Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

DEFINING MODELS OF URBAN TRAFFIC USING THE TSC TOOL

Mariana Lo Tártaro
César Torres

Departamento de Computación
FCEN – Universidad de Buenos Aires
Planta Baja, Pabellón I, Ciudad Universitaria
Buenos Aires.(1428), ARGENTINA

Gabriel Wainer

Department of Systems and Computer Engineering
Carleton University
4456 Mackenzie Building, 1125 Colonel By Drive
Ottawa, ON, K1S 5B6, CANADA

ABSTRACT

ATLAS is a specification language defined to outline city sections as cell spaces. A static view of the city section to be analyzed can be defined and a modeler is able to define complex traffic models in a simple fashion. A compiler for this specification language (called TSC) was built. The language implements the ATLAS constructions as Cell-DEVS models. The rule generation for describing the traffic behavior is based on macro templates, entitling changes in the model implementation in a flexible way. The formal specification avoids a high number of errors in the developed application, and the problem solving time is highly reduced.

1 INTRODUCTION

Urban traffic analysis and control is a problem whose complexity makes difficult the analysis with traditional analytical methods. The degree of complexity of vehicle movement in urban centers is such that modeling and simulation techniques have been gaining popularity as analysis tool. Simulation entitles the study of particular problems, allowing providing solutions based on experimentation. Here, we present the results of a project to build modeling and simulation tools with this purpose.

The first stage of this project was devoted to define and validate a high level specification language representing city sections (Davidson and Wainer 2000a). This language, called ATLAS (Advanced Traffic LAnguage Specifications) focuses on the detailed specification of traffic behavior. The models are represented as cell spaces, allowing elaborate study of traffic flow according with the shape of a city section and its transit attributes. A static view of the city section can be easily described, including definitions for traffic signs, traffic lights, etc. A modeler can concentrate in the problem to solve, instead of being in charge of defining a complex simulation.

The constructions defined in this language are mapped into DEVS (Zeigler, Kim, and Praehofer 2000) and Cell-DEVS models (Wainer and Giambiasi 2001a). DEVS provides high performance for discrete-event systems simulation. It also provides a formal framework that can be used to validate and verify the models. Cell-DEVS was proposed to describe cell spaces as DEVS models with timing delays, improving the definition of the models using explicit delays.

A real system modeled using DEVS can be described as composed of atomic or coupled submodels. A DEVS atomic model is defined by:

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

Input external events in ***X*** are received in input ports. When an event arrives, the model executes the external transition function $\boldsymbol{\delta_{ext}}$ to produce a state change. Each state has an associated lifetime **ta**. When this time is consumed the internal transition function $\boldsymbol{\delta_{int}}$ is activated to produce internal state changes. The internal state **S** can be used to provide model outputs ***Y***, which are sent through the output ports. They are sent by the output function $\boldsymbol{\lambda}$, which executes before the internal transition.

A DEVS coupled model is defined as:

$$CM = < X, Y, D, \{Mi\}, \{I_i\}, \{Z_{ij}\} >.$$

Each coupled model consists of a set of **D** basic models $\mathbf{M_i}$ connected through input/output ports. The list of influencees $\mathbf{I_i}$ of a given model is used to determine the models to which outputs must be sent. These sets are used to build the translation function $\mathbf{Z_{ij}}$, in charge of translating outputs of a model into inputs for the others. An index of influencees is created for each model (I_i). For every *j* in the index, outputs of model M_i are connected to inputs in model M_j.

Cell-DEVS (informally described in Figure 1) allows defining cellular models that can be integrated with other

DEVS. Here, each cell of a space is defined as an atomic model. Transport and inertial delays are used to define the timing behavior of each cell explicitly. 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 specified as:

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

Each cell will use the **N** inputs to compute the future state **S** using the function **τ**. The new value of the cell is transmitted to the neighbors after the consumption of the delay function. **Delay** defines the kind of delay for the cell, and **d** its duration. This behavior is defined by the δ_{int}, δ_{ext}, **λ** and **D** functions.

