Flexible Interval Planning in Concurrent Temporal Golog

Alberto Finzi and **Fiora Pirri**[1]

Abstract. In this paper we present an approach to flexible planning and scheduling based on a suitable mapping of the Constraint Based Interval Planning paradigm [7, 2] into the Situation Calculus. We show how this representation is particular suitable for executive control processes, and illustrate this with an example.

1 Introduction

A central feature in executive control is flexible tasks alternation, yielding switching-time criteria, based on tasks, goal and the current situation needs. In *in vitro* domains[2], robots are requested to performing multiple tasks either simultaneously or in rapid alternation, using diverse sensing and actuating tools, like navigation, visual exploration, mapping, perceptual analysis, etc. To guarantee that such multiple-task performance can be achieved, some approaches have proposed that executive control processes supervise the selection, initiation, execution, and termination of actions. In this sense, a well known approach to executive control is Constraint Based Interval Planning (CBIP), amalgamating planning, scheduling and resources optimization for reasoning about the competing activities involved in a flexible concurrent plan (see [7, 2, 4]). The CBIP approach, like similar ones, emerged from the planning community, and have shown a strong practical impact in executive control processes [10, 18].

From the vantage point of cognitive robotics a question to be addressed is how executive control processes interact with basic perceptual-motor and cognitive processes used for performing individual tasks, how priorities among individual processes can be established, and how resources can be allocated to them during multiple-task performance (see [5, 8, 3]).

In particular, when dealing with the executive control it seems that approaches (such as the CBIP) tailored to the practical needs of reactive planning are more suitable. In this paper we show that CBIP perspective, on executive control processes, with all its arsenal of specifications in terms of flexible time, alternation constraints, resources optimization, failure recovering, and tasks scheduling, can be easily imported into the framework of the Situation Calculus([16, 9]), exploiting already established temporal and concurrent extensions of basic theory of actions, as those provided by [12, 14, 13, 17, 5]. The resulting language is one naturally belonging to the Concurrent Temporal Golog (CTG) families of languages, and it offers the possibility of manipulating flexible plans on a multiple time line. The paper is just introductory, and several problems still need to be addressed and solved; among these we mention the need of indexing situations so as to ensure multiple independent timelines, a suitable formalization of forgetting executed processes, progressing to the current set of processes, side effects of processes, and failure management. The mapping proposed is somehow obvious due to the expressive power of the Situation Calculus, but here can be meaningful, as it would offer to the temporal planning community a way to compare the Golog specification environment with the CBIP modeling constructs, and it provides a new account for deploying Golog at the executive control. In fact, we show how it can be used to implement a parallel control system for a particularly difficult task such as the *robocup rescue*.

[1] University of Rome “La Sapienza”

[2] Domains requiring effective robots performance, even if suitably constrained so as to keep possible domino effects under control

2 Preliminaries

2.1 Situation Calculus and Golog

The Situation Calculus (SC) [9] is a sorted first order language representing dynamic domains by means of *actions*, *situations*, and *fluents*. *Actions* and *situation* are first order terms. A situation denotes a history of actions compound with the binary symbol do: $do(a, s)$ is the situation obtained by executing the action a after the sequence s. The constant symbol S_0 stands for the initial situation (i.e. the empty sequence). In this work, we assume the Temporal Concurrent Situation Calculus presented in [12, 15, 14]. To represent time in the Situation Calculus, one of the arguments to the action function symbol is the time of the action's occurrence. For example, $startGoing(a, 12.01)$ is the action of starting to move toward a at time 12.01. All actions are viewed as instantaneous. The function symbol $time(a)$ denotes the occurrence time of action a, while $start(s)$ denotes the start time of situation s. The latter is defined as: $start(do(a, s)) = time(a)$, and $time(a)$ is defined, for each action term a, to be its temporal argument, for example, $time(startGo(a, t)) = t$. In this paper, we will use the notation $s[\vec{t}]$ to represent the situation s with $\vec{t}$ free temporal variables. Following [16], we represent concurrent actions as sets of primitive actions: the actions are sorted in simple action a and concurrent c. $a \in c$ means that the simple action a is one of the concurrent actions in c. We rely on the standard interpretation of sets and their operations and relations. Since in the concurrent SC, situations are lists of concurrent actions, we have situation terms like $do(\{a_1, a_2\}, s)$. A *fluent* is a predicate whose last argument is a situation, e.g. $at(hill, do(endGo(hill, 10), break, do(stratGo(hill, 2.1), S_0)))$. Fluents predicates denote properties that can change with the action execution.

In the SC concurrent durative actions are considered as *processes* [12, 16], represented by fluents, and durationless actions are to start and terminate the processes. For example, $going(hill, s)$ is started by the action $startGo(hill, t)$ and it is ended by $endGo(hill, t')$.

