# APPLYING AI ACTION SCHEDULING TO ESA'S SPACE ROBOTICS

**Daniel Díaz**[1], **Maria Dolores R-Moreno**[1], **Amedeo Cesta**[2], **Angelo Oddi**[2], **and Riccardo Rasconi**[2]

[1]*Universidad de Alcala, Alcala de Henares, Madrid (Spain)*
[2]*CNR – Consiglio Nazionale delle Ricerche, ISTC, Rome (Italy)*

## ABSTRACT

This paper presents recent results on applying robust state-of-the-art AI Planning and Scheduling (P&S) techniques to mobile space robotic domains. The major contribution of this paper is an enhanced version of the ESTA scheduling algorithm which is to capture all relevant aspects of the robot activity scheduling problem in the space domain of interest. Furthermore, we provide an example of how the new solver can be used within a simple model-based autonomous control system for robust action scheduling and flexible reactive schedule execution for: (1) providing initial schedule solutions and (2) repairing invalidated schedules during execution because of unexpected external events.

Key words: Automation and Robotics; robust constraint-based action scheduling; flexible reactive schedule execution; precedence constraint posting.

## 1. INTRODUCTION

The European Space Agency (ESA) is increasingly interested in evolving current telerobotics capabilities towards a complete mixed-initiative [7] strategy implementation that enables shifting control modes from purely manual to fully autonomous operations. Intelligent error handling mechanisms coupled with basic high level mission planning techniques currently represent the state-of-the-art in "Automation and Robotics (A&R)". Continuing to promote autonomy for future space missions certainly entails enormous benefits such as the reduction of operational costs, the enablement of opportunistic science, or the increase of mission quality in terms of safety, science return, reliability and flexibility. Very often, the implementation of autonomous high-level competences on-board (specially in deep space missions) becomes essential, since real time command dispatching is not possible due to the astronomic distances involved, or to the scarcity of time windows for establishing communication [10].

In this context, the paper presents recent results obtained within a Ph.D. program on the topic of *Autonomy for Interplanetary Missions* funded and supported by ESA. Our current work is mainly aimed at developing a model-based autonomous control system for robust AI mission Planning and Scheduling (P&S) and flexible reactive execution, applied to highly dynamic and unstructured mobile space robotics domains. The following futuristic mission scenario is an hypothetical application example that aims at illustrating how autonomy could be applied in deep space missions.

**Mission Scenario**. *The complete scenario describes a future ESA deep space mission where autonomous rovers are in charge of bringing raw moon dust from different mines to a pilot plant that extracts oxygen from collected materials. Typical rover activities include loading, transporting and unloading the extracted minerals, as well as performing basic self-maintenance tasks. Rovers are able to autonomously synthesize efficient action plans by optimizing energy management and plan completion time. Furthermore, advanced on-board re-planning capabilities are necessary in order to hedge against environmental uncertainty during execution (i.e., rugged terrain, harsh weather conditions, etc.).*

Synthesizing commands on-board basically involves deciding sequences of actions to be safely and efficiently executed on highly hazardous environments, by considering complex temporal and resource constraints. One of the major challenges in promoting autonomy for future space missions is to provide robots with basic autonomous energy management capabilities, since non-trivial issues have to be considered such as: *what is the best moment for a robot to recharge its battery? How much energy should a robot get during a recharging cycle? How to safely cope with unexpected energy losses?* or *Does the chosen energy management strategy depend on the nature of the energy source?*

We start our analysis from a significantly reduced version of the previous scenario that only considers a unique robot. We refer to the problem as *scheduling and running a robot in an unstructured environment*. This scenario describes a twofold problem that consists of: (1) synthesizing feasible sequences of movements for a mobile robot as it performs a set of jobs over time by synchronizing the use of a set of "experimentation facilities or laboratories"; and (2) safely executing the movements by taking into account environmental contingencies. Each job consists of a chain of experiments that have to be sequentially performed on a set of geographically distributed laboratories. Each experiment is an activity that entails the processing of one item on behalf of a facility whose use is exclusive for the whole duration of the experiment (no pre-emption is allowed). Each facility can only process one item at a time, while all experiment durations are fixed. The robot is in charge of loading, transporting and unloading different items among all present facilities in order to allow their processing along all the jobs. At the beginning of the whole process all the items are kept in an initial storage area, while at the end all the items will have to be unloaded in a final storage area. Transportation times between different laboratories depend on the traveling distance, and the robot can simultaneously carry a maximum number of items. The robot activities (i.e., loading, transporting and unloading) require a specific amount of energy in order to be executed, and the rover battery is recharged through solar arrays at a specific rate. Three different types of contingencies are considered during schedule execution: activity delays, duration changes, and resource breakdowns. If one of the previous unexpected situations invalidates the schedule under execution, the robot will perform a re-scheduling step to ensure the completion of the jobs.

