Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Simulation Modelling Practice and Theory 14 (2006) 313–337
SIMULATION MODELLING PRACTICE AND THEORY
www.elsevier.com/locate/simpat

ATLAS: A language to specify traffic models using Cell-DEVS

Gabriel Wainer *

Carleton University, Department of Systems and Computer Engineering, 1125 Colonel By Drive, Ottawa, Ont., Canada K1S 5B6

Received 14 July 2004; received in revised form 7 July 2005; accepted 20 July 2005
Available online 22 September 2005

Abstract

The ATLAS specification language is devoted to build models of city sections using micro-simulation. The basic language constructions allow defining a static topology of the section to be studied. The dynamic behavior of the section can be modified by including traffic lights, traffic signs, etc. Once the urban section is outlined, models are converted into cell spaces and the traffic flow is automatically set up. Language constructions were mapped into DEVS and Cell-DEVS models that can be easily executed with a simulation tool. The models were formally specified, improving the verification of the language. Thanks to this formal approach, we ensure that the simulations are correct, avoiding a high number of errors in the developed application, and as the modelers can focus in the problem to solve, development times can be reduced.

Keywords: Traffic simulation; DEVS; Cellular Automata; Cell-DEVS; Simulation environments

* Tel.: +1 613 520 2600.
E-mail address: gwainer@sce.carleton.ca

1569-190X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2005.07.002

1. Introduction

Traffic flow in modern urban centers is a complex phenomenon that cannot be studied analytically; instead, computer simulation has gained acceptance for urban traffic analysis and control [20]. Simulators have been used to improve traffic, measure the consequences of collisions, avoid pollution, etc. Existing traffic simulators can be classified according to the level of detail they allow. *Microscopic* traffic models describe both the system entities and their interactions at a high level of detail. For example, a lane change could consider the nearby cars, as well as detailed driver decisions. *Mesoscopic* models also represent the entity details, but their interaction is analyzed with less detail. For example, the lane changes could be modeled according with relative lane densities. Finally, *macroscopic* models describe entities and their activities and interactions at a low level of detail [26,35]. For instance, lane changes can be approximated, or not represented at all. Instead, flow rate, density and speed are analyzed [23]. Each of these approaches has advantages and drawbacks. Macroscopic models are effective when global traffic flow features must be analyzed, allowing planning in the large, solving environmental issues, global traffic light coordination strategies, etc. In the other extreme, microscopic models enable detailed behavior analysis, such as local delays due to traffic conditions or, heavy traffic, local traffic light coordination or changing of lane directions according to the traffic density.

Several approaches have been used to develop traffic simulations in all of these levels. Some of them include queuing networks [6,36], Markov models, Cellular Automata [12,38,30,33], DEVS [8,22], software agents [2], object-oriented programming [34] and learning automata [25]. Several other approaches have also been used, from Game Theory [7], Petri Nets [37], up to fluid or electrical flow models equivalent to traffic. The early efforts in this area used macroscopic models to analyze traffic demand and flows on a traffic network using static parameters (such as average of daily traffic or an average for the peek hours). The goal was to find long-term forecasts that could be used for investments or dimensioning of the traffic net. Microscopic simulations are more recent, and they require higher computing power. Nonetheless, they can reproduce the real dynamics of traffic, enabling a modeler to study detailed phenomena as a function of time. We are interested in developing microsimulations 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 or sports stadiums).

Cellular Automata (CA) have been used in defining the kind of models of our interest [1,27,29,30,18,28,32]. CA define a grid of cells using discrete variables for time, space and system states [9,43]. 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. In [13], we presented a survey of the research results in microsimulations with CA. This report showed that most existing models could only represent simple aspects of the flow,

such as the standard car movement in streets and crossings. In addition, the representation of some of the most basic behaviors (i.e., vehicle speed) is done in a highly non-intuitive fashion. Besides these problems, CA are synchronous, a fact that poses precision constraints and extra compute time. The Cell-DEVS formalism [42] was proposed to solve these problems by defining cell spaces as DEVS (Discrete Events systems Specifications) models [44]. 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.

Using DEVS and Cell-DEVS as theoretical frameworks, we defined a high-level specification language called ATLAS (Advanced Traffic LAnguage Specifications) and a compiler [24], called TSC (Traffic Simulation Compiler) that enables a modeler to represent city sections based on constructions found in standard maps. Once the topology of an urban section is defined, traffic flow is automatically set up and microsimulations executed. The city shape can be easily modified by changing the constructions used or some of their parameters. Therefore, a modeler can concentrate in the problem to solve, and not in low-level programming details. The modelers can write high level specifications that will automatically generate executable models, avoiding human intervention in the intermediate steps of the development [40].

The language constructions are mapped into DEVS and Cell-DEVS models, providing the benefits of a formal approach. Consequently, the use of ATLAS allows reducing the development times of traffic simulation. In [17], we showed that the development times of simple traffic models can improve in one order of magnitude when we compare Cell-DEVS against general purpose programming languages. This is due, in part, to the complexity reduction provided by Cell-DEVS (for instance, a one-lane traffic model, represented by 185 lines of code in a high level programming language, can be specified by 15 lines of Cell-DEVS definitions). DEVS also enables reusing predefined submodels, and coupling them in seamless fashion, improving the definition of complex models and favoring reuse. Once a model has been tested, it can be coupled with others hierarchically. Besides this, complex speed behavior can be easily defined as a function of the delay of a cell. Using ATLAS, we could even improve these results (i.e., a 15-line Cell-DEVS specification like the one mentioned can be defined with a one-line construction). This reduces the time spent in the testing and maintenance phases. As we will show, structural changes in the models are done by simply changing model specifications. Once the city section is specified, coding time disappears (as existing tools generate code from the specifications without user intervention). Model testing is highly reduced as we guarantee the correct execution of the simulations (only integration testing must be carried out).

2. DEVS and CELL-DEVS

ATLAS components have been defined as DEVS and Cell-DEVS models. Two reasons support this decision. First, these formal approaches enable to prove the correctness and completeness of the simulation models. Errors found during the simulation can be fixed by analyzing the specification, without considering the underlying

software system. A second advantage is that DEVS and Cell-DEVS are discrete event formalisms, providing higher precision and speedups than the discrete time approaches. In [45], the authors showed that DEVS combined with parallel simulation techniques can produce speedups of up to 1000 times. In [42], we showed that Cell-DEVS also provides these advantages. This is particularly important for traffic models: though cars move at different speeds, we need to be able represent the fastest ones. For instance, to have precise behavior, we would need to represent speeding behavior (for instance, cars moving at 100 km/h in 60 km/h streets). We need to activate all the cells at a rate of 270 ms (considering cells of 7.5 m, the standard size used by CA traffic models), against the 670 ms required for most cases. The situation is even worse if we consider that we need a timeslot which is a common divisor of these two numbers, while a large number of cells do not need to be activation very long periods (for instance, cars stopped by traffic lights, city areas without traffic, or parking spots). Nevertheless, discrete time approaches need to activate every cell in the model at the smallest possible rate.

Let us consider a city section with 2000 cells (a square region of 655 m on the side). In a CA simulation 10 h long, we would activate 267M cells for maximum speeds of 100 km/h (or 48M for 40 km/h). Instead, when a discrete-event approach is used, we should just activate the cells used by the cars in the area. If we consider rates of 100–1000 cars/hour, we would have from 100K to 1M activations in 10 h (independently of speed).

