Dealing with Failures for Execution Consistency in Context-aware Systems

Ahmed-Chawki Chaouche, Jean-Michel Ilie, François Pêcheux

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 177 (2020) 212–219

Procedia Computer Science

www.elsevier.com/locate/procedia

The 11th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2020)
November 2-5, 2020, Madeira, Portugal

Dealing with Failures for Execution Consistency in Context-aware Systems

Ahmed-Chawki Chaouche[a,*], Jean-Michel Ilié[b], François Pêcheux[b]

[a]*MISC Laboratory, University Abdelhamid Mehri - Constantine 2, Campus Ali Mendjeli, 25000 Constantine, Algeria*
[b]*LIP6, UMR 7606 Sorbonne Université, 4 Place Jussieu, 75005 Paris, France*

Abstract

In this paper, we consider a symbolic mechanism designed to help guide the performance of actions of a Belief-Desire-Intention (BDI) agent under both concurrency and relevance criteria. Behind the planning activity which can optimistically estimate the relevance of traces, we propose to directly supervise the best effort execution of the intentions as to keep track of the most relevant ones in case of failures. To support the presented method, a typical use case is given that targets the automated guidance of an autonomous vehicle.

Keywords: Contextual planning and guidance; Relevant execution; Plan multi revision strategy; Failure handling

1. Introduction

The automated guidance of mobile entities in a changing ambient environment is a great challenge supported by many academic and industrial works. It serves several huge industrial projects such as smart city, e.g. [5, 8]. As the ambient context can change anywhere and anytime, such an objective requires the planning action to be context aware in order to dynamically adapt the execution directives on the fly.

Various planning approaches have already been proposed to handle the ambient complexity, e.g. [2]. However, few ones actually propose to exploit the attitudes of a Belief-Desire-Intention (BDI) software agent to adapt itself to unexpected changing situations [1]. In BDI approaches, each agent can be designed as a powerful software entity, which allows reasoning in terms of agent intentions, coming from the agent's beliefs and desires. Intentions are realized through some action plans that are usually assumed to be known. High-order Agents (HoA) agents [3] are

* Corresponding author. Tel.: +33-678-139-432 ; fax: +33-144-277-495.
E-mail address: ahmed.chaouche@univ-constantine2.dz

1877-0509
10.1016/j.procs.2020.10.030

particularly well-adapted BDI agents for ambient systems which gracefully handle the concurrency of intentions and autonomously learn with physical information to provide adapted execution plans working contextually.

In this paper, we aim at building the symbolic guidance of a self-driving robot fulfilling its missions contextually, in an ambient environment. The core difficulty is to maintain the viability of an execution plan (soundly and efficiently) since the concrete execution of actions may actually fail. Replanning operations are then required. Taking into account the concurrency of actions may lead to an exponential use in the planning memory, with the risk of damaging the achievement of missions under timing constraints. In contrast, we aim at producing the best effort to reinforce the execution of the selected plan even if some sub-plans are indeed removed or replanned. With regards to a complete symbolic replanning procedure, we investigate an intermediate planning operation locally to the execution control with two advantageous corollaries, free time for the BDI agent reasoning and an improved reactivity for the concrete driving operations.

Associating weights with symbolic information allows to abstractly represent the consequences of a symbolic reasoning, e.g. the authors of [9] use weights to solve conflicts between beliefs and in [4], weights are proposed to express scheduling directives between concurrent intentions. In this paper, we agree on such notions but we also assume that each intention can be reassigned with a new weight value expressing the relative relevance of the intention regarding to the current mission. In fact, scheduling concepts cannot account for the relevance of intentions, since relevance does not necessarily mean being executed first.

In this paper, We show how both the planning and the execution processes can take care of these weights to demonstrate an efficient and consistent execution plan with respect to the intentions to be achieved. The idea is to introduce both scheduling and relevance information for each intention, i.e. a double weighted function which values result from the deliberation of the Mental process: (1) The *scheduling weight* allows to partition the intentions in distinguished ordered subsets that can be achieved in sequence. In other terms, the subset of ingress intentions having a high scheduling weight value must be achieved first. Moreover, two intentions of the same subset can be achieved concurrently, independently of their associated relevance weights ; (2) The *relevance weight* allows us to specify the relevance of intentions according to their achievement priorities. In other terms, the agent gives preference to some execution plans with the aim of achieving the intentions having the highest relevance.

