Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

ANALYZING STATIC AND TEMPORAL PROPERTIES OF SIMULATION MODELS

Mamadou Kaba Traoré

LIMOS CNRS UMR 6158
Blaise Pascal University
Campus des Cézeaux, 63177 Aubière cedex, FRANCE

ABSTRACT

This paper shows how a simulation model can be specified so that its static and temporal properties can be formally analyzed. The approach adopted is based on the integration of Formal Methods (FMs) and the DEVS paradigm. FMs are known to allow symbolic manipulation and reasoning, while DEVS is known as being a well-establish Modeling and Simulation (M&S) framework. Combining them makes it possible to develop rigorous proofs of the properties of simulation models as regard to design and use requirements. This paper focuses on the so-called atomic specification. Static aspects of the model are captured with the Z formalism, while dynamic aspects are expressed in first order logic. The specification is supported by the Z/EVES tool. A case study is exhibited.

1 INTRODUCTION

Challenging issues in M&S are mostly related to metrics of models, i.e. properties expected from these models. Notable ones concern credibility, such as verification, validation and accreditation (Balci 1998, Sargent 2001), confidence in reuse and composition (Davis and Anderson 2003, Overstreet et al. 2002), level of interoperability, etc. Most of these metrics should be evaluated before the implementation stage. Consequently, there is an appeal for symbolic reasoning and the requirement to turn metrics issues into the general question of formal analysis. This could lead to some insights which are very hard and costly to reveal at the implementation level.

This paper deals with a methodological approach to make simulation models amenable to formal analysis. Section 2 recalls the DEVS M&S paradigm. Section 3 recalls the main ingredients of FMs. Section 4 presents the integration of DEVS and the Z formal method as a solution. It gives details on the Z-DEVS modeling approach. An example is shown in section 5, and concluding remarks are provided in section 6.

2 DEVS MODELING PARADIGM

DEVS is based on systems theory and deals with atomic and coupled models, the former ones being the most basic entities that describe both systems structure and behavior. The latter are obtained by composing atomic models into larger models. An atomic model is described by a set of inputs, outputs, states, and governing functions. A coupled model presents the same external interfaces as do atomic models. By coupling together output ports of one model to input ports of another, outputs are transmitted as inputs and acted upon by the receiving model. The semantics of DEVS models are supported by simulators and coordinators. Atomic models are mapped onto simulators and coupled models are mapped onto coordinators. All simulators and coordinators adhere to a generic message protocol that allows them to coordinate with each other to execute the simulation. A root coordinator is used to handle the simulation cycle, first initiating the simulation by an initialization message, and then leading the simulation loop by scheduling messages to provide the correct synchronization for the subordinate coordinator (or simulator). Abstract algorithms have been defined in (Zeigler et al. 2000) to characterize what simulators and coordinators have exactly to do.

An atomic model is defined in DEVS as the structure shown by figure 1. This model is supposed to be at any time in some state $s \in S$ for a time period defined by $e = ta(s)$. If the elapsed time e expires without an external event ($x \in X$) occurring, the system outputs the value $y = \lambda(s)$, and changes to state $\delta_{int}(s)$. If not (i.e., an external event $x \in X$ has occurred before e expires) the system only changes to state $\delta_{ext}(s, e, x)$. In both cases, the model is in a new state and the same conditions apply.

DEVS doesn't involve specific abilities for symbolic reasoning and proofs. To analyze the properties of a DEVS model, one must deal with its code and run classic accreditation methods. They are known to be costly in time and do not reveal all the properties mentioned before.

$IOS = <X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$, where
X is the set of input values,
S is the set of states,
Y is the set of output values,
$\delta_{int} : S \rightarrow S$ is the internal transition function,
$\delta_{ext} : Q \times X \rightarrow S$ is the external transition function,
$Q = \{ (s,e) / s \in S, 0 \le e \le ta(s)\}$ is the total state set,
e is the time elapsed since last transition,
$\lambda : S \rightarrow Y$ is the output function,
$ta : S \rightarrow R_{0,\infty}$ is the time advance function.

Figure 1: DEVS Atomic Model

3 FORMAL METHODS

FMs are known to allow symbolic reasoning (Clarke and Wing 1996, Kuhn et al. 2003, Lamsweerde 2000), with the following major capabilities: getting rid of ambiguities and inconsistencies while specifying a system, finding hidden properties of a specification, and automating the translation from specification to code.