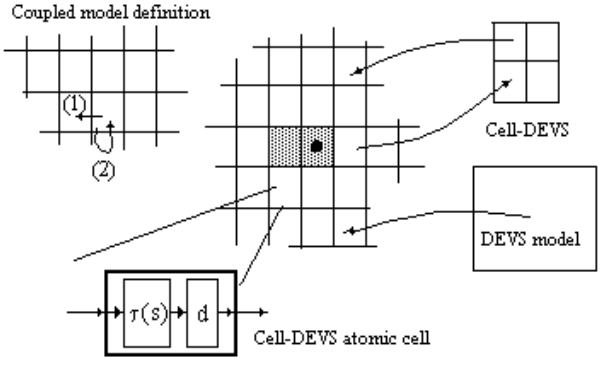


Figure 1: Informal Definition of Cell-DEVS

A Cell-DEVS coupled model is defined by:

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

A cell space **C** defined by this specification is a coupled model composed by an array of atomic cells with size $\{\mathbf{t_1 x...x t_n}\}$. Each cell in the space is connected to the cells defined by the neighborhood **N**. The cell space can be "wrapped", meaning that cells in a border are connected with those in the opposite one. Otherwise, the borders **B** should have a different behavior than the remaining cells. The **Z** function allows one to define the internal and external coupling of cells in the model. This function translates the outputs of output port *m* in cell *Cij* into values for the *m* input port of cell *Ckl*. The input/output coupling lists can be used to interchange data with other models.

The formal specifications for DEVS and Cell-DEVS were used to build the **CD++** tool (Rodriguez and Wainer 1999). This tool provides a specification language following the formal specifications described in this section.

ATLAS was defined as a set of constructions mapped into DEVS and Cell-DEVS models (Davidson and Wainer 2000b, Davidson and Wainer 2000c), whose behavior for each of the constructions was validated. Then, a compiler was built following the specifications. The compiler, called TSC (Traffic Simulator Compiler), generates code by using a set of templates that can be redefined by the user. In this way, the models can be mapped in different tools (avoiding version problems).

The following sections are devoted to show how to define traffic models using the language, focusing in the modeling problems. After, we present some results of the execution of simple models, defined earlier using the CD++ tool. This example was redefined using the TSC compiler, and the previously defined models were used to validate the compiled source code. Finally, several changes were made to the original example, allowing checking the efficiency in defining new models using the constructions defined by the compiler.

2 TSC CONSTRUCTIONS

ATLAS allows representing the structure of a city section defined by a set of streets connected by crossings. The language constructions define a static view of the model, which is considered to be built as grids composed of cells (Davidson and Wainer 2000a). ATLAS formal specifications were used to build the TSC language sentences. Following, we present the main constructions of ATLAS and its syntax in TSC.

a) Segments: they represent sections between two corners. Every lane in a given segment has the same direction (one way segments) and a maximum speed. They are specified as: Segments = { (p1, p2, n, a, dir, max) / p1, p2 ∈ City ∧ n, max ∈ ∧ a, dir ∈ {0,1} }, where **p1** and **p2** represent the boundaries of each segment (City = { (x,y) / x, y ∈ ***R*** }), **n** is the number of lanes, and **dir** represents the vehicle direction. The **a** parameter defines the shape of the segment (straight or curve, allowing to define the city shape precisely, and to include the exact number of cells), and **max** is the maximum speed allowed in the segment.

TSC syntax entitles defining the segments by delimiting them using the sentences **`begin segments`** and **`end segments`**. At least one segment must be defined, using the following syntax:

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

These values map the parameters mentioned previously, with **`shape: [curve|straight]`** and **`direction: [go|back]`**. Finally, **`parkType`** is used to define parking constructions, formally specified in the following paragraphs.

b) Parking: border cells in a segment can be used for parking, as seen in Figure 2. They are formally defined as: Parking = { (s, n1) / s ∈ Segments ∧ n1 ∈ {0,1} ∧ s = (c1, c2, n, a, dir, max) ∧ 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; if n1 = 1, on the right (lane n-1).

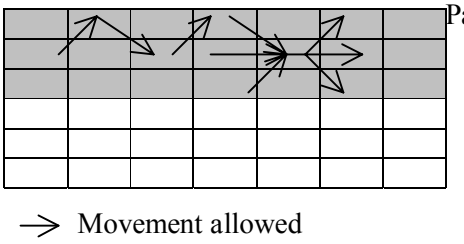


Figure 2: Parking Segments

The construction presented for Segments includes information for the parking segments. In this case,

```
parkType: [parkNone | parkLeft | parkRight |
parkBoth]
```

defines in which area of the segment a car can park.

c) Crossings: these constructions represent points in the plane where several segments intersect. They are specified as: Crossings = { (c, max) / c ∈ City ∧ max ∈ ∧ ∃ s, s' ∈ Segments ∧ s = (p1, p2, n, a, dir, max) ∧ s' = (p1', p2', n', a', dir', max') ∧ s ≠ s' ∧ (p1 = c ∨ p2 = c) ∧ (p1' = c ∨ p2' = c) }. Crossings are built as rings of cells with moving vehicles (Davidson and Wainer 2000b). A car in the crossing has higher priority to obtain the next position in the ring than the cars outside the crossing. In TSC, the definitions for crossings are delimited by the separators **`begin crossings`** and **`end crossings`**. Each sentence defines a crossing using the following syntax:

```
id = p, speed, tLight, crossHole, pout
```