The schedule of the paper is the following: Section 2 schemes an embedded extension of the HoA multi-process architecture proposed in [6] which allows controlling the driving operations by an intelligent contextual planning. Based on this architecture, Section 3 shows how the planning process can formally handle both intention weights, up to obtain an optimal execution plan available from some context. Basic concepts to deal with the relevance of actions are then explained in this section. Section 4 represents the most original part of this paper, introducing an efficient relevance-based algorithm to directly adapt the execution plan on the fly, whenever the performance of some action does not conform to the expected execution. Section 5 is our conclusion.

2. E-HoA Embedded Architecture

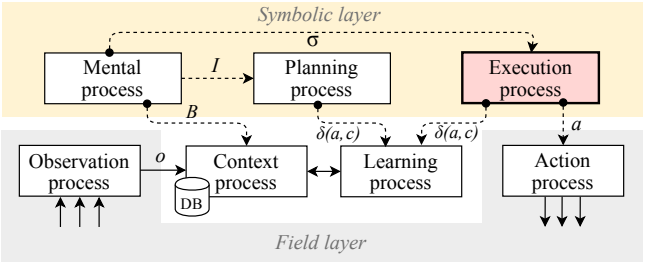

Fig. 1: Embedded Higher-order Agent architecture

Intentions	W_s	W_r	*Plan expressions*
*Order*1	2	2	$move(ph1);deliver(ph1);exit$
*Order*2	2	2	$move(ph2);deliver(ph2);exit$
*Order*3	2	2	$move(ph3);deliver(ph3);exit$
Back	1	3	$move(dep);park(dep);exit$
Recharge	2	1	$move(stn);refill(stn);exit$

Table 1: Intentions information for the scenario

Figure 1 highlights the Embedded Higher-order Agent (E-HoA) layered architecture [6], based on a set of co-operating processes that altogether define the robot behavior. Like other robotic system architectures, such as 3T, ATLANTIS and LAAS [7], E-HoA subsums the robot behavioral information with the price to handle all the event messages at the deliberative/planning layer. Nevertheless, the E-HoA symbolic layer already manages weights to ex-

press the concurrent schedule of the intention behaviors. The major process in this layer is the *Mental process* which reasons in terms of Beliefs (B), Desires (D) and Intentions (I) [1]. The Mental process, aiming at optimizing the achievement of the agent's intentions, asks the *Planning process* to compute the best execution plan (σ, list of actions a) from the original weighted intentions (see Section 3). The Planning process is specialized so to compute the best execution plan in terms of concrete action duration ($\delta(a,c)$) according to some spatio-temporal context c. Durations are learned from the driving experiences captured at the field layer of the E-HoA architecture. They are managed by two intermediate processes of great importance, as the *Learning process* extracts the historical (driving) context of the robot stored by the *Context process*.

The contribution of the present paper can be viewed as an enhancing version of the *Execution process* of the E-HoA architecture to alleviate the task of the Mental process/planning process. Originally, the Mental process delegates to the *Execution process* the task of sequencing the actions specified by the execution plan. We also offer the execution process the task of taking care of the event messages which can be solved by reviewing the execution plan (see Section 4).

Delivery Motivating Sample. The use-case scenario of this paper considers a pharmacy deposit and a network of pharmacies that asynchronously and independently mail orders to the deposit in order to receive prescriptions and drugs. A driving robot is in charge of the actual delivery. On a daily basis, the objective of the robot consists in delivering the pharmacies that require a specific list of drugs, then in coming back to the depot to park for a new routine. Immersed in an ambient environment, the robot must be able to react to the context changes and adapt its behavior accordingly [5]. Intentions are achieved by means of action plans that are assumed to be known and stored in an action plan library.

On a daily basis, the objective of the E-HoA consists in delivering the pharmacies that require a specific list of drugs, then in coming back to the depot and park. Opportunistically, during a daily tour, the robot can decide to recharge its battery without compromising its main intentions.

The E-HoA agent includes an efficient contextual planner that handles every delivery tour prerequisites. By default, the intentions of the agent are assumed to be achieved concurrently whenever possible. The intentions column of Table 1 highlights an illustrative example of what could be the intentions of the agent embedded in the robot.

