COMPILING TRAFFIC MODELS USING TSC

Mariana Lo Tártaro César Torres

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

Gabriel Wainer

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

E-mail: gwainer@sce.carleton.ca

KEYWORDS
Traffic models, DEVS, Cell-DEVS, cellular models.

ABSTRACT

ATLAS is a specification language defined to outline city sections as cell spaces. The goal is that a modeler is able to define complex traffic models in a simple fashion. A compiler for this specification language (called TSC) was built. We present the compiling techniques used to allow flexible definition of the models. The tools can be used to generate intermediate code based on templates, which can be interpreted by different DEVS modelling tools. We show how the code is generated, and how it can be applied to existing modelling tools.

INTRODUCTION

Urban traffic analysis and control is a problem whose complexity is difficult to be analyzed with traditional tools. Modelling and simulation approaches are gaining popularity as analysis tool. Here, we present the results of a project to build modelling and simulation tools with this purpose. The first stage of this project was devoted to define and validate a high level specification language defined to represent city sections (Davidson and Wainer 2000a). This language, called ATLAS (Advanced Traffic LAnguage Specifications) represents traffic as cell spaces, allowing elaborate study of flow according with the shape of a city section. The constructions defined in this language are mapped into DEVS (Zeigler et al. 2000) and Cell-DEVS models (Wainer and Giambiasi 2001). DEVS models provide high performance for discrete-event systems simulation. The **CD++** tool (Rodríguez and Wainer 1999) provides a specification language following the formal specifications for DEVS and Cell-DEVS. ATLAS was defined as a set of constructions mapped into DEVS and Cell-DEVS models (Davidson and Wainer 2000b, Davidson and Wainer 2000c). Once these models were validated, a compiler was built. 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). 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. Here, we present the main constructions of ATLAS and its syntax in TSC.

- **Segments**: they represent sections between two corners. In TSC, they are defined using the sentences `begin segments` and `end segments`. Each segment is defined as: `id = p1,p2,lanes,shape,direction,speed,parkType`, with `shape: [curve|straight]` and `direction: [go|back]`.
- **Parking**: border cells in a segment can be used for parking. `parkType: [parkNone|parkLeft|parkRight|parkBoth]` defines in which area of the segment a car can park.
- **Crossings**: they are points in the plane where several segments intersect. In TSC 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`
- **Traffic lights**: the following qualifier is added to a standard crossing definition in TSC when a crossing must include traffic lights: `tLight: [withTL|withoutTL]`.
- **Railways**: they are built as a sequence of level crossings overlapped with the city segments. When a railway is defined in TSC, the `begin railnets` and `end railnets` act as separators. Each railnet by **id = (s_1, d_1) {,(s_i, d_i)}**, where s_i is the identifier of a segment crossed by the railway, and d_i the distance to the segment s_i.
- **Men at work**: the `begin jobsites` and `end jobsites` separators allow to define all the jobsites needed. Each jobsite is: `in t : firstlane, distance, lanes`. `Firstlane` defines the first lane affected by the jobsite, `distance` is the distance to the beginning of the segment, and `lanes` the number of lanes occupied.
- **Traffic signs**: in TSC, `begin ctrElements` and `end ctrElements` delimits the control elements, with: `in t : ctrType, distance` as the definition for each control sign. Here, `ctrType: [bump|depression|intersection|saw|stop| school]` are the different control signs. `distance` defines the distance to the beginning of the segment. An extension of this construction allows us to define Potholes, which can also be included in a crossing.

CODE GENERATION IN TSC

TSC is built such that the code generation can be configured. TSC is built as a set of templates defining the way of encoding the output code according to the input specification. In this way, the compiler can be adapted to different DEVS modelling tools. The models generated can run using the CD++ tool (Rodríguez and Wainer 1999), and two different set of templates were used, proving the feasibility of the approach. CD++ lets the user to define DEVS and Cell-DEVS, were used to define executable models. DEVS atomic models can be defined as C++ functions. The template file contains one template corresponding to each ATLAS construction. Every template contains the sentences that must be generated in the simulation file. They are organized in different sections. The template format is the following:

```
|--template identif--|          |--before links--|
|--top components --|           line_1 ...
line_1 ... line_n               |--links--|
|--top ports--|                 line_1 ...
line_1 ...                      |--before zones--|
|--top links--|                 line_1 ...
line_1 ...                      |--zones--|
|--before neighbors--|          line_1 ...
line_1 ...                      |--before rules--|
|--neighbors--|                 line_1 ...
line_1 ...                      |--rules--|
|--before ports--|              block_1 ... block_n
line_1 ...                      |--after rules--|
|--ports--|                     |--end template--|
line_1 ...
```

Figure 1: Template Definition

|--top components --| are added to the top coupled model
|--top ports--| define input/output ports for the top model
|--top links--| internal/external couplings of the top model
|--before neighbors--| lines included before the neighborhood definitions.
|--neighbors--| neighborhood shape used for the model.
|--before ports--| lines included before the definition of the ports corresponding to the Cell-DEVS
|--ports--| input/output ports for the Cell-DEVS
|--before links--| lines included before the definition of the internal and external couplings.
|--links--| internal and external couplings of the model
|--zones--| define special behavior in Cell-DEVS zones
|--rules--| define the behavior of each cell corresponding to the Cell-DEVS translated from the original construct.

