

Formalizing the Semantics of Modular DEVS Models with Temporal Logic*

Maximiliano Cristiá

CIFASIS – UNR
Flowgate Security Consulting
Rosario, Argentina
mcrastia@fceia.unr.edu.ar

RÉSUMÉ : *Control Theory researchers have been using DEVS models to formalize discrete event systems for a long time (Zeigler 1976) but, despite such systems are one of the main targets of Software Engineers, the DEVS formalism has not been used and it is hardly known by the formal methods community of Computer Science. This paper is a second attempt to close the gap between these communities by setting the rules to translate modular coupled DEVS models into TLA+ specifications. TLA+ (Lampert 2002) is a widely known formalism, used by formal methods researchers and practitioners, to specify –hardware or software based– reactive or concurrent systems. The paper includes the translation into TLA+ of three atomic DEVS models and a modular coupled model, which all together constitute a typical case study.*

MOTS-CLÉS : *DEVS, TLA+, temporal logic*

1 Introduction

Modeling critical, mission critical or embedded reactive systems is nowadays an accepted practice both in academia and industry (Craiggen, Gerhart & Ralston 1993, Bowen 1993, Hinchey & Bowen 1999, Clarke & Wing 1996, Ross 2005). It is perhaps the application domain that most rapidly has accepted that using formal techniques in earlier phases of the development process worth the (apparently) added costs. DEVS is a formal modeling technique and notation originally developed by Bernard P. Zeigler in the early '70 (Zeigler 1976). It has been routinely and successfully used, researched and expanded by researchers and practitioners belonging to the Control Theory or Automation communities. However, DEVS and its success has not been recognized as such by the Formal Method community of Computer Science; remarkably, one of the most complete and referenced on-line resources of this community (Bowen n.d.) does not even list DEVS as a formal notation or method. Since we are interested in the application of (Computer Science's) Formal Methods to software development, we try to close the gap between DEVS and other formal notations.

This paper extends (Cristiá 2007) by including the encoding of modular coupled DEVS models in Temporal Logic of Actions Plus (TLA+) specifications (Lampert 2002). TLA+ is a standard, widely known formal notation for Software Engineers envisioned by Leslie Lamport as a formal language (and latter a tool set) to be used by Software Engineers to model and verify complex, software or hardware based, reactive, real-time, concurrent systems. It is worth to say that a similar approach has been followed by people working in the Control Theory community (Dacharry & Giambiasi 2005).

We still restrict our present work to model discrete event systems and we left unattended its application to continuous systems (Kofman 2001, Giambiasi, Escude & Gosh 2000). Besides, this paper does not introduce DEVS nor TLA+ beyond the features that are part of the comparison we are reporting –readers may consult (Zeigler, Kim & Praehofer 2000) and (Lampert 2002) in order to get a deeper insight of both formalisms.

DEVS semantics belongs to the class named operative semantics because DEVS models are understood by interpreting them with a simulation algorithm – others formalisms with a similar scope describe their semantics in a similar fashion (Harel 1987). In other words, the meaning of a DEVS model is given by

*This paper was written while the author was spending an invitation period at LSIS - UMR CNRS 6168 – Université Paul Cézanne – France.

the rules to simulate it on a computer or manually. On the other hand, TLA+ semantics is of the class known as logic semantics since a TLA+ specification gets its meaning from a temporal logic model. Each approach has its own advantages and disadvantages (Cristiá 2007).

One of the problems in defining the semantics through a simulation algorithm is that the algorithm is informal until someone implements it in a particular programming language, it is compiled with a particular compiler and it is executed on a particular computer –these details become important since DEVS is used to simulate critical systems. Moreover, once the simulation algorithm is formalized by implementation, the implementation is not universal and cannot be replicated easily. Implementations may differ from one another making the same DEVS model to behave differently. And if the simulation algorithm is run by hand, its informality implies that two engineers may obtain different results when simulating the same model –besides the errors that can appear due to human errors.

In this paper we aim to formalize the semantics of modular coupled DEVS models by translating them into TLA+. In theory this formalization gives the chance to formally verify any implementation of a DEVS simulator and also it allows for formal verification of DEVS models.

The paper is structured as follows. Section 2 includes a few comments on (Cristiá 2007) regarding the rules to write a TLA+ specification from an atomic DEVS model. In section 3 the rules are extended to cope with modular coupled models and in section 4 we introduce a rather complete case study. Section 5 reports our conclusions and future work.

2 Encoding Atomic Models

This section was the main issue of (Cristiá 2007) but we want to introduce a minor adjustment that we needed to do in order to adapt the translation to modular coupled DEVS models. Besides we want to comment about the representation of outputs.