Formal specification is the process of describing a system and its desired properties, using a language with a mathematically-defined syntax and semantics. The kinds of system properties might include functional behavior, timing behavior, performance characteristics, or internal structure. But also, a formal specification must involve rules for inferring useful information from the specification (the proof theory). Model checking relies on building a finite model of a system and checking by an exhaustive state space search that a desired property holds in that model (McMillan 1992). Theorem proving is a technique where both the system and its desired properties are expressed as logic-based formulas, in terms of axioms and inference rules (Clarke and Wing 1996). Theorem proving is the process of finding a proof of a property from these axioms and rules, and possibly derived definitions and intermediate lemmas.

FM can split into five paradigms: history-based, transition-based, state-based, functional, and operational (Lamsweerde 2000). However, there is no FM specifically designed for, or adapted to M&S. Ad-hoc use of FMs in M&S enterprises can be very costly. That's why we aim at building an M&S framework that integrates FMs so that efforts must not be repeated to take benefit of their reasoning abilities for finding properties of simulation models. To overcome the diversity of models, the DEVS well-established M&S paradigm has been considered as the theoretical foundation (Zeigler 1976).

4 MERGING DEVS AND FMS

A little work has been done in merging M&S and FM (Kuhn et al. 2003, Stevenson 2003). What is required is a logic characterization of structure and behavior of models in a way such that a hierarchy of verification and proof can be built through the levels of the characterization process.

We can give formal semantics to simulation models by introducing a mapping from DEVS to a FM in a way suggested by (Paige 1997). Once a DEVS model is translated into a specification in a formal language, all the analysis and manipulations are performed on the formal specification. The clear advantage of this approach is that it is possible to reuse all the previous work on the FM, for analyzing simulation models. The structure and the behavior of a model are completely known for DEVS atomic and coupled models. Moreover, DEVS is closed under coupling; in other terms, any coupled model can be translated into an atomic one. That is why we focus here on atomic models, even if extensions can be considered latter to deal with coupled models without the need to convert them into their atomic counterpart.

State-based FMs, which focus on specifying the behavior of sequential systems, perfectly fit our need to turn an atomic model into a logical specification: states of the model are described in terms of sets, relations and functions, and state transitions are given in terms of pre- and post-condition.

4.1 ZDEVS Modeling

We have defined the Z-DEVS formalism, combining DEVS, the Z specification language (a state-based method), and first order logic. It directly maps to the formal representation used by the Z/EVES theorem prover (Saaltink 1997). Thus, we can take benefit of automatic techniques provided by this formal analysis tool: syntax and type checking, schema expansion, precondition calculation, domain checking, and general theorem proving. The Z formalism has been chosen for its expressive power in logic, its wide dissemination and proven effectiveness, and more important, the availability of free supporting tools (notably Z/EVES).

The main ingredient in Z is the concept of schema (generally represented as a box), i.e. a piece of specification which links data (defined in the upper part of the box) to logical predicates (defined in the lower part of the box) that apply to them. Data are stored in variables (the names of input variables end in a question mark, those of output variables end in an exclamation mark ; the value of a variable after being updated is indicated by a prime sign). A schema A can be used by another schema B in two ways: (1) B only reads the values of the variables defined in A; this must be declared in B with the Ξ schema inclusion operator; or (2) B modifies the values of the variables defined in A; this must be declared in B with the Δ schema inclusion operator. In this work, we are dealing with the standard Z specification paradigm (Spivey 1998), but object-oriented extensions exist (Object-Z) (Smith 2000) and supporting tools are under development (ZIMOO) (Friesen et al. 1998).

The mapping strategy from DEVS to Z-DEVS relies on a computational architecture and a temporal pattern. The architecture is used to find out the static properties of the simulation model, while the temporal pattern is used to reveal its temporal properties.

4.2 Static Structure

The computational architecture (shown in figure 2) identifies autonomous data and operations of DEVS model and simulator, and turns both into schemas. This architecture is made of two encapsulated open boxes, one for the atomic DEVS model (the internal box) and the other for its simulator (the external box). Both involve autonomous data (closed boxes) and operations (octagons), some of which must be user-defined and some other are pre-defined. Data are composed of variables upon which some constraints may be defined. The X and Y data are shared with the environment of the architecture (that's why they are placed at the border of both open boxes). Operations are defined by logical predicates which may change the values of the data variables. Dotted arrows indicate which operations influence which data (or operations).

The model part is made of a diary (D), a pre-defined schema which keeps the elapsed time variable e and the time-to-next-event variable tN. It also involves schemas that represent the state space (S) and all the functions defined in DEVS (δ_{int}, δ_{ext}, λ, and ta). In addition, there is a need for a stimulate operation to deal with the required reactivity of the simulator, and an init operation to depict how the state space has to be initialized.

