FORMAL COMPATIBILITY OF EXPERIMENTAL FRAME CONCEPT AND FD-DEVS MODEL

Damien Foures, Vincent Albert, Alexandre Nketsa

9^{th} *International Conference of Modeling, Optimization and Simulation - MOSIM'12*
June 6-8, 2012 - Bordeaux - France
"Performance, interoperability and safety for sustainable development"

FORMAL COMPATIBILITY OF EXPERIMENTAL FRAME CONCEPT AND FD-DEVS MODEL

D. FOURES, V. ALBERT, A. NKESTA

CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France
University of Toulouse ; UPS ; F-31077 Toulousee
dfoures@laas.fr, valbert@laas.fr, alex@laas.fr

ABSTRACT: *This paper shows how a Finite and Deterministic (FD-DEVS) model can be integrated within the concept of experimental frame. Experimental frame (EF) is used to define the environment that will be able to achieve desired simulation execution. These execution are extracted from the specifications of the studied system and included on specific part of the EF (the acceptor). This EF is defined with FD-DEVS like the model. FD-DEVS network permits to couple together the EF and the model. Applying an EF to a model implies in some case a non commandability of part of the model. This leads us to a second contribution where we formalise FD-DEVS model restriction. This restriction permits to apply less capable EF than the model itself. EF concept is illustrated with two examples, a running example of a toaster and a case study of an intelligent cruise controller implemented in UPPAAL tools.*

KEYWORDS: *Experimental Frame, Restriction, FD-DEVS, Compatibility*

1 INTRODUCTION

This paper describes a formal approach to verify that the system behaviour which is guaranteed by a model of that system combined with the hypothesis chosen by a simulation environment allows reaching a given simulation intended purpose. We focus in this paper in simulation used for the validation of functional system's requirements. We use the concept of experimental frame initially introduced by (Zeigler, Praehofer & Kim 2000) in its framework for Modelling and Simulation. In this framework the concept of experimental frame is added to the traditional system-model-simulator view in order to take into account the specification of the conditions in which a system is observed or experimented on. Then, an experimental frame can be seen as a system that interact with the system to obtain the data of interest in given conditions. Or it can be seen as a system which interact with the model, which is an abstract representation of the system, to answer a set of question about the system of interest.

An experimental frame has basically three components as illustrated in figure 1: a *generator* which generates a set of output segments ω onto the inputs of the system or the model; an *acceptor* which selects the data of interest of the system or the model while monitoring whether the desired experimental conditions are complied with and a *transducer* which observes and analyses the output segments ρ of the system or the model. SU is the set of summarization for all the IO (Input/Output) pairs observed within the frame (Traoré 2006). The transducer is used to realize the summarization.

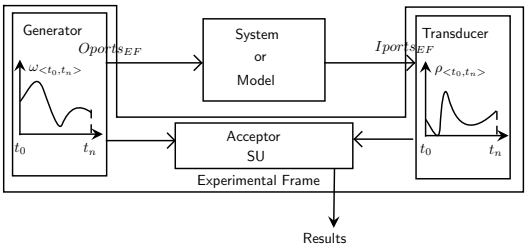

Figure 1: The experimental frame and its components

In such a framework, the *validity* of a model is defined as the degree to which a model faithfully represents a system within an experimental frame of interest. Indeed, we can consider a set of models of a same system hierarchized by a *morphism* relation. The concrete model is a model with more capabilities, meaning that it can be used for a greater number of experimental frames. However, for a given experimental frame, the abstract model can be as capable as the concrete model. Only few models can implement experimentation conditions required by an experimental frame to reach objectives and possibly supply valid simulation results.

The validity of a model is assessed within a given

experimental frame. Within this structure, we suggest a formal specification of the model to represent its domain of use, i.e. the set of acceptable input sequences and the set of provided output sequences. We suggest a formal specification of the experimental frame to represent the assumptions about the environment: the set of stimuli sequences injected into the model's inputs (generator), the set of observation sequences expected onto the model's outputs (transducer). The interaction of the model's guarantees and the assumptions about the environment must satisfy the conditions of acceptation of the simulation (acceptor). Our approach relies on the fact that the model, the generator and the transducer are formalized as FD-DEVS components (Hwang & Zeigler 2009). Finite and Deterministic DEVS (FD-DEVS) is a class of DEVS which resolves the problem of obtaining a finite reachability graph of DEVS network. It enables state-space exploration and decidability of qualitative analysis. Then, we can capture the timed I/O behavior of these components and define a set of conditions to detect incompatibilities between them. The concept of synchronous product in the automata theory instigates these conditions. This paper is an extension of a previous work (Albert, Nketsa & Seguin 2010) where time was not included in the study as a first-class variable. The acceptor is formalized with temporal logic. Temporal logic defines a set of acceptable behaviour on a given reachability graph. There are logics which refer to clock variables. If *altitude* ≤ 10 was true for at least ten units of time then *alarm* = *true*" is an example of a timed property.

Section 2 of this paper introduces FD-DEVS atomic and coupled components. It also gives fundamental definitions of behaviour or language of a FD-DEVS component. It introduces the two-slot toaster running example taken from (Hwang & Zeigler 2009). Section 3 gives a preview of our metric-based method which aims guiding experimental frame and/or model definition such that simulation usage can be improved by finding the right model and the right experimental frame for a given intended purpose. It also gives a state space metric based on trace inclusion between the model and the experimental frame which can be qualified from the conditions of compatibility which are introduced in section 4. We show in section 5 that compatibility verification can be solved using the tool Uppaal (Behrmann, David & Larsen 2004). We illustrate this with a simple example.

