When Robots are Late:
Configuration Planning for Multiple Robots with Dynamic Goals

Maurizio Di Rocco[1] and Federico Pecora[1] and Alessandro Saffiotti[1]

Abstract— **Unexpected contingencies in robot execution may induce a cascade of effects, especially when multiple robots are involved. In order to effectively adapt to this, robots need the ability to reason along multiple dimensions at execution time. We propose an approach to closed-loop planning capable of generating *configuration plans*, i.e., action plans for multi-robot systems which specify the causal, temporal, resource and information dependencies between individual sensing, computation, and actuation components. The key feature which enables closed loop performance is that configuration plans are represented as constraint networks, which are shared between the planner and the executor and are continuously updated during execution. We report experiments run both in simulation and on real robots, in which a fault in one robot is compensated through different types of plan modifications at run time.**

I. Introduction

A routine robot delivery is ongoing at Ängen, a senior residential facility in central Sweden. Rout*, an outdoor robot, is bringing some groceries to the door of Sven's apartment, where the indoor robot* Rin *will take them. While* Rout *is en route, its GPS fails; the navigation system of* Rout *is then reconfigured to use laser-based localization. This requires that speed be considerably reduced, which will cause* Rout *to arrive at Sven's apartment ten minutes later. Correspondingly,* Rin*, which had planned to pick up the groceries and then attend to another domestic chore, reschedules its activities to first perform that chore and then meet* Rout*.*

This vignette illustrates two interesting points. First, when facing an unexpected contingency, a robot needs to perform *reasoning* – e.g., to understand the causal consequences of the contingency, to project these consequences forward in time, and to replan its activities accordingly. Reasoning is needed since not all possible situations can be coded in precompiled responses. Second, a single contingency may have complex ramifications, which require reasoning on different types of knowledge. In our example, the GPS failure calls for reasoning about information requirements: how to obtain the self-localization estimate needed by the navigation system. The change in the information source results in a delay, which in turns calls for reasoning about time and time dependencies. When propagated to *Rin*, this delay induces a rescheduling. In other situations this delay may have different consequences: if *Rin* needed the groceries to perform its chore, then this task would not have been anticipated (causal reasoning); if another robot were required at the entrance of Sven's apartment, which can only host one robot, then that robot would have to wait until *Rout* has left (resource reasoning); and so on.

The types of reasoning mentioned above have been widely studied in the field of Artificial Intelligence (AI), namely in the area of planning. In fact, planning in AI was born as an exploration of logical reasoning for robots [1]. In order to help a robot to accomplish tasks in unstructured, everyday environments, an AI planner should possess capabilities that go beyond the ones most commonly addressed in the AI planning community. Namely, it should: (1) possess knowledge and reason about the physical aspects of the domain, like time, space, information requirements and resources; (2) generate plans that enable a sufficient degree of flexibility to accommodate unexpected contingencies and dynamic goals during execution whenever possible; and (3) deal with multiple robots and devices, as well as multiple devices inside each robot, and with their physical and logical dependencies. Some planners exist today that exhibit some of the above features, and we discuss them section VI, but no single system provides all of them. This paper is a first step toward the construction of such a system.

We present a *configuration planner*: a system that generates *configuration plans* for robotic systems that consist of mixed ecologies of robots and devices [2]. Configuration plans are fine-grained plans for robotic systems which specify the causal, temporal, resource and information dependencies between the sensing, computation, and actuation components in one or multiple robots. Configuration planners have been proposed before [3], [4], but they cannot deal with multiple dynamic goals, and lack explicit treatment of time and resources. By contrast, our planner can accommodate time, resources, multiple dynamic goals, and flexible execution. Moreover, our planner reacts to new goals and contingencies by making only minor adjustments to the current plan, when this is possible. This is achieved by: (1) representing a (configuration) plan as a constraint network; (2) defining the configuration planning process as a search in the space of such networks; and (3) sharing the constraint network between the planner and the executor. The first two steps allow for the integration of multiple facets in the planning problem, e.g., time, resources, and information dependencies; the third one allows for flexible execution, plan adjustment and dynamic goal posting.

In the next section we present our representation of plans and plan operators. We then describe the algorithms for plan generation and update, illustrate them on a few examples, compare our planner to related works, and conclude.

[1]Center for Applied Autonomous Sensor Systems, Örebro University, SE-70182 Sweden. {modo, fpa, asaffio}@aass.oru.se

II. Representation

Our approach is grounded on the notion of *state variable*, which models elements of the domain whose state in time is represented by a symbol. State variables, whose domains are discrete sets, represent parts of the real world that are relevant for the configuration planner's decision processes. These include the actuation and sensing capabilities of the robotic systems, and the various aspects of the environment that are meaningful. For instance, a state variable can represent the capabilities of a physical device such as a robot, whose meaningful states might be "navigating", "grasping" and "idle". Similarly, a state variable can represent the interesting states of the environment, e.g., the state of a light which can be "on", "off" or "broken". Let $\mathcal{S}$ be the set of state variables in a given application scenario.

Some devices require resources when they are in given states. We employ the concept of *reusable resource*, i.e., a resource with a limited capacity which is fully available when not required by a device. An example of reusable resource is power: a maximum wattage is available, and devices can simultaneously require power so long as the sum of requirements is less than the maximum power. We denote with $\mathcal{R}$ the set of all resource identifiers. Given a resource $R \in \mathcal{R}$, its capacity is a value $\mathrm{Cap}(R) \in \mathbb{N}$.

Finally, devices in our domain may serve the purpose of providing or requiring certain *information contents*. For instance, a software component may require range data from a laser range finder, and provide localization information. Let the set of all information contents be denoted IC.