The simulator part contains a clock (C) that handles the time of last event, and three operations which correspond to the receipt of *-message, i-message and x-message as defined in DEVS.

4.3 Temporal Structure

The static structure of the architecture is accompanied by a finite-state-automata-style pattern (shown in figure 3) that will be used to generate the temporal structure of the computational model. The nodes represented are: a given state (s), possibly reachable as an initial state, and the two possible next states of the model. At the receipt of a i-message, the init operation is performed, then the system enters in a given state and then the time advance operation is performed. When the elapsed time is zero, the *-scheduling operation is performed, then the system enters in a new state defined by the internal transition function, and then the time advance operation is performed. If an x-message is received, then the stimulate operation is performed and the system enters in a new state defined by the external transition function, and then the time advance operation is performed.

Temporal properties are usually tackled with model checking, i.e. an exploration of the structure of the state space of the model. The state space of a DEVS model being basically infinite, the idea is to group states into equivalence classes so that the infinite state is reduced into a finite one. The temporal pattern defined here gives the way for such an abstraction process.

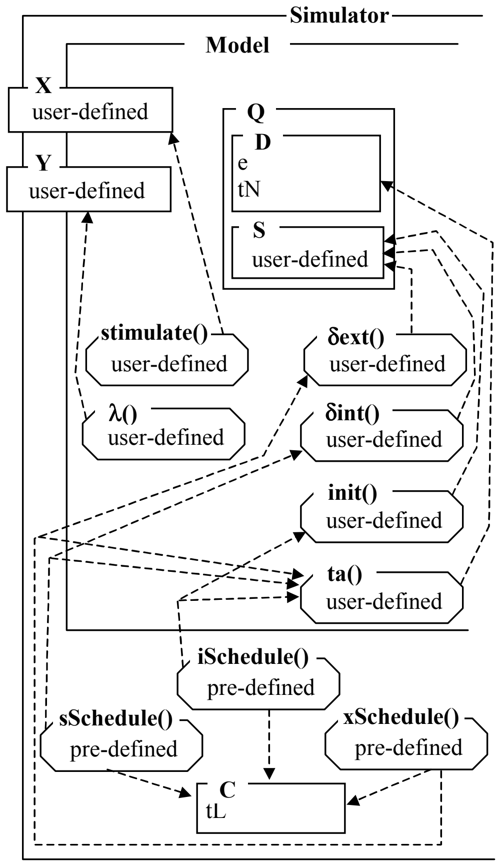

Figure 2: Computational Model

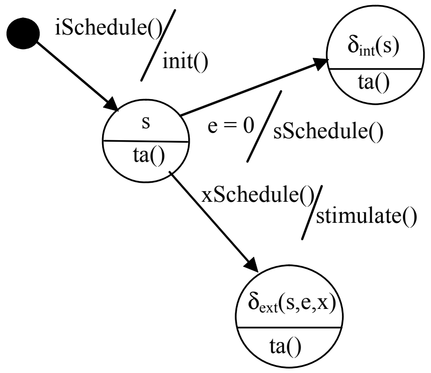

Figure 3: Semantics of Computational Model

5 CASE STUDY

Let’s consider a simple academic example: the University Bus System (UBS) (Zeigler 1984). The example has been flavored with some additional details to make it more illustrative. Basically, the UBS is made of a bus that shuttles between a downtown station and a university station, providing transportation service to specific users. A user enters the model when he lines up in a station. He exits from the model when he gets off the bus at a station. Figure 4 shows possible transitions in the model. Situations are linked by dashed lines, an the corresponding values of the selected state variables are indicated. The bus can be traveling from one station to the other (no stop is allowed between the stations), or stopped at a station (for unloading and loading operations). Depending on the fact that the bus is empty or not, the unloading operation is skipped or not when the bus arrives at a station (all the users in the bus, if any, are supposed to get off). Also, depending on the fact that some users are waiting or not at a station, the loading operation is performed or not when the bus arrives at this station (some users may not get in the bus, since its capacity is limited). There is no schedule time that the bus driver must follow. In other terms, the unloading, loading, and traveling operations are performed successively without any break between them. In addition, the same average speed is always kept between stations.