2.1 Distinguishing Internal and External Transitions

In (Cristiá 2007) we proposed two alternatives to distinguish internal and external transitions in the resulting TLA+ specifications. However, when we first tried to extend the translation to modular coupled

module <i>Example</i>
extends <i>RealTime</i>
VARIABLES <i>cpt, out, and other variables</i>
<i>NoVal</i> \triangleq CHOOSE $x \notin \{\text{“out1”}, \text{“out2”}\}$
<i>Init</i> \triangleq ...
<i>Error</i> \triangleq ... \wedge <i>out'</i> = “out1” ...
<i>Now</i> \triangleq ... \wedge <i>out'</i> = <i>NoVal</i> ...
<i>NextI</i> \triangleq <i>Error</i> \vee <i>other internal transitions</i>
<i>NextE</i> \triangleq <i>Now</i> \vee <i>other external transitions</i>
<i>Next</i> \triangleq <i>NextI</i> \vee <i>NextE</i>
<i>Spec</i> \triangleq \wedge <i>Init</i>
\wedge \Box <i>NextI</i> _{<i>st</i>}
\wedge <i>RTnow</i> (<i>st</i>)
\wedge <i>RTbound</i> (<i>Error</i> , <i>st</i> , <i>ta</i> [<i>st</i>], <i>ta</i> [<i>st</i>])
\wedge <i>RTBound</i> for other internal transitions

Figure 1: An sketch of a TLA+ specification encoding an atomic DEVS model.

DEVS models we noted that one of those alternatives is the most convenient.

The best alternative is to separate the *Next* predicate into two predicates: *NextI*, containing only the internal transitions; and *NextE*, containing only the external ones. Let us see a small example. Say there is an internal transition called *Error* (remember that in (Cristiá 2007) we said that both internal and external transitions should be mapped onto TLA+ actions), and an external transition waiting for event *Now*; also assume there are two output values, “out1” and “out2”, where the first is outputted by *Error*. Then, an sketch of the resulting TLA+ specifications is shown in Figure 1.

The separation of predicate *Next* into *NextI*, grouping all the internal transitions, and *NextE*, grouping all the external transitions, will be very useful when extending the encoding to modular composition (see section 3).

2.2 Encoding Outputs

In a discrete event setting, an output produced by a DEVS model can be regarded as an event. For instance, an internal transition can output an element of a queue which is considered as an event by the environment. In fact, in modular composition, outputs from one atomic model became external events for another atomic model.

Representing outputs as events in a temporal logic formalism is not as easy as it might seem because events are codified as predicates and predicates can

be true for an infinite number of steps, what can be interpreted as an infinite number of events. For this reason we suggest to change the value of output variables in every transition (internal and external) even if it is not necessary to output something. Precisely, for those transitions where no output should be produced we introduced the *NoVal* value as in module *Example* in Figure 1. This special value means, actually, no value at all. It has to be interpreted as no event is produced and the last event is not produced any more.

To define *NoVal* we use the `CHOOSE` operator. The expression `CHOOSE $x : F$` equals an arbitrarily chosen value x that satisfies formula F (Lamport 2002). Therefore, if F is $x \notin A$ and A is the union of all the possible outputs of the model, then *NoVal* equals something different from the outputs of the model¹.

3 Rules to Translate Modular Coupled DEVS Models

There are several ways to compose DEVS models into one bigger model. In this article we tackle the composition known as modular coupling (Zeigler et al. 2000). The models composed by modular composition interact each other solely by their input and output values. The whole idea is to model small parts of the problem as independent atomic models and then to assembly all together to get a model of the entire system. One important property of modular coupling is that the resulting model is equivalent to an atomic DEVS model. Only a subset of the inputs and outputs received or produced by the atomic models are externally visible.

As with DEVS models, there are several possibilities of composing TLA+ specifications (Lamport 2002). However, all of them imply, in a way or another, expressing composition as conjunction. A TLA+ specification is, in essence, a logical formula describing all of the possible histories of the system (Lamport 2002). Writing a specification as the composition of two or more specifications means, then, to write a logical formula as the conjunction of the formulas describing each specification.

A modular coupled DEVS model is composed of seven elements:

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select \rangle \quad (1)$$

Then, in translating a modular coupled DEVS model

¹`CHOOSE` is not a nondeterministic operator. It is also know as Hilbert's ε operator (Lamport 2002).

into a TLA+ specification we need to give translation rules for six of its seven elements, since the set $\{M_d\}$ with $d \in D$ is the collection of atomic DEVS models of which the modular model is composed (translation of atomic models is treated in (Cristiá 2007)). The rules for the remaining six elements are as follows.

