Formal Framework for the DEVS-Driven Modeling Language

Conference Paper · September 2011

CITATIONS
5

READS
26

3 authors:

Mamadou Kaba Traoré
University of Bordeaux
120 PUBLICATIONS **502** CITATIONS
SEE PROFILE

Ufuoma Bright Ighoroje
Universität des Saarlandes
8 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Oumar Maïga
University of Sciences, Techniques and Technology of Bamako
17 PUBLICATIONS **44** CITATIONS
SEE PROFILE

FORMAL FRAMEWORK FOR THE DEVS-DRIVEN MODELING LANGUAGE

Ufuoma Bright Ighoroje [(a)], Oumar Maïga [(b)], Mamadou Kaba Traoré [(c)]

[(a)]African University of Science and Technology, Abuja, Nigeria
[(b)]University of Bamako, Mali
[(c)]Blaise Pascal University, Clermont-Ferrand 2, France

[(a)]uighoroje@aust.edu.ng, [(b)]maigabababa78@yahoo.fr, [(c)]traore@isima.fr

ABSTRACT

The DEVS-Driven Modeling Language (DDML) is a graphical modeling language that is based on Discrete Event System Specification (DEVS). Models built with DDML are highly expressive and communicable and validation of model properties can be done by simulating these models following the DEVS simulator protocol. We can take advantage of the usefulness of formal methods and apply symbolic manipulation and reasoning to deduce properties of models that cannot be derived from simulation. Since DDML focuses on three levels of abstraction in the hierarchy of system specification, we propose to do formal reasoning at each level of abstraction by applying a semantic mapping function to formal methods that can capture the properties of the model at each level. We do this because we can gain more insight about a model by observing different perspectives. This formal framework for DDML is the focus of this paper.

Keywords: DEVS, Formal Methods, DDML

1. INTRODUCTION

Modeling is a way of thinking about systems by developing abstract models, usually at multiple levels of abstractions with each level revealing a perspective of the system that cannot be observed at other levels. Simulation and testing have been employed to verify and validate the properties of abstract models against the system under study by exploring some of the possible behaviors and scenarios. On the other hand, formal analysis provides a very attractive and increasingly appealing compliment to simulation by conducting exhaustive exploration of all possible behaviors through symbolic reasoning exercises. The advantage here is that we can combine formal analysis and simulation to derive properties of models and compare them with system properties. We can also take benefit of the advances in formal analysis and the existing tools that are available to ensure that the models built accurately represent the desired properties of the system. Since DEVS (Zeigler, Praehofer, and Kim 2000) is a "universal" and powerful simulation modeling formalism, integrating methods for formal analysis with DEVS would bring the desired results.

We say DEVS is universal because other formalisms have been proven to have an equivalent (or approximate) DEVS representation. DEVS supports full range of dynamic system representation. A Differential Equation System Specification (DESS) can have an approximate Discrete Time System Specification (DTSS) by discretization (selection of a sufficiently small constant time interval). A DTSS model, in turn has an equivalent DEVS representation. Quantization of events in a DESS system can result in an approximate DEVS model. As such DEVS approach can be used to model discrete systems and provide approximate representations of continuous and hybrid systems.

DEVS also promotes separation of concerns by separating the model, simulator, and experimental frame. Although DEVS is powerful, it is a semi-formal specification. It is up to the modeler to express the system structure, behavior, and traces in ways that are most appealing. This "freedom" has led to the development of several DEVS based modeling formalisms. To fill these voids, we propose the DDML (DEVS-Driven Modeling Language). DDML combines DEVS, visual modeling, and formal analysis. In addition, DDML provides an approach to unify the two variants of DEVS (Classic DEVS and Parallel DEVS).

This paper is structured as follows. In section 2, we review some related works. In section 3, we present the concrete syntax of DDML. In section 4, we present the formal framework for DDML and show the different levels of abstractions in DDML specification and the properties that can be derived.