A. Representing Configuration Plans and Goals

We employ *activities* to represent predicates on the possible evolution of state variables:

Definition 1: An *activity* a is a tuple $(\mathrm{x}, \mathbf{v}, I, u, \mathrm{In}, \mathrm{Out})$, where

- $\mathrm{x} \in \mathcal{S}$ is a *state variable*;
- $\mathbf{v}$ is a *possible state* of the state variable x;
- $I = [I_s, I_e]$ is a *flexible temporal interval* within which the activity can occur, where $I_s = [l_s, u_s], I_e = [l_e, u_e], l_{s/e}, u_{s/e} \in \mathbb{N}$ represent, respectively, an interval of admissibility of the start and end times of the activity;
- $u : \mathcal{R} \to \mathbb{N}$ specifies the *resources used* by the activity;
- $\mathrm{In} \subseteq \mathrm{IC}$ is a set of *required information contents*;
- $\mathrm{Out} \subseteq \mathrm{IC}$ is a set of *provided information contents*.

For example, the activity

$$\begin{aligned} a_1 \quad = \quad & (\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, \\ & [[10,10][30,50]], \emptyset, \{\mathrm{position}\}, \emptyset) \end{aligned}$$

represents the temporal fact that the navigation functionality MoveFromTo is in the state of being moving from kitchen to living room in a time interval starting at 10 and ending anytime between 30 and 50. This functionality does not use any resource, it needs position information, and it does not produce any information. The activity

$$a_2 = (\mathrm{RobotLocation}, \mathbf{kitchen}, [[0,0],[10,10]], \emptyset, \emptyset, \emptyset)$$

represents a temporal fact about the robot's location.

Henceforth, we indicate with $(\cdot)^{(a)}$ an element of the five-tuple pertaining to activity a. The pair $(\mathrm{x}^{(a)}, \mathbf{v}^{(a)})$ of an activity a asserts a particular state $\mathbf{v}$ of the state variable x; the flexible temporal interval $I^{(a)}$ represents possible temporal intervals of occurrence of the state $\mathbf{v}^{(a)}$ of state variable $\mathrm{x}^{(a)}$. Note also that a pair of activities (a, b) is *possibly concurrent* if $I^{(a)} \cap I^{(b)} \neq \emptyset$. A pair (a, b) of possibly concurrent activities thus indicates that state variables $\mathrm{x}^{(a)}$ and $\mathrm{x}^{(b)}$ can be, respectively, in states $\mathbf{v}^{(a)}$ and $\mathbf{v}^{(b)}$ at the same time.

Unspecified parameters of an activity are indicated with $(\cdot)$ — e.g., $(\mathrm{x}, \cdot, I, u, \mathrm{In}, \mathrm{Out})$ indicates a predicate asserting that state variable x can be in any state during interval I, using resources as indicated by u, etc.

Activities can be bound by *temporal constraints*, which restrict the occurrence in time of the predicates. Temporal constraints can be of two types:

- *Binary temporal* constraints in the form $a\,\mathrm{C}\,b$ prescribe the relative placement in time of activities a, b — these constraints are relations in Allen's Interval Algebra [5], and restrict the possible bounds for the activities' flexible temporal intervals $I^{(a)}$ and $I^{(b)}$;
- *Unary temporal* constraints in the form $\mathrm{C}\,a$ prescribe bounds on the start or end time of an activity a — these constraints are commonly referred to as *release time constraints* and *deadlines*.

Allen's interval relations are the thirteen possible temporal relations between intervals, namely "precedes" (p), "meets" (m), "overlaps" (o), "during" (d), "starts" (s), "finishes" (f), their inverses (e.g., p^{-1}), and "equals" ($\equiv$). For example, the relation $a_2 \mathrm{m} a_1$ represents that the above activity a_2 ends as soon as activity a_1 starts.

When state variables are used to represent a system, the overall temporal evolution of such system is described by a *constraint network*:

Definition 2: A *constraint network* is a pair $(\mathcal{A}, \mathcal{C})$, where $\mathcal{A}$ is a set of *activities* and $\mathcal{C}$ is a set of *constraints* among activities in $\mathcal{A}$.

A constraint network can be used to represent a *configuration plan*. Configuration plans are said to be *feasible* if they are consistent with respect to the resource, state, and temporal requirements. Specifically,

Definition 3: A configuration plan $(\mathcal{A}, \mathcal{C})$ is *feasible* iff:

- the constraint network is *temporally consistent*, i.e., there exists at least one allocation of fixed bounds to intervals such that all temporal constraints are satisfied;
- activities do not over-consume resources, i.e., $\sum_{a \in A} u^{(a)}(R) \leq \mathrm{Cap}(R), \forall R \in \mathcal{R}$, where $A \subseteq \mathcal{A}$ is a set of possibly concurrent activities;
- activities do not prescribe that state variables assume different states in overlapping temporal intervals, i.e., $\mathbf{v}^{(a)} \neq \mathbf{v}^{(b)}, \forall (a, b) \in A \times A : \mathrm{x}^{(\mathrm{a})} = \mathrm{x}^{(\mathrm{b})}$, where $A \subseteq \mathcal{A}$ is a set of possibly concurrent activities.

A *goal* for a configuration planning problem is also represented as a constraint network, therefore expressing temporal, resource, state and information requirements. Typically, a goal (A_g, C_g) is an under-specified configuration plan. Initial conditions are feasible sub-networks of a goal. Maintaining constraints on the configuration plan rather than committing to a specific configuration plan directly enables dynamic goal posting, execution monitoring, and incremental adaptation to contingent events, as we show in Section III.

B. Domain

Given a goal (A_g, C_g) and a configuration plan $(\mathcal{A}, \mathcal{C})$ which contains the goal, the feasibility of the configuration plan is not a sufficient condition for achieving the goal. This is because feasibility does not enforce information and causal requirements. The way these requirements are to be enforced depends on a *domain*:

Definition 4: A *configuration planning problem* is a pair $((A_g, C_g), \mathcal{D})$, where (A_g, C_g) is a goal constraint network, and $\mathcal{D}$ is a *domain*. The domain is a collection of *operators*, which describe the information and causal dependencies between activities.

