Interleaving Temporal Planning and Execution: IXTET-EXEC*

Solange Lemai and **Félix Ingrand**†
LAAS/CNRS,
7 Avenue du Colonel Roche, F-31077 Toulouse Cedex 04, France
{*slemai,felix*}@*laas.fr*

Abstract

Execution control of plans is a very active domain of research, but remains a major challenge when performed on board real autonomous systems such as robots or satellites. In such a context, where execution concurrency, resources contention and environment dynamic characterize the domain, the use of a temporal planner and a temporal execution control system is desirable. This paper presents IXTET-EXEC, a recent extension of the temporal planner IXTET which allows execution control, plan repair, and replanning when necessary. We present how IXTET-EXEC is embedded in the LAAS architecture and how it interfaces with the other components. We detail the various internal algorithms used by the system, and we illustrate the current implementation with a short example. We conclude with a list of open issues and some ideas on how we plan to address them.

Introduction

Execution control of plans is a very active domain of research, but remains a major challenge when performed on board real autonomous systems such as robots or satellites. Indeed both planning process and execution control process need then to be well integrated and must interleave their activities to perform the mission they are in charge of, while respecting the real-time constraints of the environment. In most cases, embedding such planning and execution control capabilities requires the use of systems which explicitly represent and reason about time.

In this paper we present an extension to the IXTET temporal planner, named IXTET-EXEC, which specifically addresses this problem. IXTET-EXEC enables the system to execute and monitor the execution of a temporal flexible plan. It takes into account runtime failures and timeouts from the underlying functional components and incorporates those failures in the plan. If some flexibility was left for the failed action, it may try another way to achieve it, otherwise it tries to repair the plan to still achieve the goal, and if this fails too, it replans the whole plan. Of course, we are well aware that such a local repair approach may not work in all domains. Still, we believe that there are applications for which the dynamic of the environment allows for repair and replanning while executing a "locked" part of the plan. In any case, if the situation becomes critical, and one does not have enough time remaining to plan after an unexpected failure, one can always rely on predefined emergency plans and procedures to put the system in a safe state.

A number of studies have already addressed similar issues of how to interleave planning and execution control. In [Despouys & Ingrand 1999], the authors proposed an extension to a procedural executive to anticipate the coming choices and search which branch may lead to the success of the following execution. Nevertheless, it remained an extension of a procedural executive, instead of an execution control of a temporal plan. The Rogue/Prodigy system presented in [Haigh & Veloso 1998] is also interleaving planning and execution control in a robotics environment, however, this approach does not explicitly represent time nor resource (although it addresses other interesting issues such as learning). CPEF (Continuous Planning and Execution Framework [Myers 1999]) is another system using an agent based organization to address this problem, still it seems to be more adapted to domains such as military campaigns than the control of autonomous robots or satellites. In [McCarthy & Pollack 2002], the authors present the execution control part of the autominder system. Although the paper says little about the planning process itself, the paper presents interesting ideas on how to repair and locally replan when failures occur. The ASPEN/CASPER approach proposed by [Chien *et al.* 2000], provides an interesting framework to perform continuous planning interleaved with execution. The planner receives and propagates states, resources and temporal updates. Potential future conflicts are incrementally resolved by performing iterative repair techniques. During replanning, and depending on the domain application, nothing is executed or a portion of the old plan is executed. In that case, a commitment window, corresponding to the replanning time interval forbids the modification of committed activities. However, this approach does not handle conflicts which appear within the commitment window. Another approach which seamlessly integrates temporal planning and execution control is IDEA [Muscettola *et al.* 2002]. This approach relies on two main ideas: (1) most components can be seen as agents which share a com-

*Part of this work was funded by a contract with CNES and ASTRIUM.

†This author is currently on sabbatical at NASA Ames Research Center, Moffett Field, CA, USA.

mon virtual machine defining their reactive planning behavior (planning here has to be taken in a wide sense) (2) all these agents share parts of a global temporal model which specifies the internal "behavior" of the agent, as well as the communication between agents. Although it appears to be a very promising approach, it does not yet provide a framework for resources management.

The paper is organized as follows. The first section presents the general organization of the system as well as IxTeT the pure planning part of IxTeT-eXeC. It also presents the procedural executive with which it interfaces. Section 2 describes the IxTeT-eXeC component and details the methods and algorithms used to perform temporal plan execution monitoring, plan repair and replanning. We illustrate the current state of implementation with an example, and conclude with a number of extensions currently being implemented.

General Organization

This section presents an overview of the general organization of the system. We introduce the overall architecture in which our system is intended to be used, as well as the preexisting IxTeT planning system and the Propice procedural executive.

The LAAS Architecture

One of the main reasons to extend IxTeT with execution, dynamic planning and replanning capabilities is to use it on board complex autonomous systems such as autonomous robots or satellites. Those systems are usually developed and deployed using a particular architecture and its associated tools. In our case, we used the LAAS architecture to integrate IxTeT-eXeC and this section describes this architecture as well as how these different components relate to each other.

The LAAS architecture [Alami *et al.* 1998] was originally designed for autonomous mobile robots. This architecture remains fairly general and is supported by a consistently integrated set of tools and methodology, in order to properly design, easily integrate, test and validate a complex autonomous system. As shown on Figure 1, it has a number of levels, with different temporal constraints and uses different data representations. Proper tools have been developed to meet these specifications and to implement each level of the architecture. The levels are:

- *The decision level:* This higher level includes the deliberative capabilities of the agent such as: producing task plans, recognizing situations, faults detections, etc. In our case it embeds:
  - a procedural executive (PRS/Propice [Ingrand *et al.* 1996]), which is connected to the underlying level, to which it sends requests that will ultimately initiate sensors/effectors actions and start processing tasks. It is responsible for controlling actions and procedures execution while being at the same time reactive to events from the underlying level and commands from the operator. The temporal properties of this executive are to guarantee its execution reaction time.

