0	(0,0) = 1 and (0,1) = 0
(0,0)	TRUE /*Otherwise: state unchanged */

Here, the transport delays are used to model the time used by a vehicle to leave a cell and get into the next one. It depends on the present speed of the vehicle, and uses function related with the car speed and the maximum

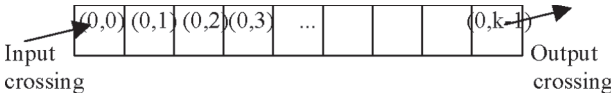

FIGURE 4
One lane segment.

speed allowed in the segment. The local computation defines the movement of a vehicle. The first rule represents a vehicle arriving to an empty cell from the previous one. The second rule represents the car abandoning the present cell towards the front. Otherwise, the cell preserves the present value.

The coupled model corresponding with the segment is defined as:

$$CS1(k, max) =< X_{list}, Y_{list}, I, X, Y, n, \{t_1, ..., t_n\}, \eta, N, C, B, Z >$$

$\mathbf{Y}_{\text{list}} = \mathbf{X}_{\text{list}} = \{ (0,0), (0,k\text{-}1) \}$
$\mathbf{I} = <P^x, P^y >$, where $P^x = \{<X_{\eta+1}(0,0), \text{binary}>, <X_{\eta+1}(0,k\text{-}1), \text{binary}>\}$, $P^y = \{<Y_{\eta+1}(0,0), \text{binary}>, <Y_{\eta+1}(0,k\text{-}1), \text{binary}>\}$.

These ports will be noted follows (with i=0):

Port	Name
$X_{\eta+1}$(i,0)	x-c-vehicle
$X_{\eta+1}$(i,k-1)	x-c-space
$Y_{\eta+1}$(i,0)	y-c-space
$Y_{\eta+1}$(i,k-1)	y-c-vehicle

$\boldsymbol{X} = \boldsymbol{Y} = \{ 0, 1 \}$; $\mathbf{n} = 1$; $\mathbf{t}_1 = k$; $\eta = 3$,
$\mathbf{N} = \{(0, -1), (0,0), (0,1)\}$; $B = \{(0,0), (0,k\text{-}1) \}$; and
$\mathbf{Z}$ is built using the basic definitions of the Cell-DEVS formalism

The parameters k and *max* are used to build this coupled model. The maximum speed is used as a parameter for the random number generation, and the k parameter is obtained computing the segment length.

To do this, **k = segment_len(s: Segments)** computes the distance between boundaries, and divides it by the cell size. If s= *((x1,y1), (x2,y2), n, a, dir, max*) is a segment, then,

If (a = 0) **then** $\ell = |\sqrt{|x1 - x2|^2 + |y1 - y2|^2}|$; /* Line length in a Cartesian plane */
else $\ell = (2\pi(|\sqrt{|x1 - x2|^2 + |y1 - y2|^2}|)/2))/2$; /* Circle perimeter */
return $k = \lceil \ell / cell\_size \rceil$

The length of each cell in km is defined by the function *cell_size*. We have the length of a car plus the standard distance between them defined in (Wagner et al. 1997, Nagel et al. 1998). This distance is 7.5 m for each cell. For instance, if we consider the specification of *rA*, we have:

$$k = \left\lceil \sqrt{|x1 - x2|^2 + |y1 - y2|^2} \Big/ cell\_size \right\rceil = \left\lceil \sqrt{0 + 130^2} \Big/ 7.5 \right\rceil = 18$$

The cells (0,0) and (0, k-1) compose the external interface of the model, because they must interchange information with the crossings, as depicted in Figure 5. Hence, the behavior of the cell (0,0) is redefined:

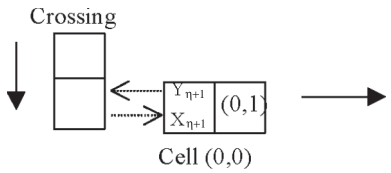

FIGURE 5
Neighborhood/coupling for the cell (0,0).

$\eta = 2$; **N** = { (0,0), (0,1) }

The function τ for this cell is the same than that defined earlier, with the first rule changed by:

τ (N)	N
1	portvalue(x-c-vehicle) = 1 and (0,0) = 0

The function *portvalue* returns the present value of a given port. Whenever the port *x-c-vehicle* is 1, there is a vehicle wanting to leave the crossing. Therefore, this rule represents the departure of a car from the crossing to the segment. The cell (0,k-1) is coupled with the first cell of the crossing, as showed in Figure 6. Then, its behavior and neighborhood were redefined

$\eta = 2$; **N** = { (0,-1), (0,0) }

The function τ have redefined the second rule, now specified by:

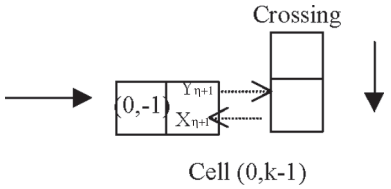

FIGURE 6
Neighborhood/coupling for the cell (0,k-1).

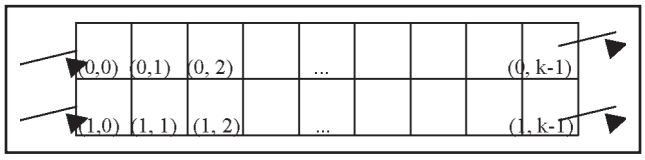

FIGURE 7
A Two lane segment.

τ (N)	N
0	(0,0) = 1 and portvalue(x-c-space) = 0 send(1, y-c-vehicle)

Delay = inertial.

The function *send* defines the new state of the selected external ports. The port *x-c-space* tells if there is space in the input cell of the crossing. Whenever this cell and the previous one in the crossing are empty, a car can advance. In this way, we represent that cars in the crossing have higher priority than those trying cross. We use an inertial delay is used, to represent that a car can get into the crossing only if there is a free space for a while. If a fast car arrives to the input cell before the delay, the crossing car must remain waiting.

When streets with **two lanes** are defined, we translate the segment *s* = (*p1, p2, 2, a, dir, max*) into a two-dimensional Cell-DEVS with the following structure:

Each row of this space acts as a border of the model. Vehicles in the first row can change to the right, and those in the second row can move to the left. Therefore, each row must be specified separately. The atomic cells in the first row will be defined using the one-lane cells as a base. The main changes include the interface (I), neighborhood (N) and behavior (τ), now defined as:

$\mathbf{I} = < \eta, P^x, P^y >$, where $\eta = 6$; $\mathbf{N} = \{$ (0,0), (0,1), (-1,0), (-1,1), (0,-1), (-1,-1) $\}$;

The function τ for these cells is defined as in the one lane model, adding the following rules to model the lane changes:

τ (N)	N
1	(0,0) = 0 and (0,-1) = 0 and (-1,-1) = 1 and (-1,0) = 1
0	(0,0) = 1 and (-1,1) = 0 and (-1,0) = 0

These rules of lane change consider that a vehicle tries first to move straight, and it has priority to use the position in front of it. The first rule here represents a vehicle arriving in diagonal. To define the priority access, the diagonal movement checks if there is no car waiting to us the cell in diagonal. If that is not the case, it can advance.

The main difference in the definition for the lane 1 is in the neighborhood definition and the local computing function, whose definition is symmetric to the previous:

N = { (0,0), (0,1), (1,0), (1,1), (0,-1), (1,-1) };

The function τ for these cells extends those of the one lane model as follows:

τ (N)	N
1	(0,0) = 0 and (0,-1) = 0 and (1,-1) = 1 and (1,0) = 1
0	((0,0) = 1 and (1,1) = 0 and (1,0) = 0)

These rules also define the lane changes, considering priority movements for the cars advancing in a straight line.

The coupled model corresponding to the segment is defined by:

$$TC2(k, max) =< X_{list}, Y_{list}, I, X, Y, n, \{t_1, ..., t_n\}, \eta, N, C, B, Z >$$

$\mathbf{Y}_{\text{list}} = \mathbf{X}_{\text{list}} = \{ (0,0), (1,0), (0,k\text{-}1), (1,k\text{-}1) \}$
$\mathbf{I} = <P^x, P^y >$, where $P^x = \{<X_{\eta+1}(0,0), \text{binary}>, <X_{\eta+1}(1,0), \text{binary}>, <X_{\eta+1}(0,k\text{-}1), \text{binary}>, <X_{\eta+1}(1,k\text{-}1), \text{binary}>\}$

$$P^y = \{< Y_{\eta+1}(0, 0), \text{binary} >, < Y_{\eta+1}(1, 0), \text{binary} >, < Y_{\eta+1}(0, k\text{-}1), \text{binary} >, < Y_{\eta+1}(1, k\text{-}1), \text{binary} >\}$$

These ports will be named as in the previous section, but considering that $0 \leq i \leq 1$.

$\boldsymbol{X} = \boldsymbol{Y} = \{ 0, 1 \}$; $\mathbf{n} = 2$; $\mathbf{t}_1 = 2$; $\mathbf{t}_2 = k$; $\eta = 6$; **N** { (0,0), (0,1), (1,0), (1,1), (0,-1), (1,-1) }; **B** = { (0, k-1), (1, k-1), (0,0), (1,0) }; and **Z** is built using the definition given by the definition given in the Cell-DEVS formalism.

The interface of this model is composed by the cells of the first and last columns, used to interchange information with each of the crossings. The external ports and the rules for the crossings are extensions of those defined for the one lane model. The output cells of the crossing will behave according to the segment to which they are coupled. The output cells allowing cars or trucks are defined following:

τ(N)	N
1	(0,0) = 0 and (0,-1) = 1 and [x-s-room = 1 or IsTruck(x-s-room) or (x-s-room = 0 and random < p_{out})] send(0,y-s-vehicle) /* Arrival of a car that will stay in the crossing */
(0,-1)	(0,0) = 0 and Truck(0,-1) and [x-s-room = 1 or

	IsTruck(x-s-room) or (x-s-room = 0 and random $<$ p_{out})] send(0,y-s-vehicle)
	/* Arrival of a truck staying in the crossing */
0	(0,0) = 0 and (0,-1) = 1 and x-s-room = 0 and random $\geq$ p_{out} send(1,y-s-vehicle)
	/* Arrival of a car that will leave the crossing */
0	(0,0) = 0 and Truck(0,-1) and x-s-room = 0 and random $\geq$ p_{out} send((0,-1), y-s-vehicle)
	/* Arrival of a truck that will leave crossing */
0	(0,0) = 1 and (0,1) = 0 send(0,y-s-vehicle)
0	Truck(0,0) and (0,1) = 0 send(0,y-s-vehicle)
(0,0)	TRUE send(0,y-s-vehicle)
	/* Otherwise, the present state is kept */

Here, p_{out} is a constant that represents the probability of vehicle leaving the crossing. A vehicle can leave the crossing if there is enough space in the segment and a random value is higher than p_{out}. A truck passing through cells connected to segments not allowing trucks will keep advancing. The behavior for these cells is defined by:

τ(N)	N
1	(0,0) = 0 and (0,-1) = 1 and (x-s-room = 1 or (x-s-room = 0 and random $<$ p_{out})) send(0,y-s-vehicle)
	/* Arrival of a car that will stay in the crossing */
(0,-1)	(0,0) = 0 and Truck(0,-1) send(0,y-s-vehicle)
	/* Arrival of a truck that will stay in the crossing */
0	(0,0) = 0 and (0,-1) = 1 and x-s-room = 0 and random $\geq$ p_{out} send(1,y-s-vehicle)
	/* Arrival of a car that will leave the crossing */
0	(0,0) = 1 and (0,1) = 0 send(0,y-s-vehicle)
0	Truck(0,0) and (0,1) = 0 send(0,y-s-vehicle)
(0,0)	TRUE send(0,y-s-vehicle)
	/* Otherwise, the present state is kept */

These rules represent that, when there is space in the segment and a random value is greater than p_{out}, the cars can leave the crossing. When a truck arrives to these cells, it just advances to the next cell if it is empty. A truck cannot it leave the crossing through these cells.

The local computing function of the input cells is defined by:

τ (N)	N
1	(0,0) = 0 and (0,-1) = 1 send(1, y-s-room)
1	(0,0)=0 and x-s-vehicle = 1 and (0,-1)=0 send(1, y-s-room)
(0,-1)	(0,0) = 0 and Truck(0,-1) ; send(1, y-s-room)
x-s-vehicle	(0,0) = 0 and IsTruck(x-s-vehicle) and (0,-1)=0 send(1, y-s-room)
0	(0,0) = 1 and (0,1) = 0 and (0,-1) = 0 send(0, y-s-room)
	/* No vehicle with priority is in the crossing */
0	Truck(0,0) and (0,1) = 0 and (0,-1) = 0 send(0, y-s-room)
	/* No vehicle with priority is in the crossing */
0	(0,0) = 1 and (0,1) = 0 and ((0,-1) = 1 or Truck(0,-1)) send(1, y-s-room)
	/* No vehicle with priority is in the crossing */
0	Truck(0,0) and (0,1) = 0 and ((0,-1) = 1 or Truck(0,-1)) send(1, y-s-room)
	/* No vehicle with priority is in the crossing */
(0,0)	TRUE /* Otherwise, keep the previous state */

The ports should be updated explicitly, because the new state for each of the cells is not transmitted. In the third and fourth rules, the new state represents the presence of a truck. Then, if the coupled segment is binary, it will not accept trucks. Therefore, a value of 1, representing that the cell is busy is sent to the segment. The segment will wait the cell to be empty, which will be done by sending a 0 value through the output port. In this way, if there is a vehicle with priority in the crossing, the cell sends a 1. In this way, a new arriving vehicle knows that it must stop before the crossing.

The constructions here presented also include rules to translate segments with three to five lanes (or more). They are built as an extension to the behavior presented in this section (Davidson and Wainer, 1999).

