Introducing Planning in Discrete Event Systems *

Pedro Cabalar [1], Ramón P. Otero [2], Manuel Cabarcos [1] and Alvaro Barreiro [2]

[1] Dept. of Computer Languages and Systems. University of Vigo.
email: {cabalar,cabarcos}@uvigo.es
[2] Dept. of Computer Science. University of Corunna.
email: {otero,barreiro}@dc.fi.udc.es
http://www.dc.fi.udc.es/ai/

Abstract. Precedent studies have revealed a strong relationship between discrete events formalisms and formalisms for temporal reasoning in Artificial Intelligence. Both areas typically deal with temporal or dynamic domains, but under different points of view. Discrete events formalisms are intended for the analysis of the represented system, predicting its behavior by simulation. Artificial Intelligence approaches pay special attention to the adequacy of the representation to be used, which usually allows solving also planning problems. This paper is a study on the introduction of planning techniques into the DEVS model, used for modeling of discrete event systems. To this aim, we have chosen a basic representation, the *Generalized Magnitudes* temporal reasoning formalism, which let us represent a complete DEVS model of a system and simultaneously provides interesting properties for planning. The paper identifies the basic steps that a planning algorithm should carry out providing also some examples. Finally, it poses possible problems that may appear and establishes the future research lines.

1 Introduction

In previous works [5, 4] a strong relationship between discrete events formalisms and formalisms for temporal reasoning in Artificial Intelligence (AI) has been discussed. This has led to a growing interest in multidisciplinary research in both areas, especially if we bear in mind that they typically share the same application field: temporal or dynamic domains. However, the aim of each area is slightly different. While discrete events formalisms are intended for the analysis and design of the represented system, AI approaches emphasize the appropriateness of the representation to be used. Thus, discrete events formalisms are used in order to predict the behavior of the modeled system, making use of simulation, that is, solving questions like "what happens if ... ?". The evolution of the system is described as the interaction of its different components, but leaving open the choice for the internal representation of each component (it is simply not

* This research was supported in part by the Government of Galicia (Spain), grant XUGA10503B/96

relevant for the simulation process). On the AI part, the choice of the internal representation is an essential feature that allows solving not only simulation problems (called *temporal projection* in Reasoning About Change in AI) but also other kind of reasoning processes without counterpart in the simulation area. For instance, it is possible to solve problems like "what should happen in order to ...?" (known as *goal-driven* and related to planning techniques in AI). The area that deals with applying these reasoning capabilities to simulation has been called *Knowledge Based Simulation* [7].

In this paper, we develop a study on the introduction of Knowledge Based Simulation into the DEVS formalism [8], used for modeling and simulation of discrete events systems, but representing it on the *Generalized Magnitudes* temporal reasoning formalism. We take benefit from the previous work relating DEVS and GMs together with the interesting properties provided by the latter formalism for applying planning methods.

The paper is organized as follows. In section 2 we explain the need of selecting an adequate basic formalism and the advantages of using the GMs scheme. In the next section, we explain basic concepts about planning and we present a simple example with propositional variables. In section 4 we study how the basic planning procedure must be extended to take into account numerical temporal relations and internal transitions. Finally, section 5 concludes the paper.

2 The choice of a basic formalism

As said before, planning methods in Artificial Intelligence usually rely on formal treatment of the expressions that represent the modeled system. Since the DEVS formalism does not define a particular representation for the internal specification of each system component, we will have to fix such representation in order to design a planner for DEVS models. We have found that a very adequate choice to this end is the use of the *Generalized Magnitudes* (GMs) temporal reasoning formalism.