Definition 5: An *operator* is a pair $(a, (A, C))$ where

- $a = (\mathrm{x}, \mathbf{v}, \cdot, \cdot, \cdot, \mathrm{Out})$ is the *head* of the operator;
- $A = A_p \cup A_e \cup \{a\}$ is a set of activities, where
  - A_p is a set of *preconditions*, i.e., requirements, in terms of state variable values, information input, and resource usage, needed to achieve the state $\mathbf{v}^{(a)}$ of state variable $\mathrm{x}^{(a)}$ and to produce $\mathrm{Out}^{(a)}$;
  - A_e is a set of *effects*, i.e., state variable values entailed by the achievement of state $\mathbf{v}^{(a)}$ of state variable $\mathrm{x}^{(a)}$;
- C is a set of temporal constraints among activities in A.

Computing a configuration plan consists in selecting and instantiating operators from the domain into the goal constraint network. Unlike in classical planning [6], the relevance of an operator (γ^{-1}) is not determined by unifying effects with sub-goals, rather by the unification of an operator's head with a sub-goal. The head of an operator is a non-ground activity which describes the value of a state variable and the information provided as a result of applying the operator. Preconditions and effects are nevertheless modeled, as their presence in the constraint network is dealt with differently at execution time (see Section IV).

An operator can be used to specify the information requirements needed for achieving a given functionality. For instance, the MoveFromTo operator, which does not provide any information content, requires the current position of the robot:

$$\begin{aligned}
a &= (\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, \cdot, \cdot, \cdot, \emptyset) \\
A_p &= \{a_1, a_2\}, A_e = \{a_3\}, \text{where} \\
&\quad a_1 = (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{position}\}) \\
&\quad a_2 = (\mathrm{RobotLocation}, \mathbf{kitchen}, \cdot, \cdot, \cdot, \cdot) \\
&\quad a_3 = (\mathrm{RobotLocation}, \mathbf{livingroom}, \cdot, \cdot, \cdot, \cdot) \\
C &= \{a \,\mathrm{d}\, a_1, a \,\mathrm{m}^{-1}\, a_2, a \,\mathrm{m}\, a_3\}
\end{aligned}$$

The head of the operator is a predicate on the functionality MoveFromTo. The operator is considered relevant when the constraint network contains an activity $(\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, \cdot, \cdot, \cdot, \cdot)$, i.e., when a (sub-)goal stating that the robot must move from the kitchen to the living room is present in the network. The operator also prescribes the temporal relations that must exist between the activities, namely that the MoveFromTo functionality should occur during the availability of the position data ($a \,\mathrm{d}\, a_1$), that it should be met by the precondition of the robot being in the kitchen ($a \,\mathrm{m}^{-1}\, a_2$), and that it meets the effect of the robot being in the living room ($a \,\mathrm{m}\, a_3$).

An operator can also be used to represent a means to achieve certain information requirements. For example, the operator

$$\begin{aligned}
a &= (\mathrm{VisualSLAM}, \mathbf{running}, \cdot, u(\mathrm{CPU}) = 10, \cdot, \{\mathrm{position}\}) \\
A_p &= \{a_1, a_2\}, A_e = \emptyset, \text{where} \\
&\quad a_1 = (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{range\_data}\}) \\
&\quad a_2 = (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{ref\_frame}\}) \\
C &= \{a \,\mathrm{d}\, a_1, a \,\mathrm{m}^{-1}\, a_2\}
\end{aligned}$$

specifies one way to achieve the necessary information requirement (position) for the MoveFromTo operator, namely through visual SLAM. This localization functionality requires (1) a functionality which provides range data, (2) a reference frame for the computation of the position estimate, and (3) 10% of the CPU resource. Also, the operator states that range data should be available during the entire duration of the localization process, and that the reference frame is needed at the beginning of the process.

The above operator does not specify how to obtain the needed information inputs. For instance, the range data might be provided through the following operator:

$$\begin{aligned}
a &= (\mathrm{StereoCamDriver}, \mathbf{on}, \cdot, u(\mathrm{Cam1}) = 1, \cdot, \{\mathrm{range\_data}\}) \\
A_p &= \{a_1\}, A_e = \emptyset, \text{where } a_1 = (\mathrm{Light}, \mathbf{on}, \cdot, \cdot, \cdot, \cdot) \\
C &= \{a \,\mathrm{d}\, a_1\}
\end{aligned}$$

An operator may also specify that the reference frame is obtainable by invoking a functionality of the stereo camera's pan-tilt unit:

$$\begin{aligned}
a &= (\mathrm{PanTilt}, \mathbf{return\_ref\_frame}, \cdot, \cdot, \cdot, \{\mathrm{ref\_frame}\}) \\
A_p &= \emptyset, A_e = \emptyset, C = \emptyset
\end{aligned}$$

The above operators can be applied to obtain a configuration plan from the following goal constraint network:

$$\begin{aligned}
A &= \{a_0 = (\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, I_0, \cdot, \cdot, \cdot)\}, \\
C &= \emptyset
\end{aligned}$$