3.2 Segments with trucks

The constructions presented in the previous section were extended to allow the representation of trucks. Two different constructions are available according to the kind of traffic allowed in the street. The standard models are used in streets where heavy traffic is not allowed. Otherwise, the models presented in these sections are applied. A segment including trucks is defined by:

$$TruckSegments = \{(p1, p2, n, a, dir, max)/p1, p2 \in City \wedge n, max \in N \wedge \mathrm{a}, dir \in \{0, 1\}\}$$

With the parameters defined as for standard segments. Each TruckSegment will be translated into a Cell-DEVS with transport delays. The TruckSegment cell atomic model is defined as:

$$\mathrm{TS_{ij}} =< \mathrm{I}, X, \mathrm{S}, Y, \mathrm{N}, \delta_{\mathrm{int}}, \delta_{\mathrm{ext}}, \mathrm{delay}, \mathrm{d}, \tau, \lambda, \mathrm{D} >$$

$X = Y = \mathrm{N}$; **delay** = transport; **d** = truck_sp(max);
S:

$$\mathrm{s} = \begin{cases} 1 & \text{if there is a car in the cell;} \\ 0 & \text{if the cell is empty; and} \\ \mathrm{k} = \mathrm{rmod}10 \wedge \mathrm{r} \in [2, 5] & \text{if here is a truck.} \end{cases}$$

D, λ, δ_{int} and δ_{ext} are defined by the Cell-DEVS formalism to achieve the transport delay behavior.

τ will be discussed later;

N = {(3,0), (2,-5), (2,-4), (2,-3), (2,-2), (2,-1), (2,0), (2,1), (1,-5), (1,-4), (1,-3), (1,-2), (1,-1), (1,0), (1,1), (1,2), (0,-5), (0,-4), (0,-3), (0,-2), (0,-1), (0,0), (0,1), (0,2), (-1,-5), (-1,-4), (-1,-3), (-1,-2), (-1,-1), (-1,0), (-1,1), (-1,2), (-2,-5), (-2,-4), (-2,-3), (-2,-2), (-2,-1), (-2,0), (-2,1) }

A truck is defined by $\{ \mathrm{s} / \mathrm{s} \in \mathrm{N} \wedge \mathrm{s} > 1 \wedge \mathrm{r} = \mathrm{s} \bmod 10 \wedge \mathrm{r} \in [2, 5]\}$. Each *s* represents an identifier for a truck with length *r*. The length is used to define the space needed to change between lanes. Each truck has a unique identifier (the *s* parameter) that can be used to recognize it. *Truck_sp()* is a function returning a delay representing the current speed. The value returned depends on the maximum speed allowed in the street. The cell's neighborhood is depicted in Figure 8.

The cell's behavior defined by the τ function is divided in two groups. The first one is related with the truck's movement, and the second is related with car's advance. When a truck moves, several cells should change together. The first cell is in charge to choose the next movement, checking

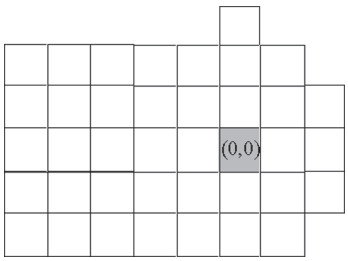

FIGURE 8
Cell's Neighborhood.

if there is space to move all the truck. The remaining cells will follow the first one. There are three valid movements. The first one movement consists in advancing one cell in the same lane. If there is no space, it tries to move to the right (reflecting that the trucks should use the right lanes). Otherwise, it tries to go to the left, as showed in Figure 9 (Davidson and Wainer, 2000b).

Any vehicle advancing straight will have a higher priority than those coming from the other lanes. Therefore, to make a straight movement we only need to check if there is a place in the previous cell. When a truck tries to move to the right, the head verifies if the right lanes are free, that is, $\forall\ k \in [0, \text{remainder}((i,j),10)] \Rightarrow (i\text{-}1,j\text{-}k) = 0$. These trucks will have a higher priority than the cars trying to make the same movements. Instead, when moving to the left, the trucks should check that no other vehicles are trying to occupy those cells. Two lanes must be checked because other trucks or cars in the second lane trying to move have higher priority to do it. To avoid collisions with trucks, car movements defined in the section 5.1 were extended. When a car tries to move to the left, it checks that enough room is available (*(1,0) = 0 and (1,1) = 0*) and no trucks are trying to move to the same cell. A car arrives to the origin cell from the right only if the car is not able to move forward (*(-1,0) != 0*), there is enough space to move (*(0,-1) = 0 and (0,0) = 0*), and a truck is not trying to get to the same position. When a car moves to the right, we check that there is enough space. In addition, no other vehicle must be trying to reach same position. Finally, a car can arrive from the left. We must check there is no space to advance, the car cannot stay in its own lane, and it cannot move to the left. Besides, it checks if there is a truck moving to the origin cell.

As the segments contain trucks or cars, a policy ruling the movement of both must be defined. Any vehicle advancing straight will have a higher priority than the ones coming from the other lanes. Therefore, to make a straight movement we only need to check if there is a place in the previous cell. A truck trying to move to the right must check if there is enough room to do it. These trucks will have a higher priority than the cars trying to make the same movements. Instead, when moving to the left, the trucks should check that no vehicles are trying to occupy those

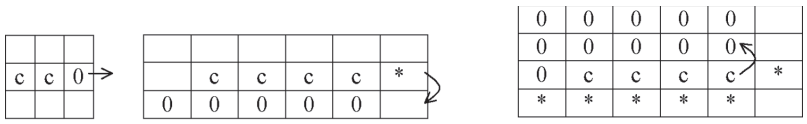

FIGURE 9
Truck movements. (a) Straight; (b) Change to the right lane; (c) Change to the left.

cells. The neighborhood defined earlier allows to check these movements by using the following rules:

τ(N)	N
(0,-1)	(0,0) = 0 and Truck(0,-1) and IsHead(0,-1)
(0,-1)	(0,0) = 0 and Truck(0,-1) and (0,1) = (0,-1)

Here, we represent the arrival of a truck to the origin cell from the cell in the back. *Truck(i,j)* is a macro used to verify that the position contains a part of a truck, that is, Truck(i,j) $\equiv$ remainder((i,j),10) $\in$ [2,5]. *IsHead(i,j)* is a macro that checks if the position (i,j) is the head of the truck. To do so, it verifies that the previous positions do not include part of the same truck. The following rules represent a truck leaving the cell. In the first case, the truck leaves the cell using a straight movement. In the second one, the truck moves to the right.

τ (N)	N
0	Truck(0,0) and (0,1) = 0 and IsHead(0,0)
0	Truck(0,0) and (0,1) = 0 and (0,2) = (0,0)

In the next case, the head and the rest of the truck are also distinguished. Only the head checks if there is enough space to move (*Free_Right_Truck(0,0)*), and the rest just follow it.

τ(N)	N
0	Truck(0,0) and Free_Right_Truck(0,0) and IsHead(0,0)
0	Truck(0,0) and (-1,1) = (0,0)

Here, *Free_Right_Truck(i,j)* verifies if the right lane of the cell (i,j) is free, that is, Free_Right_Truck(i,j) $\equiv$ $\forall$ k $\in$ [0, remainder((i,j),10)] $\Rightarrow$ (i-1,j-k) = 0. The following rule represents a truck arriving to the origin cell from the left. The head should check that the chosen movement is to the right (*(1,1) != 0*), and there is space to do it (*Free_Right_Truck(1,0)*).

τ(N)	N
(1,0)	(0,0) = 0 and Truck(1,0) and (1,1) != 0 and IsHead(1,0) and Free_Right_Truck(1,0)
(1,0)	(0,0) = 0 and Truck(1,0) and (0,1) = (1,0)

A truck leaving the origin cell and moving to the left is represented following. The truck's head checks if there is enough room to move to the

left (*2LeftLanesFree(0,0)*). Two lanes are necessary because other trucks or cars trying to move have higher priority to do it.

τ(N)	N
0	Truck(0,0) and 2LeftLanesFree(0,0) and IsHead(0,0)
0	Truck(0,0) and (1,1) = (0,0)

Following, we represent a truck arriving to the origin cell from the right. If it is the truck's head, the movement should be checked by seeing that the truck cannot advance (*(1,1) != 0*) or turn right (*!(Free_Right_Truck(-1,0))*). It also should have enough room to do the movement (*2LeftLanesFree(-1,0)*).

τ(N)	N
(-1,0)	(0,0)=0 and Truck(-1,0) and (-1,1)!=0 and IsHead(-1,0) and !(Free_Right_Truck(-1,0)) and 2LeftLanesFree(-1,0)
(-1,0)	(0,0) = 0 and Truck(-1,0) and (0,1) = (-1,0)

The car movements also must be defined. The first rule following represents the arrival of a car to the cell. The second one represents a car abandoning a cell, and moving forward. In these cases, the delay function is d = speed(max), where *speed* returns a delay proportional to the car speed depending on the speed limit.

τ(N)	N
1	(0,0) = 0 and (0,-1) = 1
0	(0,0) = 1 and (0,1) = 0

The next rule represents a car leaving a cell to the left. It checks that enough room is available (*(1,0) = 0 and (1,1) = 0*) and no trucks with higher priority are trying to move to the same cell (*!(Truck(2,1))*).

τ(N)	N
0	(0,0) = 1 and (1,0) = 0 and (1,1) = 0 and !(Truck(2,1))

The following rule represents a car arriving to the origin cell from the right. The car should not be able to move forward (*(-1,0) != 0*), it should have enough space to move (*(0,-1) = 0 and (0,0) = 0*), and a truck should not be able to arrive to the same position (*!(Truck(1,0))*).

τ(N)	N
1	(0,0) = and (-1,-1)=1 and (-1,0) !=0 and (0,-1) =0 and !(Truck(1,0))

When a car moves to the right, we must check that enough space to move is available. In addition, no trucks (with higher priority) or cars must be trying to reach same position *((-2,0) = 0 or (-2,1) = 0)*.

τ(N)	N
0	(0,0) = 1 and (-1,0) = 0 and (-1,1) = 0 and !(Truck(0,1)) and ((-2,0) = 0 or (-2,1) = 0)

Finally, a car can arrive from the left. We must check there is no space to advance, the car cannot stay in its own lane, and it cannot move to the left. Besides, it checks if there is a truck moving to the origin cell.

τ(N)	N
1	(0,0)=0 and (1,-1)=1 and (1,0)!=0 and !(Truck(1,0)) and (0,-1)=0 and ((2,-1)!= 0 or (2,0) != 0 or Truck(3,0))

After defining the individual behavior of each cell in the model, a Cell-DEVS coupled model should be built for the segment. In this case, the coupled model is defined by:

$$SLT(k, max, \#s) =< Xlist, Ylist, I, X, Y, n, \{t_1, ..., t_n\}, \eta, N, C, B, Z >$$

Xlist = **Ylist** = { (i,0) / i ∈ [0, #s] } U { (i,k-1) / i ∈ [0, #s] }
I = <P^x, P^y >, with P^x= {<$X_{\eta+1}$(i,0), *N*> / i ∈ [0, #s]} U {<$X_{\eta+1}$(i,k-1), *N*> / i ∈ [0, #s]} and P^y= {<$Y_{\eta+1}$(i,0), *N*> / i ∈ [0, #s]} U {<$Y_{\eta+1}$(i,k-1), *N*> / i ∈ [0, #s]}.
X = *Y* = N; **n** = 2; $\mathbf{t}_1$ = #s; $\mathbf{t}_2$ = k; **C** = { C_{ij} / i ∈ [0, #s-1] ∧ j ∈ [0, k-1] };
B = { (i,j) / ∈ [0, #s-1] ∧ j ∈ [0, 5] } U { (i,j) / i ∈ [0, #s-1] ∧ j ∈ [k-2, k-1] }; and
Z is built by following the definition for Cell-DEVS models.

Here, SLT(k, max, #s) represents a segment of *#s* lanes of length *k* each with speed level *max*. Variable *k* is computed using *Cells-Nr(t)*, a function that takes the length of the segment and the size of each cell.

The following input/output ports names are used in the following rule definitions:

Name	Comment
x-c-vehicle	A vehicle leaves a crossing to a segment.
x-c-room	There is enough room in the crossing so a vehicle can advance from the segment.
y-c-room	There is enough room in the segment so a vehicle can leave the crossing.
y-c-vehicle	A vehicle in the segment is trying to cross.

The border cells of the coupled model are connected with the crossings, therefore, their behavior is different from the rest of the cell space. The column 0 will be devoted to receive trucks from the crossing. The trucks in a crossing are represented using only one cell. Hence, when they return to a segment their full length must be reconstructed. This rule represents a car arriving to the origin cell from the crossing:

τ(N)	N
1	(0,0) = 0 and x-c-vehicle = 1

The following rule represents the arrival of a truck from the crossing:

τ(N)	N
x-c-vehicle	(0,0) = 0 and IsTruck(x-c-vehicle)

The following rules will represent that the truck size expands when it leaves the crossing:

τ(N)	N
0	Truck(0,0) and (0,1) = 0 and CompleteTruck(0,0) send(0,y-c-room)
0	Truck(0,0) and (0,1) = 0 and !CompleteTruck(0,0) and !IsHead_TC1(0,0)) send((0,0),y-c-room)
(0,1)	(0,0) = 0 and Truck(0,1) and !CompleteTruck(0,1) send((0,1),y-c-room)
0	Truck(0,0) and (0,1) = 0 and IsHead_TC1(0,0) send((0,0),y-c-room)

Here, *IsTruck(k)* verifies that a value passed as parameter represents a truck, that is, IsTruck(k) $\equiv$ remainder(k,10) $\in$ [2,5]. *CompleteTruck(i,j)* is used to see if the truck has been generated completely.