In this paper we propose a basic controller that implements a single Sense-Plan-Act (SPA) closed-loop for action scheduling, command dispatching and schedule execution monitoring with the aim of addressing the previous problem. We exploit state-of-the-art AI scheduling techniques both to provide initial schedule solutions and to repair invalidated schedules during execution. More concretely, we use an advanced constraint-based, resource-driven solving procedure based on the last reported results in constraint-based P&S techniques, with particular attention to the "Precedence Constraint Posting" (PCP) approach as described in [4, 13, 3]. Furthermore, the basic execution features for enabling simulation of command dispatching, monitoring and dynamic re-scheduling are also presented.

The remainder of the paper is structured as follows: Section 2 describes in detail the problem of scheduling and running a robot in an unstructured environment. Section 3 introduces the constraint-based, resource-driven solving procedure which provides the controller with advanced reasoning capabilities, as well as a meta-heuristic optimization framework for solution optimization. In Section 4 an overall functional description of the basic autonomous controller is sketched. Finally, a conclusions and future work section closes the paper.

## 2. PROBLEM FORMULATION

In the current section we define the problem here addressed and the constraint-based representation that we use for problem solving.

### 2.1. Problem description

The problem of scheduling and running a robot in an unstructured environment aims at: (1) synthesizing feasible sequences of movements for a mobile arm-equipped robot *R* which is involved at performing a set of jobs over time by synchronizing the use of a set of experimentation facilities or laboratories $F = \{\mu_1, \ldots, \mu_n\}$, and (2) safely executing them by taking into account environmental contingencies.

Each job consists of a set of experiments (or Activities) $A_j$=$\{l_{0j}, u_{1j}, a_{1j}, l_{1j}$ , $\ldots$, $u_{ij}, a_{ij}, l_{ij}$, $\ldots$, $u_{nj}, a_{nj}, l_{nj}, u_{(n+1)j}\}$ to be sequentially processed, where $a_{ij}$ is the $ith$ activity belonging to Job $j$ performed by facility $\mu_{ij} \in F$, while $u_{ij}$ and $l_{ij}$ are, respectively, the *Unload* and *Load* activities that the robot must perform at facility $\mu_{ij}$ to release and grab items respectively.

$l_{0j}$ is the Load activity performed by the robot at the initial storage location, while $u_{(n+1)j}$ is the Unload activity performed by the robot at the final storage location. Transportation activities $\tau_{ij}$ are considered as a different type of activities that correspond to the possible movements performed by the robot between two different facilities $< \mu_i, \mu_j >$ within the same or distinct jobs, including initial and final storage locations.

The execution of all the activities is subject to the following constraints:

- *Resource availability*: each activity $a_{ij}$ requires the exclusive use of $\mu_{ij}$ during its entire execution, i.e., no preemption is allowed. All the activities belonging to the same job demand distinct facilities. The activities $u_{ij}$ and $l_{ij}$ require the exclusive use of the robot *R*.
- *Processing time constraints*: all activities $a_{ij}$, $u_{ij}$ and $l_{ij}$ have a fixed processing time. Both the experimentation facilities and the robot can perform one operation at a time.
- *Transportation time constraints*: for each pair $< \mu_{ix}, \mu_{jy} >$ of facilities, the robot *R* is in charge of moving all items processed by the facility $\mu_{ix}$ to the facility $\mu_{jy}$. This entails performing a Load activity $l_{ix}$ at $\mu_{ix}$, transporting the item at $\mu_{jy}$ (i.e., performing a $\tau_{ij}$ activity), and finally performing an Unload activity $u_{jy}$ at $\mu_{jy}$. The time necessary to travel from $\mu_{ix}$ to $\mu_{jy}$ is directly proportional to the travelling distance, and is modelled in the problem in terms of *sequence dependent setup times* $st_{ij}$ which must be enforced between each $< l_{ix}, u_{jy} >$ activity pair. All setup constraints satisfy the *triangle inequality property*, i.e., given three facilities $\mu_i$, $\mu_j$ and $\mu_k$, the condition $st_{ij} \leq st_{ik} + st_{kj}$ always holds.
- *Robot maximum capacity constraint*: the robot is able to transport a maximum of *C* items at a time. This entails that the robot may chain different Load or Unload activities a maximum number of *C* times.