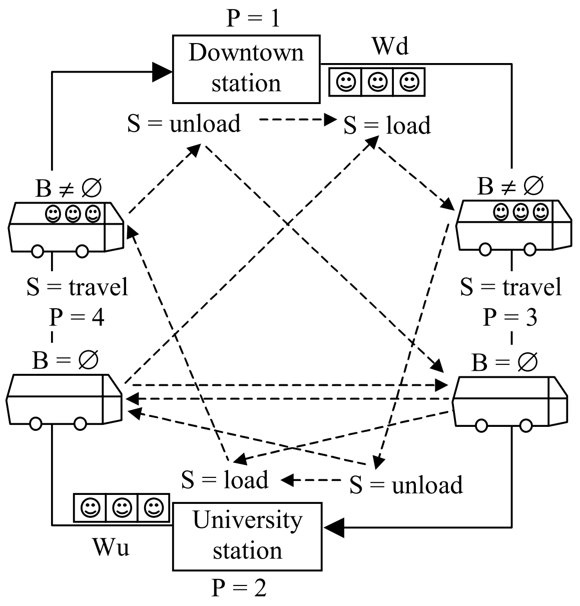

Figure 4: University Bus System

The Z-DEVS specification can be headed by the definition of all user-defined types that have at least one representative variable in the model. Abstract types, as well as concrete types can be considered in a model. Hence, the modeler can choose the level of abstraction at which formal analysis is required.

The abstract set USER represents the bus users. The STATUS type characterizes what the bus is currently doing. Also, parameters of the model must be declared and their values defined. An axiomatic box is used for this purpose instead of a schema box. Axiomatic boxes are used to introduce global variables and to specify constraints on their values (while the variables declared within a schema are local). The UBS parameters are: the capacity of the bus (α), the time to travel from downtown station to university station (β1), the time to travel in the reverse direction (β2), the time needed by a user to get in the bus (ε), and to get from the bus (ν). Infinity is represented by ω.

[USER]
STATUS ::= *unload* | *load* | *travel* | *traveling*

$\alpha : \mathbb{N}$
$\beta 1 : \mathbb{N}$
$\beta 2 : \mathbb{N}$
$\varepsilon : \mathbb{N}$
$\nu : \mathbb{N}$
$\omega : \mathbb{N}$

$\alpha = 40$
$\beta 1 = 60$
$\beta 2 = 65$
$\varepsilon = 1$
$\nu = 2$
$\omega = 30000$

The elapsed time must be greater than or equal to zero. Throughout the entire specification, the set of integers has been used as time base, assuming that any other time base can be interpreted in terms of integers.

Diary
$e : \mathbb{N}$
$tN : \mathbb{N}$

$e \geq 0$

X = {hello, where_in} and Y = {bye, where_out}. Hello (respectively bye) indicates that a user is entering (respectively leaving) the system. Where_in (respectively where_out) indicates the related station (1 for downtown station, and 2 for university station). The system can be entered and left only through one of these two stations.

Wd is the sequence of users waiting at the downtown station. Wu is the sequence of users waiting at the university station. B is the sequence of passengers in the bus. P ∈ {1, 2, 3, 4} is the current position of the bus (1: downtown station, 2: university station, 3: from downtown to the university, 4: from the university to downtown). St ∈ {“load”, “unload”, “travel”} is the current status of the bus. σ is used to compute the time advance.

Input

$hello: USER$

$where\_in: \mathbb{N}$

$where\_in \in \{1, 2\}$

Output

$bye: USER$

$where\_out: \mathbb{N}$

$where\_out \in \{1, 2\}$

State

$Wd: \text{seq } USER$

$Wu: \text{seq } USER$

$B: \text{seq } USER$

$P: \{1, 2, 3, 4\}$

$St: STATUS$

$\sigma: \mathbb{N}$

The external event is simply removed from the input of the model and stored in the adequate waiting line.

ExternalTransition

$\Xi Input$

$\Xi Diary$

$\Delta State$

$where\_in = 1 \Rightarrow Wd' = Wd \frown \langle hello \rangle$

$where\_in = 2 \Rightarrow Wu' = Wu \frown \langle hello \rangle$

$\sigma' = \sigma - e$

The rules for internal transition are the following:

- *If the bus travels empty towards the downtown station, and users are waiting at that station, then a stop is required at the next stage for loading (hence, the unloading stage is skipped).*
- *But if no one is waiting at that station, then the travel continues towards the university station.*
- *If the bus is not empty while traveling towards downtown, then a stop is required for unloading.*
- *During the unloading operation, passengers in the bus must get off one by one.*
- *If all the passengers get off the bus and no user is waiting at the station, then the bus can go on.*
- *But if there are users waiting at the station, then they can start getting in the bus.*
- *Users must get in the bus one by one.*
- *If the bus is full, it can start its travel to the next station (even some users are still waiting).*

Similar rules stand from the university station perspective (Wd is replaced by Wu, P = 1 by P = 2, P = 3 by P = 4, and P = 4 by P = 3).