Knowing that scheduling and relevance information are introduced for each intention $i \in I$ by the double weighted function $W_s, W_r : I \rightarrow \mathbb{N}$, $W_s(i)$ represents the scheduling weights of i and $W_r(i)$ represents its relevance weights. The corresponding weighted intention is denoted $i^{(W_s,W_r)}$. In our motivating example, we have $I = \{Order1^{(2,2)}, Order2^{(2,2)}, Order3^{(2,2)}, Back^{(1,3)}, Recharge^{(2,1)}\}$, such that the intentions $Order1$, $Order2$ and $Order3$ correspond to 3 pharmacy orders to be delivered, *Back* means returning to the depot and *Recharge* to fill up the robot battery. The three *Order* intentions and both the *Recharge* intentions have the same W_s value, hence can be achieved concurrently, whereas, the *Back* intention must be scheduled after all the other intentions. Due to the W_r weight, the *Back* intention is more relevant than all the *Order* intentions which are themselves more relevant than the *Recharge* intention.

The core difficulty to build a plan of a daily tour is to predict traffic jams that would slow down the delivery dramatically and try avoiding them, taking into account the context. Actually, it is interesting to anticipate and compute the best plan according to the expected durations of the actions.

3. From Reasonning to Scheduled Actions

The activity of the Planning process starts with a preliminary stage, which consists in formalizing a modular plan expression, called *agent plan* ($\overline{P}$). It is derived from the weighted set of intentions selected by the Mental process and their associated plans, named intention plans ($\widehat{P}$). According to the scheduling weights of the intentions, the agent plan is specified as an algebraic expression over the action plans attached to the intentions, namely the intention plans. This can be formally done by using the AgLOTOS algebraic language [4]. In reference of our frame example, the agent plan corresponding to the intentions of Table 1 has the following shape: $\overline{P} ::= (\widehat{P_1}|||\widehat{P_2}|||\widehat{P_3}|||\widehat{P_R}) \gg \widehat{P_B}$, where '$|||$' is the parallel operator and '$\gg$' is the sequential one. Hence, all but one of the mentioned intention plans are considered concurrently by the planning process and only $\widehat{P_B}$, which is the intention plan for *Back*, will be executed after the 4 former one. Further for sake of simplicity, we consider that each intention plan is an elementary action plan

kept simple as this is described in the last column of Table 1, In fact, only the prefexing operator ';' is used to specify that the actions must be performed in sequence.

Contextual Planning System. Then, based on the formal semantics of AgLOTOS, the Planning process is able to build a predictive tree structure called Contextual Planning System (CPS). This is used to express the evolution of the agent plan upon the different interleavings of the actions in plans respecting contextual constraints, mainly pre- and post-conditions bound to actions. Figure 2 illustrates a part of the CPS construction exploited in our frame pharmacy example. States are labelled by the predicted evolution of a given context. Here, the context merely focuses on a spatio-temporal information (ℓ,t), i.e. a location and a time value, since this seems at first glance a good abstraction of a real context. Spatially, the context evolves according to the move actions and the temporal evolution is deduced with respect to the duration of actions estimated by the Learning process. The notation $\delta(a,ctx)$ will represent such a duration value for any action a to be performed from some context ctx. Formally, the CPS construction exhibits a set of transitions representing the possible evolution from an initial contextual planning state. It is assumed to be finite.

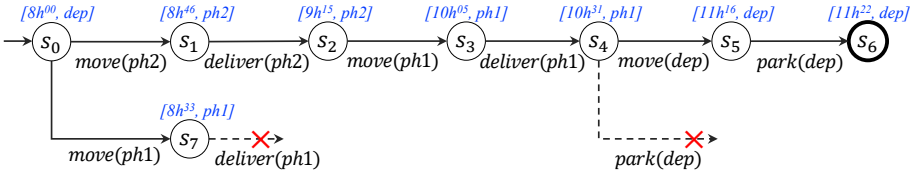

Fig. 2: Example of CPS traces

Definition 3.1. *(Contextual planning state)*
Let CTX be a set of the possible contexts managed by the Context process and $\widehat{\mathcal{P}}$ be the set of all the possible intention plans. A contextual planning state is a tuple $(\overline{[P]},ctx,T)$, where $\overline{[P]}$ is any planning state, $ctx \in CTX$ is the current context in this state and $T \subseteq \widehat{\mathcal{P}}$ is the set of terminated intention plans.