The first rule represents that the truck has been generated completely. Consequently, the crossing should know that it is ready to receive another

vehicle. This is done sending a 0 value to the crossing. The second rule represents that part of the truck advanced to the following cell, but the truck is still being reconstructed. Therefore, the cell in the crossing is busy. The third rule represents the generation of a new part of the truck in the origin cell. The last rule is equivalent to the second, but considering that the origin cell contains the truck's head. Therefore, its output must be delayed (in the other rules, 0 delays are used).

The following rules specify the size reduction of a truck arriving to a crossing:

τ(N)	N
(0,-1)	(0,0)=0 and Truck(0,-1) and CompleteTruck(0,-1) send(0,y-c-vehicle)
0	Truck(0,0) and x-c-room = 0 send((0,0),y-c-vehicle)
0	(0,0)=0 and Truck(0,-1) and !CompleteTruck(0,-1) send(0,y-c-vehicle)

The first rule represents the arrival of a truck to a crossing when there is enough room to cross. The value representing the truck's number is transmitted only if the cell contains the head of the truck. The second rule represents a truck arrived to the crossing. The third rule represents the input of the remaining parts of the truck. Finally, the following rules are related with the interchange of cars. The next one is in charge of sending a car to the crossing:

τ(N)	N
0	(0,0) = 1 and x-c-room = 0 send(1,y-c-vehicle)

Here, *SLT*(*k, max, #s*) represents a segment of *#s* lanes of length *k* each with speed level *max*. Variable *k* is computed using the function *segment_len* defined earlier. Here, if we consider the example presented in section 2, we can define, $SLT_{rG1}(20, 60, 4)$ as:

$\mathbf{Y}_{\text{list rG1}} = \mathbf{X}_{\text{list rG1}} = \{ (i,0) / i \in [0, 4] \} \cup \{ (i,19) / i \in [0, 4] \}$;
$\mathbf{I}_{\text{rG1}} = <P^x, P^y>$, with $P^x = \{<X_{\eta+1}(i,0), N> / i \in [0, 4]\} \cup \{<X_{\eta+1}(i,19), N> / i \in [0, 4]\}$ and $P^y = \{<Y_{\eta+1}(i,0), N> / i \in [0, 4]\} \cup \{<Y_{\eta+1}(i,19), N> / i \in [0, 4]\}$.
$\boldsymbol{X}_{\text{rG1}} = \boldsymbol{Y}_{\text{rG1}} = \boldsymbol{N}$; $\mathbf{n}_{\text{rG1}} = 2$; $\mathbf{t}_{1\,\text{rG1}} = 4$; $\mathbf{t}_{2\,\text{rG1}} = 20$; $\mathbf{C}_{\text{rG1}} = \{ TS_{ij} / i \in [0, 4\text{-}1] \wedge j \in [0, 19] \}$;
$\mathbf{B}_{\text{rG1}} = \{ (i,j) / \in [0, 3] \wedge j \in [0, 5] \} \cup \{ (i,j) / i \in [0, 3] \wedge j \in [18, 19] \}$.

The border cells of the coupled model are connected with the crossings, therefore, their behavior is different from the rest of the cell space. The

column 0 will be devoted to receive trucks from the crossing, represented using only one cell. Hence, when they return to a segment their full length must be reconstructed. If a truck has been regenerated completely, the crossing should know that it is ready to receive another vehicle. This is done sending a 0 value to the crossing. On the opposite side, a truck size must be reduced when it arrives to a crossing. A truck enters a crossing only when there is enough room to cross. The value representing the truck number is transmitted only if the cell contains the head of the truck. The remaining parts of the truck are discarded.

3.3 Crossings

Crossings are formally specified as:

$$Crossings = \{(c, max)/c \in City \wedge max \in \mathbf{N} \wedge \exists s, s' \in Segments \wedge s$$
$$= (p1, p2, n, a, dir, max) \wedge s' = (p1', p2', n', a', dir', max') \wedge s \neq s' \wedge$$
$$(p1 = c \vee p2 = c) \wedge (p1' = c \vee p2' = c)\}$$

with the different parameters defined as for the segments constructions.

Crossings are represented as a ring of cells in which vehicles move up to deciding to get through another path. This approach was used previously in (Chopard et al. 1995, Chopard et al. 1996, Chopard et al. 1997). A car in the intersection has higher priority to obtain a position into the ring than the cars out of the crossing. In the crossing, the cars advance continuously in order to avoid deadlocks. Each crossing should have at least one input and one output cell. In this way, the vehicles getting into a crossing from a given segment will always have one available output, as depicted in Figure 10.

Atomic cells are based on the one lane model of section 3. Besides, as each cell is connected with a segment influencing its behavior, different functions must be provided. Here, τ: S x N $\longrightarrow$ S will execute different rules for input or output cells. Output cells behavior is defined by:

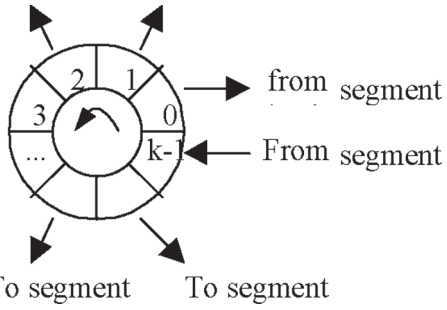

FIGURE 10
A Crossing.

τ(N)	N
1	(0,0) = 0 and (0,-1) = 1 and [portvalue(x-s-space) = 1 or (portvalue(x-s-space) = 0 and random $<$ p_{out})] send(0, y-s-vehicle) /* A car kept into the crossing arrives */
0	(0,0) = 0 and (0,-1) = 1 and portvalue(x-s-space) = 0 and random $\geq$ p_{out} send(1, y-s-vehicle) /* A car abandoning the crossing arrives */
0	(0,0) = 1 and (0,1) = 0 send(0, y-s-vehicle) /* A car abandons the present cell */
(0,0)	TRUE send(0, y-s-vehicle) /*Otherwise, the cell keeps the present state */

The first rule of this definition represents a car arriving to the cell being considered. Here, *random* is a probability function that can be tuned to define random routing. If the value returned by the function is higher than the constant p_{out}, the vehicle abandons the origin cell towards to the corresponding segment. Therefore, a car remains in the crossing if the output segment is busy (*portvalue(x-s-space) = 1*), or the random test fails. In these cases, the output port *y-s-vehicle* (which corresponds to the output segment) must be set to 0. Otherwise, the segment will detect the car and will react as in a transfer. The second rule represents the arrival of a vehicle from the previous cell that will abandon the crossing. The first cell in the segment must be empty, and the random function should return a value higher than the predefined constant. The third rule represents that a vehicle abandons the present cell.

The behavior for input cells is defined by:

τ (N)	N
1	(0,0) = 0 and [(0,-1) = 1 or portvalue(x-s-vehicle) = 1] send(1, y-s-space) /* Arrival of a new car */
0	(0,0) = 1 and (0,1) = 0 and (0,-1) = 0 send(0, y-s-space) /* No cars with priority into the crossing */
0	(0,0) = 1 and (0,1) = 0 and (0,-1) = 1 send(1, y-s-space) /* Cars with priority into the crossing */
(0,0)	TRUE /* Otherwise, keep the previous state */

The input cells can receive vehicles from the previous cell in the crossing, or from a segment coupled to it (using the port *x-s-vehicle*). The first rule represents the arrival of a car to a cell. The origin must be empty, and a vehicle is available. The port related with the segment must be updated to tell that the cell is busy. The second and third rules represent that a car abandons a cell when the front cell is free. If the back cell is empty, the

output port *x-s-space* is updated with a 0 to tell the segment that there is enough room to receive a vehicle. Otherwise, a 1 is sent to the segment to tell that a car in the crossing is trying to use the cell, due that cars in the crossing have higher priority to use them.

The coupled model corresponding to the crossings is defined by:

$$Crossing(k, In, Out) =< X_{list}, Y_{list}, I, X, Y, n, \{t_1, ..., t_n\}, \eta, N, C, B, Z >$$

which defines a crossing of k cells, where the positions in the set *In* are the crossing inputs, and *Out* their outputs. Here,

$\mathbf{Y}_{\text{list}} = \mathbf{X}_{\text{list}} = \{ (0,i) / 0 \leq i < k\}$; $\boldsymbol{X} = \boldsymbol{Y} = \{0, 1\}$;
$\mathbf{I} = <P^x, P^y>$, with $P^x = \{<X_{\eta+1}(0,i), \text{binary}> / 0 \leq i < k \}$, $P^y = \{<Y_{\eta+1}(0,i), \text{binary}> / 0 \leq i < k \}$
$\mathbf{n} = 1$, $\mathbf{t}_1 = k$, $\mathbf{N} = \{ (0,-1), (0,0), (0,1) \}$, $\mathbf{B} = \{\emptyset\}$.
Z is built using the definition given in the Cell-DEVS formalism, considering the present neighborhood.

The parameters (k, *In*, *Out*) are built checking the direction of the segments connected to the crossing. The number of lanes of the crossing is computed as: $k = \sum_{s \in S \wedge s=(c1,c2,n,a,dir) \wedge (c1=p \vee c2=p)} n$. As we can see, we compute the number of lanes in each segment connected to the crossing. Then, a unique ordering for the input/output segments is defined so that the closer segments are connected to neighboring cells. This ordering considers the angle between the segment and the line defined by y = 0. The function

$$Ports\_In\_Out[(c, maxc) : Crossings, T : Segments] : In =$$
$$\{(s, i)/s \in Segments, i \in \boldsymbol{N}\}, Out = \{(s, i)/s \in Segments, i \in \boldsymbol{N}\},$$

receives a crossing, the set of segments connected to it, and returns the input and output sets for the crossing. The position in which each segment is coupled into the crossing the segments are ordered according with the incidence angle respect to the line y = y1, with c = (x1,y1). The greater the angle, the higher position into the crossing.

First, we obtain the segments in S such that any of its borders is the crossing c:

$$S' = \{s/s \in S \wedge s = (c1, c2, n, a, dir, max) \wedge (c1 = c \vee c2 = c)\}$$

Then, we evaluate the incidence angle of these segments. The positions of the crossing are assigned as increasing numbers, according with the position used to be coupled. This information is stored in the set V, containing tuples (s, i, α), which represent that the first lane of segment s is connected with i cell of the crossing (*c,maxc*), where α is the incidence angle of s.

$$V = \{((c_1, c_2, n, a, dir, max), i, \alpha) / [(c \neq c_1 \wedge \alpha = Angle(c, c_1)) \vee (c \neq c_2 \wedge \alpha = Angle(c, c_2))]\}, \text{ where}$$

$$i = \begin{cases} \sum_{(c'_1,c'_2,n',a',dir',max') \in S' \wedge Angle(c'_1,c'_2) < \alpha} n' & if\ (dir = 1 \Rightarrow c_2 = c) \\ & \wedge\ (dir = 0 \Rightarrow c_1 = c) \\ \sum_{(c'_1, c'_2, n', a', dir', max') \in S' \wedge Angle(c'_1, c'_2) \leq \alpha \wedge (c'_1, c'_2, n', a', dir', max') \neq (c_1, c_2, n, a, dir, max)} n' & if\ (dir = 0 \Rightarrow c_2 = c) \\ & \wedge\ (dir = 1 \Rightarrow c_1 = c) \end{cases}$$

We consider that if two segments have the same incidence angle, the segment going into c is connected first ($[dir = 1 \Rightarrow c_2 = c] \wedge [dir = 0 \Rightarrow c_1 = c]$). Then, we add the number of lanes in the segments whose angle is smaller than α (if there is another segment with the same angle, it will be after s). In the second case, as ($[dir = 0 \Rightarrow c_2 = c] \wedge [dir = 1 \Rightarrow c_1 = c]$), s has the opposite direction to c. Then, we add the number of lanes in the segments with incidence angle smaller or equal to α (if there is another segment with the same angle, it will be before s).

The incidence angle (α) of the segment defined by the points (*x1,y1*), (*x2,y2*) respect to the line y = y1. We know that sin(α) = |y1-y2|/| $\sqrt{|x1 - x2|^2 + |y1 - y2|^2}$|. Then: $\alpha = arcsin\left(\frac{|y1-y2|}{\sqrt{|x1-x2|^2+|y1-y2|^2}}\right)$. If [(y1 > y2) $\vee$ (y1 = y2 $\wedge$ x1 < x2)] then $\alpha = \alpha + \pi$. We have to consider that the segment defined by (*x1,y1*) and (*x2,y2*) will have a different incidence angle in each of the extremes, as it is shown in Figure 11. To obtain the angle regarding any of them, we must define it as the first parameter in the Angle function.

Finally, the tuples in V are separated in those corresponding to input segments and output segments. This is done by checking the direction of the vehicles defined for each segment.

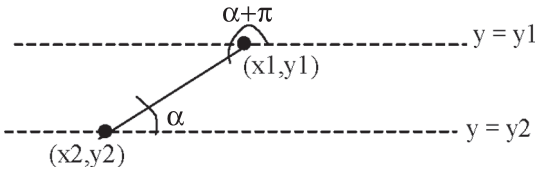

FIGURE 11
Incidence angles.

$$In = \{(s,i)/(s,i,\alpha) \in V \wedge s = (c1, c2, n, a, dir, max) \wedge$$
$$[(c1 = c \wedge dir = 0) \vee (c2 = c \wedge dir = 1)]\}$$
$$Out = \{(s,i)/(s,i,\alpha) \in V \wedge s = (c1, c2, n, a, dir, max) \wedge$$
$$[(c1 = c \wedge dir = 1) \vee (c2 = c \wedge dir = 0)]\}$$

When trucks are allowed in the crossing, the following construction is used:

$$TruckXings = \{(c, maxc)/maxc \in \boldsymbol{N} \wedge \exists t, t' \in$$
$$(TruckSegments\ USegments) \wedge t = (p1, p2, n, a, dir, max) \wedge t'$$
$$= (p1', p2', n', a', dir', max') \wedge t \neq t' \wedge$$
$$(p1 = c \vee p2 = c) \wedge (p1' = c \vee p2' = c)\}$$

This set is defined as points in a bidimensional space, representing the places where two or more segments are crossed. It is built using Segments and TruckSegments sets. These crossings must recognize the segments allowed to receive trucks. The standard binary crossings are used for car crossings. Instead, if one of the *s* segments belongs to TruckSegments, the crossing is defined as part of TruckXings.

