Discrete Event Simulation in an Environment for Temporal Expert Systems

Ramón P. Otero, Alvaro Barreiro, Pedro Cabalar and David Lorenzo

Dept. Computación, Fac. Informatics, University of A Coruña,
E-15071, A Coruña, Spain
email: otero@udc.es

Abstract. The relationship between *Generalized Magnitudes* (a Temporal Reasoning Scheme) and Discrete Event Systems Specifications is examined, revisiting a previous work of translation between both formalisms. As a result of this study, the Generalized Magnitudes Scheme is extended by adding autonomous activation — the system determines the next instant in which it must be re-evaluated. This new feature allows using the Generalized Magnitudes for representing and simulating Discrete Events models directly, which contributes several interesting representational properties. Finally, the computing process of the next activation instant is described.

1 Introduction

Temporal Reasoning in Artificial Intelligence is first understood as the ability to reason with temporal expressions, making inferences with temporal relations. Following this point of view, there are well known temporal formalisms, almost all of them, presented as extended logics that incorporate temporal features. However, we are interested in other Temporal Reasoning schemes – needed for Expert Systems – that are focused on reasoning about the evolution of the domain, rather than on reasoning about time itself. The formalisms used under this focusing can be related to classical Systems Theory formalisms, like the DEVS formalism [Zeigler76, Zei84]. A previous work [Otero94] has shown the possibility to represent DEVS structures in the temporal formalism called *Generalized Magnitudes* (GMs) focused on the domain evolution.

In this paper, we revisit the translation made between the two formalisms and we present a new approach that resolves open questions and increments the representational power of the GMs Scheme, taking concepts from Systems Theory. This new approach implies incorporating autonomous activation by deducing the instant, implicitly represented in the knowledge expressions, in which the knowledge must be evaluated. Autonomous activation would correspond to the concept of internal transition in the DEVS formalism.

As a consequence of incorporating autonomous activation, the resulting extended GMs scheme can be used for representing directly Discrete Event Systems, without a previous step of modeling using DEVS formalism. We will comment the differences between modeling and simulation in the GMs formalism versus classical Systems Theory approaches [Pichler92].

We also describe how to compute the next activation instant (what we have called *next evaluation time point*) making an incremental study of the possible knowledge expressions. After this, several complex expressions are examined considering a more formal description of this computation.

Finally, we get the conclusions and introduce some open topics.

2 Previous work: the translation oriented approach

DEVS	**GMs Scheme**
State Variables Input Variable Output Variable	Generalized Magnitude
State Transition Functions Output Functions	Knowledge Expression
State Transition Output	Knowledge Expression Evaluation (application of knowledge)

Table 1. Proposed equivalence between basic concepts

The incorporation of autonomous activation to the GMs Scheme has been already treated in [Otero94]. That work was intended for representing Discrete Events models by following a two steps procedure: first, representing the model in DEVS, and then translating it to GMs. The translation consisted in defining GMs for each DEVS element (inputs, outputs, state variables, transition functions and time advance function). Equivalences between basic concepts from boths formalisms are shown in table 1.

The whole work rested on two important features of the GMs formalism: the special temporal identifier `now` and the property of *pertinence*.

In the GMs scheme, the `now` identifier can be referred in the *Knowledge Expression* (KE) used to determine the value of a GM. When a reference to `now` occurs, it is evaluated returning the current time value in the continuous base time. On the other hand, a GM is said to be *pertinent* when, in a particular moment, it offers outstanding knowledge, i.e. it is capable of providing the system with new facts. One kind of pertinence is due to input changes whereas another one is due to references to `now`. When translating DEVS into GMs, the former has been used to represent DEVS *external transitions*, whereas the last allows representing *internal transitions*.

Using these two features (pertinence and references to `now`), a GM called `tnext` was defined. Such GM, that was evaluated after a certain elapsed time, was used to implement an internal transition, which can be seen as an autonomous activation of the system. Of course, it became necessary providing that GM with a special behavior: its value pointed out directly the instant in which the system would be automatically re-evaluated.

As we can notice, the study of the DEVS translation has not provided a general solution from the GMs point of view, since it forces to a previous modeling step in DEVS and needs the unnatural introduction of a GM with a special behavior. This has led us to take into account an extension to the formalism that includes *per se* autonomous activation.

