Constraint-Based Modelling of Discrete Event Dynamic Systems

Gérard Verfaillie and **Cédric Pralet** and **Michel Lemaître**
ONERA, 2 av. Édouard Belin, BP 74025, F-31055 Toulouse Cédex 4, France
{Michel.Lemaitre,Cedric.Pralet,Gerard.Verfaillie}@onera.fr

Abstract

Numerous frameworks dedicated to the modelling of discrete event dynamic systems have been proposed to deal with programming, simulation, validation, situation tracking, or decision tasks (automata, Petri nets, Markov chains, temporal logic, situation calculus, STRIPS ...). All these frameworks present significant similarities, but none offers the flexibility of more generic frameworks such as logics or constraints.

In this article, we propose a generic framework for the modelling of discrete event dynamic systems, whose main components are state and event timelines and constraints on these timelines. Although any kind of constraint can be defined on timelines, we focus on some useful ones: pure temporal constraints, instantaneous state and event constraints, instantaneous and non instantaneous transition constraints.

Finally we show how the proposed framework subsumes existing apparently different frameworks such as automata, Petri nets, or classical frameworks used in planning and scheduling, while offering the great flexibility of a constraint-based modelling.

Introduction

The goal of this article is to propose a generic *constraint-based* framework for the modelling of *discrete event dynamic systems*, that is of systems whose state evolves over time via *instantaneous changes* possibly due to *instantaneous events*.

Numerous frameworks exist to model such systems. One can cite *automata*, *synchronous languages* (Benveniste *et al.* 2003) which allow automata to be compactly described, and *temporal logics* (Pnueli 1977) which allow properties of automata to be compactly described, but also *Petri nets*, *Markov chains* and *Markov Decision Processes* (Puterman 1994) which both allow stochastic changes to be described, the *STRIPS* framework (Fikes & Nilsson 1971) and the *situation calculus* (Levesque *et al.* 1997) both used in planning, and the usual models used in scheduling.

Although all these frameworks present significant similarities (discrete *instants* of transition, more or less compact representation of *states* and *transitions*), comparing them is somewhat difficult, unless translating all of them into the most basic ones and less compact ones: automata or Markov chains.

On the other hand, although *constraint-based modelling* is known to combine *compactness* and *flexibility* in terms of modelling with *efficiency* in terms of problem solving, it remains mainly used to deal with *static* problems, that is problems that do not involve *time*, despite some notable exceptions: mainly the *scheduling* problems (see (Baptiste, Pape, & Nuijten 2001)) and to a certain extent *planning* problems (see for example (Kautz & Selman 1992; van Beek & Chen 1999)). With only a few exceptions (see for example (Delzanno & Podelski 2001)), it is not used to deal with *validation* problems on dynamic systems or with *situation tracking* problems, such as *failure diagnosis*. We think that such a situation is mainly due to the absence of a generic constraint-based framework, dedicated to the modelling of discrete event dynamic systems and indifferently usable for simulation, validation, situation tracking, or decision tasks.

This is such a framework we propose in this article. It is based on the assumption of a *continuous time* and of *discrete instants* of event or change, and on the notion of *timelines*: *state timelines* to represent the way the state of the system evolves over time and *event timelines* to represent the way events occur. These timelines can be compactly represented via variables: *temporal variables* to represent the instants of change or event, and *atemporal variables* to represent values at these instants. Using the great flexibility of a constraint-based modelling, any kind of constraint can be defined on these timelines via constraints on temporal and atemporal variables. However, among them, *pure temporal*, *instantaneous state*, *instantaneous event*, *instantaneous transition*, and *non instantaneous transition* constraints are *a priori* very useful and would deserve to be particularly studied.

The framework proposed in this article is inspired but different from works carried out at the frontier between planning and scheduling problems, where the notion of *timeline* is used to represent the way state and resources evolve over time and to reason on time, state, and resources (Laborie & Ghallab 1995; Muscettola 1994; Ghallab 1996; Muscettola *et al.* 1998; Barták 1999; Frank & Jónsson 2003).

Section **Modelling Assumptions** introduces basic assumptions related to *time*, *states*, and *events*. Section **Timelines** introduces the *timeline-based* representation, whereas Section **Constraint networks on timelines** defines what is a *constraint network on timelines* (CNT) and Section **Use-**

ful types of constraint focuses on some *a priori* very useful types of constraint. In section **Subsumed frameworks**, we show how the proposed framework subsumes automata, Petri nets and classical frameworks used in planning and scheduling. Section **What remains to be done** concludes with the remaining work and some possible extensions.

This article focuses on *modelling* issues and says nothing about *algorithmic* issues (constraint propagation, search, ...), which will be the subject of future studies and articles. We do that because we think that the first obstacle, and perhaps the main one, to the systematic use of constraint-based modelling and reasoning in the context of discrete event dynamic systems is the modelling question.

Note that this work has nothing to do with the works on *dynamic CSPs* (Verfaillie & Jussien 2005). Dynamic CSPs aim at dealing with dynamic models, that is with changes which may occur in CSP models. In this work, we want to deal with static models of dynamic systems, that is with static models which include the system dynamics.

Modelling Assumptions

Time

We want to reason on instants, on the *order* between them, but also on their *values*. These values are assumed to belong to a *continuous* set. This is why we use $\mathbb{R}$, with the natural order over reals, to model time.

States and State Changes

States We assume that the state of a system can be modelled using a finite set of *state variables* representing the attributes of the state of this system. With each state variable, is associated a *domain* of values which can be finite or infinite, continuous or discrete, symbolic or numeric. In such conditions, the state of the system at any time is modelled by an assignment to each state variable of a value in its domain.

It must be noted that state variables can be used to represent *passive* attributes of the state (such as, for a robot, its position or its available level of energy), as well as *active* ones (such as, still for a robot, the mode of an observation instrument or the fact that the robot is currently moving in some way). In other words, state variables can be used to represent what we usually refer to as the *state* of the system (position, energy level, ...), as well as what we usually refer to as *actions*, when they are not instantaneous (an observation, a movement, ...).

