Adapting Model-Based Testing Techniques to DEVS Models Validation

Diego A. Hollmann[1,2], **Maximiliano Cristiá**[1] **and Claudia Frydman**[1,2]
[1]**FCEIA - UNR, CIFASIS, Rosario, Argentina.** | [2]**LSIS - AMU, Marseille, France.**
{hollmann, cristia, frydman}@cifasis-conicet.gov.ar

Keywords: Model Validation, DEVS, Discrete Event Simulation, Simulation Criteria, Software Engineering

Abstract

One way to validate a model of a system against its requirements is to simulate it several times under different conditions and observe its behavior in order to compare it with what the system is supposed to do. This paper presents a technique that is widely used in the software testing community but barely known in the modeling and simulation community. We formalize a family of criteria to conduct DEVS model simulations in an ordered way and to cover the most significant simulation scenarios to increase the confidence that the model has been properly validated.

1. INTRODUCTION

During the simulation of a model it would be desirable to perform all possible simulation scenarios and compare these behaviors against the requirements. Unfortunately, exhaustive simulation is impractical in almost all projects, since it involves an infinite number of simulation scenarios. Considering this, the selection of an appropriate set of simulation scenarios is a crucial issue to assure that a model has been simulated enough in order to increase the confidence that it complies the requirements.

In the software testing field there is an analogous scenario. According to Utting and Legeard [13] software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior. There are several works in this area trying to formalize the software testing process. In [6] Hieorns et al review the state of the art of how formal specifications are used to assist testing. Being so important in software development, the testing process has been improved up to the point of turning it almost automatic, in many cases obtaining quite good results.

An important part of this automation is possible because there are formally defined testing criteria. Some of these criteria are based on the exploration of the specification and others on the exploration of the source code of the program. Particularly, we work with the formers and we propose to adapt them to model simulations.

Another relevant work was presented by de Souza et al in [10] where they propose a family of coverage testing criteria for specifications based on Statecharts. This formalism is mostly used to specify the behavior of reactive systems and has some similarities, in this sense, with DEVS [14].

Most of these works belong to a subfield of testing known as model-based testing (MBT). One relevant MBT method is the Test Template Framework (TTF), presented in [11] by Stocks and Carrington and implemented by Cristiá et al in [4] where a framework is introduced to formally define test data sets providing structure to the testing process.

We believe that the TTF, and other MBT methods as well, can be applied in the model simulation process, but they are almost unknown by the modeling and simulation community. In general, the simulation of models is performed according to the experience or intuition of a specialist. Therefore no formal guidelines or criteria are being followed to define an adequate set of simulations, making the simulation process informal and error prone. On the other hand being the selection of the simulations a “manual” activity it can not be automatized in a significant way until it is not formalized.

The idea of this work is to introduce those techniques, but changing the testing setting by a simulation setting, particularly by applying these techniques to DEVS models which requires to take into account its characteristics. Taking as starting point the TTF for the data structures of a DEVS model and the work of Souza et al [10] for the reactive aspects of the model, we describe a set of criteria to conduct simulations of DEVS models.

2. CONTRIBUTION OF THIS PAPER

The major contribution of this paper is to present an alternative method to conduct the simulation process of DEVS models in order to validate them, introducing a very well known technique in the *testing world* into the *model and simulation community*.

Currently, as we mentioned in the introduction, the simulation process is preformed according to the experience or intuition of a specialist without following any formal guidelines. We believe that simulating DEVS models following the techniques presented in this work will increase the confidence that the model is correct validating aspects or functions of the model that could be overlooked by the specialist.

3. RELATED WORK

The family of simulation criteria proposed in this work is mainly based on the testing framework of Stocks and Carrington mentioned in the previous section. We adapted some of those testing techniques to the model simulation context. Another work that has inspired ours is that of de Souza et al, where they present a family of coverage criteria for Statecharts specifications.

Labiche and Wainer in [7] made a review of the Verification and Validation (V&V) of discrete event system models. They propose to apply or adapt existing software testing techniques to the V&V of DEVS models. Just, what we intend to do with the present work is to adapt some testing techniques in order to validate DEVS models.

On the other hand, there are several works that use verification techniques, like model checking to verify the correctness of a model. For instance, Napoli and Parente [9] present a model-checking algorithm for Hierarchical Finite State Machines as an abstract DEVS model. They also focus on the generation of simulation scenarios for DEVS, but as counter-examples obtained by the application of their model-checking algorithm. The main problem of model-checking techniques is the so-called *state explosion*[1].