Domain Theory. In the SC a dynamic domain can be described by a *Basic Action Theory* (BAT) which is composed of the classes of axioms: $\Sigma \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap}$. Here Σ is the set of foundational axioms for situations. We refer to the version introduced in [14] in order to represent timed concurrent actions. For example, these axioms impose that, given a situation $do(c, s)$, all the actions $a \in c$ occur at the same time (i.e. $coherent(c)$).

$\mathcal{D}_{una}$ are uniqueness axioms for each action. $\mathcal{D}_{S_0}$ is a sets of sentences describing the initial state (i.e. S_0) of the domain.

$\mathcal{D}_{ssa}$ specify the *successor state axioms*, for each fluent $F(\vec{x}, s)$. Here we assume the successor state axioms modified in order to represent concurrent actions. This slight modification can be illustrated by the following example:

$$pointingTo(x, do(c, s)) \equiv startPointingTo(x) \in c \vee \\ pointing(x, s) \wedge endPointingTo(x) \notin c.$$

Finally, $\mathcal{D}_{ap}$ represents the action precondition axioms. In order to extend the $Poss$ predicate to concurrent actions, the following axioms are introduced:

$$Poss(a, s) \supset Poss(\{a\}, s).$$
$$Poss(c, s) \supset (\exists a) a \in c \wedge (\forall a)[a \in c \supset Poss(a, s)].$$

In a concurrent domain, *action preconditions axioms* on simple actions are not sufficient: two simple actions may each be possible, but their concurrent execution should not be permitted. This problem is called *precondition interaction problem* [16] (see [11] for a discussion) and its solution requires some additional precondition axioms.

Temporal Concurrent Golog. Golog is a situation calculus-based programming language for denoting complex actions composed of the primitive (simple or concurrent) actions defined in the BAT. Golog programs are defined by means of standard (and not so-standard) Algol-like control constructs: i. action sequence: $p_1; p_2$, ii. test: $\phi?$, iii. nondeterministic action choice $p_1 | p_2$, iv. conditionals, while loops, and procedure calls. An example of a Golog program is:

```
while $\neg at(hill, 3)$ do
    if $\neg(\exists x) going(x)$ do $\pi(t, (t < 3)? : startGo(hill, t))$
```

The semantics of a Golog program δ is a SC formula $Do(\delta, s, s')$ meaning that s' is a possible situation reached by δ once executed from s. Some of the construct definitions are the following

$$Do(nil, s, s).$$
$$Do((p_1 : p_2) : p_3, s, s') \doteq Do(p_1 : (p2 : p_3), s, s')$$
$$Do(a : p, s, s') \doteq Do(p, do(a, s), s')$$
$$Do(p_1 | p_2, s, s') \doteq Do(p_1, s, s') \vee Do(p_2, s, s')$$
$$Do(\pi(x, p(x)), s, s') \doteq (\exists x) Do(p(x), s, s')$$

In this paper we consider a Golog language version endowed with the parallel execution between processes. Analogously to [3] concurrency is modeled by interleaved processes and the parallel construct $\|$ is defined as follows:

1. $Do(p_1 \| p_2, s, s') \doteq Do(p_2 \| p_1, s, s')$.
2. $Do((p_1 : p_2) : p_3 \| p, s, s') \doteq Do(p_1 : (p2 : p_3) \| p, s, s')$.
3. $Do(p_1 \| nil, s, s') \doteq Do(p_1, s, s')$.
4. $Do(a_1 : p_1 \| a_2 : p_2, s, s') \doteq Do(a_1 : p_1 \| p_2, do(\{a_2\}, s), s') \vee$ $Do(p_1 \| a_2 : p_2, do(\{a_1\}, s), s') \vee Do(p_1 \| p_2, do(\{a_1, a_2\}, s), s')$
5. $Do((\phi)? : p_1 \| p, s, s') \doteq \phi[s] \wedge Do(p_1 \| p, s, s')$.
6. $Do((p_1 | p_2) \| p_3, s, s') \doteq Do(p_1 \| p_2, s, s') \vee Do(p_1 \| p_3, s, s')$.
7. $Do(\pi(x, p(x)) \| p, s, s') \doteq (\exists x) Do(p(x) \| p, s, s')$.

2.2 Constraint Based Interval Planning

In this section we briefly introduce the ontology and the basic concepts of Constraints Based Interval Planning paradigm, whose primitives are illustrated in Figure 1 (note that $[d, D]$ denotes the interval where d is the minimum time distance and D the maximum).

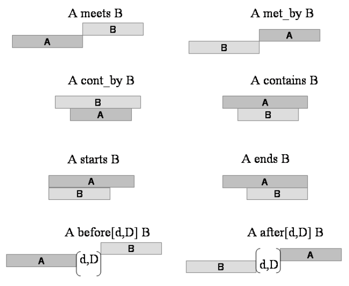

Figure 1. Interval relations in the underlying temporal model.