Parameters ***p*** and ***speed*** represent *(p1,p2)* and *max* of the formal specification. ***Pout*** defines the probability of a vehicle to abandon the crossing, used to simulate random routing of different vehicles. The remaining parameters are related with specific types of crossings, and will be explained in the following paragraphs.

d) Traffic lights: crossings with traffic lights are defined as: TLCrossings = { c / c ∈ Crossings }. Here, *c* ∈ *TLCrossings* defines a set of models representing the traffic lights in a corner and the corresponding controller, depicted in Figure 3. Each of these models is associated with a crossing input. The model sends a value representing the color of the traffic light to a cell in the intersection corresponding to the input segment affected by the traffic light. The following qualifier is added to a standard crossing definition in TSC when a crossing must include traffic lights: **`tLight: [withTL|withoutTL].`**

e) Railways: they are built as a sequence of level crossings overlapped with the city segments. The railway network is defined by: RailNet = { (Station, Rail) / Station is a model, Rail ∈ RailTrack }, where RailTrack = { (s, δ, seq) / s ∈ Segments ∧ δ ∈ ∧ seq ∈ }. *RailNet* represents a set of stations connected to railways, thus defining

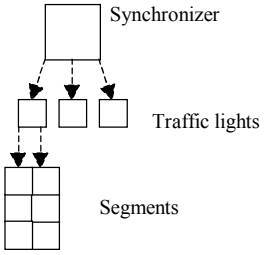


Figure 3: Traffic Lights Definition

a part of the railway network. *Railtrack* associates a level crossing with other existing constructions in the city section. Each element identifies the segment that is crossed (***s***) and the distance to the railway from the beginning of the section (***δ***). Finally, a sequence number (***seq***) is assigned to each level crossing, defining its position in the *RailTrack*. When a railway is defined in TSC, the **`begin railnets`** and **`end railnets`** act as separators. Each *RailNet* is defined using the following syntax:

```
id = (si, di) {,(si, di)}
```

where **`si`** defines an identifier of a segment crossed by the railway, and **`di`** defines the distance between the beginning of the segment s_i and the railway. The compiler automatically generates the sequence number.

f) Men at work: the construction defining men at work is specified by: Jobsite = { (s, ni, δ, #n) / s ∈ Segments ∧ s = (c1, c2, n, a, dir, max) ∧ ni ∈ [0, n-1] ∧ δ ∈ ∧ #n ∈ [1, n+1-ni] ∧ #n ≡ 1 mod 2 }. Here, each (*s*, *ni*, *δ*, *#n*) ∈ *Jobsite* is related with a segment where the construction works are being done. It includes the first lane affected (***ni***), the distance between the center of the jobsite and the beginning of the segment (***δ***), and the number of lanes occupied by the work (***#n***). These values are used to define a rhombus over the segment where vehicles cannot advance, as shown in Figure 4.

In TSC, the **`begin jobsites`** and **`end jobsites`** separators allow to define all the jobsites needed. Each jobsite is:

```
in t : firstlane, distance, lanes
```

In this case, **`firstlane`** defines the first lane affected by the jobsite, **`distance`** is the distance between the center of the jobsite and the beginning of the segment, and **`lanes`** is the number of lanes occupied.

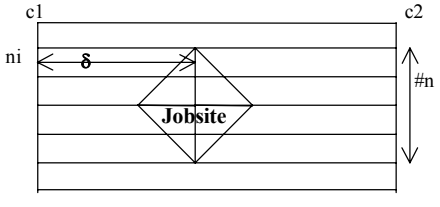


Figure 4: Segment with Men at Work

g) Traffic signs: they are defined by: Control = { (s, t, δ) / s ∈ Segments ∧ δ ∈ ∧ t ∈ {bump, depression, pedestrian crossing, saw, stop, school} }. Each tuple here identifies the segment where the traffic sign is used, the type of sign, and the distance from the beginning of the segment up to the sign. In TSC, **`begin ctrElements`** and **`end ctrElements`** delimits the control elements, with:

```
in t : ctrType, distance
```

being the definition for each sign. Here, **`ctrType: [bump | depression | intersection | saw | stop | school]`** defines the different signs. The **`distance`** parameter defines the distance to the beginning of the segment. An extension of this construction allows us to define potholes, whose size is one cell. The definition of these elements is done using the **`begin holes`** and **`end holes`** separators. Each hole is defined as:

```
in t : lane, distance
```

