Dynamic Task Scheduling For Advanced Ground Facility

Jean-Clair Poncet, Pierre Guestault
AXLOG Ingénierie
19-21, rue du 8 mai 1945, 94110 Arcueil – France
Jean-Clair.Poncet@axlog.fr, Pierre.Guestault@axlog.fr
Alain Verheyden, Jacques Marée
Spacebel S.A.
5-7 I. Vandammestraat, B-1560 Hoeilaart – Belgium
Alain.Verheyden@spacebel.be, Jacques.Maree@spacebel.be
Giancarlo Pittella
ESA-ESRIN EOP – ED
Via Galileo Galilei, Frascati – Italy
Giancarlo.Pittella@esa.int

Abstract. The development of small satellites is an increasing trend. It enables reducing development and exploitation costs at the platform and payload levels. However, the ground stations required to satellite exploitation could also be modernized using some new work automation approaches. This paper presents the architecture of an Advanced Ground Facility to manage small satellite missions. It emphasizes the automated planning and scheduling aspects of the work, where station activity is managed automatically. The task schedule is built using a dedicated approach that enables dynamic management of the tasks performed. The work is illustrated by preliminary results.

1 Introduction

Future missions, and mostly Earth Observation (EO) missions are likely to be based on smaller satellites (< 500Kg), flying alone or clustered, carrying less payload but more specialized instruments designed for thematic applications. The development of small satellites is cheaper and faster (< 2 years) and benefits from more recent technology than conventional satellite. On the other hand, their lifetime should also be reduced (2 or 3 years in average). The small satellite concept is attractive in a general context of budget cuts and hence makes space more accessible for new entrant actors (commercial or not), operating possibly at national or regional scale.

Obviously, such small missions should benefit from ground segments offering the same qualities: cheaper and faster! However, the reduction of the global operational cost for small missions cannot be achieved solely by reducing the development cost of the dedicated Payload Ground Segment, the operational costs should also be reduced. This might be achieved in different ways which must be adapted depending on the scenario considered, using for instance:

1. Sharing of the infrastructure between several missions: the fixed operational costs are spread over different missions.
2. Unattended operations: less human involvement (day hours only) and shorter end-to-end delivery time.
3. Design in accordance with a (proved) business model: suppress unessential operational costs.

Ground Facility owners need to have an estimate, at an early stage of the system development, of the cost of the ground operations, helping them to build an attractive offer to potential customers.

This implies the design of a new generic Advanced Ground Facility (AGF) with an autonomous decision capability (e.g. because of the more frequent revisiting time hence the higher conflicts' probabilities) and which can easily be expanded to the forthcoming missions.

The AGF interacts with two main actors: the satellite from which it receives the payload telemetry data and the mission center generating demands for acquisitions (acquisition plan) to be executed and issuing specific request for product ordered by the end user.

These actors can be extended to other external entities providing for example the ancillary data required for the thematic processing, the acquisition services as performed actually by the ESA stations, archiving or processing facilities like for the ERS and ENVISAT missions.

The AGF is able to work in a stand-alone environment or within a pool of AGF where a master AGF controls and distributes the activities between its dependent facilities.

The AGF is a modular entity: the architecture is based on a core AGF that can be extended with functions covering specific needs (thematic processing, long-term archiving, etc). These functions can be made accessible internally or externally.

2 Advanced Ground Facility

The AGF functional architecture (see Figure 1) is composed of a station planning function receiving the acquisition requests from the Mission Centers and managing the schedules of the station activities. The Monitoring & Control is responsible for the execution of the schedule: triggering the execution of each activity, controlling their execution and reporting their execution status to the station planning. The acquisition subsystem manages the acquisition of the payload telemetry from antenna RF equipment to the storage of the decommutated instrument source packets. Each mission processor computes, from the raw data and auxiliary data, the high level product that can be directly disseminated to the users.