2 PRELIMINARIES

2.1 FD-DEVS

Here, we remind the FD-DEVS formalism as defined in (Hwang & Zeigler 2009)

2.1.1 Atomic FD-DEVS

An atomic FD-DEVS is a 7-tuple:

$A = \langle X, Y, S, s_0, \tau, \delta_x, \delta_y \rangle$ where:

- X is a finite set of input events. X can be split into two variables: $X = \{(p, v) |\ p \in Iports,\ v \in X_p\}$;
- Y is a finite set of output events, Y can be split into two variables: $Y = \{(p, v) |\ p \in Oports,\ v \in Y_p\}$;
- S is a finite set of states, $s_0 \in S$ is the initial state;
- $\tau : S \rightarrow \mathbb{Q}_{[0,\infty]}$ is the *time advance function*, where $\mathbb{Q}_{[0,\infty]}$ is the set of nonnegative rational numbers plus infinity. This function is used to determine the lifespan of a state;
- $\delta_x : S \times X \rightarrow S \times \{0, 1\}$ is the *external state transition function* that defines how an input event changes a state, and whether the internal schedule will be updated or not. The internal schedule of a state $s \in S$ is updated by $\tau(s')$ if $\delta_x(s) = (s', 1)$, otherwise (i.e., $\delta_x(s) = (s', 0)$), the schedule is preserved;
- $\delta_y : S \rightarrow Y^\phi \times S$ is the *output or internal state transition function*, where $Y^\phi = Y \cup \{\phi\}$ and $\phi \notin Y$ denotes the *silent event*;[1]

FD-DEVS model has an explicit time base, in contrast to DEVS. The *time base*, denoted by $\mathbb{T}$, is the set of nonnegative real numbers, i.e. $\mathbb{T} = [0, \infty)$. $t_e \in \mathbb{T}$ is the *elapsed time*. It is continuously increasing and its value denotes the time passage since $t_e = 0$. $t_s \in \mathbb{Q}_{[0,\infty]}$ is an other internal state variable, called *lifespan* or *schedule time span*. $t \in \mathbb{T} \cup \{\infty\}$ is considered as a upper limit of t_e. The existing range of t_e is defined by function $tr : \mathbb{T} \cup \{\infty\} \rightarrow 2^{\mathbb{T}}$ s.t. $tr(t) = [0, t]$ if $t < \infty; tr(t) = [0, \infty)$ if $t = \infty$. $Q_p = \{(s, t_s, t_e) |\ s \in S,\ t_s \in \mathbb{Q}_{[0,\infty]},\ t_e \in tr(t_s)\}$ is the set of legal states. $Q_{imp} = \{(imp, \infty, t_e) |\ imp \notin S,\ t_e \in \mathbb{T}\}$ is the set of illegal states s.t. $Q_p \cap Q_{imp} = \varnothing$. Then, total state set Q is defined as $Q = Q_p \cup Q_{imp}$. $Z = X \cup Y^\phi$ is the *total event set* of M.

2.2 FD-DEVS *Network*

A FD-DEVS *network* (also called coupled FD-DEVS model) is 6-tuple:

$C = \langle X, Y, D, \{M_1\}, C_x, C_y \rangle$ where:

[1] δ_y can be split into two functions: the output function $\lambda : S \rightarrow Y$ and the internal transition function $\delta_{int} : S \rightarrow S$

- X (res. Y) is finite set of input (res. output) events, such that $X \cap Y = \varnothing$.
- D is a finite set of names of subcomponents.
- $\{M_1\}$ is an index set of FD-DEVS models, where $i \in D$. M_i can be either an atomic or coupled FD-DEVS model.
- $C_x \subseteq X \times \bigcup_{i \in D} X_i$ is a set of input couplings where X_i is the set of inputs events of subcomponent $i \in D$.
- $C_y \subseteq X \times \bigcup_{i \in D} Y_i \times (\bigcup_{j \in D} X_j \cup Y)$, where $i \neq j$ is an set of output couplings, where Y_i is the set of output events of subcomponent $i \in D$.

2.3 Timed event and Behaviour

To define a sequence of state changes associated with events, we need to introduce a *timed event* and its sequence.
► A *timed event* is a pair of an event $z \in Z$ and its occurence time $t \in \mathbb{T}$ thus it is denoted as (z, t).
► *Concatenation* of two events (z_1, t_1) and (z_2, t_2) is denoted by $(z_1, t_1)(z_2, t_2)$, which can be defined if $t_1 \leq t_2$.
► The identity of concatenation operation is the null-event, denoted by ϵ. The null-event sequence over a time interval $[t_l, t_u] \subseteq \mathbb{T}$ is denoted by $\epsilon_{[t_l,t_u]}$.
► Given an even set Z and a time interval $[t_l, t_u] \subseteq \mathbb{T}$, the set of total event sequences is denoted by $\Omega_{[t_l,t_u]}$, and is defined by $\Omega_{[t_l,t_u]} = \{(z,t)^* | z \in z \cup \{\epsilon\}, t \in [t_l, t_u]\}$ which is the set of concatenations of finite or infinite timed events (plus $\epsilon_{[t_l,t_u]}$) over Z and $[t_l, t_u]$.
► Given $\Omega_{[t_l,t_u]}, \omega = (z_1, t_1)(z_2, t_2) \in \Omega_{[t_l,t_u]}$ where $t_l \leq t_1 \leq t_2 \leq t_u$ is equivalent to $\omega = \epsilon_{[t_l,t_1]}(z_1, t_1)\epsilon_{[t_1,t_2]}(z_2, t_2)\epsilon_{[t_2,t_u]}$.

The authors also define the state trajectory function of C by using a function $\Delta : Q \times \Omega_{[t_l,t_u]} \to Q$. Let $q = (..., (s_i, t_{si}, t_{ei}), ...) \in Q$ be a total state at time t_l. The state trajectory associated with a sequence of multiple events can be computed by applying a sequence of "null-or-one-event sequences", repeatedly. Based on this state trajectory function, the behavior of C is defined as the all possible event sequences with which the state of C does not enter to the illegal state imp. Formally, the behavior or language of C over a finite observation length $t \in \mathbb{T}$, denoted by $L(C, t)$, is

$$L(C, t) = \{\omega \in \Omega_{[0,t]} | \Delta(q_0, \omega) \in Q_p\}. \quad (1)$$

The infinite-observation length behavior or language of A, denoted by $L(C)$, is

$$L(C) = \{\omega \in \Omega_{[0,\infty)} | \Delta(q_0, \omega) \in Q_p\}. \quad (2)$$

2.4 Two-slot Toaster

A two-slot toaster coupled FD-DEVS model is given in (Hwang & Zeigler 2009). $C_{T12} = \langle X, Y, D, \{M_i\}, C_x, C_y \rangle$, where $X = \{?push1, ?push2\}$, $Y = \{!pop1, !pop2\}$, $D = T1, T2$, $C_x = \{(?push1, ?T1.push), (?push2, ?T2.push)\}$, $C_y = \{(T1.!pop, !pop1), (T2.!pop, !pop2)\}$. $T1$ and $T2$ are two atomic components such that $X = \{?push\}$, $Y = \{!pop\}$, $S = \{I, T\}$, where I and T stand for "idle" and "toast", respectively, $s_0 = I$, $\delta_x(I, ?push) = (T, 1)$, $\delta_x(T, ?push) = (T, 0)$, $\delta_y(T) = (!pop, I)$. $\tau(I) = \infty$ and $\tau(T) = 20$ for T1. $\tau(I) = \infty$ and $\tau(T) = 40$ for T2.