Fig. 1 shows the break-even point for the number of activations using CA and DEVS. The X-axis is divided in three areas: two for CA (maximum speed: 40 and 100 km/h), and one for DEVS. In each category, we include the number of cars in the cell space (1–2000 cars maximum). The Y-axis shows different average speeds for vehicles running under DEVS (as different speeds will change the activation times for each cell). The Z-axis shows the total number of cells activated in a 10-h simulation. Both the number of cars and their speed affect the performance of DEVS models. Nonetheless, in most cases we obtain a smaller number of activations when compared against CA. Performance is worse only for a large number of cars (1000–2000) at high speed. This hypothetical case (a large number of cars speeding), is nonvalid: when such a large number of vehicles exist, most of the cells will be blocked (traffic jam). The situation is even worse when we consider that we need to model traffic lights, parking lanes or special vehicles (ambulances, police cars), which makes it more complex to choose an adequate timeslice to use. Instead, DEVS and Cell-DEVS enable to use different time scales in each of the submodels employed, due to the existence of explicit timing functions that are independent of a global clock.

A real system modeled using DEVS can be described as composed of several submodels. Each of them can be behavioral (atomic) or structural (coupled). A DEVS atomic model is described as

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$$

Each state (S) has an associated lifetime (ta). When this time is consumed, the internal transition function (δ_{int}) is activated to produce an internal state change. The model can generate outputs (Y) using the output function (λ), which executes before

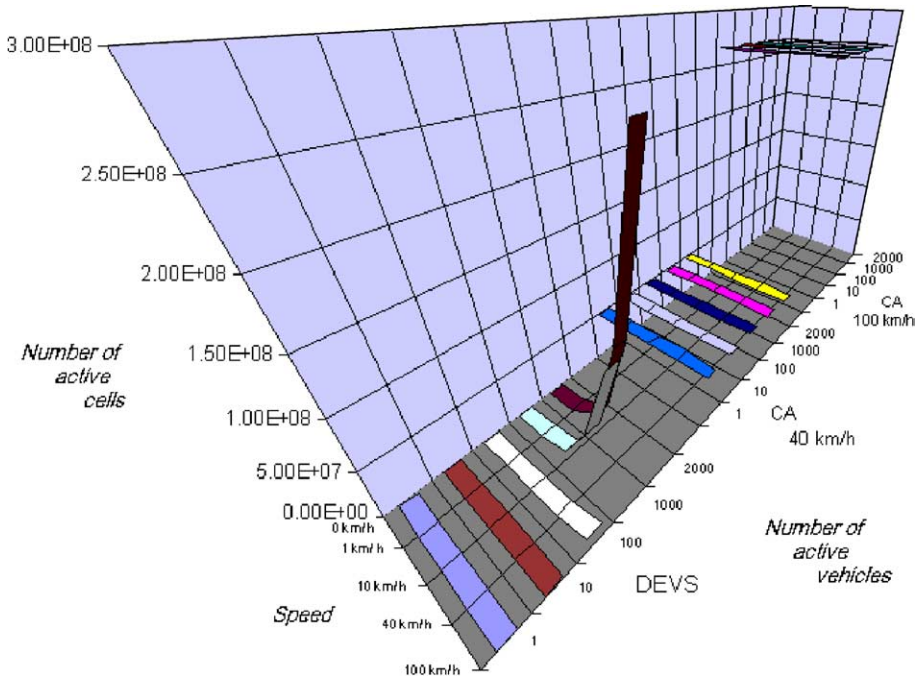

Fig. 1. Comparing the number of cells activated using DEVS and CA.

the internal transition. The model specification also defines the behavior of the external transition function (δ_{ext}) under input events (X). Atomic models can be integrated into a model hierarchy, allowing reuse of the models. As DEVS is closed under coupling, this integration can be done safely. A DEVS coupled model is defined as

$$\mathrm{CM} = \langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}\rangle$$

Coupled models consist of a set of basic models (D) interconnected. The set of influences (I_i) of a given component defines the models to which it must send outputs. The Z_{ij} function is in charge of translating outputs of a model into inputs for others, using the index of influences created for each model: for every j in I_i, outputs of the model M_i are connected to inputs in the model M_j. X and Y define the model's external inputs/outputs.

Cell-DEVS [42] allows one to define cellular models in which each cell is defined as an atomic DEVS (Fig. 2). Cell-DEVS atomic models are specified by

$$\mathrm{TDC} = \langle X, Y, S, N, \mathrm{delay}, \delta_{int}, \delta_{ext}, \tau, \lambda, ta\rangle$$

Each cell uses N input values (received from the cell's neighborhood or from other DEVS models), and computes the new state using the function τ. The outputs of a cell are not transmitted instantaneously, but after the consumption of a *transport* or *inertial* **delay**, which defines the timing behavior of the cell explicitly. Transport

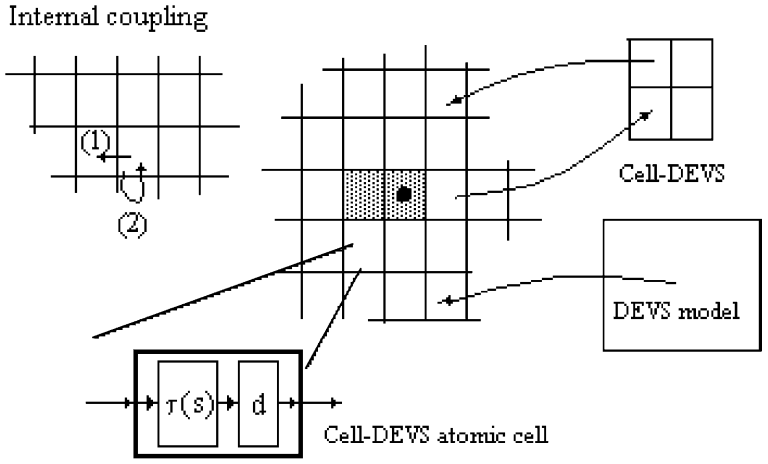

Fig. 2. Informal definition of a Cell-DEVS model.

delays allow one 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. This behavior is driven by the δ_{int}, δ_{ext}, λ and *ta* functions.

3. ATLAS constructions

ATLAS language constructions let a modeler to define a city section with detail. The language enables representing the topology of the area to be analyzed by reproducing a city map. This approach has been followed by most microsimulation tools, such as HUTSIM [21], Tramsims [4] Traffic Simulator [10], AIMSUN [3], Corsim [5,19], TRANSIM [31] or PARAMICS [6,35]. ATLAS formal specifications were used to build a compiler implementing the language [24]. The compiler, called ATLAS/TSC (Traffic Simulation Compiler) takes as input a set of ATLAS constructions, and generates a Cell-DEVS specification. The compiler was built based on a set of templates that defines how to code the output rules, and translated into Cell-DEVS specifications which can be executed using the CD++ simulator [41]. These templates can be changed to generate new rules, or to change the existing ones, providing independence from the simulation environment. For instance, if the syntax of CD++ changes or another toolkit is intended to be used, we can change the templates, and the traffic models previously developed would still be valid. Each construction defines a static view of the model, which have an implicit associated dynamic behavior.