Each crossing (c, maxc) ∈ TruckXings is defined as a one-dimensional Cell-DEVS with transport delays:

$$C_{0j} =< I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

I = < η, P^x, P^y >, with η = 3; P^x = { (X_1, *N*), (X_2, *N*), (X_3, *N*) }; P^y = { (Y_1, *N*), (Y_2, *N*), (Y_3, *N*) }; $\boldsymbol{X} = \boldsymbol{Y}$ = N;

S:

$$s = \begin{cases} 1 & \text{if there is a car in the cell;} \\ 0 & \text{if the cell is empty;} \\ k = r \bmod 10 \wedge r \in [2,5] & \text{if there is a truck.} \end{cases}$$

N = { (0,-1), (0,0), (0,1) }; **delay** = transport; **d** = speed(maxc);

D, λ, δ_{int} and δ_{ext} are defined by the Cell-DEVS formalism with transport delays.

The function τ will be defined informally (a detailed explanation of the formal specifications can be found in (Davidson and Wainer 2000b)). The output cells of these crossing behave according to the segment to which they are coupled. Depending if cars or trucks are allowed, different approaches are used. As explained earlier, a vehicle can leave the crossing if there is enough space in the segment where it is going and a random

test is positive. A truck passing through cells connected to segments not allowing trucks will keep moving. The output cells connected to segments where trucks are not allowed use the rules defined earlier for cars. However, if the vehicle is a truck, it cannot leave the crossing through these cells.

The local computing function of the input cells is similar, but it receives integer values representing a truck identifier. If the coupled segment is binary, it will not accept trucks. Therefore, a value of 1, representing that the cell is busy is sent to the segment. The segment will wait the cell to be empty, which will be done by sending a 0 value through the output port. Otherwise, if there is a vehicle with priority in the crossing, the cell sends a 1. In this way, a new arriving vehicle knows that it must stop before the crossing.

The coupled model corresponding to the TruckXing *(c, maxc)* is defined by:

$$TruckXing(k, In, In\_Cars, Out, Out\_Cars, 1, maxc) =< X_{list}, Y_{list}, I, X, Y, n, \{t_1, ..., t_n\}, \eta, N, C, B, Z >$$

$\mathbf{Y}_{\text{list}} = \mathbf{X}_{\text{list}} = \{ (0,i) / i \in [0, k] \}$; $\boldsymbol{X} = \boldsymbol{Y} = N$; $\mathbf{n} = 1$; $\mathbf{t}_1 = k$; $\mathbf{C} = \{TC_{0j}/j \in [0, k-1] \}$; $\mathbf{B} = \{\emptyset\}$;

$\mathbf{I} = <P^x, P^y>$, with $P^x = \{<X_{\eta+1}(0,i), \text{binary}> / i \in [0, k]\}$, $P^y = \{<Y_{\eta+1}(0,i), \text{binary}> / i \in [0, k]\}$.

Z is built using the specification of Cell-DEVS.

This is a crossing of k cells, with a maximum speed of *maxc*. The positions of *In* are the inputs to the crossing, and *Out/Out_cars* are the outputs. Here, *Out_cars* represents the set of segment that cannot receive trucks. These sets are obtained by computing

$$\{I, O\} = Ports\_In\_Out((c, maxc), Segments\ U\ TruckSegments)$$

The function *Ports_In_Out* takes the specification of the segments connected to the crossing and their direction. The result of the {I, O} sets define which cells are connected with each input/output segment. The procedure to obtain these sets was explained earlier, and now we consider the type of input/output cells using the segments to which they are connected.

Using these definitions, $\boldsymbol{\tau}$ will be specified. A different behavior should be provided for input and output cells. A crossing c = (p, maxc) influences the segments s to which it is connected, that is,

$$I_c = \{M_s/s \in (Segments\ U\ TruckSegments) \wedge s = (p1, p2, n, a, dir, max) \wedge (p1 = p\ or\ p2 = p)\}$$

Therefore, a segment s influences to the two crossings in its border:

$$I_s = \{M_{c1}\}U\{M_{c2}\}, if s = (p1, p2, n, a, dir, max) \wedge (\exists v1, v2 \in \boldsymbol{N} : c1, c2 \in (Crossings\ U\ TruckXings) \wedge c1 = (p1, v1) \wedge c2 = (p2, v2))$$

These ports are coupled by defining the Z function. To define the cells in the crossing and the segment that will be connected to the {I, O} sets computed previously. As the coupling is done in the first (0) and last (k-1) cells, for each (s,i) ∈ I, s = (p1, p2, n, a, dir, max) we must know the number of cells in the segment. These values are obtained computing the length of the segment (using p1 and p2) and dividing the result by the size of each cell. In this case, Z is defined by:

$Z_{sc} : Y_{\eta+1}(j, k\text{-}1)_s \rightarrow X_{\eta+1}(0,i+j)_c, \forall (j \in N, j \in [0, n\text{-}1])$
$Z_{ct} : Y_{\eta+1}(0,i+j)_c \rightarrow X_{\eta+1}(j, k\text{-}1)_s, \forall (j \in N, j \in [0, n\text{-}1])$

Instead, for each (s,i) ∈ O with s = (p1, p2, n, a, dir, max), Z is defined by:

$Z_{cs} : Y_{\eta+1}(0,j+i)_c \rightarrow X_{\eta+1}(n\text{-}1\text{-}j, 0)_s, \forall (j \in N, j \in [0, n\text{-}1])$
$Z_{sc} : Y_{\eta+1}(n\text{-}1\text{-}j, 0)_s \rightarrow X_{\eta+1}(0,j+i)_c, \forall (j \in N, j \in [0, n\text{-}1])$

If we consider the crossing c6 presented in the example introduced in Section 2, we can build $TruckXings_{c6}$(9, In, In_Cars, Out, Out_Cars, 1, 30) as follows

$\mathbf{Y_{list\ c6}} = \mathbf{X_{list\ c6}} = \{ (0,i) / i \in [0, 9] \};$
$\mathbf{I}_{c6} = <P^x, P^y>$, with $P^x = \{<X_{\eta+1}(0,i), \text{binary}> / i \in [0, 9]\}$, $P^y = \{<Y_{\eta+1}(0,i), \text{binary}> / i \in [0, 9]\}$.
$\boldsymbol{X}_{c6} = \boldsymbol{Y}_{c6} = N; \mathbf{n}_{c6} = 1; \mathbf{t}_{1c6} = 9; \mathbf{C}_{c6} = \{ TC_{0j} / j \in [0, 9\text{-}1] \}; \mathbf{B}_{c6} = \{\emptyset\}.$

The *Ports-In-Out* function returns:
In={G1, H1}; In-Cars={E1, D1, I1}; Out={G2,H2}; Out-Cars={D2,I2}

Once the behavior for crossings and segments is completed, the coupling between them must be defined. A crossing c = (*p, maxc*) influences the segments s to which it is connected, as follows:

$$I_c = \{M_s / s \in (Segments\ U\ TruckSegments) \wedge s = (p1, p2, n, a, dir, max) \wedge (p1 = c\ or\ p2 = c)\}$$

After, a segment should s influence to the two crossings in its borders as follows:

$$I_s = \{M_{c1}\} U \{M_{c2}\}, if\ s = (p1, p2, n, a, dir, max)\ and\ (\exists v1, v2 \in \boldsymbol{N} : c1, c2 \in (Crossings\ U\ TruckXings) \wedge c1 = (p1, v1) \wedge c2 = (p2, v2))$$

The interconnection is built using the {I, O} sets computed previously. Using this information and length of the segment, we build the Z function, defined by:

$Z_{sc} : Y_{\eta+1}(j, k\text{-}1)_s \rightarrow X_{\eta+1}(0,i+j)_c, \forall (j \in N, j \in [0, n\text{-}1])$
$Z_{cs} : Y_{\eta+1}(0,i+j)_c \rightarrow X_{\eta+1}(j, k\text{-}1)_s, \forall (j \in N, j \in [0, n\text{-}1])$

Instead, for each $(s, i) \in O$ with $s = (p1, p2, n, a, dir, max)$, Z is defined by:

$Z_{cs} : Y_{\eta+1}(0,j+i)_c \rightarrow X_{\eta+1}(j, 0)_s, \forall (j \in N, j \in [0, n-1])$

$Z_{sc} : Y_{\eta+1}(j, 0)_s \rightarrow X_{\eta+1}(0,j+i)_c, \forall (j \in N, j \in [0, n-1])$

Let us suppose that we want to analyze the connection between *c*6 and the segment *rG*. Here,

$I_{c6} = \{ M_s / s \in (Segments \cup TruckSegments) \wedge s = (p1, p2, n, a, dir, max) \wedge (p1 = c6 \text{ or } p2 = c6) \} = \{rI1, rH1, rD1, rG1\}$

$I_{rG1} = \{c5\} \cup \{c6\};$

Inputs: $I = \{1, 9, 13, 18\}$. Here, rG1 is related with $i = 1 \in I$, then,

$Z_{rG1,c6} : Y_4(j, 19)_{rG1} \rightarrow X_4(0,j+1)_{c6}, \forall (j \in N, j \in [0, 3])$

$Z_{c6,rG1} : Y_4(0,j+1)_{c6} \rightarrow X_4(j, 19)_{rG1}, \forall (j \in N, j \in [0, 3])$

Outputs: $O = \{5, 11, 15, 17, 20 \}$. Here, for rG2 is related with $i = 5 \in O$, then,

$Z_{c6,rG2} : Y_4(0,j+5)_{c6} \rightarrow X_4(j, 0)_{rG1}, \forall (j \in N, j \in [0, 3])$

$Z_{rG2,c6} : Y_4(j, 0)_{rG1} \rightarrow X_4(0,j+5)_{c6}, \forall (j \in N, j \in [0, 3])$

3.4 Traffic lights

In ATLAS, the crossing with traffic lights is defined using the following construction:

$$TLCrossings = \{c / c \in Crossings\}.$$

Each element in this set is mapped into one DEVS, transmitting the light color to the corresponding segment in the intersection. Another DEVS is in charge of synchronizing all the lights in the corner. An upper level Cell-DEVS can be built to coordinate all the controllers in a city section.

For every $c \in$ TLCrossings, the DEVS model Sync(#tl) = < I, X, S, Y, δint, δext, λ, D > is created. This controller defines the color of each traffic light. The parameter #tl defines the number of traffic lights, depending on the number of input segments:

$$T_{in} = \{s / s \in Segments \wedge s = (c1, c2, n, a, dir, max)$$
$$\wedge [(c1 = c \wedge dir = 0) \vee (c2 = c \wedge dir = 1)]\}$$

For each $t \in T_{in}$, a DEVS model representing the traffic light is built: TL(#c) = < I, X, S, Y, δint, δext, λ, D >. This model informs its color to an input segment. This is informally depicted in Figure 12 (specification details of these models can be found in (Davidson and Wainer 1999)). For every $c \in$ *TLCrossings*, a DEVS model Sync(#tl) =<I, X, S, Y, δint, δext, λ, D > is created. This is a high level controller defining the color of each traffic light connected to the corner. The parameter *#tl* defines the

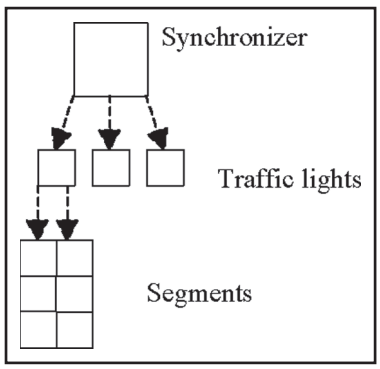

FIGURE 12
Crossing with traffic lights.

number of traffic lights, depending on the number of input segments to the crossing. This set is defined by:

$$T_{in}(c) = \{s/s \in Segments \wedge s = (c1, c2, n, a, dir, max)$$
$$\wedge[(c1 = c \wedge dir = 0) \vee (c2 = c \wedge dir = 1)]\}$$

For instance, for the crossing *c6*, we build T_{in}(c6) = {rG1, rH2, rD1, rI2}. Therefore, Sync(4) = <I, X, S, Y, δint, δext, λ, D>. For each $s \in T_{in}(c)$, we build a DEVS model representing the traffic light: TL(s) = < I, X, S, Y, δint, δext, λ, D >. This model informs its color to an input segment, and waits an order of the *Sync* model to change the light color. Specification details of these DEVS models can be found in (Davidson and Wainer 2000a).

Every input segment of the crossing should reflect the existence of the traffic light. The basic rules for the segments were modified to represent this behavior. The new model is defined by:

$$M_R =< X_{list}, Y_{list}, I, X, Y, n, \{t_1, t_2\}, \eta, N, C, B, Z, select >$$

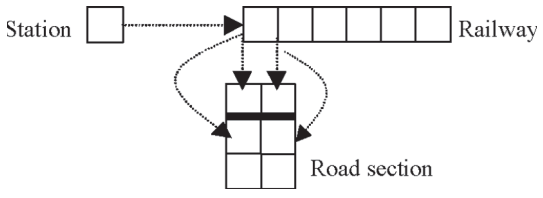

FIGURE 13
Level crossing definition.