► Given a timed event sequence $\omega_{[0,70]} = (?push1, 5)(?push2, 20)(!pop1, 25)(!pop2, 60)$ $\omega_{[0,70]} \in L(T12, 70)$ because $\Delta((((I, \infty, 0), (I, \infty, 0), \infty, 0), \omega_{[0,70]}) = (((I, \infty, 45), (I, \infty, 10)), \infty, 10) \in Q_p$.

► Given a timed event sequence $\omega'_{[0,70]} = (?push1, 5)(?push2, 20)(!pop1, 30)(!pop2, 60)$ $\omega'_{[0,70]} \notin L(T12, 70)$ because $\Delta((((I, \infty, 0), (I, \infty, 0), \infty, 0), \omega'_{[0,70]}) = (((imp, \infty, 65), (imp, \infty, 50)), \infty, 40) \in Q_{imp}$.

We give in the figure 2 the reachability graph for the two-slot toaster. It exhibits all possible events that can occur at each state. The initial state is I1I2 for toaster 1 and toaster 2 in "idle" state until a $?push1$ or a $?push2$ event occurs.

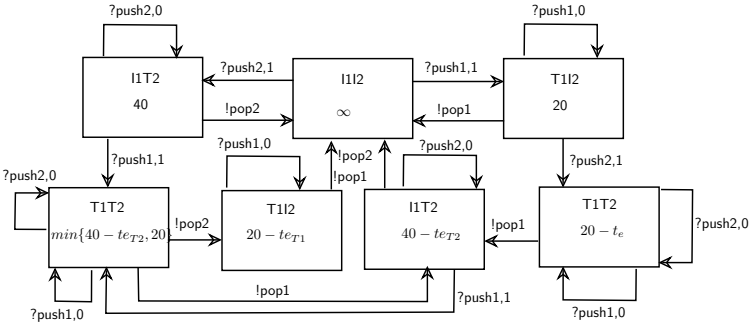

Figure 2: Reachability graph of two-slot toaster

3 A PREVIEW OF METRIC-BASED SIMULATION ASSESSMENT

3.1 Our method

Our approach is based on a formal description of the model capabilities, called the Simulation Domain of Use (SDU) and experimental frame assumptions called the Simulation Objectives of Use (SOU). We assume that the model designer and the simulation experimenter are different persons which is often the case in practice. The SDU describes the guarantees

provided by the model and the SOU describes the hypotheses about the simulation environment and the expected results to reach a given simulation intended purpose (i.e. a functional validation of a system of interest). The challenge is to define metrics to measure at which degree the interactions between the model and the experimental frame applied to this model satisfy the simulation intended purpose.

The method consists in step by step assessment of metrics where coverage analysis guides experimental frame and/or model definition. We consider different types of metrics: scope, precision and state space metrics. Scope metric identifies provided and required input/output ports of the model and the EF respectively and their range of values. Precision metric focused on the distance between two successive values of a datum. State space metric is made by trace inclusion to measure the extent of design verification provided by a set of test vectors.

Generally, the full compatibilty is not required, a model must be necessary to satisfy a given intended purpose but it can be more capable. Ideally it may be necessary and sufficient which means that it is simple enough for a given intended purpose.

3.2 State space metric

Consider a model given by $M = \langle X_M, Y_M, S_M, s_{0_M}, \tau_M, \delta_{x_M}, \delta_{y_M} \rangle$ and an experimental frame given by $EF = \langle X_{EF}, Y_{EF}, S_{EF}, s_{0_{EF}}, \tau_{EF}, \delta_{x_{EF}}, \delta_{y_{EF}} \rangle$.

Compatibility of input/output ports is a prerequisite for using simulation with respect to an experimental frame. M and EF can be connected if they have compatible input/output ports: (1) $Iports_{EF} \subseteq Oports_M$ and (2) $Oports_{EF} = Iports_M$.

The first condition ensures that all the events of interest required by the experimental frame are supplied by the model and the model may supply more events than necessary. The second condition ensures that all the events planned by the experimental frame can be performed and all the inputs necessary to perform the simulation are defined by the experimental frame. A particular attention must be paid to the case $Oports_{EF} \subset Iports_M$. We assume that EF and M can be connected in that case however, it is necessary to make sure that there is no dependency between a non-assigned input of the model and an observed output in which case the simulation results could be biased. Dependency between inputs/outputs of a component may be a specific condition which is beyond the scope of this paper. For reasons of simplicity we assume that the names used to designate EF and M input/output ports are identical.

If the event sets are compatible we can compare the set of executions of the model, which satisfy the properties given by the acceptor, with the set of execution expected by the generator and the transducer.

The behaviour of the model over a finite observation length $t \in \mathbb{T}$, is given by $L(M,t) = \{\omega \in \Omega^M_{[0,t]} \mid \Delta_M(q_0,\omega) \in Q_p\}$. The behaviour of the experimental frame over a finite observation length $t \in \mathbb{T}$, is given by $L(EF,t) = \{\omega \in \Omega^{EF}_{[0,t]} \mid \Delta_{EF}(q_0,\omega) \in Q_p\}$. Let $Z_{M/EF}$ be the total event set of M restricted to the total event set of EF. Let φ be a property expressed on $Z_{M/EF}$, we note $\|L(M,t)\|_{Z_{M/EF}}$ the set of executions of M which satisfies φ such that $\|L(M,t)\|_{Z_{M/EF}} = \{\omega \in \Omega^M_{[0,t]} \mid \Delta_M(q_0,\omega) \in Q_p, \Delta_M(q_0,\omega) \models \varphi\}$.