3.1. Segments

Each street in the city is defined as a sequence of one-lane **segments** (Fig. 3). Segments are one way and have a maximum speed associated (one segment in each

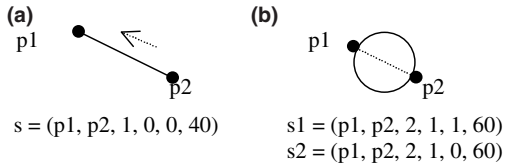

Fig. 3. Specification of segments: (a) one lane straight segment; (b) two-lane curve segments with different direction.

direction is needed to build two-way streets). In ATLAS/TSC, segments are specified using the delimiters **`begin/end segments`**. At least one segment must be defined, using this syntax

```
id = p1, p2, lanes, shape, direction, speed, parkType
```

Here, **`p1`** and **`p2`** represent the boundaries of the segment. Then, we define the number of **`lanes`** in the segment, and the vehicle **`direction`** (**`go`** means that vehicles move towards **`p2`**, and **`back`**, otherwise). The, we define the **`shape`** of the segment (**`curve`** or **`straight`**), and the maximum **`speed`**.

Segments are the basic building blocks, which enable us to define the shape of the section to be studied. This construct enables the modeler to essay alternatives (for instance, reversing the lane directions in peak hours, retracing the topology in a conflicting area, analyzing the influence of changes in the maximum speeds, etc.).

3.2. Crossings

Every segment is connected to a **crossing** (street intersections), which interconnects a number of segments which provide inputs or outputs according to their direction. As we can see in Fig. 4, a crossing is represented as a ring: some cells receive cars from a segment; others are used to leave the crossing. A car moving in the intersection has higher priority to obtain the following position in the ring than those outside the crossing. Each vehicle advances continuously in order to avoid deadlocks. This approach was successfully used in [12,11,10].

The crossing construction enables car routing, and permits modeling every possible connection (right or left turns, U-turns, etc.) in a simple and intuitive way.

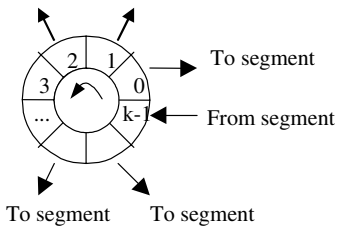

Fig. 4. Crossing.

Besides, the application of the construction to model roundabouts or dead-ends is straightforward. In TSC, crossings are delimited by the separators **`begin/end crossings`**. Each sentence defines a crossing using the following syntax: **`id = p1, p2, speed, tLight, crossHole, Pout`**.

Parameters **`p1`**, **`p2`** represent the position of the crossing, and **`speed`** defines the maximum speed in the crossing. **`Pout`** defines the probability of a vehicle to abandon the crossing, used to simulate random routing.

3.3. Traffic lights

Every existing microsimulation language includes models for traffic lights, as they are essential in controlling traffic flow. Correct traffic light synchronization can improve the traffic according to the traffic density, weather conditions, congestion, pedestrian crossings, etc. We defined a traffic light construction, specified as: **`tLight: [withTL | withoutTL]`**.

Traffic lights are defined as an extension to the crossing construction, representing the lights and a controller. Each light in the intersection will be associated to an input cell in the crossing, as showed in Fig. 5. The traffic light model selects the light color and sends it to the output cell in the segment corresponding to the input crossing. The synchronizer coordinates all the lights in the crossing, and they may be controlled by a common coordinator for the city section.

3.4. Railways

Vehicle flow in a city section can be seriously influenced by level crossings (Fig. 6). We have included a construction to model railways, which can be useful in analyzing train schedules, barrier synchronization schemes, evaluation of building bridges/tunnels, etc. Each train follows a predefined advance sequence through level crossings overlapped with the city segments. In TSC, the railway network is defined by the **`begin/end railnets`** clauses, which act as separators for the definition of a *railnet* (a part of the railway network in a city section). Each railnet is defined using the following syntax:

`id` = (**`s`**$_1$, **`d`**$_1$) {, (**`s`**$_i$, **`d`**$_i$)}

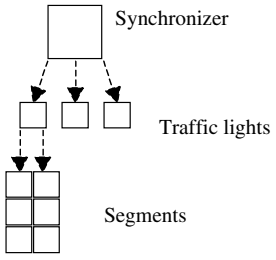

Fig. 5. Crossing with traffic lights.

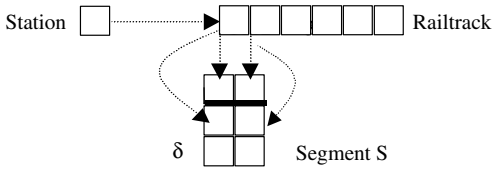

Fig. 6. Level crossing definition.

where $\mathbf{s}_i$ is the identifier of a segment crossed by the railway, and $\mathbf{d}_i$ is the distance between the beginning of the segment and the railway. The railway topology is defined by the places where the level crossings are located. The compiler automatically assigns a sequence number to each level crossing in a *RailTrack*. A *Station* represents a source of trains for the *RailNet*.

3.5. Traffic signs

We defined a construction to represent different traffic signs in order to constrain vehicle movement. This construction is useful, for instance, to study the effect of replacing stop signs by traffic lights, the cost/benefit of having a pedestrian crossing or a raised crossing, etc. In TSC, traffic signs are defined by the **`begin/end ctrElements`** delimiters, and: **`in s : Type, distance`** is the definition for each sign, with **`Type : [bump|depression|pedXing|saw|stop|school]`**. The **`distance`** defines the location of the sign in the segment (Fig. 7). Traffic signs define a zone of one cell length in all the lanes, where the behavior of the segment is modified according to the sign (in general, reducing speed).

An extension of this construction allows one defining potholes, which produce a speed reduction of the vehicles passing through them. Pothole fixing tasks are usually complex, as their number in a city can be large. The availability of this construction enables analyzing the priority for fixing them, and, combined with the Roadwork construction, it allows defining policies to improve traffic conditions. In TSC, they are defined by the **`begin/end holes`** separators. Each hole is defined as: **`in s : lane, distance`** representing the position of the hole on a given segment **`s`**. A pothole can also be included in a crossing. In this case, **`crossHole : [withHole | withoutHole]`** defines if the crossing contains a pothole or not.

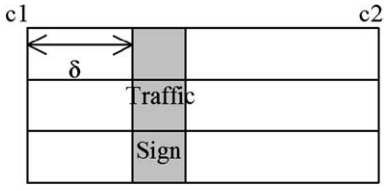

Fig. 7. Definition of a traffic sign.

3.6. Roadwork

We also defined a construction to describe men at work, which enables analyzing congestion in job areas. These traffic obstructions are specified with the **`begin/end jobsites`** separators, and **`in s : firstlane, distance, lanes`**.

Each **`jobsite`** is related to a segment **`s`** where the work is being done. The remaining parameters include the first lane affected (**`firstlane`**), the distance between the center of the jobsite and the beginning of the segment (**`distance`**), and the number of lanes occupied (**`lanes`**). These parameters are depicted in Fig. 8. As we can see, we use them in order to define a rhombus over the segment. The vehicles cannot pass through this area, and they must deviate. This construction is useful to analyze traffic behavior in the area surrounding a Roadwork. The same construction can also be used to define car collisions.

3.7. Parking lanes

Cars parking in the streets can produce delays in the traffic flow and lead to congestion. Nevertheless, other types of parking require extra infrastructure, and public parking is an important source of funds for the cities (Fig. 9). We have included a construction that lets the modeler to experience with different parking strategies (timeslots, parking for permit holders only, night parking, etc.). In addition, ticketing policies can be decided according to the influence of a wrongly parked vehicle. Segments can include information for parking, with **`parkType : [parkNone|parkLeft|parkRight|parkBoth]`**. Every segment identifies the lane where car parking is allowed. When a car arrives into a parking lane, it will stop

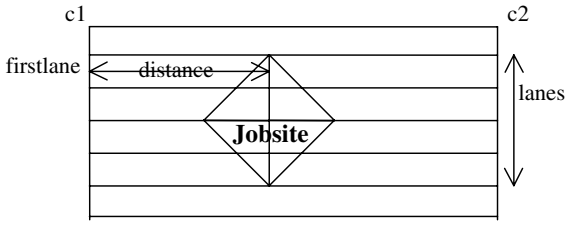

Fig. 8. Segment with Roadwork.