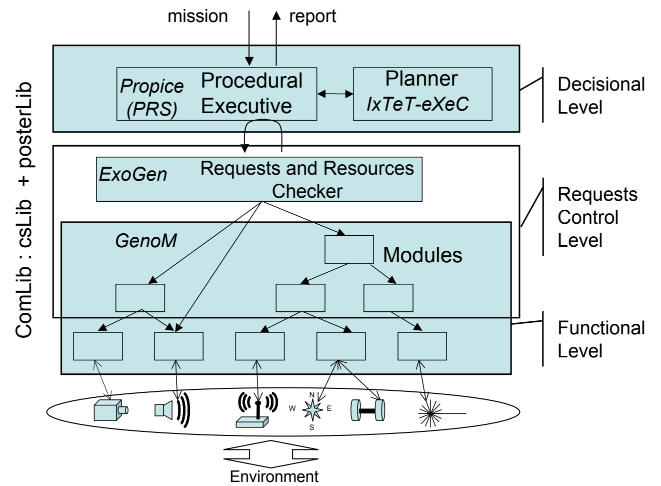

Figure 1: The LAAS Architecture.

  - a temporal planner/executive (in our case IxTeT-eXeC, an extension to IxTeT [Ghallab & Laruelle 1994]) which will be in charge of producing and executing temporal plans. In the context of embedded execution, this planner needs to be reactive and take into account execution failures (action failures or timeout).
  This is the component on which this paper focuses.

- *The functional level:* Located at the lowest level, it includes all the basic built-in robot action and perception capabilities. These processing functions and control loops (image processing, motion control, ...) are encapsulated into controllable communicating modules (developed using GenoM [Fleury, Herrb, & Chatila 1994]). Each module provides a number of services and processing tasks available through requests sent to it. Upon completion or abnormal termination, reports (with status) are sent back to the requester. Note that modules are fully controlled from the decisional level. The temporal requirements of the modules depend on the type of processing they perform. Modules running servo loop (which have to be ran at precise rates and intervals without any lag) will have a higher temporal requirement than a motion planner, or a localization algorithm.

- *The requests control level:* Located in between the two previously presented levels, the Requests and Resources Checker [Ingrand & Py 2002] checks the requests sent to the functional modules (either from the procedural executive, but also internally from the functional level itself), as well as the resources usage. It is synchronous with the functional modules, in the sense that it sees all the requests sent to them, and all the reports coming back from them. It acts as a filter which allows or disallows requests to pass, according to the current state of the system (which is built online from the past requests and past reports) and according to a formal model (given by the user) of allowed and forbidden states of the functional system. When reports of the requests are being sent back to the

R2C, it passes them to the requester, after updating its internal state. The temporal requirements of this level are hard real-time.

This paper focuses on the Decisional Level and how its different components deal with reactive planning and re-planning. In particular, we present a recent extension to IxTeT: IxTeT-eXeC which first enables the system to execute and monitor the execution of a plan produced by IxTeT, and second takes into account run time failures, timeouts, incorporates those failures in the plan, and repairs and/or replans when necessary.

The IxTeT planning system

The IxTeT system is a lifted partial-order temporal planner based on CSPs. The temporal representation describes the world as a set of multi-valued functions of time (piece-wise constant), called $attributes$, and resources over which borrowing, consumption or production can be specified. The planner deals with a set of deterministic planning operators, called $tasks$, which are temporal structures giving partial specifications of the evolution of attributes over the task duration. Using ungrounded operators, the search process explores a tree in the plan space whose root node is a structure similar to a task which specifies the initial situation, goals with different associated dates and expected contingent events across the planning horizon.

Attributes are temporally qualified by the predicate *hold*, which asserts the persistence of an attribute value over an interval, and the predicate *event*, which states an instantaneous change of values. Resources are expressed by the predicates *use*, which represents a borrowing of a quantity of a sharable resource over an interval, *consume* and *produce* which state the consumption or production of a given resource quantity.

A deterministic planning operator, called a *task* (see Figure 3 for an example), is a temporal structure composed of a set of *events* describing the change of the world induced by the task, a set of *hold* assertions on attributes to express required conditions or the protection of some fact between two events, a set of resource usages, and a set of temporal and binding constraints on the different time-points and variables of the task. Durations are expressed as continuous intervals. Note that tasks may also contain a set of sub-tasks, thus allowing a hierarchical definition.

IxTeT relies on two CSP managers. A *time-map manager* implements a Floyd Warshall like propagation schema to ensure the global consistency of the temporal network. A *variable constraint manager* combines 2-consistency and forward checking to handle both discrete and numeric variables. *Domain restriction*, *equality* and *inequality constraints* are propagated. A recent extension has improved the expressiveness of the planner [Trinquart & Ghallab 2001]: the variable manager now handles numeric variables over infinite sets (such as float or integer) and complex numeric constraints. It allows for instance to represent the quantity of resource consumed or produced as a numeric variable. Furthermore, mixed constraints between temporal and atemporal variables can be expressed as for instance: $?distance = ?speed * (t_2 - t_1)$. They are managed by a supervisor that transfers information from one CSP to the other one when required. These CSP managers take part in the elaboration of flexible plans: they compute for each variable a minimal domain which reflects only the necessary constraints in the plan.