3 The embedded autonomous activation

As we have seen before, we are looking for a way of adding autonomous activation to the GMs Scheme but, at the same time, we try to get such feature from the basic properties of the formalism. This implies that we must get an implicit way of representing autonomous activations and deduce when they will take place.

As a starting point, we can suppose that the system is continuously evaluated at any moment of time. Of course, evaluation of an infinite set of time points would be impossible at all. However, given the kind of models we are dealing with, we know that only several discrete time points are actually outstanding for knowledge evaluation. Therefore, the knowledge representation structure must be analyzed for finding out those outstanding time points.

Once the system has made a knowledge evaluation, it must predict the next evaluation time point. Since external inputs will always cause a knowledge activation, we only need to determine if an activation due to time changing by itself may occur during the interval comprised between two external inputs. The only GM that could change its value during such interval is `now`. Therefore, the next evaluation time point will be the next outstanding value for `now`, that is, a moment that causes that a condition that refers to `now` changes from true to false or vice versa.

The computing process of the next evaluation time point will be described later. For the present, we can notice several interesting consequences from this approach.

First of all, the time advance function does not need to be explicitly represented. It can be moved and distributed into each transition function, that is, each knowledge expression for the GM. The system will deduce the evaluation time points from references to `now`. Notice that, in fact, the `tnext` solution was a particular case of such analysis, that limited the possible references to `now` to those obtained from the DEVS translation.

Another result is getting a more natural representation (from the GMs Scheme point of view), since we have not introduced any formal change, but a semantical one, that is, a change of the meaning of the knowledge expressions. This involves a great advantage for the Temporal Expert Systems developers that already know the GMs Scheme, and even for migrating existing Temporal Expert Systems to the extended GMs Scheme.

As a consequence of the two preceding results, Discrete Events models can be directly represented using the extended GMs Scheme. Thus, we can override the previous phase of DEVS modeling and the following translation that was necessary in the first approach. If we do so, the equivalences between basic concepts of both formalisms are not as clear as before. For instance, one important outcome is that a knowledge expression can be seen now, from the DEVS point of view, as a mixture of external and internal transition function. This means that transition functions can mix indifferently references to external and internal events and, furthermore, a same condition of the transition function will even be able to behave dynamicly as an external or an internal transition.

The following section shows an example of representation of a DEVS system translated to GMs Scheme, but using an implicit time advance function.

4 A DEVS example

Figure 1 shows an example from [Zei89] of a generic control system. A system moves through several checkstates (denoted as `Pi`) in concert with the input received from a sensor. There is a time window (`tmin`) for the reception of that input and a generic function (`expected`) that decides if an input is expected for the current checkstate. Transitions are denoted as:

```
hold in <NEXT_PHASE_VALUE> <NEXT_SIGMA_VALUE>
```

where `sigma` is a state variable that determines directly the time advance function.

In figure 2 we have represented the translation of this DEVS example into the GMs formalism. Equivalences of table 1 have been applied , excepting for the time advance function. We have defined GMs `phase` and `chstate` that corresponds to the states variables of the DEVS specification, and GM `output` that corresponds to the output of the system.

The GM `tlast` is used for converting the absolute time reference represented by `now` into a relative reference to the last transition instant. For clarity reasons, we have also used a GM called `received` that will be true only when an input occurs.

The most important part of the example is the way in which we have represented the internal transitions. Notice that no `sigma` variable nor time advance function have been used. Instead of that, we have represented the temporal conditions – using references to `now` – that must hold for a given transition. Of course, the system will have to compute which is the next value of `now` that activates a transition (the next evaluation time point).

5 Computing the next evaluation time point

We have seen that, since the time advance function is given in an implicit way, the system must deduce which is the next evaluation time point. The general process is described by figure 3.

The graph shows the behavior of the system in absence of external inputs. An external input would assign directly a value for `now` (the instant in which the input occurred) and would jump to the knowledge application phase.

```
------- initial conditions -------

initial phase: WAIT
initial sigma: tmin(P1)
initial checkstate: P1

------- external transition -------

when receive value on sensor-port
        case of: phase
        WAIT    :       hold-in EARLY 0
        WINDOW  :       if value = expected (Pi)
        then hold-in SEND-COMMAND 0
        else hold-in ERROR 0

------- internal transition -------

case of: phase
        WAIT    :       hold-in WINDOW window (Pi)
        WINDOW  :       hold-in LATE 0
        SEND-COMMAND: set checkstate = next (checkstate)
                hold-in WAIT tmin (next(Pi))
        ERROR   :       passivate

------- output function -------

case of: phase
        EARLY   :       send "(Pi) input arrived too early" to error-port
        SEND-COMMAND:   send control command(Pi) to command port
        ERROR   :       send "(Pi) error in sensor value" to error-port
        LATE    :       send "(Pi) input arrived too late" to error-port
        else    :       send null message
```