State Changes We assume that the state of a system can change via *instantaneous changes* and only this way. Continuous changes cannot be hence precisely modelled and only approximated via a sequence of instantaneous changes. In such conditions, a change in the state of the system is modelled by an instantaneous simultaneous change in the assignment of a non empty subset of the state variables. Moreover, we adopt the convention that, when the assignment of a state variable v changes at time t from value val to value val', v is assigned value val before t, t excluded, and value val' after t, t included.[1]

State changes can occur at any time, but we assume that the instants at which they occur form a *discrete* subset of $\mathbb{R}$. Consequently, the assignment of a state variable remains constant from an instant t of change to the next instant $t' > t$ of change, that is equal to the value it took at t over the semi-closed interval $[t, t'[$ (see Figure 1).

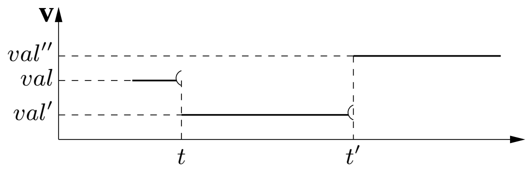

Figure 1: State variable over time

Events and Event Occurrences

Events The same way we assumed that the state of a system can be modelled using a finite set of state variables, we assume that the events that may occur can be modelled using a finite set of *event variables*. With each event variable, is associated a *domain* of values which can be finite or infinite, continuous or discrete, symbolic or numeric. At this domain, we systematically add a value *nothing* ($\perp$) to represent the absence of value. In such conditions, at any time, the set of events that are present is modelled by an assignment to each event variable of a value in its domain, possibly equal to $\perp$.

It is for example possible to associate an event variable with each type of event, with its value pointing out its content and the value $\perp$ pointing out the absence of event of this type.

Event Occurrences We assume that events are *instantaneous phenomena*. In such conditions, an event occurrence is modelled by an instantaneous simultaneous assignment of a value different from $\perp$ to a non empty subset of the event variables.

Events can occur at any time but, as with state changes, we assume that the instants at which they occur form a *discrete* subset of $\mathbb{R}$. Consequently, the assignment of an event variable remains equal to $\perp$ between two successive instants of event t and $t' > t$, t and t' excluded, that is on the open interval $]t, t'[$ (see Figure 2).

State Changes and Event Occurrences

No assumption is *a priori* made about any *correlation* or *causality* relation between state changes and event occurrences. State changes and event occurrences can be simultaneous. State changes can occur without any event and events without any state change.

[1]This convention, used for example in *synchronous languages*, is very useful to model instantaneous events which lead to instantaneous changes at the same time, for example a failure which leads instantaneously a system to a given failure mode.

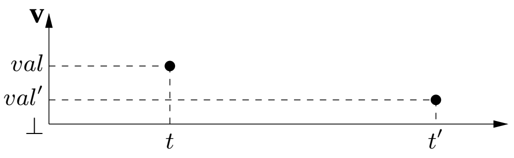

Figure 2: Event variable over time

Timelines

In this section, we show how *timelines* can be used to represent compactly the way state and event variables evolve over time.

Definition 1 *A timeline tl is defined as a quintuple $\langle v, d, I, t_I, v_I \rangle$ where v is a state or event variable, d its domain of values, I a sequence of instants, t_I a sequence of temporal variables, and v_I a sequence of atemporal variables.*

If v is a state variable, then we speak of a state timeline. Else, we speak of an event timeline.

Sequence I is assumed to be countable.[2] *Thus, I can be seen as a sequence of instant indices. Let be $I = [1, \ldots i, \ldots]$, $I^+ = [0, 1, \ldots i, \ldots]$, and $I^- = [2, \ldots i, \ldots]$. Sequence t_I associates with each instant $i \in I$ a temporal variable $t_i \in \mathbb{R}$ which represents the temporal position of instant i. Sequence v_I associates with each instant $i \in I^+$ an atemporal variable $v_i \in d$ which represents the value of v at instant i.*

Instants are temporally ordered. So, we enforce that $\forall i \in I^-$, $t_{i-1} \leq t_i$ and $(t_i = t_{i-1}) \rightarrow (v_i = v_{i-1})$. Moreover, in case of an event timeline, we enforce that $v_0 = \perp$.

A timeline $tl = \langle v, d, I, t_I, v_I \rangle$ represents the way v evolves over time. Sequence I is the sequence of the instants at which changes or events may occur (they may occur, but are not mandatory), t_I is the sequence of their temporal positions, and v_I the sequence of values of v at these instants. The first instant (0) in this sequence is fictitious and has no associated temporal position. It is used to represent the *initial value* of v, equal to $\perp$ in case of an event variable (no event at the initial instant). Figure 3 shows the tabular representation of such a timeline.

	0	1	$\ldots$	i	$\ldots$
t		t_1	$\ldots$	t_i	$\ldots$
v	v_0	v_1	$\ldots$	v_i	$\ldots$

Figure 3: Tabular representation of a timeline

It is important not to mistake the sequence I of *instants* for the sequence t_I of their *temporal positions*. From now, it is also important not to mistake *state* and *event* variables for *temporal* and *atemporal* variables that appear in timelines. Only the latter are actually *mathematical variables*. The former are *functions over time*. When confusion will be possible, we will keep the term *variable* for temporal and atemporal variables and use the term *timeline* for state and event variables. Moreover, when no confusion will be possible, we will speak indifferently of v and tl, making no distinction between a state or event variable and its associated timeline.

