RESTRICTING DEV-PROMELA WITH A HIERARCHY OF SIMULATION FORMALISMS

Aznam Yacoub
Maamar el Amine Hamri
Claudia Frydman
Aix-Marseille Université, CNRS, ENSAM, Université de Toulon,
LSIS UMR 7296
13397, Marseille, France
{aznam.yacoub, amine.hamri, claudia.frydman}@lsis.org

ABSTRACT

The DEv-PROMELA formalism is a new formalism that allows combining formal verification and discrete event simulation. In this paper, we propose to use a hierarchy of simulation formalisms to restrict the DEv-PROMELA language. From sequential machine to the most expressive DEv-PROMELA formalism, this hierarchy helps designers to design models and to analyze systems in a progressive manner.

Keywords: DEv-PROMELA, discrete event formalisms, DEVS, formal verification.

1 INTRODUCTION

Since the emergence of the Theory of Modelling and Simulation (Zeigler 1976), many discrete event formalisms, and especially many extensions and subclasses of DEVS, such as Parallel DEVS (Chow and Zeigler 1994), Generalized DEVS (Giambiasi and Carmona 2006), Real-Time DEVS, RTA-DEVS (Saadawi and Wainer 2010), Fuzzy DEVS, etc. were proposed and developed to model particular aspects of systems. Each formalism has different expressiveness capabilities, and choosing the best formalism and the best level of abstraction for the modelling of systems depends on the level of knowledge that the modellers have about these systems. Indeed, as stated by Zeigler et al. (2000), "Formalisms are proposed, and sometimes accepted, because they provide convenient means to express models for particular classes of systems and problems". However, the multiplicity of formalisms have sometimes blurred the relationship between models. For instance, some questions which were addressed to a timed model could no longer be handled by another timed model expressed in a different formalism. Furthermore, proving relationship between two models can be awkward, or needs specific works and manual mapping, for instance in the case of Timed Automata and DEVS (Giambiasi et al. 2003).

However, in order to make easier the understanding of basic concepts of discrete event modelling, different attempts of classification were made in the litterature. For instance, stating that each extension encapsulates the others, increasing the DEVS modelling capabilities, Wainer (2009), Giambiasi (2009), Hwang (2011), Hwang (2014) propose different hierarchies of discrete event formalisms, according to their expressiveness capabilities.

SpringSim-TMS/DEVS 2017, April 23-26, Virginia Beach, VA, USA

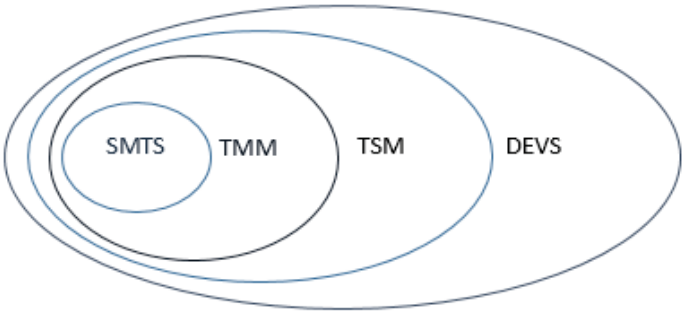

Figure 1: Hierarchy of discrete-event formalisms.

If such these attempts make more clear the relations between formalisms, they are underused, especially in modern verification and validation techniques, which try to combine several formal methods and simulation tools. In the most of case, these new strategies propose the mapping between two or more integrated formalisms (Abdulhameed et al. 2014, Maiga et al. 2012, Aliyu et al. 2016) without reusability of subformalisms or extended formalisms.

More recently, the DEv-PROMELA formalism (Yacoub et al. 2015, Yacoub et al. 2016) provides a methodology for deep integration between simulation formalisms and formal methods, especially between PROMELA (Holzmann 1991) and Classic DEVS (Zeigler 1976). DEv-PROMELA is built as a PROMELA extension which supports formal checking and discrete event simulation. Morphisms between PROMELA and DEv-PROMELA, on the one hand, and DEVS and DEv-PROMELA, on the other hand, ensure that the DEv-PROMELA models can be verified by model-checking, and that they can also be simulated using any Classic DEVS simulators. However, because a DEv-PROMELA model is both a PROMELA model and a DEVS model, it is possible to restrict the language in order to model DEVS subclasses.