2. RELATED WORKS

Related works have addressed formal analysis of DEVS models. These proposals range from formal model-checking of sub-classes of DEVS, or transformation of DEVS into formal methods for verification purposes, generation of traces from DEVS models for testing

Saadawi and Wainer (2010) proposed a subclass of classic DEVS by mapping the time advance to a rational number which they call Rational Time-Advance (RTA) DEVS thus imposing restrictions on the elapsed time to transform RTA-DEVS to Timed Automata (TA) to enable reachability algorithms to be implemented in UPPAL. Earlier, (Saadawi and Wainer 2009) had proposed a technique for verification of DEVS models based on Model-checking. The technique is to specify graphically DEVS models using E-CD + + and transforming these models into TA in UPPAAL.

Hong, Song, Kim, and Park (1997) proposed the Real-Time DEVS formalism (RT-DEVS) which introduces a time advance function that maps each state to a range with maximum and minimum time values. Further work on verifying RT-DEVS has been done by

using timed automata and UPPAAL with transformation from RT-DEVS to UPPAAL. (Hong and Kim 2005) proposed a method of verification of DEVS models in the environment DEVSim++. The approach is to specify the model in DEVS (operational formalism) and use the temporal logic (TL) formalism assertions to specify the properties and time constraints of the system. They use a projection technique (external TLA) to reduce the state space

Ernesto (2008) worked on the development of an alternative theoretical foundation for DEVS by defining DEVS models as labeled transition systems (LTS). With this, we can use existing tools for LTS to reason with DEVS and compare DEVS with other formalisms defined as LTS.

Hwang and Zeigler (2009) defined a class of DEVS, called finite and deterministic DEVS (FD-DEVS) by introducing assumptions that enable us to define a reachability graph.

Weisel, Petty and Mielke (2005) discussed a formal theory for semantic composability that examines simulation composability using formal definitions and reasoning.

Cristia (2007) proposed a transformation method of DEVS models in TLA +. The main conclusion here is that DEVS models describing discrete event systems can be easily translated into TLA+ specifications. This would be beneficial for DEVS since it lays the basis for a formal semantics of this powerful modeling language.

Hernandez and Giambiasi (2005) showed that verification of general DEVS models through reachability analysis is not decidable. They based their deduction on building a DEVS simulation Turing machine. They argue that reachability analysis maybe possible only for restricted classes of DEVS. This result however was based on introducing state variables into DEVS formalism with infinite number of values.

These works mentioned above have focused on only one aspect in the Zeigler's hierarchy of system specification by transforming DEVS into a formal method that can capture the properties of the model at that level. Then formal reasoning can be done with the formal structure. We argue that more insights about a model can be derived by studying other levels. We realize that one formal method might not be suitable to fully capture all aspects of a system. Hence, we propose a framework that would provide logical semantics for reasoning at different levels of abstraction. Furthermore, we make use of different formal methods at different levels, depending on the expressive power of the method chosen.

This framework that we propose is based on a graphical modeling formalism – the DEVS-Driven Modeling Language (DDML). DDML is inspired by DEVS that combines graphical modeling and formal analysis. It adopts the DEVS simulation protocol in its operational semantics. It is also a unifying framework for the two variants of DEVS (CDEVS and PDEVS).

3. THE DEVS DRIVEN MODELING LANGUAGE (DDML)

Fig. 1 is the concrete syntax of DDML showing its elements and the relationships between elements.

A DDML model interacts with its environment via IOFrames (input and output events) which are realized with input and output ports (represented graphically as arrows). There are two types of models – atomic (describes a system that cannot be decomposed into sub-systems) and coupled (composed of sub-models with couplings between the models. Couplings are partitioned into External Input Coupling (EIC), Internal Coupling (IC), and External Output Coupling (EOC). The select flags within the coupled model are used to resolve synchronization issues between child-models. This corresponds to the select function in CDEVS. Some situations can lead to a voting/Condorcet's paradox. Several flags can be added to indicate paradoxes. Atomic model is drawn as a box with input and output arrows. A coupled model is drawn in a similar shape containing its child-models, their couplings (with lines as shown) and select flags.