– *Battery capacity constraints*: the robot has a battery with a maximum $Bat_{max}$ *(units of energy)* capacity of charge or *saturation level*. A specific usage threshold $Bat_{thres}$ *(%)* delimits the total amount of energy that can be demanded as a safeguard for unexpected situations (i.e., if $Bat_{max}$*=100* and $Bat_{thres}$*=80(%)* it cannot be consumed less than 20 units of energy).

– *Energy production constraint*: the battery is continuously charged at a monotonic rate $\sigma_{charge}$ *(unit energy/unit time)* through solar cells until the saturation level is reached (from this moment on, all collected energy is discarded).

– *Energy consumption constraints*: the robot activities (i.e., Load, Unload and Transporting) require a certain amount of energy that have to be available at the starting time of their execution. While the Unload and Load activities always demand the same constant amount of energy, the consumption of the Transporting activities is estimated as a value that is directly proportional to the travelling distances.

The first part of the problem is referred to a purely scheduling optimization problem whose goal is to (statically) synthesize a feasible schedule where both experiments and robot activities are synchronized towards the successful execution of all jobs. A feasible schedule solution is a schedule where the start and end times of all activities are assigned while temporal and resource constraints are satisfied. We are interested in schedule solutions which minimize their total completion time or makespan. The second part of the problem is involved with safety executing the resulting feasible schedule by considering the possible disturbances caused by the environmental threats. The execution of the baseline schedule is addressed in a further section in detail.

## 2.2. Constraint-based problem representation

Our solution considers the scheduling problem as a special type of a Constraint Satisfaction Problem (CSP) [9]. A general description of a scheduling problem as CSP involves a set of variables with a limited domain each, and a set of constraints that limits some value combinations. Hence, a feasible CSP solution is defined as an assignment of domain values to all variables which is consistent with all imposed constraints. A general CSP solution scheme can be seen as an iterative procedure that interleaves two main steps in each cycle:

– A *decision making step* where a variable is chosen to be assigned with a specific domain value. The decision of the assignment to do next could be taken by either systematically following an exhaustive search technique (such as a simple depth-first search), or by using more efficient approaches that use variable and value ordering heuristics to guide the search process. Typical general purpose heuristics generate variable and value orderings by selecting the "most constrained variable (MCV)" and the "least constraining value (LCV)" respectively.

– A *propagation step* where a set of "inference rules" prune unfeasible solutions in advance, by removing elements from variable domains when a decision step is performed. Path consistency algorithms such as "all pairs shortest paths" are typically used.

New problem modelling assumptions are considered in order to adapt the initial problem formulation to our constraint-based solution scheme. Hence, the problem is described now as follows (see figure 1): each facility $\mu_{ij}$ is considered as a binary resource that process all job operations $a_{ij}$. The Unload and Load activities are devised to be performed by a single robotic arm able to manage one item at a time (also modelled as a binary resource). We introduce an additional kind of activity $c_{ij}$ that requires the use of a cumulative resource (with a maximum capacity of $C$) to be processed by the robot, with the aim of modelling the robot capability of *simultaneously carrying multiple items*. The execution of $c_{ij}$ activities starts when the corresponding Load activity $l_{ij}$ starts, and finishes at the termination of the Unload activity $u_{i+1j}$.

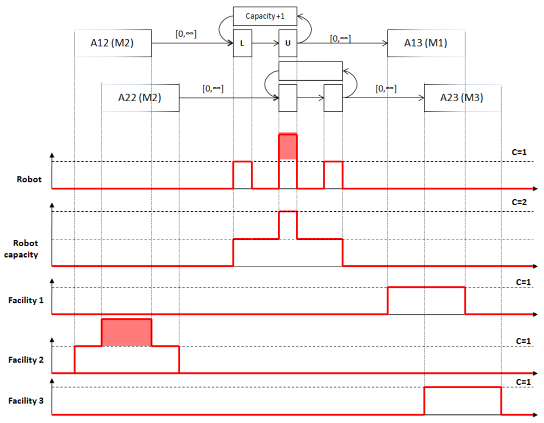

**Figure 1:** A cumulative resource is introduced to model the multi-capacity robot usage.

The battery is represented as another cumulative resource according to the model presented in [18], with a maximum capacity of $Bat_{max}$ and an usage threshold of $Bat_{thres}$. The battery resource processes two different sets of activities (see figure 2): the robot activities (i.e., Load, Unload and Transporting) that consume the battery, and the energy collecting activities that recharge the battery. The energy consuming activities are modelled so as to start when a specific amount of energy is demanded and finish at the "horizon" (the end of the schedule), i.e., this amount of energy is no longer available and thus the battery remains blocked until the end; meanwhile the energy producing activities are modelled as static activities that are anchored at the origin of the schedule and finish when energy is collected, i.e., the battery remains blocked from the beginning until a certain amount of energy becomes available.