- X Is the set of input values received by N . Here applies the same considerations as with the same element in atomic models (see (Cristiá 2007)).
- Y Is the set of output values produced by N . Same considerations of previous item.
- D This is the set of names of the atomic models. Each name in D is translated into the name of an instance of the corresponding TLA+ module. For example say that $D = \{Belt, Arm\}$. Then the modular coupled model *Cell* is translated as follows:

```

module Cell
extends RealTime
VARIABLES all the variables of the atomic models
Belt  $\triangleq$  instance Belt
Arm  $\triangleq$  instance Arm
.....
    
```

The variables of the atomic modules must be declared in the coupled module and some renaming would be necessary to avoid name clashes.

- $\{I_d\}$ Each I_d is the set of influences on model d . This element is used only in order to define the translation functions. In our encoding the influences of a given atomic model have no explicit place; we use them in the same step along with the translation functions.

- $\{Z_{i,d}\}$ For each $d \in D \cup \{N\}$ and for each $i \in I_d$, $Z_{i,d}$ is the input/output translation function. If i is N then the function translates a global input into an input for an atomic model; if d is N then the function translates an internal output into a global output; in the remaining cases the function translate internal outputs into internal inputs.

In our encoding of atomic DEVS models, outputs are stored in state variables, and inputs and external transitions become actions. External actions are fired once they are enabled; and they are enabled when their preconditions are true. Then, for each pair (i, d) we will define an action, collectively called Z actions, of one of three different kinds:

1. Input-Input. It simply "calls" an external action of some atomic model; this corresponds to Z functions translating inputs into inputs. The general form is $AtomicModel!ExternalAction$.
2. Input-Output. The precondition is the output value that the corresponding Z function should translate plus the precondition of the external action of the influenced model, and its post-condition will be the post-condition of the external action of the influenced model. The general form is $outAM = some_value \wedge AtomicModel!ExternalAction$, where $outAM$ is an output variable of an atomic model.
3. Output-Output. The precondition is the output value that the corresponding Z function should translate and the post-condition is the output value of the modular model. The general form is $outAM = some_value \wedge out' = other_value$, where $outAM$ is an output variable of an atomic model and out is an output value of the modular model.

Find an example in Figure 10 of section 4.

Select Is a function defining the priorities of internal transitions of different atomic models. *Select* decides which internal transition must be executed if there is more than one enabled at the same time in different atomic models. In a pure TLA+ model this is not necessary a problem since TLA+ semantics rests on the interleaving model of concurrency (IMC) (Lamport 2002). According to the IMC, if at a particular step in some model's history there is more than one action enabled, then any history resulting from executing any of them is a possible history of the system. Hence, there is no special language construction to define priorities over actions. However, we can represent *Select* with a predicate stating that if two internal actions are enabled then the history of the system must verify the execution of one of them. For example, if A and B are instances of two different TLA+ modules encoding two atomic DEVS models, f is an internal action in A and g is an internal action in B , then the specification of their modular composition saying that f has priority over g is:

$$Spec \triangleq \begin{aligned} &\wedge \dots \\ &\wedge \Box Select(A!f, B!g, A!f) \\ &\wedge \dots \end{aligned}$$

where

$$Select(A!f, B!g, A!f) \triangleq \begin{aligned} &ENABLED(A!f) \\ &\wedge ENABLED(B!g) \\ &\Rightarrow A!f \end{aligned}$$

Although this seems a good solution we think it deserves more attention in the future.

3.1 The Semantics of a Coupled Modular DEVS Model

In the last section we showed how to translate each element of a coupled modular DEVS model. In this section we discuss how to give the semantics of the modular model as a whole. Our point is, as with atomics models, that a part of the semantics of modular DEVS models is informal and is not contained in the tuple of elements describing the model.

According to (Zeigler et al. 2000) the semantics of a modular coupled DEVS model is that each atomic model executes or behaves independently of each other until the moment when one component produces an output that influences another component. In this moment the output is translated into an input event, by applying the adequate Z function, which is consumed by the influenced model.

In terms of temporal logic, that semantics means that any history of the modular model should verify that:

1. The initial state verifies the initial state predicates of all the atomic models;
2. Any pair of consecutive states belongs to the transition relation defined by the disjunction of:
 - (a) The *internal* actions of each and any of the atomic models; or
 - (b) The Z actions.
3. All the components use the same time line.
4. Only Z actions of the first kind are externally visible, i.e. they are regarded as external actions.