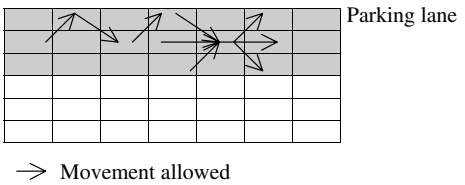

Fig. 9. Parking segments.

during a given amount of time. This is modeled using a long transport delay (several minutes or hours).

4. Applying ATLAS to define a city section

This section is devoted to show how to define a city section using ATLAS, introducing the kind of problems we are trying to solve. Fig. 10 shows a section of the city of Buenos Aires, Argentina. It is a part of residential neighborhood, where the traffic flow is non-significant, even in peak hours. Nevertheless, the complexity in the trace of the area makes traffic jams likely to occur. The neighborhood includes a park, a railway, several one way streets, dead ends, and several avenues (one with four lanes in each direction). In several of these streets, parking is allowed, while in others it is forbidden.

The basic idea is that by applying ATLAS in this kind of city trace, we can provide different strategies to solve existing problems, and analyze the simulation results to choose the best solution. Fig. 11 shows a part of this city section in detail, labeling the segments and crossings.

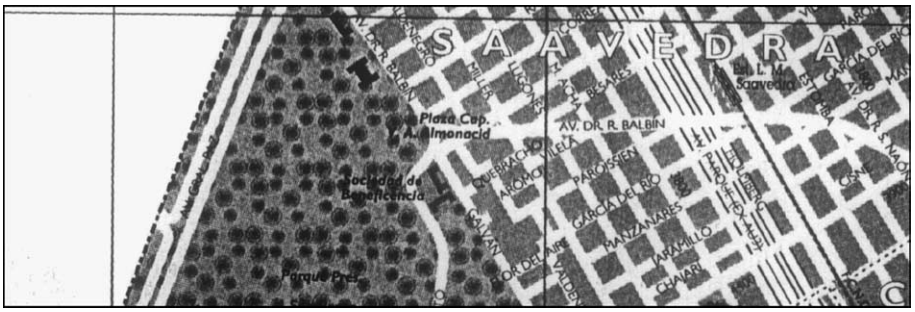

Fig. 10. A section of Buenos Aires, Argentina.

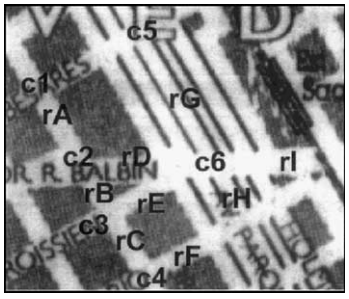

Fig. 11. Specification of the segments and crossings in the city section.

Using ATLAS/TSC, this section can be specified as follows: the specification considers a plane representing this city section starting in the crossing *c1*. Segment *rA* is a one-way/one-lane segment. The maximum speed allowed is 40 km/h, and it is part of a straight street. Segments *rB* and *rC* are the following segments belonging to the same street. Segment *rD* is a two-way segment, therefore, two components *rD1* and *rD2* are defined (each with different direction). Segment *rI* is the following segment in the same street. Finally, *rG* is a two-way segment with 4 lanes in each way. The crossing specifications show the position and maximum speed allowed for each of them The railway construction shows that a level crossing intersects the segment *rI* (*rI1* is crossed at 10 m from the corner, and *rI2* at 90 m before the following crossing).

As we can see, ATLAS constructions let the user to define a model by simply specifying the map topology (Fig. 12). This approach represents a clear advantage over other modeling and simulation applications:

- The user only has to focus in defining the city section map correctly. Any errors in the model are related with wrong definitions of the city map, which can be easily fixed.
- Programming problems are avoided: the user does not need to write source code to implement the models.

```
begin segments
rA=(2,1),(7,16),1,straight,go,40, parkNone
rB=(7,16),(11,25),1,straight,go,40, parkNone
rC=(11,25),(14,34),1,straight,go,40, parkNone
rD1=(7,16),(22,16),2,straight,go,60, parkNone
rD2=(7,16),(22,16),2,straight,back,60, parkNone
rE=(11,25),(22,16),1,straight,back,40, parkNone
rF=(14,34),(21,31),1,straight,go,40, parkNone
rG1=(17,2),(22,16),4,straight,go,60, parkNone
rG2=(17,2),(22,16),4,straight,back,60,parkNone
rH1=(22,16),(24,26),2,straight,go,60,parkNone
rH2=(22,16),(24,26),2,straight,back, 60,parkNone
rI1=(22,16),(40,16),2,straight,go,60, parkNone
rI2=(22,16),(40,16),2,straight,back,60,parkNone
end segments

begin crossings
c1=(1,2), input, exponential, 10
c2=(7,16),30, withoutTL, withoutHole, .65
c3=(11,25),30, withoutTL, withoutHole, .8
c4=(14,34),30, withoutTL, withoutHole, .8
c5=(22,16), input, exponential, 30
c6=(22,16),30, withoutTL, withoutHole, .7
end crossings

begin railnets
rIRail = (rI1,8),(rI2,11)
end railnets
```

Fig. 12. Definition of the Area in ATLAS compiler.

- Testing and experimentation are improved. Any modifications will affect the section specifications only.

5. Mapping ATLAS models into DEVS

The constructions defined by ATLAS allow us describing the shape and behavior of traffic models. Each of these constructions is translated into DEVS and Cell-DEVS using a set of procedures defined with detail in [15,16].

5.1. Segments

Each segment is translated into a bidimensional Cell-DEVS. As showed in Section 3.1, each segment can have different number of lanes. We need to provide different behavior for the local computing functions according to the number of lanes, because different border conditions exist in each case. The simplest case allows us defining one lane segments (such as *rA* in the example presented in Section 4). The construction is translated into a one-dimensional Cell-DEVS with transport delays.

Fig. 13 informally depicts a Cell-DEVS with k cells, in which the border cells are connected to crossings. The Cell-DEVS atomic models corresponding to the one-lane segment are defined as

$$S1 = \langle X, S, Y, N, \delta_{\text{int}}, \delta_{\text{ext}}, \text{delay}, d, \tau, \lambda, ta \rangle$$

$X = \{(X_1, \text{binary}),\ (X_2, \text{binary}),\ (X_3, \text{binary})\}$; $Y = \{(Y_1, \text{binary}),\ (Y_2, \text{binary}),\ (Y_3, \text{binary})\}$.

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

$N = \{(0,-1), (0,0), (0,1)\}$; delay = transport; d = speed(max).
λ, δ_{int} and δ_{ext} behave as defined in the Cell-DEVS formalism with transport delays [42].
τ: $S \times N \to S$ is defined as follows:

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

Fig. 13. One lane segment.

The first rule represents a vehicle arriving to an empty cell. The second rule represents the car abandoning the present cell towards the front. Otherwise, the cell preserves the present value (these basic rules can be easily extended to check congestion conditions by looking ahead several cells; as ATLAS/TSC is based on templates, new rules can be easily incorporated into the template). Transport delays are used to model the time can be a vehicle spends leaving a cell and getting into the next one. This time is generated by using a function which combines a random value with the current speed: the current speed of the car is represented by the delay of the car in crossing the cell, and this value can be partially modified using a random value (choosing different distributions we can model varied speed behaviors). In this way, we can consider congestion (defined by the movement rules) and also varied speed behavior from different drivers.