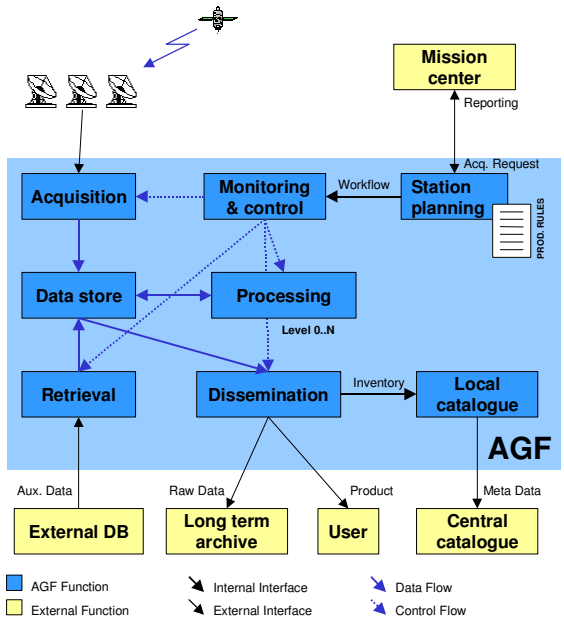

Figure 1 — AGF functional architecture

The design drivers behind this architecture are:

Automation: the station planning subsystem is in charge of the automatic handling of the user requests. The driving of operations is based on the knowledge of constraints (description of the station resources) and priorities.

Plug & Play Approach: at subsystem level, the architecture is based on modular SW components, which can be configured for a particular mission.

This approach gives the ability to benefit from the latest technology evolution or new standards and provide SW independence from underlying HW (portability).

Some components that are dedicated to particular mission are implemented in a plug & play approach: at the installation time, the component registers itself to the system providing information on its capability (description of the provided services) and constraints (performances and resources required).

Workflow: the planning service is responsible for the generation of the schedule of the station activities in a short-term horizon. This generation is dynamic and is automatically started at each occurrence of an event reopening the established schedule (new request issued, resource state modification).

The schedule of the station activities is transmitted to the Monitoring & Control in a workflow expressed in the BPEL[1] standard format.

The usage of this format allows specifying precisely the execution dependency of the station tasks and thus avoiding idle state during the schedule execution.

The AGF is devoted to fulfill as efficiently as possible these three major requirements, using automatic dynamic scheduling to increase automation level.

3 Station Planning

When looking at the AGF network, there are two different levels of planning that are considered. On the one hand, at the network level, the activity of each AGF station is managed according to tasks submitted to the network. Of course, a coherence of the dissemination of task realization onto the different stations is required.

On the other hand, at the station level, the set of tasks devoted to the station is ordered to satisfy the different requests in the shortest time, while optimizing the resource usage when possible.

In the following, we focus on the second aspect of the problem, dealing with scheduling of requests at the station level.

3.1 Station System Architecture

The Station Planning (see Figure 2) is implemented as a Web Service. It receives commands from the Mission Centers and operators. It handles also the notifications sent by the Monitoring & Control.

Notifications about the execution of the requests are asynchronously returned to the Mission Centers.

---

[1] Business Process Execution Language

The product requests are processed by the "Tasks Generator" that uses inference rules to break the request into a list of constrained activities: a chain of tasks that perform the product and make it available at the end. At this level, the decomposition of the request is achieved in a straightforward way (no selection based on user profile or priority rule). The resources that are involved remain at a certain level of abstraction: for example, the resource type is identified but not the device (computer, antenna, disk, network, etc) that is used at the end.

This list of activities is forwarded to the planning engine that "binds" resources and devices using the Planner/Scheduler, adding time constraints to avoid bottlenecks. The planning is done according to available resources and their associated performances. The resource manager provides this information. For a same incoming list of tasks, there can be several possible solutions; an optimization criterion (time based or resource based) is specified to select the most suitable one according to scheduling policy.