Let ACT be the set of symbolic actions. The spatio-temporal derivation of a contextual planning state $(\overline{[P_1]},ctx_1,T_1)$ caused by an action $a \in ACT$ of duration value $\delta(a,ctx_1)$ where $ctx_1 = (\ell_1,t_1)$, leads to the transition $(\overline{[P_1]},(\ell_1,t_1),T_1) \xrightarrow{a} (\overline{[P_2]},(\ell_2,t_2),T_2)$, such that $\overline{[P_1]} \xrightarrow{a} \overline{[P_2]}$, $pre(a) \models (\ell_1,t_1)$, $(a = move(\ell) \implies \ell_2 = \ell) \wedge (a \neq move(\ell) \implies \ell_2 = \ell_1)$, $t_2 = t_1 + \delta(a,(\ell_1,t_1))$, $(a = exit) \wedge (\widehat{P} = exit) \implies T_2 = T_1 \cup \{\widehat{P}\}$, and $post(a) \models (\ell_2,t_2)$.

The initial contextual planning state of our scenario is $(\overline{[P]},(dep,8h),\emptyset)$, standing for the robot starts its round tour at 8 AM from the depot, considering the agent plan $\overline{[P]}$ as the daily mission. Back to Figure 2, the CPS trace $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \rightarrow s_6$, viewed as an execution plan, is reachable from the initial context. This trace specifies that there are two pharmacy deliveries, for $ph1$ and $ph2$, before coming back to the depot to park the robot. Observe also that from the state s_7, a temporal constraint avoids the action $deliver(ph1)$ since this action is assumed not to be possible before $9h$. From the state s_4, the *park* action is not valid due to a spatial constraint expressing that the robot ought to be in the depot location.

Plan Relevance. To guide the agent efficiently, the Planning process can select an execution trace which maximizes the number of intentions that can be achieved. This can be captured over all the possible traces of the CPS. We propose to select interesting traces from the CPS by exploiting the relevance weights of the achieved intentions. Further, the mapping $end(\sigma)$ yields the set of achieved intentions for the trace σ. It is deduced from the set of terminated intention plans (T), declared within the final contextual planning state of σ. Based on this mapping, we compare the traces on their respective relevances, featured by their *(normalized) relevant configurations*.

Definition 3.2. *(Relevant configuration)*
A relevant configuration R_σ is a map indexed over I, such that $R_\sigma(i) = W_r(i)$ if $i \in end(\sigma)$, and $R_\sigma(i) = 0$ otherwise. A normalized relevance configuration $\overline{R_\sigma}$ is a sorted map directly deduced from R_σ, by considering a decreasing order of the intention relevance values.

Corollary 3.1. *(Maximum relevance trace)*
Consider any two traces σ_1 and σ_2 of a CPS, σ_1 is more relevant than σ_2 iff $\overline{R_{\sigma_1}} > \overline{R_{\sigma_2}}$. A maximum relevance trace is defined for the CPS as a trace having the maximum normalized relevance configuration over the traces of the CPS.

Table 2: Examples of relevant traces

Trace σ	$end(\sigma)$	R_σ	$\overline{R_\sigma}$	$\delta(\sigma)$
σ_1	$\{Back, Order1, Order2\}$	322000	322000	3h22
σ_2	$\{Back, Order2, Order3\}$	302200	322000	3h47
σ_3	$\{Order1, Order2, Order3, Recharge\}$	022201	222100	3h13

Table 2 demonstrates how 3 traces of a same finite CPS can be associated with a normalized relevance configuration based on the end analyses of their final contextual planning states. Notice that both the traces σ_1 and σ_2 have the same maximum normalized configuration (i.e. $\overline{R_{\sigma_1}} = \overline{R_{\sigma_2}}$), hence cannot be distinguished, but the trace σ_3 will not be considered any more since having a lesser relevance. Moreover, many multi criteria could be used in order to distinguish the optimal trace in between maximum traces. In this paper, we simply consider the expected duration of the traces. In the table, σ_1 must be selected as the optimal trace. By considering that any trace σ is ordered, the notation $first(\sigma)$ and $last(\sigma)$ are the first and last transitions in σ, $tr.act$ is the action of tr, and $tr.src$, $tr.dest$ respectively represent the source and destination contextual planning states of tr. Thus, the expected duration of any σ is defined as $\delta(\sigma) = last(\sigma).dest.ctx.t - first(\sigma).src.ctx.t$.