A timeline $tl = \langle v, d, I, t_I, v_I \rangle$ is said to be *finite* if I is finite. It is said to be *completely assigned* if all the temporal and atemporal variables in t_I and v_I are assigned. Let $tl = \langle v, d, I, t_I, v_I \rangle$ be a finite completely assigned timeline, with $I = [1, \ldots i, \ldots l]$. We refer to l as its *length* and to the closed interval $[t_1, t_l]$ as its *temporal horizon* H.

From the assumptions of Section **Modelling Assumptions** (a state variable remains constant and an event variable remains equal to $\perp$ between two successive instants of change or event), it is easy to derive from any finite completely assigned timeline $tl = \langle v, d, I, t_I, v_I \rangle$ the function which associates with any $t \in H$ (and not only with any $t \in t_I$) the value that v takes at t (v_i when $i = max\{i' \in I \mid t_{i'} \leq t\}$ for a state timeline; v_i if there exists $i \in I$ such that $t_i = t$ and $\perp$ otherwise for an event timeline) but also the value it takes just before t_1 (v_0 for a state timeline; $\perp$ for an event timeline) and the one it takes just after t_l (v_l for a state timeline; $\perp$ for an event timeline). Figure 4 shows a partial graphical representation of this function: *piecewise constant* function for a state timeline and *multi-dirac* function for an event one.

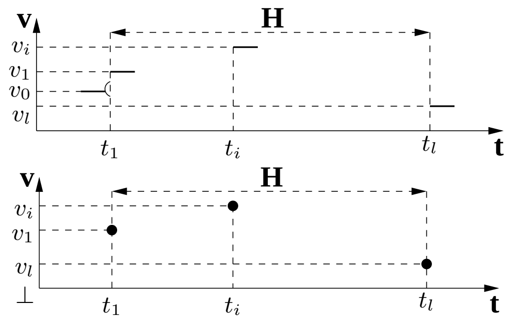

Figure 4: Functions over time, associated with a finite completely assigned timeline of length l, in case of a state timeline (above) or an event timeline (below)

Constraint networks on timelines

Constraint network definition

In this section, we show how *constraints* can be defined on timelines, in order to represent the combined evolutions of the state and event variables that are either *possible* or *required*.

Definition 2 *A constraint network CNT on timelines is a pair $\langle TL, C \rangle$ where:*

- *TL is a finite set of timelines which all share the same sequence I of instants and the same sequence t_I of their temporal positions;*

[2] A set is denumerable if and only if it is equipollent to the finite ordinals. It is countable if and only if it is either finite or denumerable.

- *C is a finite set of constraints on the timelines in TL (see definition 3).*

We note $V = \{v \mid \langle v, d, I, t_I, v_I \rangle \in TL\}$, $\forall i \in I^+$, $V_i = \{v_i \mid v \in V\}$, $V_I = [V_i \mid i \in I^+]$. SV, SV_i, and SV_I (respectively EV, EV_i, and EV_I) can be similarly defined by restricting ourselves to state timelines (respectively event timelines). Finally, we note $Var = t_I \cup V_I$

t_I is the set of temporal variables in the CNT, V_I the set of atemporal variables, and Var the whole set of variables, either temporal or atemporal. The same way as with timelines, we can define what a *finite* CNT and a *completely assigned* one are.

Definition 3 *A constraint c on a set TL of timelines is a quadruple $\langle qt, cd, sc, df \rangle$ where:*

- *qt is a finite sequence $[q_1, \ldots q_j, \ldots q_m]$ of quantifiers, with $q_j \in \{\forall, \exists\}$;*
- *cd is a finite sequence $[c_1, \ldots c_j, \ldots c_m]$ of conditions, each condition c_j being a boolean function over I^j;*[3]
- *sc is a function which associates with any sequence $[i_1, \ldots i_j, \ldots i_m] \in I^m$ satisfying the conditions in cd a basic constraint scope $sc(i_1, \ldots i_m)$, that is a finite sequence of variables in Var;*
- *df is a function which associates with any sequence $[i_1, \ldots i_j, \ldots i_m] \in I^m$ satisfying the conditions in cd a basic constraint definition $df(i_1, \ldots i_m)$, that is a boolean function over the Cartesian product of the domains of the variables in $sc(i_1, \ldots i_m)$.*

If $m = 0$, then $qt = \emptyset$ and $qt = \emptyset$, sc is a basic constraint scope, and df a basic constraint definition.

A *basic constraint* is a classical CSP constraint, defined as usual by its scope sc, which is a finite sequence of variables, and its definition df, which is a boolean function over the Cartesian product of the domains of the variables in sc (Rossi, Beek, & Walsh 2006). Quantification qt is used to specify in one non basic constraint a possibly infinite set of basic constraints by iterating on I which may be infinite. Condition cd is used to limit the elements of I on which to iterate. Functions sc and df are used to associate a basic constraint, that is a scope $sc(i_1, \ldots i_m)$ and a definition $df(i_1, \ldots i_m)$, with any sequence $[i_1, \ldots i_m] \in I^m$. Scopes can be specified by extension when I is finite or $m = 0$. They must be specified by intension otherwise. Definitions can be specified by extension when I is finite or $m = 0$ and when the domains of the involved variables are finite. They must be specified by intension otherwise.

To take a very simple example, let us consider a system which is represented by one state variable v whose value changes at each instant. We want to express that $\forall i \in I$, $v_i \neq v_{i-1}$. The associated CNT constraint is $c = \langle qt, cd, sc, df \rangle$ where $qt = [\forall]$ (sequence of quantifiers reduced to the only quantifier $\forall$), $cd = [true]$ (no condition on I), and $\forall i \in I$, $sc(i) = [v_i, v_{i-1}]$ (scope limited to variables v_i and v_{i-1}) and $df(i) \equiv (v_i \neq v_{i-1})$ (definition given by the $\neq$ relation between both variables).

[3] I^j is the Cartesian product of I, j times.