EF and M are fully compatible iff all executions of M satisfying φ restricted to the total event set of EF are executions of EF: $\|L(M,t)\|_{Z_{M/EF}} = L(EF,t)$ (case 1 in figure 3).

The exact matching being not required, intuitively, this means that:

- $\|L(M,t)\|_{Z_{M/EF}} \subseteq L(EF,t)$:the behaviour of the model is in the envelope of significant behaviour with respect to the EF (case 2 in figure 3).
- $L(EF,t) \subseteq \|L(M,t)\|_{Z_{M/EF}}$:all the experimentations planned by the EF can be performed on the model (case 3 in figure 3).

When $\|L(M,t)\|_{Z_{M/EF}} \subseteq L(EF,t)$ is false, there are model executions which are not EF executions. We identify two cases (figure 3):

► 2.1. There is a test coverage risk. This may be the case if the model considers different input event sequences than those planned by the EF. The EF does not therefore explore all executions of the model. This means that either the unexplored executions are not relevant for the experimentation, or that the EF is not comprehensive enough.
► 2.2. There is a bias in the model or in the reference definition. This may be the case as the model considers different output event sequences than those expected by the EF. This means either that the simulation results are incorrect or that the EF assumptions are false.

When $L(EF,t) \subseteq \|L(M,t)\|_{Z_{M/EF}}$ is false, there are executions envisaged by the EF which are not model executions. Here again, we identify two cases (figure 3):

► 3.1 Some experiments are outside the usage domain of the simulation model. This may be the case if the EF considers different output event sequences than those accepted by the model. The EF therefore plans to explore simulation executions outside

the scope recommended by the model and the simulation results can no longer be guaranteed. The model or EF must be modified.
▶ 3.2 There is a risk concerning the completeness of the model. This may be the case if the EF considers different input event sequences than those supplied by the model. This means that either there is something to be learnt from the simulation if we reduce the uncertainties of the EF, or that the simulation overlooks the implementation of some cases.

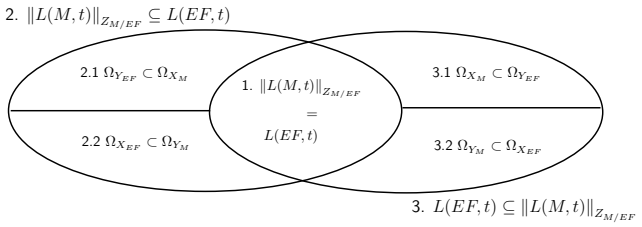

Figure 3: Level of EF/Model behavioural compatibility

Consider the following property on the two-slot toaster example: a ?push1 event is always followed by a !pop1 event twenty seconds later. This informal property is captured by the $LTL_{\triangleleft}$ (Di Giampaolo, Geeraerts, Raskin & Sznajder 2010) formula $\varphi \equiv \Box(?push1 \rightarrow \triangleright_{20}!pop1)$.

The executions below are executions on a time base $[0, 50]$ which satisfy φ :

▶ $(?push1, 0), (!pop1, 20)$
▶ $(?push1, 10), (!pop1, 30)$
▶ $(?push1, 0), (?push1, 10), (?push1, 12), (!pop1, 20)$
▶ $(?push1, 10), (?push2, 20), (?push1, 22), (!pop1, 30)$
▶ ...
Even with this simple example the set of executions which satisfy a given property on a given time base may explose rapidly. The experimental frame restrains the set of executions of a model to a subset of executions. The experimental frame which is the less rectrictive is the one which provides all input events required by the model in all states and accepts all output events provided by the model in all states.

We give hereafter an example for each case in figure 3:

▶ Given the two-slot toaster section 2.4 and $L(EF, [0, 50]) = (!push1, 0)(?pop1, 20)$ the condition 2 is not satisfied (condition 3 is satisfied). The EF expects a $!pop1$ event from the model 20 seconds after it sends a $?push1$ event. This execution can be applied to the model, however, there is a set of execution which has not been explored by the experimental frame. For example the event sequence $(?push1, 10), (!pop1, 30)$ has not been explored by the EF. This is case 2.1.

▶ Given the two-slot toaster section 2.4 and $L(EF, T) = (!push1, 0)(!push1, 10)(?pop1, 30)$ the condition 2 is not satisfied. The EF expects a $!pop1$ event from the model 20 seconds after it sends a second $?push1$ event while the model ignore the effect of further $?push1$. EF assumptions are false. This is case 2.2.

▶ Consider now a model without $\delta_x(T, ?push) = (T, 0)$ and $L(EF, T) = (!push1, 0)(!push1, 10)(?pop1, 20)$, the condition 3 is not satisfied. The EF considers a second $?push1$ event before next $!pop1$ event which is not accepted by the model. This is case 3.1.

▶ Given the two-slot toaster section 2.4 and $L(EF, T) = (!push1, 0)(!push1, 10)(?pop1, 20)(?pop1, 30)$, the condition 3 is not satisfied since the EF considers a sequence of input event which is not supllied by the model, i.e. the second !pop1 event will never occur. This execution can be applied to the model and the property is satisfied, however either there is a set of executions which are overlooked by the model or there is an uncertainty in the EF. This is case 3.2.

4 SOLVING TRACE INCLUSION

State space metric can be qualified by synchronous product between EF and model to identify the compatibility of event sequences between EF and model and verify the reachability of searched states given by the acceptor. It consists in restricting the model to the total event set of the experimental frame. In general a temporal property consists in driving the model in a given state and observing a state or an event onto the model output. We verify that the generator can exercise a given portion of the model according to a given property and that the model can propagate the information to the transducer, which in turn does not limit the model not accepting one of its output events.

4.1 Restrictions of FD-DEVS model

The restriction of an FD-DEVS component $A = \langle X, Y, S, s_0, \tau, \delta_x, \delta_{int}, \lambda \rangle$ to a subset of its total event set $Z' = X' \cup Y'$ such that $X' \subseteq X, Y' \subseteq Y, Oports' \subseteq Oports, Iports' \subseteq Iports$ is a FD-DEVS component $A' = \langle X', Y', S, s_0, \tau, \delta_{x/x'}, \delta_{int}, \lambda_{/y'} \rangle$ such that:

- $\delta_{int} : S \rightarrow S$ is the *internal state transition function* such that all δ_{int} defined in A are also defined in A';
- $\delta_{x/x'} : S \times X' \cup \{\varepsilon\} \rightarrow S$, where ε is the set of non-observable events, is the *external state transition function* such that

– for all $x \in X'$, for all $q \in Q, s \in S$, if $(q, x, s) \in \delta_x$ then $(q, x, s) \in \delta_{x/x'}$
– for all $x \in X' \setminus X$, for all $q \in Q, s \in S$, if $(q, x, s) \in \delta_x$ then $(q, \varepsilon, s) \in \delta_{x/x'}$

The restriction of an external state transition function masks the events which are not in X' with non-observable events ε.