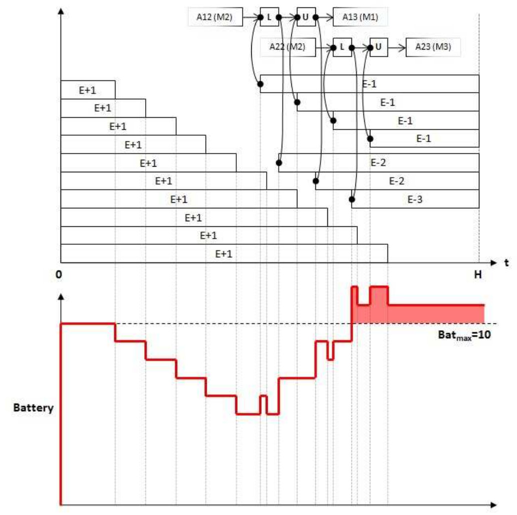

**Figure 2:** Example of an intersected energy collection/consumption profile

## 2.3. Related work

We can mainly distinguish two different types of resources in literature, depending on how a given activity demands and affects the availability of the resource: *reusable* and *consumable* resources. The first type is also known as *renewable* or *cumulative*, and activities that use them may demand a certain quantity of resource during an interval of time. The sum of all activity demands cannot overpass the maximum level capacity. If the resource has unit capacity then it is known as *unary* or *discrete*, and the activities that use it have to be totally ordered since overlaps cannot occur. Examples of cumulative resources are the container of the robot used to transport items as a multi-capacity resource, and the robotic arm or the experimentation facilities as unary resource. The second resource type is also known as *reservoir* and it is a multi-capacity resource that can be consumed and/or produced by an activity. The battery of the robot is an example of consumable resource that is recharged by the solar cells and consumed by the robotic activities.

Research in CSP-based scheduling has been mainly focused on the development of effective heuristic-biased models that involve unary and cumulative resources, such as the precedence constraint posting algorithm proposed by [4] or the resource profile-driven algorithm ESTA [3] respectively. To the best of our knowledge, the majority of the work done in CSP-based scheduling with consumable resources is referred to "resource constraints representation and propagation problems" rather than proposing strategies for problem-solving. Hence, we can find expressive formalisms for expanding the basic Simple Temporal Network (STN) [5] foundations to explicitly deal with resource constraints in general, by defining positive and negative resource allocations to resource productions or consumptions events respectively. Some examples are: the work described in [11], that defines an augmented STN called "Activity Network" that allows a *piecewise constant* representation of the resource usage, on top of which an envelope-based resource consistency checking mechanism is built; or the "Linear Resource Temporal Network (LRTN)" proposed by [6] that enables bounding the resource availability for activity networks with linear resource impact, i.e., permits a *piecewise linear* representation of the resource usage along the duration of the activities.

A different direction to cope with consumable resource constraints (used in this work) proposes a simplified model that aims at reducing consumable constraints to a cumulative scheme by performing some simple transformations. An example is described in [18] and formally defines producer and consumer events with a "classical cumulative scheme" through the following conversion: since a producer event makes some amount of resource available at a specific time instant, such event is expressed as an activity that blocks the resource from the start until the resource becomes available; in the same way, a consumer event is modelled as an activity that uses the resource at a time point and remains blocked until the end, since the resource is no longer available.

## 3. THE CONSTRAINT-BASED, RESOURCE-DRIVEN REASONER (EXTENDED-ESTA)

Since cumulative resources are needed to model the multi-capacity robot usage and the battery, we have chosen the ESTA algorithm [3] as reference constraint-based solving procedure to implement our solution. Furthermore, we have studied and adapted the basic principles of a recent extension of the SP-PCP (Shortest Path-based Precedence Constraint Posting) algorithm proposed in [12]. More concretely, we have adopted the new set of *dominance conditions* introduced in [12] as a set of four basic rules to decide the conflict resolution strategy by considering sequence-dependent setup times.