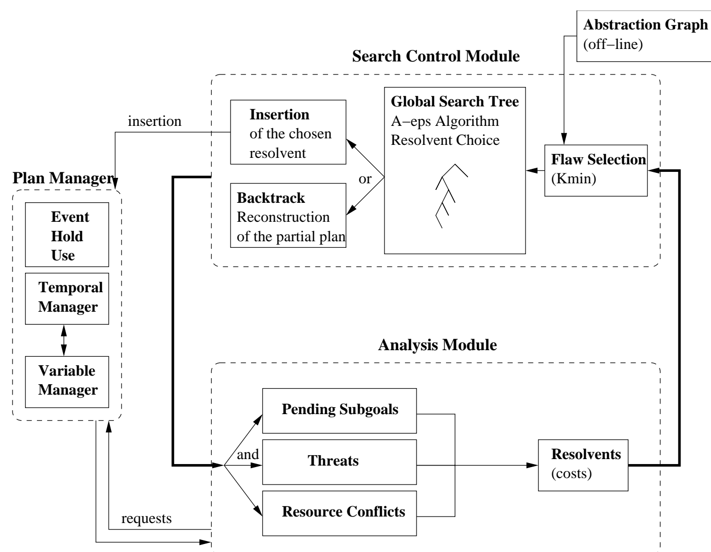

Figure 2: IxTeT modules.

The initial plan is a particular task that describes a problem scenario, that is: initial values of a set of instantiated attributes; expected changes on some contingent attributes; expected availability profile of resources; the goals to be achieved; and a set of temporal constraints between these elements.

Figure 2 presents the organization of the different IxTeT modules. The planning algorithm explores a search tree of partial plans. The root of the tree is the initial plan; branches represent new tasks or constraints inserted into the current plan in order to solve one of its *flaws*. Three kinds of flaws are considered:

– *pending subgoals* are events or assertions that have not yet been established; resolvers for a pending subgoal consist in inserting an establishing event and a causal link (a $hold$ predicate that protects the attribute value); such an event can already be in the plan or it may need the insertion of a new task;

– *threats* are possibly inconsistent events or assertions; when a threat is detected, it can be solved by adding temporal constraints or variable binding constraints;

– *resource conflicts* are detected as over-consuming cliques in a particular graph; resolvers include precedence constraints, variable binding constraints, inequality constraints between resource allocations, or insertion of resource production tasks [Laborie & Ghallab 1995; Trinquart, Lemai, & Cambon 2002].

The control algorithm detects all current flaws in a partial plan; if there is no flaw, a solution is found; otherwise a flaw is selected, possible resolvers for this flaw are listed,

one is chosen; it is inserted into the partial plan on which the algorithm proceeds recursively. IxTeT is complete in the sense that if there exists a solution, it will find it.

However, in a partial plan, the number of flaws to analyze may be very large. At a given level of the planning process, some flaws are more relevant than others. IxTeT uses a hierarchy on the different attribute names (and, as a consequence, on the different flaws) to structure the search space [Garcia & Laborie 1995]. This hierarchy verifies the ordered monotonicity property: at a given abstraction level, the resolution of a flaw creates only new flaws belonging to the current or less abstract levels. This abstraction hierarchy can be automatically generated from the description of the domain by analyzing the conditions and the main effects of the tasks. During the search, the abstraction level is dynamically computed and the flaw analysis is limited to the attributes in the current level.

There are a few limitations to this functional representation and CSP-based approach to planning. One of them is that it performs a feasibility not an optimization search. It is not easy to introduce an optimization criterion over the set of plans. This can be done either through heuristics (in the current version of IxTeT, heuristics are mostly used to reduce the search time) and/or through a postprocessing stage for local plan improvements. The other drawback is that the planner cannot deliver early in the planning process a partial plan on which execution may start running. This is indeed a limitation for interleaving planning and execution. Note however that such interleaving modality can be required for highly reactive systems but it may not be acceptable for a critical system: execution may start on a partial plan that contains a flaw to be discovered later, but backtracking becomes impossible once execution is started.

Nevertheless, the advantages of the CSP-based functional approach are numerous. We already underlined the expressiveness of the representation, its handling of time, resources, controllable and contingent events. The representation also leads to very flexible plans, partially ordered and partially instantiated, that can be further constrained at execution time (this is part of the least commitment strategy). Finally, let us stress that since the planner performs a search in the plan space, it can be adapted to incremental planning and to other plan merging operations. Our approach for interleaving planning and execution relies on these properties.

Execution Control within the Procedural Executive

Although it is not the focus of this paper, it is necessary to describe how IxTeT-eXeC interacts with Propice, the procedural executive, as well as the flexibility Propice has to perform the tasks given by IxTeT-eXeC.

In the decisional level presented above, one can see that the procedural executive is the only component closing the loop with the next lower level, as well as getting missions and high level goals from the user. Indeed, this component is mainly designed to react to events and goals and act upon accordingly (see [Ingrand *et al.* 1996] for a detailed account of these functionalities). Nevertheless, at some point, it may need a new plan to perform a new goal, for which no predefined plan nor procedure are available.

Note that Propice does not directly interpret the plan built by the planner, this particular task is carried out by IxTeT-eXeC. Thus, when Propice requests a new plan to achieve a particular goal, it receives tasks (part of the plan produced) to perform. Such tasks are in fact linked to particular start events of the plan graph IxTeT-eXeC has decided to execute. Upon reception of these tasks, Propice refines them (if necessary) and/or picks up the appropriate procedure to carry them, depending on the current context. In any case, this usually results in requests being sent to the underlying functional modules. Note that at the functional level, faults and problems may arise. However, Propice has some latitudes to recover from these errors, before they get reported to IxTeT-eXeC, as a last resort.

- First, in Propice, all subgoals are automatically reposted until a "complete" failure is reached. By complete failure, we mean that all the applicable procedures (with all possible context bindings) have failed. For example, an action to take an image in a particular direction may fail while using a first camera (presumably faulty), but may succeed using a spare one, providing the action passed to Propice was given with this latitude left (i.e. this particular variable was not under the control of the temporal planner and its executive).
- Second, often procedures are written in such a way that they test at run time what is the best execution path to take according to the context, and may recover from immediate failures.

These local recoveries do not result from an explicit planning process, but merely from good engineering practices (such as flipping a "presumably stuck" switch on and off...) and procedural programming. Nevertheless, they participate to the overall robustness of the approach.