In spite of the presence of quantifiers, it is important not to mistake this framework with the *Quantified CSP* framework (Börner *et al.* 2003). Here, quantification is associated with *variable indices* and used to specify compactly possibly infinite sets of constraints, whereas quantification is associated with *variable values* in the QCSP framework.

Constraint satisfaction

Let us consider a finite CNT $\langle TL, C \rangle$ and a complete assignment A of it, that is of the set Var of involved variables. We can define recursively what is the *satisfaction* of a constraint $c \in C$ by A.

Definition 4 *A complete assignment A of a finite CNT $\langle TL, C \rangle$ satisfies a constraint $c \in C, c = \langle qt, sc, df \rangle$ if and only if it satisfies the quadruple $\langle \emptyset, qt, sc, df \rangle$. A complete assignment A of a finite CNT satisfies a quadruple $\langle Is, qt, sc, df \rangle$, where Is is a sequence of elements of I, if and only if:*

- *if $qt = \emptyset$: $(df(Is))(A_{\downarrow sc(Is)}) = true$*
- *if $qt = [q] \cup qt'$:*
  - *if $q = \forall$: $\forall i \in I$ such that $cd(Is \cup [i])$, A satisfies $\langle Is \cup [i], qt', sc, df \rangle$;*
  - *if $q = \exists$: $\exists i \in I$ such that $cd(Is \cup [i])$ and A satisfies $\langle Is \cup [i], qt', sc, df \rangle$.*

In the first case (empty sequence of quantifiers), the quadruple specifies a basic CSP constraint and constraint satisfaction is defined as usual in the CSP framework. The second case (non empty sequence of quantifiers), can be split into two sub-cases according to the first quantifier in the sequence: $\forall$ or $\exists$. Note that a universal quantifier leads to a *conjunction* of constraints, whereas an existential one leads to a *disjunction*.

We say that a complete assignment A of the variables Var of a finite CNT is *consistent* if and only if it satisfies all the constraints in C.

Complexity of constraint checking

If all the variables have finite domains of values of *maximal size* md, if all the basic constraints implicitly defined by the non basic ones are of *maximal arity* ma, if all the non basic constraints have sequences of quantifiers of *maximal size* ms, and if the CNT is of *maximal length* l, then the *time complexity* of checking the satisfaction of a constraint by a complete assignment is $O(l^{ms} \cdot c(ma, md))$, if we note $c(ma, md)$ the time complexity of checking the *satisfaction* of a basic constraint of maximal arity ma over domains of maximal size md. Without any surprise, this complexity grows exponentially with the maximal size ms of the sequences of quantifiers used in the constraint specifications.

Useful types of constraint

Section **Constraint network definition** introduced a very generic way of specifying constraints on timelines. But, it may be interesting to focus on some specific cases which may be *a priori* very useful when modelling and reasoning on discrete event dynamic systems. In this section, we consider *pure temporal*, *instantaneous state*, *instantaneous event*, *instantaneous transition*, and *non instantaneous transition* constraints.

Pure temporal constraints

Pure temporal constraints, which involve only *temporal variables*, are useful to constrain the temporal positions of the instants in the timelines.

A pure temporal constraint is defined as a constraint where scopes $sc(i_1, \dots i_m)$ are made only of variables in t_I: $\forall [i_1, \dots i_m] \in I^m$, $sc(i_1, \dots i_m) \subseteq t_I$.

A stronger interesting restriction would consist in limiting to 2 the arity of the basic constraints and in enforcing that their definitions be of the form $df(i_1, i_2) \equiv ((t_{i_1} - t_{i_2}) \in [lb, ub])$ in case of binary constraints and $df(i) \equiv (t_i \in [lb, ub])$ in case of unary constraints, resulting in only *simple temporal constraints* (Dechter, Meiry, & Pearl 1991).

Note the presence of implicit *simple temporal constraints* in each timeline, enforcing that $\forall i \in I^-, t_{i-1} \leq t_i$. These constraints can be modelled using one non basic constraint $c = \langle qt, cd, sc, df \rangle$, where $qt = [\forall]$, $cd = [i > 1]$, and $\forall i \in I^-$, $sc(i) = [t_{i-1}, t_i]$ and $df(i) \equiv (t_{i-1} \leq t_i)$.

Instantaneous state constraints

Instantaneous state constraints involve only *atemporal state variables* at the same instant, at which can be added the *temporal variable* at this instant. They are useful to express the combinations of values of the state variables at the same instant that are possible or required, possibly depending on the temporal position of this instant.

An instantaneous state constraint is defined as a constraint where the length of the sequence of quantifiers is limited to 1 and $\forall i \in I$, $sc(i) \subseteq SV_i \cup \{t_i\}$.

For example, let us assume a robot equipped with two observation instruments which cannot be simultaneously active. This requirement can be modelled using two state timelines $is1$ and $is2$, each one with a boolean domain, representing the activity status of each instrument, with $true$ associated with instrument activity, and one non basic constraint $c = \langle qt, cd, sc, df \rangle$, where $qt = [\forall]$, $cd = [true]$, and $\forall i \in I$, $sc(i) = [is1_i, is2_i]$ and $df(i) \equiv \neg(is1_i \wedge is2_i)$. This constraint specifies that $\forall i \in I$, $\neg(is1_i \wedge is2_i)$.

To take another example, let us assume that we want the robot to be at a given location lo_G by time t_G. This requirement can be modelled using one state timeline lo, representing the robot location, and one non basic instantaneous state constraint $c = \langle qt, cd, sc, df \rangle$, where $qt = [\exists]$, $cd = [true]$, and $\forall i \in I$, $sc(i) = [lo_i, t_i]$ and $df(i) \equiv ((lo_i = lo_G) \wedge (t_i \leq t_G))$. This constraint specifies that $\exists i \in I$ such that $((lo_i = lo_G) \wedge (t_i \leq t_G))$.