4. Concrete Execution of Actions

When the Mental process gets an optimal trace σ from the Planning process, it delegates the Execution process to execute it under global constraints and subscribe to the progress information. In fact, it must be aware that some of the represented intentions fail as the trace is executed. This offers the way for the Mental process to take high-level decisions that may involve both the Planning and Execution processes.

Execution Plan. An execution plan is a pair $\langle \sigma, cond \rangle$, where the first parameter is an optimal trace to execute and the second one represents global contextual conditions to apply on σ. For instance, this offers a way to check whether the depot is reached when the working-day is overpassed, e.g. $t \geq 12h \implies \ell = dep$. The execution of the trace σ consists in requesting the performance of each action in σ. $cond$ allows the Mental process to constrain the execution of σ, which is possible since the action performances are monitored by taking care of the robot contextual evolution.

Furthermore, we mention the notion of execution state of the Execution process: $[\sigma] = \langle \sigma_{todo},\ cur,\ \sigma_{done},\ \sigma_{failed},\ \sigma_{aborted} \rangle$, where σ_{todo} represents the transitions of σ whose action are not yet performed, cur represents the transition which action is in progress, whereas σ_{done}, σ_{failed} and $\sigma_{aborted}$ are respectively the successful transitions, the failed transitions and the ones aborted. Initially to any trace σ to execute, the execution state is $\langle \sigma,\ first(\sigma),\ \emptyset,\ \emptyset,\ \emptyset \rangle$.

Action Constraints. The performance of each action in the trace is contextually constrained by the checking of predicates associated with the action. In the paper, we distinguish between pre-, invariant and post- conditions, which mean spatio-temporal conditions that respectively must hold before, during and at the end of the action. Actions are not only constrained by predicates put on them directly, but also by constraints inherited from the intentions they refer to or more generally from the Mental process on the trace. We assume that $cond$ for σ is expressed as 3 predicates $pre(\sigma)$, $inv(\sigma)$ and $post(\sigma)$. Next, all the action constraints are specified at the action level and with respect to any action a mentioned in σ, knowing that a contributes to the execution of the intention i (we call that i is behind a). We denote $\sigma(i)$ the subset of the transitions in σ whose actions are behind the intention i and more generally, we use the notation $end(\sigma)$ already presented to specify the set of intentions that are behind the transitions of σ. Observe that the behind relation is properly defined from the CPS semantical considerations, making transitions with respect to the intention plans. The following formulas express the symbolic action constraints formally:

- $pre_\sigma(a) := pre(a) \wedge pre\,(i \mid a = first(\sigma(i)).act) \wedge pre\,(\sigma \mid a = first(\sigma).act)$
- $post_\sigma(a) := post(a) \wedge post\,(i \mid a = last(\sigma(i)).act) \wedge post\,(\sigma \mid a = last(\sigma).act)$
- $inv_\sigma(a) := inv(a) \wedge inv\,(i \mid a = tr.act,\ tr \in \sigma(i)) \wedge inv\,(\sigma \mid a = tr.act,\ tr \in \sigma)) \wedge\ inv(\sigma \rhd a)$

Temporal Control of Actions. In order to enforce the monitoring of the Action process, we define a finite deadline for each action in σ, over which the action is considered as failed. This is why we augment $inv_\sigma(a)$ with the last conjunct

term $inv(\sigma \rhd a)$. So, for each current transition cur extracted from σ_{todo}, this is a highly dynamical constraint put on the action $cur.act$ which is computed from the learned duration $\delta(cur) = \delta(cur.act, cur.src.ctx)$ and an extra time called $extra_{\sigma_{todo}}(cur)$. We exploit the expected duration of σ_{todo}, denoted $\delta(\sigma_{todo})$, to share its value over the transitions in σ_{todo}. A smart sharing is proposed here, as a hybrid proportion of the learned duration of a transition and the relative relevance of the intention behind this transition. So, the bigger these values, the larger the transition deadline.