Specifically, a particular application of the above operators may refine the given constraint network to the following:

$$\begin{aligned}
A = \{ & a_0 = (\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, I_0, \emptyset, \emptyset, \emptyset) \\
& a_1 = (\mathrm{VisualSLAM}, \mathbf{running}, I_1, u(\mathrm{CPU}) = 10, \\
& \qquad \{\mathrm{ref\_frame}, \mathrm{range\_data}\}, \{\mathrm{position}\}) \\
& a_2 = (\mathrm{RobotLocation}, \mathbf{kitchen}, I_2, \emptyset, \emptyset, \emptyset) \\
& a_3 = (\mathrm{RobotLocation}, \mathbf{livingroom}, I_3, \emptyset, \emptyset, \emptyset) \\
& a_4 = (\mathrm{StereoCamDriver}, \mathbf{on}, I_4, \\
& \qquad u(\mathrm{Cam1}) = 1, \emptyset, \{\mathrm{range\_data}\}) \\
& a_5 = (\mathrm{PanTilt}, \mathbf{return\_ref\_frame}, I_5, \emptyset, \\
& \qquad \emptyset, \{\mathrm{ref\_frame}\}) \\
& a_6 = (\mathrm{Light}, \mathbf{on}, I_6, \emptyset, \emptyset, \emptyset)\}, \\
C = \{ & a_0 \,\mathrm{d}\, a_1, a_0 \,\mathrm{m}^{-1}\, a_2, a_0 \,\mathrm{m}\, a_3, a_1 \,\mathrm{d}\, a_4, a_1 \,\mathrm{m}\, a_5, a_4 \,\mathrm{d}\, a_6\}
\end{aligned}$$

This network represents a temporally consistent configuration plan in which resources are never used beyond their capacity, and state variables are never required to assume different values in overlapping temporal intervals. The plan is therefore feasible. Furthermore, the plan contains activities providing the required information contents as determined by the operators in the domain. However, not all causal dependencies are necessarily achieved by construction. If, e.g., the initial condition does not state that the light is on, the configuration planner would regard the activity a_6 as yet another sub-goal to satisfy, and might do so by applying the following operator:

$$\begin{aligned} a &= (\text{Light}, \mathbf{on}, \cdot, \cdot, \cdot, \cdot) \\ A_p &= \emptyset, A_e = \{a_1\}, \text{where } a_1 = (\text{LightController}, \mathbf{on}, \cdot, \emptyset, \cdot, \cdot) \\ C &= \{a \, \mathrm{p}^{-1} \, a_1\} \end{aligned}$$

This operator models an actuation process (Light represents an environment variable), and its application would refine the configuration plan by adding an activity $a_7 = (\text{LightController}, \mathbf{on}, I_7, \emptyset, \emptyset, \emptyset)$ to the network, along with the constraint $a_6 \, \mathrm{p}^{-1} \, a_7$, prescribing that the LightController be in state **on** before the light is required to be on. Note that the light control functionality has no information requirements ($\text{In}^{(a_1)} = \emptyset$).

III. Constraint-Based Search

The planning process used in our approach is incremental in nature, and yields a refined constraint network, which itself represents a feasible configuration plan which achieves the given goal. The resulting constraint network represents one or more temporal evolutions of the state variables that guarantee the achievement of the goal under nominal conditions. Feasible and goal-achieving configuration plans are obtained in our approach by means of four interacting solvers:

Temporal solver. The temporal consistency of the constraint network is checked through temporal constraint propagation by means of a Simple Temporal Problem (STP) [7] solver. The solver propagates temporal constraints to refine the bounds $[l_s, u_s], [l_e, u_e]$ of the activities in the network, and returns failure if and only if temporally consistent bounds cannot be found.

Resource scheduler. This solver enforces that resources are never over-consumed. The maximum capacities of resources restrict which activities can occur concurrently, and this solver posts temporal constraints to the constraint network enforcing that over-consuming peaks of activities are avoided [8].

State variable scheduler. State variable scheduling ensures that activities do not prescribe conflicting states in overlapping intervals. Similarly to the resource scheduler, this solver posts temporal constraints which impose a temporal separation between conflicting activities.

Information dependency reasoner. Operators model the information dependencies between functionalities[1]. This solver instantiates into the constraint network relevant operators (in the form of activities and temporal constraints) so as to enforce the information dependencies.

Causal reasoner. Operators in the domain also model causal dependencies between states. This solver instantiates into the constraint network relevant operators (in the form of activities and temporal constraints) so as to enforce the causal dependencies of the configuration plan.

As noted, resource over-consumption and multiple concurrent states are averted by imposing temporal constraints which sequence potentially concurrent activities; trivially, there are alternative sequencing decisions that can be made to resolve such a conflict, e.g., enforcing $a \, \mathrm{p} \, b$ or $a \, \mathrm{p}^{-1} \, b$. Also operator selection is subject to alternative choices, as more than one operator may provide the necessary information output and/or support the necessary causal dependency (e.g., the presence of light in the environment may be obtained as a result of invoking the light controller or by opening the blinds.) Note that only temporal feasibility enforcement is not subject to multiple choices, as the problem is tractable. In our approach, all requirements which may entail alternative courses of action are seen as *decision variables* in a high-level Constraint Satisfaction Problem (CSP). Given a decision variable d, its possible values constitute a finite domain $\delta^d = \{(A_r^d, C_r^d)_1, \ldots, (A_r^d, C_r^d)_n\}$, whose values are alternative constraint networks, called *resolving constraint networks*. The individual solvers are used to determine resolving constraint networks $(A_r^d, C_r^d)_i$, which are iteratively added to the goal constraint network (A_g, C_g).

```
Function Backtrack($(A_g, C_g)$): success or failure
1  $d \leftarrow$ Choose($(A_g, C_g)$, $h_{var}$)
2  if $d \neq \emptyset$ then
3      $\delta^d = \{(A_r^d, C_r^d)_1, \ldots, (A_r^d, C_r^d)_n\}$
4      while $\delta^d \neq \emptyset$ do
5          $(A_r^d, C_r^d)_i \leftarrow$ Choose($d$, $h_{val}$)
6          if $(A_g \cup A_r^d, C_g \cup C_r^d)$ is temporally consistent then
7              return Backtrack($A_g \cup A_r^d, C_g \cup C_r^d$)
8          $\delta^d \leftarrow \delta^d \setminus \{(A_r^d, C_r^d)_i\}$
9      return failure
10 return success
```

In order to search for resolving constraint networks, we employ a systematic search (see Algorithm Backtrack), which occurs through standard CSP-style backtracking. The decision variables are chosen according to a variable ordering heuristic h_{var} (line 1); the alternative resolving constraint networks are chosen according to a value ordering heuristic h_{val} (line 5). The former decides which (sub-)goals to attempt to satisfy first, e.g., to support a functionality by applying another operator, or to resolve a scheduling conflict. The latter decides which value to attempt first, e.g., whether to prefer one operator over another. Note that adding resolving constraint networks may entail the presence of new decision variables to be considered.