Attributes and Intervals The CBIP paradigm assumes a dynamic system modeled as a set of *attributes* whose state changes over time. Each attribute, called *state variable*, represents a concurrent thread, describing its history over time as a sequence of states and activities. Both states and activities are represented by temporal intervals called *tokens*. The history of states for a state variable over a period of time is called a *timeline*. For example, given a rover domain, $position$ is a possible attribute; $going(a, b)$ from time 1 to 3, and $at(b)$ from time 3 to 5 are intervals representing, respectively, an activity and a state. Each token can be described by the tuple $\langle v, p(\vec{x}), t_s, t_e \rangle$, where v is the attribute (e.g. $position$), p is the name of an activity, $\vec{x}$ are its parameters (e.g. $going(a, b)$), and t_s, t_e are numerical variables indicating start and end times respectively.

To represent intervals on a timeline we will use the notation $[t_1, t_2]\ p\ [t_3, t_4]$, meaning that p is s.t. $t_s \in [t_1, t_2]$ and $t_e \in [t_3, t_4]$ (e.g. given the timeline pos, $[0, 0]\ at(hill)\ [3, 4]$).

Domain Constraints. Given a set of attributes A and a set of intervals I, a *CBI model* $M = (A, I, R)$ is specified by a set of constraints R, that is, each token $T = \langle v, p(\vec{x}_p), t_s, t_e \rangle$ has its own *configuration constraint* $G_T(v, \vec{x}_p, t_s, t_e)$, called *compatibility* (see [7]), representing all the possible legal relations with other intervals; for example compatibility establishes which token must proceed, follow, be co-temporal, etc. to others in a legal plan. These relations are, in turn, defined by equality constraints between parameter variables of different tokens, and by simple temporal constraints on the start and end variables. The latter are specified in terms of metric version of temporal relations *a la Allen* [1]. Here we restrict our attention to the following set of temporal relations: $meets$, met_by, $contained\_by$, $contains$, $before[d, D]$, $after[d, D]$, $starts$, $ends$. For instance, $going(x, y)\ meets\ at(y)$, and $going(x, y)\ met\_by\ at(x)$ specifies that each $going$ interval is followed and preceded by a state at.

Planning Problem. Given the *CBI model* M specifying the planning domain, a *planning problem* is defined by $P = (M, P_c)$, where P_c is a *candidate plan*, representing an incomplete instance of a plan. The candidate plan consists of: i. a *planning horizon* specified by a pair of temporal values (h_s, h_e), with $h_s < h_e$; ii. a timeline $\mathcal{T}_\sigma = (T_{\sigma_1}, \ldots, T_{\sigma_n})$ for each state variable σ, containing a set of tokens $T_i = \langle \sigma, P(\vec{x}), t_{s_i}, t_{e_i} \rangle$; iii. a set of ordering constraints among the token in the timeline: $h_s \leq t_{e_1} \leq t_{s_2} \leq \ldots$; iv. the set of constraints $\{C_1, \ldots, C_n\}$ associated with the tokens laying on the timelines. For instance, given the rover domain with the attributes $Location$ and $Navigation$, a candidate plan can be represented by a planning horizon $(0, 10)$, and by the two timelines for the $Location$ (Lc) and $Navigation$ (Nv) attributes, and the two associated incomplete sequence of tokens (together with their respective constraints), e.g.

$$\begin{array}{ll} Lc: & [0,0]\ at(a)\ [3,3]\ \ [5,10]\ going(d,e)\ [6,10], \\ Nv: & [0,0]\ stop\ [3,5]\ \ [6,9]\ move\ [8,10], \end{array}$$

Notice that the *candidate plan* defines both the initial situation and the goals. A token T_i in a *candidate plan* is said to be *fully supported* if its G_{T_i} compatibility is satisfied, in a sense to be specified. For instance, $[5,10]$ $going(d,e)$, in the example above is not fully supported, since $going(d,e)$ met_by $at(d)$ is to be satisfied. A candidate plan is called a *potential behavior* if: a. each token on each timeline is fully supported; b. all timelines fully cover the planning horizon; c. all timeline tokens are bound to a single value. In other worlds, a *possible behavior* represents a possible evolution of the dynamic system. A candidate plan can be seen as an incomplete specification of a possible behavior where gaps, unsupported tokens, and uninstantiated variables can be found (see example above). A candidate plan is said to be a *complete plan* if it satisfies both the properties a. and b. specified above. More generally, a candidate plan is a plan depending on a *plan identification function* (see [7] for further details).

Given the *planning problem* specified by the *CBI model* and the *candidate plan*, the planning task is to provide a *complete plan* with the maximum flexibility: the planner should minimally ground the (temporal and not) variables to allow for on-line binding of the values. For example, given the rover example, assuming the candidate plan presented above (assuming also that each activity takes at least one time unit), a *sufficient plan* could be

$$\begin{array}{l} Lc: [0,0]at(a)[3,3]going(a,d)[4,8]at(d)[6,9]\ going(d,e)\ [6,10], \\ Nv: [0,0]\ stop\ [3,3]move[4,8]stop[6,9]move[8,10]. \end{array}$$

Notice that the above specification completely fill the timelines till the end of the horizon and each token is fully supported.