The parameters **p1**, **p2** are used to build a coupled model corresponding to the segment. The maximum **speed** is used to compute the speed function. The segment position is used to compute the number of cells. We use a length for each cell representing the average length of a car plus some extra space (7.5 m), as defined in [39,31]. For segment *rA* we have: $k = \left\lceil \sqrt{|x1 - x2|^2 + |y1 - y2|^2}/cell\_size \right\rceil = \left\lceil \sqrt{0 + 130^2}/7.5 \right\rceil = 18$. Then, the coupled model corresponding to a one-lane segment construction is specified as follows:

$$\mathrm{CS1}(k, \max) = \langle X_{\mathrm{list}}, Y_{\mathrm{list}}, X, Y, n, \{t_1, \ldots, t_n\}, \eta, N, C, B, Z \rangle$$

$Y_{\mathrm{list}} = X_{\mathrm{list}} = \{(0,0),(0,k-1)\}$; $X = \{\langle X_{\eta+1}(0,0), binary\rangle, \langle X_{\eta+1}(0,k-1), binary\rangle\}$, $Y = \{\langle Y_{\eta+1}(0,0), binary\rangle, \langle Y_{\eta+1}(0,k-1), binary\rangle\}$. These ports will be named as follows (the i parameter corresponds to the lane number; in our case, as a 1-lane segment model is being considered, then $i = 0$): $X_{\eta+1}(i,0) = x\text{-}c\text{-}vehicle$; $X_{\eta+1}(i,k-1) = x\text{-}c\text{-}space$; $Y_{\eta+1}(i,0) = y\text{-}c\text{-}space$; $Y_{\eta+1}(i,k-1) = y\text{-}c\text{-}vehicle$. $n = 1$; $t_1 = k$; $\eta = 3$. $N = \{(0,-1),(0,0),(0,1)\}$; $B = \{(0,0),(0,k-1)\}$; and Z is built using the neighborhood definition, as specified by Cell-DEVS formalism.

Cells $(0,0)$ and $(0,k-1)$ comprise the external interface of the model, because they must interchange information with the crossings corresponding to the street corners. The coupling scheme for these border cells is described in Fig. 14. We can see that the cell $(0,0)$ receives a new car in the port *x-c-vehicle*, and informs the state of the cell through the *y-c-space* port. In cell $(0,k-1)$, we inform the state of the segment through the port *y-c-vehicle*, and receive information from the crossing through port *x-c-space*.

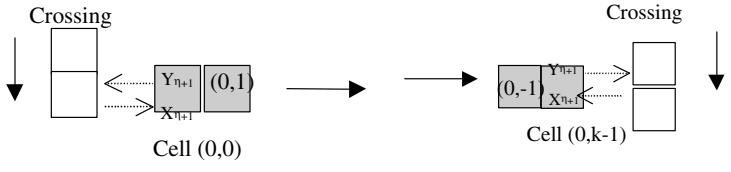

Fig. 14. Coupling for the cells $(0,0)$ and $(0,k-1)$.

Hence cell $(0,0)$ is different: $\eta = 2$; $N = \{(0,0),(0,1)\}$, while the first rule of the τ function rules must be changed to

$\tau(N)$	N
1	Portvalue(*x-c-vehicle*) = 1 and $(0,0) = 0$

This new rule represents the arrival of a new car from a crossing to the segment: whenever a vehicle wants to get out of the crossing (e.g., port *x-c-vehicle* is 1) and the receiving cell is empty, the vehicle moves to the cell. The cell $(0, k-1)$ must be coupled with a cell in the crossing and the equivalent behavior is defined for this cell.

If the segment to be translated has two lanes (Fig. 15), we create a two-dimensional Cell-DEVS with the following structure.

Each row of this space acts as a border: vehicles in the first row can decide to change to the right, and those in the second row can choose to move to the left. Each row is specified separately as an extension to the one-lane cells. The main changes in the first row include the interface (X, Y), neighborhood (N), and the τ function rules we use are the same than those defined for one-lane models, adding the following rules in the cases that we want to represent lane changes (other behaviors, according to the characteristics of driving characteristics in the country/city of interest can be incorporated as extensions to these ones)

$\tau(N)$	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 $(-1,0) = 0$

The first rule here represents a vehicle arriving in diagonal. Lane change rules consider that a vehicle try to move straight first. A vehicle moving forward has the priority to access to the position in front of it. To define the priority access, the diagonal movement checks if there is no car waiting to move forward. If that is not the case, the diagonal movement can be executed. The main difference in the definition for the second lane includes changing the neighborhood and the local computing function, whose definition is symmetric to the previous.

The interface of this model is integrated by all the cells in the first and last columns, which can be used to interchange information with the limiting crossings, which are extensions of those defined previously for the one lane model. Similar

Fig. 15. A two-lane segment.

mechanisms are used to translate the segment constructions into models with three to five or more lanes. Further details can be obtained in [16].

5.2. Crossings

Atomic cells description is based in those defined for the one lane model of Section 5.1 (the rules must define that each cell in a crossing is connected to a segment). The local computing function τ: $S \times N \to S$ will execute different rules for input or output cells. Details about these functions can be found in [16]. For each crossing, we build a coupled model defined by

$$\text{Crossing}(k, In, Out) = \langle X_{\text{list}}, Y_{\text{list}}, X, Y, n, \{t_1, \ldots, t_n\}, \eta, N, C, B, Z \rangle$$

This model defines a crossing of k cells, where the positions in the set *In* are input cells, and *Out* output cells. The parameters (k, In, Out) are obtained by checking the direction of the segments connected to the crossing. Initially, we use their direction and check which cells must be used as inputs or outputs. Then, we use the number of lanes of each segment is used to compute the total number of cells needed: we compute the number of lanes in each segment connected to the crossing, and we define a unique ordering for the input/output segments (so that segments close to each other are connected to neighboring cells in the crossing).

5.3. Traffic lights

The first step to translate the traffic light construction is to build a DEVS model *Sync* for each of them, representing the controller in charge of setting up the color of each traffic light connected to the corner. For instance, when we consider the crossing *c6* in the example of Section 4, we build $T_{\text{in}}(c6) = \{rG1, rH2, rD1, rI2\}$. A model $Sync_{c6}(4)$ is created, and for each input segment for the crossing, a DEVS model representing the traffic light is built. This model transmits the traffic light color to the segment providing inputs to the crossing, and changes the light color when the *Sync* model decides it is time to do it. Specification details of these DEVS models can be found in [15].

5.4. Railways

As stated earlier, the railway network is a set of *RailTracks*, each defining the level crossings after a train station. For every element in *RailNet*, two models are defined. The first one is a DEVS representing a station. It is built as a generator according to the train's timetable. It is connected with a *RailTrack* model, defined as a one-dimensional Cell-DEVS with transport delays that models all the level crossings. Each cell represents the presence of a train advancing independently of the cells in front of it. The train speed is defined by a function using transport delays (which computes the delay between crossings, depending on distance and speed). The behavior of a segment crossed by a railway should be changed to stop the vehicles while a train is passing. The cells affected are those around $col = \lceil \delta / cell\_size \rceil$, defined as $Cpn =$

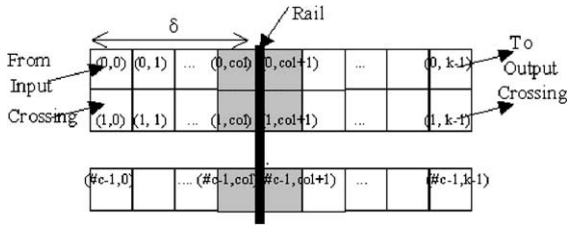

Fig. 16. Segment with level crossing.