Fig. 1. DEVS specifications of a generic control system

The interesting part of the process is the computation of the next evaluation time point for each GM. The minimum of those values will be the next evaluation time point for the whole system. When computing each time point, only the values for `now` that make true an expression will be interesting. The values for the rest of the GMs do not need to be computed, since they have just been evaluated in the knowledge application phase.

We will make an incremental study of knowledge expression cases. We will assume that `Ei` are the assignment expressions and that `Ci` and `Fi` are expressions that do not contain any reference to `now`.

```
GM phase (Initial Value: WAIT)
Knowledge Expression:
WINDOW      if previous(phase) = WAIT and timeof(now)=
                                 tmin(previous(chstate))+tlast;
LATE        if previous(phase) = WINDOW and timeof(now)=
                                 twindow(previous(chstate))+tlast;
WAIT        if previous(phase) = WINDOW and received and
                                 expected(value);
PASSIVE     if previous(phase) = WINDOW and received and
                                 unexpected(value);
EARLY       if previous(phase) = WAIT and received;

GM chstate (Initial Value: P1)
Knowledge Expression:
next(previous(chstate)) if phase = WAIT;

GM Output
Knowledge Expression:
"send command"   if phase = WAIT;
"error message"  if phase = PASSIVE;
"late message"   if phase = LATE;
"early message"  if phase = EARLY;

GM tlast (Initial Value: initial-absolute-time)
// time of the previous phase change
Knowledge Expression:
timeof(previous(phase));

GM received (Initial Value: false)
// indicates if there is input
Knowledge Expression:
true        if timeof(now)=timeof(value);
false;
```

Fig. 2. Translation into GMs formalism

5.1 No references to `now`

This would be the most trivial case. It would consist of a list of `if` clauses without any reference to `now`:

```
E1 if C1;
E2 if C2;
...
```

This kind of knowledge expression would not produce future activations caused by internal transitions. It would be equivalent to have only external

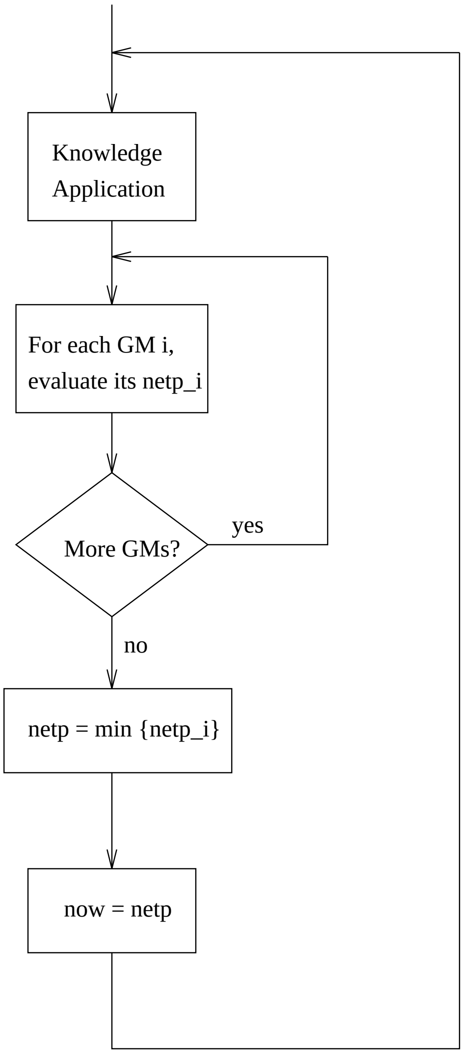


Fig. 3. General process

transitions.

5.2 Only references to now

In order to simplify the calculus of the next evaluation time point, we will assume that references to `now` are always cleared in relational expressions (they do not occur, for instance, at both sides of an equals sign). The shape of a knowledge expression for this case would be:

```
E1 if timeof(now)=F1;
```

```
E2 if timeof(now)=F2;
...
```

and the resulting time point would be the minimun of `Fi`.