In any case, when all attempts have failed, or when the system fails to achieve the IxTeT task in the time interval given by IxTeT-eXeC, then Propice reports the failure (or the timeout) and the planner can then starts its own recovery.

IxTeT-eXeC

As stressed before, the planning system IxTeT presents interesting properties in the context of plan execution and plan modification: it elaborates very flexible plans and performs the search in the plan space. Some adaptations have already been made to perform plan merging operations [Gaborit 1996] and incremental planning, integrated with a procedural executive [Gout, Fleury, & Schindler 1999] but limited to the insertion of new goals. The purpose of IxTeT-eXeC is to extend IxTeT (which is only a planner and a priori knows nothing about execution) to interleave more closely planning and temporal execution, especially to:

- regularly update the plan under execution,
- reactively replan in case of failure,
- incrementally replan upon arrival of new goals.

The key component in IxTeT-eXeC is a temporal executive which interacts with the planning system. The general

schema of execution is the following. First, given a description of the task operators and of an initial plan containing the initial situation and the goals, a complete plan is elaborated. This plan is then executed. At each step of the execution, the temporal executive selects the appropriate timepoints from the temporal network, sends the corresponding commands to the procedural executive for task expansion and integrates the reports sent in return. In case of failure, the temporal executive invalidates the part of the plan concerned by the failure. Then, taking advantage of the temporal flexibility of the plan and using IxTeT procedures, it tries to repair the plan while continuing the execution of its valid part. If this plan repair fails, the temporal executive aborts the execution, abandons the current plan and restarts a complete planning process from the new situation and the not yet achieved goals.

This section details the algorithms implemented to achieve important and specific functionalities of our system: concurrent plan execution and repair.

Temporal execution

The temporal executive controls the temporal network of the plan to decide the execution of tasks and maps the abstract timepoints to their real execution time.

As said before, a recent extension of IxTeT allows the definition of mixed constraints between temporal and atemporal variables. If the model description does not contain any mixed constraint, the temporal network is an STN [Dechter, Meiri, & Pearl 1989]. The propagation algorithm used during the planning process is equivalent to a PC1 and leads to the *minimal* network (the edge constraints are minimal with respect to the intersection). During plan execution, the consistent assignment of an execution time to a timepoint is equivalent to adding a constraint between the origin and the timepoint. The insertion of a constraint between two timepoints is propagated to all "triangles" containing at least one of these timepoints. This propagation keeps the STN minimal and guarantees that a complete execution is possible. Note that we did not use some more efficient execution algorithm like the one proposed in [Muscettola, Morris, & Tsamardinos 1998] and based on a filtering of nondominating edges and local propagation in the filtered network, since we need to keep all temporal information in case the plan is modified during execution.

If the model contains mixed constraints, the temporal network may become a TCSP [Dechter, Meiri, & Pearl 1989] containing disjunctive constraints. In that case, the path consistency algorithms are not guaranteed to compute the minimal network. Up to now, we have restricted our approach to plans without mixed constraints, see the Conclusion section for possible extensions.

The timepoints of an IxTeT plan correspond to different types of event: start or end of a task, some contingent external event (as for instance the timepoints defining a visibility window for a space application), or some internal event of a task (used to represent the variation of a resource profile, or some more complex dependency between tasks . . .). The current implementation only takes into account start and end timepoints. But we are considering a future extension which

Execution Cycle
$ExecutedPlan$: plan currently under execution
$ExecutableTPs$: set of executable timepoints
t_{exec}: execution time of the next executable timepoint
$ExecTPs$: set of timepoints to execute during the cycle

1.cycle forever
2. wake up if (current_time $\leq t_{exec}$) or ($replan$)
   or ($MsgQueue$ not empty)
3. $cycle_start_time$ ←current_time
4. $cycle_end_time \leftarrow cycle_start_time + timestep$
5. Sense()
6. PlanRepair()
7. Act()
8 get next t_{exec}
9. add executable TPs occuring at t_{exec} to $ExecTPs$
10.end cycle

Sense()
1.if ($MsgQueue$ not empty)
2. for each Msg
3. if ($report$ is nominal)
4. set_execution_time(cycle_start_time)
5. forget_the_ past()
6. if ($report$ is failed)
   // partial invalidation of ExecutedPlan
7. if (cycle_start_time ≥ timepoint_lower_bound)
8. set_execution_time(cycle_start_time)
9. else
10. create_new_timepoint()
11. insert_new_state()
12. remove_cls_on_common_attributes()
13. forget_the_ past()
14. $replan$ ←true
15. update $ExecutableTPs$, $ExecTPs$
16. $NewSearchTree$ ←true

would allow the execution of other timepoints, to check the occurrence of an external event for instance.

Furthermore, three types of tasks are considered. *Non preemptive* tasks cannot be terminated by the controller and the end timepoint is uncontrollable. *Early preemptive* and *late preemptive* tasks can be terminated by the controller, as soon as possible in the first case, as late as possible otherwise. Note that IxTeT is not able to handle non controllable temporal variables and contingent durations can be squeezed during propagation. IxTeT-eXeC can only detect when an uncontrollable timepoint times out. Further work needs to be done to make the time-map manager verify the Dynamic Controllability property described in [Morris, Muscettola, & Vidal 2001].