InternalTransition

$\Delta State$

$P = 1 \wedge St = unload \wedge B = \langle\rangle \wedge Wd = \langle\rangle$

$\Rightarrow P' = 3 \wedge St' = travel \wedge \sigma' = \beta 1$

$P = 1 \wedge St = unload \wedge B = \langle\rangle \wedge Wd \neq \langle\rangle$

$\Rightarrow St' = load \wedge \sigma' = \varepsilon$

$P = 1 \wedge St = unload \wedge B \neq \langle\rangle$

$\Rightarrow B' = tail\ B \wedge \sigma' = \varepsilon$

$P = 1 \wedge St = load \wedge \#B < \alpha \wedge Wd = \langle\rangle$

$\Rightarrow P' = 3 \wedge St' = travel \wedge \sigma' = \beta 1$

$P = 1 \wedge St = load \wedge \#B < \alpha \wedge Wd \neq \langle\rangle$

$\Rightarrow B' = B \frown \langle head\ Wd \rangle \wedge \sigma' = \nu$

$P = 1 \wedge St = load \wedge \#B = \alpha$

$\Rightarrow P' = 3 \wedge St' = travel \wedge \sigma' = \beta 1$

$P = 4 \wedge B = \langle\rangle \wedge Wd = \langle\rangle$

$\Rightarrow P' = 3 \wedge \sigma' = \beta 1$

$P = 4 \wedge B = \langle\rangle \wedge Wd \neq \langle\rangle$

$\Rightarrow P' = 1 \wedge St' = load \wedge \sigma' = \varepsilon$

$P = 4 \wedge B \neq \langle\rangle$

$\Rightarrow P' = 1 \wedge St' = unload \wedge \sigma' = \nu$

$P = 2 \wedge St = unload \wedge B = \langle\rangle \wedge Wu = \langle\rangle$

$\Rightarrow P' = 4 \wedge St' = travel \wedge \sigma' = \beta 2$

$P = 2 \wedge St = unload \wedge B = \langle\rangle \wedge Wu \neq \langle\rangle$

$\Rightarrow St' = load \wedge \sigma' = \varepsilon$

$P = 2 \wedge St = unload \wedge B \neq \langle\rangle$

$\Rightarrow B' = tail\ B \wedge \sigma' = \varepsilon$

$P = 2 \wedge St = load \wedge \#B < \alpha \wedge Wu = \langle\rangle$

$\Rightarrow P' = 4 \wedge St' = travel \wedge \sigma' = \beta 2$

$P = 2 \wedge St = load \wedge \#B < \alpha \wedge Wu \neq \langle\rangle$

$\Rightarrow B' = B \frown \langle head\ Wu \rangle \wedge \sigma' = \nu$

$P = 2 \wedge St = load \wedge \#B = \alpha$

$\Rightarrow P' = 4 \wedge St' = travel \wedge \sigma' = \beta 2$

$P = 3 \wedge B = \langle\rangle \wedge Wu = \langle\rangle$

$\Rightarrow P' = 4 \wedge \sigma' = \beta 2$

$P = 3 \wedge B = \langle\rangle \wedge Wu \neq \langle\rangle$

$\Rightarrow P' = 2 \wedge St' = load \wedge \sigma' = \varepsilon$

$P = 3 \wedge B \neq \langle\rangle$

$\Rightarrow P' = 2 \wedge St' = unload \wedge \sigma' = \nu$

The diary is simply updated with the σ value.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{TimeAdvance}} \\
\Xi State \\
\Delta Diary \\
\hline
tN' = tN + \sigma \\
\hline
\end{array}
$$

A passenger exiting the bus, also exits the system at one of the two stations. Initially, the bus is empty, located downtown, and ready to load users. No one is waiting neither at the downtown station, nor at the university station. Stimuli must be specified consistently with X.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{OutputFunction}} \\
\Xi State \\
\Delta Output \\
\hline
P \in \{1, 2\} \\
B \neq \langle\rangle \\
bye' = head\,B \\
where\_out' = P \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{Init}} \\
State \\
\hline
P = 1 \\
B = \langle\rangle \\
Wu = \langle\rangle \\
Wd = \langle\rangle \\
St = load \\
\sigma = 0 \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\hline
\multicolumn{1}{l}{\textit{Stimulate}} \\
\Delta Input \\
h?: USER \\
w?: \mathbb{N} \\
\hline
hello = h? \\
where\_in = w? \\
\hline
\end{array}
$$