Instantaneous event constraints

Instantaneous event constraints involve only *atemporal event variables* at the same instant, at which can be added the *temporal variable* at this instant and *atemporal state variables* at the previous instant. They are useful to express the combinations of values of the event variables at the same instant that are possible or required, possibly depending on the temporal position of this instant and on the combinations of values of the state variables just before this instant, in order to model for example *action preconditions*.

An instantaneous event constraint is defined as a constraint where the length of the sequence of quantifiers is limited to 1 and $\forall i \in I$, $sc(i) \subseteq EV_i \cup \{t_i\} \cup SV_{i-1}$.

For example, let us assume a robot which has at its disposal a finite set A of actions, which cannot be simultaneously triggered. Moreover, let us assume that each action $a \in A$ requires a level $e(a)$ of energy to be triggered. This requirement can be modelled using one event timeline ca representing the triggered action, with a domain equal to $A \cup \{\perp\}$ ($\perp$ if no action is triggered), one state timeline ce representing the current level of energy, and one non basic instantaneous event constraint $c = \langle qt, cd, sc, df \rangle$, where $qt = [\forall]$, $cd = [true]$, and $\forall i \in I$, $sc(i) = [ca_i, ce_{i-1}]$ and $df(i) \equiv ((ca_i \neq \perp) \rightarrow (ce_{i-1} \geq e(ca_i)))$. This constraint specifies that $\forall i \in I$, $((ca_i \neq \perp) \rightarrow (ce_{i-1} \geq e(ca_i)))$.

Instantaneous transition constraints

Instantaneous transition constraints involve only *atemporal state or event variables* at the same instant, at which can be added the *temporal variable* at this instant and *atemporal state variables* at the previous instant. They are useful to express the combinations of values of the state and event variables at the same instant that are possible or required, possibly depending on the temporal position of this instant and on the combinations of values of the state variables just before this instant, in order to model for example *instantaneous action effects*.

An instantaneous transition constraint is defined as a constraint where the length of the sequence of quantifiers is limited to 1 and $\forall i \in I$, $sc(i) \subseteq V_i \cup \{t_i\} \cup SV_{i-1}$.

For example, let us consider an impulse switch whose position (open or close) can change in case of any impulse. However, let us assume that this switch may fail by remaining stuck at the position it had before failure. These facts can be modelled using three timelines, each one with a boolean domain, and one non basic instantaneous transition constraint. A first state timeline sp represents the current switch position (open or close), with $true$ associated with open. A second state timeline st represents the state of the switch (stuck or not), with $true$ associated with stuck. A third event timeline im represents the current impulse (present or not), with $true$ associated with present and $false$ with absent ($\perp = false$). The physical constraints are represented by one non basic instantaneous transition constraint $c = \langle qt, cd, sc, df \rangle$, where $qt = [\forall]$, $cd = [true]$, and $\forall i \in I$, $sc(i) = [sp_i, sp_{i-1}, st_i, im_i]$ and $df(i) \equiv ((sp_i \neq sp_{i-1}) \leftrightarrow (\neg st_i \wedge im_i))$, expressing that the switch position can change if and only the switch is not stuck and an impulse occurs. This constraint specifies that $\forall i \in I$, $((sp_i \neq sp_{i-1}) \leftrightarrow (\neg st_i \wedge im_i))$.

Non instantaneous transition constraints

Non instantaneous transition constraints are a bit more complex. They involve *atemporal state or event variables* between two instants i_1 and i_2, i_1 and i_2 included, at which can be added the *temporal variables* at instants i_1 and i_2, and *atemporal state variables* at instant $i_1 - 1$. They are useful to express the combinations of values of the state and event variables that are possible or required between two instants, possibly depending on the temporal position of both instants and on the combinations of values of the state variables just before the first instant, in order to model for example *non instantaneous action effects*.

A non instantaneous transition constraint is defined as a constraint where the length of the sequence of quantifiers is limited to 2 and $\forall [i_1, i_2] \in I^2$ such that $i_1 < i_2$, $sc(i_1, i_2) \subseteq \cup_{i_1 \leq i \leq i_2} V_i \cup \{t_{i_1}\} \cup \{t_{i_2}\} \cup SV_{i_1 - 1}$.

For example, let us assume a robot which has at its disposal a finite set A of actions, which cannot run simultaneously for any reason. Moreover, let us assume that each action $a \in A$ has a duration which is not precisely known, but belongs to an interval $[dmin(a), dmax(a)]$, and that an action a cannot be immediately followed by the same action a. These facts can be modelled using one state timeline ca representing the current action, with a domain equal to A, at which we can add a special value representing the absence of current action, and one non basic non instantaneous transition constraint $c = \langle qt, cd, sc, df \rangle$, where $qt = [\forall, \exists]$, $cd = [true, i_1 < i_2]$, and $\forall [i_1, i_2] \in I^2$, $sc(i_1, i_2) = [ca_i \, , (i_1 - 1) \leq i \leq i_2] \cup [t_{i_1}, t_{i_2}]$ and $df(i_1, i_2) \equiv (((ca_{i_1 - 1} \neq a) \wedge (ca_{i_1} = a)) \rightarrow ((\wedge_{i_1 < i < i_2} (ca_i = a)) \wedge (ca_{i_2} \neq a) \wedge (dmin(a) \leq (t_{i_2} - t_{i_1}) \leq dmax(a))))$. It is straightforward to show from Definition 4 that this constraint specifies that $\forall i_1 \in I$, $(((ca_{i_1 - 1} \neq a) \wedge (ca_{i_1} = a)) \rightarrow (\exists i_2 \in I$, $((i_1 < i_2) \wedge (\wedge_{i_1 < i < i_2} (ca_i = a)) \wedge (ca_{i_2} \neq a) \wedge (dmin(a) \leq (t_{i_2} - t_{i_1}) \leq dmax(a)))))$.