The algorithms **Execution Cycle**, **Sense()**, **PlanRepair()** and **Act()** present how IxTeT-eXeC is implemented. After the elaboration of a complete plan by IxTeT, its execution is started. Each time the executive needs to do something, i.e. a message has been received, or it is time to execute some timepoint, or some plan repair process is in progress, it wakes up and follows the execution cycle described above.

```
PlanRepair()
1.if (replan)
2.  if (NewSearchTree)
3.     set_searchtree_root(ExecutedPlan)
4.  NewSearchTree ← false
5.  get limit_time
6.  while (!solution_found) and
    (current_time ≤ limit_time)
7.     solution_found
              ←plan_one_step(cycle_end_time)
8.  if (solution_found)
9.     ExecutedPlan ← Solution Plan
10.    replan ← false
11. else
12.    ExecutedPlan ← get_best_partial_plan()
13. update ExecutableTPs, ExecTPs
```

```
Act()
1.while (ExecTPs not empty) and (!end_cycle)
   and (!timeout)
2.  (ExecTP,exec_time) ← get_first_TP(ExecTPs)
3.  ExecuteNow ←true
    //ExecTPs can contain timepoints to execute
    //in another cycle (if wake up for replan or Msg)
4.  if(exec_time > cycle_end_time)
5.     end_cycle ←true
6.  else
7.     if (replan)
8.        if (ExecTP is start TP)
9.           flaw ←check_starting_task()
10.          if (flaw)
11.             ExecuteNow ←false
12.             if (ExecTP_ub > cycle_end_time)
13.                add ExecTP to WaitingExecTPs
14.                suppress ExecTP from ExecTPs
15.             else timeout ←true
16.    if (ExecuteNow)
17.       if (exec_time ≤ cycle_start_time)
18.          if (ExecTP_ub ≥ cycle_start_time)
19.             exec_time ← cycle_start_time
20.          else timeout ←true
21.       if (ExecTP not controllable and not received)
22.          timeout ←true
23.       else
24.          if (ExecTP is start TP)
25.             set_execution_time(exec_time)
26.             forget_the_ past()
27.             NewSearchTree ←true
28.          send_command()
29.          update ExecutableTPs, ExecTPs
30.add WaitingExecTPs to ExecTPs
```

In these algorithms, $ExecutedPlan$ refers to the plan being executed. At the beginning, it corresponds to the flexible plan resulting from the initial complete planning process. A timepoint of its temporal network is *executable* if all timepoints that must directly precede it have already been executed. The temporal executive determines what should be the next timepoint to execute and its execution time (t_{exec}). In fact, several timepoints may have to be executed during one cycle. The set of these timepoints, $ExecTPs$, is initialized with the set of timepoints occuring at t_{exec} and updated during the cycle with the new executable timepoints which have to be executed before the end of the cycle. The execution time of a timepoint depends on its type. It corresponds to the lower bound for start timepoints and end timepoints of early preemptive tasks; and to the upper bound for end timepoints of late preemptive and non preemptive tasks. In the last case, this "execution time" only corresponds to a deadline used to detect possible timeouts.

Two types of commands are sent to the procedural executive: $(START\ TaskId\ parameters)$ or $(END\ TaskId)$ (if the task is preemptive). A task is fully instantiated just before starting its execution. A report is sent back by the procedural executive each time a task is completed, which is mapped into the end timepoint of the task. Note that the real execution time assigned to an end timepoint is the time at which the report message has been received. More precisely a completion report contains the following information: a completion status (nominal or failed), and in case of failure, the actual state. This state is described as the set of new values for the attributes of the task (state and resource attributes). We also plan to integrate a resource report at each relevant completion to update the actual quantities of resource consumed or produced by a task to detect as soon as possible future resource contention. In the future, a message from the procedural executive may also correspond to a new goal to insert in the plan, see the Conclusion for further extensions.

The *sense* part of the cycle integrates the messages from the procedural executive. In the nominal case, it amounts to assigning the current time to the end timepoint of the task and propagating this value in $ExecutedPlan$ (*set_execution_time()*). New executable timepoints may appear, and $ExecTPs$ is updated. For instance, $ExecTPs$ may now contain the start timepoints of parallel tasks immediately following the completed task, that will be executed in the same cycle. The failed case is detailed in the next subsection.

The *act* part of the cycle "executes" the timepoints in $ExecTPs$ according to their precedence constraints. *get_first_TP()* (Algo: **Act()**, line 2) determines which timepoint to handle next and its execution time according to the temporal network (lower/upper-bound). Line 4 checks if the timepoint has to be taken into account during the current cycle. If not, no other timepoint is due during this cycle. Otherwise, the "execution" of the timepoint depends on its type. For a start timepoint: its execution time is assigned the value determined lines 17-20 and propagated (line 25), the corresponding command is sent. For the end timepoint of a preemptive task: the command is sent, but the execution time is set only once the report message is received in the *sense* part of the next cycle. Finally, a timeout is detected if a non preemptive task is not terminated yet, but should be (line 21). Each time a new timepoint is instantiated, $ExecutableTPs$ and $ExecTPs$ are updated (line 29).

The uncertainty on the duration of the execution cycle has

some consequences on the exact execution time of start or end of tasks. The *timestep* (Algo: **Execution Cycle**, line 4) is an estimation of the maximal duration of the cycle. It is defined by the user and may vary with the application. The model description and the planning process are independent of the timestep. But the user has to be aware that two timepoints that have to be executed within an interval less than one timestep, will be executed during the same cycle according to their precedence constraints. Note that the cycle can possibly take less time, and then the executive can react to messages more quickly.

Finally, the temporal execution of a plan can lead to various needs for replan:

1. uncontrollable and controllable timepoints time out,
2. excessive use or insufficient production of resource,
3. new goals to insert,
4. failed tasks.

The adopted strategy consists of two steps: repairing the plan (cases 2, 3 and 4) and executing its valid part while there remains some temporal flexibility; if this fails, aborting the execution and elaborating a new plan. The next subsection details the plan repair process.

Plan Repair

In most cases, failed tasks have not produced the effects initially expected in the plan. The plan repair consists of two steps. First, invalidate the part of the plan depending on these effects. This process includes removing the causal links supplied by the failed task, thus revealing new open conditions in the future. For the moment, the tasks present in the plan are not removed, to limit the amount of decisions. The second step tries to recover the lost properties of the plan by adding new tasks and resolving conflicts.