The strong relation between DEVS and GMs has been shown in previous studies. A first translation from DEVS models to GMs models was presented in [5] and extended in [4] by the addition of a new feature to GMs formalism (called *autonomous activations* and that allows representing internal transitions). Basically, the correspondence of representational entities between DEVS and GMs can be seen in figure 2. A system is represented by a set of elementary units, called *generalized magnitudes* (GMs), that would correspond to variables. Each GM may have an associated knowledge expression that provides the way of obtaining its value and that may use the values of other GMs. Knowledge expressions constitute the way in which GMs can be related. They correspond to a partitioned transition function, directed towards each state variable.

The use of GMs formalism to represent DEVS models provides many interesting features. For instance, it can be used as the unique formalism to represent the whole DEVS model, including both the specification of the internal behavior

DEVS	GMs
state variable input variable output variable	generalized magnitude
output function transition function	knowledge expression
transition	knowledge application

Fig. 1. Correspondence between DEVS and GMs structures

of its components and the interaction among them. This feature will simplify the design of planning methods.

Other interesting features for DEVS modeling are the homogeneous representation of transitions (both internal and external are represented using knowledge expressions), or the implicit time advance function (the time point for the next internal transition is computed from the expressions relating temporal values). However, in order to introduce planning techniques, the most important characteristic is that the GMs representational model is an AI formalism, combining a clear semantics with a high practical orientation. Thus, while it can be strongly related to theoretical action-based formalisms for Reasoning about Actions and Change, it is also usable for real systems modeling. The relation to theoretical approaches is a promising feature since most planning systems rely on rule-based representations, using algorithmic techniques with weak theoretical foundations. A logical formalization for the GMs scheme is currently under study so that we can design, in a similar way as in approach in [3], a soundness and completeness proof for our planning procedure (at least in its propositional case).

3 Planning

A planning problem will be posed in terms of an initial and a goal situation. In order to solve the planning problem, we must provide a *plan* which specifies the sequence of inputs that, starting at the initial situation, causes the system to pass through several intermediate states until reaching the goal one. A single planning problem may be solved by different plans.

These concepts are formally rephrased by the following definitions:

Definition 1 (State) A *state* is defined as a tuple (t, F) where t is a temporal label and F is a set of atomic ground facts of the form $x : c$, where x is a GM (a variable) and c is a constant value. The set F must not contain more than one fact for each GM. We say that a state is *complete* when it contains a fact for all the defined GMs. Otherwise, we say that the state is *partial*. □

Note that each state has an associated time value t which reflects its location in a continuous time basis. A planning problem is then defined in the following way:

Definition 2 (Planning Problem Description) A *planning problem description* is a tuple of the form $\langle s, g, B \rangle$ where s and g are partial states that receive the name of *initial* and *goal* respectively, whereas B is a set called the *Knowledge Base* containing all the defined knowledge expressions. $\square$

In order to define a plan, we will need to describe the sequence of situations that the system must pass through, but also, the events that cause such sequence (external or internal). In the GMs formalism, a transition is caused either by an external introduction of new facts at a given instant or by an internal event.

Definition 3 (Plan) Let $\psi = \langle s, g, B \rangle$ a planning problem description. A *plan* Π for ψ is a linear graph where:

- The sequence of nodes $\{s_1, \cdots, s_n\}$ are states
- Let $s_i = (t_i, F_i)$ be any state in the plan, each arc (s_{i-1}, s_i) is associated to a partial state Δ_i called the set of *cause events*, such that $\Delta_i = (t_i, F_i')$ and $F_i' \subseteq F_i$.
- s is consistent with respect to s_1 and g is consistent with respect to s_n

$\square$

The partial set Δ_i describes a transition. Its time label t_i is the instant at which it has occurred (and so, the instant for the resulting state), and the facts it contains, F_i', are the events (therefore, they must be among the facts of the resulting state F_i [2]). An arc with an empty events set $(t_i, \emptyset)$ will correspond to an internal transition.

The definition also asserts that the first state in the plan must be consistent with respect to the initial state. In the same way, the goal must be consistent with respect to the final state of the plan. Remember that s and g were partial descriptions. The plan may require to make them more specific (that is, satisfying more conditions) or even describe them as complete states.