We formalize this interpretation in Figure 2.

4 A Case Study

In this section we will first state some informal requirements about a typical industrial production cell,

module <i>ModularModel</i>
extends <i>RealTime, Reals</i>
VARIABLES <i>cpt, variables atomic models</i>
<i>vars</i> \triangleq <i><all the variables></i>
<i>AM</i> ₁ \triangleq instance <i>first atomic module; rename</i>
.....
<i>AM</i> _{<i>n</i>} \triangleq instance <i>last atomic module; rename</i>
<hr/> <i>Init</i> \triangleq <i>AM</i> ₁ ! <i>Init</i> \wedge ... \wedge <i>AM</i> _{<i>n</i>} ! <i>Init</i>
<i>Z</i> ^{<i>j</i>₁} _{<i>i</i>₁} \triangleq ...
.....
<i>Z</i> ^{<i>j</i>_{<i>k</i>}} _{<i>i</i>_{<i>m</i>}} \triangleq ...
<i>NextE</i> \triangleq <i>only Z functions of first kind</i>
<i>NextI</i> \triangleq \vee <i>AM</i> ₁ ! <i>NextI</i> \vee ... \vee <i>AM</i> _{<i>n</i>} ! <i>NextI</i>
\vee <i>Z functions of second and third kind</i>
<i>Next</i> \triangleq <i>NextI</i> \vee <i>NextE</i>
<i>Spec</i> \triangleq <i>Init</i> \wedge [\square <i>Next</i>] _{<i>vars</i>} \wedge <i>RTnow</i> (<i>vars</i>)

Figure 2: TLA+ specification for a modular model.

then we will show an atomic DEVS model for each component, then a modular DEVS model representing the production cell is introduced and finally its corresponding TLA+ specification is presented.

4.1 Requirements

The production cell is composed of a conveyor belt, a robot arm and a press. The conveyor belt has a sensor which senses the items put on the belt and signals each of them 3 seconds after they have been detected (this is a safety time for the items traveling on the belt). The arm can take an item off the belt, carry it to the press, release it on the press and return to the belt. The take action should fire only when there is an item in a position to be lifted. The item can be released one second after receiving a free event from the press. Moving the arm in either direction are internal transitions that take 5 seconds. The press takes 1 second in pressing an item, then it is free. Clearly, there is some lack of synchronization but it was deliberated to simplify the case study. This problem is based on an example of (Evans 1994).

4.2 The Atomic DEVS Models

Figures 3, 4 and 5 describes the atomic DEVS models for the belt, the press and the robot arm, respectively. The first two are quite similar since both of them have two phases, one input, one output, and one state with an infinite lifetime.

On the other hand, the model for the robot arm is more complicated although it is a simplification since we assume that the arm takes (or releases) and starts to move in either direction, in one step. In this model the set *Phases* is $\{Empty, Holding, Ready\} \times \{AtLeft, MLeft, MRight, AtRight\}$. The external transition function waits for two inputs (the order to take an item located near its initial position or to release it in its final position), while the internal transition function has three alternatives depending on the phase: two of them are symmetric, representing the infinite wait at the initial and final positions; and the third represents the arm releasing the item at its final position and beginning to move backwards.

$$\begin{aligned}
Belt &= \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle \\
X &= \{ItemIn\} \\
Y &= \{ItemOut\} \\
S &= \{ItemOn, Nothing\} \times \mathbb{R}_0^+ \\
\delta_{int}(s, \sigma) &= (Nothing, \infty), \quad s = ItemOn \\
\delta_{ext}((s, \sigma), e, ItemIn) &= \\
&\begin{cases} (ItemOn, 3), & s = Nothing \\ (s, \sigma - e), & s = ItemOn \end{cases} \\
\lambda(s, \sigma) &= ItemOut, \quad s = ItemOn \\
ta(s, \sigma) &= \sigma
\end{aligned}$$

Figure 3: DEVS model for the belt.

$$\begin{aligned}
Press &= \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle \\
X &= \{Item\} \\
Y &= \{Free\} \\
S &= \{Pressing, Empty\} \times \mathbb{R}_0^+ \\
\delta_{int}(s, \sigma) &= (Empty, \infty), \quad s = Pressing \\
\delta_{ext}((s, \sigma), e, Item) &= \\
&\begin{cases} (Pressing, 1), & s = Empty \\ (s, \sigma - e), & s = Pressing \end{cases} \\
\lambda(s, \sigma) &= Free, \quad s = Pressing \\
ta(s, \sigma) &= \sigma
\end{aligned}$$

Figure 4: DEVS model for the press.