- $\lambda_{/y'} : S \to Y'$ is the *output function* such that $\lambda_{/y'}(s) = \lambda(s)$ for all $y \in Y'$.

  The restriction of an output function keeps the output function for a subset of event and puts aside the output function of all other events.

The figure 4 illustrates a restricted version of the two-slot toaster FD-DEVS. In this reachability graph events which are not in the total event set of the experimental frame are either masked if there are input events or put aside if there are output events.

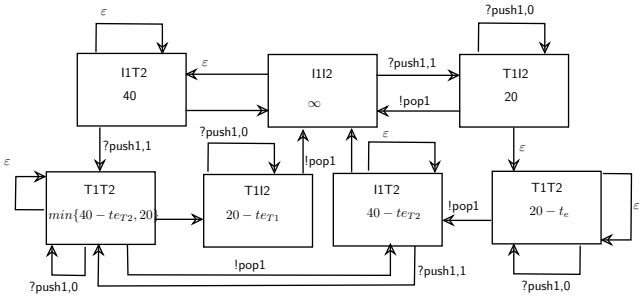

Figure 4: Reachability graph of the two-slot toaster FD-DEVS restricted to $Z_{EF} = (!push1, ?pop1)$

4.2 Synchronous product of two FD-DEVS atomic components

We verify the compatibility conditions between an EF and a model described by two FD-DEVS components only if their event sets are compatible. The two components synchronize on shared events, and keep all other events.

Two FD-DEVS components A and B can be compatible if all the events of interest of A can be synchronized with an event of B. The set of synchronizable events between component A and B is called $synch(A, B)$. The function $\Gamma_A : S_A \to 2^{Z_A}$ is a function that assigns to each state of A a subset of events that are available in that state. The function $\Gamma_B : Q_B \to 2^{Z_B}$ is a function that assigns to each sequential state of B a subset of events that are available in that state. Given the component A in a state $q_i = (s_i, ts_i, te_i)$ and the component B in a state $q_j = (s_j, ts_j, te_j)$, we say that an event $z \in Y_A \cap X_B$ can be synchronized, i.e. $z \in synch(A, B)$, if (1) $\exists(q_i, z) \in \Gamma_A$, (2) $\exists(q_j, z) \in \Gamma_B$ and (3) $min\{ts_i - te_i, ts_j - te_j\} = ts_i - te_i$. Given the component B in a state $q_i = (s_i, ts_i, te_i)$ and the component A in a state $q_j = (s_j, ts_j, te_j)$, we say that an event $z \in Y_B \cap X_A$ can be synchronized, i.e. $z \in synch(A, B)$, if (1) $\exists(q_i, z) \in \Gamma_B$, (2) $\exists(q_j, z) \in \Gamma_A$ and (3) $min\{ts_i - te_i, ts_j - te_j\} = ts_i - te_i$.

The composition of two FD-DEVS components A and B is a FD-DEVS component $N = \langle \delta, S, \tau, s_0 \rangle$ where: $s_0 = s_{0_A} \times s_{0_B}$ is the initial state, $S = S_A \times S_B$ is the set of states, $\tau : S \to min\{\tau_A(s_i), \tau_B(s_j)\}$ such that $(s_i, s_j) \in S$. $\delta : Q \times Z \to S$ is the state transition function such that:

$\delta = \{((s_i, s_j), z, (s'_i, s'_j))|(q_i, z) \in \Gamma_A \wedge (q_j, z) \in \Gamma_B \wedge z \in synch(A, B)\}$
$\cup\{((s_i, s_j), z, (s'_i, s'_j))|(q_i, z) \in \Gamma_B \wedge (q_j, z) \in \Gamma_A \wedge z \in synch(A, B)\}$
$\cup\{((s_i, s_j), z, (s_i, s'_j))|q_i \in S_A \wedge (q_j, z, q'_j) \in \delta_{y_B} \wedge z \notin synch(A, B)\}$

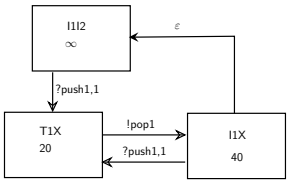

Figure 5: Reachability graph of the two-slot toaster FD-DEVS restricted to $Z_{EF} = (!push1, ?pop1)$ after state aggregation

Consider an EF that would generate a time event sequence $w_{[0,50]} = (!push1, 5)(?pop1, 25)(!push1, 35)(?pop1, 45)$. This EF means to validate the behavior of a *one slot toaster* but he has access only to the coupled model (*two slot toaster*). It is necessary to restrict the model. We have seen in figure 4 the restrictive model. The same model after state aggregation is given in figure 5. To obtain the aggregated model it is necessary to see if we can differentiate two states. For example here, we have restrict the model. *!pop2* becomes non-observable and *?push2* becomes non-controlable. Now, from the inital state (I1I2), when a *push1* occur the EF cannot know if the model is on state *T1I2* or in state *T1T2*. Then, we aggregate both states on one state called *T1X*. After the restriction, it is possible to make the synchronous product between the model and the EF, $\Delta((((I1, \infty, 0), (I2, \infty, 0)), \infty, 0), \omega_{[0,50]}) = (((imp, \infty, 15)(I2, \infty, 50)), \infty, 15) \in Q_{imp}$. At $t = 45$ the relation given previously $min\{t_{s_i} - t_{e_i}, t_{s_j} - t_{e_j}\} = t_{s_i} - t_{e_i}$ ($min\{20-0, 45-35\} \neq 20-0$ is not respected (condition (3)), and the synchronous product is not possible.