The planning process will rely on a goal-driven algorithm similar to those of STRIPS [2] or PRODIGY, although in this case we must deal with a dynamic domain and temporal sequences of situations. Facts contained in the goal state are seen as wanted effects. Thus, for each wanted effect, its associated knowledge expression is analysed backwards, obtaining the possible causes that can make such fact true. Then, these causes become new goals to be satisfied, repeating this process iteratively.

The goal-driven process can be carried out thanks to the properties of the chosen formalism. Since the expressions are directed to each state variable instead of being general constraints, we can use them also backwards, guessing the possible causes for an effect.

It must be noticed that, for satisfying a given goal, several independent causes can be found, giving rise to different solutions. Thus, we will typically obtain a

[2] In GMs formalism input variables are represented as GMs, like state and output variables are.

solutions-tree rather than a single solution. Besides, since the knowledge expression may represent a transition, new goals obtained backwards may be placed in a previous situation rather than in the current one.

For a better understanding of the general planning process we propose a simple example. Suppose we do not want that an animal walks free and that the possible ways of avoiding this are killing it or closing it in a trap. In order to kill it, we must shoot a gun that must be previously loaded. Otherwise, the shoot noise frightens the animal and this one escapes far away. In order to close the animal, we may push it into the trap, if the animal is not far away, or entice it, if there is food inside. This scenario would be represented by the following knowledge base B:

```
{free: false if not alive or at_trap}
{alive: false if shoot and previous(loaded)}
{far: true if shoot and alive}
{at_trap: true if (previous(food) and entice) or
                  (not previous(far) and push)}
{loaded: true if load}
```

As it can be seen, only propositional GMs have been used for clarity sake. In the same way, we will not use time labels either. In such a simplified environment, the only temporal relation is reduced to the `previous` operator, which makes reference to the value obtained in the precedent evaluation of the GM. This kind of examples would correspond to those typically studied in Reasoning about Actions and Change, in AI.

Assume now, that we have a planning problem description $\langle s, g, B \rangle$, where B is the one shown before and s and g are:

```
s = {far:true, free:true, food:true, loaded:true}
g = {free:false}
```

that is, we want the animal to stop being free, starting from a situation in which there is food inside the trap, the animal is far away, and the gun is loaded.

A sketch of the basic planning process for this example is represented by the tree in figure 2. The root node is the goal partial state {`free:false`}. In order to make it true, two different solutions are found, `alive:false` and `at-trap:true`, which are represented as two branches at the same level. Each solution is considered a new assumption and constitutes, in its turn, a new goal to be satisfied. When we add an assumption for satisfying a subgoal, a test for consistency must be done in order to avoid inconsistent states.

The tree is expanded until no new assumptions can be done. At that point, the reasoning for the current state has finished. The figure shows this by surrounding all facts for the same state inside a rectangle.

The planner examines now if the leaf solution includes a reference to the `previous` operator or not. If it is so, the process goes on at a precedent state, causing a temporal jump backwards (represented as an arrow in the figure).

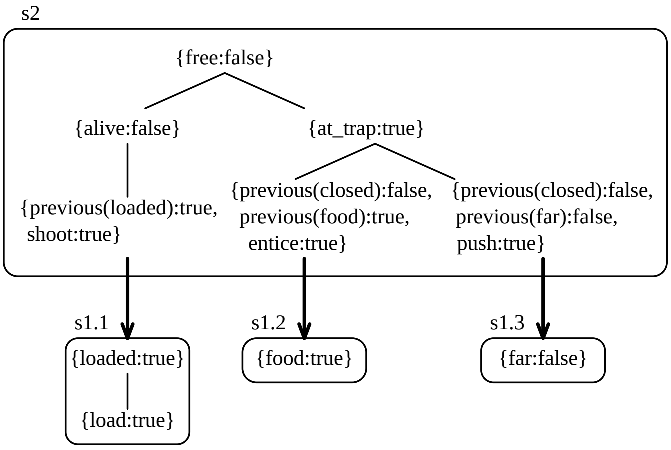

Fig. 2. Solutions tree for the "animal" example

The goal for the precedent state is constructed with all the facts affected by the `previous` operator in the current one.