In this paper, we propose to use the hierarchy developed in (Giambiasi 2009), and which is represented in the Figure 1, in order to restrict the DEv-PROMELA formalism. The objective is to facilitate the progressive modelling of complex discrete event systems using a unique syntactic language. Consequently, modelling a DEVS becomes easier by progressively relaxing constraints, and going from a subclass to another. The paper is organized as follows: Sections 2 and 3 introduces the DEv-PROMELA formalism and the simulation formalisms hierarchy. In Section 4, we show how the DEv-PROMELA language can be syntactically restricted in order to model DEVS subclasses.

2 DEV-PROMELA

2.1 Syntax

DEv-PROMELA (Yacoub et al. 2015) is presented as an extension of PROMELA for the modelling, verification and simulation of discrete event models. The PROMELA syntax is out of the scope of this paper. For more informations, readers are encouraged to read (Holzmann 2003). As an extension, the DEv-PROMELA language is built by adding missing concepts of simulation into the PROMELA language: events, time, transition and determinism. Therefore, DEv-PROMELA is built upon the syntax of PROMELA (Algorithm) in which some syntactic elements were added:

- a new abstract datatype for representing infinite and unbounded real values, especially needed for modelling time and data like floating numbers;

- events are modelled by integer constant values;
- each statement is prefixed with a meta-information that describes how the statement is performed by reacting to an event, or autonomously after an amout of time;

For instance, Program 1 shows a condition structure. The condition `x == 2` (l.5) is evaluated after *t1* units of time, and after the generation of the `changex` event.

Program 1 Simple condition structure in DEv-PROMELA.

```
1: int x,y = 2;
2: real t1,t2;
3:
4: if
5: :: [clt : t1 → emit : changex] ( x == 2 ) → x = 3;
6: :: [clt : t2 → emit : changey] ( y == 2 ) → y = 4;
7: fi;
```

2.2 Semantics

After modifying the syntax in order to allow modelling of DEVS concepts, DEv-PROMELA introduces the semantics of the DEVS abstract simulator.

Semantics of a DEv-PROMELA process A DEv-PROMELA process P with a set of statements *SL* is an automaton $T = (S_\tau, E, \delta_i, \delta_e, s_0, F)$ where

- $S_\tau = \{s_i = (t_s, id, l_1, ..., l_m, \in \mathbb{R}^+ \times \mathbb{N} \times \prod_{i=1}^{m} L_i \times \prod_{j=1}^{n} G_j \times \prod_{k=1}^{o} C_k)\}$ is the set of states. *id* is the identifier of the state related to the statement *sl* which defines it; the sets L_i (resp. G_j) are the sets of values of each local (resp. global) variable l_i (resp. g_j). C_k is the set of values of channels;
- E is the set of events; E contains at least the silent event denoted ε;
- $\delta_{int} : Q_f \to Q_0 \times E$ is the internal transition partial function; δ_{int} is partial because it can be not defined for all $q \in Q_f$ (especially when $ta(s) = \infty$);
- $\delta_{ext} : Q \times E \to Q$ is the external transition partial function; δ_e is partial because it can be not defined (especially when $e = \varepsilon$);
- s_0 is the initial state;
- F is the set of final states.

Moreover, the following sets are defined:

- $ta : \begin{cases} S_\tau \to \mathbb{R}^+ \\ s \mapsto t_s \end{cases}$ is the state lifetime function; the lifetime of each state is given by the delay before executing the next statement in the specifications;
- $Q = \{q = (s, dt), \forall s \in S_\tau\}$ such that $0 \leq dt \leq ta(s)$ is the set of total states; dt denotes the time elapsed in the state s;
- $Q_0 = \{q = (s, 0), \forall s \in S_\tau\} \subset Q$;
- $Q_f = \{q = (s, ta(s)), \forall s \in S_\tau\} \subset Q$.

Consider a DEv-PROMELA process P in a state s at time t, and the next statement l with its event descriptor. We can admit the process P is in fact in a state $q = (s, dt)$ (if dt denotes the elapsed time since the last event).

If l denotes an internal transition and if $t = ta(s)$, then the statement l is enabled. The event associated with the transition is emitted to all the other processes composing the program, before the transition is triggered, and the next event for the process P is defined by :

$$d_{e'} = getCurrentDate + ta(s')$$

with $((s',0),e') = \delta_{int}(s)$. If l denotes an external transition on an event e, then the transition is triggered only if the process receives the event e. In this case, denote t the date of the event e. The next state is given by $q' = \delta_{ext}(q,e)$ with $q' = (s',0)$. If δ_{ext} is not syntactically defined for $((s,dt),e)$, then the next state is given by $q' = (s,dt)$ and $ta(s) = t_s$. This behaviour corresponds to the semantics given by the DEVS abstract simulator.