3 Representing the Temporal Model in the Temporal Concurrent SC

In this section we show how to represent a CBI Model in the Temporal Concurrent SC framework.

Attributes and Intervals. For each token $\langle v, p(\vec{x}), t_s, t_e\rangle$ we introduce a fluent $P_v(\vec{x}, t_s, s)$ and two actions $start\_p(\vec{x}, t)$ and $end\_p(\vec{x}, t)$ representing, respectively, the $p(\vec{x})$ process (here t_s is the start time) starting and ending events.

The temporal model will be defined by a BAT. In particular the successor state axiom (SSA) is defined as follows:

$$\begin{array}{c} P_v(\vec{x}, t_s, do(c,s)) \equiv \exists a.P\_start(\vec{x}, a, s) \wedge a \in c \wedge time(a) = t_s \vee \\ (\exists t).P_v(\vec{x}, t, s) \wedge \neg(\exists a').P\_end(\vec{x}, a', s) \wedge a' \in c. \end{array}$$

Here $P\_start(\vec{x}, a, s)$ ($P\_end(\vec{x}, a, s)$) is true if a starts (ends) $P_v(\vec{x}, t_s, s)$ in s. Continuing the previous example, $\langle Lc, going(x), t_s, t_e\rangle$ can be represented by the $going(x,t,s)$ fluent whose SSA is represented as follows:

$$\begin{array}{r} going(x,t,do(c,s)) \equiv start\_going(x,t) \in c \vee \\ going(x,t,s) \wedge (\forall t')end\_going(x,t') \notin c. \end{array}$$

The $start\_p$ and end_p actions are specified by action preconditions axioms, where $\phi_p(\vec{x}, t, s)$ are sentences specifying the conditions under which the $start$ and end actions can be executed:

$$\begin{array}{l} Poss(start\_p(\vec{x}, t), s) \equiv \phi_p(\vec{x}, t, s) \\ Poss(end\_p(\vec{x}, t), s) \equiv \phi_p(\vec{x}, t, s) \end{array}$$

Domain Constraints. Given the CBI model $M = (A, I, R)$, the R constraints can be captured in the temporal BAT by exploiting the axiom preconditions needed in the concurrent action specification (to address the precondition interaction problem). For instance, the constraint on the token duration can be expressed by:

$$\begin{array}{c} Poss(c,s) \supset (\exists t)[A\_ends(\vec{x}, a, s) \wedge A(\vec{x}, t, s) \wedge a \in c \supset \\ d \leq time(a) - t \leq D] \end{array}$$

where $A\_end(\vec{x}, a, s)$ ($A\_start(\vec{x}, a, s)$) is true if a ends (starts) $A(\vec{x})$ in s. Analogously the Allen-like temporal constraints introduced above (see Figure 1) can be easily represented in the concurrent temporal BAT (see Figure 1):

- $A(\vec{x})$ $meets$ $B(\vec{x})$:

$$\begin{array}{c} Poss(c,s) \supset \exists a.A\_end(\vec{x}, a, s) \wedge a \in c \supset \\ \exists a'.B\_start(\vec{x}, a', s) \wedge a' \in c. \end{array}$$

- $A(\vec{x})$ met_by $B(\vec{x})$:

$$\begin{array}{c} Poss(c,s) \supset \exists a.A\_start(\vec{x}, a, s) \wedge a \in c \supset \\ \exists a'.B\_end(\vec{x}, a', s) \wedge a' \in c. \end{array}$$

- $A(\vec{x})$ $starts$ $B(\vec{x})$:

$$\begin{array}{c} Poss(c,s) \supset \exists a.A\_start(\vec{x}, a, s) \wedge a \in c \supset \\ \exists a'.B\_start(\vec{x}, a', s) \wedge a' \in c. \end{array}$$

- $A(\vec{x})$ $ends$ $B(\vec{x})$:

$$\begin{array}{c} Poss(c,s) \supset \exists a.A\_end(\vec{x}, a, s) \wedge a \in c \supset \\ \exists a'.B\_end(\vec{x}, a', s) \wedge a' \in c. \end{array}$$

- $A(\vec{x})$ $contained\_by$ $B(\vec{x})$:

$$\begin{array}{c} Poss(c,s) \supset [\exists a.A\_start(\vec{x}, a, s) \wedge a \in c \supset B(\vec{x}, s) \wedge \\ \neg\exists a'.B\_end(\vec{x}, a', s) \wedge a' \in c] \wedge \\ [\exists a.B\_end(\vec{x}, a, s) \wedge a \in c \supset \neg A(\vec{x}, s) \vee \\ \exists a'.A\_end(\vec{x}, a', s) \wedge a' \in c]. \end{array}$$