When jumping to a precedent state, we could reach a situation previously considered in the same branch. This represents a transitions cycle (a sequence that can be repeated as many times as desired). When a cycle is detected, the branch is pruned since none of the states involved in it has been consistent with respect to the initial situation and so, it will never be reached.

Some problems must be remarked. The first one is recognizing when a particular path in the solutions tree may finally constitute a valid plan. The sequence of states in the plan would be that of the path following the order from the leaf node to the root (the goal state). The most direct way of obtaining a plan is when the current goals are consistent with respect to the initial situation. For instance, in state `s1.1` it is not necessary to further explain the fact `loaded:true` since we can consider the whole situation as the initial one (we would not need to load the gun, as it is initially loaded). When a solution is consistent with respect to the initial state, the branch is not expanded any more and we have obtained a valid plan.

However, we might reach a leaf solution, not consistent with the initial state, and without containing references to the past. State `s1.3` is in such case (we require `far:false`, but the animal is initially far away). When this happens, we would try to construct a possible plan by guessing that the obtained leaf state is the immediately successor of the initial situation. In this case, the plan would not be valid since we do not have a way of making `far` false when it was previously true.

We have represented in figure 3 the two valid plans resulted from the tree in figure 2. We have represented together with each arc its associated cause events set Δ_i.

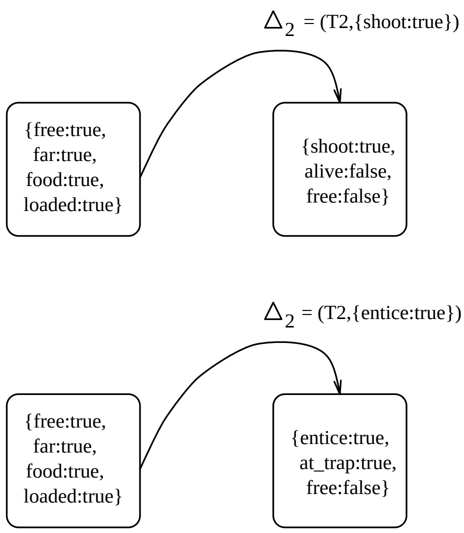

Fig. 3. Possible plans for the "animal" example

The planner has obtained two ways of avoiding the animal to be free: (1) shooting it directly (since the gun is already loaded); and (2) enticing the animal (as there is food inside).

Although the plan recognition may seem direct, there are, in fact, more factors that may complicate the process, even though working within the propositional frame. Thus, the representational flexibility of the GMs formalism also affects the planning process, offering more possibilities, but complicating the consistency test and the plan recognition. For instance, in the GMs formalism, input, output and state variables are represented by using the same unit: the GM. This even allows that the same GM behaves as an input, output or state variable at different moments. But simultaneously, a simple task like recognizing which are the inputs for each transition in the plan may be not so trivial. For example, the planner could also have proposed a transition directly assigning values to `loaded`, `alive`, `at-trap` or `far`, that intuitively correspond to state variables.

Another essential feature of the GMs formalism that affects the planning process is the *persistence* or *inertia principle*. This feature avoids an already well known problem that becomes apparent when treating with dynamic domains: the *frame problem*. When we use a simple transition based representation it is not enough with specifying the effects caused by a given event, but we must also specify explicitly those facts that must remain unchanged. This problem also manifests in Systems Theory: each transition description usually implies pointing out the result value for all the state variables (though many of them are not actually involved in the introduced change). The GMs formalism avoids this problem by applying the inertia principle, i.e., *the value of a GM persists* *unless there is some evidence on the contrary.* From the DEVS point of view, this would mean that the transition function only describes the effects caused by a given event, assuming by default that the rest remains unchanged. Thus, for example, we can specify that shooting a loaded gun causes the animal to be dead without needing to state that the value of `food` is not affected. The GMs reasoning process allows detecting which part of the knowledge base is relevant with respect to a given input. This is called *pertinence control.* We would say, for instance, that GM `alive` becomes *pertinent* when we shoot a loaded gun.