Another recent work that applies verification techniques over discrete event simulation is [5] where da Silva and de Melo presents a method to perform simulations orderly and verify properties about them using transitions systems. They focused their work on the verification of properties by simulations but not on the generation of simulations in order to validate the model.

Li et al in [8] present a framework to test DEVS tools. In their framework they combine black-box and white-box testing approaches. Actually, this work is not really related to ours because they do not validate or verify a DEVS model, whereas they test DEVS implementations. However, it is useful to see how they introduce software testing techniques in the DEVS world.

4. FAMILY OF SIMULATION CRITERIA

As we mentioned in the introduction, this work intends to introduce some model-based testing techniques into the model simulation world, specially for DEVS models. A DEVS Model is defined by the structure [14]:

$$M = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta)$$

where:

- X is the set of input event values, i.e., the set of all possible values that an input event can assume;
- Y is the set of output event values;
- S is the set of state values;
- δ_{int}, δ_{ext}, λ and ta are functions that define the system. dynamics.

4.1. Domain Partition

Given a DEVS model, usually the set of possible simulations is infinite, no matter if the model has finite sets of inputs, states and outputs. Taking the idea of the TTF we adapt the partition strategy to DEVS models. Given the set of all possible simulations of a model, we propose to divide it into equivalence classes by applying one or more *criteria*. We call each of these divisions *simulation configuration class (SCC)*.

We present briefly the *uniformity hypothesis* [2], statement related to the uniformity of a program's behavior on ranges of data. Given a certain subset or range of the input data the hypothesis assumes the program has the same behavior for each element in that range. In software testing Hierons et al assert "if the program is correct for one set of input values in this range, then it is correct for all such input values" [6].

Therefore, we say that those classes, in which the set of possible simulations was divided, are equivalence classes because it is assumed that the model has uniform behavior for some subsets of simulations. If the uniformity hypothesis holds and an error in the model is found for some simulation of a given class, then the same error must be found with any simulation of the class.

As in [11] some strategies or criteria assume that every element of a class is equivalent to all the others for the simulation of the model. However, as noted by Stocks and Carrington, this assumption is often invalid and they propose to apply repeatedly the strategies in order to partition the classes into subclasses until the tester (simulator in this case) considers that the classes are small enough or each functionality of the model is covered [4].

Each element of a SCC has all the information necessary to perform a specific simulation in order to observe a particular behavior of the model. These elements of the class consist of an initial state (to initialize the simulation) and an input sequence containing the events to simulate and its corresponding time (when to simulate that event).

Following this idea, to describe a SCC we need to define the set of possible initial states and the set of possible input sequences for the simulation. An input sequence is a sequence of pairs $(event, time)$. Therefore a class can be defined by one of the followings forms:

- a set of states $S_i \subseteq S$ and a sequence of pairs (x, t), with $x \in X$ and $t \in \mathbb{R}_0^+$, or
- a state $s \in S$ and a set of sequences of pairs (x_i, t_i) with $x_i \in X$ and $t_i \in \mathbb{R}_0^+$, or
- a combination of both, i.e. a set of states $S_i \subseteq S$ and a set of sequences of pairs (x_i, t_i) with $x_i \in X$ and $t_i \in \mathbb{R}_0^+$.

The first form defines a SCC by a set of possible initial states and by a unique input sequence. In the second one the initial state is unique and it is given a set of input sequences. The third form is more generic, combining the previous ones, i.e. the classes are described by a set of possible initial states and a set of input sequences. The total class of simulations for a given DEVS models is defined by S and $\{es \in \text{seq}\ (X \times \mathbb{R}_0^+)\}$, i.e. all possible states and all possible input sequences.

From now on, to denote the set of possible initial states we will use *IniSt* and for the set of input sequences *InpSeq*. Defining these two sets, is enough to define a SCC.

4.2. Partition Criteria

Below we present the different partition criteria proposed in this work. The criteria are applied to different aspects of DEVS models. Some criteria apply to the external transition function definition, others to the internal transition function definition and others to the definition of the states or input and outputs sets.

It is important to mention that at any time of the partition process it is possible that some criteria could not generate new partitions. Furthermore, it does not matter the order in which the criteria are applied, the result is the same.

4.2.1. Transition Functions Defined by Cases

It is very common to define the external and internal transition functions by cases. The first and more intuitive criterion is to partition the set of possible simulations into several classes, one for each case in the definition of the function.