- $A(\vec{x})$ $contains$ $B(\vec{x})$ is recursively defined once we introduce two auxiliary fluent/processes $AmeetsB(\vec{x})$ and $AendsB(\vec{x})$, s.t.:

$$\begin{array}{l} A(\vec{x})\ starts\ AmeetsB(\vec{x}),\ \ AmeetsB(\vec{x})\ cont\_by\ A(\vec{x}), \\ A(\vec{x})\ starts\ AendsB(\vec{x}),\ \ AendsB(\vec{x})\ cont\_by\ A(\vec{x}), \\ AmeetsB(\vec{x})\ meets\ B(\vec{x}),\ \ AmeetsB(\vec{x})\ cont\_by\ AendsB(\vec{x}), \\ AendsB(\vec{x})\ ends\ B(\vec{x}). \end{array}$$

- $A(\vec{x})$ $before[d, D]$ $B(\vec{x})$ is recursively defined by:

$$A(\vec{x})\ meets\ A\_bf\_B(\vec{x}, d, D),\ \ A\_bf\_B(\vec{x}, d, D)\ meets\ B(\vec{x}),$$

  where $A\_bf\_B(\vec{x}, d, D, s)$ is an auxiliary fluent/process whose duration ranges over the interval $[d, D]$.

- $A(\vec{x})$ $after[d, D]$ $B(\vec{x})$:

$$\begin{array}{l} Poss(c,s) \supset \exists a,t.A\_start(\vec{x}, a, s) \wedge a \in c \wedge time(c) = t \supset \\ \quad after\_B(\vec{x}, t, d, D, s). \end{array}$$

  where $after\_B(\vec{x}, t, d, D, s)$ is an auxiliary fluent which is true if there exists an action a ending $B(\vec{x})$, with $d \leq time(s) - time(a) \leq D$.

Once all the temporal constraints are specified in this way, the $Poss(c,s)$ definition can be obtained by the closure of its necessary conditions.

Planning Problem. Once the domain temporal constraints have been represented in the Concurrent Temporal Situation Calculus, the planning problem is defined by a *candidate plan* representing both the initial situation and the system goals. We recall that the candidate plan is defined by: i. a planning horizon (t_h, T_h); ii. timelines $\mathcal{T}_i$ for each state variable σ_i; iii. ordering constraints between tokens in the same timelines; iv. a set of constraints C_i each of the kind: $[t_{s_i}, T_{s_i}]\ p_i(\vec{r})\ [t_{e_i}, T_{e_i}]$.

To represent the *candidate plan* as *golog program* in the concurrent temporal golog, we do not deploy occurencies and narratives [13] since it is not a domain constraint, as it defines the control knowledge (goals). We assume a complete specification of $\mathcal{D}_{S_0}$; the Golog scripting language is used to represent the C_i constraints (in the SC we have to distinguish among initial situation and goals). This is possible introducing, for each C_i, a procedure definition of the following kind:

$$\begin{aligned}\mathbf{proc}(c_i,\ (T_{e_i} > \mathit{horizon})?\ |((T_{e_i} \leq \mathit{horizon})\wedge\\ (\exists t, t').\mathit{end\_P}_i(\vec{r}, t, t')\wedge\\ t_{s_i} \leq t' \leq T_{s_i} \wedge t_{e_i} \leq t \leq T_{e_i})?).\end{aligned}$$

This procedure is composed of only two tests: if the end time constraint is beyond the horizon the constraint is neglected, otherwise, the start and end timepoints have to satisfy the temporal constraints. Here $\mathit{end\_P}_i(\vec{r}, t, t', s)$ is a fluent properties which is true iff $P_i(\vec{x}, t, s)$ ends in s at t'. For example, $[5, 10]\mathit{going}(d, e)[6, 10]$ can be represented as

$$\begin{aligned}\mathbf{proc}(c_2,\ (10 > \mathit{horizon})?\ |(10 \leq \mathit{horizon})\wedge\\ (\exists t, t').\mathit{end\_going}(d, e, t, t')\wedge\\ 5 \leq t' \leq 10 \wedge 6 \leq t \leq 10)?).\end{aligned}$$