A pothole can also be included in a crossing. Previously defined in the Crossings paragraphs, **`crossHole: [withHole|withoutHole]`** defines if a crossing contains a pothole or not.

h) Experimental frameworks: experimental framework constructions provide inputs and outputs to the city section to be studied. They are associated with segments receiving inputs, or those used as outputs, and are defined as:

InputSegments = { s / s = (p1, p2, n, a, dir, max) ∧ s ∈ Segments ∧ [(dir = 0 ∧ (∃ v ∈ **N** : (p2,v) ∈ Crossings)) ∨ (dir = 1 ∧ (∃ v ∈ **N** : (p1,v) ∈ Crossings))] }
OutputSegments = { s / s = (p1, p2, n, a, dir, max) ∧ s ∈ Segments ∧ [(dir = 0 ∧ (∃ v ∈ **N** : (p1,v) ∈ Crossings)) ∨ (dir =1 ∧ (∃ v ∈ **N**: (p2,v) ∈ Crossings))] }

3 DEFINING TRAFFIC MODELS USING TSC

TSC takes an input written in ATLAS and provides, as output, a specification in Cell-DEVS written to be executed in the CD++ tool. Running a model in CD++, we can analyze in detail the traffic flow in the chosen area. TSC was built based on a set of templates that defines how to code the output rules. The set of templates can be changed in runtime. In this way, independence of the development tool used can be achieved. For instance, the syntax in the CD++ tool could be changed, or other tools allowing the definition of Cell-DEVS models could be used. In these cases, we just have to change the templates used to generate the model behavior.

Let us suppose that we want to define a city section specified by the map depicted in Figure 5. As we can see, we have defined 6 segments, connected by 3 crossings. For instance, the segment *t6* includes 2 lanes. We include a jobsite, three holes and two control elements.

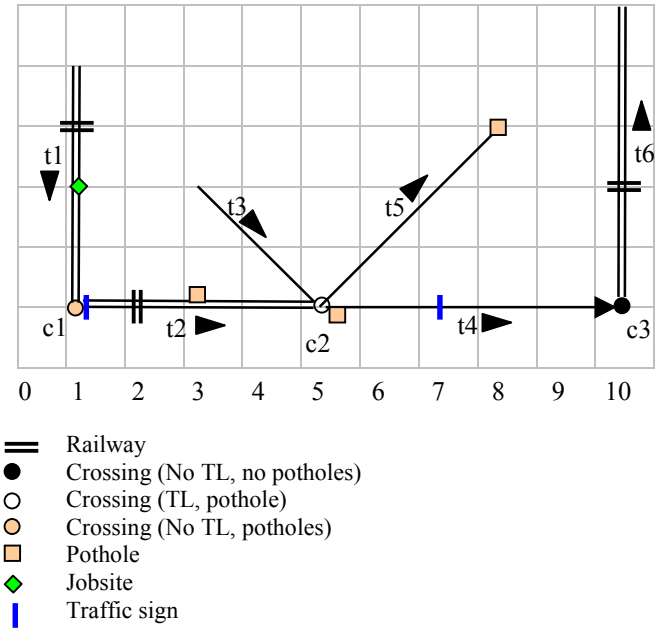


Figure 5: Shape of a City Section

The definition of this city section in TSC has been defined in the Figure 6. The constructions defined in that figure are used as inputs for the compiler, which translates them into DEVS and Cell-DEVS models.

The first step carried out by the compiler is the invocation to a parser that processes the map using the definitions presented in the previous section. Using the size definitions of each construction, a Cell-DEVS model of the given size is built. According to the number of input/output segments in a crossing, the number of cells needed for the crossings are created. Traffic lights, railways and other constructions are used to modify the basic behavior defined for the segment and crossing Cell-DEVS models, according to the definitions.

The parser is also used to validate the city map. The map should include at least one segment, and a segment cannot have the same beginning and end. More than one crossing cannot be defined at the same point. The railways, potholes, control signals and jobsites should be defined within an existing segment, and cannot trespass the segment boundaries. The railways cannot cross the segments close to their borders. Finally, 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).

Once the components are generated and validated, every lane in a segment is linked to an input cell in a corresponding crossing, using the rules defined in (Davidson and Wainer 2000b; Davidson and Wainer 2000c). In this stage the tool checks that each crossing has at least one input and one output segment.