State variables are used to partition states in an atomic model. A DDML state is defined by a configuration on a set of finite state variables. Each state has properties based on this configuration. The Initial state is used to define all the state variables and to define the subroutines that are used in other states. An atomic model usually starts from an initial state, and then by external (in response to an external input event), internal (automatically at end of a lifespan), confluent (when there's a conflict in transition), or conditional transitions. It changes its state to passive (lifespan is infinite time), finite (lifespan between 0 and + ∞) or transient (lifespan is 0) states. When in a state, the system might undergo some activities. States are shown in boxes with 4 compartments for name, properties, activities and time advance. The transitions are drawn with arrows with the exception of the conditional transition drawn in a diamond box. Note the labels on the transitions. For internal (lamda is an output function and assignments refer to the reconfiguration of state variables before entering the next state), external (input is the trigger input event that causes the transition). Condition shows the criteria for particular transitions. The sub-diagram "illustrating transitions" shows the graphical origin of the transitions in DDML notation.

Note the following constraints not shown in the diagram. For EIC, source = self, target $\neq$ self; IC: source $\neq$ self, target $\neq$ self; EIC: source $\neq$ self, target = self. For Transient state: ta = 0; Passive state: ta = $+\infty$; Finite state: $0 < ta < + \infty$. "Illustrating Transitions" shows external transition must appear before the edge of the box (top or bottom) and directed towards the lateral sides. Internal transition must originate at the right edge of the box and confluent transition originates from the top right corner and is directed towards the back.