The inertia principle must also be present when searching for a valid plan. As an example, the plans in figure 3 only specify the pertinent facts for each resulting state in a transition. In this way, `food` remains true in all the states in both plans although this fact is only explicitly stated at the initial situation.

4 Planning for DEVS: a more complex example

The intention of the previous example was to show the usual planning methods in transition domains, and those derived problems not solved yet. However, our aim is to begin planning for DEVS, as far as we can, searching for a practical application and studying the needed restrictions for avoiding those problems.

In order to outline those tasks that should be added to the basic planning process, let us see a more complete example already presented in [8]. Suppose a system that moves through several check states (denoted as `P1`, `P2`, etc) in concert with the input received from a sensor. There is a time window (`twindow`) for the reception of that input and a generic function (`expected`) that decides if an input is expected for the current check state. The example is represented in figure 4. 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.

As already presented in [4], the translation to the GMs formalism can be seen in figure 5. Suppose now that we have a planning problem description where B is the knowledge base of the figure, and the initial and goal situations are:

```
s = (0,{phase:WAIT,chstate:"P1",tlast:0,received:false})
g = (Tg,{output:"error"})
```

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

initial phase: WAIT
initial sigma: tmin(P1)
initial chstate: 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 twindow(Pi)
        WINDOW  :       hold-in LATE 0
        SEND-COMMAND: set chstate = next (chstate)
                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. 4. DEVS specifications of a generic control system

that is, we want to reach a state in which the GM `output` gets the value `"error"` starting from a state placed at instant 0 with those facts specified in B as default values at beginning time. The solutions tree is represented in figure 6. Some things have changed with respect to the simpler example. First, we can notice how each situation has an associated time label variable (`T1`, `T2`, and `T3`). The introduction of time labels and numeric values also implies that, together with facts for GMs, we must also handle equality and comparative formulae like, for instance:

```
T3 < twindow(previous(chstate))+tlast
T3 = timeof(value)
```

```
{phase <WAIT,WINDOW,LATE,PASSIVE,EARLY> :
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
                                not expected(value);
EARLY       if previous(phase) = WAIT and received;
| WAIT at creation}

{chstate <text> :
next(previous(chstate)) if phase = WAIT;
| "P1" at creation}

{Output <text> :
"send command"  if phase = WAIT;
"error"   if phase = PASSIVE;
"late"    if phase = LATE;
"early"   if phase = EARLY;}

{tlast :
timeof(previous(phase));  // time of the previous phase change
| initial-absolute-time at creation}

{received <true,false>:
true        if timeof(now)=timeof(value);
false;
// indicates if there is input
| false at creation}
```

Fig. 5. Translation into GMs formalism

The tree has been constructed directly using expressions backwards. In order to satisfy the goal `output:"error"` we find the condition:

```
"error"   if phase = PASSIVE;
```

in its knowledge expression, and so, we add as new assumption the goal `phase:PASSIVE`. If we analyse now the knowledge expression for `phase`, we find that the way for making it `PASSIVE` would be:

```
PASSIVE     if previous(phase) = WINDOW and received and
                                not expected(value);
```