Given these procedures, an incomplete plan over a timeline $\mathcal{T}_j$ can be define by the following procedure:

$$\begin{aligned}&\mathbf{proc}(\mathit{plan\_}\mathcal{T}_j,\\ &\quad \pi(n,\ (\mathit{select}(n))?\ :\ \mathit{plan}_j(n)\ :\ c_1)\ :\\ &\quad \pi(n,\ (\mathit{select}(n))?\ :\ \mathit{plan}_j(n)\ :\ c_2)\ : \ldots :\\ &\quad \pi(n,\ (\mathit{select}(n))?\ :\ \mathit{plan}_j(n)\ :\ c_k)),\end{aligned}$$

where $\mathit{plan}(n)$ is a planner whose depth is bounded by n representing the maximal number of gaps (tokens) between c_i and c_{i+1}. $\mathit{plan}(n)$ implements a simple planning algorithm, e.g. we can deploy the following straightforward algorithm:

$$\begin{aligned}&\mathit{proc}(\mathit{plan}_j(n),\\ &\quad \mathit{true}?\ |\ \pi(a, (\mathit{primitive\_action}(a, j))?\ :\ a)\ :\ \mathit{plan}(n-1))\end{aligned}$$

where $\mathit{primitive\_action}(a, j)$ is to select a primitive action belonging to the $\mathcal{T}_j$ timeline. For example, the $\mathit{Navigation}$ timeline introduced in Section 2.2, can be represented by a $\mathit{plan\_}\mathcal{T}_{Nav}$ with c_2 defined as before and c_1 as follows:

$$\mathbf{proc}(c_1,\ (10 > \mathit{horizon})?\ |(10 \leq \mathit{horizon} \wedge \mathit{end\_at}(a, 0, 3))?).$$

Once an incomplete plan is over a timeline $\mathcal{T}_i$, given a set of timelines $\{\mathcal{T}_i\}$, a candidate plan becomes a parallel execution of its own procedure $\mathit{plan\_}\mathcal{T}_i$:

$$\mathbf{proc}(\mathit{c\_plan},\ \mathit{plan\_}\mathcal{T}_1\ : \mathit{nil}\ \|\ \ldots\ \|\ \mathit{plan\_}\mathcal{T}_k\ :\ \mathit{nil}\ ).$$

Given a BAT encoding the action theory, and the temporal constraints among the activities, any ground situation σ s.t. $BAT \models Do(\mathit{c\_plan}, S_0, \sigma)$ represents a CBI *possible behavior*. More precisely, let $f(\sigma)$ be a behavior at a ground situation σ, $M = (I, A, R)$ a CBI model, defined in BAT, then for any *candidate plan* P_c there is a $\mathcal{D}_{S_0}$ and a $\mathit{c\_plan}$ CTGolog procedure such that p is a *possible behavior* of (M, P_c), if there exists a σ (ground) with $f(\sigma) = p$ and $BAT \models Do(\mathit{c\_plan}, S_0, \sigma)$.

A CBI *sufficient plan* is a complete CBI plan with maximal flexibility. In the SC framework we can represent a sufficient plan as the couple $\langle s[\vec{x}, \vec{t}], \mathit{Constr}(\vec{x}, \vec{t})\rangle$ where $\mathit{Constr}(\vec{x}, \vec{t})$ is a minimal set of constraints (among time variables $\vec{t}$ and argument variables $\vec{x}$) s.t.

$$BAT \cup \{\mathit{Constr}(\vec{x}, \vec{t})\} \models Do(\mathit{c\_plan}, s_0, s[\vec{x}, \vec{t}]).$$

Given this representation, it is possible to show that if $p_{sc} = \langle s[\vec{x}, \vec{t}], \mathit{Constr}(\vec{x}, \vec{t})\rangle$ is s.t. the previous property holds, then the associated p_{CBI} CBI plan is a *sufficient plan*. Notice that the mapping does not work in the other direction, i.e. there exists a CBI flexible plan p_{CBI} which cannot be captured by a p_{sc}. This is due to the fact that a complete CBI plan is identified with a situation $s[\vec{x}]$, where the order of two (starting or ending) events a_1 and a_2 belonging to two different timelines is already decided: at planning time the compiler decides if a_1 starts before, after, or concurrently with a_2. Instead, in a CBI sufficient plan this order between events can be defined at execution time. A complete SC mapping of the sufficient CBI plan needs a more complex representation where each plan is associated to a set of flexible situations, we leave this issue to future work.

4 Example

We consider a rescue domain where a rover is to explore an unknown environment in order to map and localize victims. We assume the rover endowed with a pan-tilt and stero-cameras. Visual perception is exploited to detect interesting locations where it's worth to go and perform the observations. Basically the robot has to provide two main activities: exploring and mapping; search for victims. While the robot is in the exploration mode a rough visual perception ($\mathit{vp\_monitor}$) is always active in order to tag the map with salient regions. A more complex visual analysis is performed ($\mathit{vp\_analsys}$) in order to detect victims, during this activity the rover must be stopped while the pan-tilt and range-finder are coordinated in order to scan a salient portion of the visual space.