4.3 A Modular DEVS Model for the Production Cell

Now we assembly the atomic models in a model representing the behavior of the entire production cell in Figure 6.

As the reader can see, the model is no more than the modular composition of the atomic models. Note that the system as a whole has just one input and one output and that the output is used also as an internal event to tell the arm that it can release the item.

$$\begin{aligned}
 Arm &= \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle \\
 X &= \{Take, Release\} \\
 Y &= \{PosLeft, PosRight\} \\
 S &= Phases \times \mathbb{R}_0^+ \\
 \delta_{int}(s, \sigma) &= \begin{cases} ((Holding, AtRight), \infty), & s = (Holding, MRight) \\ ((Empty, AtLeft), \infty), & s = (Empty, MLeft) \\ ((Empty, MLeft), 5), & s = (Ready, AtRight) \end{cases} \\
 \delta_{ext}((s, \sigma), e, Take) &= \begin{cases} ((Holding, MRight), 5), & s = (Empty, AtLeft) \\ (s, \sigma - e), & s \neq (Empty, AtLeft) \end{cases} \\
 \delta_{ext}((s, \sigma), e, Release) &= \begin{cases} ((Ready, AtRight), 1), & s = (Holding, AtRight) \\ (s, \sigma - e), & s \neq (Holding, AtRight) \end{cases} \\
 \lambda(s, \sigma) &= \begin{cases} Releasing, & s = (Ready, AtRight) \\ PosLeft, & s = (Empty, MLeft) \end{cases} \\
 ta(s, \sigma) &= \sigma
 \end{aligned}$$

Figure 5: DEVS model for the arm.

$$\begin{aligned}
 Cell &= \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select \rangle \\
 X &= \{ItemIn, Free\} \\
 Y &= \{PressFree\} \\
 D &= \{Press, Belt, Arm\} \\
 I_{Press} &= \{Arm\} \\
 I_{Belt} &= \{Cell\} \\
 I_{Arm} &= \{Belt, Press\} \\
 I_{Cell} &= \{Press\} \\
 Z_{Arm,Press}(Releasing) &= Item \\
 Z_{Cell,Belt}(ItemIn) &= ItemIn \\
 Z_{Belt,Arm}(ItemOut) &= Take \\
 Z_{Press,Arm}(Free) &= Release \\
 Z_{Press,Cell}(Free) &= PressFree
 \end{aligned}$$

Figure 6: DEVS model for the cell.

4.4 Translating the Modular Model

The translation of model *Cell* into a TLA+ specification implies the translation of the atomic models as well as the modular model itself. The translations of the atomic models follow the rules proposed in (Cristiá 2007) plus the adjustments introduced in section 2.

4.4.1 Translating the Atomic Models

The TLA+ specification for model *Belt* is in Figure 7. The phases has the same names as in the DEVS model, function *ta* is defined in the CONSTANTS and ASSUME sections, *NoVal* is defined as a value different from the outputs produced by the belt, and the external transition is represented as an action with the same name of the event waited for the transition.

For the action representing the internal transition we selected an appropriate name (*ItemOut*). Note also that in every action *cpt* is updated with the value of *now* in this moment. As we suggest in section 2, the actions representing internal transitions are grouped in predicated *NextI*, while the actions corresponding to external transitions are in *NextE* (even in this model where we have only one of each type). In summary, the module complies with the encoding we proposed for atomic models in (Cristiá 2007).