### 3.1. Adapting PCP to our robot scheduling problem

ESTA was initially developed for solving the scheduling problem known in OR literature as (RCPSP/max) [2]. It follows an advanced precedence constraint posting schema that creates a meta-representation of the temporal CSP network (i.e., the robot scheduling problem represented as a temporal constraint network) to explicitly deal with resource aspects. Thus, ESTA algorithm basically consists on iteratively interleaving the two following steps until a conflict-free schedule is found:

– *Temporal analysis.* Corresponds to the first step of the algorithm and consist on creating a basic temporal network (ground-CSP) to represent and reason about the temporal constraints of the problem. Thus, such temporal constraint network corresponds to a Simple Temporal Problem (STP) formulation where time points represent start and end times of activities, and temporal constraints between time points represent both the duration of the activity and the precedence relations between pairs of activities. Temporal propagation (for computing current bounds for all time points after posting a new temporal precedence constraint) and solution extraction[1], are operations directly performed over this STP formulation.

– *Resource analysis.* The second step of ESTA basically consist on the following sequence: firstly, a meta-CSP representation is created by identifying a set of capacity violations inferred from previous ground-CSP, where variables correspond to the remaining resource conflicts and values to the set of feasible activity orderings to solve them; secondly, a resource conflict is selected by applying a variable ordering heuristic; and finally, selected conflict is solved by using a value ordering heuristic that imposes a new precedence constraint (over the ground-CSP) between some pair of competing activities that contributes to the conflict.

In other words, ESTA algorithm implements a (one-pass) greedy resource-driven scheduler that uses an earliest start-time resource profile projection (ground-CSP) to later perform a resource analysis and iteratively select and level "resource contention peaks" (i.e., overcommitments). More concretely, resource analysis consist on synthesizing the meta-CSP by computing (sampling) the Minimal Critical Sets (MCSs) [8], i.e., sets of activities that overlaps in time and demand same resource causing over-commitments, such that whatever subset does not cause a resource conflict (see figure 3).

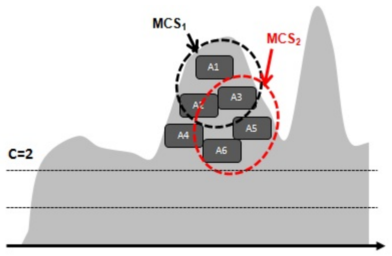

**Figure 3:** Meta-CSP generation example (MCSs computation).

The search strategy for selecting and solving resource contention peaks is biased by the following variable and value ordering heuristics:

– Once the decision variables are computed (candidate MCSs), a most constrained variable ordering heuristic chooses the MCS with the smallest temporal flexibility.

[1]Solution extraction provides a conflict-free schedule in the form of Earliest Start Schedule (ESS): a consistent temporal assignment of all time points with their lower bound values that is also resource consistent.

**Algorithm 1**: Conflict selection and resolution process.

```
Conflict ← SelectConflict (MetaCSP)
Precedence ← SelectPrecedence (Conflict)
GroundCSP ← PostConstraint (GroundCSP, Precedence)
```

– Conflict resolution is performed by a least constrained value ordering heuristic that levels the contention peak by posting a simple precedence constraint between two activities that belong to the related MCS according to the following criteria: the greater the flexibility is retained after posting a precedence ordering constraint, the more desirable it is to post that constraint. This kind of search heuristics that uses the temporal flexibility retained between each pair of activities to bias the variable and value selection decisions are typically known as *slack-based heuristics* [19].

Algorithm 1 briefly shows the basic steps corresponding to the conflict resolution process previously explained: within the `SelectConflict()` step, a collection of candidate MCSs is computed; the `SelectPrecedence()` step selects the most constrained MCS and the ordering choice to solve it; the `PostConstraint()` step imposes the new leveling constraint within the ground CSP.

## 3.2. Extending ESTA

Some enhancements have been introduced to basic ESTA algorithm in order to adapt it to the specificities of our robot scheduling problem. Let us recall that two types of resources are managed in our problem: on the one hand, binary resources are used to model the experimentation facilities and the robot usage; on the other hand, a cumulative resource is used to model the robot multi-capacity and energy management aspects. We essentially introduce the following modifications to the original ESTA algorithm: the `SelectPrecedence()` and `PostConstraint()` functions that implement the variable and value ordering decisions are able to detect different types of resource contention peaks and impose the corresponding (simple or setup time-bounded) precedence constraint. More concretely, the profile projection analysis now synthesizes two different sets of MCSs separately:

– A first MCS set corresponding to the resource profiles of the experimentation facilities, robot multi-capacity usage and battery. This set is synthesized in the same way than the original ESTA does.