The automaton underlying a DEv-PROMELA process is thus built by making transitions between control states (represented by each statement). The automonous event descriptors define the lifespan of the initial control state, while the reactive descriptors define external transitions between the initial control state and the end control state.

Semantics of a DEv-PROMELA program A DEv-PROMELA program P_r is a transition system $Tr = (S, \Lambda, \rightarrow)$ where

- $S = S_{\tau_0} \times ... \times S_{\tau_n}$ is the cartesian product of the set of states of each process;
- Λ is the union of the sets of all statements;
- $\rightarrow$ the set of transitions. $t \in \rightarrow$ and $t = s \rightarrow s'$ if:
  - given two states $s = (s_{p_i}, s_{q_j}, ...)$ and $s' = (s'_{p_i}, s'_{q_j}, ...)$. Then, $s \xrightarrow{l} s'$ with $l \in \Lambda$ if there exists an internal transition from s_{p_i} to s'_{p_i}, or from s_{q_j} to s'_{q_j}, ... and if it does not exist any other internal transition which can be triggered before the date t. In other words, the next event of P_r is the minimum value of all the next events of each process and external events. If two events can occur at the same date, the first proceeded event is that generated by the highest priority process;
  - given two states $s = (s_{p_i}, s_{q_j}, ...)$ and $s' = (s'_{p_i}, s'_{q_j}, ...)$. Then, $s \xrightarrow{l} s'$ with $l \in \Lambda$ if there exists an external transition from s_{p_i} to s'_{p_i}, or from s_{q_j} to s'_{q_j}, ... and if it does not exist any other external transition which can be triggered before the date t. If two events can occur at the same date, the first proceeded event is that generated by the highest priority process;
  - if $(s'_{p_i}, e) = \delta_{i_p}(s_{pi})$, then $s' = (s'_{p_i}, \delta_{e_q}(s_{q_j}, e), ...)$. Thus, s' is the state of the program after processing all internal and external transitions at the date t. Moreover, this property ensures that a process can not proceed its own generated event.

In the case in which we consider that global variables and channels are handled at the system level like in PROMELA and don't need synchronization, the semantics of DEv-PROMELA Program slightly changes. In this case, a DEv-PROMELA program P_r is a transition system

$$T = (S, \Lambda, \rightarrow)$$

where

- S is the cartesian product of the set of states of each process, the set of global variables and channels that compose the program;
- Λ is the set of all statements, including statements changing the value of global variables and channels.
- $\rightarrow$ the set of transitions;

These two definitions are strictely identical. Indeed, the second definition generates a transition system that simulates the first one.

Thanks to the underlying algebraic structure and the semantics, it is shown that, given a DEv-PROMELA model, it exists at least a DEVS model that strictly simulates it. The demonstration is not included again in this paper. However, it can be intuitively deduced, whereas a DEv-PROMELA model behaves like a Classic DEVS model.

3 SIMULATION FORMALISM HIERARCHY

This section makes brief recalls about the simulation formalisms hierarchy proposed by Giambiasi (2009).

3.1 Sequential Machine With Transitory States (SMTS)

SMTS allows representing reactive systems which will react to events only when they are in a *steady state*. For instance, an old analogic elevator which cannot be stopped when going up can be modelled as a SMTS system. Formally, a SMTS model M is defined as a 7-uplet:

$$M =< X, Y, S, \delta_i, \delta_e, \lambda_s, \lambda_t >$$

where

X is a finite set of input events;
Y is a finite set of output events;
$S = S_T \cup S_S$ where $S_T \cap S_S = \emptyset$, S_S is the set of steady states (i.e. the model remains in these states until an event occurs), and S_T is the set of transitory states;
$\delta_i : S_T \rightarrow S$ is the internal transition function which describes the autonomous behaviour of the model;
$\delta_e : S_S \times X \rightarrow S$ is the external transition function which describes the reactive behaviour of the model;
$\lambda_t : S_T \rightarrow Y$ which defines the output event generated by the model when it transits from a transitory state to the next state;
$\lambda_s : S_S \times X \rightarrow Y$ which defines the output event generated by the model when it transits from a steady state to the next state.