5.3 Conditions with now and conditions without it

This case is a mixture of the two preceding ones. For instance, it could look like:

```
E1 if timeof(now)=F1;
E2 if C2;
E3 if timeof(now)=F3;
E4 if C4;
E5 if C5;
E6 if timeof(now)=F6;
E7 if timeof(now)=F7;
...
```

The next time point will depend on the truth values of conditions `Ci`. It will be the minimum of those `Fi` preceding the first condition `Cj` that is true. If all the `Ci` are false, it will be directly the minimum of all the `Fi`.

In case of simultaneous true conditions, there is no conflict, since the conditional clauses are executed as an *if-then-else* structure. Because of this same reason, when a `Cj` is true we can ignore all the conditions with references to `now` below.

5.4 A reference to now mixed with other expressions

```
E1 if timeof(now)=F1 and C1;
```

If we use the logical operator `and`, it will depend again on the truth value of the other conditions (`C1` in this case): if `C1` is false, it will not be any activation, whereas if it is true, the next time point will be `F1` (this would take the same behavior as case 5.2). The use of `or` operator would be analogous.

5.5 Use of order relation operators

```
E1 if timeof(now)>F1
E2 if timeof(now)<F2
```

Both cases would mean continuous evaluation, that is, we cannot specify a discrete instant in which the condition will change to true. This would make no sense in a Discrete Events Systems study and so, it would not yield any activation.

5.6 Mixing several references to now

Disjunctions of expressions with `now` would take as next evaluation time point the minimum of all the values, whereas conjunctions would only yield a value if all the referred values are the same. Mixing both operators would imply handling sets of values for evaluating a complex expression. The next activation would be taken at the minimum of the final set of values. Anyway, we will see in the following section a more general way of treating this kind of expressions.

6 A more formal treatment

The two last cases of the previous study have showed that complex expressions require a more formal analysis. For instance, the expression:

```
E1 if not (timeof(now)>F1) and not (timeof(now)<F2);
```

would not yield, at a first sight, any future activation. Nevertheless, when `F1` is equal to `F2`, the expression would be equivalent to:

```
E1 if timeof(now)=F1;
```

and should yield an activation at moment `F1`.

Type of Condition	Corresponding Interval
true condition	all possible values
false condition	$\emptyset$
timeof(now) = x	single point {x}
timeof(now) > x	(x,∞)
timeof(now) < x	(t_{app},x)

Table 2. Interval calculus for single expressions

Logical Operator	Corresponding Operation
and	intersection
or	union
not	complementary set

Table 3. Interval calculus for logical operators

This leads us to take into account all the values – expressed as intervals – that describe the truth temporal range of each condition. Table 2 describes how to obtain these intervals from single subexpressions, whereas table 3 shows how to combine them using logical operators.

Using both tables, we can always get a final set of values expressed as a union of disjoint intervals. Besides, since we are only interested in those future instants in which the condition is true, the final set of values must be truncated by the interval $(-\infty, t_{app})$, where t_{app} is the instant in which the knowledge is applied. The result will be what we can call *future truth range* of the condition.

The next evaluation time point of a GM must be determined from the future truth ranges of all the conditions contained in its knowledge expression. At a first sight, we could just take the minimum left end point of all those intervals. However, this is not always possible, since it may be the case in which the lower interval of all the future truth ranges is left open. In such case, we would not have a minimum value.

This left open interval problem can be seen as the formalization of the continuity problem that arose when studying expressions with order relation operators (case 5.5). When we face this exception, we can take two different decisions:(1) ignoring left open intervals; or (2) extending knowledge application for being executed in an interval instead of a discrete instant. However, only the first approach is applicable, since the second one would imply transitions that do not occur in an exact instant, which cannot happen when dealing with Discrete Event Systems.

Therefore, we can make the following:

Definition 1 *The* next evaluation time point *is the minimum left end point of all the left closed intervals contained in the future truth ranges.*

When there are not left closed intervals, there does not exist a next evaluation time point, and the GM does not cause a future activation.

Figure 4 shows two examples of computation of the next evaluation time point (`netp`).

In the first case, there is a knowledge application at instant $t = 2$ due to a change to value 3 of the GM C, and being the value of B different from 2. In such situation, GM A takes the value 8, and the future truth ranges of each condition are shown in the first column on the right of the knowledge expression. There is only one left closed interval, $[15, \infty)$, and its left end point is the next evaluation time point.