The Cell-DEVS model resulting of translating the specification of Figure 6 can be seen in the Figure 7. This figure shows parts of the Cell-DEVS specification generated by the compiler. The code here showed is executable in the CD++ tool.

```
begin segments
t1=(1,5),(1,1),2,straight,go,60, parkNone
t2=(1,1),(5,1),2,straight,go,60, parkRight
t3=(3,3),(5,1),1,straight,go,40, parkNone
t4=(5,1),(10,1),1,straight,go,40, parkNone
t5=(5,1),(8,4),1,curve,go,40, parkNone
t6=(10,8),(10,1),2,straight,back,60, parkLeft
end segments

begin crossings
  c1=(1,1),11, withoutTL, withHole, .75
  c2=(5,1),12, withTL, withoutHole, .9
  c3=(10,1),13,withoutTL,withoutHole, .8
end crossings

begin railnets
rn1 = (t1,1),(t2,1),(t6,2)
end railnets

begin jobsites
in t1 : 1,2,1
end jobsites

begin holes
in t2 : 1,2
in t4 : 1,0
in t5 : 1,3
end holes

begin ctrElements
in t2 : stop,0
in t4 : saw,2
end ctrElements
```

Figure 6: Definition of the Section in TSC

The first lines in the figure show the *Top* model generated by the compiler. It includes one submodel for each of the models defined in the specification, an experimental framework (*Generators* and *Consumers*), traffic light controllers, and a definition of the translation function for coupled models. Then, we show the Cell-DEVS specification for the segment *t4* and the crossing *c3*. In the segment *t4* we only show the coupled model definition. This includes the parameter definition (size, type of delays, borders, neighborhood shape), and the external couplings (defined using the specification for crossings connected to the segment).

The same parameters are included for the *c3* crossing. In this case we also show the specification for the rules generated to define the traffic behavior in the crossing. The first rule shows the arrival of a vehicle to the cell (from the previous cell in the crossing or from an external segment). After, a '1' (representing a vehicle) is sent through the port *y-t-room*. In this way, the segment connected to the cell will know that a car is willing to leave the crossing and move to the segment. The second rule represents a car leaving the present cell. In this case, the *send* function informs that the cell is now empty. The third rule also represents a car leaving the present cell, but, in this case, the *send* function informs that the cell is busy, because the previous cell in the crossing is busy (and cars in the crossing have higher priority over new arriving cars).

```
[TOP]
components: t1 t2 t3 t4 t5 t6 c1 c2 c3
components: rn11@RailNet t2tl@TrafficLight
components: t5Cons@Consumer t1Gen@Generator
components: rn12@RailNet t6Cons@Consumer
components:  t3tl@TrafficLight  t3Gen@Generator
components: c2stl@SynchroTrafficLight
components: rn1@SynchroRailNet rn10@RailNet
link : y-t-train0bt@rn11 x-vt-train01@t2
...
link : y-vt-train2@rn1 x-if-train@rn12

[t4]
type : cell
width : 5
height : 1
delay : transport
border : nowrapped
neighbors : t4(0,-1) t4(0,0) t4(0,1)
in : x-c-vehicle00 x-c-room04
out: y-c-vehicle04 y-c-room00
link : x-c-vehicle00 x-c-vehicle@t4(0,0)
link : y-c-room@t4(0,0) y-c-room00
link : x-c-room04 x-c-room@t4(0,4)
link : y-c-vehicle@t4(0,4) y-c-vehicle04
localtransition : t4-lane0-rule
...
[c3]
type : cell
width : 3
height : 1
delay : transport
border : nowrapped
neighbors : c3(0,-1) c3(0,0) c3(0,1)
in : x-t-vehicle2 x-t-room0 x-t-room1
out: y-t-room2 y-t-vehicle0 y-t-vehicle1
link : x-t-vehicle2 x-t-vehicle@c3(0,2)
link : y-t-room@c3(0,2) y-t-room2
link : x-t-room0 x-t-room@c3(0,0)
link : x-t-room1 x-t-room@c3(0,1)
link : y-t-vehicle@c3(0,0) y-t-vehicle0
link : y-t-vehicle@c3(0,1) y-t-vehicle1
localtransition : c3-cellIn-rule

[c3-cellIn-rule]
rule : {1 + send(1, y-t-room)} 13 { (0,0) = 0
and ( (0,-1) = 1 or x-t-vehicle = 1 ) }
rule : {0 + send(0, y-t-room)} 13 { (0,0) = 1
and (0,1) = 0 and (0,-1) = 0 }
rule : {0 + send(1, y-t-room)} 13 { (0,0) = 1
and (0,1) = 0 and (0,-1) = 1 }
...
```

Figure 7: CD++ Source Code Generated for the Example

4 AN APPLICATION EXAMPLE

In (Díaz, Vázquez and Wainer 2001), we showed the definition of a city section using the tool CD++. ATLAS constructs were used as basic definitions to build the models, and the tool was used to develop them according to the specifications.