$$\begin{array}{l}
 \text{module Belt} \\
 \text{extends RealTime, Reals} \\
 \text{CONSTANTS } ta, S, \mathbb{R}_0^+ \\
 \text{ASSUME} \\
 \quad \wedge S = \{\text{"ItemOn"}, \text{"Nothing"}\} \\
 \quad \wedge ta \in [S \rightarrow \mathbb{R}_0^+] \\
 \quad \wedge \forall s \in S : \\
 \quad \quad ta[s] = \text{case } s = \text{"ItemOn"} \rightarrow 3 \\
 \quad \quad \quad \quad \quad \quad s = \text{"Nothing"} \rightarrow \infty \\
 \text{VARIABLES } cpt, belt, out \\
 \text{vars} \triangleq \langle cpt, belt, out \rangle \\
 \text{NoVal} \triangleq \text{CHOOSE } x \neq \text{"ItemOut"} \\
 \text{TypeInv} \triangleq \wedge belt \in S \\
 \quad \quad \quad \wedge out \in \{\text{"ItemOut"}, \text{NoVal}\} \\
 \quad \quad \quad \wedge cpt \in \mathbb{R}_0^+ \\
 \text{Init} \triangleq \wedge cpt = now \\
 \quad \quad \quad \wedge belt = \text{"Nothing"} \\
 \quad \quad \quad \wedge out = \text{NoVal} \\
 \text{ItemIn} \triangleq \wedge belt = \text{"Nothing"} \\
 \quad \quad \quad \wedge now - cpt \leq ta[pos] \\
 \quad \quad \quad \wedge belt' = \text{"ItemOn"} \\
 \quad \quad \quad \wedge out' = \text{NoVal} \\
 \quad \quad \quad \wedge cpt' = now \\
 \text{ItemOut} \triangleq \wedge belt = \text{"ItemOn"} \\
 \quad \quad \quad \wedge belt' = \text{"Nothing"} \\
 \quad \quad \quad \wedge out' = \text{"ItemOut"} \\
 \quad \quad \quad \wedge cpt' = now \\
 \text{NextE} \triangleq \text{ItemIn} \\
 \text{NextI} \triangleq \text{ItemOut} \\
 \text{Next} \triangleq \text{NextI} \vee \text{NextE} \\
 \text{Spec} \triangleq \wedge \text{Init} \wedge [\Box \text{Next}]_{vars} \\
 \quad \quad \quad \wedge RTnow(vars) \\
 \quad \quad \quad \wedge RTBound(\text{ItemOut}, \\
 \quad \quad \quad \quad \quad \quad vars, ta[belt], ta[belt])
 \end{array}$$

Figure 7: TLA+ specification for the belt.

The TLA+ module for the DEVS model *Press* is in Figure 8. Since the corresponding atomic DEVS model for the press is quite similar to that for the belt, the resulting TLA+ module is similar to the module for the belt. Then, similar comments apply in this case.

```

module Press
extends RealTime, Reals
CONSTANTS ta, S, R0+
ASSUME
  ∧ S = {"Pressing", "Empty"}
  ∧ ta ∈ [S → R0+]
  ∧ ∀ s ∈ S :
      ta[s] = case s = "Pressing" → 1
                  s = "Empty" → ∞
VARIABLES cpt, press, out
vars ≜ ⟨cpt, press, out⟩
NoVal ≜ CHOOSE x ≠ "Free"
TypeInv ≜ ∧ press ∈ S
              ∧ out ∈ {"Free", NoVal}
              ∧ cpt ∈ R0+
Init ≜ ∧ cpt = now
          ∧ press = "Empty"
          ∧ out = NoVal
Item ≜ ∧ press = "Empty"
          ∧ now − cpt ≤ ta[pos]
          ∧ press' = "Pressing"
          ∧ out' = NoVal
          ∧ cpt' = now
Rise ≜ ∧ press = "Pressing"
          ∧ press' = "Empty"
          ∧ out' = "Free"
          ∧ cpt' = now
NextE ≜ Item
NextI ≜ Rise
Next ≜ NextE ∨ NextI
Spec ≜ ∧ Init ∧ [□Next]vars
          ∧ RTnow(vars)
          ∧ RTBound(Rise, vars, ta[belt], ta[belt])
    
```

Figure 8: TLA+ specification for the press.

Finally, the TLA+ module of the arm is in Figure 9. This module is larger and more complex than those for the belt and the press since the corresponding atomic DEVS model is also larger and more complex. However, the structure of the module is similar to the other two since the rules for translating all of them are the same. Note that here predicates *NextI* and *NextE* are proper predicates since there are several transitions. Also, note that since there are three actions representing internal transitions, there are three predicates named *RTBound*.

As you can see through these examples, the rules defined in (Cristiá 2007) for translating atomic DEVS models cope with interesting and non trivial models.