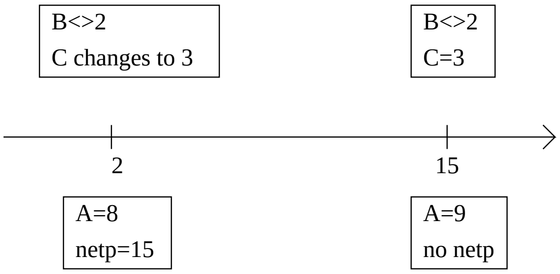
The following knowledge application takes place at moment $t = 15$ (we are assuming that no external input has occurred). At that situation, there is not a left closed interval and so, there is no future activation.

The second case is another example of absence of future activation. Notice that there is no transition of A to value 8, since we cannot determine the exact instant in which the change of value occurs.

Note also that, though conditions that yield left open intervals do not cause future activations, they can be normally used in a knowledge application caused by any other event. In that sense, we can distinguish two meanings of references

Example 1
GM A
Knowledge expression:

`7 if B=2;`	$\emptyset$	$\emptyset$
`8 if C=3 and timeof(now)<15;`	$(2, 15)$	$\emptyset$
`9;`	$[15, \infty)$	$(15, \infty)$



Example 2
GM A
Knowledge expression:

`7 if B=2;`	$\emptyset$
`8 if C=3 and timeof(now)>15;`	$(15, \infty)$
`9;`	$(2, 15]$

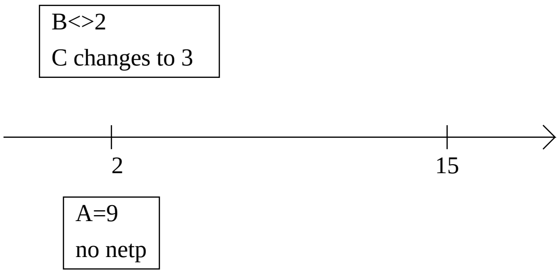


Fig. 4. Examples of computation of the next evaluation time point.

to `now`: as a temporal condition in a knowledge application (this was the traditional meaning) or as a future activation of knowledge. These two meanings of time also appear in the DEVS formalism. The first one corresponds to the elapsed time (when used in external transitions) and the second one corresponds to the time advance function.

7 Conclusions

The extension made to the GMs formalism involves a generalization of the former solution for the DEVS translation, with the advantage of preserving the basic properties of the scheme. Moreover, it is clear that autonomous activation can become a useful tool for implementing some typical temporal tasks that may appear when defining Temporal Expert Systems and where not treated before.

Another advantage is that a uniform specification of temporal expressions is achieved. The same representation is used for time in knowledge activations and for time as a temporal condition (like any other state variable).

Representing Discrete Event Systems directly with the GMs Scheme appears as an interesting possibility. From a Systems Theory point of view, the extended GMs formalism presents a different approach whose main features are: (1) only one transition function for internal and external events — even mixing both kind of events in the same transition function is possible; (2) the transition function is partitioned and attached to the associated state and output variable; and (3) the time advance function is integrated implicitly into the transition function expression.

Finally, the study has outlined several interesting open topics. Particularly, the possibility of treating knowledge applications in an interval of time, instead of an exact instant. This could lead to a different line of research that, although it does not fit exactly with the field of study of Discrete Events Systems, could cover more general models without implying necessarily continuous evaluation.

Another open topic would be the optimization of the computing process, trying to define those cases in which previous computations of the next evaluation time point can be reused for the whole system or a part of it.

Acknowledgements This investigation was supported in part by the Government of Galicia (Spain), grants XUGA10502A/92 and XUGA10503B/94.

References

[Otero94] R.P. Otero, A. Barreiro, H. Praehofer, F. Pichler, and J. Mira. Stimsmedtool: Integration of expert systems with systems modelling and simulation. *Lecture Notes in Computer Science*, (763):347–356, 1994.

[Pichler92] F. Pichler and H. Schwärtzel (Eds.). *CAST, Methods in Modelling*. Springer Verlag, 1992.

[Zeigler76] B. P. Zeigler. *Theory of Modelling and Simulation*. John Willey, 1976.

[Zei84] B. P. Zeigler. *Multifaceted Modelling and Discrete Event Simulation*. London: Academic Press, 1984.

[Zei89] B. P. Zeigler. Devs representation of dynamical systems: Event-based intelligent control. *Proceedings of the IEEE*, 77(1):72–80, 1989.