$\{(i,col)/i \in [0,n-1]\} \cup \{(i,col+1)/i \in [0,n-1]\}$, where n is the number of lanes in s. The cars in cells $\{(i,col)/i \in [0,n-1]\}$ must stop when a train is crossing (described in Fig. 16).

5.5. Traffic signs

A segment including one of the traffic signs showed in Section 3.5 will be defined as any other segment, and the delay time used is increased in the cells influenced by the sign. In this way, we can represent a slower speed of the vehicles approaching to the sign.

5.6. Roadwork

The Jobsite constructions are translated into Cell-DEVS models using different rules according to the size and position of the jobsite and its number of lanes. Cells inside jobsite will always keep a value of 0. As cars cannot advance into the jobsite, we must group the cells before it according to the movements that are permitted. Fig. 17 shows a sketch of a jobsite in which traffic advances from left to right. In this case, vehicles can move to the left diagonal (LD), the right diagonal (RD), either (2D), or advance in a straight line. We have defined different rules for each of these cases.

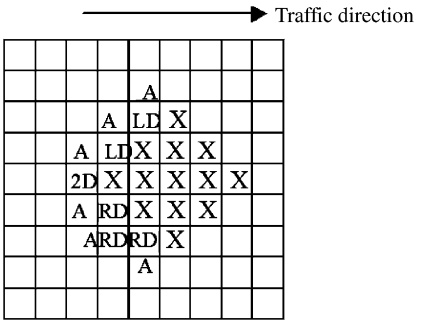

Fig. 17. Jobsite cells.

6. Implementation models

After defining ATLAS constructions and their translation in to DEVS models, we faced implementation of the TSC compiler. The first steps were focused on the validity of the language. In [44], three types of validity are distinguished: **replicative**, **structural** and **predictive**. We carried out validation activities in each of these levels, checking ATLAS validity with respect to two factors:

- validity of the constructions' behavior: we tested that every construction proposed was valid;
- validity of the coupled constructions: besides the individual behavior provided by each construction, we need to obtain a valid behavior for larger models, integrated by a number of components defined using different constructs.

After, we faced activities on simulation verification. As we used CD++ as the development platform, and the tool follows the formal definitions presented in [42,41], model execution is correct (CD++ simulator was formally verified for correctness). Hence, our activities were mainly devoted to check if the constructions proposed provided a valid dynamic behavior. Therefore, we first used CD++ and built a set of independent models for each of the constructions, analyzing the validity of the rules proposed in Section 5. After, we tested the integration of the constructions.

(A) *Replicative validity:* A model is considered replicably valid if, for every experiment within an experimental frame, the behavior of the model and the real system under analysis agree within acceptable tolerance. In our case, we have guaranteed replicative validity using two different approaches. First, the rules we used to define the dynamic behavior of the vehicles were built as direct mapping from CA models that were already proven valid at the replicative level [12,11,10,39,31]. The second phase was devoted to check that the Cell-DEVS specifications were equivalent to those we used. To do so, we showed that the Cell-DEVS constructions were homomorphic to the CA definitions at the level of input/output trajectories [14].

(B) *Structural validity:* A model structurally valid is able to replicate the data observed from the system, but also to mimic system interactions in step-by-step, component-by-component fashion. As CA are structural models, the original rules used also provided structural validity. Structure of coupled models is guaranteed to be correct, as they are automatically built using well-known results in geometry and their coupled behavior is guaranteed by the use of DEVS.

In order to guarantee validity, the constructions used by the compiler must be homomorphic to represent the original ones, which were already validated. We used the same specifications to build the compiler, which was enough to ensure validity. We also attacked a new level of structural validation, related to checking map properties within the specifications written in ATLAS. Different types of conditions can be automatically checked:

- Every segment has an associated crossing (even in dead ends a crossing is needed to allow U-turns).

- Every crossing is associated to an existing segment.
- There are no crossing-to-crossing or segment-to-segment couplings.
- The maximum speed is the same for a street (as in some cases different speeds are allowed, only a warning is issued).
- Street direction is correct in each of the segments.
- At least one segment exists.
- A segment does not have the same beginning and end.
- There are no isolated segments or crossings.
- No more than one crossing exists at the same point.
- Railways, potholes, control signals and jobsites should be defined within an existing segment, and cannot exceed the segment boundaries.
- Railways cannot cross the segments close to their borders.
- Segments in which parking is permitted must have at least 2 lanes (to allow parking in one side) or 3 lanes (if parking in both sides is allowed).

After carrying out this step, we validated the behavior of city sections consisting of several components. Because of this phase, we were able to find some constructions whose behavior was not valid. For instance, our original definitions for parking and railways (which were defined using rules that were not validated previously) showed incorrect behavior when compared with the real system involved.

(C) *Predictive validity:* A model is predictably valid if replicative validity holds, and the ability to predict unseen system behavior can be achieved. After obtaining structural validity, we conducted some experiments in order to see the scope of predictive validity. We developed some examples based in actual city sections (such as the one described in Section 4), and obtained results that seem to be valid for certain phenomena not covered by structural validity. When comparing field data for the sections under study, and checked it against the simulation results, we obtained acceptable tolerance margins (less than 3%).

In Fig. 11, we saw that there is a train station close to the crossing *c1*. This level crossing uses an automatic barrier that closes when a train is approaching to the station. While a train is at the station, the barrier is closed. We have simulated the present behavior of the train using an average delay of 4 min for the level crossing (which is the actual time taken by the train). Fig. 18 shows the results obtained

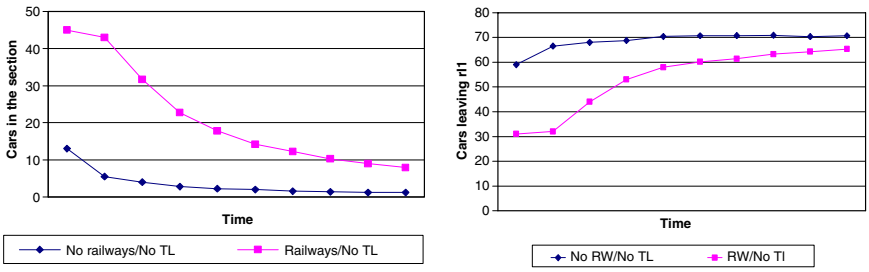

Fig. 18. Execution results in the area.

within a fixed period (10 min simulated time) using different input rates according to the actual traffic flow in the area.

The first graphic in Fig. 18 shows the number of cars in the section. In the first minutes we can see congestion in the area due to the closing of the barrier (although traffic flow was almost constant). Then, we repeated the tests replacing the level crossing by a tunnel. The influence of the railway can be clearly seen in the second figure, which shows the number of cars leaving the segment *rI1*. The level crossing makes that about half of the cars cannot leave the area. The results were validated by comparing them with the traffic flow rates in the same time slot (using data obtained while the barrier is open to compare), which were similar to those shown in the figure.

In order to study the influence of non-synchronized traffic lights, we modified the specifications of crossings *c1* and *c2* as follows:

```
c1 = (22, 16), 10, withTL, withoutHole, 3.
c2 = (7, 16), 10, withTL, withoutHole, 3.
```

In Fig. 19, we can see that the number of cars in the city section increments when traffic lights are included. We can also see the combined influence of traffic lights and railways: as lights are not synchronized, the number of cars in the area is higher.