If actions in A produce some resource, such as on-board memory in case of data downloading, and if we assume that production is effective at the end of each action, each action $a \in A$ producing $r(a)$, this can be modelled by using a state timeline cr representing the current level of this resource and by adding $cr_{i_2 - 1}$ and cr_{i_2} in scopes $sc(i_1, i_2)$, and condition $(cr_{i_2} = cr_{i_2 - 1} + r(a))$ in the right side of definitions $df(i_1, i_2)$.

Subsumed frameworks

In this section, we show how the proposed framework subsumes existing ones such as *automata*, *Petri nets*, *STRIPS planning*, as well as classical models used in scheduling.

Automata

An *automaton* is usually defined as a quadruple $\langle S, E, T, s_0 \rangle$ where:

1. S is a finite set of *states*;
2. E is a finite set of *transition labels*;
3. $T \subseteq S \times E \times S$ is a set of *transitions*;
4. $s_0 \in S$ is the *initial state*.

An automaton specifies possible transitions: a transition $e \in E$ is possible from state $s \in S$ to state $s' \in S$ if and only if $\langle s, e, s' \rangle \in T$. It is easy to show that an automaton $\langle S, E, T, s_0 \rangle$ is equivalent to a CNT $\langle TL, C \rangle$ where:

1. TL is made of two timelines: one state timeline cs of domain S and one event timeline e of domain $E \cup \{\perp\}$;
2. C is made of two constraints:
   (a) an instantaneous state basic constraint $c_0 = \langle qt_0, cd_0, sc_0, df_0 \rangle$, with $qt_0 = \emptyset$, $cd_0 = \emptyset$, $sc_0 = \{cs_0\}$, and $df_0 \equiv (cs_0 = s_0)$, specifies the initial state;
   (b) an instantaneous transition constraint $c = \langle qt, cd, sc, df \rangle$, with $qt = [\forall]$, $cd = [true]$, and $\forall i \in I$, $sc(i) = [cs_{i-1}, e_i, cs_i]$ and $df(i) \equiv (\langle cs_{i-1}, e_i, cs_i \rangle \in T)$, specifies the following possible transitions.

Constraints on CNT appear as a compact way of specifying *synchronized products of automata* (Arnold & Nivat 1982).

Petri nets

A *Petri net* is usually defined as a quadruple $\langle P, T, Ip, Op \rangle$ where:

1. P is a finite set of *places*;
2. T is a finite set of *transitions*;
3. Ip is an *input function* from $P \times T$ to $\mathbb{N}$, which associates a positive integer (possibly null) with each place $p \in P$ and each transition $t \in T$;
4. Op is a similar *output function*.

A *marking* m (which can be considered as a state) is defined as a function from P to $\mathbb{N}$, which associates an integer $m(p)$ with each place $p \in P$. To be triggered from marking m, a transition $t \in T$ must satisfy the following condition: $\forall p \in P$, $m(p) \geq Ip(p, t)$. If a transition $t \in T$ is triggered from marking m, the result is a marking m' where $\forall p \in P$, $m'(p) = m(p) - Ip(p, t) + Op(p, t)$. As with automata, it is easy to show that a Petri net $\langle P, T, Ip, Op \rangle$ is equivalent to a CNT $\langle TL, C \rangle$ where:

1. a state timeline m_p of domain $\mathbb{N}$ is associated with each place $p \in P$, at which we add one event timeline e of domain $T \cup \{\perp\}$;[4]
2. C is made of two sets of constraints:
   (a) an instantaneous event constraint $c^E_{p,t} = \langle qt^E_{p,t}, cd^E_{p,t}, sc^E_{p,t}, df^E_{p,t} \rangle$ is associated with each place $p \in P$ and each transition $t \in T$, with $qt^E_{p,t} = [\forall]$, $cd^E_{p,t} = [true]$, and $\forall i \in I$, $sc^E_{p,t}(i) = [e_i, m_{p,i-1}]$ and $df^E_{p,t}(i) \equiv ((e_i = t) \rightarrow (m_{p,i-1} \geq Ip(p, t)))$, to specify transition preconditions;
   (b) an instantaneous transition constraint $c^T_{p,t} = \langle qt^T_{p,t}, cd^T_{p,t}, sc^T_{p,t}, df^T_{p,t} \rangle$ is associated with each place $p \in P$ and each transition $t \in T$, with $qt^T_{p,t} = [\forall]$, $cd^T_{p,t} = [true]$, and $\forall i \in I$, $sc^T_{p,t}(i) = [e_i, m_{p,i-1}, m_{p,i}]$ and $df^T_{p,t}(i) \equiv ((e_i = t) \rightarrow (m_{p,i} = m_{p,i-1} - Ip(p, t) + Op(p, t)))$, to specify transition effects.

[4]In a Petri net, two transitions cannot be triggered at the same time.

STRIPS planning

Planning problems may be of very different kind and, despite many efforts, there is no unique framework able to cover all of them (Ghallab, Nau, & Traverso 2004). This is why we restrict ourselves to the most classical one: the *STRIPS* framework (Fikes & Nilsson 1971) where a planning problem is defined as a quadruple $\langle F, A, Is, G\rangle$ where:

1. F is a finite set of boolean variables, called *fluents*;
2. A is a finite set of *actions*, where each action $a \in A$ is defined by a triple $\langle p_a, e^-_a, e^+_a\rangle$, with $p_a, e^-_a, e^+_a \subseteq F$ and $e^-_a \cap e^+_a = \emptyset$, where p_a, e^-_a, and e^+_a are action *preconditions*, *negative effects*, and *positive effects*;
3. $Is \subseteq F$ is a set of fluents, which defines the *initial state*;
4. G is a finite set of logical conditions on F, called *goals*.