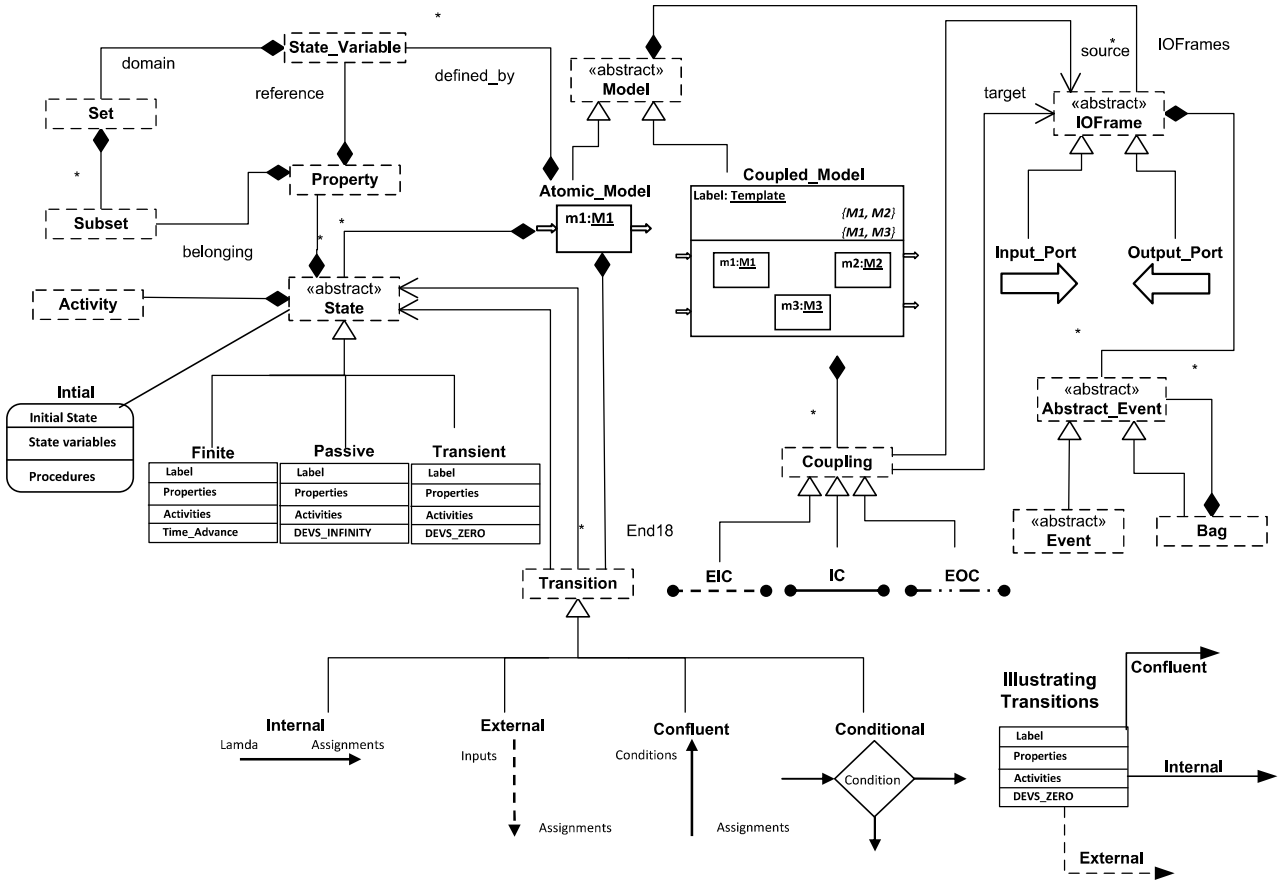

Figure 1: DDML Concrete Syntax showing notations for Coupled Model, Atomic Model, Input and Output Ports, Couplings (EIC, IC, and EOC), States (Initial, Finite, Passive, and Transient), and Transitions (External, Internal, Confluent, and Conditional). Others that do not have graphical notations are components of other graphical elements.

4. FORMAL FRAMEWORK FOR DDML

The DDML paradigm focuses on three levels of abstraction of the hierarchy of system specification (Zeigler, Praehofer, and Kim 2000):

- *Coupled Network (CN)* concerned with structural properties and functional couplings.
- *Input Output System (IOS)* concerned with system dynamics characterized by states and state transitions.
- *Input Output Relation Observation (IORO)* concerned with traces and trajectories of the system.

The formal semantics of DDML shall be expressed at these levels by using different formal methods to represent the corresponding properties that can be get. This is done so that we can derive different insights about the model. We can also take advantage of exisiting tools for formal an analysis to derive properties of the model. Fig. 2 summarizes the formal framework for DDML.

At the CN level, a semantic mapping function maps DDML processes onto concurrent processes defined in CSP (communicating sequential processes) (Hoare 1985). At the IOS level, a semantic function maps the state transition diagram onto an LTS (labeled transition system). At the IORO level, the system traces (which are expressed as footprints of the state transition system) are mapped onto CTL (computational tree logic). Due to space limitations, we shall show only the semantic mapping at the IOS level and give a taste of the properties that can be derived at other levels.

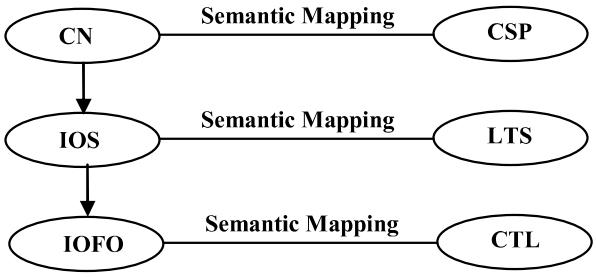

Figure 2: Formal framework for DDML

4.1. DDML at the IOS Level

The IOS level of DDML can be mapped to a labeled transition system (LTS). An LTS is a tuple $(S, \Lambda, \rightarrow)$ where S is a set (of states), Λ is a set (of labels) and $\rightarrow \subseteq S \times \Lambda \times S$ is a ternary relation (of labeled transitions). If $p, q \in S$ and $\alpha \in \Lambda$, then $(p, \alpha, q) \in \rightarrow$ is written as

$p \xrightarrow{\alpha} q$. This represents a transition from state p to state q with label α. Labels can represent different things (typically input expected or triggering actions).

An atomic DDML model is a tuple of the form:

$B = <X_B, Y_B, S_B, S_{0B}, \psi, C(V_B), T_{int}, T_{ext}, C(e), OP, \varphi, Act>$

Where

$X_B = \{(p, X_p) | p \in IPorts \text{ and } dom(p) = X_p\}$

Such that $\#X_B < +\infty$ is the set of input ports;

$Y_B = \{(q, Y_q) | q \in OPorts \text{ and } dom(q) = Y_q\}$

Such that $\#Y_B < +\infty$ is the set of output ports;

$V_B = \{(v, S_v) | v \in StateVar \text{ and } dom(v) = S_v\}$

$\#V_B < +\infty$ is the set of state variables;

S_B is a finite set of state clusters;

$S_{0B} \in S_B$ is the initial state of B ;

OP is the set of operations defined on the state variables;

Given that 'ops' is an operation and 'a' is a state, the notation a.ops indicates that 'a' is operated by 'ops' to change the state of 'a'.