We also compared the degree of congestion during peak times. This is done by modifying the experimental framework in order to increase the rate of traffic generated:

```
c1 = (1, 2), input, exponential, 20.
c5 = (22, 16), input, exponential, 70.
```

In Fig. 20, we see that in some cases, the number of cars in the area is more than double than in off-peak hours. The input/output ratio is higher in peak hours, due to a higher number of cars flowing through the area. Nevertheless, having the double of cars only increases the input/output ratio in 10–20%. This shows that the topology of the area is complex enough to have a small I/O ratio even in non-peak hours.

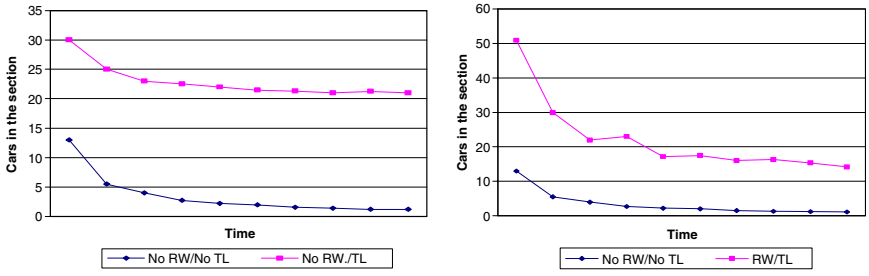

Fig. 19. Execution results adding traffic lights.

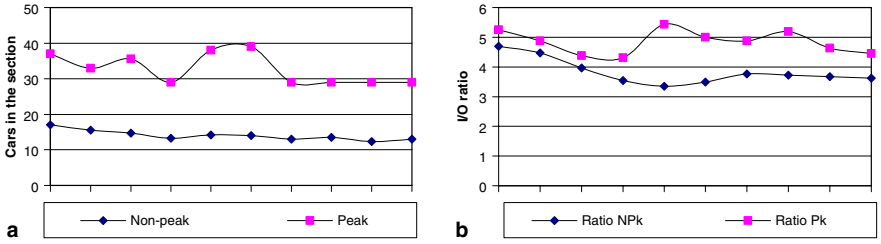

Fig. 20. Non-peak traffic against peak traffic: (a) number of cars in the area; (b) I/O ratios.

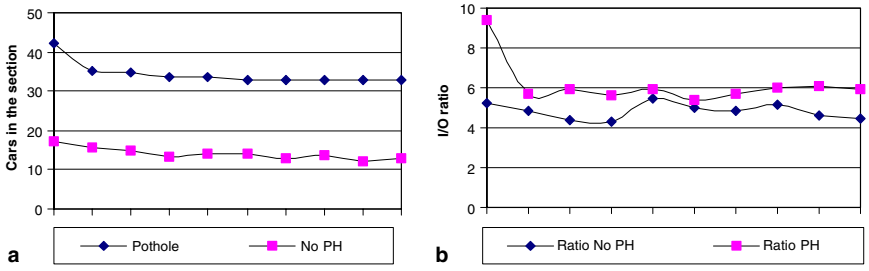

Fig. 21. Peak traffic with a pothole: (a) number of cars in the area; (b) I/O ratios.

We analyzed the influence of having a pothole (in crossing *c6*). As explained earlier, the behavior of a car crossing a pothole is to reduce its speed. The hole is defined as follows:

```
c6 = (22, 16), 30, withoutTL, withHole, .7.
```

Results included in Fig. 21 show that the I/O ratio reduces about 1/3 when the pothole is fixed. The number of cars in the area reduces one half, as the cars can leave the section earlier, thus, the number does not change as abruptly as in the previous cases, because the pothole produces a speed reduction in the main crossing (keeping the number of cars in the area constant). The results obtained when using the crossing with potholes are still not validated (as there are no potholes). We intend to do field work in the area whenever if a pothole is detected. In that case, we could be able to analyze the predictive validity for this construction.

7. Conclusion

We have presented a specification language used to define sections of cities as cell spaces. The language allows defining a static view of a city section, allowing including different components: traffic lights, railways, men at work, transit signals, parked

cars, etc. This approach provides an application oriented specification language, which allows the definition of complex traffic behavior using simple constructions. The models are formally specified, avoiding a high number of errors in the developed application, thus reducing problem solving time. The constructions are mapped into DEVS and Cell-DEVS. These translated models are formally specified, improving errors in their definition. Our cell-based framework, allows user to study vehicle movement in detail, allowing analysis of different to existing problems (car crashes, parking problems, etc.).

DEVS and Cell-DEVS helped us verifying the correctness of the simulation results. The development used the CD++ tool as a basis. CD++ was developed using the definitions of the Cell-DEVS formalism, and it allows defining both DEVS and Cell-DEVS models. DEVS atomic models can be written in C++, and coupled and Cell-DEVS models are described using a built-in specification language. The formalisms provide separation between the model and simulator concepts, and the correctness of the simulation algorithms implemented in CD++ has been proven. Therefore, we only required providing a correct translation between the language constructions and the CD++ specifications. This was done by comparing the specifications generated using the tools with the original specifications written by hand. In every case, the code generated was equivalent, ensuring correctness of the compiled code.

As we could see, there was a significant reduction in the coding effort. The specification of the example presented in Section 4 (24-lines) was translated into over 120 lines of Cell-DEVS specifications. An equivalent model defined in a standard programming language had more than 1400 lines of code. As we could see, changing the behavior in the model only involved changing a few lines in the specification. Providing this behavior in a standard programming language involved reading and modifying about 30–40% of the source code. This shows how the use of this specific purpose language can improve the definition of traffic models. The results showed to be even better when more complex models were created, as the testing of a complex city section involved a much more complex testing effort, which could be avoided using ATLAS.

At present, a graphical user interface is being created, to allow easy definition of the models. The GUI will validate the static model based on information given by the map and the constructions used. We also intend to build a translator of Geographical Information Systems representing city maps into ATLAS. Finally, efficient execution of the models is being considered by means of parallel execution of Cell-DEVS. We are also defining new extensions to the ATLAS compiler, allowing including routing information, behavior of the drivers, and congestion avoidance mechanisms.

Acknowledgements

This project was developed by a team of students under my supervision at Carleton University (Ottawa, ON, Canada) and the University of Buenos Aires (Argen-

tina), which collaborated in different aspects of the research: Shannon Borho, Alejandra Davidson, Alejandra Díaz, Mariana Lo Tártaro, César Torres, and Verónica Vázquez. The research was funded by NSERC, the Canadian Foundation for Innovation, the Ontario Innovation Fund, and the ANPCYT (Argentina, National Agency for Science and Technology).