The possible values for resource contention or unique state decision variables are temporal precedences among activities. Values for information decision variables are ground

[1]In our approach, the domain is such that information dependencies constitute an acyclic propositional Horn theory.

operators, as shown in the previous Section. Lastly, values for causal decision variables are either ground operators, or *unifications* with activities that already exist in the constraint network. Two activities a and b can be unified if $\mathrm{x}^{(a)} = \mathrm{x}^{(b)} \wedge \mathbf{v}^{(a)} = \mathbf{v}^{(a)}$. Unifications are enforced by imposing a temporal equality constraint $a \equiv b$ among the activities. Supporting unification is obviously necessary to allow the search to build on previously added activities — e.g., leveraging that the light has already been turned on to support a previously branched-upon causal dependency. More importantly, unification also allows to accommodate on-going sensing and execution monitoring processes during configuration planning. For instance, activity $a = (\mathrm{Light}, \mathbf{on}, I^{(a)}, \emptyset, \emptyset, \emptyset)$ could be supported by unification with an activity $a_{\text{sensed}} = (\mathrm{Light}, \mathbf{on}, [[0, 0][13, 13]], \emptyset, \emptyset, \emptyset)$ which models the temporal interval within which a light source was sensed by a sensor in the environment.

IV. Plan Execution and Dynamic Plan Update

The ability to support on-line sensing is directly enabled by the constraint-based representation: sensing is reduced to dynamically updating the constraint network with new activities and constraints representing the sensed state of the environment; the same mechanism also supports prediction (i.e., "sensing in the future") and other on-line plan modifications, such as temporal delays and dynamically posted goal constraint networks.

Our approach is based on the alternation of planning and plan execution monitoring. The former consists of the planning procedure shown above. The latter consists of two processes: *sensing* and *plan update*. The sensing process adds to the constraint network activities and temporal constraints representing the current view of the environment as provided by sensors. The plan update process maintains and updates temporal constraints which bound on-going activities (sensed states or functionalities in execution) with the current time. This is done in $O(n^2)$ through incremental temporal constraint propagation [7], where n is the number of activities in the constraint network. Also, this process imposes constraints that verify the existence of preconditions and trigger the manifestation of effects contained in the plan. Specifically, the presence of a precondition is verified by attempting to unify the activity representing the precondition with a sensed activity. If the unification is not possible, the precondition is delayed by inserting a temporal constraint, and is re-evaluated at the next iteration. The process enforces the occurrence of activities representing effects by posting temporal constraints which fix their start time to the current time. The effect of the constraints posted by these processes is that functionalities start when possible, are delayed until the preconditions hold, and their effects are imposed when necessary. This step also requires polynomial computation.

In our current implementation, all solvers monitor the network for new decision variables. Thus "re-planning" occurs by temporal propagation, resource or state variable scheduling, or operator application, depending on the situation.

Note that this allows to keep the computational impact of replanning at a minimum (e.g. operator application need not occur if scheduling is sufficient, scheduling need not occur if temporal propagation is sufficient). This mechanism is what enables dynamically posted goals, as in other temporal constraint-based continuous planners [9], [10], but here we also deal with resources, sensor data and information constraints.

All the components so far described post activities and/or constraints into the temporal network and their relations can be compared to the ones existing between components of a classical control system. The dynamic goal posting corresponds to the desired state for the system to control; in order to achieve this state, that can possibly change over time, several solvers try to manipulate, i.e. formulate control signals, the temporal network. Once decisions are taken, control signals are injected into the state if they did not lead to temporal inconsistencies (validation performed by the temporal solver). Finally the state of the world is continuously fed-back to the system through the observer. A schematic representation of this comparison is depicted in Fig.1. We show an example of this behavior in the next Section.

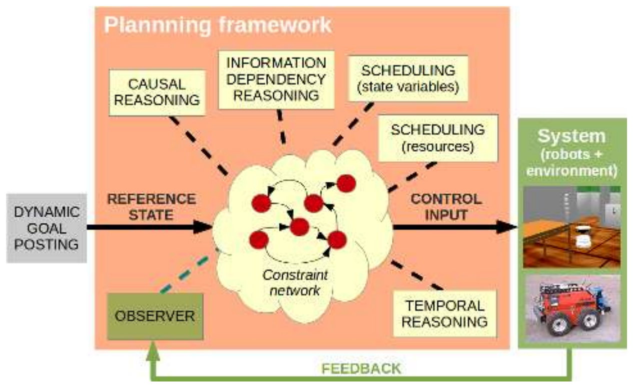

Fig. 1. High level reasoners (causal reasoner, information dependency reasoner, schedulers) modify the constraint network so as to achieve the dynamically changing desired state (dynamic goal posting). Their decisions are temporally validated (temporal reasoning) and sent to the system as control signals. Reasoning accounts for the current state of the system, which is continuously maintained in the constraint network (observer).

V. Experiments

In this section we show experiments inspired by the introductory scenario. It is time for lunch, and the outdoor robot, *Rout*, delivers groceries to Sven's domestic robot *Rin*, which is waiting at the door. Sven is in the kitchen; this is a small environment, and at most one robot can be there at any point in time, i.e., the kitchen is modelled as a resource with capacity one which is used by all activities which bring a robot to the kitchen. We show two variants of the scenario: in the first, Sven's apartment is equipped with one robot which, upon receiving the goods, delivers them to Sven; in the second, Sven possesses two indoor robots, *Rin1* and *Rin2*, with similar capabilities. All robots are equipped with software modules for localization and navigation: *Rout* can employ either a GPS module or a laser range finder, while the indoor units are equipped with Kinects. The first variant was executed with real robots, experiments in the second variant were performed on the Gazebo simulator [11]. All robots run ROS [12], and the communication with the planner is realized through the light-weight PEIS middleware [2].