Let δ_{ext} and δ_{int} be the transition functions of a DEVS model defined by cases:

$$\delta_{ext}(s,e,x) = \begin{cases} expr^1_{ext}(s,e,x) \text{ if } P^1_{ext}(s,e,x) \\ \vdots \\ expr^m_{ext}(s,e,x) \text{ if } P^m_{ext}(s,e,x) \end{cases}$$

$$\delta_{int}(s) = \begin{cases} expr^1_{int}(s) \text{ if } P^1_{int}(s) \\ \vdots \\ expr^m_{int}(s) \text{ if } P^m_{int}(s) \end{cases}$$

where $expr^i_{ext}$ and $expr^i_{int}$ are the result of the function if the proposition P^i_{ext} or P^i_{int}, respectively, holds. This criterion proposes to generate one partition for each proposition in the definition of the internal and external transition function.

$IniSt_i = \{s \in S \mid P_j, j \in [1,m]\}$
$InpSeq_i = \{es \in EvSeq \mid P_j, j \in [1,m]\}$
with $EvSeq = \text{seq } (X \times \mathbb{R}_0^+)$.

If the function is well defined, i.e. there is one and only one proposition that holds for the same input values, these classes should be disjoint.

4.2.2. Sets Defined by Extension

To describe DEVS models it is very usual to define some sets by extension (states, input events or output events) , i.e. listing the elements of the set. Necessarily these sets will be finite and relatively small, therefore this criterion proposes to simulate all scenarios where appears at least once each element of these sets.

For example, let S be a set defined by extension:
$S = \{s_1, s_2, ..., s_n\}$
for each $s_i \in S, i \in [1,n]$ one SCC should be defined.

If S represents the set of possible states of the model, the partitions defined applying this criterion should be:
$IniSt_i = \{s_i\}$,
$InpSeq_i = \{seq \in EvSeq\}$.

4.2.3. Standard Partitions

In almost all models, different mathematical operators appear in the definitions of the model elements (transition functions, time advance function, state) and they can be simple (addition, sets union) or more complex (functions defined in a programming language or pseudo-code). Each operator has a particular input domain and this criterion proposes to partition the set of possible simulations according to the partitions associated to each operator. Therefore, for each operator in the model a standard partition should be defined. For example, for the operator $<$ ($a < b$), the standard partition could be [4]:

$$\begin{array}{ccc} a<0,b<0 & a<0,b=0 & a<0,b>0 \\ a=0,b<0 & a=0,b=0 & a=0,b>0 \\ a>0,b<0 & a>0,b=0 & a>0,b>0 \end{array}$$

Let $\theta(x_1, ..., x_n)$ be an operator of arity n with the associated partitions $P_1(x_1, ..., x_n), ..., P_m(x_1, ..., x_n)$. When θ appears in an operation the set of possible simulation must be partitioned using the standard partitions associated with it is:
$IniSt_i = \{s \in S \mid P_i\}$,
$InpSeq_i = \{seq \in EvSeq \mid P_i\}$.

4.2.4. Domain Propagation

This is a particular criterion, since it does not generate new partitions by itself. The purpose of it is to obtain standard partitions of complex operators combining the standard partitions of simpler sub-operators.

Each sub-operation has input domain partitions of its own which are ignored by standard partitions criterion if it is applied to the complex operator. Using domain propagation the input domain partition of sub-operations are propagated to the higher level. [12]

Let $\Box$ be a complex operator defined as: $\Box(A,B,C) = (A \triangle B) \Diamond C$ where $\triangle$ and $\Diamond$ are simple operators.

Let us suppose that $\triangle$ and $\Diamond$ have the following standard partitions:

$$EP^{\triangle}(S,T) = D^{\triangle}_1(S,T) \vee \ldots \vee D^{\triangle}_n(S,T)$$
$$EP^{\Diamond}(U,V) = D^{\Diamond}_1(U,V) \vee \ldots \vee D^{\Diamond}_k(U,V)$$

We apply first $EP^{\triangle}$ to the sub-expression $(A \triangle B)$, replacing the formal parameters appearing in $EP^{\triangle}$ by A and B respectively:

$$EP^{\triangle}(A,B) = D^{\triangle}_1(A,B) \vee \ldots \vee D^{\triangle}_m(A,B)$$

with $m \leq n$.

Afterwards we do the same with $EP^{\Diamond}$, obtaining:

$$EP^{\Diamond}(A \triangle B, C) = D^{\Diamond}_1(A \triangle B, C), \vee \ldots \vee D^{\Diamond}_j(A \triangle B, C)$$

with $j \leq k$.