$C(V_B)$ is a set of constraints on state variables. A constraint is of the form

$c ::= v \in D | v \in D \wedge v' \in D' | v \in D \vee v' \in D'$

Where $v = (v_1, \dots, v_n)$ and $D \subseteq dom(v_1) \times \dots \times dom(v_n)$,

$v' = (v'_1, \dots, v'_m)$ and $D' \subseteq dom(v'_1) \times \dots \times dom(v'_m)$.

$\psi: S_B \rightarrow \mathbb{P}C(V_B)$ is a mapping between each element of S_B and a finite set of conditions on the variables in V_B.

Given $s \in S_B$,

We denote $\bar{s} = \{s' \in \prod_{v \in StateVar} dom(S_v) \, | s' \vDash \psi(s)\}$

The function ψ is defined by the following:

$\psi(S_{0_B})$ is verified by the initial state of the system

$\bigcup_{s \in S_B} \bar{s} = \prod_{v \in StateVar} dom(S_v)$ and $\forall s, s' \in S_B, \bar{s} \cap \bar{s'} = \emptyset$ for $s \neq s'$

Act is the set of activities;

$\varphi: S_B \rightarrow \mathbb{P}\, Act$ is the mapping from states to the set of activities ;

$T_{int} \subseteq (S_B \times \mathbb{R}^+) \times C(V_B) \times Y \times \mathbb{P}\, OP \times (S_B \times \mathbb{R}^+)$ is the set of internal transitions satisfying:

The internal transition $((s, d), c, l, ops, (s', d')) \in T_{int}$ will be denoted by $(s, d) \xrightarrow{<c,l,ops>}_{iB} (s', d')$ or $(s, d) \xrightarrow{<c,l,ops>} (s', d')$ to avoid confusion.

$C(e)$ is the set of conditions α of the elapsed time e of the for

$\alpha ::= e \sim t | e - t \sim t' | e + t \sim t' | e * t \sim t' | \alpha_1 \wedge \alpha_2 | \alpha_1 \vee \alpha_2$

Where $\sim \in \{=, \leq, <, \geq, >\}$, e, t, t' are positive real numbers and α_1 et α_2 are conditions (denoted by $\vDash \alpha$ if e satisfies the condition α);

$T_{ext} \subseteq (S_B \times \mathbb{R}^+) \times C(V_B) \times X \times C(e) \times \mathbb{P}\, OP \times (S_B \times \mathbb{R}^+)$ is the set of external transition.

The internal transition $((s, d), c, x, c(e), ops, (s', d')) \in T_{ext}$ will be noted $(s, d) \xrightarrow{<x,c(e),ops>}_{eB} (s', d')$ or $(s, d) \xrightarrow{<x,c(e),ops>} (s', d')$. In particular, if there is no branching condition, the transition $((s, d), \emptyset, x, c(e), ops, (s', d')) \in T_{ext}$ will be denoted $(s, d) \xrightarrow{<x,c(e),ops>}_{eB} (s', d')$.

At a given instance of time, the system is in a cluster state $s \in S_B$ with a life span d that is to say the variables satisfy $\psi(s)$. If the lifespan of the current atomic state $s_1 \in \bar{s}$ elapses before an external event occurs then the model output l is sent just before transiting to another cluster state s' such that $((s, d), l, ops, (s', d')) \in T_{int}$ (internal transition, $v.ops \vDash \psi(s')$). When an external event x occurs before the end of the lifespan of the state, the model transits to the cluster state s' such that $((s, d), x, c(e), ops, (s', d'))$ (external transition, $v.ops \vDash \psi(s')$ with a life time of d'.