A. *Variant 1: Reacting to Information Requirement Failures*

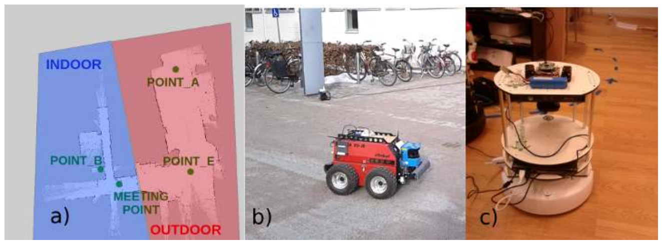

Fig. 2. Experimental setup: a) map of the environment with the related waypoints; b) outdoor robot; c) indoor robot.

In the first scenario, we employ two robotic platforms: an ATRV-Jr for *Rout*, and a Turtlebot-1 for *Rin*. Fig. 2 shows a map of the environment and the two robots used.

The plan generated for delivering groceries to Sven, who is in the kitchen, is shown in Fig. 2-a: under nominal conditions, *Rout* is expected to navigate towards the meeting point, MP, relying on the GPS signal for localization. *Rin* is expected to reach MP as soon as *Rout* has reached an intermediate point E. Once both robots are in MP, *Rin* and *Rout* exchange the groceries through the actions *unload* and *grab*, respectively. Note that this plan contains required concurrency. This is achieved through temporal constraints which model the fact that both robots cannot terminate the respective operations before the exchange has happened. In addition to delivering the groceries, *Rin* must also accomplish another task, *OTHER*, within a certain deadline d.

Shortly after execution begins, *Rout*'s GPS fails. This leads the configuration planner to support the information requirement of *Rout*'s navigation module with the laser-based localization module. However, navigation speed must be reduced to collect reliable readings from the laser. As a consequence, the expected time at which *Rout* will reach MP increases compared to the nominal plan. This violates the deadline d for the *OTHER* task, hence the planner reschedules the execution of this taks to occur while *Rout* is reaching the meeting point. A video showing a real execution of the above scenario is available at `http://aass.oru.se/~modo/IROS2013/iros2013.html`.

B. *Variant 2: The Consequences of Being Late*

In this variant we assume that the overall task consists of two goals. The first is to deliver the groceries to Sven using one of the two indoor robots. This goal is posted to the configuration planner, which allocates the task to *Rin1*. The timelines showing how the plan was executed up to time $t = 20$ is shown in Fig. 4-a. At this time, a second goal is dynamically posted, namely to deliver pills to Sven within a certain deadline d; both *Rin1* and *Rin2* are equipped with a pill dispenser. The planner synthesizes a plan which is seamlessly merged with the existing constraint network. The planner allocates *Rin1* for accomplishing the pill delivery, thus exploiting the path that *Rin1* is going to traverse to deliver the groceries. During execution, at time $t = 70$, the planner is informed that *Rout* is delayed (see Fig. 4-b). This leads to a temporal failure, as the extent of the delay makes delivering the pills within time d impossible. This failure cannot be adjusted through temporal propagation alone, and the planner is faced with the following three options:

1) **Multi robot execution.** The planner allocates to *Rin2*, which is so far idle and also equipped with a pill dispenser, the task of reaching Sven in the kitchen while *Rin1* is waiting for the groceries (see Fig. 4-c). Note that this solution requires scheduling to manage the possible concurrent access of *Rin1* and *Rin2* to the kitchen: due to the one-robot requirement in the kitchen, *Rin1* will wait for *Rin2* to return to its base station before proceeding to the kitchen.
2) **Task rescheduling.** In order make the deadline, the planner can first send *Rin1* to deliver the pills, and successively come back to fetch the groceries once *Rout* is at the door. This procedure, although more expensive in terms of makespan, allows the planner to still use one robot (see Fig. 4-d).
3) **Causal dependency change.** Another way to fulfill the goal of getting Sven to eat lunch is to send an indoor robot to the senior living facility's canteen to fetch a prepared meal. Thus another option for the planner is to find an alternative way to fulfill the causal dependency: to dispatch *Rin1* to achieve both meal acquisition and pill delivery (see Fig. 4-e), thus adapting on-line the fulfillment of a causal dependency due to a temporal contingency.

Which option is chosen by the planner depends on which guiding heuristic has been implemented. A video showing a Gazebo execution in which case 1 was selected is available at `http://youtu.be/adiwuywxEiM`.

VI. Discussion and Conclusions

Much research in robotics aims to deploy robots in dynamic, uncontrolled environments which require systems to be very adaptive. The work presented in this paper enables the generation and run-time adaptation of configuration plans, yielding robust closed loop performance in the face of perturbations such as delays, resource collapse, exogenous events, and new goals. An important tenet of our approach is that plans are only modified to the extent which is necessary. For example, a new goal can often be accommodated by simply adding actions to the current plan; and delays or resources unavailability can often be remedied by rescheduling. Indeed, full re-planning is most often necessary when assumed causal requirements fail to materialize, e.g., an object is not where it was expected to be.