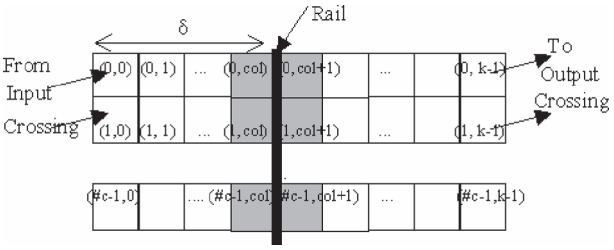

FIGURE 14
Segment with level crossing.

τ(N)	N
1	(0,-1) = 1 and (0,0) = 0
0	(0,0) = 1 and portvalue(x-c-space) = 0 and portvalue(x-light) = 0 send(1, y-c-car)
(0,0)	TRUE /*Otherwise: state unchanged */

Specialized ports inform the color of the traffic light to the cells in the segment. If its value is 1, it is red. Otherwise, it is green. These border cells are defined by $\{(i, t_2\text{-}1) / i \in [1,t_1]\}$.

Delay: inertial. Duration: speed.

These rules represent that a vehicle can cross if there is space (*portvalue(x-c-space) = 0*) and the traffic light is green (*portvalue(x-light) = 0*). Then, a 1 value is sent to the crossing telling that the car is passing. An inertial delay is used because a car can enter the crossing if there is a free space and the light is green during the time specified. Otherwise, the car must remain waiting.

Finally, $\forall$ c $\in$ Crossings, we must define the cells' coupling. Crossing(c, k, In, Out) defines a crossing of k cells, where the set In defines are input positions, and Out their outputs. The number of cells in the crossing (k) is defined counting the number of input lanes. These ones depend on the segments connected to it and their direction. We have defined a unique ordering for the input segments, such that close roads are connected to neighboring cells. This ordering is built using the incidence angle between the segment and the line y = 0.

Once the In and Out sets are defined, every pair (s,j) $\in$ In tells that the segment s is connected to an input using the j-eth cell in the crossing. This numbering of the segments is also used to establish the order to receive the green lights.

The synchronizer influences the TL models corresponding to the input segments for the crossing c. That is, $I_{sync} = \{TL_i / i \in [0, \#(In)]\}$. In

addition, every traffic light influences the behavior of the input segment, according to the established order. That is, $I_{TLi} = \{ M_s / (s,j) \in In \wedge i = \#(\{(s',j') \in In / j' < j \})\}$.

Therefore, the coupling function is defined as

$Z_{sync,TLi}$: $(y\text{-}ligth_i)_{sync} \rightarrow (x\text{-}light_i)_{TLi}$, $\forall$ i $\in$ N, i$\in$[0,#(In)]
$Z_{Tli,s}$: $(y\text{-}s\text{-}light_h)_{Tli} \rightarrow X\ \eta$ +2(h, k-1)$_s$, $\forall$ (h $\in$ N, h $\in$ [0, n-1]) $\wedge$ (s,j) $\in$ In $\wedge$ i = #({(s',j') $\in$ In / j' < j }). Here, n is the number of lanes in s (n = t1 of the model Mt), and k is the number of cells in every lane (k = t2 of the model Mt).

Finally, we must define the cell coupling $\forall$ c $\in$ Crossings. Using the procedures previously defined, the *In* and *Out* sets are defined. Every pair $(s, j) \in In$ tells that the s is an input segment connected to the j-eth cell of the crossing. That is, $I_{TLi} = \{ M_s / (s,j) \in In \wedge i = \#(\{(s',j') \in In / j' < j \})\}$. This ordering is also used by the synchronizer define the order to receive the green lights. $I_{sync} = \{TL_i / i \in [0, \#(In)] \}$. That is, the synchronizer influences the TL models corresponding to the input segments. Therefore, the translation function is defined by:

$Z_{sync,TLi}$: $(y\text{-}ligth_i)_{sync} \rightarrow (x\text{-}light_i)_{TLi}$, $\forall$ i$\in$ N, i $\in$[0,#(In)]
$Z_{Tli,s}$: $(y\text{-}s\text{-}light_h)_{Tli} \rightarrow X\eta + 2(h, k-1)_s$, $\forall$ (h $\in$ N, h $\in$ [0, n-1]) $\wedge$ (s,j) $\in$ In $\wedge$ i = #({(s',j') $\in$ In / j' < j }). Here, n is the number of lanes in s (n = t1 of the model), and k is the number of cells in every lane (k = t2 of the model).

For instance, if we analyze the crossing c6, we know that
$I_{sync} = \{ M_{TLi}\}$, $\forall$i $\in$ N, i $\in$ [0, 3] (because we have 4 traffic lights controlled by the synchronizer)
$I_{TLi} = \{ M_s / s \in T_{in}(c6)\} = \{$rG1, rH2, rD1, rI2 $\}$
$Z_{sync,TLi}$: $(y\text{-}ligth_i)_{sync} \rightarrow (x\text{-}light)_{TLi}$, $\forall$ i $\in$ N, i $\in$[0,3]
$Z_{TL1,G1}$: $(y\text{-}s\text{-}light_h)_{TL1} \rightarrow X\eta$+2(h, 19)$_{G1}$, $\forall$ (h $\in$ N, h $\in$ [0, 3])
$Z_{TL2,H2}$: $(y\text{-}s\text{-}light_h)_{TL2} \rightarrow X\eta + 2(h, 13)_{H2}$, $\forall$(h $\in$ N, h $\in$ [0, 1])
$Z_{TL3,D1}$: $(y\text{-}s\text{-}light_h)_{TL3} \rightarrow X\eta + 2(h, 14)_{D1}$, $\forall$(h $\in$ N, h $\in$ [0, 1])
$Z_{TL4,I2}$: $(y\text{-}s\text{-}light_h)_{TL4} \rightarrow X\eta + 2(h, 9)_{I2}$, $\forall$(h $\in$ N, h $\in$ [0, 1])

Here, we show the connection among the four traffic lights and each input segment for the crossing c6. The changes of wider segments are extensions of those here presented. Detailed specification can be found in (Davidson and Wainer 2000a).

3.5 Railways

We defined railways, which are built as a sequence of level crossings overlapped with the segments. Each train is defined as a Cell-DEVS following a predefined advance sequence through level crossings. The railway network is defined by:

$$Railnets = \{Rail/Rail \in RailTrack\}, where\ RailTrack = \{(s, \delta, seq)/s \in Segments \wedge \delta \in N \wedge seq \in N\}$$

Every RailTrack $\in$ Railnets represents a part of the rail network in a given city section. The shape of the network is defined by the places where level crossings are placed. Every tuple pn = (s, δ, seq) identifies the position of the level crossing. First, the segment crossed (s); then, the distance between the beginning of the section and the railway (δ), and finally, a sequence number defining the position of the crossing in the railway. This sequence starts in zero, and finishes in the number of tuples in the set.

For every element in Railnets, two models are defined. The first one is a DEVS, representing a station from where the trains depart. The second is a one-dimensional Cell-DEVS whose cells model all the level crossings. The associated station is defined as a DEVS model. It represents the train departure according to the railway's timetable. It is connected with a RailTrack model, defined as a one-dimensional Cell-DEVS with transport delays. Every cell in the model is specified as:

$$C_R(seq) =< I, X, S, Y, N, \delta int, \delta ext, delay, d, \tau, \delta, D >$$

$$\mathrm{I} =< \mathrm{P^x}, \mathrm{P^y} >; \mathrm{P^x} = \{(\mathrm{X1},\ \mathrm{binary}), (\mathrm{X2},\ \mathrm{binary})\};$$

$$\mathrm{P^y} = \{(\mathrm{Y1},\ \mathrm{binary}), (\mathrm{Y2},\ \mathrm{binary})\}.$$

$$\mathrm{X} = \mathrm{Y} = \{0, 1\}; \mathrm{N} = \{(0, -1), (0, 0)\}; \mathrm{delay} = \mathrm{transport}; \mathrm{d} = \mathrm{dtrain}();$$

$$\mathrm{S} = \begin{cases} 1 \text{ if a train is passing;} \\ 0 \qquad \text{otherwise.} \end{cases}$$

λ, δint and δext are defined by Cell-DEVS;
τ is defined as:

τ(N)	N
0	(0,0) = 1
1	(0,-1) = 1
(0,0)	TRUE /* Otherwise it preserves the present state */

The cell's state represents the presence of a train that advances independently of the cells in front of it. Therefore, the neighborhood only includes the back cell. Transport delays define the train speed, using *dtrain()*. This function computes the delay of the train between crossings, depending on the distance between them. The train movement is defined by three rules. The first one represents a train leaving the crossing. The second one represents an empty crossing that is occupied by a train. The

last one preserves the present state in every other case. A coupled model of the railway is built using the following definition:

$$RT(k, Out) = < X_{list}, Y_{list}, I, X, Y, n, \{t1, ..., tn\}, \eta, N, C, B, Z >$$

Y_{list} = {(0,i) / i ∈ N ∧ i ∈ [0, k] }; X_{list} = { (0,0) }.
I=<P^x, P^y >, with P^x={<$X_{\eta+1}$(0,0), binary>}, P^y={<$Y_{\eta+j}$(0,i), binary>/ i ∈ N ∧ i ∈ [0, k] ∧ (i, #p) ∈ Out ∧ j ∈ [1, #p] }, where Out = { (i, 2n) / i ∈ N ∧ i ∈ [0,k] ∧ ((c1, c2, n, a, dir, max), δ, i) ∈ RailTrack}.
X = Y = { 0,1 } ; n = 1; t1 = k; η = 2; N = { (0,-1), (0,0) }
C = { C_{Rij} / i = 0 ∧ j ∈ [0, k-1] }, with C_{Rij} defined earlier; B = { (0,0) };
Z is built using the definition given by Cell-DEVS.

This model represents a railway layout. Here, k represents the number of level crossings, and it is obtained counting the elements of the set (k = #RailTrack). The Out set contains the number of ports to be coupled to the segments. It includes two output ports per road. One of them allows to see if a train has arrived to the crossing. The other one informs that a train is leaving. It is built as: Out = { (i, 2n) / i ∈ N ∧ i ∈ [0,k] ∧ ((c1, c2, n, a, dir, max), δ, i) ∈ Railnets}.

The input coupling of this model is defined through the cell (0,0), which is connected with a station. Then, the behavior of this border cell is different. The following parameters are changed: η = 1; N = { (0,0) }

τ(N)	N
0	(0,0) = 1
1	portvalue(x-s-train) = 1
(0,0)	TRUE

The input port *x-s-train* is used to receive a new departure from the station.

A segment crossed by a railway should reflect the existence of the crossing. To define which cells in the road are affected, we compute col = ⌈δ/cell_size⌉, obtaining Cpn = { (i, col) / i ∈ [0,n-1] } ∪ { (i, col+1) / i ∈ [0,n-1] }, where n is the number of lanes in s. The Cell-DEVS model of the segment s (M_s) is modified by changing the behavior of the cells in the columns defined by Cpn. In these cells, a new output port will reflect the existence or absence of a train.

The cells in { (i, col) / i ∈ [0,n-1] } must reflect a stop in the straight movement when a train is crossing, as follows:

Delay: inertial; Delay length: dtrack().

τ (N)	N
0	(0,0) = 1 and (0,1) = 0 and portvalue(x-R-train) = 0
1	(0,-1) = 1 and (0,0) = 0
(0,0)	TRUE /*Otherwise: state unchanged */

The port *x-R-train* informs that a train is passing through the level crossing. If its state is 0, the rails are free. The delay function *dtrack* depends on the expected time for a train in the crossing. The inertial delay allows to define that the vehicles only advance into the crossing if it is empty for that period. Otherwise, the previous state is preempted, and the cars wait for the passing train.

The cells after the rails { (i, col+1) / i ∈ [0,n-1] }, are also affected because we should verify that no trains are crossing the railways before vehicles advance to them:

Delay: inertial; Delay length: dtrack

τ (N)	N
0	(0,0) = 1 and (0,1) = 0
1	(0,-1) = 1 and (0,0) =0 and portvalue(x-R-train) = 0
(0,0)	TRUE /*Otherwise: state unchanged */

Finally, we define the coupling between rails and segments. In this case, for every R ∈ Railnets, a station and a TrainTrack models are built (M_E and M_R). Every (s,δ, seq) ∈ R generates the following influences: I_E = { M_R }, I_R = { M_s }. The coupling between the cellular models is defined using the parameters col (earlier computed), and n (number of lanes of the model M_s), resulting in:

$Z_{E,R}$: y-R-train$_E$ → $X_{\eta+1}(0,0)_R$;

$Z_{R,s}$: $Y_{\eta+j}(0,seq)_R$ → $X_{\eta+3}(j\text{-}1,col)_s \forall j \in N, j \in [1, n]$;

$Z_{R,s}$: $Y_{\eta+n+j}(0,seq)_R$ → $X_{\eta+3}(j\text{-}1,col+1)_s \forall j \in N, j \in [1, n]$.

In our example, when we consider the coupling between the railway and the segment $I1$, we have that col = [90/7.5] = 12, and n= 2, then

$Z_{E,R}$: y-R-train$_E$ → $X_{\eta+1}(0,0)_R$;

$Z_{R,rI1}$: $Y_{\eta+j}(0,1)_R$ → $X_{\eta+3}(j\text{-}1,12)_{rI1} \forall j \in N, j \in [1, 2]$;

$Z_{R,rI1}$: $Y_{\eta+2+j}(0,1)_R$ → $X_{\eta+3}(j\text{-}1,13)_{rI1} \forall j \in N, j \in [1, 2]$.

3.6 Men at work

The *men at work* construction is translated as different models according with the number of lanes, the size and position of the jobsite. Due to these factors, different border conditions must be used in each case. A Cell-DEVS is used to represent the working site. Different functions are used to represent the cells in the rhombus and those before it. The cells

in the jobsite will always have a 0 value. The remaining cells behave as standard segments.