5 APPLICATION

5.1 Cruise controller model

In this section, we introduce the dynamic cruise controller model. This model uses two quantized integrators (Nutaro 2005). The principle of a quantized integrator is to discretize the space of state variables using a fixed value called the *quantum* size D. According to this quantum, a variable q can only take values among $q \pm kD$, where k is an integer. The solution $q(t)$ of a system described by a differential equation is approximated on a grid in the phase space of the system. The resolution of the phase space grid is D. The time h required to move from one phase space grid point to another, to occur on $q(t)$ is approximated and a state change will be informed only at this time.

The adaptive cruise controller uses a proportional-integral feedback control to maintain a given speed. The equation of the feedback control block is

$$\dot{q}_1(t) = k_P(v^* - q_1(t)) + k_I \int_{t_1}^{t_2} (v^* - q_1(t))\,\mathrm{d}t \quad (3)$$

where $\dot{q}_1(t)$ is the controlled acceleration to achieve the required speed, k_P and k_I are tuning parameters for proportional gain and integral gain respectively, v^* is the desired speed and $q_1(t)$ is the speed at time t.

Equation (3) can be written as a second-order ODE

$$\ddot{q}_1 + k_P\dot{q}_1 + k_I q_1 = k_I v^* \quad (4)$$

which is converted into two first-order ODEs

$$\dot{q}_1 = q_2 \quad (5)$$

$$\dot{q}_2 = k_I v^* - k_P q_2 - k_I q_1 \quad (6)$$

Equations (5) and (6) are described by two FD-DEVS components I1 and I2 respectively. There are five states variables: $ql \in \mathbb{R}$, the last output value of the integral; $q \in \mathbb{R}$, the current value of the integral; $\dot{q} \in \mathbb{R}$, the last known value of the derivative; $\sigma \in \mathbb{R}$, the time until the next output event and $phase \in \{ON, OFF\}$ indicates that the cruise controller is either ON or OFF.

The component I1 is such that $X = \{?turn\_on, ?brake, q_2\}$, $Y = \{q_1\}$, $S = \mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\{ON, OFF\}$, $\tau(ql, q, \dot{q}, \sigma, OFF) = \infty$, $\tau(ql, q, \dot{q}, \sigma, ON) = \sigma$, $s_0 = (0, 0, 0, \infty, OFF)$.
$\delta_x((ql, q, \dot{q}, \sigma, OFF), ?turn\_on) = ((ql, q, \dot{q}, \sigma, ON), 1)$
$\delta_x((ql, q, \dot{q}, \sigma, ON), ?brake) = ((ql, q, \dot{q}, \sigma, OFF), 1)$
$\delta_x((ql, q, \dot{q}, \sigma, ON), ?turn\_on) = ((ql, q, \dot{q}, \sigma, ON), 0)$
$\delta_x((ql, q, \dot{q}, \sigma, OFF), ?brake) = ((ql, q, \dot{q}, \sigma, OFF), 0)$
$\delta_x((ql, q, \dot{q}, \sigma, ON), q_2) =$
$((ql, q + \dot{q} * te, q_2, \frac{D-|q+\dot{q}*te-ql|}{|q_2|}, ON), 1)$
$\delta_y(ql, q, \dot{q}, \sigma, ON) = ((qn, qn, q_2, \frac{D}{|q_2|}), qn)$

with $qn = ql + d * sgn(\dot{q})$ the next value of the integral.

The function $sgn(v)$ returns -1 if $v < 0$, 0 if $v = 0$ or 1 if $v > 0$.

The component I2 is such that $X = \{?turn\_on, ?brake, q_1, v^*\}$, $Y = \{q_2\}$, $S = \mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\{ON, OFF\}$, $\tau(ql, q, \dot{q}, \sigma, OFF) = \infty$, $\tau(ql, q, \dot{q}, \sigma, ON) = \sigma$, $s_0 = (0, 0, 0, \infty, OFF)$.

$\delta_x((ql, q, \dot{q}, \sigma, OFF), ?turn\_on) = ((ql, q, \dot{q}, \sigma, ON), 1)$
$\delta_x((ql, q, \dot{q}, \sigma, ON), ?brake) = ((ql, q, \dot{q}, \sigma, OFF), 1)$
$\delta_x((ql, q, \dot{q}, \sigma, ON), ?turn\_on) = ((ql, q, \dot{q}, \sigma, ON), 0)$
$\delta_x((ql, q, \dot{q}, \sigma, OFF), ?brake) = ((ql, q, \dot{q}, \sigma, OFF), 0)$
$\delta_x((ql, q, \dot{q}, \sigma, ON), q_1) =$
$((ql, q + \dot{q} * te, k_I v^* - k_P q_2 - k_I q_1, \frac{D-|q+\dot{q}*te-ql|}{|k_I v^* - k_P q_2 - k_I q_1|}, ON), 1)$
$\delta_x((ql, q, \dot{q}, \sigma, ON), v^*) =$
$((ql, q + \dot{q} * te, k_I v^* - k_P q_2 - k_I q_1, \frac{D-|q+\dot{q}*te-ql|}{|k_I v^* - k_P q_2 - k_I q_1|}, ON), 1)$
$\delta_y(ql, q, \dot{q}, \sigma, ON) =$
$((qn, qn, k_I v^* - k_P q_2 - k_I q_1, \frac{D}{|k_I v^* - k_P q_2 - k_I q_1|}), qn)$

At each iteration, the time σ required to move from one phase space grid point to another, to occur on q of each ODE is approximated by $\frac{d-|q+\dot{q}*e-ql|}{|f(q,x)|}$. The time t is advanced to the next change to occur, i.e. the slower σ of both ODE.

5.2 Experiments with Uppaal

This part implements cruise controller on a verification platform. Uppaal (Behrmann et al. 2004) is an environment for modeling, simulation and verification of real time embedded systems. Uppaal is composed of three parts: an editor to describe system behavior, a simulator used to generate some simulation traces of the system and the model-checker, who can check invariant and reachability properties by exploring the state-space of the system.

The cruise controller, as presented above, has been the subject of a study where the following conditions have been extracted:

Requirements:
► 1. After each speed change, the vehicle must be stabilized in less than 7 seconds.
► 2. Braking disables the cruise control immediatly.
► 3. The acceleration must be less than 2 $m.s^{-2}$
► 4. The speed of 25 $km.h^{-1}$ (*safe_speed*) should never be exceeded for a set from 0 to 20 $km.h^{-1}$.
► 5. The cruise control can take every speed from 0 to 20 $km.h^{-1}$
This study has shown that the differential equation (3) allows to respect all these requirements.