After the header definition, each **line$_i$** defines the definitions that will be written in the output files. These lines should follow the syntax of the DEVS tool being used. For instance, they can contain macro variables translated when the models are generated. Besides, each **block$_i$** defines a set of rules corresponding to the template, with the following syntax: **[*identif_block*] line$_1$ ... line$_n$**

Each of the original constructs generates a Cell-DEVS using these descriptions. Different macro variables are used, because every component contains basic information that is repeated: size of the cell space, basic behavior, position of railways, etc. Therefore, we have defined a set of macro variables that are replaced by the corresponding value. When TSC finds a macro variable, it will be replaced by the corresponding value. Macro variables start and finishes with an **&** sign. For instance, **&IDENTIF&** is replaced by the corresponding identifier for this element. As a result, the high level specification in ATLAS is translated step by step to a Cell-DEVS definition without needing writing any code. Let us suppose, for instance, that the following city section is defined:

```
begin segments
t1 = (0,0),(10,0),2,straight,go,200,200,parkNone
end segments
```

This specifications defines one segment with two lanes, and when the two-lane template is used, the intermediate code seen in the Appendix is generated. In this example, the final results of generating the constructions for the t1 model are the following:

```
[TOP]
components : t1Gen@TSCGenerator t1Cons@TSCConsumer
t1
out : qtyOutSimu_t1Cons
link : y_t_car0@t1Gen x_ge_car00@t1
link : y_t_car1@t1Gen x_ge_car10@t1
link : y_co_car09@t1 x_t_car0@t1Cons
link : y_co_car19@t1 x_t_car1@t1Cons
link : quantity@t1Cons qtyOutSimu_t1Cons

[t1]
type : cell    width : 4      height : 2
delay : transport     border : nowrapped
neighbors:t1(1,-1) t1(1,0) t1(1,1)  t1(0,0) t1(0,1)
neighbors:t1(-1,-1) t1(-1,0) t1(-1,1) t1(0,-1)
in : x-ge-car00  x-ge-car10
out: y-co-car03  y-co-car13
link : x-ge-car00 x-ge-car@t1(0,0)
link : x-ge-car10 x-ge-car@t1(1,0)
link : y-co-car@t1(0,3) y-co-car03
link : y-co-car@t1(1,3) y-co-car13
localtransition : t1-segment2-lane0-rule
...

[t1-segment2-lane0-rule]
rule : 1 21 { (0,0) = 0 and (0,-1) = 1 }
rule : 1 21 {(0,0)=0 and (-1,-1)=1 and (-1,0)=1 and
(0,-1)=0}
rule : 0 21 {(0,0)=1 and (0,1)=0 }
rule : 0 21 {(0,0)=1 and (-1,0) = 0 and (-1,1)= 0 }
rule : {(0,0)} 21 { t }
...
```

Figure 2: Resulting Definition for t1 in CD++

The translation begins defining the type of template used. In this case, we are using a 2-lane segment with a generator in the start of the segment. The top components will use the specification of the model to get the model identifier (*t1* in this case), that will be used in the following replacements of the identifier macro. Then, the components of the top model are defined. In this case, we have one generator (predefined as a DEVS model included in the CD++ toolkit). The *link* statements are used to define the internal and external couplings, according to the ATLAS definitions. In this case, the generators output ports (whose names are generated using macros) are connected to the input ports of the Cell-DEVS representing the two lane model. The same procedures are repeated for the couplings in the end model. Finally, we show the generation of the Cell-DEVS model created using the Segment-2Lane template. We can see that a one line specification automatically expanded into a specifications with more than 40 lines, defining the detailed behavior of this model. In this way, definition time for traffic models can be highly reduced.

CONCLUSION

The TSC compiler allows to define city sections, with a static view of including different components. This approach provides an application-oriented specification language, which allows the definition of complex traffic behavior using simple rules for a modeler. The models are formally specified, avoiding a high number of errors in the application, thus reducing the problem solving time. The high level specification of the problem to be modeled reduces the developing efforts, as the 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.

Different sets of templates can be used to generate traffic specifications using different tools (or different rules with the same toolkit). These can be 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.

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.