Partial invalidation of the plan Upon reception of a failure message, two situations may arise. If the reception time is consistent with the bounds of the end timepoint of the task, the task is considered to be finished and its end timepoint is instantiated (Algo: **Sense**, line 8). But the task can fail at any moment and before the minimal expected end time of the task. In that case, a new failure timepoint F is created (Algo: **Sense**, line 10) and set to the current time. It corresponds to the new end timepoint. The other timepoints of the task occuring after F are considered to be failed. Their temporal constraints are relaxed and the temporal propositions ($hold$, $event$, ...) are updated. That is:

- the propositions beginning or occuring at a failed timepoint are removed, as well as the causal links with their establishers,
- in the propositions ending at a failed timepoint, the end timepoint is replaced by F.

Precedence constraints are also added between F and the executable timepoints of the plan.

Next, the new state is inserted (Algo: **Sense**, line 11). The new state is formalized as a set of $events$ on the attributes of the task, occuring at the end timepoint (or at F) and setting the value of the attributes to the new values given by the procedural executive. These events are considered as *explained* and do not need to be established by the planning process. Such an event may or may not be inserted in $ExecutedPlan$. If the plan does not contain any conflicting proposition with the event, it is inserted. If the new value is in conflict with propositions of another running task, it is not inserted. Indeed, we consider, that unless the other task is reported failed, its execution is nominal. Finally, if the new value is in conflict with some propositions of the failed task or some causal links, the event is inserted and the conflicting propositions and causal links are removed. Note that the breaking of causal links does not call the temporal constraints between tasks into question.

At this point, the plan may contain open conditions to reestablish. The repair may require the insertion of new tasks. To allow a task insertion within the current order of tasks, we need to break additional causal links (Algo: **Sense**, line 12). We adapted the work presented in [Gaborit 1996] to determine which causal links to remove. In short:

- search for the common instantiated attributes present both in the plan and in the potentially inserted tasks,
- extract the set of causal links on the common instantiated attributes belonging to the not yet executed part of the plan,
- among these causal links, remove those who do not belong to the *extension* of their establishing event. The *extension* of an event corresponds to the interval during which the value established by the event is imposed (by some assertion of a task ...).

A plan repair is then attempted.

Interleave plan repair and execution The plan repair is similar to the IxTeT plan search process in the plan space. The root of the search tree is the partially invalidated plan $ExecutedPlan$. The search tree is developed according to an Ordered Depth First Search strategy.

Plan repair is distributed, if necessary, on several cycles to allow the concurrent execution of the valid part. During the *plan repair* part of the cycle, planning is done one step at a time until a solution is found or a deadline is reached. This deadline ($limit_time$ on line 5, Algo: **PlanRepair**) corresponds to the share of the timestep allocated to the *sense* and *plan* parts. This parameter is defined by the user. This planning distribution raises two important problems:

1. On which plan relies the execution in the *act* part, especially if no solution has been found? This plan has to satisfy the condition: *The currently running tasks are fully supported in ExecutedPlan*. The plan does not contain any flaw in relation to these tasks. At each planning step, the node is labeled if the corresponding partial plan satisfies the condition. At the end of *PlanRepair* (line 12, Algo: **PlanRepair**) and if the current plan is not acceptable, the last labeled node is chosen and the corresponding plan becomes $ExecutedPlan$.
2. On which plan and which search tree relies the planning process in the next cycle? If no decision has been made meanwhile (no timepoint instantiation, no message reception), the search tree can be kept as is and further

developed during the next *RepairPlan* part. It is even possible to backtrack on decisions made in previous cycles. However, if $ExecutedPlan$ has been modified, a new search tree is mandatory. Its root node is the new $ExecutedPlan$. The planning decisions made in the previous cycles are now fixed, no backtrack is possible.

Some precautions must be taken to prevent from planning in the past. Each new timepoint inserted during the planning process is constrained to occur after $cycle_end_time$. And, to prevent the planner from looking for threats or establishing events in the past, a $forget_the_past()$ function is applied at each timepoint instantiation. So that the sets used for the flaw analysis contain, for each instantiated attribute, only the last event and the assertions occuring after it.

The execution of a partially invalid plan requires to check, before starting a new task, that it is fully supported in $ExecutedPlan$ (Algo: **Act**, line 9). If not, and if the time upper bound of the start timepoint has not been reached, its execution is postponed (Algo: **Act**, line 13). In case of timeout, the execution is failed and a complete replanning process is necessary.

This plan repair process is not guaranteed to find a valid plan everytime (backtrack nodes frozen by execution or temporal constraints too tight to add new tasks ...), but can avoid to abort execution and completely replan at each failure. By invalidating only a part of the plan, the amount of decisions is rather limited and a repaired plan may be found in a few cycles. Plan repair is especially efficient and useful for not temporally over-constrained plans and plans with some parallelism (some sets of tasks can be executed independently). This approach is illustrated with a short example in the next section.

Complete replanning

If a plan repair is not possible, a complete replanning process is mandatory. This problem has not been completely addressed yet, and thus does not appear in the algorithms presented above. The idea is to adapt the approach proposed in [Muscettola *et al.* 1998] ("planning to plan") and consider the planning process as one of the tasks of the plan, in our case a non preemptive task. Thus, the plan on which replanning is started would contain:

- the origin and end horizon timepoints of $ExecutedPlan$,
- the new global state returned by the procedural executive once the execution is completely stopped, associated with the timepoint T set to the reception time,
- a non preemptive task PLAN, with T as start timepoint (and each new timepoint in the plan is constrained to occur after its end timepoint),
- the set of not yet executed goals,
- a new goal requiring the plan to be found.

Open issues remain. One is for instance the detection and abandonment of goals that can not be achieved because of a lack of time.