Figure 6 illustrates the requirements for an expected speed of 20 $km.h^{-1}$.

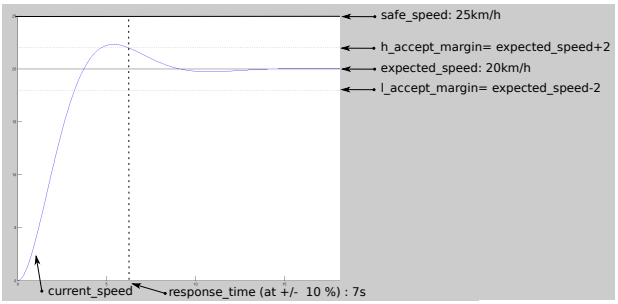

Figure 6: Illustration of stakeholders requirements for 20 $km.h^{-1}$

5.3 Model

The solution adopted for the cruise controller architecture has led to a discrete event solution of equation (3). Currently, a model is defined by a modeling language accepted by UPPAAL.

5.3.1 Model

As mentioned before, the cruise controller calculation module has two components. I1 and I2 are two quantized integrators, they compute the speed of the vehicle to maintain the expected speed. We present in figure 7, one of the two integrators. The two being similar, the reader has only to reverse lambda1 and lambda2 to obtain the second integrator. Initial state of model is *A1*, the vehicle is stationary and signal *turn_on* is waited. *turn_on* causes the transition (A1A2) and initializes (*initialize()*) each variable according to *expected_speed*. Synchronisation channels *lambda1* and *lambda2* permits to synchronize both integrators. If internal clock (x) belonging to *I1* (res.*I2*) is equal to the variable quantum size (h) then it synchronises with *I2*(res.*I1*) to exange values from precedent integration step. After synchronisation both integrators calculate in the same time new part of differential equation (*deltint()* and *deltext()*). The transition from A2 to *A3* can be explained as follow: *Lambda2?* permits to synchonise *I2* to *I1*. The part below *lambda2?* permits to catch the time and record it on an integer variable (e).

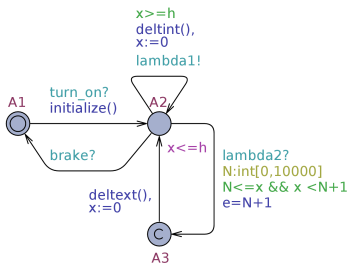

Figure 7: UPPAAL model of cruise controller

Design choices have led to add restrictions to the specifications previously established, and bring out a new requirement (quantification of first requirement):
▶ 5.1 Quantification → Speed is called stable when it reaches more or less than 2 $km.h^{-1}$ of speed requested (*accept_margin*).

5.4 Experimental Frame (EF)

The experimental frame should permit to validate the model. It should allow one to check if it is consistent with requirements? Does it allows to verify some properties ?
The EF embraces the model, more or less capable, it allows exploring some or all model part where we want to check the model behavior. The simulation experimenter must develop the shorter solution permitting to validate the model. Indeed, it is common to wait few minutes to verify property with model-checking method. This is the reason why we advocate to create a shorter solution to validate properties. Sometime, we do not have choice, and simulation experimenter wants to create exhaustive scenarios to explore some parts of model.
EF is composed of three parts, the generator which generates signals for the model, the transducer that recovers signals from model and the acceptor who permits to validate model property. This one uses information delivered by generator and transducer. We now discuss each of the three parts of the experimental frame in the context of cruise control.

5.4.1 Generator

As we said the generator permits to create one scenario execution more or less capable. In other words, the EF (through the generator) permits to explore more or less the model. Figure 8, shows generator, who generates one speed instruction for the model (*expected_speed := wanted_speed*) and 10000 time unit after, it generates *brake* signal.

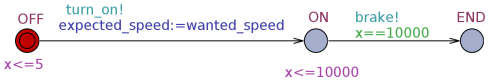

Figure 8: Simple generator (*less capable*)

Now, we will focus on the second generator in figure 9, which is more capable. This generator permits to send different speed instructions to the model, after that, it waits 15000 time unit, and generate new speed instruction and this during 100000 time units.

The scenario described previously is partially included in this one. Both scenarios permit to explore some parts of the model to detect if this model satisfy some requirement properties. If we take requirements defined previously, we know already it is impossible

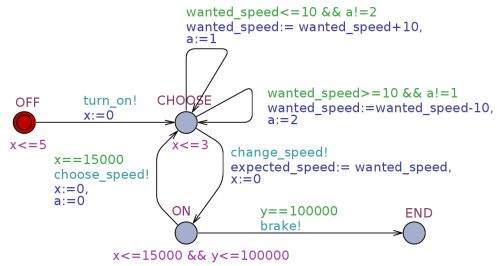

Figure 9: Complex generator (*more capable*)

to verify requirement number 2. Indeed, none of these scenarios establishes a brake signal for each states of the model. The simulation experimenter who developed this solution adds a new restriction, indeed the verification of properties with all possible speed instructions is to long. ▶ New requirement:1.1 Quantification → *Expected-speed* can take only 3 values 0, 10 and 20 $km.h^{-1}$.

5.4.2 *Transducer*

The generator model is established. Depending on the considered scenario, and properties that we want to verify, we must now come to capture available output signals. Sometimes, it is the case here, an internal signal of the model would be useful but is unfortunately not observable (the derivative of the speed). It is therefore necessary to rebuild it. One of requirements says that the accelaration must be less than 2 $m.s^{-2}$ (requirement number 3). We can see in figure 10 reconstruction of acceleration signal (*acc*). Every 200 time unit, there is velocity derivation.

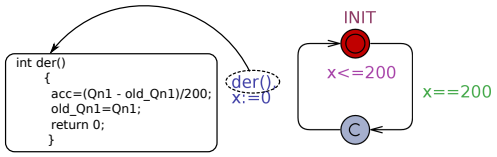

Figure 10: Transducer 1

The second transducer (figure 11) gives us information about states visited by the generator and know if we are on speed choose phases or not.