We consider the following state variables: *Pant-tilt*, *RangeFinder*, *LocMap*, *Navigation*, *VisualPerception*, *Mode*. Each state variable is associated with a set of processes/tokens. *Pan-tilt* can either be idling in pos $\vec{\theta}$ ($\mathit{Pt\_idle}(\vec{\theta})$), moving toward $\vec{\theta}$ ($\mathit{Pt\_moving}(\vec{\theta})$), or scanning ($\mathit{Pt\_scanning}(\vec{\theta})$); *Range finder*[3] states are $\mathit{Rf\_idle}$ and $\mathit{Rf\_active}$; *LocMap* maps and tracks the robot position via the $\mathit{Lc\_at}(\vec{x})$ and $\mathit{Lc\_goTo}(x)$ tokens; *Navigation* represents the navigation state through: $\mathit{Nv\_stop}$, $\mathit{Nv\_movingTo}(\mathit{speed})$, $\mathit{Nv\_wandering}$; *Visual Perception* represents the state of the visual perception module: it can be either idle ($\mathit{Vp\_idle}$), activated to detect interesting objects in the environment $\mathit{Vp\_monitor}$, or analyzing an interesting region from $\vec{x}$ robot position with pant-tilt in θ: $\mathit{Vp\_analisys}(\vec{x}, \vec{\theta})$. *Mode* can be $\mathit{Md\_map}(st)$ or $\mathit{Md\_search}(st)$, where *st* is *ok* if the activity succeeds and *no* if it fails.

Hard time constraints among the activities can be defined by a temporal model in CBI style. For example, $\mathit{Vp\_monitor}$ and $\mathit{Vp\_observe}(\vec{x}, \vec{\theta})$ are respectively associated with the mapping and the search modes, hence we have $\mathit{Vp\_monitor}\ \mathit{cont\_by}\ \mathit{Md\_map}(st)$ and $\mathit{Vp\_observe}(\vec{x}, \vec{\theta})\ \mathit{cont\_by}\ \mathit{Md\_search}$. The search mode can be started only if the local environment is mapped with success $\mathit{Md\_search}(st)\ \mathit{met\_by}\ \mathit{Md\_map}(ok)$. The victim detection requires the rover to be stopped $\mathit{Vp\_analisys}(\vec{x}, \vec{\theta})\ \mathit{cont\_by}\ \mathit{Nv\_stop}$. The visual analysis needs

[3] It is actually a telemeter returning the precise distance of the point hit.

the pan-tilt scanning $Vp\_analisys(\vec{x}, \vec{\theta})$ $cont$ $Pt\_scanning(\theta)$ and the $Pt\_scanning$ can start only if met_by $Pt\_idle(\theta)$, etc.

Since these constraints are represented by the $\mathcal{D}_{AP}$ of the BAT, the embedded CBI temporal model is directly combined with the other dynamic properties specified in the SC language. Now, given the $(0, 1000)$ plan horizon, the following partial plan should force the exploration of an unknown environment:

$Md : [0, 0]Md\_map(ok)[10,\ 100]Md\_search(ok)[11,\ 1000]$
$Nv : [0, 0]Nv\_idle[0, 1000]$
...
$Pt : [0, 0]Pt\_idle(\vec{\theta})[0, 1000]$

where, except for *Mode*, each timeline has only the initial activity defined. Following the approach presented in Section 3, this partial plan can be translated into the Golog procedure:

$$\mathbf{proc}(c\_plan,\\ plan\_Md\ : nil \parallel plan\_Nv\ :\ nil \parallel \ldots \parallel plan\_Pt\ :\ nil),$$

where $plan\_Md$ is defined by the two time constraints $[0, 0]Md\_map(ok)[10,\ 100]Md\_search(ok)[11,\ 1000]$, and the other procedures are encoded as generic planners. However, in order to make this planning activity feasible, the Golog scripting language can be exploited to directly encode some control knowledge. For example, the $Plan\_Pt$ procedure can be written as follows:

$$\mathbf{proc}(plan\_Pt,\ \pi(t, \pi(t', (\exists x.Md\_map(x, t))?| \\ (\exists x.Md\_search(x, t))?\ :\ wait\_location\ : \\ Ptscan\ :\ PtIdle : (time = t' \wedge t - t' \leq d)?)),$$

where $wait\_location\ :\ Ptscan\ :\ PtIdle$ are three Golog procedure defining the expected pant-tilt behavior during the search mode. The final test enforces a maximal d time duration for the whole procedure execution.

5 Implementation

We provided a constraint logic programming (CLP) [6] implementation of the CTGolog based control system for the rescue domain. Since in this setting the CTGolog interpreter is to generate flexible temporal plans, it must be endowed with a constraint problem solver. Analogous to [14] we rely on a logic programming language with a built-in solver for linear constraints over the reals (CLP($\mathcal{R}$)). In this setting logical formulas, allowed for the definition of predicates, are restricted to be horn clauses of the form: $A \leftarrow c_1, \ldots, c_m | A_1, \ldots, A_n$, where c_i are constraints and A_j are atoms. Specifically, we appeal to the ECRC Common Logic Programming System ECLIPSE 5.7. In this way our planner and domain axioms make use of linear temporal relations like $2 * T_1 + T_2 = 5$ and $3 * T_2 - 5 \leq 2 * T_3$, and we rely on ECLIPSE to performing the reasoning in the temporal domain. The relations managed by the ECLIPSE built-in constraint solver have # as a prefix, for example, a temporal constraint represented in the Golog interpreter is:

```
do(C : A,S,S1) :- concurrent_action(C),
 poss(C,S), start(S,T1), time(C,T2), T1 #=< T2,
 do(A,do(C,S),S1).
```