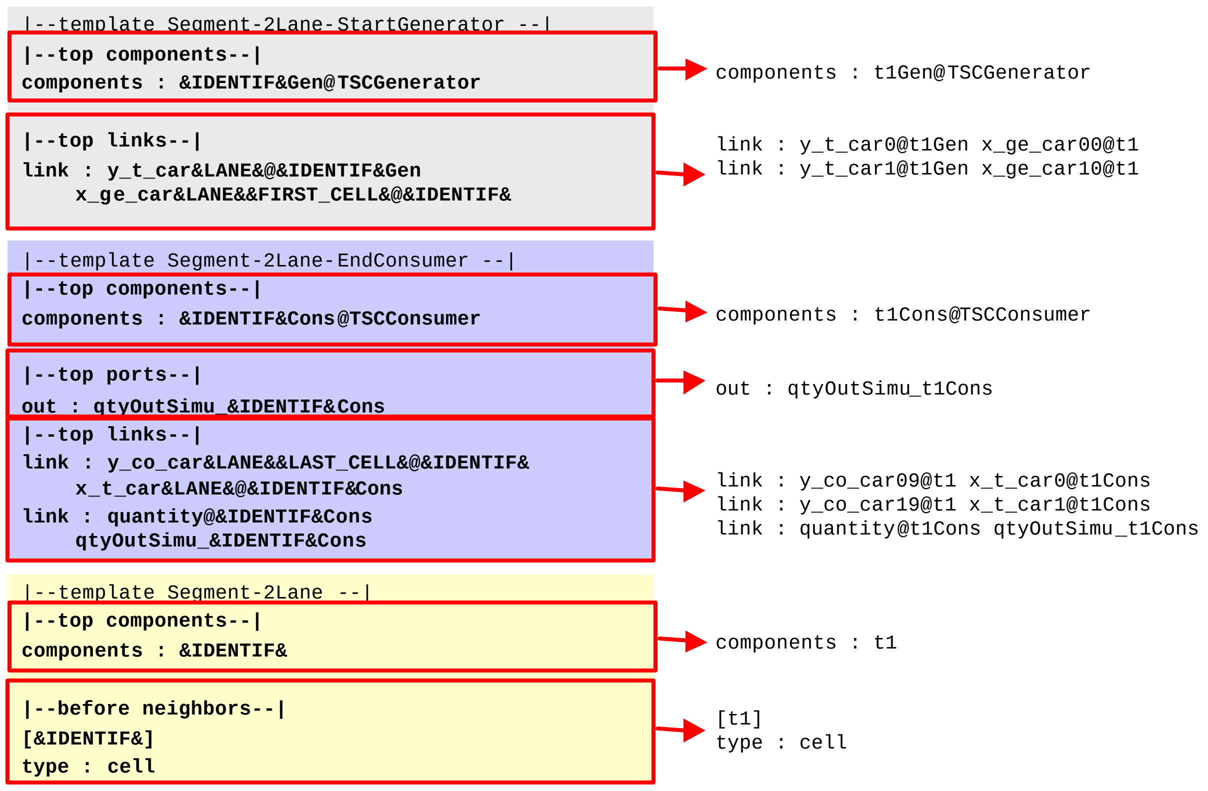
Davidson, A., Wainer, G. 2000c. Specifying truck movement in traffic models using Cell-DEVS. In *Proceedings of the 33rd Annual Simulation Symposium*. Washington, D.C. U.S.A.

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.

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

APPENDIX



```
|--template Segment-2Lane-StartGenerator --|
|--top components--|
components : &IDENTIF&Gen@TSCGenerator
```
```
components : t1Gen@TSCGenerator
```
```
|--top links--|
link : y_t_car&LANE&@&IDENTIF&Gen
     x_ge_car&LANE&&FIRST_CELL&@&IDENTIF&
```
```
link : y_t_car0@t1Gen x_ge_car00@t1
link : y_t_car1@t1Gen x_ge_car10@t1
```
```
|--template Segment-2Lane-EndConsumer --|
|--top components--|
components : &IDENTIF&Cons@TSCConsumer
```
```
components : t1Cons@TSCConsumer
```
```
|--top ports--|
out : qtyOutSimu_&IDENTIF&Cons
```
```
out : qtyOutSimu_t1Cons
```
```
|--top links--|
link : y_co_car&LANE&&LAST_CELL&@&IDENTIF&
     x_t_car&LANE&@&IDENTIF&Cons
link : quantity@&IDENTIF&Cons
     qtyOutSimu_&IDENTIF&Cons
```
```
link : y_co_car09@t1 x_t_car0@t1Cons
link : y_co_car19@t1 x_t_car1@t1Cons
link : quantity@t1Cons qtyOutSimu_t1Cons
```
```
|--template Segment-2Lane --|
|--top components--|
components : &IDENTIF&
```
```
components : t1
```
```
|--before neighbors--|
[&IDENTIF&]
type : cell
```
```
[t1]
type : cell
```

Figure 3: Translation Based on Templates