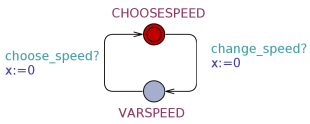

Figure 11: Transducer for complex scenario

5.4.3 *Acceptor*

Finally, the acceptor is defined. Indeed, from specification to the model validation, some requirements appeared and they conduce the simulation experimenter to design specific equations to validate the model. Take the example of the acceleration to try to validate the requirement 2. The requirement implies that the acceleration must always be less than 2 $m.s^{-2}$. We need a scenario showing a significant speed changes. It will also collect speed information. Simple scenario (figure 8), with the transducer who captures the acceleration (figure 10) from the only given information (speed) permits to validate this requirement. The requirement 2 translated in UPPAAL logical equation as following:

$$A[]\ Generator.y >= 0\ imply\ Transducer.acc <= 2 \quad (7)$$

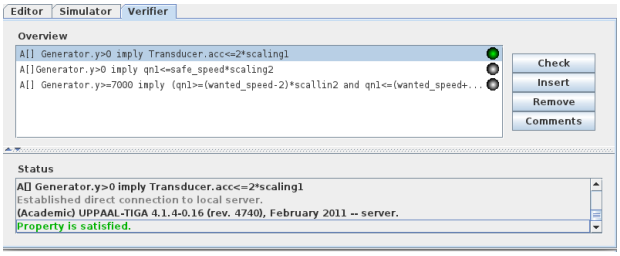

Figure 12: Acceptor

The model-checker UPPAAL indicates that the property is satisfied. One may pose the question if these results are credible? Indeed, it is difficult to determine whether an error is not incorporated into our model, our property or in our experimental frame, which can create a false positive. Here, we are not trying to provide an exhaustive list of possible problems, but simply to provide one case for each type of results. Figure 13 shows each kind of validity of result. Each trace was created with different pair model-experimental frame, errors included in couples are specified later. The specification defines an obligation of property *P1* appearance in a specific state (bottom right of each trace).

▶ 1 (True-Positive): It is the nominal case, the specification is correct, the model and the experimental frame also right. In this example, the acceleration does not exceed 2 $m.s^{-2}$. The simulation experimenter can use the result and validate the model.

▶ 2 (True Negative): In this case the model is wrong. The experimental frame can generate all the trace but the property is not present. The model is not consistent with the specifications. The simulation experimenter can use the result and modify the model. We can imagine one case in our model does not respect the maximum acceleration.

▶ 3 (False Positive): A critical case which may lead to validate a model that is inconsistent with the specification. The simulation experimenter misread the specification and does not check the right property. The simulation experimenter will use results and validate a non-compliant model. If we take our earlier

example, the simulation experimenter does not used the same system unit as the editor of the specification (feet / meters), he does not check the same property.

► 4 (False Negative): Here, the experimental frame does not allows access to the full model. The property is good but can not be explored. The simulation experimenter will announce non-compliant model or non-compliant experimental frame without being able to give more detail. This example shows the utility of

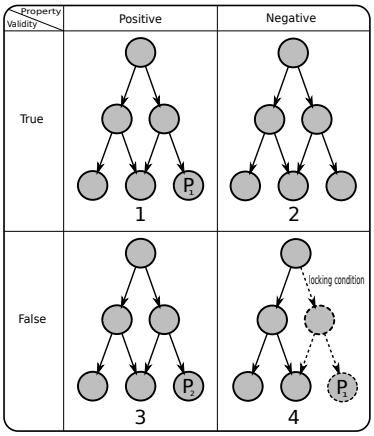

Figure 13: Types of result

an experimental frame in order to validate the model. This helps to limit the exploration of the model to properties that we want, without using some parts of the model useless for verification. We have seen through this example the emergence of new requirements after all different phases, during modeling, experimental frame design or validation.

6 CONCLUSION

In this paper, a state space metric is qualified by trace inclusion to measure the extent of design verification provided by a set of simulation procedures. We compare the set of traces of the model, which satisfy the properties given by the acceptor, with the set of traces given by the generator and the transducer. The coverage metric can be defined with five quantities (figure 8): exact matching (case 1), test coverage risk (case 2.1), bias in the simulation (case 2.2), wrong usage of the model (case 3.1), model completness risk (case 3.2). Further work would refine this quantities and explore others gauges for this metric. These metrics are ordered, i.e. measuring trace inclusion imply that ports and event sequences are compatibles.

With inclusion trace approach, we can identify a priori (before simulation execution) whether a simulation may or may not be executed (indicating to the simulationist whether the guarantees given by the model would satisfy its simulation objectives of use). Furthermore, this approach can be performed either on the development from scratch of a simulation necessary and sufficient to satisfy an intended purpose, or on the reuse of an already existing simulation model to satisfy an intended purpose. Developers are free to associate a model with different experimental frames, each corresponding to a particular simulation objective of use. If we consider that morphisms between models of a same system are well documented, juggling between abstractions and previous simulation results allows verifying the applicability of a "new" simulation without executing the simulation. In the same way, one can put another model of the system of interest in the experimental frame and verify by matching if this model can accommodate the experimental frame.

References

Albert, V., Nketsa, A. & Seguin, C. (2010). Verifying trace inclusion between an experimental frame and a model, *DEVS Integrative Modeling and Simulation Symposium* .

Behrmann, G., David, A. & Larsen, K. G. (2004). A tutorial on UPPAAL, *4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, number 3185 in *LNCS*, Springer–Verlag, pp. 200–236.

Di Giampaolo, B., Geeraerts, G., Raskin, J. & Sznajder, N. (2010). Safraless procedures for timed specifications, *in* Springer (ed.), *8th International Conference on Formal Modelling and Analysis of Timed Systems*, Vol. 6246, pp. 2–22.

Hwang, M. H. & Zeigler, B. (2009). Reachability graph of finite and deterministic devs networks, *IEEE Transactions on Automation Science and Engineering,* **6**(3): 468 –478.

Nutaro, J. (2005). Discrete event simulation of continuous systems, *Handbook of Dynamic Systems Modeling.*

Traoré, M.K.and Muzzy, A. (2006). Capturing the dual relationship between simulation models and their context, *Simulation Modelling Practice and Theory* **14**(2): 126–142.

Zeigler, B. P., Praehofer, H. & Kim, T. G. (2000). *Theory of Modelling and Simulation*, Academic Press, San Diego, California, USA.