Finally, we combine both propositions obtained and simplify:

$$EP^{\square} = EP^{\triangle}(A,B) \wedge EP^{\lozenge}(A\triangle B, C)$$

4.2.5. Time Partitions

This criterion could be included in *Standard Partition*, but we mention it as a separate criterion because in timed formalisms, like DEVS, this is a crucial issue. It is very common to use additional variables to model the time. Furthermore, one characteristic of these models is that the elapsed time appears in the external transition function definition as a variable.

Therefore, it is necessary to bear in mind not only which events must be simulated but when to simulate them.

The intention of this criterion is to simulate different scenarios where events occur at different moments, to verify the interaction of those variables used to simulate the time and the interaction between these ones with the elapsed time (for external transitions).

4.3. Combining Partitions

Once all the criteria has been applied it is recommended to combine the resulting SCCs, i.e. conjoining two or more partitions, throwing away those conjuncts whose result is an empty set (when the intersection is empty). This allow to obtain new SCCs.

Let us see this with a toy example. Let $M_T = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta)$, with $X = \mathbb{N}$ and $S = \mathbb{N} \times \mathbb{Z}$, be the model of some system and suppose that after applying some criteria the following SCCs are obtained:

- $IniSt = \{(n,z) : \mathbb{N} \times \mathbb{Z} \mid n > 10 \wedge z < 10\}$, $InpSeq = \{\langle (1,t_1),(5;t_2)\rangle \mid t_1, t_2 \in \mathbb{R}_0^+\}$
- $IniSt = \{(n,z) : \mathbb{N} \times \mathbb{Z} \mid n > 15 \wedge z < -10\}$, $InpSeq = \{\langle (1,t_1),(5;t_2)\rangle \mid t_1, t_2 \in \mathbb{R}_0^+\}$

Therefore, conjoining these two partitions the following ones are obtained (we only show some partitions for brevity):

- $IniSt = \{(n,z) : \mathbb{N} \times \mathbb{Z} \mid 10 < n < 15 \wedge 0 < z < 10\}$, $InpSeq = \{< (1,t_1),(5;t_2) > \mid t_1, t_2 \in \mathbb{R}_0^+\}$
- $IniSt = \{(n,z) : \mathbb{N} \times \mathbb{Z} \mid n = 15 \wedge 0 < z < 10\}$, $InpSeq = \{< (1,t_1),(5;t_2) > \mid t_1, t_2 \in \mathbb{R}_0^+\}$
- $IniSt = \{(n,z) : \mathbb{N} \times \mathbb{Z} \mid n > 15 \wedge z < -10\}$, $InpSeq = \{< (1,t_1),(5;t_2) > \mid t_1, t_2 \in \mathbb{R}_0^+\}$

4.4. Simulation Sequencing

Once the criteria proposed before have been applied is desirable to generate a sequence of simulations. Here we propose a tactic to do this profiting the results of the obtained from criteria.

The main progress is led by the following idea: select one initial state from one of the SCCs and simulate the associated event. Once the event has been simulated and a new state is reached there are two alternatives, wait for an internal transition occurs or simulate another event. For both cases the selection should not be "random" it must be chosen from one SCC. The new state reached before must pertain to the set of possible initial states of the SCC selected. Afterwards, the process continues repeatedly choosing an event or waiting for an internal transition until at least one element of each SCC has been simulated.

5. CASE STUDY

In this section we show the application of the criteria to an example. First we present the requirements of the system, then we describe the DEVS model and finally the simulations that follow the application of the criteria.

This particular system consists in the control of a soda can vending machine. The machine accepts coins of \$ 0.05, \$ 0.10, \$ 0.25, \$ 0.50 and \$ 1. It gives change, optimizing it (i.e. giving the less coins as possible).

The machine has cans of two different prices (normal and diet), and the system that controls the machine must comply with the following requirements:

- During an operation, if after T_{ret} units of time no coin is introduced into the machine or no soda is selected, the machine returns all the money that has been introduced.
- Prices of sodas increase as time passes. Every T_{incr} units of time both prices are increased in \$ 0.05.
- if the returned money is not collected by the user after T_{chg} units of time the machine recovers it.
- The machine has a display that shows the amount of money introduced or the change after an operation.
- At any time, before selecting a soda, the user can cancel the operation and the machine returns the money.

Some additional temporal requirements (in particular the second one) were artificially included in order to have more time variables interacting in the model allowing to increase the partitions obtained by applying the criteria of the previous section.

In the figure 1 we describe the DEVS model for this example.