We must also denote that the notion of timed events is not defined in SMTS.

3.2 Temporal Moore Machine (TMM)

TMM can be viewed as an extension of SMTS, with the introduction of two concepts (timed event and state lifespan). TMM is very useful to describe systems which will perform tasks when they receive an event, and which will then immediately return to a passive state. Thus, formally a TMM model M is an 7-uplet:

$$M =< X, Y, S, \delta_i, \delta_e, ta, \lambda >$$

where

X is a finite set of input events;

Y is a finite set of output events;
$ta : S \to \mathbb{R}^+ \cup \infty$ defines the lifetime of each state;
$S = S_T \cup S_S$ where $S_T \cap S_S = \emptyset$, S_S is the set of steady states (i.e. $s \in S_S \Rightarrow ta(s) = +\infty$), and S_T is the set of transitory states (i.e. $s \in S_T \Rightarrow ta(s) \in [0; +inf[$);
$\delta_i : S_T \to S_S$ is the internal transition function;
$\delta_e : S_S \times X \to S_T$ is the external transition function;
$\lambda : S_T \to Y$ is the output function defined only for transitory states.

We can denote the restriction on δ_i and δ_e which enforces that a steady state is followed by a transitory state, and a transitory state is followed by a steady state. In fact, in semantics of TMM and SMTS, this is not a restriction. Because the notion of lifespan and timed event were introduced, a succession of transitory states in SMTS is compressed in one transitory state with a finite lifetime in TMM.

TMM also supports coupling which allows modular building of models. This notion is important because it introduces the capability to model complex systems by components which could interact with each other.

3.3 Temporal Sequential Machine (TSM)

TSM is a distension of TMM by allowing events to occur in transitory states. TSM allows the modelling of a new elevator which can stop at each level even if a call button is pressed during an elevation phase. A TSM model M is an 7-uplet:

$$M =< X, Y, S, \delta_i, \delta_e, ta, \lambda >$$

where

X is a finite set of input events;
Y is a finite set of output events;
$ta : S \to \mathbb{R}^+ \cup \infty$ defines the lifetime of each state;
$S = S_T \cup S_S$ where $S_T \cap S_S = \emptyset$, S_S is the set of steady states (i.e. $s \in S_S \Rightarrow ta(s) = +\infty$), and S_T is the set of transitory states (i.e. $s \in S_T \Rightarrow ta(s) \in [0; +inf[$);
$\delta_i : S_T \to S$ is the internal transition function;
$\delta_e : S \times X \to S$ is the external transition function;
$\lambda : S_T \to Y$ is the output function defined only for transitory states.

We can note that restriction of the succession of input/ouput events was removed, allowing TSM without autonomous cycle.

Using such a hierarchy helps designers to focus on interesting models and involves implicit requirements. At least, the structure imposed by each formalism must be respected by the model (otherwise, there is not a model !). This notion is important for model validity and ensures that the model is a good representation of the system. Then, the next section shows how this hierarchy can be used with DEv-PROMELA in order to focus on interesting aspects of a system under study.

4 RESTRICTING DEV-PROMELA

Using such a hierarchy like the one introduced in the previous section is useful if modellers want to restrict the behaviour of the model. While DEv-PROMELA embbeds the semantics of Classic DEVS, and allows formal verification by using model-checking, restricting the DEv-PROMELA language will allows using model-checking on models expressed in any formalisms of the previous hierarchy. This is possible only

because the formalisms described in the previous section are DEVS subclasses. All the restrictions described in this section are purely syntactic.

4.1 Sequential Machine with Transitory States (SMTS)

SMTS can be modelled using the SMTS-PROMELA formalism, which is a DEv-PROMELA model with these syntax restrictions:

1. A SMTS-PROMELA model have only one process, no channel and no global variable.
2. An event descriptor can emit an output.
3. Only one type of descriptor (autonomous or event) is permitted by statement.
4. Lifetime of transitory states is 0.

The first constraint ensures the model is atomic (a SMTS is not modular). The second constraint allows modelling of the ouput function. The third constraint allows differentiation between steady and transitory states.