As the other cars should avoid advancing into the jobsite, we must group the cells before it according to the movements that are allowed. As it is shown in the following figure, vehicles can move to the left diagonal (LD), the right diagonal (RD), both (2D), or advance in a straight line. Therefore, different rules have been defined for each case.

The presence of men at work produces that the vehicles cannot advance in a road. These traffic obstructions are specified as:

$$Jobsite = \{(s, ni, \delta, \#n)/s \in Segments \wedge s = (c1, c2, n, a, dir, max)$$
$$\wedge ni \in [0, n-1] \wedge \delta \in \boldsymbol{N} \wedge \#n \in [1, n+1-ni] \wedge \#n \equiv 1 \ mod \ 2\}$$

Every tuple o = (s, ni, δ, #n) defines the segment where the construction is being done. It includes the first lane affected, the distance between the center of the jobsite and the beginning of the section, and the number of lanes occupied, as showed in Figure 15. These values are used to define a rhombus over the segment where the vehicles cannot advance. The cars arriving to the jobsite must deviate.

A group of different models has been defined, according with the number of lanes, the size and position of the jobsite. Different border conditions were defined in every case. The basic behavior for these models was defined as a Cell-DEVS representing the working site. We use different behavior for the cell in the rhombus and those before it. The first ones will always have a 0 value. The other ones should avoid to advance into the jobsite, as in Figure 16.

For instance, the cells whose movement to both diagonals is allowed are defined by:

$C_{ij} = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$

I = < η, P^x, P^y >, with η = 9; $\boldsymbol{X} = \boldsymbol{Y} = \{0, 1\}$; **N** = { (0,0), (0,1), (1,0), (1,1), (0,-1), (1,-1), (-1,1), (-1,-1), (-1,0) };

S:

$$s = \begin{cases} 1 \text{ if there is a vehicle in the cell;} \\ 0 \text{ otherwise.} \end{cases}$$

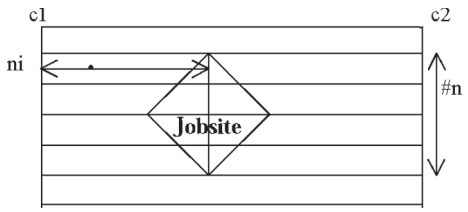

FIGURE 15
Segment with men at work.

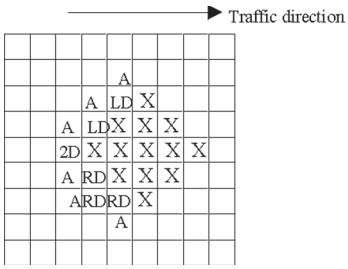

FIGURE 16
Jobsite cells.

delay = transport; **d** = speed(max);
λ, δint and δext are defined by Cell-DEVS;
τ is defined as:

τ (N)	N
1	(0,-1)=1 and (0,0)=0 or (0,0)=0 and (0,-1)=0 and (-1,-1)=1 and (-1,0)=1 or (0,0)=0 and (0,-1)=0 and (1,-1)=1 and (1,0) = 1
0	(0,0) = 1 and (0,1) = 0) or (0,0) = 1 and (1,1) = 0 and (1,0) = 0 or (0,0) = 0 and (0,-1) = 0 and (-1,-1) = 1
(0,0)	TRUE /*Otherwise: state unchanged */

As we can see, the first rule represents the arrival of a car. If the previous cell is empty, the car is inserted in the present cell. Otherwise, we check if the cell to the SW is busy, and the cell in front of it too. Then, we check the case when the cell to the NW is busy and the cell in front of it too. The second rule represents that a car abandons the cell with a straight movement. If this in not possible, we check to see if the cell in front is in the jobsite and the cell to the NE is free to move. If this does not occur, the cell to the SE should be able to move.

3.7 Traffic signs

A construction to specify speed traffic signs was defined. This construction is defined by:

$$TrafficSigns = \{(s, t, \delta)/s \in Segments \wedge \delta \in \mathbf{N} \wedge t \in \{bump, depression, school, pedestriancrossing, others\}\}$$

Each tuple identifies the segment, the kind of traffic sign, and the distance up to it in the section, showed in Figure 17.

The cells where a traffic sign have influence increase the length of the delay. In this way, a slower speed of the cars is represented.

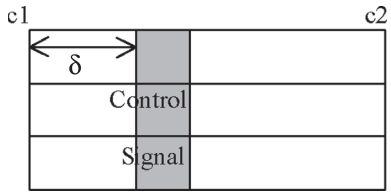

FIGURE 17
Definition of a traffic sign.

An extension of this construction allows us to define Potholes, whose size is one cell, depicted in Figure 18. The behavior of this construction is to produce a speed reduction of the vehicles passing through these cells. They are defined by:

$\text{Pothole}_S = \{ (s, n1, \delta) / s \in \text{Segments} \wedge s = (c1, c2, n, a, dir, max) \wedge n1 \in [0, n-1] \wedge \delta \in N \}$

$\text{Pothole}_C = \{ c / c \in \text{Crossings} \}$

A segment with a traffic sign is as in the previous sections, but delay is increased for those cells influenced by the signal. In this way, a slower speed of the cars is represented. For instance, in the example introduced on Section 2, $(rA, school, 20) \in TrafficSigns$ and $(rA, 0, 10) \in Pothole_s$, the rules of segment *rA* are not changed, but the delay of the cells (0,2), (0,3) and (0,10) of *rA* increases according to the phenomenon represented.

3.8 Parking

The following construction allows the definition of parking cars at the border cells:

$Parking = \{ (s, n1) / s \in Segments \wedge n1 \in \{0,1\} \wedge s = (c1, c2, n, a, dir, max) \wedge n > 1 \}$

Every pair (s, n1) identifies the segment and the lane where car parking is allowed. If n1 = 0, the cars park on the left lane. If n1 = 1, the right lane is used (lane n-1). When a car arrives into a parking lane, it will stop on that lane during a given amount of time, as showed in Figure 19. This is modeled using a long transport delay (representing several minutes or hours).

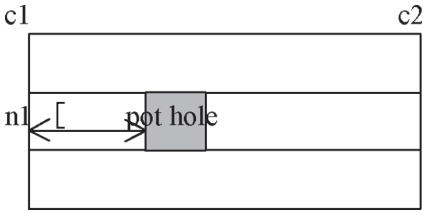

FIGURE 18
Definition of a Pothole.

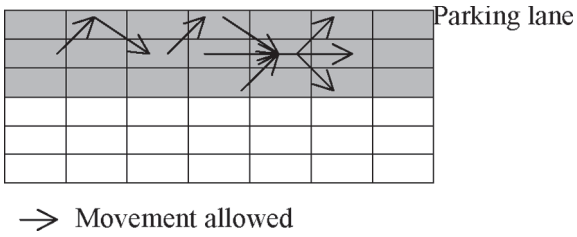

FIGURE 19
Parking segments.

Given a lane $(s, n1)$ to be used for parking cars, and being Ms is the model specified for the segment s, the lanes where cars are being parked and their contiguous lanes are defined by:

Cpark = { (0, i) / 0 ≤ i ≤ t2-1 } if n1 = 0, or Cpark = { (t1-1, i) / 0 ≤ i ≤ t2-1 } if n1 = 1;

Ccont1 = { (1, i) / 0 ≤ i ≤ t2-1 } if n1 = 0, or Ccont1 = { (t1-2, i) / 0 ≤ i ≤ t2-1 } if n1 = 1; and

Ccont2 = { (2, i) / 0 ≤ i ≤ t2-1 } if n1 = 0, or Ccont2 = { (t1-3, i) / 0 ≤ i ≤ t2-1 } if n1 = 1.

Here, {t1, t2} define the dimension of the cell space (rows, columns). As previously explained, n1=0 means that parking on the left is allowed. Instead, n1=1 means that parking should be done on the right. These cells have a different behavior than those in the rest of the segment. The parking cells (*Cpark*) do not allow that vehicles advance: they can only receive cars from the first contiguous lane. Therefore, the rules modeling straight movements were eliminated from these cells.

τ (N)	N
1	(0,0) = 0 and (-1,-1) = 1 and (-1,0) = 1
1	(0,0) = 0 and (1,-1) = 1 and (1,0) = 1
1	(0,0) = 0 and (1,-1) = 1 and (1,0) = 1 and ((2,-1) = 1 or (2,0) = 1)

In this case, the transport delay used for the parked cars is a long delay representing the time in which a car is normally parked. The remaining rules for these cells do not change. The cells in the set *Ccont1*, will behave different depending on the parameter *n1*. If *n1* = 0, the cars are parked on the lane 0, and the cells Ccont1 = { (1, i) / 0 ≤ i ≤ t2-1 } behave different from the rest. In these cells, the rules checking if there is a moving car in the left lanes have been eliminated. When the origin cell receives a car from the parking, it does not check if it is advancing straight in its own lane, because this movement is not allowed. Likewise, when the origin cell

sends a car to the parking lane, it does not check if there is another car behind the parking space.

τ (N)	N
0	(0,0) = 1 and (1,1) = 0
1	(0,0) = 0 and (1,-1) = 1 and (0,-1) = 0

Otherwise, if n1 = 1 (that is, the cars park on the k-eth lane), the cells to the left of the parking are defined by Ccont1 = { (t1-2, i) / $0 \leq i \leq$ t2-1 }. The rules for these cells are modified symmetrically to the previous ones.

τ (N)	N
0	(0,0) = 1 and (-1,1) = 0
1	(0,0) = 0 and (-1,-1) = 1 and (0,-1) = 0

Finally, the second lane contiguous to the parking should be changed. Depending on the parameter n1, the cells to be changed are Ccont2 = { (2, i) / $0 \leq i \leq$ t2-1 } (if n1 = 0) or Ccont2 = { (t1-3, i) / $0 \leq i \leq$ t2-1 } (n1 = 1). Here, we avoid analyzing the lanes where cars are parked to see if these cars are moving.

3.9 Experimental framework

The constructions defined in the previous sections are connected to an experimental framework defined as a set of segments providing inputs and outputs to the city section to be studied. They are defined as:

$$InputSegments = \{s/s = (p1, p2, n, a, dir, max) \wedge s \in Segments \wedge [(dir = 0 \wedge (\exists v \in \boldsymbol{N} : (p2, v) \in Crossings)) \vee (dir = 1 \wedge (\exists v \in \boldsymbol{N} : (p1, v) \in Crossings))]\}$$

$$OutputSegments = \{s/s = (p1, p2, n, a, dir, max) \wedge s \in Segments \wedge [(dir = 0 \wedge (\exists v \in \boldsymbol{N} : (p1, v) \in Crossings)) \vee (dir = 1 \wedge (\exists v \in \boldsymbol{N} : (p2, v) \in Crossings))]\}$$

For each $s \in$ *InputSegments*, a DEVS model is defined. Its goal is to generate vehicles that are inserted in the city section to be simulated, and it can be tuned to reflect different traffic conditions. These models are defined by:

$$Generator(\#c) =< I, X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

Instead, the segments $s \in OutputSegments$ define a DEVS model devoted to consume vehicles and compute statistics:

$$Output(\#c) =< I, X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, D >$$

In both cases, *#c* represents the number of lanes of the segment s to which they are connected. In Figure 20 we present a sketch of the behavior of these models.

4 VEHICLE ROUTING AND CONGESTION MONITORING

The original definitions used for ATLAS models were based on random routing: every time a car arrives to a crossing, the following route is chosen at random. This was expanded to include a way to define the routing information for the vehicles. We have used an approach based in Origin/Destination (O/D) matrixes. This tool provides information about routes and transportation between different zones or regions. There are several methods to estimate O/D matrixes. Using the definition of a region (represented as a directed graph), the methods use the available information (traffic flow, delays existing in each link of the graph). We will suppose

Generator(#c)

```
δext(s,e,x) {
   passivate (); /* It is never activated */
}

η(s) {
   send car to y-s-carlane;
/*car contains a 1 if a car is sent to the lane, or a 0 if no car
wants to enter the lane */
}

δint(s,e,x) {
 case phase
   active:
     case car:
          0: car = 1;
             σ = tgen(); /* It schedules a car after tgen */
             phase = active;
             lane = (lane + 1) mod #c
          1: car = 0;
             σ = 0; /* Instantaneous transition scheduled */
             phase = active;
     end case
   passive:
          /* Nothing to be done */
 end case
}
```

Output(#c)

```
δext(s,e,x) {
  When x arrives to the port x-s-carj  (0   j < #c)
          Record statistics;
}

η(s) {
    /* Nothing to be done */
}

δint(s,e,x) {
    /* Nothing to be done */
}
```

FIGURE 20
Definition of the DEVS models of the experimental framework.

that an O/D matrix has been provided, and it will be used by the simulator to make decisions related to vehicle routing (Díaz et al. 2000).

There are several ways of implementing O/D matrixes, and we have chosen an approach based on a road table. Each register in this table will specify a road connecting a pair of origin/destinations, and the time a car spend in that road. Different tables can be used according to different parameters (for instance, date, time, type of vehicles). Therefore, the table will have the following structure:

Time Vehicle type { *ID Origin-node Destination-node* {*link1 link2 ... linkn*} *Travel-time* }

The structure of the O/D matrix and the function used to make the routing decisions can be changed without affecting the simulation models. In this way, both problems can be treated independently. In a first stage the city shape is defined using ATLAS. Then, a directed graph can be built based on the segment and crossing identifications. Using this graph, an O/D matrix can be built. The simulation models devoted to represent routing use a function that queries the O/D matrix and provides an answer.

Using this approach, we define a static route for each car, which will not be changing during all of the simulation. Using the segment/crossing definition, we can build a graph representing the structure of the city section to be simulated. Using this base, we built an origin/destination matrix. For each pair origin/destination in the matrix, we build a route using Dijkstra's algorithm for shortest paths in a graph. Using this information, we build a complete acyclic route from an input crossing to an output crossing.