– A second MCS set returned from the contention peaks analysis performed over the robot usage resource. In this case, setup times associated to Load/Unload activities are also taken into account by introducing the underlying rationale of the extended dominance conditions: distances between each pair of activities are analyzed such that a conflict is found if the separation between them is less than the corresponding minimum setup time allowed.

Figure 4 illustrates the different kind of activities related to both binary and cumulative resources as well as some precedence constraints between them.

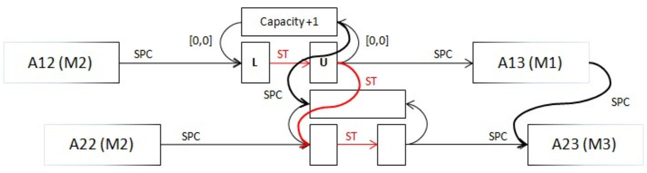

**Figure 4:** Some examples of simple precedence constraints (SPC) and precedence constraints with setup times (ST).

Once both sets of candidate MCSs are computed, they are merged to select the MCS characterized by the least temporal flexibility. Similarly, the solution of the conflict consists in imposing a *simple precedence constraint* if the selected pair of activities belonged to the first MCS set, or a *precedence constraint with a setup time* otherwise with the following exception: if a *crossed conflict situation* is solved (see figure 5), a specific profile projection analysis is performed with the aim of avoiding possible dead-ends. If the cumulative-related activity attached to the target activity constrained by the new ordering is involved in a peak, the opposite precedence ordering constraint is imposed.

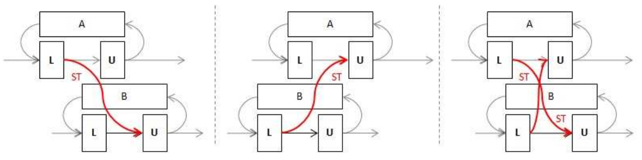

**Figure 5:** The possible crossed orderings that may lead to a dead-end.

### 3.3. Providing better solutions

Since the previous one-pass, greedy solution does not guarantee optimality, we used an efficient optimization framework (in contrast to the costly backtracking-based schemes) with the aim of providing better results. We adopted the advanced IFLAT [14] iterative sampling optimization schema that allows us to minimize the makespans and overcome situations where unsolved conflicts are encountered. The underlying idea is to iteratively run the extended-ESTA algorithm such that different paths are randomly explored within the search space, according to a "Large Neighborhood Search" meta-heuristic strategy [1].

The algorithm 2 illustrates the IFLAT process. The procedure takes two parameters as input: an initial solution

**Algorithm 2**: IFLAT optimization algorithm

```
Input: S, MaxFail
Output: $S_{best}$
$S_{best} \leftarrow$ S
counter ← 0
while (counter ≤ MaxFail) do
    // Retracting step
    RELAX (S)
    // Repairing step
    FLATTEN (S)
    if (Mk (S)<Mk ($S_{best}$)) then
        $S_{best} \leftarrow$ S
        counter ← 0
    else
        counter ← counter + 1
```

*S* and the amount of backtracking (*MaxFail*) that limits the number of consecutively failed attempts at improving the solution. The exploration of different solutions is broadened by a meta-heuristic strategy that basically interleaves the following two steps on each cycle: firstly, a *retraction step* removes an arbitrary number of solving constraints (with a specific retraction probability) from the "critical path" of the last solution; and secondly, a *flattening step* attempts at repairing the "partially destroyed solution" by running extended-ESTA again. If a better makespan solution is found, the best solution ($S_{best}$) is updated and the *counter* is reset to 0. Otherwise, if no improvements have been found within the *MaxFail* iterations, the algorithm returns the best solution encountered.

## 4. THE AUTONOMOUS CONTROL SYSTEM

In this section we sketch a basic autonomous control system that implements a single SPA closed-loop for action scheduling, command dispatching and schedule execution monitoring inspired by the work [16]. The control system is aimed at safely executing the feasible schedule synthesized during the (off-line) initial scheduling phase, by considering the possible disturbances caused by the environmental threats. We consider the following types of exogenous events as possible sources of uncertainty:

– *Activity delay*. An activity is delayed if its execution is postponed a certain amount of time.

– *Duration change*. It is a temporal disturbance that extends the execution time of an activity.

– *Resource breakdown*. Temporary or definitive, partial or total unavailability of one or more resources that are processed in the baseline solution schedule.

Figure 6 shows the main functional building blocks of the autonomous controller.