Therefore, a SMTS-PROMELA model P with a set of statements L is an automaton $T = (S_\tau, E, \delta_i, \delta_e, s_0, F)$ where

- $S_\tau = \{s_i = (t_s, i, l_1, ..., l_m, \in \mathbb{N} \times \prod_{i=1}^{m} L_i)\}$ is the set of states. i is the identifier of the state related to the statement l which defines it; the sets L_i are the sets of values of each local variable l_i;
- S_T, the set of transitory states (those denotes with a transitory statement in the specifications);
- S_S, the set of steady states (those denotes with an event statement in the specifications);
- E is the set of events; E contains at least the silent event denoted ε;
- $\delta_i : S_T \to S_\tau \times E$ is the internal transition function;
- $\delta_e : S_S \times E \to S_\tau \times E$ is the external transition function;
- s_0 is the initial state;
- F is the set of final states.

Interpretation of a SMTS-PROMELA At a some time, the model is in a steady state, in which it stays until an event occurs. The new state is then the result of applying the δ_e function to the current state. Otherwise, the model is on a transitory state and instantaneously transits to the next state following the δ_i function. In both cases, an event is generated.

Theorem 1. *A SMTS-PROMELA model is a SMTS model.*

Given a SMTS-PROMELA model $T = (S_\tau, E, \delta_i, \delta_e, s_0, F)$, we can build a SMTS model

$$M =< X, Y, S, \delta_i, \delta_e, \lambda_T, \lambda_S >$$

where

$X \subset E$ is a finite set of input events;
$Y \subset E$ is a finite set of output events;
$S = S_\tau$ and $S_T \cap S_S = \emptyset$ by definition (a statement is decorated either by a transitory descriptor or an event descriptor);

$\delta_i : S_T \rightarrow S$ is the internal transition function which describes the autonomous behaviour of the model;
$\delta_e : S_S \times X \rightarrow S$ is the external transition function which describes the reactive behaviour of the model;
$\lambda_T : S_T \rightarrow Y$ which defines the output event generated by the model when it transits from a transitory state to the next state;
$\lambda_S : S_S \times X \rightarrow Y$ which defines the output event generated by the model when it transits from a steady state to the next state.

While a SMTS-PROMELA behaves like a SMTS (a more rigourous proof using the bisimulation relationship is possible), and it is possible to build a SMTS model from the SMTS-PROMELA model, then a SMTS-PROMELA model is a SMTS model. Moreover, a SMTS-PROMELA model is a DEv-PROMELA model (intuitively, because a SMTS model is a DEVS model).

4.2 Temporal Moore Machine (TMM)

TMM-PROMELA enforces these restrictions:

1. A TMM program can not involve channels or global variables;
2. Only one type of descriptor (autonomous or event) is permitted by statement.
3. An event descriptor can be followed only by an autonomous descriptor, and vice-verse.

The first constraint ensures that the synchronozing actions for channels and global variables do not break the TMM semantics. The second ensures that a steady state is followed by a transitory state and vice-verse.

Therefore, a TMM-PROMELA model is an automaton $T = (S_\tau, E, \delta_i, \delta_e, s_0, F)$ where

- $S_\tau = \{s_i = (t_s, i, l_1, ..., l_m, \in \mathbb{N} \times \prod_{i=1}^{m} L_i)\}$ is the set of states. i is the identifier of the state related to the statement l which defines it; the sets L_i are the sets of values of each local variable l_i;
- $S_T \subset S_\tau$, the set of transitory states (those denotes with a transitory statement in the specifications);
- $S_S \subset S_\tau$, the set of steady states (those denotes with an event statement in the specifications);
- E is the set of events; E contains at least the silent event denoted ε;
- $\delta_i : S_T \rightarrow S_S \times E$ is the internal transition function;
- $\delta_e : S_S \times E \rightarrow S_T$ is the external transition function;
- s_0 is the initial state;
- F is the set of final states.

Theorem 2. *A TMM-PROMELA model is a TMM model.*

Given a TMM-PROMELA model $T = (S_\tau, E, \delta_i, \delta_e, s_0, F)$, we can build a TMM model

$$M =< X, Y, S, \delta_i, \delta_e, ta, \lambda >$$

where

$X \subset E$ is a finite set of input events;
$Y \subset E$ is a finite set of output events;
$ta : S \rightarrow \mathbb{R}^+ \cup \infty$ defines the lifetime of each state (from the event descriptor);

$S = S_\tau$ where $S_T \cap S_S = \emptyset$ by construction (constraint 2), S_S is the set of steady states (i.e. $s \in S_S \Rightarrow ta(s) = +\infty$), and S_T is the set of transitory states (i.e. $s \in S_T \Rightarrow ta(s) \in [0; +inf[$);
$\delta_i : S_T \rightarrow S_S$ is the internal transition function;
$\delta_e : S_S \times X \rightarrow S_T$ is the external transition function;
$\lambda : S_T \rightarrow Y$ is the output function defined only for transitory states.