5.1. Simulations

At first we show a possible way to generate the SCCs (partitions of the set of all possible simulations) for this example using the criteria presented before. As we mention in the previous section, these classes are equivalence classes, therefore it is enough to select one element of each class to simulate the functionality or behavior of the model that the

$M_{sv} = (S, X, Y, \delta_{int}, \delta_{ext}, \lambda, ta)$

$S = MachState \times Display \times OpTime \times NormalPrice \times DietPrice \times IncrTime \times MoneyStorage \times OperationMoney \times MoneyReturned$

where:

$MachState = \{\text{idle}, \text{operating}, \text{finishingOp}, \text{cancelingOp}, \text{waitingRetChange}\}$

$OpTime = \mathbb{R}_0^+ \cup \{\infty\}$

$Display = IncrTime = NormalPrice = DietPrice = \mathbb{R}_0^+,$

$MoneyStorage = OperationMoney = MoneyReturned = Coins1d \times Coins50c \times Coins25c \times Coins10c \times Coins5c$

where:

$Coins1d = Coins50c = Coins25c = Coins10c = Coins5c = \mathbb{N}_0$

$X = \{5, 10, 25, 50, 100, \text{getNormal}, \text{getDiet}, \text{cancel}, \text{moneyRetreated}\}$

$Y = Display \times MoneyReturned$

$$\delta_{ext}((m,d,ot,np,dp,it,ms,om,mr),e,x) =$$
$$\begin{cases} (\text{operating}, d+x, 0, np, dp, it-e, ms, om \oplus x, \bar{0}) & \text{if } x \in \{100,50,25,10,5\} \wedge m \in \{\text{idle}, \text{operating}\} \\ (\text{finishingOp}, d-np, 0, np, dp, it-e, ms \oplus om, \bar{0}, \bar{0}) & \text{if } x = \text{getNormal} \wedge d \geq np \\ (\text{finishingOp}, d-dp, 0, np, dp, it-e, ms \oplus om, \bar{0}, \bar{0}) & \text{if } x =' getDiet' \wedge d \geq dp \\ (\text{cancelingOp}, d, 0, np, dp, it-e, ms, \bar{0}, om) & \text{if } x = \text{cancel} \\ (\text{idle}, 0, 0, np, dp, it-e, ms, \bar{0}, \bar{0}) & \text{if } x = \text{moneyRetreated} \end{cases}$$

$$\delta_{int}((m,d,ot,np,dp,it,ms,om,mr)) =$$
$$\begin{cases} (\text{operating}, d, T_{ret}, np, dp, it, ms, om, \bar{0}) & \text{if } m = \text{operating} \wedge ot < it \\ (\text{waitingRetChange}, d, T_{chg}, np, dp, it, ms \ominus (d \oslash ms), om, d \oslash ms) & \text{if } m = \text{finishingOp} \wedge ot < it \\ (\text{waitingRetChange}, d, T_{chg}, np, dp, it, ms, \bar{0}, mr) & \text{if } m = \text{cancelingOp} \wedge ot < it \\ (\text{idle}, 0, \infty, np, dp, it, ms, \bar{0}, \bar{0}) & \text{if } m = \text{waitingRetChange} \wedge ot < it \\ (\text{idle}, 0, \infty, np, dp, it, ms, \bar{0}, \bar{0}) & \text{if } m = \text{idle} \wedge ot < it \\ (m, d, ot-it, np+0.05, dp+0.05, T_{incr}, ms, om, mr) & \text{if } it < ot \end{cases}$$

$\lambda((m,d,ot,np,dp,it,ms,om,mr)) = (d, ms)$

$ta((m,d,ot,np,dp,it,ms,om,mr)) = min(ot, it)$

$$(coins1d, coins50c, coins25c, coins10c, coins5c) \oplus x = \begin{cases} (coins1d+x, coins50c, coins25c, coins10c, coins5c) & \text{if } x = 100 \\ (coins1d, coins50c+x, coins25c, coins10c, coins5c) & \text{if } x = 50 \\ (coins1d, coins50c, coins25c+x, coins10c, coins5c) & \text{if } x = 25 \\ (coins1d, coins50c, coins25c, coins10c+x, coins5c) & \text{if } x = 10 \\ (coins1d, coins50c, coins25c, coins10c, coins5c+x) & \text{if } x = 5 \end{cases}$$

$$(coins1d, coins50c, coins25c, coins10c, coins5c) \ominus x = \begin{cases} (coins1d-x, coins50c, coins25c, coins10c, coins5c) & \text{if } x = 100 \\ (coins1d, coins50c-x, coins25c, coins10c, coins5c) & \text{if } x = 50 \\ (coins1d, coins50c, coins25c-x, coins10c, coins5c) & \text{if } x = 25 \\ (coins1d, coins50c, coins25c, coins10c-x, coins5c) & \text{if } x = 10 \\ (coins1d, coins50c, coins25c, coins10c, coins5c-x) & \text{if } x = 5 \end{cases}$$