The following formula expresses the contextual deadline value of any transition tr in σ, knowing the starting time t of this transition and the fact that the intention i is behind tr. A reasonably monitoring assumption is that there must be the expression of a formal finite deadline $deadline(\sigma)$ declared in $cond$, in contrary the default $deadline(\sigma)$ would be considered as infinite, inducing no deadline whatever the transitions in σ.

$$deadline_\sigma(tr) := tr.src.ctx.t + \delta(tr) + extra_\sigma(tr)$$
$$extra_\sigma(tr) := \left(\frac{\delta(tr)}{\delta(\sigma)} + \frac{W_r(i)}{\sum_i^{I_\sigma} W_r(i)}\right)/2 * \left(deadline(\sigma) - \left(tr.src.ctx.t + \delta(\sigma)\right)\right) \mid a = tr.act,\ tr \in \sigma(i)$$

Algorithm 1 $execute(\sigma)$

```
1:  I_cur := end(σ)
2:  σ_todo := σ
3:  σ_done := σ_failed := σ_aborted := ∅
4:  while σ_todo ≠ ∅ do
5:     cur := pop(σ_todo)
6:     a := cur.act
7:     ctx := getCtx()    /* from Context proc. */
8:     if pre_σ(a) ⊭ ctx then
9:        σ_failed := σ_failed ∪ {cur}
10:       I_cur := I_cur \ {i} s.t. cur ∈ σ(i)
11:    else
12:       inv_σ(a) := inv_σ(a) ∧ inv(σ ▷ a)
13:       o := launch(a)    /* send a to Action proc. */
14:       if o = fail ∨ (¬inv_σ(a) ∧ ¬post_σ(a)) then
15:          σ_failed := σ_failed ∪ {cur}
16:          I_cur := I_cur \ {i} s.t. cur ∈ σ(i)
17:       else
18:          σ_done := σ_done ∪ {cur}
19:          if σ_todo(i) = ∅ s.t. cur ∈ σ(i) then
20:             I_cur := I_cur \ {i} s.t. cur ∈ σ(i)
21:          end if
22:       end if
23:    end if
24:    σ' := update(σ_todo, I_cur, ctx)
25:    if σ' = ∅ then
26:       σ' := relevant_reduce(σ_todo, I_cur, ctx)
27:    end if
28:    σ_aborted := σ_aborted ∪ (σ_todo \ σ')
29:    σ_todo := σ'
30: end while
```

Handling Symbolic Action and Failures. The algorithm 1 develop the execution of a trace σ. Since the transitions of σ_{todo} are scheduled and considered one after one, it only iterates on the first of them, further called the current transition cur. This transition is removed from σ_{todo} (line 5) and is added to the end of σ_{done}, provided to be executed successfully by the Action process (line 18). More precisely, when $pre_\sigma(a)$ holds true for the associated action, the Action process is requested to launch it, under the predicates $post_\sigma(a)$ and the dynamical invariant $inv_\sigma(a)$ attached to a. Through this algorithm, the Execution process monitors 3 kinds of action failures:

- **Logical failure.** This means that the action prerequisites does not hold (line 8),
- **Effective failure.** This occurs when the Action process launches the action and cannot achieve its objectives (often, for low-level reasons [6]), hence the worst case corresponds to the situation where the action post-condition remains false up to obtain a false value for the invariant (line 14). We also assume that the Action process possibly reacts earlier and informs the Execution process in real time, triggering the failure of the action,
- **Controlled failure.** This specifies that the invariant condition does not hold, and this includes a negative checking of the dynamical constraint $inv(\sigma \rhd a)$ (line 14).

As the Execution process is informed by the Action process in real time, it can react and decide about the achievement of the action. A corner point of the algorithm is the updating of the set I_{cur} of the intention not yet achieved. In fact, a transition in σ_{todo} which does not have an intention of I_{cur} behind it, cannot be executed. In case the performance of some action fails, the corresponding transition joins σ_{failed} (line 15) and this could have the consequence to abort some other transitions (line 28). In fact, in different parts of our algorithm, the set I_{cur} can be reduced, in particular in the cases of failure (lines 10 and 16).

Algorithm 2 $update(\sigma_{todo},\ I_{cur},\ ctx) : \sigma'$

1: $\sigma' := \emptyset$
2: $ctx_{cur} := ctx$
3: **while** $tr := pop(\sigma_{todo}) \neq \emptyset$ **do**
4: **if** $tr \in \sigma(i) \mid i \notin I_{cur}$ **then**
5: **continue**
6: **end if**
7: **if** $pre_{\sigma}(tr.act) \not\models ctx_{cur}$ **then**
8: **return** $\emptyset$
9: **end if**
10: $tr.dest.ctx.t := ctx_{cur}.t + \delta(a,\ ctx_{cur}.t)$
/* from Learning process */
11: **if** $tr.act = move(\ell')$ **then**
12: $tr.dest.ctx.\ell := \ell'$
13: **else**
14: $tr.dest.ctx.\ell := ctx_{cur}.\ell$
15: **end if**
16: $add(\sigma',\ tr)$
17: $ctx_{cur} := tr.dest.ctx$
18: **end while**
19: **return** σ'