δ_i and δ_e are building using the transition function of the TMM-PROMELA model. Therefore, we can build a TMM that simulates the TMM-PROMELA model. Then, a TMM-PROMELA model is a TMM model.

Because a TMM-PROMELA is a TMM, it is possible to compose a TMM program by modelling each process with a TMM atomic model. The closure under coupling described in (Giambiasi 2009) ensures the TMM-PROMELA program is a TMM.

4.3 Temporal Sequential Machine (TSM)

TSM-PROMELA specifications are DEv-PROMELA specifications with :

1. A TSM program can not involve channels or global variables;
2. A pure event descriptor can be followed by an autonomous descriptor;
3. Clock is reset.

Therefore, a TSM-PROMELA model is an automaton $T = (S_\tau, E, \delta_i, \delta_e, s_0, F)$ where

- $S_\tau = \{s_i = (t_s, i, l_1, ..., l_m, \in \mathbb{N} \times \prod_{i=1}^{m} L_i)\}$ is the set of states. i is the identifier of the state related to the statement l which defines it; the sets L_i are the sets of values of each local variable l_i;
- $S_T \subset S_\tau$, the set of transitory states (those denotes with a transitory statement in the specifications);
- $S_S \subset S_\tau$, the set of steady states (those denotes with an event statement in the specifications);
- E is the set of events; E contains at least the silent event denoted ε;
- $\delta_i : S_T \rightarrow S_\tau \times E$ is the internal transition function;
- $\delta_e : S_\tau \times E \rightarrow S_\tau$ is the external transition function;
- s_0 is the initial state;
- F is the set of final states.

Theorem 3. *A TSM-PROMELA model is a TSM model.*

The demonstration is similar to the previous proofs. Given a TSM-PROMELA model $T = (S_\tau, E, \delta_i, \delta_e, s_0, F)$, we can build a TSM model

$$M =< X, Y, S, \delta_i, \delta_e, ta, \lambda >$$

where

$X \subset E$ is a finite set of input events;
$Y \subset E$ is a finite set of output events;
$ta : S \rightarrow \mathbb{R}^+ \cup \infty$ defines the lifetime of each state;
$S = S_T \cup S_S = S_\tau$;
$\delta_i : S_T \rightarrow S$ is the internal transition function;
$\delta_e : S \times X \rightarrow S$ is the external transition function;

$\lambda : S_T \rightarrow Y$ is the output function defined only for transitory states.

Therefore, it is possible to build a TSM structure that behaves like a TSM-PROMELA model. Moreover, the constraints 2 and 3 ensure the TSM-PROMELA model behaves like a TSM model. Then, a TSM-PROMELA model is a TSM model. The closure under coupling property ensures that a TSM-PROMELA program is a TSM model.

4.4 Classic DEVS

Finally, the DEv-PROMELA model is a Classic DEVS model without restriction. Using the hierarchy allows modeller to be sure to respect some structural properties. By translating the syntax of DEv-PROMELA into formal specifications, these structure properties can be verified using a theorem proving on the language (syntax proof) for example.

Note that for each sub-formalism, it is easier to show that a model is structurally equivalent to a PROMELA model. Indeed, removing the event descriptors generates a PROMELA model which is structurally equivalent to the DEv-PROMELA base model. However, all the previous formalisms are subclasses of DEv-PROMELA. This means that, in each subformalism, the respective underlying structure is embbeded in the ones of the DEv-PROMELA model. Therefore, we can easily deduce that it is possible to find a PROMELA model which have the same structure as the former model. Then, model-checking can be applied on each submodel in order to prove time-independant properties on the model.

5 CONCLUSION

In this paper, we have shown that formalism hierarchies could be used for restricting or extending the DEv-PROMELA formalism. The restrictions are done only by applying syntactic constraints on the language. These changes influence the resulting structure and the semantics. Therefore, by reducing or increasing the expressiveness capabilities, modellers can focus only on interesting aspects at each phase of modelling. The hierarchical relation between formalisms ensures the preservation of properties. Then, a DEv-PROMELA model can be built from a PROMELA model, while the DEv-PROMELA model is the final step before the implementation.