The operator can define "MODES", expressing the AGF policy at a given moment. The modes are conditioned by a temporal applicability (for example from a given date, or during weekend)

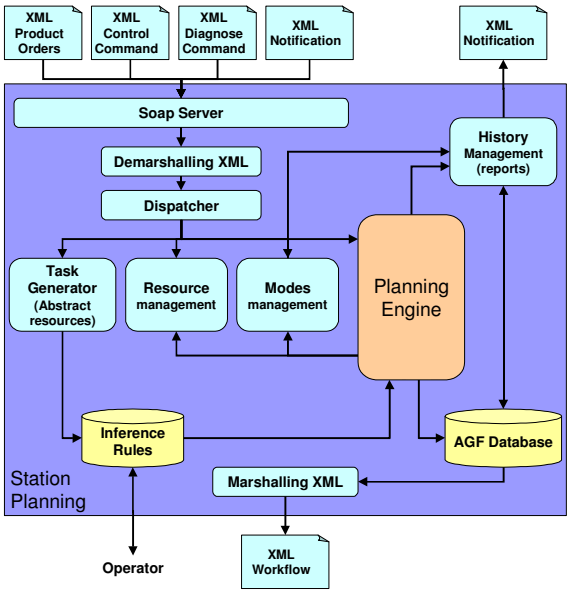

Figure 2 — Station Planning Architecture

The resource manager is regularly notified of the resources availability and asks the corresponding modules for recording the resource performances. The Mode Manager can modify the panel of resources that the planning engine can use at each moment in the schedule horizon, depending on resource modes. Typically, a resource can be marked as "not usable" during a given period of time, even if the resource is actually available.

Finally, the History Manager receives diagnostic or report requests and uses the AGF database to build an answer. The History Manager also receives from the planning engine a plan for statistics evaluation. This module is also responsible for storing the notifications received from the Monitoring & Control and for dispatching them into the AGF database and to the main AGF service.

3.2 Planning Architecture

The planning engine is composed of three main software components (see Figure 3):

- The report manager is in charge of generating reports on the AGF activity according to past and current task schedules.
- The planning engine generates some new task schedules when called by the Supervisor.
- The Supervisor is in charge of managing incoming requests. When some requests are submitted, the Supervisor decides on the computation of a new plan, when computing a new plan and how this plan is computed, using a global schedule approach or a plan repair algorithm.

The supervisor is also in charge of monitoring at high level the plan that is currently executed in the station. It can decide to change this (and if necessary to call the planning engine) according to possible drifts in plan execution.

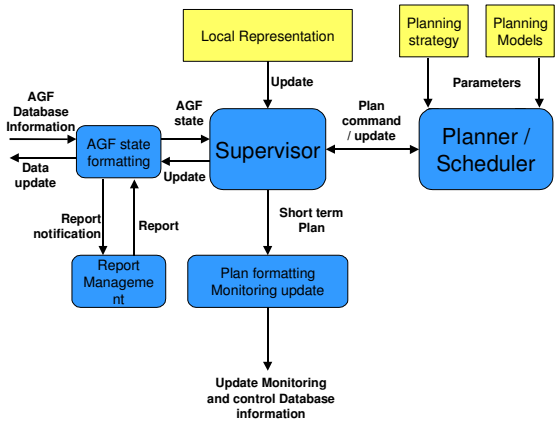

Figure 3 — The Planning component architecture

Among these three components, only the Supervisor is interfaced to other components of the AGF. These interfaces are all directed to the AGF database. The Supervisor polls the database to be informed of new requests stored in it. It also writes its proper data in this database, so that the rest of the AGF can view them.

4 Dynamic Scheduling

4.1 Problem Statement

In the AGF, the problem consists in scheduling some elementary tasks and in allocating specific sets of resources to them. This is close to classical jobshop-scheduling problem [1]. However, considering the structure of the workflows of elementary tasks associated to the different product realizations, it appears that some parts of the schedule can be conditioned by the results of previous tasks. Moreover, some tasks, e.g. acquisitions, have some strict start and end dates that cannot be moved. The following 9 points can summarize the global scheduling problem:

1. The basic problem consists in scheduling sets of elementary tasks according to their dependencies. These dependencies are stated by the realization rules of the product to which they belong.
2. Some resources (processor, data storage, antenna) are affected to the tasks according to the needs.
3. Some tasks could be conditioned by the success or the quality of results of a previous scheduled task. This creates some alternatives (choice points) in the AGF schedule that must be managed.
4. Some tasks are dependent on conditions external to the AGF. For instance, acquisition is dependent on the passage of the satellite over the horizon of the antenna. Then the scheduler enables the right resources at the right time to perform these tasks.
5. Tasks have some priority orders and deadlines according to their associated product priority and nature.
6. The scheduler works on different time horizons. A weekly schedule enables to consider AGF possibilities for long term. A short-term schedule (hourly) enables to manage the AGF daily.
7. Some tasks need resource initialization before their execution. For example, some parameters of an antenna must be initialized (e.g. orientation) to make a data acquisition of a given satellite. This implies time constraint between data acquisition start time and the current time.
8. Multiple resources can be involved in one task execution. For example, CPU processor and bandwidth resource are needed to the execution of a dissemination task.
9. The AGF owns a heterogeneous resources pool. AGF resources can be disjunctive (only one task can be executed at a time), cumulative renewable (the quantity of resource consumed is restored at the end of the task) or consumable.

Among these scheduling constraints, the most complex is the possibility of conditional plans according to tasks results. It is obvious that the resulting choice points increase combinatorial complexity; each possibility has to be explored when building the future AGF activity plan.

4.2 Scheduling Conditional Tasks

Introducing conditional tasks have an important impact on the way to produce the AGF plan of activity. Obviously, according to a set of tasks to schedule, different valid schedule exist. A selection can be made among these schedules using an optimization criterion (time or resource usage for instance). By considering choices points, the complexity is dramatically include. For instance, considering the short example in Figure 4, where two possible schedules of the two flows are shown. In the first one, the task B succeed and task C is executed after. In the second one, the task B failed and task D is executed. Finding a schedule needs to explore all the possibilities in tasks arrangements and in choice points evaluation. Moreover, the effective result of the different choice points is known only at execution time. This means that we can not *a priori* determine the optimal schedule as we don't know exactly what set of tasks will be executed.

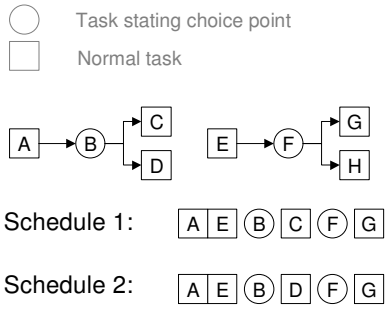

Figure 4 — Short schedule example

Generally speaking, we consider that a choice point resulting from a task result corresponds to three alternatives at most: value *0* (success of the task), 1 (failure of the task) or 2 (other alternative; e.g. lower quality of result). This restriction matches the specification of task flows and reduces the complexity by maximizing the number of different choices.

Another constraint stated in the specification was to maintain good system reactivity; it is necessary to perform task scheduling in a reasonable time (less than ten minutes on a standard desktop computer). This means that we will certainly not be able to determine all the possible schedules according to possible tasks results before starting execution.

In order to match the conditions introduced in the schedule by some elementary tasks, the final AGF activity plan[2] is computed one branch after the other according to the tree representation of the plan. This is performed using the algorithm in Figure 5.

This algorithm starts by computing a first schedule corresponding to the case of each choice point is evaluated to 0 (expected success of the all the tasks). This first schedule enables to fix an order on the different choices points. Once we have this first schedule, we can determine the other parts of the plan, meaning the possible schedule in case of one or more choice points are not evaluated to 0.

The objective is to be able to start to execute the first computed schedule while continuing to determine alternative ones. Algorithm so computes first the alternatives schedules that are the earliest in the plan to feed the execution when necessary. The process ends when all the possible alternatives have been taken into account, determining the complete AGF plan.

```
Begin
    Compute the schedule in case of all
    choice points have value 0.
(1) If
    (It exists a part of plan not yet
    scheduled)
    Or
    (It exists a task not yet scheduled)
    Then
        Select the part to schedule next
        according to earliest date
        Determine the set of tasks to
        schedule on this part
        Schedule this part of plan
        Return to (1)
    End if
End
```

Figure 5 — Algorithm

This iterative algorithm enables to maintain a good reactivity of the system. Next section illustrates its behavior on a practical example.