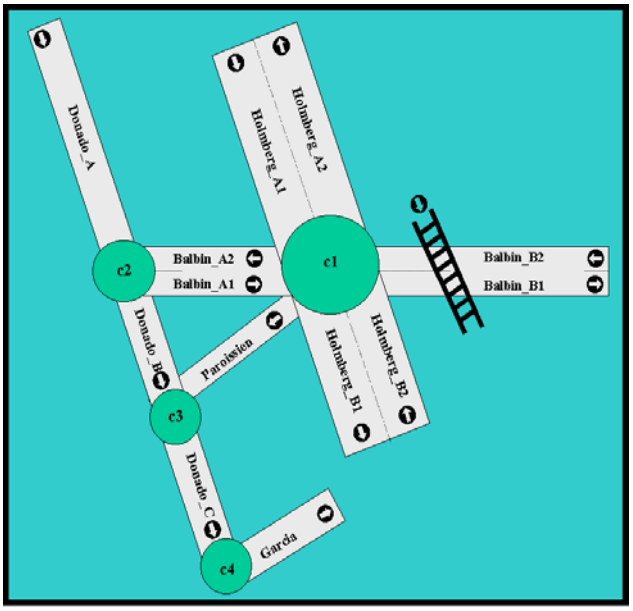


Figure 8: A Sketch of the Area to be Simulated

The sector defined is a part of residential neighborhood of the city of Buenos Aires, Argentina, in which traffic flow is non-significant, even in the peak hours. Nevertheless, the complexity in the city trace of the area produces frequent traffic jams. The section includes a park, a railway, several one way streets, dead ends, and avenues (one with four lanes in each direction). In several of these streets, parking is allowed, while in others it is forbidden. The following figure shows a sketch of the section.

As it can be seen in Figure 8, there is a train station close to the crossing *c1*. The level crossing uses an automatic barrier that closes when a train is approaching to the station. Therefore, while a train is stopped at the station, the barrier is closed. The city section was defined using the TSC language, and based on the specification, a model of traffic in the area was executed. This sector can be specified as shown in Figure 9.

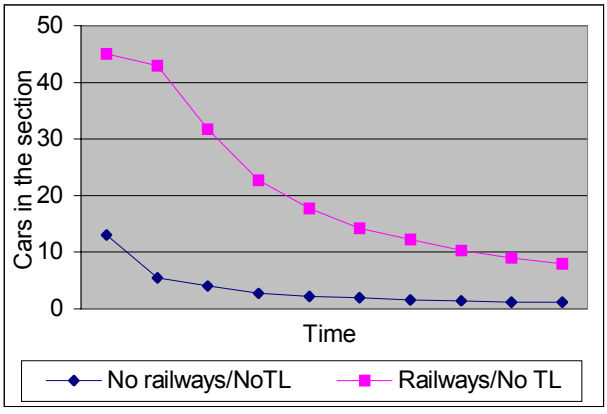
The first tests executed the specification using the present behavior for the level crossing. The execution results of the models generated by the compiler were compared with those obtained by coding the model specification by hand, previously defined in (Díaz, Vázquez and Wainer 2001). The results obtained were similar with a significant reduction in the coding effort. The present specification (24 line length) generated source code of over 1000 lines of Cell-DEVS specifications for this example. This shows how the use of this specific purpose language can improve the definition of traffic models. In a previous study (Wainer and Giambiasi 2001b), the use of Cell-DEVS showed important improvements when compared against modeling the same kind of problems using other simulation languages, making even more significant the final gains in development times.

```
begin segments
Donado_B=(7,16),(11,25),1,straight,go,40,
parkNone
Donado_A=(2,1),(7,16),1,straight,go,40,
parkNone
Donado_C=(11,25),(14,34),1,straight,go,40,
parkNone
Balbin_A1=(7,16),(22,16),2,straight,go,60,
parkNone
Balbin_A2=(7,16),(22,16),2,straight,back,60,
parkNone
Paroissien=(11,25),(22,16),1,straight,back,40
,parkNone
Garcia=(14,34),(21,31),1,straight,go,40,
parkNone
Holmberg_A1=(17,2),(22,16),4,straight,go,60,
parkNone
Holm-
berg_A2=(17,2),(22,16),4,straight,back,60,par
kNone
Holm-
berg_B1=(22,16),(24,26),2,straight,go,60,park
None
Holmberg_B2=(22,16),(24,26),2,straight,back,
60,parkNone
Balbin_B1=(22,16),(40,16),2,straight,go,60,
parkNone
Balbin_B2=(22,16),(40,16),2,straight,back,60,
parkNone
end segments

begin crossings
c1=(22,16),30, withoutTL, withoutHole, .7
c2=(7,16),30, withoutTL, withoutHole, .65
c3=(11,25),30, withoutTL, withoutHole, .8
c4=(14,34),30, withoutTL, withoutHole, .8
end crossings

begin railnets
Balbin = (Balbin_B1,8),(Balbin_B2,11)
end railnets
```