$d \oslash (coins1d, coins50c, coins25c, coins10c, coins5c) = (coins1d', coins50c', coins25c', coins10c', coins5c'),$

where:

$coins1d' = min(coins1d, d \div 1)$

$coins50c' = min(coins50c, (d - coins1d') \div 0.50)$

$coins25c' = min(coins25c, (d - coins1d' - coins50c') \div 0.25)$

$coins10c' = min(coins10c, (d - coins1d' - coins50c' - coins25c') \div 0.10)$

$coins5c' = min(coins5c, (d - coins1d' - coins50c' - coins25c' - coins10c') \div 0.05)$

$\bar{0} = (0,0,0,0,0)$

Figure 1. DEVS Model of a soda can vending machine

SCC suggest. At the end of this section, we present, for some classes, one possible simulation. Again, because of the lack of space, when the partitions created by some criterion are too many we will just enumerate some of them. (Extended version of this paper `http://dcc.fceia.unr.edu.ar/~dhollmann/DEVS_Validation.pdf`)

We apply the following criteria as explained in each case:

Transition Function Defined by Cases (External) The first criterion that we apply to this example uses the definition of the external transition function, and generates one partition for each case in that definition. With these partitions we intend to simulate different scenarios to show how the model transitions, or how the model behaves in some particular circumstances, i.e. each case of the definition of the external transition function.

To describe the SCCs for this example, we will use several times a generic $s \in S$ defined as $s = (m, d, ot, np, dp, it, ms, om, mr))$.

The partitions generated are:

- $IniSt = \{s : s \in S\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x \in \{100, 50, 25, 10, 5\} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S \mid d \geq np\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getNormal} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S \mid d \geq dp\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getDiet} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{cancel} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{moneyRetreated} \wedge t \in \mathbb{R}+_0\}$

Standard Partitions We now apply the standard partitions criterion over the partitions generated before, i.e. the SCCs obtained from the external transition function definition. The criterion is applied over the operator $\geq$ that appears twice ($d \geq np$ and $d \geq dp$) and the standard partition for this operator is equal to the standard partition for the $<$ described in the previous section. We will apply again this criterion later. The followings new partitions are some of the result of applying this criterion. The first three correspond to the application of the criterion over the operation $d \geq np$ and the other three over the operation $d \geq dp$.

- $IniSt = \{s : s \in S \mid d < 0 \wedge np < 0\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getNormal} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S \mid d = 0 \wedge np < 0\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getNormal} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S \mid d > 0 \wedge np < 0\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getNormal} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S \mid d < 0 \wedge dp < 0\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getDiet} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S \mid d = 0 \wedge dp < 0\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getDiet} \wedge t \in \mathbb{R}+_0\}$
- $IniSt = \{s : s \in S \mid d > 0 \wedge dp < 0\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getDiet} \wedge t \in \mathbb{R}+_0\}$

Sets defined by extension: In this example we have two sets defined by extension, X and $MachState$. Since these two sets have a relative small number of elements, we should define one SCC for each element of them, as this criterion proposes (some classes)

- $IniSt = \{s : s \in S \mid m = \mathsf{idle}\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x \in X \wedge t \in \mathbb{R}_0^+\}$
- $IniSt = \{s : s \in S \mid m = \mathsf{operating}\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x \in X \wedge t \in \mathbb{R}_0^+\}$
- $IniSt = \{s : s \in S\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = 5 \wedge t \in \mathbb{R}_0^+\}$
- $IniSt = \{s : s \in S\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = 100 \wedge t \in \mathbb{R}_0^+\}$
- $IniSt = \{s : s \in S\}$,
  $InpSeq = \{\langle(x,t)\rangle \mid x = \mathsf{getNormal} \wedge t \in \mathbb{R}_0^+\}$

Time Partitions: Now we apply this particular criterion, bearing in mind the relation between the elapsed time, e, and the variables used for the time advance: operation time, ot, and the price increment time, it. The partitions are defined comparing each of those variable and altering its values and relation between them, i.e. $ot < it$, $it < ot$, etc.