The current time of the model is the last time an event occurs in the model. It must always be lower than or equal to the time-to-next-event value kept in the diary.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{Clock}} \\
\Xi Diary \\
tL: \mathbb{N} \\
\hline
tL \leq tN \\
\hline
\end{array}
$$

Scheduling operations make use of the schema piping operator (…). The $|$ sign allows to define a schema in a horizontal form (instead of the usual vertical form).

$$
\begin{array}{l}
sSchedule \;\widehat{=} \\
[\Delta Diary; \Delta Clock; t?: \mathbb{N} \mid t? = tN \wedge tL' = t? \wedge e' = e - (t? - tL)] \\
\gg OutputFunction \gg InternalTransition \gg TimeAdvance \\
iSchedule \;\widehat{=} [\Delta Clock; t?: \mathbb{N} \mid tL' = t?] \gg Init \gg TimeAdvance \\
xSchedule \;\widehat{=} \\
[\Delta Diary; \Delta Clock; t?: \mathbb{N} \mid tL \leq t? \wedge t? \leq tN \wedge e' = t? - tL \wedge tL' = t?] \\
\gg Stimulate \gg ExternalTransition \gg TimeAdvance
\end{array}
$$

6 ANALYSIS

From the point where the Z-DEVS specification is complete, it can be used to analyze the model, check for inconsistencies and incompleteness, prove properties, and generate traces. These properties are common to any system, but they are specifically relevant to simulation systems due to the importance of these systems in decision making processes. Being able to find properties of simulation models without dealing with the code is the great benefit here (the ultimate goal is to build models of better quality, which improve the accuracy of decisions).

6.1 Static Properties

With the Z/EVES tool, the following capabilities can be browsed: (i) checking for syntax and type errors, (ii) checking for domain errors, (iii) checking for local and global inconsistencies, (iv) exploring specification by schema expansion, (v) exploring schemas' preconditions, (vi) checking for model invariants, (vii) proving a correct refinement, (viii) defining test scenarios and, (ix) proving test theorems. Hereafter, we emphasize major metrics-related issues and link them to these capabilities:

- *Consistency:* complex systems modeling can drive towards pitfalls (e.g., conflicting predicates in the model, omission of initializing conditions...). Partial checks for errors and inconsistencies must be performed repetitively through schemas specification until they are proven to be correct.
- *Verification:* it is nothing but the confrontation of a model with properties that the modeler is expecting. Methods like those in (McMillan 1992) can be used for Z-DEVS specifications.
- *Validation:* here, the model is confronted to properties established in the real system. E.g., the set of inputs and outputs collected over the time is a real-world property expected from the model (historical validation (Sargent 2001)).
- *Reuse and composability:* they relate to the set of conditions needed for a given purpose and that a

model must match to be used or reused in that context, typically in component-based large scale M&S. Again, the model has to be confronted to specific properties. Here, a key issue is how the conditions for reuse or composability have to be captured in terms of logical predicates.

6.2 Temporal properties

Let's consider a state class of the temporal pattern as aggregating the state of the bus (empty or not), the state of the waiting lines (empty or not), the status of the bus, and the direction of the current connection. Let's also partition the external transition of the temporal pattern into two cases: an external event occurs at downtown station (called $\delta_{ext}(1)$ and indicated by a blue arrow) or at university station ($\delta_{ext}(2)$, green arrow). Let's indicate internal transitions by a red arrows. Then, the pattern shown in figure 5 can be applied at each state class, generating therefore a graph which nodes are the set of state classes.

In figure 5, the S1 state depicts the case the bus is not empty and is moving from university to downtown, while the downtown waiting line is empty and the university waiting line is not. When $e = 0$ (i.e., nothing occurs until the travel is complete), the bus comes to the unloading stage (state S2). Otherwise, an external transition at downtown station ($\delta_{ext}(1)$) leads to state S2 (now, the downtown waiting line is not empty), or the like at university station ($\delta_{ext}(2)$) leads to state S1 again. The generation process ends if from each node start red, green and blue arrows. Figure 6 shows the temporal structure of the UBS, which can be submitted to any model checker.

7 CONCLUSION

We have proposed an approach that combines the DEVS well-established M&S paradigm with the Z formal paradigm. Then, ambiguities and inconsistency in requirements could be discovered early, when they can be corrected with much less expense than after code has been developed. Also, hidden properties can be revealed, using a theorem proving tool such as Z/EVES. The main benefit of this approach is the systematic way it provides for making simulation models amenable to formal analysis, hence allowing to find their properties without repeated and costly efforts required by ad-hoc use of formal methods. The ultimate goal is to build models of better quality and to increase our understanding of key concepts such as VV&A, reuse and composability.