A state s is defined as a function from F to $\mathbb{B}$, which associates a boolean value $s(f)$ with each fluent $f \in F$. The following conditions must be satisfied by states and transitions:

1. in the initial state s_0, $(s_0)(f)$ if and only if $f \in Is$;
2. an action $a \in A$ can be executed in a state s if and only if $\forall f \in p_a$, $s(f)$;
3. if an action $a \in A$ is executed in a state s, the result is a state s' where:
   (a) $\forall f \in e^-_a$, $\neg s'(f)$;
   (b) $\forall f \in e^+_a$, $s'(f)$;
   (c) $\forall f \in F - (e^-_a \cup e^+_a)$, $s'(f) = s(f)$.

The request usually associated with a planning problem is to produce a plan, that is a sequence of actions, whose execution allows the system to go from the initial state to a state that satisfies the goal conditions. It is easy to show that a planning problem $\langle F, A, Is, G\rangle$ is equivalent to a CNT $\langle TL, C\rangle$ where:

1. a state timeline s_f of boolean domain is associated with each fluent $f \in F$, at which we add one event timeline act of domain $A \cup \{\bot\}$;
2. C is made of five sets of constraints:
   (a) an instantaneous state basic constraint $c^{Is}_f = \langle qt^{Is}_f, cd^{Is}_f, sc^{Is}_f, df^{Is}_f\rangle$ is associated with each fluent $f \in F$, with $qt^{Is}_f = \emptyset$, $cd^{Is}_f = \emptyset$, $sc^{Is}_f = [s_{f,0}]$, and $df^{Is}_f \equiv (s_{f,0} \leftrightarrow f \in Is)$, to specify the initial state;
   (b) an instantaneous event constraint $c^P_{f,a} = \langle qt^P_{f,a}, cd^P_{f,a}, sc^P_{f,a}, df^P_{f,a}\rangle$ is associated with each action $a \in A$ and each fluent $f \in p_a$, with $qt^P_{f,a} = [\forall]$, $cd^P_{f,a} = [true]$, and $\forall i \in I$, $sc^P_{f,a}(i) = [act_i, s_{f,i-1}]$ and $df^P_{f,a}(i) \equiv ((act_i = a) \rightarrow s_{f,i-1})$, to specify action preconditions;
   (c) an instantaneous transition constraint $c^{E-}_{f,a} = \langle qt^{E-}_{f,a}, cd^{E-}_{f,a}, sc^{E-}_{f,a}, df^{E-}_{f,a}\rangle$ (resp. $c^{E+}_{f,a} = \langle qt^{E+}_{f,a}, cd^{E+}_{f,a}, sc^{E+}_{f,a}, df^{E+}_{f,a}\rangle$) is associated with each action $a \in A$ and each fluent $f \in e^-_a$ (resp. $f \in e^+_a$), with $qt^{E-}_{f,a} = qt^{E+}_{f,a} = [\forall]$, $cd^{E-}_{f,a} = cd^{E+}_{f,a} = [true]$, and $\forall i \in I$, $sc^{E-}_{f,a}(i) = sc^{E+}_{f,a}(i) = [act_i, s_{f,i}]$, and $df^{E-}_{f,a}(i) \equiv ((act_i = a) \rightarrow \neg s_{f,i})$ (resp. $df^{E+}_{f,a}(i) \equiv ((act_i = a) \rightarrow s_{f,i})$), to specify negative (resp. positive) effects;
   (d) an instantaneous transition constraint $c^N_{f,a} = \langle qt^N_{f,a}, cd^N_{f,a}, sc^N_{f,a}, df^N_{f,a}\rangle$ is associated with each action $a \in A$ and each fluent $f \in F - (e^-_a \cup e^+_a)$, with $qt^N_{f,a} = [\forall]$, $cd^N_{f,a} = [true]$, and $\forall i \in I$, $sc^N_{f,a}(i) = [act_i, s_{f,i-1}, s_{f,i}]$, and $df^N_{f,a}(i) \equiv ((act_i = a) \rightarrow (s_{f,i} = s_{f,i-1}))$, to specify null effects;
   (e) a unique instantaneous state constraint $c^G = \langle qt^G, cd^G, sc^G, df^G\rangle$, with $qt^G = [\exists]$, $cd^G = [true]$, and $\forall i \in I$, $sc^G(i) = (F(G))_i$ (if $F(G)$ is the set of fluents involved in G), and $df^G(i) \equiv \wedge_{g \in G} g_i$, to specify goal conditions.

Job-shop scheduling

In the scheduling domain, there is no reference framework similar to the *STRIPS* framework used in planning. There are only problems of very different kind. We focus here on a limited version of the so-called *job-shop scheduling*, one of the most classical scheduling problems, defined by a finite set T of tasks with, associated with each task $t \in T$, a duration du_t, an earliest start time es_t, and a latest end time le_t. One assumes that all the tasks must be performed and that they all require a common non sharable resource: two tasks cannot use this resource at the same time. Let $nt = |T|$ be the number of tasks. This problem can be modelled by a finite CNT $\langle TL, C\rangle$ of length $l = 2 \cdot nt$ where:

1. one state timeline ct of domain $T \cup \{0\}$ represents the currently active task, with 0 representing the absence of currently active task (there is no need for any event timeline);
2. a constraint $c_t = \langle qt_t, cd_t, sc_t, df_t\rangle$ is associated with each task $t \in T$, with $qt_t = [\exists]$, $cd_t = [true]$, $\forall i \in I$, $sc_t(i) = [ct_i, t_i, t_{i+1}]$, and $df_t(i) \equiv ((ct_i = t) \wedge (es_t \leq t_i \leq t_{i+1} \leq le_t) \wedge ((t_{i+1} - t_i) = du_t))$.

What remains to be done

The main result of this article is the proposal of a framework which, the first time as far as we know, allows discrete event dynamic systems (from automata and Petri nets to planning and scheduling) to be modelled in a uniform way using the basic notion of constraint.

The first task is to assess further the modelling power of the proposed framework, by addressing other frameworks such as *temporal logics* (Pnueli 1977), *situation calculus* (Levesque *et al.* 1997), or *timed automata* (Alur & Dill 1994), as well as various real-world problems.