**The resource-driven solver**. Constitutes the main feature of the controller and is implemented with the constraint-based, heuristic-biased extended-ESTA presented in previous section. It is exploited at the core of

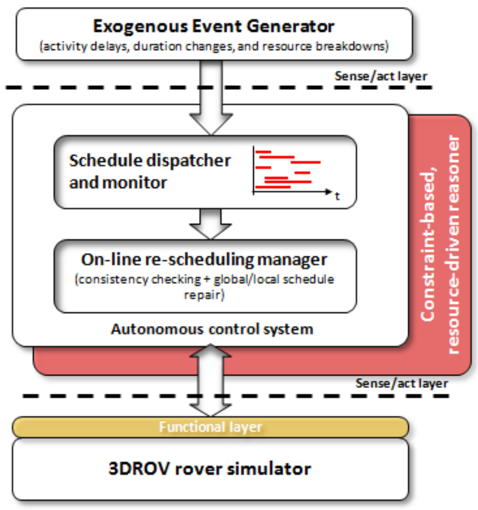
Exogenous Event Generator
(activity delays, duration changes, and resource breakdowns)
Sense/act layer
Schedule dispatcher and monitor
t
On-line re-scheduling manager
(consistency checking + global/local schedule repair)
Autonomous control system
Constraint-based, resource-driven reasoner
Sense/act layer
Functional layer
3DROV rover simulator

**Figure 6:** Single SPA control loop-based scheme.

the execution, i.e., the scheduling step is considered as the main principle to close the SPA control loop. Extended-ESTA is firstly used embedded within the IFLAT optimization scheme to (off-line) provide initial schedule solutions with minimum makespan; and later in isolation to (on-line) regain invalidated schedules during execution. The resource-driven solver is composed of the following modules:

**The schedule dispatcher and monitor**. Contains the current schedule in execution, and timely dispatches the commands by detecting if external disturbances arrive. More concretely, the schedule dispatcher and monitor is in charge of *reading* the schedule at each execution cycle by querying for the time points that match with the current time, synthesizing and dispatching the related robot commands, and *observing* the arrival of new exogenous events.

We identified nine basic commands that represent the beginning and completion of the robot activities (i.e., grasping, releasing and transporting): *StartLoad($\mu_i$)*, *EndLoad($\mu_i$)*, *StartUnload($\mu_i$)*, *EndUnload($\mu_i$)*, *StartMoving($\mu_i$)* and *EndMoving($\mu_i$)*, given a experimentation facility $\mu_i$ or initial and final storage location. As soon as an exogenous event is detected (i.e., activity delay, duration change or resource breakdown), the monitor invokes the on-line re-scheduling manager to reflect such disturbance on the schedule under execution and trigger a reactive re-scheduling step if needed.

**The dynamic re-scheduling manager**. Provides the basic mechanisms for (on-line) temporal and resource consistency checking, and schedule repairing. The algorithm 3 illustrates the execution of a schedule. The procedure takes two parameters as input: the initial schedule solution to be dispatched and the re-scheduling strategy, and exits with a successful or a failed execution report. The algorithm iteratively checks if a new exogenous event arrives and perform the following steps when detected:

**Algorithm 3**: The Execution Management Algorithm of a schedule solution

```
Input: solution schedule S, rescheduling parameters retract
Output: Execution report
while (new E detected) do
    InjectEvent (E, S)
    if (¬(Propagation (S) fails) ∧ ¬(S is resource consistent)) then
        if (retract) then
            RemoveChoice (S)
        S ← ExtendedESTA (S)
        if (S not found) then
            STOP (Fail)
    else if (Propagation (S) fails) then
        STOP (Fail)
```

– The effects of the exogenous event are reflected on the schedule as follows: a new temporal constraint is posted if the disturbance corresponds to an activity delay or duration change; otherwise, if a resource breakdown is detected, a “ghost activity” is synthesized to simulate a lost of a certain amount of resource during a period of time.

– A temporal and resource *consistency checking step* is performed through global temporal propagation. If no inconsistencies are detected, the execution continues as normally.

– If a *resource inconsistency* is detected, a *global or local schedule repairing step* is performed; otherwise, if a *temporal inconsistency* is detected, the execution stops. The local reactive strategy simply calls the re-scheduling process, whereas the global strategy removes all the constraints imposed by the previous solving processes (`SelectPrecedence()`) before re-scheduling.