4.3 Scheduling Example

To understand how this algorithm works, let us consider the example of Figure 6.

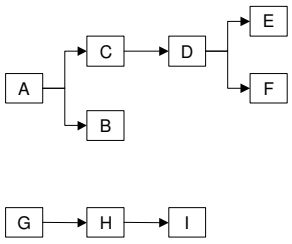

Figure 6 — Workflow example

We consider that the current schedule plan consists in the first workflow including the ordered task set *[A, B, C, D, E, F]*. In this workflow, a choice point is set according to the result of task *A*. For instance, let us assume that *A* corresponds to an acquisition; if this acquisition succeeds, then process level 0 data (task *C*), process level 1 data (task *D*) and according to results of processing, disseminate (task *E*) if success, or erase data (task *F*) if failure. The schedule corresponding to this workflow is illustrated in the first plan of Figure 7.

In this current schedule plan, we want to introduce the ordered task set *{G, H, I}* that corresponds to the product sequence of a new user request.

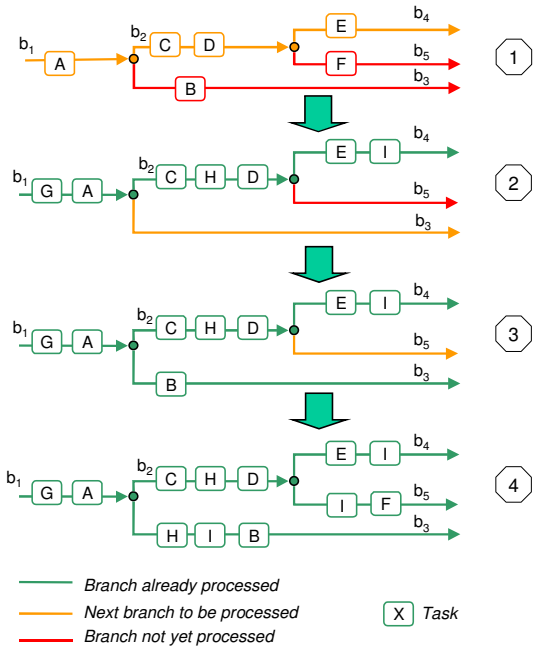

Figure 7 — Conditional flows scheduling

According to the algorithm, some tasks have to be inserted in the schedule, so that the complete graph of possible schedules have to be processed again to insert them. The next step determines the path made of branches *{b_1, b_2, b_4}* (sample 2 in Figure 7). All the tasks are taken into account. A possible resulting schedule in case of task *A* and *D* success is *[G, A, C, H, D, E, I]*. Note that it is only an example of possible schedule that can be returned

[2] In this approach, we call "plan" the complete set of possible schedule that should be computed according to choice points evaluation in order to realize all the process.

according to real tasks resource usage, dependencies, etc. However, to fix the ideas, let us say that the scheduler returns the solution *[G, A, C, H, D, E, I]*.

Once this step is complete, we iterate the algorithm. Some branches have not been recomputed: b_3 and b_5 (sample 2 in Figure 7). According to the earliest date, the next part to schedule is $\{b_3\}$. It corresponds to the case of *A*'s failure. A possible schedule is for instance: *[G, A, H, I, B]*. Note that the two computed schedules have the part *[G, A]* in common. The structure of this part is stated as a hard constraint when computing branch $\{b_3\}$ in order to ensure that the different alternative plans are compatible.

The last step consists in determining the schedule for the last branch, $\{b_5\}$, corresponding to the last possibility in choice point in which *A* succeeds and *D* does not. This last schedule is *[G, A, C, H, D, I, F]* (sample 4 in Figure 7). Once again, the part made of $\{b_1, b_2\}$ is common to both schedules.

In this example, the new workflow does not add additional choice points. But in some cases it does: the process is exactly the same but the number of possible plan is increased. The position of choice points is determined by the first plans to be scheduled and reused to compute other plans.