The next step is the extension of this work to coupled models and their mapping on a functional FM paradigm, like concurrent processes (Graeme and Duke 1992). An important issue is the way the mapping from DEVS to Z-DEVS can be automated. Another issue is the way a model must be partitioned to allow the finding of temporal properties. There are on-going efforts tackling these issues.

REFERENCES

Balci, O. 1998. Verification, Validation, and Accreditation. *WSC'98*: 42-48.

Clarke, E., and J.M. Wing. 1996. Formal Methods: State of the Art and Future Directions. ACM Computing Surveys 28 (4). 626-643.

Davis, P.K., and R.H. Anderson. 2003. Improving the Composability of Department of Defense Models and Simulations. RAND Corporation.

Friesen, V., A., Nordwig, and M. Weber. 1998. Object-Oriented Specification of Hybrid Systems Using UMLh and ZimOO. *ZUM'98*: 328-346.

Graeme, S., and R. Duke. 1992. Specifying Concurrent Systems Using Object-Z. *15th Australian Computer Science Conf.*: 1-14.

Kuhn, D.R., D. Craigen, and M. Saaltink. 2003. Practical Application of Formal Methods in Modeling and Simulation. *SCSC'03*, Montreal, Canada. July 20-24.

Lamsweerde, A.V. 2000. Formal Specification: a Roadmap. *Conf. on the Future of Software Engineering*. 147-159.

McMillan, K.L. 1992. Symbolic Model Checking. An Approach to the State Explosion Problem. Ph.D. Thesis in Comp. Sc., Carnegie Mellon University.

Overstreet, C. M., R. E. Nance, and O. Balci. 2002. Issues in Enhancing Model Reuse. *Int. Conf. on Grand Challenges for Modeling and Simulation*, Jan. 27-31, San Antonio, Texas, USA.

Paige, R.F. 1997. A Meta-Method for Formal Method Integration. *4th Int. Symposium of Formal Methods Europe*. 473-494.

Saaltink, M. 1997. The Z/EVES system. Lecture Notes in Comp. Sc. 1212. Springer-Verlag: 72–85.

Sargent, R.G. 2001. Verification and Validation: Some Approaches and Paradigms for Verifying and Validating Simulation Models. *WSC'01*: 106-114.

Smith, G. 2000. The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers.

Spivey, J.M. 1998. The Z Notation : A Reference Manual. 2nd Ed., Prog. Research Group, University of Oxford.

Stevenson, D.E. 2003. From DEVS to Formal Methods: a Categorical Approach. *SCSC'03*, Montreal, Canada, 1-6. July 20-24.

Zeigler, B.P. 1976. Theory of Modelling and Simulation. Wiley & Sons, N.Y.

Zeigler, B.P. 1984. Multifacetted Modelling and Discrete Event Simulation. Academic Press Inc., London.

Zeigler, B.P., H. Praehofer, and T.G. Kim. 2000. Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems. 2nd Ed. Academic Press. Davis.

AUTHOR BIOGRAPHY

MAMADOU KABA TRAORE is Assistant Professor of Computer Science at the Blaise Pascal University of Clermont-Ferrand (France). His current research interests focus on formal methods and DEVS. His e-mail address is <traore@isima.fr> and his Web address is <http://www.isima.fr/~traore/>.

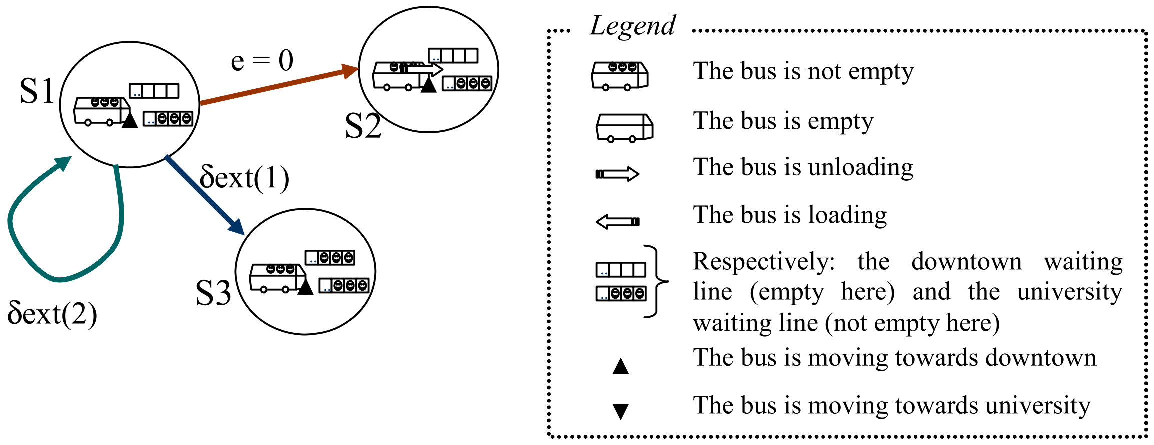
S1
e = 0
S2
δext(1)
S3
δext(2)
Legend
The bus is not empty
The bus is empty
The bus is unloading
The bus is loading
Respectively: the downtown waiting line (empty here) and the university waiting line (not empty here)
▲ The bus is moving towards downtown
▼ The bus is moving towards university