Figure 9: Definition of the Area in TSC

The simulations represent an execution of 10 minutes simulated time. The experimental framework generated different input rates according to the actual traffic flow in the area. After running the first tests, we eliminated the level crossing (supposing that a bridge is built in the area), and repeated the tests. As a result, we could analyze the effect of the railways in the area. Figures 10 and 11 show the results obtained in both cases.

The first graphic in Figure 10 shows the number of cars in the section. In the first minutes we can see a traffic jam in the area due to the closing of the barrier (although the traffic flow generated was constant throughout the simulation). The influence of the railway can be clearly seen in the second figure, which shows the number of cars leaving the Balbin-B1 segment. We can see that the traffic flow in that segment improves, eliminating the problem caused by the level crossing. This obstacle makes that about half of the cars cannot leave the area.



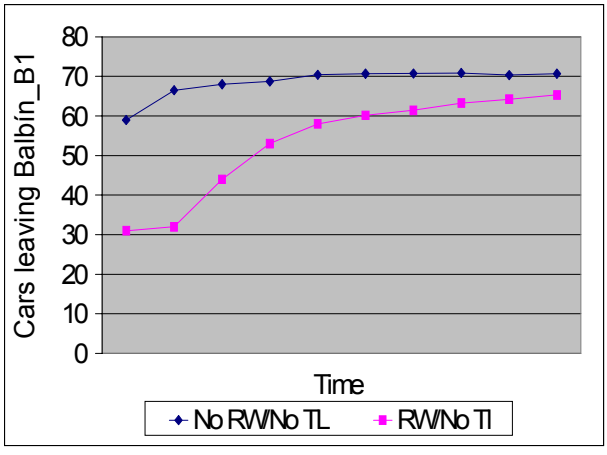
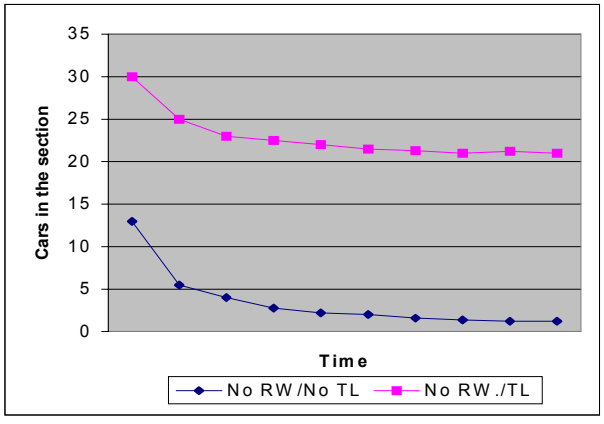


Figure 10: Execution Results in the Area



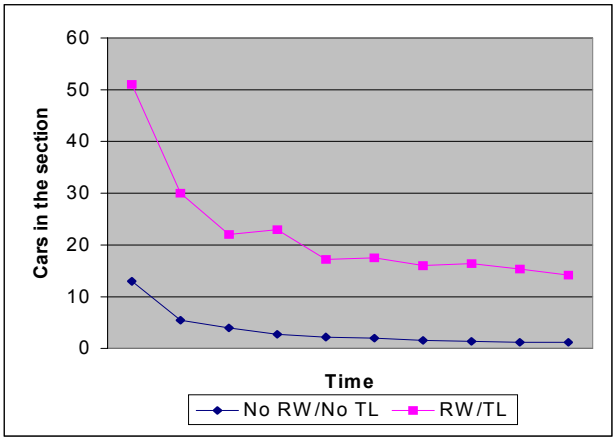


Figure 11: Execution Results Adding Traffic Lights

A second test changed the specifications for crossings *c1* and *c2*, by including non synchronized traffic lights. The following lines were modified in the crossing section:

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

In Figure 11 we can see that the number of cars in the city section is incremented when the traffic lights are included.