According to this fix choice point policy, the algorithm could give non-optimal solutions on the last branches to be scheduled. However, it enables dynamic and efficient re-scheduling of AGF activities while maintaining the deadline constraints stated on previously scheduled tasks.

4.4 Scheduling Models

The schedule generation is based on a constraint programming approach. The scheduling model is then described as constraints to fit the implementation approach. The model relative to classical jobshop scheduling problems [2] is not explained, only the model relative to problem specificities is.

To model the tasks behavior (especially, the processing tasks) we must take into account that their resource consumption could change over time according to the number of tasks executed at the same time on the resource. We also need a good approximation of their duration to evaluate the overall schedule time according to task dependencies. Instead of models based on an upper approximation of the task duration independently from its resource usage, we prefer another one taking into account this dependency. Thus the duration of task is modeled according to the amount of resource that is allocated to it.

The model is based on a task-centric approach of the problem. All variables instantiated during the schedule computation are associated to the tasks. For each one, we specify:

- The resources associated to its execution:

  $[R_1,\ldots,R_n]$, where R_i is the resource identifier of the i^{th} resource, on which is executed the task,

- At all time stamp in the schedule:

  The status of the task (currently scheduled or not),

  The foreseen resource consumption.

4.5 Time Stamp

In order to represent these data in a discrete environment — only using finite domain variables — we define a time stamp. However, instead of defining some regular time step along the schedule horizon, we define a time stamp based on task scheduling events, as shown in Figure 8. In this figure, 4 time stamps are identified corresponding to the different time events at which resource usage change [5].

The main advantage of this approach is that it reduces the overall number of time steps without under-constraining the problem, as the resource consumption of a task only changes when another task execution is started or stopped. Nevertheless, this dynamic specification is also the main drawback, as the time stamps definition is made only at variables instantiation time.

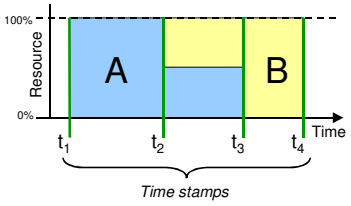

Figure 8 — time stamp

The data associated to tasks are then the following:

- The task's time bounds: T_s, T_e.
- The matrix:

$$\begin{bmatrix} Cap_{(1,1)} & & Cap_{(1,n)} \\ & Cap_{(i,j)} & \\ Cap_{(k,1)} & & Cap_{(k,n)} \end{bmatrix}$$

Where $Cap_{(I,j)}$ is the resource consumption at time stamp i on the j^{th} resource.

4.6 Resource Consumption

The task duration model must express that the duration of the task depends on the resource consumption. Figure 9 illustrates this property.

On this figure, *D* represents the minimum duration of the task, corresponding to duration required to perform the

task if the entire resource is affected to its execution (hatched blue area). The real resource assignment represented by the brown shape must be such that the brown area is equal to the hatched blue area. The length of the brown shape determines the real duration of the task according to resource allocation.

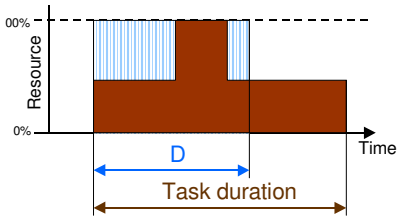

Figure 9 — Task duration

Using the variables introduced before, a definition of the task duration is given by:

$$\forall j, \quad \sum_{i=1}^{\text{Nb time steps}} (Cap)_i^j * T_i = D$$

$$\forall j, \forall i, \quad (T_E < T_i) \vee (T_S \geq T_i) \Leftrightarrow (Cap_i^j = 0)$$

Using the same variables, the resource consumption model can be defined, stating that:

- The whole resource must be fully consumed at every time.
- All tasks consume same quantity at any time.