Contextual Replanning of Execution. At the end of each iteration in the execute algorithm, thus after the performing of the action associated with the current transition, the update algorithm 2 is called in order to update σ_{todo}. All the transitions which do not have an intention of I_{cur} behind them, are simply ignored and aborted (ligne 5). For the other transitions, the expected starting context is iteratively updated. An empty set is returned whether the precondition of the action associated to the transition tr does not prevail. In this case, I_{cur} is reduced, causing the abortion of the transitions of $\sigma(i)$, such that i is behind tr.

When the update function fails to return an updated trace σ_{todo}, by returning an empty set (line 8), a brute force approach would consist to return a fail outcome to the Mental process, in order to compute a new execution plan from scratch, despite the space and time complexity of a CPS building. In contrast, we consider that some transitions can be removed from σ_{todo}, provided to maintain a maximum relevance quality for the resulting reduced σ_{todo}. In this case, a call to the *relevant_reduce* function is required, the behavior of which is represented by the algorithm 3. The key point of this function consists in studying every non-empty and strict subset of the set I_{cur}, defined as the set $CONF$.

Definition 4.1. *Let $CONF$ be an ordered set built from the order relation $\succ$ over the configurations such that $\forall\ conf_1, conf_2 \in CONF,\ conf_1 \succ conf_2 \Leftrightarrow R_{conf_1} > R_{conf_2}$.*

Algorithm 3 $relevant\_reduce(\sigma_{todo},\ I_{cur},\ ctx) : \sigma$

1: $\overline{R_{best}} := 0$
2: $\sigma_{best} := \emptyset \qquad \sigma' := \emptyset$
3: $CONF := 2^{I_{cur}} \setminus \{\emptyset,\ I_{cur}\}$
4: **while** $conf := pop(CONF) \neq \emptyset$ **do**
5: **if** $\overline{R_{conf}} < \overline{R_{best}}$ **then**
6: **break** /* Relevance optimization */
7: **end if**
8: **if** $\sigma' := update(\sigma_{todo}, conf, ctx) = \emptyset$ **then**
9: **continue**
10: **end if**
11: **if** $(\overline{R_{best}} = 0) \vee (\delta(\sigma') < \delta(\sigma_{best}))$ **then**
12: $\sigma_{best} := \sigma'$
13: **end if**
14: $\overline{R_{best}} := \overline{R_{conf}}$
15: **end while**
16: **return** σ_{best}

In Algorithm 3, some intentions of I_{cur} are necessarily no more represented. Then, an update of σ_{todo} is realized with respect to such a subset of intentions (line 8) and this causes the removal of transitions which have these intentions behind, from σ_{todo}. In fact, a reduced trace σ' could be the best reduced σ_{todo} (line 16). In order to introduce the relevance criterion, the algorithm takes advantage of the capacity of (partially) ordering the subsets of I_{cur} according to the relevance weights of the intentions. A normalized value $\overline{R_{conf}}$ is assigned to each subset $conf \in CONF$, so that the various subsets of I_{cur} are considered in the $\overline{R}$ decreasing order (line 3). As a consequence, as soon as the first feasible trace is computed from the update function (line 8), we know that this trace is one of the best relevant traces. In fact, when there are distinct sub-traces that have the same normalized relevance value, we again propose to select the trace having the shortest duration as the optimal trace (line 11).

Complexity Discussion. The complexity of our development mainly depends on the complexity of the functions *update* and *relevant_reduce*. The first one is linear in the size of σ ($|\sigma|$) whereas the second depends on the size of

the set $CONF$, with a worst complexity of $2^{|end(\sigma)|}$. This is not a real drawback when there are few intentions beyond σ. Moreover, we can estimate that this bound is rarely reached due to the ordering exploitation of $CONF$.