Example

Let consider a robot with two arms (LH and RH), initially located in $L3$. This robot has to take two objects ($O1$ and $O2$, respectively located in $L1$ and $L2$), and to bring them in $L4$. The robot capabilities are described as a set of four tasks: MOVE from a location to another one, TAKE an object with one of its arms, CARRY the object from a location to another one and PUT the object. The initial state and goals as well as an example of a task description in the IxTeT formalism are illustrated in Figure 3. The CARRY task is early preemptive. It will be terminated as soon as the robot arrives in its final location $?l2$ with the object $?obj$. The proposition (1) asserts that the object is on the robot and the proposition (2) guarantees that the robot is in $?l2$ 1 second before the possible end of the task.

Figure 4 presents the initial plan found by IxTeT-ExeC (bold circles represent start and end timepoints of the tasks, arrows represent the precedence constraints between timepoints). The execution starts and a failure occurs: the robot lets $O2$ fall on the floor at the location $L5$ while it is going to $L1$.

Figure 5 presents the plan repaired by IxTeT-ExeC. The failure occurred at the beginning of the CARRY task, a failure timepoint (23) has been created and timepoints 12 and 13 relaxed. The part of the plan concerned by the invalidation is related only to the attributes representing the position of $O2$. The task PUT(L4,O2,LH) is no more supported, but the task TAKE(O1,L1,RH) remains valid and can be executed. The shaded timepoints represent the tasks added by the plan repair. Note that this repaired plan is not optimal. Since no initial task has been removed (especially MOVE(L1,L4) is no more useful), the plan contains an extra MOVE from $L5$ to $L1$.

As said before, two parameters are defined by the user: the timestep and how much of it is allocated to the plan repair. Their values mainly depend on the size of the plan. In fact, several factors play a part in the duration of the execution cycle, among them: the number of timepoints executed during the cycle, the duration of the propagation in the STN (varies with the number n of timepoints in the plan, the complexity is in $(n^2 + n)$), the duration of the plan invalidation in case of failure and the duration of the most expensive planning step. The simple example above has been run on a SunBlade100. Plan invalidation takes 190ms. Plan repair requires 29 steps and 1 backtrack and is distributed on 2 cycles for a 1s timestep (plan repair: 85%), on 4 cycles for a 600ms timestep (plan repair: 80%). During the other cycles, a 5Hz control rate is achieved.

Conclusion and Prospectives

This paper presents some preliminary results on IxTeT-ExeC, an extension to the IxTeT planning system, which is able to interleave more closely planning and temporal execution control. In particular, it regularly updates the plan under execution, it reactively repairs the plan in case of failure, and it incrementally replans upon arrival of new goals.

We have presented how IxTeT-ExeC is integrated in the overall LAAS architecture, and how it relates to the procedu-

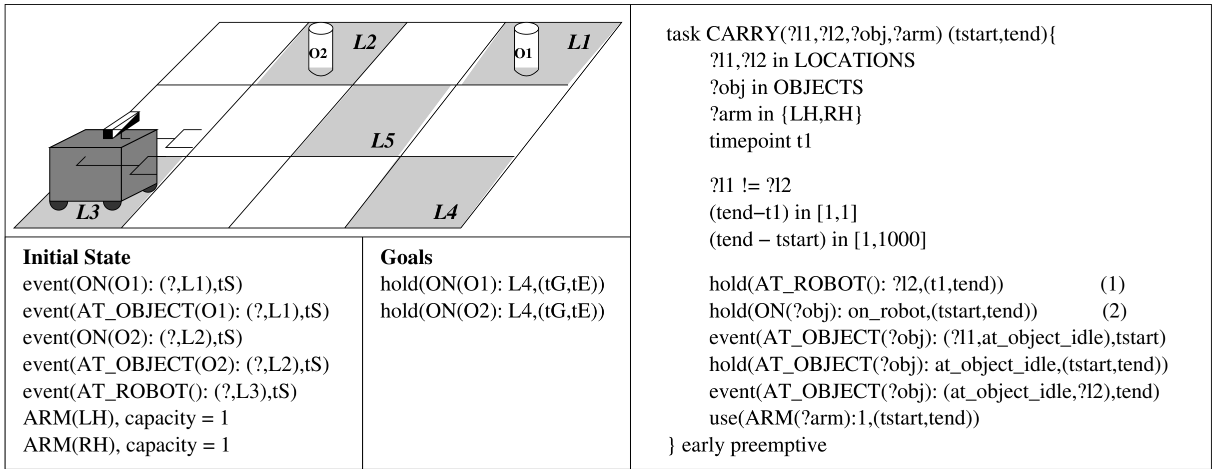

Figure 3: Example of IxTeT formalism.

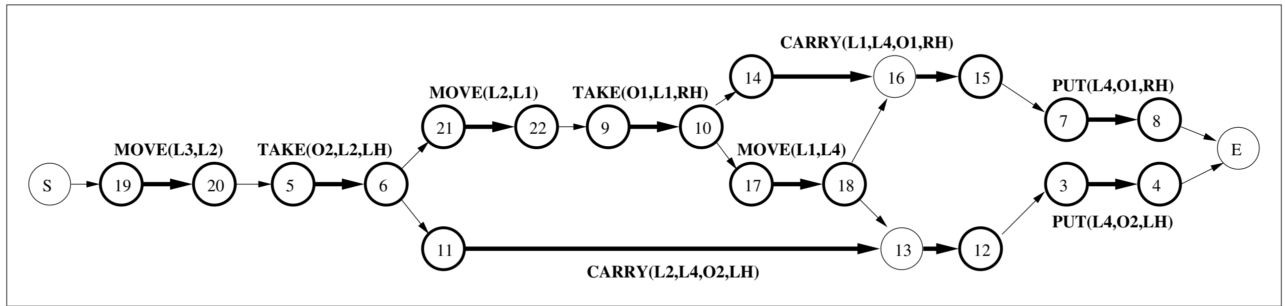

Figure 4: The initial plan.