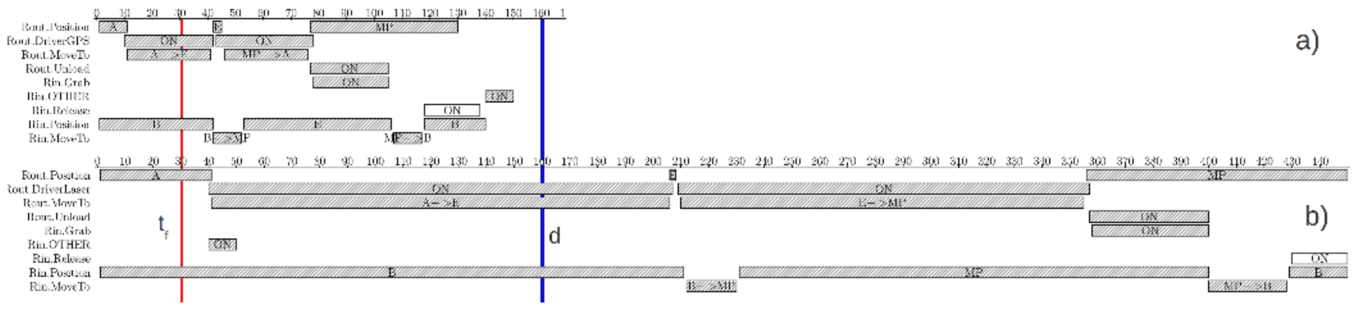

Fig. 3. Plan execution for the grocery delivery scenario with real robots (variant 1): a) plan exploiting the navigation with GPS before the failure at t_f; b) navigation with laser that respects the deadline d.

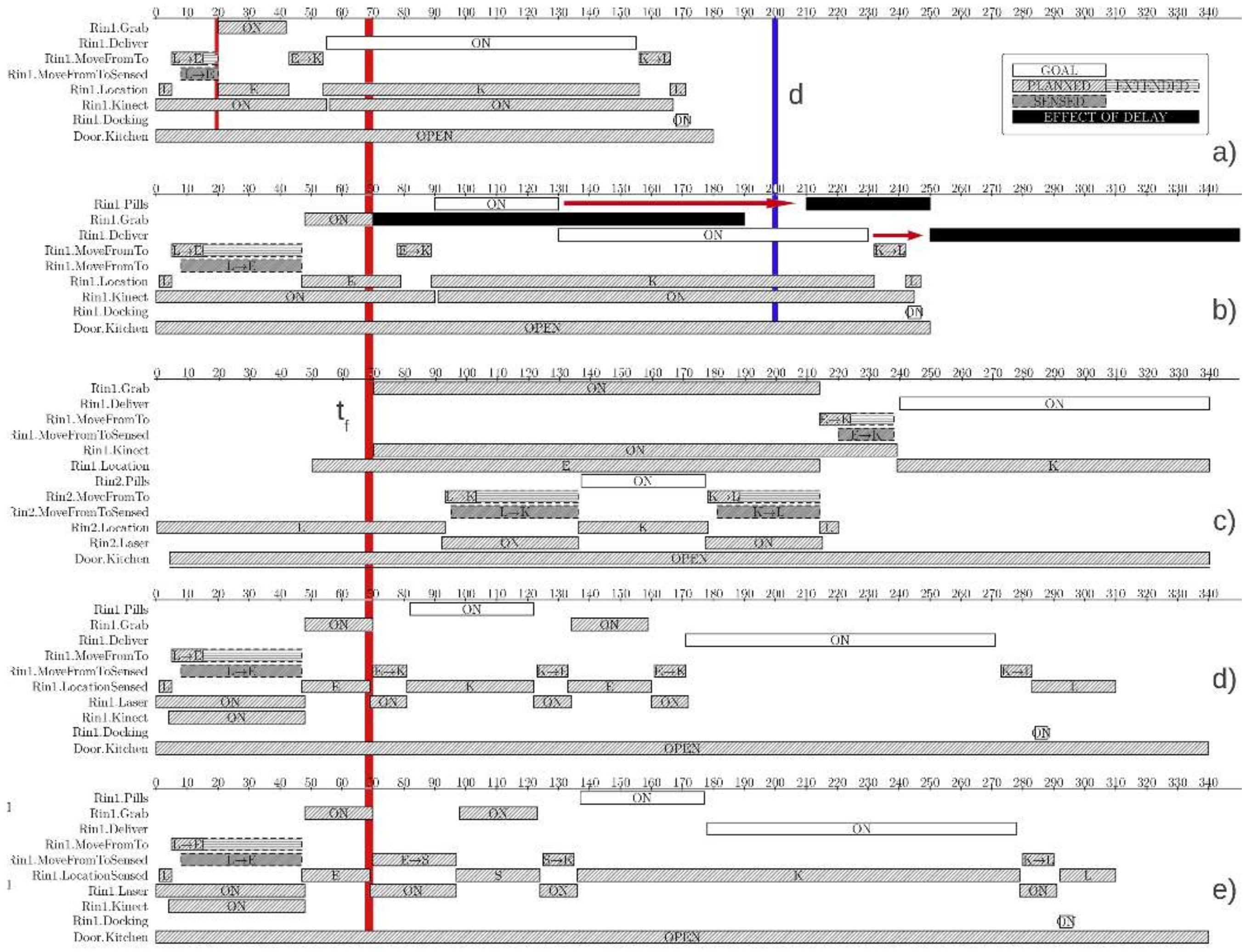

Fig. 4. Plan execution for the grocery delivery scenario with real robots (variant 2): a) execution of the initial plan — at time $t = 0$ the first goal is posted and the plan is further enriched at time $t = 20$; b) the delay induced by the delivery makes the plan fail due to the failure of the deadline; c) after plan failure another robot joins the plan execution d) the delay is absorbed detouring the robot to execute first the delivery of the pills and then sending it back to fetch the package; e) the fetching of the package is substituted by an equivalent causal action which allows to complete the plan.