Given an atomic DDML model

$B = <X_B, Y_B, S_B, S_{0B}, \psi, C(V_B), T_{int}, T_{ext}, C(e), OP, \varphi, Act>$

It can be shown that B is equivalent to LTS

$L(A) = < S_L, init, \Sigma, D, T >$

Where,

$S_L = Q = \{((s, e), d) | s \in S_B \; and \; 0 \leq e \leq d\}$;

$int = ((S_{0B}, 0), d)$ is the initial state ;

$\Sigma = (\bigcup_{p \in IPorts} X_p) \cup (\bigcup_{q \in Oports} Y_q) \cup (\mathbb{R}^+ \cup \{0, +\infty\})$ is the set of events (alphabet) ;

$D = \{((s, e), d) \xrightarrow{y} ((s', 0), d') | \exists ops \in \mathbb{P}OP, e = d \; and \; ((s, d), y, ops, (s', d')) \in T_{int}\} \cup \{((s, e), d) \xrightarrow{x} ((s', 0), d') | \exists ops \in \mathbb{P}OP, 0 \leq e < d \; and \; ((s, d), x, c(e), ops, (s', d')) \in T_{ext}\}$

$T = \{((s, e), d) \xrightarrow{t} ((s, e'), d) | e' = e + t < d\}$

To illustrate the how DDML can express properties at this level graphically, we shall consider a DDML IOS model of a traffic light. The functional diagram of the TrafficLight is shown in Fig. 3.

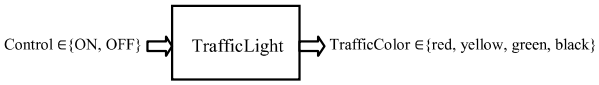

Figure 3: Simplified Traffic Light Atomic Model

The output evens in the TrafficLight system are the displays of colors (red, yellow, green, or black). This controls the flow of cars in a road network. Hence, we have an output port (trafficColors). The TrafficLight can be controlled by a Control (with ON and OFF as possible inputs).

The DDML IOS model of the TrafficLight is shown in Fig. 4. The state of the system is determined by the value of the color attribute. Hence we have the states: STOP, READY_TO_GO, READY_TO_STOP, GO, and OFF. The value of the color attribute is the indicated in the state diagrams. OFF is a passive state (time advance is INFINITY, and it does not undergo any internal transition).

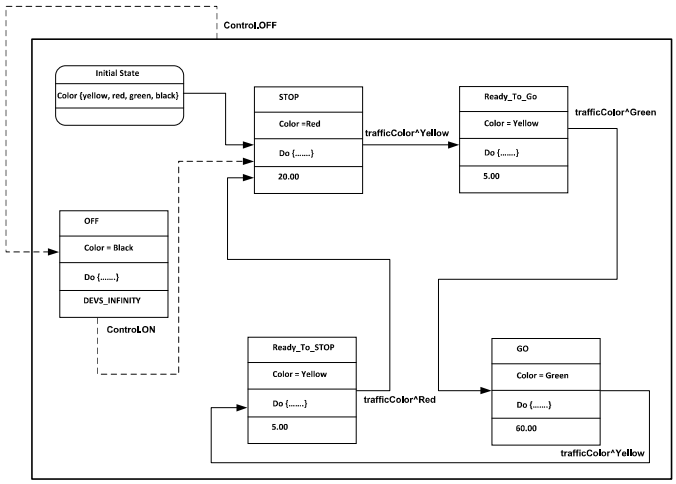

Figure 4: DDML IOS model of TrafficLight

From OFF, when the system receives an ON signal through its Control port (***Control.ON***), it undergoes an external transition to STOP state. The system remains in STOP for 20 seconds before an internal transition to READY_TO_GO. It outputs Yellow through the trafficColor port (***trafficColor^Yellow***). From any state, if Control.OFF occurs, the system transitions to OFF state (external transition).

We can do formal analysis on this system by transforming it into a labeled transition system and leveraging formal analysis tools like LTSA (Labeled Transition System Analyzer) (Magee and Kramer 2006). LTSA is a model checking tool that uses algorithms to check for desirable and undesirable properties for all possible sequences of events and actions and to see that it conforms to specification. LTSA uses FSP (Finite State Processes) as a concise way to represent an LTS and the properties of the system can be animated and visualized. Fig. 5 is a snapshot of LTSA with our TrafficLight model. Labels are triggers for transition. We can trace all the possible sequences of events and perform checks for safety, deadlocks, and completeness.

For example, we can ask questions like "*how would the system react when it is in READY_TO_GO state (State 2) and it receives Control.ON?*" Analysis shows that the model does not take care of this possibility. This would make us to refine the model to ensure that it is complete.