REFERENCES

Abdulhameed, A., A. Hammad, H. Mountassir, and B. Tatibouët. 2014. “An Approach Combining Simulation and Verification for SysML using SystemC and Uppaal”. In *CAL 2014, 8ème conférence francophone sur les architectures logicielles*.

Aliyu, H. O., O. Maïga, and M. K. Traoré. 2016. “The high level language for system specification: A model-driven approach to systems engineering”. *International Journal of Modeling, Simulation, and Scientific Computing* vol. 07 (01).

Chow, A. C. H., and B. P. Zeigler. 1994. “Parallel DEVS: A Parallel, Hierarchical, Modular, Modeling Formalism”. In *Proceedings of the 26th Conference on Winter Simulation*, WSC ’94, pp. 716–722. San Diego, CA, USA, Society for Computer Simulation International.

Giambiasi, N. 2009. “From Sequential Machines to DEVS Formalism”. In *Proceedings of the 2009 Summer Computer Simulation Conference*, SCSC ’09, pp. 216–222. Vista, CA, Society for Modeling; Simulation International.

Giambiasi, N., and J.-C. Carmona. 2006. "Generalized discrete event abstraction of continuous systems: {GDEVS} formalism". *Simulation Modelling Practice and Theory* vol. 14 (1), pp. 47 – 70.

Giambiasi, N., J. L. Paillet, and F. Chane. 2003, Dec. "From timed automata to DEVS models". In *Proceedings of the 2003 Winter Simulation Conference, 2003.*, Volume 1, pp. 923–931 Vol.1.

Holzmann, G. 2003. *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional.

Holzmann, G. J. 1991. *Design and Validation of Computer Protocols*. Prentice-Hall, Inc.

Hwang, M. H. 2011. "Taxonomy of DEVS Subclasses for Standardization". In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pp. 152–159, Society for Computer Simulation International.

Hwang, M. H. 2014. "Taxonomy of DEVS Variants". In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, DEVS '14, pp. 22:1–22:6, Society for Computer Simulation International.

Maiga, O., U. Bright Ighoroje, and M. Kaba Traoré. 2012, June. "Integration des Methodes Formelles dans la Specification, la Verification et la Validation des Modeles de Simulation a Evenements discrets". In *9th International Conference on Modeling, Optimization & SIMulation*. Bordeaux, France.

Saadawi, H., and G. Wainer. 2010. "Rational Time-advance DEVS (RTA-DEVS)". In *Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim '10, pp. 143:1–143:8, Society for Computer Simulation International.

Wainer, G. A. 2009. *Discrete Event Simulation and Modeling: Theory and Applications - Model-Based Design*. 1st ed. Boca Raton, FL, USA, CRC Press, Inc.

Yacoub, A., M. Hamri, C. Frydman, C. Seo, and B. Zeigler. 2015. "Towards an extension of Promela for the modeling, simulation and verification of discrete-event systems". *27th European Modeling and Simulation Symposium, EMSS 2015*, pp. 340–348.

Yacoub, A., M. e.-a. Hamri, and C. Frydman. 2016. "Using DEv-PROMELA for Modelling and Verification of Software". In *Proceedings of the 2016 Annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, SIGSIM-PADS '16, pp. 245–253, ACM.

Zeigler, B. P. 1976. *Theory of Modeling and Simulation*. John Wiley.

Zeigler, B. P., T. G. Kim, and H. Praehofer. 2000. *Theory of Modeling and Simulation*. 2nd ed. Academic Press, Inc.

AUTHOR BIOGRAPHIES

AZNAM YACOUB received his PhD in Computer Science at the Aix-Marseille University. His main research interests include formal verification and discrete-event simulation for the verification and validation of software. His email address is aznam.yacoub@lsis.org.

MAAMAR EL AMINE HAMRI is PhD in Computer Science from the Aix-Marseille University. He mainly works on DEVS and its extensions G-DEVS and Min-Max DEVS. His email address is amine.hamri@lsis.org.

CLAUDIA FRYDMAN is Professor of Computer Science at the Aix-Marseille University. Her main research interests include discrete-event simulation. Her email address is claudia.frydman@lsis.org.