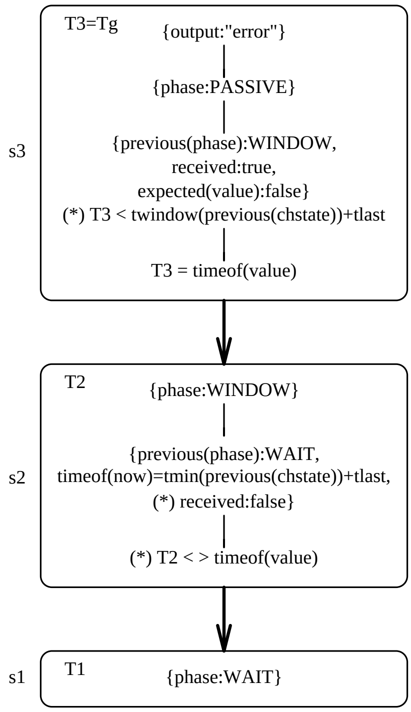

Fig. 6. Solutions tree for the control system example

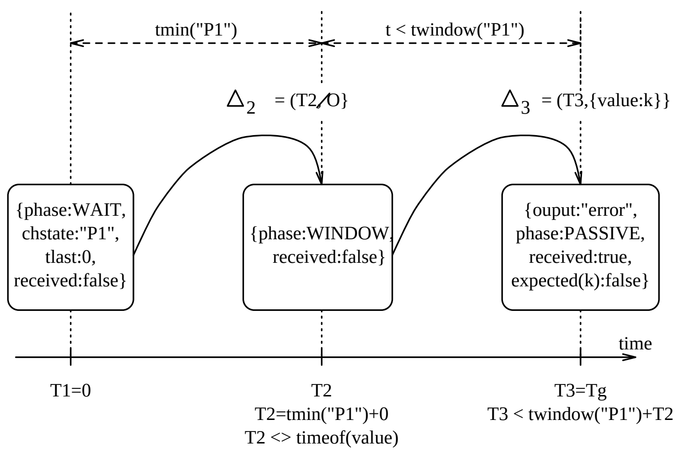

Fig. 7. Final plan for the control system example

adding, in its turn, these three conditions as new goals to be satisfied. Once this addition is done, a consistency checking is carried out. If we want that `phase` takes this value we must also guarantee that none of the conditions for values different from `PASSIVE` are true. Conditions for values `WINDOW` and `EARLY` will not hold since they require that `previous(phase)=WAIT` and we have fixed it to `WINDOW`. In a similar way, value `WAIT` is not possible since it requires `expected(value):true` and it is currently false. The only possible value would be:

```
LATE         if previous(phase) = WINDOW and timeof(now)=
                                   twindow(previous(chstate))+tlast;
```

This means that there exists a possibility of assigning simultaneously values `PASSIVE` and `LATE` to GM `phase`. To avoid this, we must guarantee that condition for `LATE` is false, adding the consistency constraint:

```
timeof(now) < > twindow(previous(chstate))+tlast;
```

We have marked this formula with a `(*)` pointing out that it has been added for consistency maintenance and it does not constitute a goal. It can also be noticed that we have replaced `timeof(now)` by `T3` which is the time of the current state, and that we have only taken into account the case in which it is lower than the obtained temporal expression. Actually, the case in which `T3 > twindow(previous(chstate))+tlast` would be considered another possible solution not represented here since it does not lead to a valid plan [3]. In a third step, goal `received` is satisfied by making `T3=timeof(value)`. At this point, we cannot add any new assumption and, since we have `previous(phase):WINDOW` we construct a precedent situation `s2` with goal `phase:WINDOW`. This process is repeated until we reach in `s1` a state consistent with respect to the initial one.

Deciding if the possible plan is finally valid is not so direct now. In order to make situation `s1` consistent with respect to `s` we must add the rest of facts in `s` and make `T1=0`. But these values will affect to the rest of conditions in the plan. For instance, all the references to `previous(chstate)` in the following states `s2` and `s3` would be replaced by `"P1"`, since this is the value for that GM at `s` and it remains unchanged. In the same way, we can replace `tlast` in the obtained temporal conditions by its value at each situation, i.e., the moment of the previous change of `phase`.