Figure 5: Pattern for UBS Temporal Structure

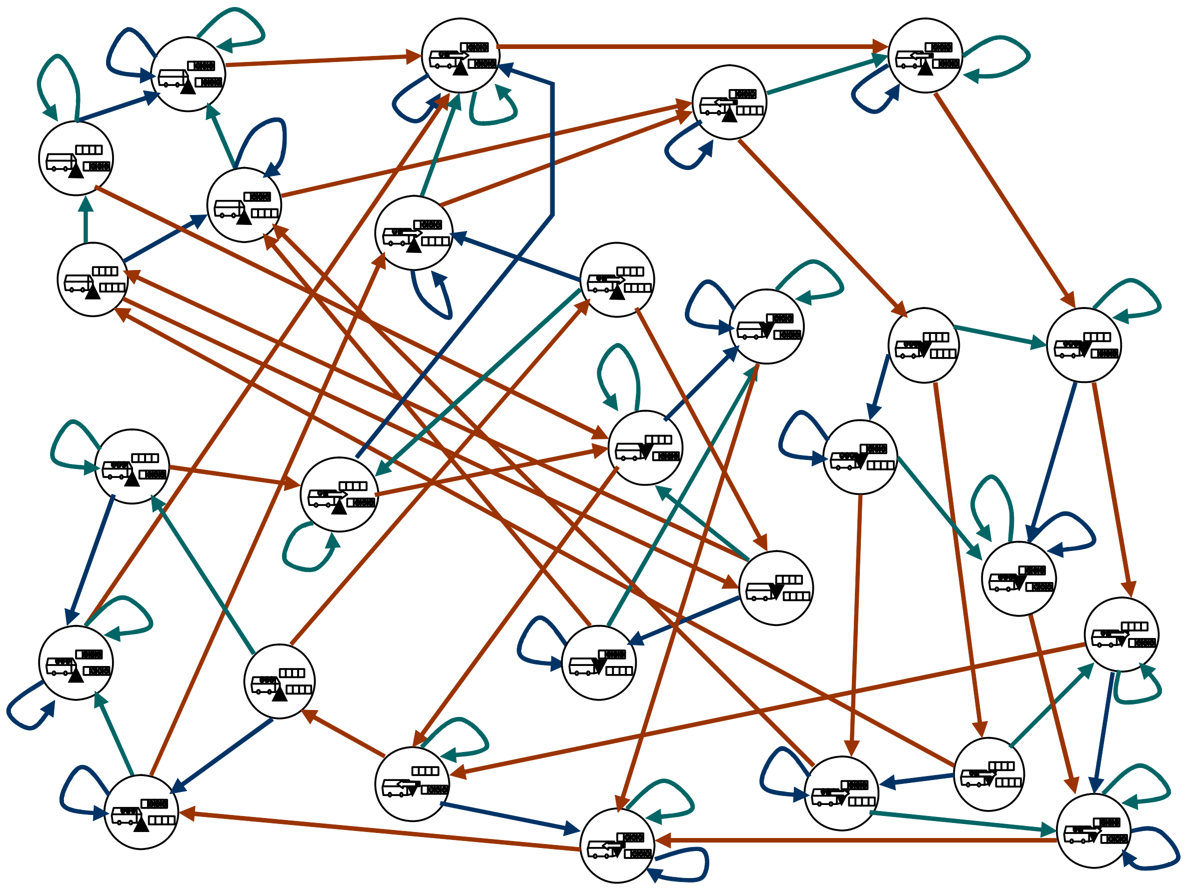

Figure 6: UBS Temporal Structure