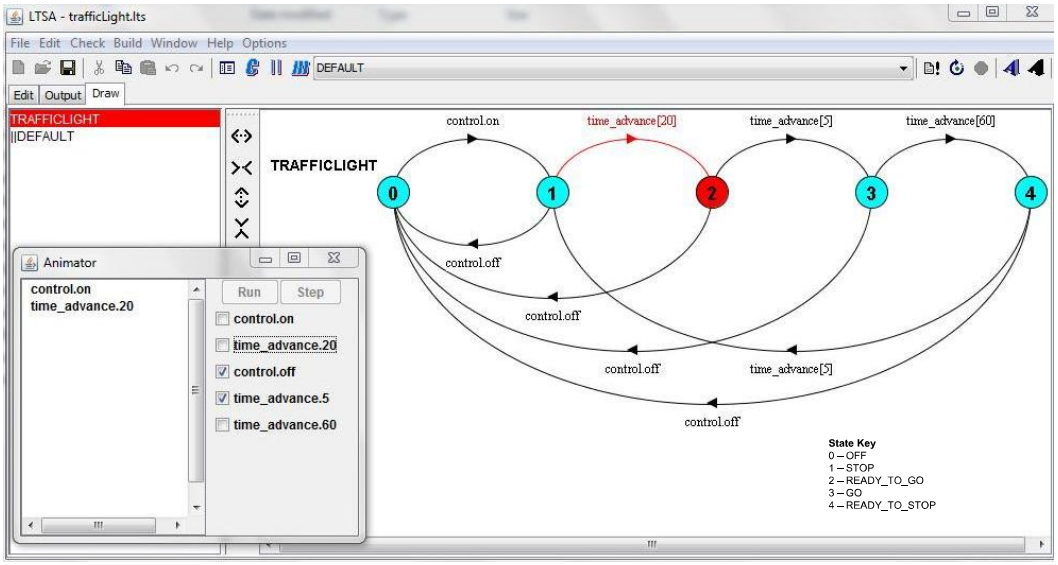

Figure 5: Analysis of TrafficLight in LTSA

4.2. DDML at the CN Level

The focus at the CN level is the structure and functional couplings of the system. To illustrate, we shall consider a model of a road-network (RN). Fig. 6 shows a schematic road network. A more advanced system would contain the road network, traffic lights, and authorization and synchronization channels. The CN model is shown in fig. 7.

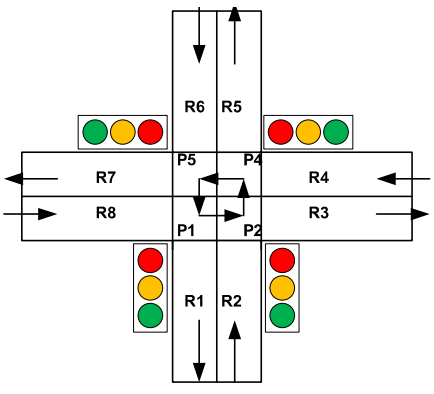

Figure 6: Road Network (RN)

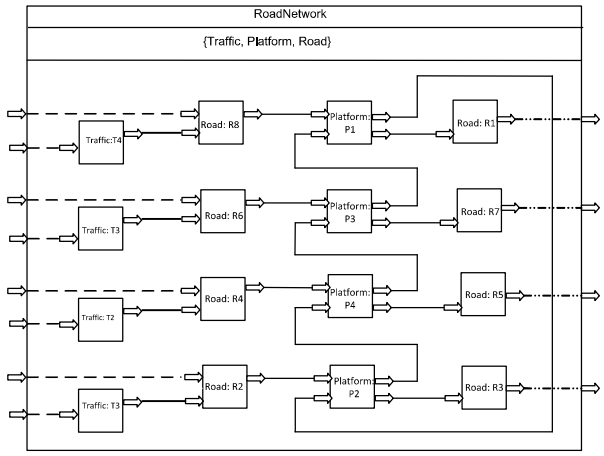

Figure 7: DDML CN model of the RoadNetwork

The coupled model (RN) consists of atomic models: TrafficLights (T1-4), Roads (R1-8) and Platforms (P1-4). The TrafficLights are for flow control of cars. A platform is used for interchange of cars within the road network. There are 4 EIC couplings between RN and R8, R6, R4, and R2. These correspond to the events whereby cars enter the road network. These cars are interchanged between roads and platforms (as shown by the ICs). EOCs (between RN and R1, R7, R5, and R3) include events whereby cars leave the road network. The select flag indicates the priorities when components are imminent.