Several approaches have been proposed in the AI planning literature to cope with temporal contingencies [13], [14], [10], most of which rely on temporally flexible plans which can be adapted in low-order polynomial time to temporal uncertainty. In [9], temporal planning has been coupled with execution monitoring, and the approach has been successfully employed with real robotic systems. In [15], execution monitoring takes into account resource usage, and the technique is exemplified on single robot navigation tasks. These systems have proved the effectiveness of constraint reasoning in robotic domains. Our work builds upon and extends this tradition, by proposing a framework that integrates seamlessly planning and execution monitoring, while taking into account information dependencies and resource constraints.

All aspects of plan generation and execution are complicated in the case of multi robot systems and robot ecologies [2], like the ones that we address here, due to the multiple causal and information dependencies among different robots, sensors and actuators [16]. Some of these aspects are considered in [17], which proposes constraint-based reasoning techniques for assembling coalitions of robots that perform a common task using shared resources. However, that work considers only binary resources (i.e., resources that can be used by one robot at a time) and does not deal with time.

A more fine-grained approach is the AsymTre architecture [3], [18]: each robot is equipped with a set of software modules, called schemas, able to sense and modify the environment. Schemas are interconnected through information channels which are decided by a planning process. AsymTre did not consider resource and temporal requirements, but later extensions [19] support a limited form of resource reasoning. However, the method does not provide closed-loop execution monitoring; also, dealing with the removal of a shared action may lead to problems that are not addressed by the proposed techniques.

Lundh and colleagues [4] propose a configuration planner that uses an explicit representation of the world. Differently from AsymTre, Lundh's system leverages a propositional logic description of the world based on standard planning techniques. Reasoning is performed by coupling an action planner with a configuration planner: the former provides a sequence of actions that have to be further refined by the latter, by deciding the relevant software modules and their communication linkage. This system allows a detailed representation of the evolution of the world — however, it is decoupled from execution, and therefore suffers from many of the aforementioned problems. An improved version of this work [20] takes into account multiple goals through a merging sequence. Similarly to AsymTre, Lundh's approach lacks the ability to perform integrated plan monitoring and execution, and on-line plan adaptation.

Compared to the works above, our approach is unique in its combining reasoning about information, causal, temporal and resource requirements, as well as providing closed-loop behavior in which new goals and unexpected contingencies can be accommodated during execution. We claim that these features are pivotal to the development and deployment of real autonomous multi-robot systems.

The system presented in this paper is currently being used for planning and flexible execution of a robot ecology in the context of the EU project Rubicon. Future improvements concern: the scalability of the system with respect to larger ecologies (in particular 3 robots and a dozen of sensors/actuators spread out in a building); the introduction of costs and quality criteria that will allow for the generation of optimal plans; introduction of other types of knowledge such as spatial constraints and expectation about human actions.

Acknowledgements

This work was funded by the EC Seventh Framework Programme (FP7/2007-2013) grant agreement no. 269914 Rubicon – "**R**obotic **Ubi**quitous **Co**gnitive **N**etwork" (`http://www.fp7rubicon.eu/`).

References

[1] R. Fikes and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[2] A. Saffiotti, M. Broxvall, M. Gritti, K. LeBlanc, R. Lundh, J. Rashid, B. Seo, and Y. Cho, "The PEIS-ecology project: vision and results," in *IROS*, 2008, pp. 2329–2335.

[3] L. Parker and F. Tang, "Building multirobot coalitions through automated task solution synthesis," *Proc of the IEEE*, vol. 94, no. 7, pp. 1289–1305, 2006.

[4] R. Lundh, L. Karlsson, and A. Saffiotti, "Autonomous functional configuration of a network robot system," *Robotics and Autonomous Systems*, vol. 56, no. 10, pp. 819–830, 2008.

[5] J. Allen, "Towards a general theory of action and time," *Artificial Intelligence*, vol. 23, no. 2, pp. 123–154, 1984.

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

[7] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.

[8] A. Cesta, A. Oddi, and S. F. Smith, "A constraint-based method for project scheduling with time windows," *Journal of Heuristics*, vol. 8, no. 1, pp. 109–136, January 2002.

[9] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen, "A Deliberative Architecture for AUV Control," in *Int. Conf. on Robotics and Automation (ICRA)*, 2008.

[10] J. Barreiro, M. Boyce, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, T. Smith, and M. Do, "EUROPA: A platform for AI planning," in *Proc of ICAPS-ICKEPS*, 2012.

[11] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. of the IEEE/RSJ Int Conf on Intelligent Robots and Systems*, 2004, pp. 2149–2154.

[12] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[13] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood, "Casper: Space exploration through continuous planning," *Intelligent Systems*, vol. 16, no. 5, pp. 70–75, 2001.

[14] P. Doherty, J. Kvarnström, and F. Heintz, "A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems," *Autonomous Agents and Multi-Agent Systems*, vol. 19, no. 3, pp. 332–377, 2009.

[15] S. Lemai and F. Ingrand, "Interleaving temporal planning and execution in robotics domains," in *Proc. of the National Conference on Artifical Intelligence*, 2004, pp. 617–622.

[16] P. McGillivary, K. Rajan, J. de Sousa, F. Leroy, and R. Martins, "Integrating autonomous underwater vessels, surface vessels and aircraft as persistent surveillance components of ocean observing studies," in *Autonomous Underwater Vehicles (AUV)*, 2012, pp. 1–5.

[17] L. Vig and J. Adams, "Multi-robot coalition formation," *Robotics, IEEE Transactions on*, vol. 22, no. 4, pp. 637–649, Aug.

[18] L. Parker and Y. Zhang, "Task allocation with executable coalitions in multirobot tasks," *Proc of the Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[19] Y. Zhang and L. E. Parker, "Considering inter-task resource constraints in task allocation." *Autonomous Agents and Multi-Agent Systems*, vol. 26, no. 3, pp. 389–419, 2013.

[20] R. Lundh, "Robots that help each-other: Self-configuration of distributed robot systems," Ph.D. dissertation, Örebro University, Örebro, Sweden, 2009.