4.4.2 Translating the Modular Model

Now that we have all the modules for the atomic models we can translate the coupled modular DEVS model *Cell* into a TLA+ specification (find it in Figure 10). Note that module *Cell* has the structure we have suggested in section 3. In particular, all the variables (some with a convenient renaming) have been declared; however, all the time-related variables, *cpt* and *now*, were not renamed meaning that the time line for all the atomic models is the same. The modules corresponding to atomic models are incorporated into the module through the clause **instance**, where some variables are renamed accordingly to the names used in the VARIABLES section. The *Z* functions were defined according to the *Z* functions of the DEVS model; note that Z_{Belt}^{Cell} is of the first kind, Z_{Cell}^{Press} is of the third kind, while the rest of the *Z* actions are of the second kind. As proposed, all the internal actions plus the *Z* actions of the second and third kinds are grouped in one predicate (*NextI*), while *Z* actions of the first kind are grouped in another predicate (*NextE*).

```

module Cell
extends RealTime, Reals
VARIABLES cpt, arm, pos, belt, press, oA, oB, oP, out
vars ≜ ⟨cpt, arm, pos, belt, press, oA, oB, oP, out⟩
B ≜ instance Belt with out ← oB
A ≜ instance Arm with out ← oA
P ≜ instance Press with out ← oP
NoVal ≜ CHOOSE x ≠ "PressFree"
Init ≜ B!Init ∧ A!Init ∧ P!Init ∧ out = NoVal
ZBeltCell ≜ ∧ B!ItemIn
              ∧ UNCHANGED vars \ {cpt, belt}
ZPressArm ≜ ∧ oA = "Releasing" ∧ P!Item
              ∧ UNCHANGED vars \ {cpt, press}
ZArmBelt ≜ ∧ oB = "ItemOut" ∧ A!Take
              ∧ UNCHANGED vars \ {cpt, arm, pos}
ZArmPress ≜ ∧ oP = "Free" ∧ A!Release
              ∧ UNCHANGED vars \ {cpt, arm, pos}
ZCellPress ≜ ∧ oP = "Free" ∧ out' = "PressFree"
              ∧ UNCHANGED vars \ {cpt, out}
NextE ≜ ZBeltCell
NextI ≜ ∨ B!NextI ∨ A!NextI ∨ P!NextI
              ∨ ZPressArm ∨ ZArmBelt ∨ ZArmPress ∨ ZCellPress
Next ≜ NextI ∨ NextE
Spec ≜ ∧ Init ∧ [□Next]vars
          ∧ RTnow(vars)
    
```

Figure 10: TLA+ module for the cell.

module <i>Arm</i>
<p>extends <i>RealTime, Reals</i></p> <p>CONSTANTS ta, S, \mathfrak{R}_0^+</p> <p>ASSUME</p> $\wedge S = \{\text{"Empty"}, \text{"Holding"}\} \times \{\text{"AtLeft"}, \text{"MLeft"}, \text{"MRight"}, \text{"AtRight"}\}$ $\wedge ta \in [S \rightarrow \mathfrak{R}_0^+]$ $\wedge \forall s \in S :$ $ta[s] = \text{case } s \in \{\text{"Holding"}, \text{"AtRight"}, \text{"Empty"}, \text{"AtLeft"}\} \rightarrow \infty$ $s \in \{\text{"Holding"}, \text{"MRight"}, \text{"Empty"}, \text{"MLeft"}\} \rightarrow 5$ $s = \text{"Ready"}, \text{"AtRight"} \rightarrow 1$
<p>VARIABLES cpt, pos, arm, out</p> <p>$vars \triangleq \langle cpt, pos, arm, out \rangle$</p> <p>$NoVal \triangleq \text{CHOOSE } x \notin \{\text{"PosLeft"}, \text{"PosRight"}\}$</p> <p>$TypeInv \triangleq \wedge pos \in \{\text{"AtLeft"}, \text{"MLeft"}, \text{"MRight"}, \text{"AtRight"}\} \wedge arm \in \{\text{"Empty"}, \text{"Holding"}\}$</p> $\wedge out \in \{\text{"PosLeft"}, \text{"PosRight"}, NoVal\} \wedge cpt \in \mathfrak{R}_0^+$
<p>$Init \triangleq cpt = now \wedge pos = \text{"AtLeft"} \wedge arm = \text{"Empty"} \wedge out = NoVal$</p> <p>$Take \triangleq \wedge pos = \text{"AtLeft"} \wedge arm = \text{"Empty"} \wedge now - cpt \leq ta[pos]$</p> $\wedge pos' = \text{"MRight"} \wedge arm' = \text{"Holding"} \wedge out' = NoVal \wedge cpt' = now$ <p>$Release \triangleq \wedge pos = \text{"AtRight"} \wedge arm = \text{"Holding"} \wedge now - cpt \leq ta[pos]$</p> $\wedge pos' = \text{"AtRight"} \wedge arm' = \text{"Ready"} \wedge out' = NoVal \wedge cpt' = now$ <p>$ReleaseInt \triangleq \wedge pos = \text{"AtRight"} \wedge arm = \text{"Ready"}$</p> $\wedge pos' = \text{"MLeft"} \wedge arm' = \text{"Empty"} \wedge out' = \text{"Releasing"} \wedge cpt' = now$ <p>$StopR \triangleq \wedge pos = \text{"MRight"} \wedge arm = \text{"Holding"}$</p> $\wedge pos' = \text{"AtRight"} \wedge out' = \text{"PosRight"} \wedge cpt' = now \wedge arm' = arm$ <p>$StopL \triangleq \wedge pos = \text{"MLeft"} \wedge arm = \text{"Empty"}$</p> $\wedge pos' = \text{"AtLeft"} \wedge out' = \text{"PosLeft"} \wedge cpt' = now \wedge arm' = arm$ <p>$NextI \triangleq ReleaseInt \vee StopR \vee StopL$</p> <p>$NextE \triangleq Take \vee Release$</p> <p>$Next \triangleq NextI \vee NextE$</p> <p>$Spec \triangleq \wedge Init \wedge [\Box Next]_{vars} \wedge RTnow(vars)$</p> $\wedge RTBound(StopR, vars, ta[(pos, arm)], ta[(pos, arm)])$ $\wedge RTBound(StopL, vars, ta[(pos, arm)], ta[(pos, arm)])$ $\wedge RTBound(ReleaseInt, vars, ta[(pos, arm)], ta[(pos, arm)])$