The final complete plan, with all the derived replacements, is represented in figure 7. The resulting plan passes through two transitions. The first one has an empty events set and so, it represents an internal transition. The system changes from the `WAIT` phase to the `WINDOW` phase when the minimal wait time for process `P1` has expired. The planner has also obtained a consistency condition that asserts that no external input for GM `value` can be done simultaneously to the internal transition.

[3] This condition would imply a previous internal transition to state `LATE` and thus, `previous(phase)` could not be `WINDOW`.

The second transition is caused by an external input Δ_3 that associates a value `k` to GM `value`. The value must not be `expected` in order to cause an error output. Besides, the instant in which the input is done, `T3` has to be chosen so that the elapsed time from the last transition does not exceed the `twindow` limit for `P1`. Otherwise, the system would automatically change to the `LATE` phase.

The example reveals that in order to extend the basic planner for dealing with typical DEVS systems it will be necessary to mix formal treatment of mathematical and logical expressions. This implies that solutions may usually be expressed as mathematical relations rather than concrete values (note how, in the example, we can choose any value for `T3` that holds the required conditions). The consistency checking will be the task most affected by the introduction of mathematical expressions. For instance, in the example, though the reason why we have rejected solution `T3 > twindow(previous(chstate))+tlast` in state `s3` can be seen at a first sight, the kind of formal transformations to be carried out for getting such conclusion could be rather complex.

5 Discussion

The current study has been useful as an introduction to the reasoning process necessary for the addition of planning capabilities to simulation. Planning is carried out by a goal-driven process that is possible thanks to the properties of the basic formalism we used: (1) directed expressions, (2) independent evolution of the system variables, and (3) inertia principle.

We have presented several problems to be treated in future work. The most critical one is the analysis of mathematical expressions, frequently used in DEVS systems. A possible solution could be the inclusion of mathematical treatment into a logical reasoning system (in a similar way to Equational Logic Programming models). However, these solutions handle a complex logical representation and usually imply a high computational overload. Together with these two drawbacks, if we bear in mind that in most cases we are not interested in reaching a set of definite numerical values as goal, but rather, that the goal consists of a numerical range of values, the Qualitative Reasoning approach [1] seems to be a more adequate focusing.

Another open line is the theoretical foundation for the planning algorithm. The logical formalization is not an easy task since it implies defining non classical logics that allow handling dynamic systems. Besides, as discussed in [6], the kind of reasoning carried out in a planning procedure does not fit exactly with a deductive process but rather with an abductive one (guessing what facts have to be added in order to obtain a conclusion). These topics are currently under study.

References

1. P. Dague. Qualitative reasoning: A survey of techniques and applications. *AI Communications*, 8(3/4):119–192, 1995.

2. R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Journal of Artificial Intelligence*, 2:189–208, 1971.
3. Joachim Hertzberg and Sylvie Thiebaux. Turning an action formalism into a planner. *Journal of Logic and Computation*, 1994.
4. R. P. Otero, A. Barreiro, P. Cabalar, and D. Lorenzo. Discrete event simulation in an environment for temporal expert systems. *Lecture Notes in Computer Science*, 1030:271–282, 1996.
5. R. P. Otero, A. Barreiro, H. Praehofer, F. Pichler, and J. Mira. Stims-medtool: Integration of expert systems with systems modelling and simulation. *Lecture Notes in Computer Science*, 763:347–356, 1994.
6. Murray P. Shanahan. Explanation in the situation calculus. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 160–165, 1993.
7. L. E. Widman, K. A. Loparo, and W. R. Nielsen (Eds.). *Artificial Intelligence, Simulation and Modeling.* John Wiley and Sons, New York, 1989.
8. B. P. Zeigler. DEVS representation of dynamical systems: Event-based intelligent control. *Proceedings of the IEEE*, 77(1):72–80, 1989.