- $IniSt = \{s : s \in S \mid ot = it = 0\}$,
  $InpSeq = \{\langle(x,0)\rangle : x \in X\}$
- $IniSt = \{s : s \in S \mid ot = it \wedge it > 0\}$,
  $InpSeq = \{\langle(x,0)\rangle : x \in X\}$
- $IniSt = \{s : s \in S \mid ot > it \wedge it = 0\}$,
  $InpSeq = \{\langle(x,0)\rangle : x \in X\}$
- $IniSt = \{s : s \in S \mid it > ot \wedge ot = 0\}$,
  $InpSeq = \{\langle(x,0)\rangle : x \in X\}$

Standard Partitions (over the remaining operators): There are five operators, $+$, $-$ and $\oplus$ in the external transition function and $\ominus$ and $\oslash$ in the internal transition function that still we did not analyze.

- $\oplus$, by definition, is based on $+$ and with the same type involved, $\mathbb{N}_0$, therefore they could have the same standard partition. However, since they involve only elements in $\mathbb{N}_0$ no further significant partitions can be proposed. Except if we want to simulate those cases where the implementation of those operations in the modeling language could rise some errors, e.g. overflow errors. In this case, the errors are not properly in the model but in its implementation. This is more related to a testing problem rather than validating through simulations.

- Where the operator "$-$" interacts with those variables used for representation of the time, the partitions for those cases are already described before (Time Partitions).

- Some of the partitions for the others occurrences of the operator "$-$" are:
  - $IniSt = \{s : s \in S \mid d = np = 0\}$,
    $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$
  - $IniSt = \{s : s \in S \mid d = np \wedge np > 0\}$,
    $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$
  - $IniSt = \{s : s \in S \mid 0 < d < np\}$,
    $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$
- With the operator $\oslash$ the Domain Propagation criterion could be applied since $\oslash$ is formed by two simpler operators, $-$ and $\div$. However, both operators have the same standard partition, therefore no new partition is generated combining the partitions generated of each sub-operator.

  Some partitions generated:

  - $IniSt = \{s : s \in S \mid d = 0\}$,
    $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$
  - $IniSt = \{s : s \in S \mid 0 < d < conis10d \wedge coins50c' < coins50c \wedge coins25c' < coins25c \wedge coins10c' < coins10c \wedge coins5c' < coins5c\}$,
    $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$
  - $IniSt = \{s : s \in S \mid d > 0 \wedge conins1d = 0\}$,
    $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$

Transition Function Defined by Cases (Internal): Finally, we apply once more the criterion Transition Function Defined by Cases, but this time we apply it to the internal transition function. This is a particular case since these transitions does not depend directly on the incoming events. In the previous criteria, the SCCs indicate how the system (model) must be (initial state) before running the simulation with the provided event sequence. Here, instead, it is necessary to drive the system to a particular state and wait for the internal transition occurs. Therefore, the resulting SCCs aim to this idea. These define also an initial state (or a set of possible initial states) and an events sequence (or a set of events sequences). However, the expected results of the simulation are not obtained immediately after the last input event is performed but after some time, when the desired internal transition occurs.

Some partitions:

- $IniSt = \{s : s \in S \mid m = \mathsf{operating} \wedge ot < it\}$,
  $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$
- $IniSt = \{s : s \in S \mid m = \mathsf{operating} \wedge ot = it\}$,
  $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$
- $IniSt = \{s : s \in S \mid m = \mathsf{idle} \wedge ot > it\}$,
  $InpSeq = \{\langle(x,t)\rangle : x \in X, t \in \mathbb{R}_0^+\}$

Combining Partitions: Once that we have applied all the partition criteria, we make the different conjunctions and we keep those where the result is non empty. Again, because of the lack of space, we do not show the partitions here, however these can be observed in the extended version of this paper.

Finally, we must choose one (or more) element(s) from each partition generated before, to simulate. S_{ini} represents the initial state of the model and *is* the input sequence to simulate.

Some possible simulations:

- $S_{ini} = (\mathsf{idle}, 0, 0, 0.75, 0.80, 1000, \bar{0}, \bar{0}, \bar{0})$
  $is = \langle(\mathsf{cancel}, 1)\rangle$
- $S_{ini} = (\mathsf{idle}, 0, 0, 0.75, 0.80, 1000, \bar{0}, \bar{0}, \bar{0})$
  $is = \langle(\mathsf{getNormal}, 1)\rangle$
- $S_{ini} = (\mathsf{idle}, 0, 0, 0.75, 0.80, 1000, \bar{0}, \bar{0}, \bar{0})$
  $is = \langle(\mathsf{moneyRetreated}, 1)\rangle$
- $S_{ini} = (\mathsf{operating}, 0, 0, 0.75, 0.80, 1000, \bar{0}, (1,0,0,0,0), \bar{0})$
  $sch = \langle(\mathsf{getNormal}, 1)\rangle$

6. DISCUSSIONS

In this section we briefly discuss the proposed criteria.