Figure 9: The TLA+ specification of DEVS model Arm.

5 Conclusions

The main conclusion of this work is that modular coupled DEVS models describing discrete event systems can be easily translated into TLA+ specifications. This translation is not only possible but beneficial for DEVS since it lays the basis for a formal semantics of this powerful modeling language. Having a TLA+ specification of a DEVS model enables for formal verification of the model or to model-check it with the tools already available for TLA+ specifications. In fact, our encoding has made explicit some important rules to simulate a modular coupled DEVS model. However, we need to further investigate the best way to translate function *Select* present in modular coupled models.

In the future we will try to formalize the rules given so far as the first step towards the construction of a translation tool. Also, we need to prove whether this formalization preserves the notion that a modular coupled DEVS model is equivalent to an atomic model, or not –and in the last case we should adjust the rules. Then, we need to investigate whether other forms of composition (by including ports, for instance) can be translated or not, and in this case under what conditions the encoding works. Then, continuous systems might be considered but we do not foresee to much future in this direction.

Acknowledgments

References

- Bowen, J. (1993). Formal methods in safety-critical standards, *Proc. Software Engineering Standards Symposium (SESS'93)*, IEEE Computer Society Press, Brighton, UK,, pp. 168–177.
- Bowen, J. (n.d.). Formal methods, <http://vl.fimnet.info/>.
- Clarke, E. & Wing, J. (1996). Formal methods: state of the art and future directions, *ACM Computing Surveys* **18**(4): 626–643.
- Craigen, D., Gerhart, S. & Ralston, T. (1993). An international survey of industrial applications of formal methods, *Technical Report NIST GCR 93/626-V1 & NIST GCR 93-626-V2*, National Institute of Standards and Technology.
- Cristiá, M. (2007). A TLA+ encoding of DEVS models, *International Modeling and Simulation Multiconference*, Buenos Aires (Argentina), pp. 17–22.
- Dacharry, H. P. & Giambiasi, N. (2005). From Timed Automata to DEVS models: Formal verification, *SMC 2005 Spring Simulation Multiconference*.
- Evans, A. S. (1994). Specifying and verifying concurrent systems using Z, in M. Naftalin, T. Denzler & M. Bertran (eds), *FME '94: Industrial Benefit of Formal Methods*, pp. 366–380.
- Giambiasi, N., Escude, B. & Gosh, S. (2000). GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems, *Transactions of SCS* **17**(3): 120–134.
- Harel, D. (1987). Statecharts: A visual formalism for complex systems, *Science of Computer Programming* **8**: 231–274.
- Hinchey, M. & Bowen, J. (1999). *Industrial-Strength Formal Methods in Practice*, Formal Approaches to Computing and Information Technology, Springer-Verlag.
- Jackson, M. (1995). *Software Requirements and Specifications*, ACM Press.
- Kofman, E. (2001). Quantized-state control. a method for discrete event control of continuous systems, *Latin American Applied Research*.
- Lamport, L. (2002). *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley Professional.
- Ross, P. E. (2005). The exterminators, *IEEE Spectrum* **42**(9): 30–35.
- Zave, P. & Jackson, M. (1997). Four dark corners of requirements engineering, *ACM Transactions on Software Engineering and Methodology* **6**(1).
- Zeigler, B. P. (1976). *Theory of Modelling and Simulation*, Robert F. Krieger Publishing.
- Zeigler, B. P., Kim, T. G. & Praehofer, H. (2000). *Theory of Modeling and Simulation*, Academic Press, Inc., Orlando, FL, USA.