**The simulation execution facilities**. In order to simulate the command dispatching process we used the simulation environment for rover planetary surface operations 3DROV [15]. The 3DROV simulator allows us to timely send via TCP/IP the robot commands generated by the schedule dispatcher and monitor module, and visually check the execution performance (in real-time) on an advanced simulation environment which represents a rover running over a Martian scenario.

An intermediate functional layer is located between the controller and the 3DROV simulator with the aim of filling the gap between both. The high level commands generated by the controller are translated into the low level robot movements (i.e., in terms of moving distances and rotation degrees), by considering aspects such as the underlying topological layout or the specific parameters of the rover's locomotion subsystem.

Additionally, an *Exogenous Event generator* (see Figure 6) allows us to create and inject a set of a specific set of exogenous events[2] within the controller during schedule execution with the aim of simulating a certain level of environmental uncertainty.

## 5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a basic autonomous control system that implements a single SPA closed-loop for action scheduling, command dispatching and schedule execution monitoring. The controller aims at addressing the problem of "scheduling and running a robot in an unstructured environment" by using an advanced constraint-based, heuristic-biased solving method at its core, both to provide initial schedule solutions and to repair invalidated schedules during execution.

The work here described represents the first steps towards a more general and scalable autonomous execution architecture. Such envisioned control architecture is mainly aimed at integrating planning capabilities and distribute the internal state management competences of the robot along a collection of coordinated SPA control closed-loops rather than on a single one. Communication between control loops is intended to be implemented through distributed synchronization and goal-dispatching mechanisms that will allow them to exchange information and project the evolution of their internal state on the others.

**Acknowledgment**. Daniel Diaz is supported by the European Space Agency (ESA) under the Networking and Partnering Initiative (NPI) *Autonomy for Interplanetary Missions* (ESTEC-No. 2169/08/NI/PA). CNR authors are partially supported by MIUR under the PRIN project 20089M932N (funds 2008). The second UAH author is supported by Castilla-La Mancha project PEII09-0266-6640. Special thanks to Michele Van Winnendael for his support throughout the whole course of the study.

## REFERENCES

[1] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[2] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research*, 112(1):3–41, 1999.

[3] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. A constraint-based method for project scheduling with time windows. *J. Heuristics*, 8(1):109–136, 2002.

[4] C. Cheng and S.F. Smith. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings 12th National Conference on AI (AAAI-94)*, 1994.

[5] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[6] Jeremy Frank and Paul H. Morris. Bounding the resource availability of activities with linear resource impact. In *In Proc. Int. Conf. on AI Planning and Scheduling, ICAPS*, 2007.

[7] Michael A. Goodrich, Dan R. Olsen, Jacob W. Cr, and Thomas J. Palmer. Experiments in adjustable autonomy. pages 1624–1629, 2001.

[8] P. Laborie and M. Ghallab. Planning with Sharable Resource Constraints. In *Proceedings of the* $14^{th}$ *Int. Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[9] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences*, 7:95–132, 1974.

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

[11] Nicola Muscettola. Incremental maximum flows for fast envelope computation. In *ICAPS*, pages 260–269, 2004.

[12] A. Oddi, R. Rasconi, A. Cesta, and S.F. Smith. Iterative-Sampling Search for Job Shop Scheduling with Setup Times. In *COPLAS-09. Proc. of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems at ICAPS*, 2009.

[13] A. Oddi and S.F. Smith. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14th National Conference on AI (AAAI-97)*, pages 308–314, 1997.

[14] Angelo Oddi, Amedeo Cesta, Nicola Policella, and Stephen F. Smith. Combining Variants of Iterative Flattening Search. *Journal of Engineering Applications of Artificial Intelligence*, 21:683–690, 2008.

[15] P. Poulakis, L. Joudrier, S. Wailliez, and K. Kapellos. 3DROV: A Planetary Rover System Design, Simulation and Verification Tool. In *Proceedings of the 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-08)*, 2008.

[16] Riccardo Rasconi, Amedeo Cesta, and Nicola Policella. Validating scheduling approaches against executional uncertainty. *Journal of Intelligent Manufacturing*, 21:49–64, 2010. 10.1007/s10845-008-0172-7.

[17] Hani El Sakkout and Mark Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5:359–388, 2000. 10.1023/A:1009856210543.

[18] Helmut Simonis and Trijntje Cornelissens. Modelling producer/consumer constraints. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 449–462, London, UK, 1995. Springer-Verlag.

[19] S.F. Smith and C. Cheng. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings 11th National Conference on AI (AAAI-93)*, 1993.

[2] An exogenous event is characterized by the time aware, that points out the arrival time of the event, and the associated disturbance effect.