The *Transition functions Defined by Cases* generates partitions to cover the different cases in which the transition functions are defined. The intention is to validate the correctness and completeness of these functions.

The *Sets defined by extension* intends to generate partitions where each element of the sets (defined by extension) is simulated at least once, validating each possible input event or state value. It is interesting to combine the partitions obtained from this criterion with the ones obtained with the previous criterion, simulating then all input events in each case of the external transition function.

The *Standard Partitions* aims to validate the mathematical aspects of the model, i.e. the correct use of comparison, algebraical and other operators.

The *Domain Propagation* extends the previous criterion. It is useful when it is not simple to apply the standard partition criterion over a complex operator.

Finally, the *Time Partitions* is the most meaningful criterion when using timed formalisms, like DEVS. It aims to simulate different scenarios where events occur at different instants validating the time advance function and other temporal aspects such as the use of the elapsed time and the auxiliary variables representing time.

7. CONCLUSIONS AND FUTURE WORK

The main advantage of performing the simulations of a model with the technique presented in this work is that one does not need the experience of a specialist or group of specialists to validate the model. The validation depends on following a set of *rules*. This decreases the possibility to overlook some critical simulation avoiding the discover of some error in the model.

Another advantage of this work is the possibility to automatize (ideally the whole process) the validation of DEVS models by simulations implementing these techniques. An important issue to achieve this automation, besides the definition of the criteria, is to use a standard to describe DEVS models. There is a lot of work intending to do this, but this is still an open area. A standard language for DEVS would contribute to automatize our proposal because it would allow an automatic parsing of the model.

An overview of this automatized process is: parse the model specification (described in some standard language), apply the partition criteria to generate the simulations, translate the simulation configurations into some simulation language and simulate them. Observe the results and compare them with the specification.

Notice that we do not run the simulations, this is also part of the automation process. To do this, it is necessary to refine the proposed simulations. These are in an abstract language, therefore they must be rewritten in the simulation tool language, to run them. A possible way to do it, is adapting the work in model-based testing [3].

The final remark is the possibility of re-using these techniques to test software derived from a DEVS models. A DEVS model could be used as the specification of a system to be implemented in some programming language. The simulation sequences generated from the application of the partition criteria could be used as test cases to test the implementation.

We will focus our work on adding new criteria, improving the ones presented here and on the automation of the validating process by simulation.

REFERENCES

[1] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[2] L. Bougé, N. Choquet, L. Fribourg, and M. C. Gaudel. Test sets generation from algebraic specifications using logic programming. *Journal of Systems and Software*, 6:343–360, November 1986.

[3] M. Cristiá, D. A. Hollmann, P. Albertengo, C. S. Frydman, and P. R. Monetti. A Language for Test Case Refinement in the Test Template Framework. In *ICFEM*, pages 601–616, 2011.

[4] M. Cristiá and P. Monetti. Implementing and Applying the Stocks-Carrington Framework for Model-Based Testing. In K. Breitman and A. Cavalcanti, editors, *Formal Methods and Software Engineering*, volume 5885 of *Lecture Notes in Computer Science*, pages 167–185. Springer Berlin Heidelberg, 2009.

[5] P. S. da Silva and A. C. V. de Melo. On-the-fly verification of discrete event simulations by means of simulation purposes. In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pages 238–247, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[6] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan. Using formal specifications to support testing. *ACM Computing Surveys*, 41(2):1–76, 2009.

[7] Y. Labiche and G. Wainer. Towards the verification and validation of DEVS models. In *in Proceedings of 1st Open International Conference on Modeling & Simulation, 2005*, pages 295–305, 2005.

[8] X. Li, H. Vangheluwe, Y. Lei, H. Song, and W. Wang. A testing framework for DEVS formalism implementations. In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pages 183–188, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[9] M. Napoli and M. Parente. In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pages 59–66, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[10] S. R. S. Souza, J. C. Maldonado, S. C. P. Fabbri, and P. C. Masiero. Statecharts Specifications: A Family of Coverage Testing Criteria. In *XXVI Conferência Latinoamericana de Informática – CLEI'2000*, Tecnologico de Monterrey – México, September 2000.

[11] P. Stocks and D. Carrington. A Framework for Specification-Based Testing. *IEEE Trans. Softw. Eng.*, 22:777–793, November 1996.

[12] P. A. Stocks. Applying Formal Methods to Software Testing, 1993.

[13] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[14] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation, Second Edition*. Academic Press, London, 2000.