Every car is initialized with the route to be used. The original definitions of ATLAS constructions were modified to allow this behavior. The first modification includes a unique identification for each segment/crossing that will be used with routing purposes. Besides the original port definition (*car* and *room*, indicating that there is a car or space in the origin cell), the coupled models defined for each segment/crossing include now a new port devoted to transfer the routing path (*path* in the following figure).

The behavior for the border cells was changed accordingly, as showed in Figure 21. This new behavior was also extended for models from 1 to 5 or more lanes (each of them must be defined in a different way due to the definition of the border cells for each of the models). After changing the segment definition, the crossing behavior was also modified. In this case, the crossings will be in charge of routing using the O/D matrixes. A crossing is defined as a Cell-DEVS coupled model

CrossingOD(k, In, Out, maxc) = < *Xlist, Ylist, I, X, Y, n,* $\{t_1,...,t_n\}$, η, *N, C, B, Z, select* >

Ylist = **Xlist** = { (0,i) / $0 \leq i < k$}; **I** = <P^x, P^y >

P^x = {<$X_{\eta+1}$(0,i), binary>, <$X_{\eta+2}$(0,i), binary>, <$X_{\eta+3}$(0,i), *N* >/ $0 \leq i < k$ }

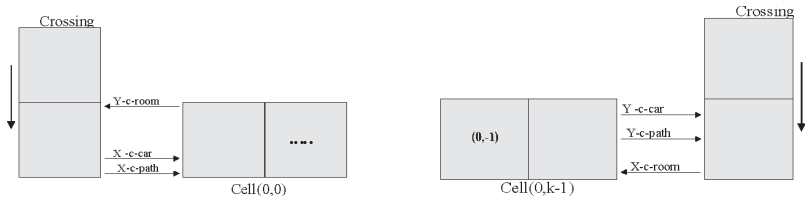

FIGURE 21
Coupling of the Border Cells.

$P^y = \{<Y_{\eta+1}(0,i)$, binary$>$, $<Y_{\eta+2}(0,i)$, binary$>$, $<Y_{\eta+3}(0,i)$, $N > / 0 \leq i < k \}$

Each cell in the crossing is defined by:

C_{0j} *(crossing-No)* $= < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \delta, D >$

$\mathbf{I} = < \eta, P^x, P^y >$, η g= 3, $P^x = \{ (X_1$, Record), $(X_2$, Record), $(X_3$, Record) $\}$, $P^y = \{ (Y_1$, Record), $(Y_2$, Record), $(Y_3$, Record) $\}$; $\boldsymbol{X}, \boldsymbol{Y} \in$ N,

S = (Car, crossing, path, segment-no), where

$$Car = \begin{cases} 1 \text{ there is a vehicle} \\ 0 \text{ otherwise.} \end{cases}$$

crossing $\in$ N: unique identifier of the crossing;

$$path = \begin{cases} \{t_1.t_2.\ldots.t_n\} \text{ where } ti \in N \wedge (\forall i(\exists r \in \text{Segments}/ \\ \qquad\qquad \text{segment} - \text{no}(r) = ti)) \\ 0 \qquad\qquad \text{otherwise.} \end{cases}$$

Segment-no $\in$ N: identifier of the segment to which the cell is connected to.

N = { (0,-1), (0,0), (0,1) }; **delay** = transport; **d** = (speed(maxc))

The τ function is in charge of defining the car behavior in the cell, according to the new routing scheme. Different behavior is defined for the input and output cells, using the O/D matrixes.

We now show the definition of a simple example using the new definitions. The example was implemented using the CD++ tool (Wainer, 2002), and it is depicted in Figure 22. The model consists of 5 segments and 4 crossings, and will describe the behavior of traffic using the routing scheme defined previously. The path is specified by a real number with the following format: d.ddddd where 1=< d <=9 is the identifier of a segment.

This city section is composed of 5 segments and 4 crossings interconnected. Different cars arrive through the segment 1, and all of the cars will follow the same path (segment2-segment5). The segment 2 includes a pothole delaying the advance of cars. The definition of the congestion function considered that a segment with 2 or more cars is congested. The goal of this example is to show how the dynamic routing is achieved.

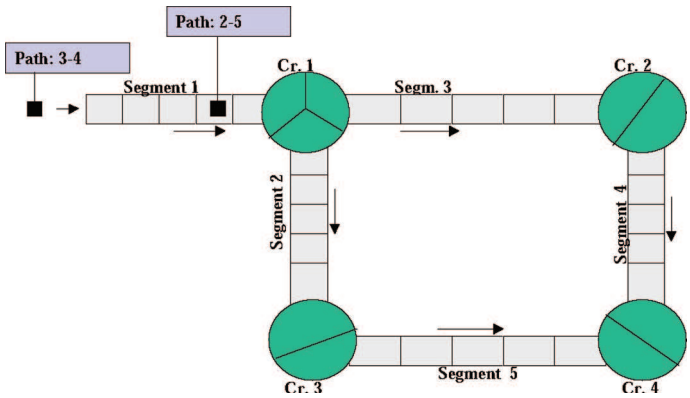

FIGURE 22
Sample city section.

Figure 23 shows the simulation result in the segment 1. The first row represents the state variable showing the presence of a car. The second line represents the path to be followed by the car. The first car will take the path 2-5, whereas the second will go through the path 3-4. This behavior can be found in figure 7.

As we can see in Figure 24, the crossing shows if there is a car in the cell (row 0), the path to be followed (row 1), the identification of the crossing (row2) and the segments to which the crossing is connected to (row 3). For instance, we see that the crossing contains a 2 in the cell (3,1), meaning that the cell is connected to the segment number 2. The crossing defines the cell used to route a car, and after done, the first element in the route is deleted.

Traffic flow rate in a city section can influence the vehicle movement and the decisions taken by the drivers. Most existing modeling approaches based on cell spaces do not consider the information related to congestion,

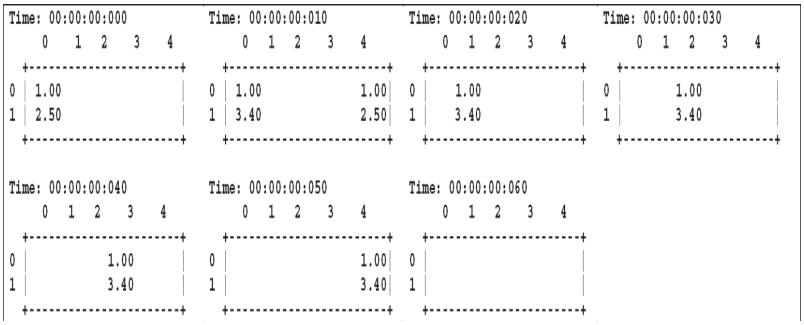

FIGURE 23
Execution trace of Segment 1.

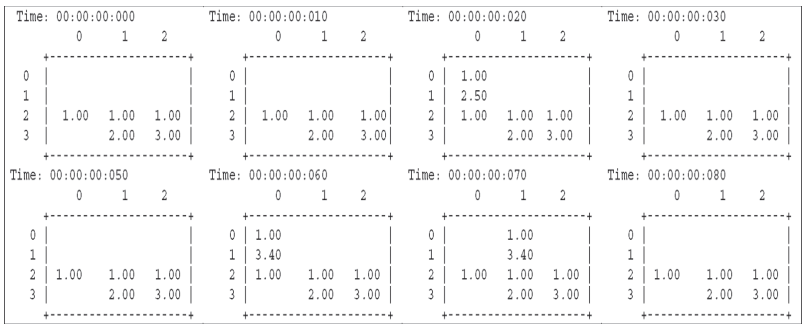

FIGURE 24
Execution trace of the Crossing 1.

or provides mechanisms to reproduce vehicle routing. We have included a construction to represent this behavior. The models use a function that queries the O/D matrix and provides a route to be followed by a vehicle. Then, a measure of traffic congestion can be used. Based on this information and using the O/D matrixes, the cars can change their original routes. A new DEVS model, devoted to monitor congestion, was added. Now, every segment is provided with a controller to measure the number of cars. This model is defined as:

$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$

$X =\{<$ x-r-carIn, N$>, <$ x-r-carOut, N$>\}$; $Y =\{<$ y-r-weight, N$>\}$

S = k $\in$ N representing the number of cars in the segment under consideration,

$$\begin{aligned}\delta_{\text{ext}}()\{\quad & \textbf{when}(x - r - carIn = 1)k = k + 1;\\ & \textbf{when}(x - r - carOut = 1)k = k - -1;\\ & passivate;\\ \}\end{aligned}$$

λ () { send k through the port y-r-weight }

Once the DEVS congestion controllers were defined, the border cells of the Cell-DEVS representing the segments were changed to transmit information about the cars arriving or leaving a segment. New input/output ports were added to transmit this data to the coupled models corresponding to the crossings, depicted in the figure 25:

The border cells must inform the number of cars entering and leaving the segment to the DEVS congestion monitor. For instance, for one-lane segments, this specification is now translated into a Cell-DEVS defined by:

$$C_0(segment\_no) =< I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

I = $< \eta$, P^x, $P^y >$, with η g= 3; d = speed(max); N = { (0,-1), (0,0), (0,1) }; delay = transport

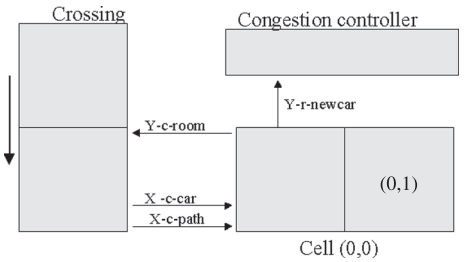

FIGURE 25
Coupling scheme of congestion controllers, crossings and segments.

$X, Y \in$ N; S : {s, phase, σqueue, σ}, with s = (destination, path)

$$destination \in \mathrm{N} = \begin{cases} \neq 0 \text{ if there is a vehicle in the cell} \\ 0 \text{ otherwise.} \end{cases}$$

$$path = \begin{cases} \{t_{1..}.t_n\}\ \mathrm{ti} \in \mathrm{N} \wedge (\forall \mathrm{i}(\exists \mathrm{r} \in \mathrm{Segments}/ \\ \qquad \mathrm{Segment\_no(r)} = \mathrm{ti})) \\ 0 \qquad \text{otherwise.} \end{cases}$$

λ, δ_{int} y δ_{ext} are defined by Cell-DEVS with transport delays.

τ: S x N $\rightarrow$ S. The behavior of the local computing function can be roughly defined by:

τ (N)	N
Dest = Dest(0,-1); Path = Path(0,-1)	Dest(0,-1) != 0 and Dest(0,0) = 0
Dest = 0; Path = 0	Dest(0,0) != 0 and Dest(0,1) = 0

In this case, the first rule represents a vehicle arriving to the cell, coming from the previous cell. The second rule represents the advance of the vehicle to the following cell. As we can see, the identifier of the destination cell represents the vehicle, and the path of the vehicle is transferred between cells.

After defining the behavior for the controller and the segments (with 1 to 5 lanes), a new coupled model for the crossings was created.

Then, the rules used for defining the crossing behavior were modified. Now, every crossing will receive information from the congestion controller, and, based on the availability of paths to arrive to the same destination and the congestion information, a routing decision is taken. The models now include the definition of routing mechanisms associated with each cell, defined as:

$$C_j(crossing\_no) =< I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, ta >$$

I = < η, P^x, P^y >, with η = 3, P^x = { <x-t-destinat, Record>, <x-c-congestion, Record>, <x-t-path, Record> }, P^y = { <y-t-room, Record>, <y-c-vehicle, Record>, <y-c-path, Record> }; $X, Y \in N$;

S: {s, phase, σqueue, σ}, with s = (destination, path, crossing_no, segment_no), with

⌈ !=0 if there is a vehicle (representing the destination crossing).

$$\text{destination} = \begin{cases} \\ 0 \text{ otherwise.} \end{cases}$$

$$\text{path} = \begin{cases} \{t_1.t_{2...}t_n\}, \text{ti} \in N \wedge (\forall i(\exists r \in \text{Segments} \\ \qquad /\text{segment\_no}(r) = \text{ti})) \\ 0 \qquad \text{otherwise.} \end{cases}$$

crossing_no $\in$ N: crossing identifier;

segment_no $\in$ N: identifier of the segment to which the output cell of the crossing is connected.

N = { (0,-1), (0,0), (0,1) }; delay = transport; d = speed(maxc); λ, δ_{int} and δ_{ext} are defined by Cell-DEVS with transport delays. ; τ: S x N $\rightarrow$ S. The behavior of the local computing function can be defined by:

τ (N)	N
Dest= Dest(0,-1); Path= Path(0,-1) Crs_no= crs_no(0,0) Seg_no= seg_no(0,0)	Dest(0,0)= 0 and Dest(0,-1) != 0 Send(1, y-t-room)
Dest= (x-t-dest); Path= (x-t-Path) Crs_no= crs_no(0,0) Seg_no= Seg_no(0,0)	Dest(0,0)=0 and Dest(0,-1)=0 and (x-t-dest)!=0 and (x-t-dest)!= crs_no(0,0) and !Congestion((x-c-congest, next-seg(path(0,-1)), seg_no(0,0))) Send(1, y-t-room)
Dest= (x-t-dest); Seg_no= Seg_no(0,0) Path=New_Path(Crs_no(0,0), Dest(0,-1), Path(0,-1));	Dest(0,0)=0 and Dest(0,-1)=0 and (x-t-dest)!=0 and (x-t-dest) != crs_no(0,0) and Congestion((x-c-congest,

Crs_no= crs_no(0,0)	next-segment(path(0,-1), seg_no(0,0))) Send(1, y-t-room)
Dest= 0; Path= 0 Crs_no=crs_no(0,0); Seg_no= Seg_no(0,0)	Dest(0,0)= 0 and Dest(0,-1)= 0 and (x-t-dest) != 0 and (x-t-dest)= crs_no(0,0); Send(0, y-t-room)

In this case, the first rule introduced represents the arrival of a vehicle to a cell that was in the crossing and preserves the original path. The second rule represents the input of a vehicle to the crossing that has not arrived to the destination, conserving the original route. The following rule represents the input of a vehicle that changes the path due to congestion in the area. The fourth rule eliminates a vehicle that has arrived to the destination. The crossing number uniquely defines the crossing where this cell is defined. In this way, an O/D matrix can be used, and the crossing identified to permit different paths to be taken. The cell state represents the existence of a vehicle, the path to be followed, and the crossing identifier. Using this information, every vehicle arriving to the crossing can be routed according to the congestion information. When a vehicle arrives to a crossing, it will be sent to the input cell, which will be in charge of deciding the vehicle routing. The input cells will use the congestion information sent by the congestion monitors, as presented in Figure 26.