References

[1] R. Akçelic, A roundabout case study comparing capacity estimates from alternative analytical models, in: Second Urban Street Symposium, Anaheim, California, USA, 2003.
[2] B. Balmer, N. Cetin, K. Nagel, B. Raney, Towards truly agent-based traffic and mobility simulations, in: Autonomous Agents and Multi-Agent Systems Conference, New York, NY, USA, 2004.
[3] J. Barceló, J. Casas, J. Ferrer, D. García, Modeling advanced transport telematic applications with microscopic simulators: the case of AIMSUN, in: Proceedings of the 10th SCS European Simulation Symposium, 1998.
[4] C. Barret, R. Beckman, K. Berkbigler, Transportation analysis simulation systems (TRANSIMS), vol. 0—overview, Los Alamos National Laboratory Report, LA-UR 99-1658, 1999.
[5] M. Bumble, L. Coraor, L. Elefteriadou, Exploring CORSIM runtime characteristics: profiling a traffic simulator, in: Proceedings of the 33rd Annual Simulation Symposium, IEEE Press, Washington, DC, USA, 2000.
[6] G. Cameron, B. Wylie, D. Mc Arthur, PARAMICS: moving vehicles on the connection machine, in: Proceedings of IEEE Supercomputing '95, 1995.
[7] O. Chen, M. Ben-Akiva, Game theoretic formulation of the interaction between dynamic traffic control and dynamic traffic assignment, Technical Report, ITS, M.I.T, 1998.
[8] S. Chi, J. Lee, Y. Kim, Using the SES/MB framework to analyze traffic flow, Transactions of the SCS 4 (14) (1997) 211–221.
[9] B. Chopard, M. Droz, Cellular Automata Modeling of Physical Systems, Cambridge University Press, 1998.
[10] B. Chopard, A. Dupuis, P. Luthi, A CA model for urban traffic and its applications to the city of Genoa, in: Proceedings of Traffic and Granular Flow, 1997.
[11] B. Chopard, P.A. Queloz, P. Luthi, CA model of car traffic in two-dimensional street networks, Journal of Physics A 29 (1996).
[12] B. Chopard, P.A. Queloz, P. Luthi, Traffic Models of a D road network, in: Proceedings of the 3rd CM users' Meeting, Parma, Italy, 1995.
[13] A. Davidson, A. Díaz, V. Vázquez, G. Wainer, A comparative study of CA application for simulation of urban traffic models, Technical Report 99–005, Departamento de Computación, FCEN/UBA. 1999.
[14] A. Davidson, Defining models of urban traffic using Cell-DEVS, M.Sc. thesis, Universidad de Buenos aires, 2000 (in Spanish).
[15] A. Davidson, G. Wainer, Specifying control signals in traffic models, in: A. Davidson, G. Wainer (Eds.), Proceedings of AI, Simulation and Planning in High Autonomous Systems, AIS'2000, Tucson, Arizona, USA, 2000.
[16] A. Davidson, G. Wainer, Specifying truck movement in traffic models using Cell-DEVS, in: Proceedings of the 33rd Annual Simulation Symposium, IEEE Press, Washington, DC, USA, 2000.
[17] A. Díaz, V. Vázquez, G. Wainer, Application of the ATLAS language in models of urban traffic, in: Proceedings of 34th IEEE/SCS Annual Simulation Symposium. Seattle, WA, USA, 2001.
[18] J. Esser, M. Schreckenberg, Microscopic simulation of urban traffic based on CA, International Journal of Modern Physics C 8 (1997) 1025–1036.
[19] J. Fitzgibbons, R. Fujimoto, R. Guensler, M. Hunter, A. Park, H. Wu, Simulation-based operations planning for regional transportation systems, in: National Conference on Digital Government, Seattle, WA, USA, 2004.

[20] V. Hlupic, Discrete-event simulation software: what the users want, Simulation 73 (6) (1999).
[21] I. Kosonen, M. Pursula, A simulation tool for traffic signal control planning, in: Third International Conference on Road Traffic Control, IEE Conference Publication Number 320, London, UK, 1990.
[22] J. Lee, S. Chi, Using symbolic DEVS simulation to generate optimal traffic signal timings, Simulation 81 (2005) 153–170.
[23] E. Lieberman, A. Rathi, Traffic simulation. Technical Report, Oak Ridge National Laboratory. Available from: <http://www.onrl.gov/>, 1997.
[24] M. Lo Tártaro, C. Torres, G. Wainer, TSC: a compiler for the ATLAS language, in: Proceedings of 2001 Winter Simulation Conference, IEEE Press, Arlington, VA, USA, 2001.
[25] C. Ünsal, P. Kachroo, J. Bay, Simulation study of multiple intelligent vehicle control using stochastic learning automata, Transactions of the SCS 14 (4) (1997).
[26] H. Mahmassani, DYNASMART-X system Part 1: overview, structure, functions, key components, Sessions on Dynamic Traffic Assignment II and III, INFORMS Annual Meeting, San Diego, CA, 1997.
[27] V. Maniezzo, CA and roundabout traffic simulation, in: Proceedings of Sixth International Conference on Cellular Automata for Research and Industry, Amsterdam, Netherlands, LNCS 3305, 2004.
[28] S. Marinosson, Simulation of the autobahn traffic in North Rhine-Westphalia, in: Proceedings of 5th International Conference on Cellular Automata for Research and Industry, Geneva, Switzerland, LNCS 2493, 2002.
[29] K. Nagel, J. Esser, M. Rickert, Large-scale traffic simulations for transport planningAnnual Reviews of Computational Physics VII, 22, World Scientific, Singapore, 2000, pp. 151–202.
[30] K. Nagel, Cellular Automata models for transportation applications, in: Proceedings of 5th International Conference on Cellular Automata for Research and Industry, Geneva, Switzerland, LNCS 2493, 2002.
[31] K. Nagel, P. Stretz, M. Pieck, S. Leckey, T. Donnelly, C. Barret, TRANSIMS traffic flow characteristics, Transportation Research Board, 77th Annual Meeting, Washington, DC, 1998.
[32] M. Rickert, K. Nagel, M. Schreckenberg, A. Latour, Two lane traffic simulations using CA, Physica A 231 (1996) 534–550.
[33] R. Rodríguez Zamora, Using de Bruijn diagrams to analyze 1d CA traffic models, in: Proceedings of Sixth International Conference on Cellular Automata for Research and Industry, Amsterdam, Netherlands, LNCS 3305, 2004.
[34] B. Sadoun, An efficient simulation methodology for the design of traffic lights at intersections in urban areas, Simulation 79 (2003) 243–251.
[35] A. Sahraoui, R. Jayakrishnan, Microscopic-macroscopic models systems integration: a simulation case study for ATMIS, Simulation 81 (2005) 353–363.
[36] M. Schmidt, 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, USA, 2000.
[37] C. Tolba, D. Lefebvre, P. Thomas, A. El Moudni, Continuous and timed Petri nets for the macroscopic and microscopic traffic flow modeling, Simulation Modelling Practice and Theory 13 (5) (2005) 407–436.
[38] M. Treiber, A. Hennecke, D. Helbing, Congested traffic states in empirical observations and microscopic simulations, Physical Review E 62 (2000) 1805–1824.
[39] P. Wagner, K. Nagel, P. Wolf, Realistic Multi-line traffic rules for cellular automaton, Physica A 234 (1997) 687.
[40] G. Wainer, S. Borho, J. Pittner, Defining and visualizing models of urban traffic, in: Proceedings of the SCS 1st Mediterranean Multiconference on Modeling and Simulation, Genoa, Italy, 2004.
[41] G. Wainer, CD+: a toolkit to define discrete-event modelsSoftware Practice and Experience, vol. 32, no. 3, Wiley, 2002, pp. 1261–1306.
[42] G. Wainer, N. Giambiasi, *N*-dimensional Cell-DEVS, in: G. Wainer, N. Giambiasi (Eds.), Discrete Events Systems: Theory and Applications, vol. 12, no. 1, 2002, pp. 135–157.
[43] S. Wolfram, A New Kind of Science, Wolfram Media, 2002.

[44] B. Zeigler, T. Kim, H. Praehofer, Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, 2000.
[45] B. Zeigler, Y. Moon, D. Kim, G. Ball, The DEVS environment for high-performance modeling and simulation, IEEE Computational Science and Engineering 4 (3) (1997).