Thus, two processing tasks executed at the same time on a given processor are supposed to use the same amount of the resource (50% CPU). Identically, two dissemination tasks take 50% of the bandwidth each. This is given by the following formula:

$$\forall j, \forall i, \quad \sum_{t \in \text{Tasks}} (Cap_t)_i^j \in [0,100]$$

$$\forall j, \forall i, \forall t, \quad (Cap_t)_i^j = K_i^j$$

For each task *t*, for each resource *i* at each time stamp j, the sum of resource capacity used is between *0* and *100* resource units. And for each task *t*, each resource *i* and each time stamp *j*, the instantaneous capacity used depends only on the resource and the time stamp.

4.7 Search Strategies

The resolution mechanism of the constraint programming approach takes the complete problem and tries to solve it as a whole [3]. However, as previously said, user requests may be submitted to the system at any time. Taking these requests into account in the schedule is equivalent to inserting new tasks in the current schedule. A plan-improving algorithm, also called plan repair algorithm, can be very efficient to solve this kind of problem.

The plan repair algorithm for this problem is based on the improvement of an existing solution, valid or not. It tries, by using a partial search method, to find a valid solution close to the existing one. The improvement mechanism is based on:

- Adding new tasks to the initial schedule.
- Moving or deleting tasks already present in the plan in case of invalid initial solution.

In addition to this plan repair approach, a specific constraint inference algorithm has been implemented. It is based on inference over the tasks duration to improve propagation of variables during resolution. Figure 10 shows an example of this constraint application. The duration D represents the minimum duration during which tasks *A* and *B* share the same resource. The constraint infers on the duration of these two tasks according to the associated resource allocation. In the example, during the period *D*, tasks *A* and *B* equitably share the resource. Their duration is increased of *D/2*.

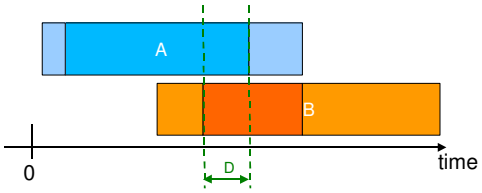

Figure 10 — Task duration constraint

5 Schedule Control and Supervision

As previously explained, the role of the Supervision component is to check the right execution of the current schedule, and to decide when and how it is necessary to modify it.

Figure 11 describes the architecture of the Supervisor.

The schedule monitor permanently polls the database table containing the current task execution information. It builds a representation of the current system state and transmits it to the planning control. The planning control analyses this state and decides whether it is necessary to change the current task schedule (i.e., to compute a new schedule by calling the planner).

The planning control uses a rule-based system to perform the decision-taking activity. This system is based on the RETE rule inference algorithm. This algorithm decomposes a set of rules into a directed graph that allows quickly inferring of the right-hand sides of these rules. In

planning control, these rules are used to determine whether a new schedule has to be computed, and how this is done.

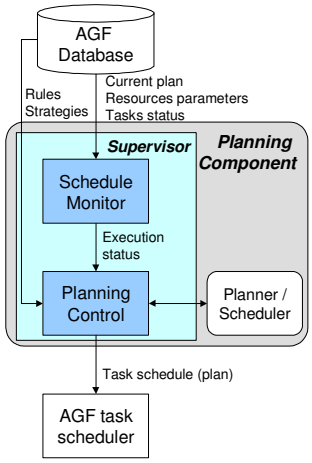

Figure 11 — Supervisor structure

The rules are written in a Prolog-like style, as depicted in Figure 12. The tail of the rule represents the premises. They can be structured using conjunction marks (with coma character), or disjunction (using semicolon character). The *head* of a rule represents the consequence. Having a single goal in the head does not limit rule expression, as several rules using the same premises can be specified. The rules can also be used to change task priorities according to deadline, changing spare resource policy, etc.

The rules are gathered using rule sets, also named configurations. These configurations determine the system policy. The different rule sets are managed in a configuration stack. Each time a configuration is added to the system, it subsumes the previous one. Each time a configuration is removed, the previous one is restored. A basic system configuration is given which cannot be removed.