Figure 27 presents the execution of the model introduced in Figure 22. The segments were implemented as Cell-DEVS models using two state variables. The first variable defines the vehicle destination (or 0 if the cell

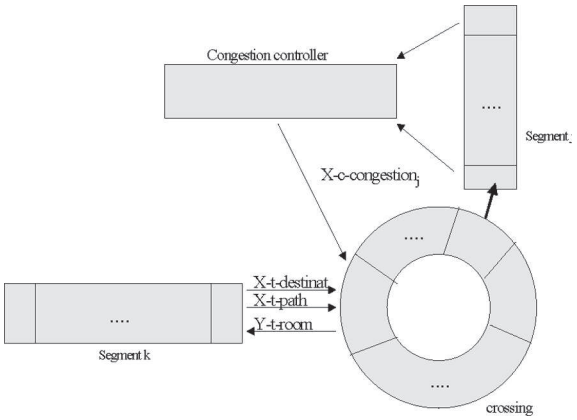

FIGURE 26
Coupling scheme of output cells of segments, crossings, and congestion controllers.

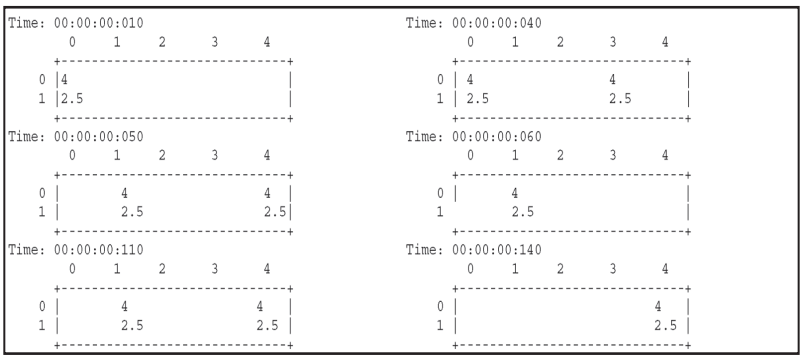

FIGURE 27
Execution results in Segment 1.

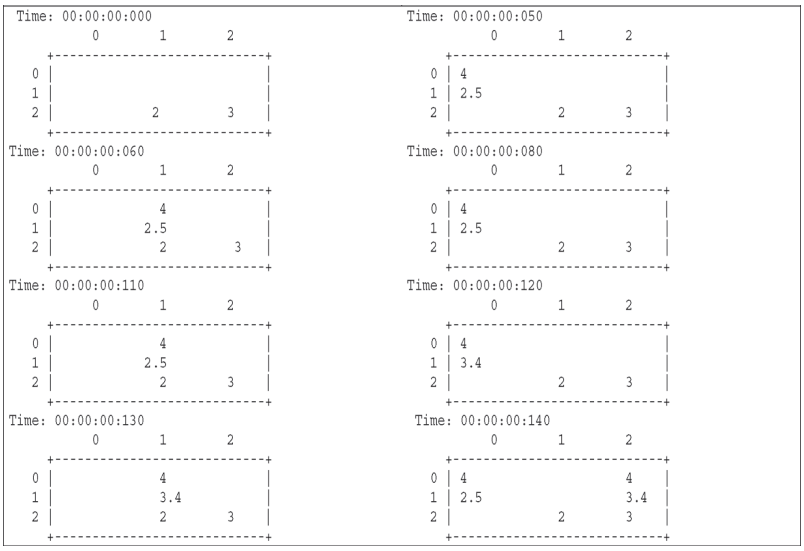

FIGURE 28
Execution results in Crossing 1.

is empty), and the second state variable include a definition of the path to be followed by the vehicle. The following figure shows the execution results for the segment 1:

In this example we see that in the simulated time 00:00:010, 00:00:040, 00:00:070 and 00:00:100 new cars have arrived to the cell (0,0). Every vehicle will follow the route segment 2-segment 5. When the cars finish traversing the segment, they will be routed to the crossing 1.

This crossing construction implements the dynamic routing techniques previously explained. The cell 0 of the crossing is an input cell, while the remaining two are used for outputs. The following figure shows the values of the different state variables used in each cell of the crossing. The state variables showed in the lines 0 and 1 represent the vehicle information used for the segments (destination and route). Line 2 represents the segments to which the output cells are connected.

The first two cars arriving to the crossing (in 00:00:050 and 00:00:080) keep the original path (segment 2-segment 5). As the cell 1 in the crossing is connected to the segment 2, the vehicle is sent to the crossing through this cell.

The last two cars arriving to the crossing (at 00:00:110 and 00:00:140) must take a new path, because the congestion function for the segment 2 returns a value representing that the segment is congested. Therefore, the cars ask to the O/D matrix for a new path, and the model returns the path segment 3-segment 4. Then, they leave the crossing through the cell 2, connected to the segment 3, following the rules defined in the previous sections.

5 CONCLUSION

ATLAS is a traffic modeling language that permits defining city sections, with a static view of including different components. This approach provides an application-oriented specification language, which allows the definition of complex traffic behavior using simple rules for a modeler. The models are formally specified, avoiding a high number of errors in the application, thus reducing the problem solving time.

The high level specification of the problem to be modeled reduces the developing efforts, as the techniques presented permit to automatically build the structure for coupled models, and to generate rules for atomic models. In this way, changes in the system specification can be done in a simple fashion, without spending time in coding or testing every proposed solution to existing problems. In this way, a traffic analyzer can focus in the problem solving task, avoiding implementation or low level details.

The constructions are mapped into DEVS and Cell-DEVS. These translated models are formally specified, and its correctness was proved, avoiding errors in their definition. Using this approach we could obtain:

- Efficiency: by describing a high level specification of the problem to be modeled, we have reduced the effort needed in developing the application. The models execute using a discrete-event approach, which, as showed in (Zeigler et al. 1997), (Wainer and Giambiasi 2002)

and (Wainer 2006), provide higher precision and speedups than the discrete time approaches. Likewise, the proposal automatically builds the structure for coupled models, generates rules for atomic models. In this way, changes in the system specification can be done in a simple fashion, without spending time in coding or testing every proposed solution to existing problems.

- Adaptation: new rules can be easily incorporated, as we showed with different examples here (traffic lights, truck behavior, potholes, etc).
- Abstraction: the specifications were translated into executable models. In this way, a traffic analyzer can focus in the problem solving task, avoiding implementation or low level details.

We have built a compiler for the specification language (introduced in (Wainer 2005)) based on these formal specifications. In addition, a graphical user interface allows easy definition of the models (Wainer et al. 2004). The GUI validates the static model based on information given by the map and the constructions used. Finally, efficient execution of the models is being considered by means of parallel execution of Cell-DEVS.

ACKNOWLEDGEMENTS

Different students collaborated in different aspects of this research, mainly Alejandra Díaz, and Verónica Vázquez. The research was funded by NSERC, the Canadian Foundation for Innovation, and the Ontario Innovation Fund.

REFERENCES

[1] Balmer, B., Cetin, N., Nagel, K., Raney, B. "Towards Truly Agent-Based Traffic and Mobility Simulations". Autonomous Agents and Multi-Agent Systems conference. New York, NY. USA. 2004.

[2] Chen, O., Ben-Akiva, M. "Game Theoretic formulation of the interaction between dynamic traffic control and dynamic traffic assignment". Technical Report, ITS, M.I.T. 1998.

[3] Chopard, B., Droz, M. "Cellular automata modeling of physical systems". Cambridge University Press. 1998.

[4] Chopard, B., Dupuis, A., Luthi, P. "A CA Model for Urban Traffic and its applications to the city of Genoa". *Proceedings of Traffic and Granular Flow*. 1997.

[5] Chopard, B., Queloz, P. A., Luthi, P. "CA Model of Car Traffic in two-dimensional street networks". J. Phys. A, vol. 29,1996.

[6] Chopard, B., Queloz, P. A., Luthi, P. "Traffic Models of a 2D road network". *Proc. of the 3rd CM users' Meeting*. Parma. ITALY. 1995.

[7] Davidson, A., Wainer, G. "Definition of a specification language for urban traffic simulation using the Cell-DEVS formalism" (in Spanish). *Technical Report 99-003, Departamento de Computación, FCEN/UBA*. 1999.

[8] Davidson, A., Wainer, G. a. "Specifying traffic signs in traffic models". A. Davidson, G. Wainer. In *Proceedings of AI, Simulation and Planning in High Autonomous Systems, AIS'2000*. Tucson, Arizona. U.S.A. 2000.

[9] Davidson, A., Wainer, G. b. "Specifying truck movement in traffic models using Cell-DEVS". In *Proceedings of the 33rd Annual Simulation Symposium*. IEEE Press. Washington, D.C. U.S.A. 2000.

[10] Díaz, A, Vázquez, V., Wainer, G. "Application of the ATLAS language in models of urban traffic". In *Proceedings of 34th IEEE/SCS Annual Simulation Symposium*. Seattle, WA. U.S.A. 2001.

[11] Esser, J., Schreckenberg, M. "Microscopic simulation of urban traffic based on CA". Intl. Journal of Modern Physics C8,1025-1036. 1997.

[12] Lee, J., Chi, S. "Using Symbolic DEVS Simulation to Generate Optimal Traffic Signal Timings". SIMULATION. 81: 153-170. 2005.

[13] Maniezzo, V. "CA and roundabout traffic simulation". Proceedings of Sixth International conference on Cellular Automata for Research and Industry. Amsterdam, Netherlands. LNCS 3305. 2004.

[14] Marinosson, S. "Simulation of the Autobahn Traffic in North Rhine-Westphalia". Proceedings of 5th International Conference on Cellular Automata for Research and Industry. Geneva, Switzerland. LNCS 2493. 2002.

[15] Nagel, K., Esser, J., Rickert, M. "Large-scale traffic simulations for transport planning". Annual Reviews of Computational Physics VII (World Scientific, Singapore, 2000). [22], 2000, 151-202.

[16] Nagel, K. "Cellular Automata models for transportation applications". Proceedings of 5th International Conference on Cellular Automata for Research and Industry. Geneva, Switzerland. LNCS 2493. 2002.

[17] Nagel, K., Stretz, P., Pieck, M., Leckey, S., Donnelly, T., Barret, C. "TRANSIMS traffic flow characteristics". *Transportation Research Board, 77th Annual Meeting*. Washington, D. C. 1998.

[18] Rickert, M. , Nagel, K. , Schreckenberg, M., Latour, A. "Two Lane Traffic Simulations using CA". Physica A 231, 1996, 534-550.

[19] Rodríguez Zamora, R. "Using de Bruijn diagrams to analyze 1d CA traffic models". Proceedings of Sixth International conference on Cellular Automata for Research and Industry. Amsterdam, Netherlands. LNCS 3305. 2004.

[20] Sadoun, B. "An Efficient Simulation Methodology for the Design of Traffic Lights at Intersections in Urban Areas". SIMULATION, 79: 243 - 251. 2003.

[21] Schmidt, M. "Decomposition of a traffic flow model for a parallel simulation". In *Proceedings of AI, Simulation and Planning in High Autonomous Systems, AIS'2000*. Tucson, Arizona. U.S.A. 2000.

[22] Tolba, C., Lefebvre, D., Thomas, P., El Moudni, A. "Continuous and timed Petri nets for the macroscopic and microscopic traffic flow modeling". Simulation Modeling Practice and Theory, Volume 13, Issue 5, July 2005, Pages 407-436.

[23] Treiber, M., Hennecke, A., Helbing, D. "Congested Traffic States in Empirical Observations and Microscopic Simulations". Physical Review E 62, 2000, 1805-1824.

[24] Wagner, P., Nagel, K., Wolf, P. "Realistic Multi-line traffic rules for cellular automaton". Physica A, 234:687, 1997

[25] Wainer, G., Borho, S., Pittner J. "Defining and visualizing models of urban traffic". In Proceedings of the SCS 1st Mediterranean Multiconference on Modeling and Simulation. Genoa, Italy. 2004.

[26] Wainer, G. "CD++: a toolkit to define discrete-event models". *Software, Practice and Experience.* Wiley. Vol. 32, No.3. November 2002. pp. 1261-1306.

[27] Wainer, G. "ATLAS: a language to specify traffic models using Cell-DEVS". In *Simulation Modelling Practice and Theory.* Vol. 14, pp. 313–337 (2006).

[28] Wainer, G., Giambiasi, N. "N-Dimensional Cell-DEVS". G. Wainer, N. Giambiasi. In *Discrete Events Systems: Theory and Applications*, Kluwer. Vol. 12, No. 1. pp. 135-157. 2002.

[29] Wolfram, S. "A New Kind of Science". Wolfram Media. 2002.

[30] Zeigler, B., Kim, T., Praehofer, H. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". Academic Press. 2000.

[31] Zeigler, B., Moon, Y., Kim, D., Ball, G. "The DEVS environment for high-performance modeling and simulation". IEEE Computational Science and Engineering , Vol. 4, No. 3. 1997.