Traffic lights are useful in ordering the traffic. Nevertheless, in this case, they are not synchronized, making the number of cars in the area higher than without traffic lights. This is even worse for this particular case, where the number of cars in the area is so small that is not constrained by the ordering of the traffic light.

5 CONCLUSION

We showed how to use the TSC compiler to define city sections to analyze traffic conditions. The language allows defining a static view of a city section by 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.

Using this approach we could obtain the following advantages:

- **Efficiency**: by describing a high level specification of the problem to be modeled, we have reduced the effort needed in developing the application. The tool automatically builds the structure for coupled models, generates rules for atomic models, and takes care of validating the DEVS specifications. 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**: besides the ideas presented in the article, we needed to change some of the rules defined originally in ATLAS, due to implementation constrains of CD++. Therefore, two sets of templates were created. One of them followed pre-

cisely the ATLAS specifications, assuming that the underlying Cell-DEVS tools were able to implement them. A second set modified these specifications to be executed in CD++. The same original specifications were used in both cases, generating DEVS models adapted to each of the tools.

- **Abstraction**: the specifications were translated into executable models, without needing to write a line of source code. In this way, a traffic analyzer can focus in the problem solving task, avoiding implementation or low level details.

At present different research activities are related with this project. The first one considers using a GIS with information of city sections to be translated automatically into ATLAS constructions, entitling modeling and simulation of traffic models based on geographical information. This set of tools will provide a GUI for the system, and will be used to provide automatic validation of the constructions based on the map definitions.

A second set of activities is related with the extension of the TSC compiler to include other constructions already defined in ATLAS (for instance, big size vehicle movement, dynamic routing, congestion avoiding). Also, different control techniques are being applied to the traffic light controllers. Finally, we are analyzing performance issues by running the models in a parallel version of the CD++ toolkit.

ACKNOWLEDGMENTS

This work was partially supported by ANPCYT (National Agency of Science and Technology, Argentina; research project No. 11-04460) and NSERC (Natural Sciences and Engineering Research Council, Canada).

REFERENCES

Davidson, A., Wainer, G. 2000a. ATLAS: a language to specify traffic models using Cell-DEVS. Technical Report 00-003, Departamento de Computación, FCEN/UBA. Argentina. Submitted.

Davidson, A., Wainer, G. 2000b. Specifying control signals in traffic models. In *Proceedings of AI, Simulation and Planning in High Autonomous Systems, AIS'2000*. Tucson, Arizona. U.S.A.

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

Díaz, A., Vázquez, V., Wainer, G. 2001. Application of the ATLAS language in models of urban traffic. In *Proceedings of the 34th Annual Simulation Symposium*. Seattle, Washington, U.S.A.

Rodríguez, D., Wainer, G. 1999. New Extensions to the CD++ tool. In *Proceedings of Summer Computer Simulation Conference*. Chicago, U.S.A.

Wainer, G., Giambiasi, N. 2001a. Timed Cell-DEVS: modeling and simulation of cell spaces. In *Discrete Event Modeling & Simulation: Enabling Future Technologies*. Ed.: H. Sarjoughian, F. Cellier. Springer-Verlag.

Wainer, G., Giambiasi, N. 2001b. Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation*. Vol. 76, No. 1. January 2001.

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

AUTHOR BIOGRAPHIES

MARIANA LO TÁRTARO received her B.Sc. M. Sc. degree at the Universidad de Buenos Aires in 1997 and 2000 respectively. At present she works as a software developer in Computer Associates (Buenos Aires, Argentina). Her e-mail address is <Mariana.LoTartaro@ca.com>.

CÉSAR TORRES received his B.Sc. M. Sc. degree at the Universidad de Buenos Aires in 1996 and 2000 respectively. At present he works as a software developer in Computer Associates (Buenos Aires, Argentina). His e-mail address is <Cesar.Torres@ca.com>.

GABRIEL WAINER received his M. Sc. (1993) and Ph.D. degree (1998) from the Universidad de Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. He is currently Assistant Professor at the SCE Department, Carleton University (Ottawa, Canada). He was Assistant Professor at the Computer Sciences Department of the Universidad de Buenos Aires, Argentina, being a Visiting Research Scholar at the University of Arizona, Tucson, AZ. He has been the PI of several research projects. He is author of a book on real-time systems and another on Discrete-Event simulation. He is a member of the Board of Directors of the Society for Computer Simulation International, and a member of a group on standardization of DEVS modeling tools. His e-mail and web addresses are <Gabriel.Wainer@sce.carleton.ca> and <www.sce.carleton.ca/faculty/wainer.html>.