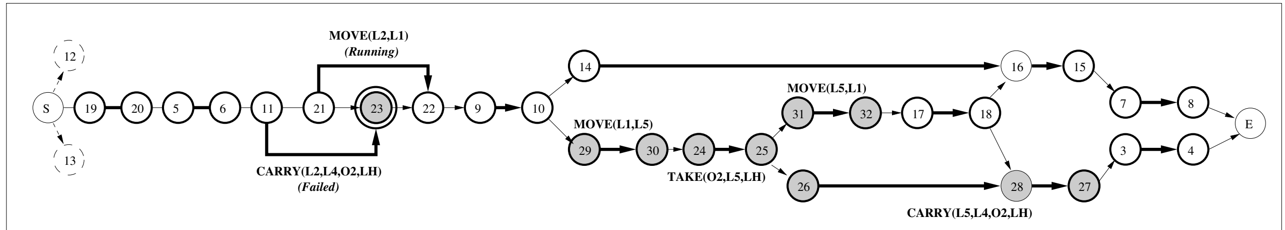

Figure 5: The plan after repair.

ral executive which relays the actions to the functional level and passes back the reports of success or failure of those same actions. The process of plan repair in IxTeT-eXeC allows, to some extent, concurrent planning and execution. It is well adapted for domains where subsystems are rather independent and allow some sets of tasks to be executed in parallel. Moreover, this repair technique is "safe" if the domain is such that no failure is fatal, and can always be recovered from. In any case, for critical situations where the system does not have time to repair or replan, one can always consider using predefined emergency plan or procedure, which can be fired by the procedural executive to put the system in a safe state (safe enough to allow a lengthy replanning from scratch).

The work presented here is still ongoing, and we have already identified a number of desirable features, and in some cases, potential methods and solutions to address them:

- We plan to handle uncontrollable timepoints [Morris, Muscettola, & Vidal 2001].
- One of the main advantages of IxTeT is its handling of production, consumption or borrowing of resources. The quantities can be defined as variables ranging over continuous domains. We aim at exploiting this flexibility to update the actual levels of resource during execution, detect future resource contention, repair if possible (add a

production task ...) or replan (if a resource is no more available).

- The insertion of new goals is quite similar to the plan repair process. A goal is sent by the procedural executive. It is inserted in the plan as a *hold* proposition with the adequate temporal constraints. As for plan invalidation, causal links on common attributes are removed to allow the insertion of new tasks and the plan is "repaired" to satisfy the new open condition.
- We plan to allow the use of mixed constraints and adapt the plan execution in case the time map contains disjunctive constraints. Preliminary ideas would consist in using a PC algorithm and then a backtrack algorithm [Dechter, Meiri, & Pearl 1989] at the end of the planning process to guarantee a minimal network, and adapting the Incremental Directional Path Consistency algorithm [Chleq 1995] by using the *loose intersection* defined in [Schwalb & Dechter 1997] to check consistency when adding a new constraint.

Another important aspect of this work is to embark it and test it on real robotics platforms. Considering that all the other tools and functional modules are currently available on a number of robots at LAAS (Diligent, Dala, etc), and that the development of IxTeT-eXeC is made under Linux (the operating systems used on all these platforms) we do not foresee any particular implementation problem. However, depending on the complexity of the planning task and the dynamic of the environment, we still need to test how well the current implementation will perform on real applications.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming* 17(4):315–337.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Chleq, N. 1995. Efficient algorithms for networks of quantitative temporal constraints. In *CONSTRAINTS-95, pages 40–45*.

Dechter, R.; Meiri, I.; and Pearl, J. 1989. Temporal constraint networks. In *KR'89: Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann.

Despouys, O., and Ingrand, F. 1999. Propice-plan: Toward a unified framework for planning and execution˙ In *Proceedings of the European Conference on Planning (ECP)*.

Fleury, S.; Herrb, M.; and Chatila, R. 1994. Design of a modular architecture for autonomous robot. In *IEEE International Conference on Robotics and Automation*.

Gaborit, P. 1996. Planification distribuée pour la coopération multi-agents. *Thèse de Doctorat, Université Paul Sabatier, Toulouse*.

Garcia, F., and Laborie, P. 1995. Hierarchisation of the search space in temporal planning. In *Proceedings of the European Workshop on Planning (EWSP)*, 235–249.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in Ixtet, a Temporal Planner. In *Proceedings of the International Conference on AI Planning Systems*, 61–67.

Gout, J.; Fleury, S.; and Schindler, H. 1999. A new design approach of software architecture for an autonomous observation satellite. In *Proceedings of iSAIRAS*.

Haigh, K. Z., and Veloso, M. M. 1998. Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots* 5(1):79–95.

Ingrand, F., and Py, F. 2002. An Execution Control System for Autonmous Robots. In *IEEE International Conference on Robotics and Automation*.

Ingrand, F.; Chatila, R.; Alami, R.; and Robert, F. 1996. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation*.

Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1643–1649.

McCarthy, C., and Pollack, M. 2002. A plan-based personalized cognitive orthotic. In *Proceedings of the International Conference on AI Planning Systems*.

Morris, P. H.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. 1998. Remote agent : To boldly go where no ai system has gone before. *Artificial Intelligence* 103.

Muscettola, N.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Principles of Knowledge Representation and Reasoning*, 444–452.

Myers, K. L. 1999. Cpef: Continuous planning and execution framework. *AI Magazine* 20(4):63–69.

Schwalb, E., and Dechter, R. 1997. Processing disjunctions in temporal constraint networks. *Artificial Intelligence* 93:29–61.

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *Proceedings of the European Conference on Planning (ECP)*.

Trinquart, R.; Lemai, S.; and Cambon, S. 2002. One step on the left, one step on the right and back to the middle : Exploring temporal domains in a pop fashion. In *Proceedings of the AIPS Workshop on Temporal Planning*.