The formal semantics of DDML CN models can be given in terms of CSP or other process algebras. We can use the FDR (Failures-Divergence Refinement) model checker to check the models for desirable and undesirable properties.

4.3. DDML at the IORO Level

At the IORO level, we can derive properties about the trajectories of the system. These trajectories are footprints of the DDML IOS and they can be mapped to CTL. We can get several footprints depending on the starting state and the sequence of activities that occur. Fig. 8 shows the footprint of the traffic from when it is in an OFF state. It receives an ON signal from its Control port and transitions to the Stop state where it remains for 20 seconds and displays Yellow signal before moving to the READY_TO_GO state where it remains for 5 seconds, and so on.

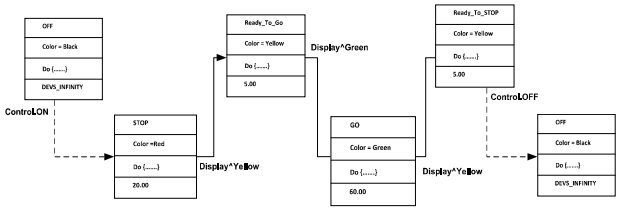

Figure 8: DDML IORO Model of Traffic Light

As explained earlier, this DDML IORO model can be easily mapped to CTL for formal analysis.

5. CONCLUSION

We presented a framework for formal reasoning of DDML models at three levels of abstraction using different formal methods that allow us to draw different insights about the model. We presented the formal semantics of DDML IOS by semantic mapping to LTS. By leveraging model-checking tools like LTSA we could do some analysis to check for desirable and undesirable properties, safety, progress, and safety properties. Future work would investigate how this can be done at the CN and IORO levels.

ACKNOWLEDGMENTS

The work in this paper is partly funded by grants from RAMSES (*Réseau Africain pour la Mutualisation et le Soutien des pôles d'Excellence Scientifique*).

REFERENCES

Cristia, M. 2007. A TLA+ encoding of DEVS models. *International Modeling and Simulation Multiconference*, Buenos Aires (Argentina), pp. 17–22.

Ernesto, P. 2008. *Modeling and simulation of dynamic structure discrete-event systems*. Thesis (Ph.D). MCGill University.

Hernandez, A., and Giambiasi, N. 2005. State Reachability for DEVS Models. *Proceedings of Argentine Symposium on Software Engineering*.

Hoare, C.A.R. 1985. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall.

Hong, J., Song, H., Kim, T., and Park, K. 1997. A Real-time discrete-event system specification formalism for seamless real-time software development. *Discrete Event Systems: Theory and Applications*, vol. 7, pp. 355–375.

Hong, K.J., and Kim, T. G. 2005. Timed I/O Test Sequences for Discrete Event Model Verification. AIS 2004, LNAI 3397, pp. 275–284.

Hwang, M.H., and Zeigler, B. P. 2009. Reachability Graph of Finite and Deterministic DEVS Networks. *IEEE Transactions on Automation Science And Engineering*, 6 (3).

Magee J., Kramer J. 2006. "*Concurrency: State Models and Java Programs*". 2nd Edition.

Saadawi, H., Wainer, G. 2010. From DEVS to RTA-DEVS. *IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications*, 2010 pp.207-210.

Saadawi, H., Wainer, G. 2009. Verification of Real-Time DEVS Models, *Proceedings of SpringSim Multi Simulation Conference*, San Diego, CA March 2009.

Weisel, E.W., Petty, M.D., Mielke, R.R. 2005. A Comparison DEVS Semantic Composability Theory. *Proceedings of the Spring 2005 Simulation Interoperability Workshop*

Zeigler, B., Praehofer, H., Kim, T. 2000. *Theory of Modeling and Simulation*. 2nd Edition. Academic Press.