When timelines are *finite* (and thus the set of involved variables), this framework allows *situation tracking*, *validation* or *decision* problems to be cast uniformly as CSP or QCSP and solved using any CSP or QCSP solver: CSP for situation tracking, validation, and optimistic decision problems, and QCSP for pessimistic decision ones.[5] In such a setting, it would be useful to develop or to adapt *constraint propagation* mechanisms associated with the most useful constraints we identified, which can be seen as *global constraints*. Beyond, it will be necessary to explore ways of answering requests on *infinite* timelines, as this is done with automata, Petri nets, and also planning problems.

About possible extensions, a first one would consist in going from *hard* constraints (which are used here to model possible/impossible facts as well as hard requirements) to *soft* ones, in order to represent and reason on *plausibility* and *utility* degrees, as done in (Pralet, Verfaillie, & Schiex 2006). This would allow us to capture for example *Markov Decision Problems* (Puterman 1994) and *probabilistic planning* (Kushmerick, Hanks, & Weld 1995). A second orthogonal extension would consist in relaxing the assumption that a state variable remains *constant* between two successive instants in a timeline and in considering *linear*, *monotonic*, or other evolutions (Trinquart & Ghallab 2001; Penberthy & Weld 1994). Finally, a third one could relax the assumption of a *total order* between instants in timelines.

Acknowledgements

This work has been done thanks to the CNES-ONERA-LAAS *AGATA* project (*Autonomy Generic Architecture : Tests and Applications*; see http://www.agata.fr), whose aim is to develop technical tools allowing space system autonomy to be improved. It took advantage of the feedback from many people involved in the project, we would like to thank.

References

[Alur & Dill 1994] Alur, R., and Dill, D. 1994. A Theory of Timed Automata. *Journal of Theoretical Computer Science* 126(2):183–235.

[Arnold & Nivat 1982] Arnold, A., and Nivat, M. 1982. Comportements de processus. In *Actes du Colloque AFCET "Les Mathématiques de l'Informatique"*, 35–68.

[Baptiste, Pape, & Nuijten 2001] Baptiste, P.; Pape, C. L.; and Nuijten, W. 2001. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers.

[Barták 1999] Barták, R. 1999. Dynamic Constraint Models for Complex Production Environments. In *Proc. of the Joint ERCIM/Compulog-Net Workshop*.

[Benveniste *et al.* 2003] Benveniste, A.; Caspi, P.; Edwards, S.; Halbwachs, N.; Guernic, P. L.; and de Simone, R. 2003. The Synchronous Languages Twelve Years Later. *Proc. of the IEEE* 91(1):64–83.

[Börner *et al.* 2003] Börner, F.; Bulatov, A.; Jeavons, P.; and Krokhin, A. 2003. Quantified Constraints: Agorithms and Complexity. In *Proc. of CSL-03*, 244–258.

[Dechter, Meiry, & Pearl 1991] Dechter, R.; Meiry, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

[5]When the domains of some temporal or atemporal variables are continuous, the resulting CSP/QCSP problems are hybrid discrete/continuous.

[Delzanno & Podelski 2001] Delzanno, G., and Podelski, A. 2001. Constraint-based Deductive Model Checking. *Software Tools for Technology Transfer* 3(3):250–270.

[Fikes & Nilsson 1971] Fikes, R., and Nilsson, N. 1971. STRIPS: a New Approach to the Application of Theorem Proving. *Artificial Intelligence* 2:189–208.

[Frank & Jónsson 2003] Frank, J., and Jónsson, A. 2003. Constraint-Based Attribute and Interval Planning. *Constraints* 8(4):339–364.

[Ghallab, Nau, & Traverso 2004] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

[Ghallab 1996] Ghallab, M. 1996. On Chronicles: Representation, On-line Recognition and Learning. In *Proc. of KR-96*, 597–606.

[Kautz & Selman 1992] Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *Proc. of ECAI-92*, 359–363.

[Kushmerick, Hanks, & Weld 1995] Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An Algorithm for Probabilistic Planning. *Artificial Intelligence* 76:239–286.

[Laborie & Ghallab 1995] Laborie, P., and Ghallab, M. 1995. IxTeT: an Integrated Approach for Plan Generation and Scheduling. In *Proc. of ETFA-95*, 485–495.

[Levesque *et al.* 1997] Levesque, H.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming* 31(1-3):59–83.

[Muscettola *et al.* 1998] Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* 103(1-2):5–48.

[Muscettola 1994] Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweden, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann. 169–212.

[Penberthy & Weld 1994] Penberthy, J., and Weld, D. 1994. Temporal Planning with Continuous Change. In *Proc. of AAAI-94*, 1010–1015.

[Pnueli 1977] Pnueli, A. 1977. The Temporal Logic of Programs. In *Proc. of FOCS-77*, 46–57.

[Pralet, Verfaillie, & Schiex 2006] Pralet, C.; Verfaillie, G.; and Schiex, T. 2006. Decision with Uncertainties, Feasibilities, and Utilities: Towards a Unified Algebraic Framework. In *Proc. of ECAI-06*, 427–431.

[Puterman 1994] Puterman, M. 1994. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

[Rossi, Beek, & Walsh 2006] Rossi, R.; Beek, P. V.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*. Elsevier.

[Trinquart & Ghallab 2001] Trinquart, R., and Ghallab, M. 2001. An Extended Functional Representation in Temporal Plannning: Towards Continuous Change. In *Proc. of ECP-01*.

[van Beek & Chen 1999] van Beek, P., and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. In *Proc. of AAAI-99*, 585–590.

[Verfaillie & Jussien 2005] Verfaillie, G., and Jussien, N. 2005. Constraint Solving in Uncertain and Dynamic Environments: A Survey. *Constraints* 10(3):253–281.