Application to the Scenario. We consider that the Mental process has selected an execution plan corresponding to the main relevant trace in Figure 2, namely σ, such that $\delta(\sigma) = 3h22$, and $last(\sigma).dest.ctx.t = 11h22$. We assume now that the global deadline condition specified by the Mental process is $deadline(\sigma) = 12h00$, inducing a global extra time over σ of $extra(\sigma) = 0h38$. For the Execution process, the first (current) transition to deal with is the first transition of σ, $cur = first(\sigma)$ such that $cur.ctx = (dep, 8h)$, $cur.act = move(ph2)$ and the expected duration for the action is $\delta(cur) = 0h46$. By applying the $extra$ formula, the computed extra time is $extra_\sigma(cur) = 7min$. The performance of the action $move(ph2)$ requested to the Action process is expected to work under the invariant attached to $move(ph2)$, including the conjunct $(ctx.t < 8h53)$.

Let us now consider that there is no specification of deadline in the execution plan. In this case, there is no dynamical control for the performances of actions. According to this situation, we assume that the $update$ function that runs after the achievement of the action $move(ph2)$ at state s_1 detects that the prerequisite of $move(dep)$ mentioned at the state s_4 does not match with the reached context. This triggers the function $relevant\_reduce$. Because the intention behind $move(dep)$ has a higher relevant value against the other intentions, the $relevant\_reduce$ function will rather remove the intention $Order1$ in order to maintain the intention $Back$.

5. Conclusion and Discussion

Dedicated to Autonomous vehicles, the E-HoA multi-process software architecture allows a BDI agent to build its execution plan contextually and soundly react to unexpected situations. In this paper, we proposed to deal with relevance weights bound to the selected intentions. Due to a nice coding of the intentions behind each execution plan, we show how to efficiently compare the respective relevance of feasible traces proposed by the Planning process. However, since this corresponds to defining a partial order on the feasible execution plans, the expected durations of actions in traces are also exploited as a second criteria to define an optimal execution plan.

Facing unexpected events, this coding is also exploited by the execution process to contextually monitor the physical performance of the execution plan. Distinguishing between logical, effective and controlled failures of guarded actions, the execution plan is reduced as the contextual failures occur, while maintaining the most relevant intentions as much as possible. The fact to maintain as much as possible the feasible and relevant part of an execution plan is an interesting approach from a practical point of view [10]. It is also an efficient alternative to a replanning of actions from scratch, since the Planning process could suffer from combinatorial explosion problems due to the interlacing of the possible actions.

Among the immediate perspectives, we aim at improving the proposed framework to better handle dynamic changes in the execution plan. For instance, the suppression of transitions in case of failures could be advantageously exploited to inject some sub-plan within the execution trace. Another concern is the capability to maintain an urgent action or plan despite some unexpected failure.

References

[1] Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (Eds.), 2009. Multi-Agent Programming. Springer.
[2] Cashmore, M., Coles, A., Cserna, B., Karpas, E., Magazzeni, D., Ruml, W., 2019. Replanning for situated robots, in: ICAPS, pp. 665–673.
[3] Chaouche, A.C., El Fallah Seghrouchni, A., Ilié, J.M., Saïdouni, D.E., 2014. A Higher-order Agent Model with Contextual Management for Ambient Systems, in: TCCI XVI. volume 8780 of *LNCS*, pp. 146–169.
[4] Chaouche, A.C., El Fallah Seghrouchni, A., Ilié, J.M., Saïdouni, E.D., 2016. Learning from situated experiences for a contextual planning guidance. Journal of Ambient Intelligence and Humanized Computing 7, 555–566.
[5] El Fallah Seghrouchni, A., Ishikawa, F., Hérault, L., Tokuda, H. (Eds.), 2016. Enablers for Smart Cities. John Wiley & Sons, Inc.
[6] Ilié, J.M., Chaouche, A.C., Pêcheux, F., 2020. E-HoA: A Distributed Layered Architecture for Context-aware Autonomous Vehicles, in: Int. Conf. ANT'20, Elsevier. p. 530–538.
[7] Kortenkamp, D., Simmons, R., Brugali, D., 2016. Robotic Systems Architectures and Programming. Springer. pp. 283–306.
[8] Tapia, D.I., Abraham, A., Corchado, J.M., Alonso, R.S., 2010. Agents and ambient intelligence: case studies. JAIHC 1, 85–93.
[9] Touazi, F., Cayrol, C., Dubois, D., 2015. Possibilistic reasoning with partially ordered beliefs. Journal of Applied Logic 13, 770 – 798.
[10] Zhao, Galland, Knapen, Bellemans, Yasar, 2018. Agent-based dynamic rescheduling of daily activities, in: Int. Conf. ANT'18, pp. 979 – 984.