Other temporal constraints are expressed in the action preconditions, for example, considering the pan-tilt processes:

```
poss(pt_pos_start(X,T),S) :-
  pt_idle(X,T1,S),T1 #< T,start(S,T2),T2 #>= T,
  nv_stop(T11,S),T11 #<T.
```

An example of the successor state axioms is the following.

```
pt_idle(X,T1,do(C,S)) :-
   pt_idle(X,T1,S) not member(pt_pos_start(_,T2),S);
   member(pt_pos_end(X,T1),S).
```

Given the BAT specification, for each timeline it is possible to specify a control procedure

```
proc(pt_go(X), pi(t1, [pt_pos_start(X,t1)]:
                 pi(t2, [pt_pos_end(X,t2)] )) ).
```

Once the flexible temporal plan is compiled, it can be executed. We assume an execution monitor *cycleExec* which sends and receives commands at each time tick so that constraints can be, step by step solved and/or propagated. A dummy implementation of *cycleExec* is shown below.

```
planExec :-
      do(c-plan,s0,S),!,cycleExec(1,s0,S,S1).
cycleExec(T,S0,S0,S0) :- !.
cycleExec(T,S0,S,S1) :-
    checkMsg(T), exec(T,S0,S,S1), checkMsg(T),
        T1 is T+1,!, cycleExec(T1,S1,S,S2).
```

6 Summary and Outlook

We presented an approach to the embedding of the CBIP paradigm in the Golog framework. Several issues are left to future work, among them: progression and forgetting the past, parallel planning over independent timelines, failure management.

REFERENCES

[1] J.F. Allen, 'An interval-based representation of temporal knowledge', in *IJCAI*, (1981).

[2] A.K. Jonsson D.E. Smith, J. Frank, 'Bridging the gap between planning and scheduling', *Knowledge Engineering Review*, **15**(1), (2000).

[3] Y. Lesperance G. De Giacomo and H. Levesque, 'Congolog, a concurrent programming language based on the situation calculus', **121**, (2000).

[4] Malik Ghallab and Herv Laruelle, 'Representation and control in ixtet, a temporal planner', in *AIPS 1994*, pp. 61–67.

[5] H. Grosskreutz and G. Lakemeyer, 'ccgolog – a logical language dealing with continuous change', *Logic Journal of the IGPL*, **11**(2), 179–221, (2003).

[6] Joxan Jaffar and Michael J. Maher, 'Constraint logic programming: A survey', *Journal of Logic Programming*, **19/20**, 503–581, (1994).

[7] Ari K. Jonsson, Paul H. Morris, Nicola Muscettola, Kanna Rajan, and Benjamin D. Smith, 'Planning in interplanetary space: Theory and practice', in *Artificial Intelligence Planning Systems*, pp. 177–186, (2000).

[8] Doherty P. Kvarnstrm, J. and P. Haslum, 'Extending talplanner with concurrency and resources'.

[9] J. McCarthy, 'Situations, actions and causal laws', Technical report, Stanford University, (1963). Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410-417.

[10] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams, 'Remote agent: To boldly go where no AI system has gone before', *Artificial Intelligence*, **103**(1-2), 5–47, (1998).

[11] J.A. Pinto, 'Integrating discrete and continuous change in a logical framework', *Computational Intelligence*, **14**(1), 39–88, (1998).

[12] J.A. Pinto and R. Reiter, 'Reasoning about time in the situation calculus', *Annals of Mathematics and Artificial Intelligence*, **14**(2-4), 251–268, (September 1995).

[13] Javier Pinto, 'Occurrences and narratives as constraints in the branching structure of the situation calculus', *Journal of Logic and Computation*, **8**(6), 777–808, (1998).

[14] Fiora Pirri and Raymond Reiter, 'Planning with natural actions in the situation calculus', 213–231, (2000).

[15] R. Reiter, 'Natural actions, concurrency and continuous time in the situation calculus', in *Proceedings of KR'96*, pp. 2–13, (1996).

[16] Raymond Reiter, *Knowledge in action : logical foundations for specifying and implementing dynamical systems*, MIT Press, 2001.

[17] Raymond Reiter and Zheng Yuhua, 'Scheduling in the situation calculus: A case study', *Annals of Mathematics and Artificial Intelligence*, **21**(2-4), 397–421, (1997).

[18] B. Williams, M. Ingham, S. Chung, P. Elliott, M. Hofbaur, and G. Sullivan, 'Model-based programming of fault-aware systems', *AI Magazine*, (Winter 2003).