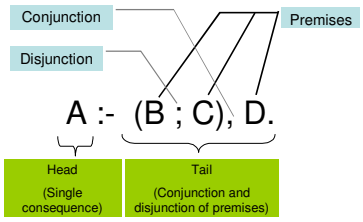

Figure 12 — Scheduling rule structure

6 Experiment Results

A prototype has been implemented using the Finite Domain library of SICStus Prolog. As the system is currently under development, it has not been fully tested yet. However, early made in first time on simple examples, and further on more representative sets of tasks show a good solution, near the optimal one, is found in few minutes. This time scale is convenient as the time to reschedule the system is evaluated as a tenth of minutes, which augurs satisfying system behavior on the real set of tasks that is still to be determined.

The data related to the most significant example are the following:

- A pool of 6 resources of 3 different types is defined, whose properties are described in Table 1 below.

RESOURCES	Renewable	Type	Capacity
1	No	1	10
2	No	1	12
3	Yes	2	16
4	Yes	2	12
5	Yes	2	20
6	Yes	3	12

Table 1 – Example's resource data

- 39 tasks have to be scheduled. Each of them has an initial time delay after which they must necessarily be executed. They also have a strict deadline date. Each task can only be processed using a resource of one of the three types. Among the 39 tasks, 8 have to be processed using resources of type 1, 21 using resources of type 2 and 10 using resources of type 3.
- The time horizon of the schedule is fixed to 1000 time units (a time unit is equivalent to a 20 seconds).
- The optimization function maximized the number of tasks included in the schedule, trying to schedule as many tasks as possible while respecting activation times and deadlines.

	First solution	Solution #8	Solution #14	Optimal Solution
Computation time	1	38	142	584
Cost optimization criterion value (minimization)	-5	-100	-150	-165
Number of scheduled tasks	1	20	30	34

Table 2 - Search results (times in seconds)

The results in Table 2 correspond to an execution of the algorithm implementation on an x86-linux based

environment installed on a 2.4 GHz CPU platform. The solutions #8 and #14 are found during the search; they are exposed to illustrate the evolution of the solution quality according to search time and optimal solution.

In this realistic example, the best solution is found in less than 10 minutes, which is a correct time according to expected system performances. However, the algorithm gives a solution with quality ratio near to 91% in only 2 minutes and 22 seconds. This lets augur very satisfying system performances for dynamic rescheduling, taking into account system variability rate and task durations that are expected to be several minutes or several tens of minutes.

Future work will consist in implementing a repair algorithm based on task selection and time value selection heuristics to decrease even more the search time needed to obtain the optimal solution. This shall satisfy the capability of system evolution in stressing conditions (with higher variability rate of its environment).

7 Conclusion

In this paper, we have presented an automated approach to activity management of an AGF. It enables decreasing human intervention in the deployment of ground-based infrastructure for small missions. The main advantages would be then to make ground station for small missions cheaper and potentially usable by a large scope of users.

The proposed approach to scheduling problem is based on existing techniques, suited to particular ground station management problem. It encompasses the schedule plan generation and the control of schedule execution. The first results gathered while developing the Planning Service are encouraging and augur that the final prototype of AGF is going to be able to manage the ground station in an efficient way.

Based on generic scalable models, this approach could also be enlarged to the whole network of AGF to implement efficient ground station network, providing end-to-end services to final users. This could be an opportunity for interesting future work in this domain.

8 Acknowledgements

The work presented in this article is funded by ESA/ESRIN under contract number 15831/02/I-SB. This project is carried out by a consortium consisting of Kongsberg Spacetec (Norway) and Axlog (France) under the lead of Spacebel.

Thanks to Nelly Strady-Lécubin and Philippe Morignot for useful comments on first drafts of this paper and/or insightful discussions and/or design and implementation help.

8.1 References

[1] Anant Singh Jain & Sheik Meeran (1998); *A State-Of-The-Art Review Of Job-Shop Scheduling Techniques*, at http://citeseer.nj.nec.com/jain98stateart.html

[2] Claude Lepape (1985); *A Daily Workshop Scheduling System*, in Expert System, vol 85, pp. 95-211.

[3] J.K. Lenstra, Kan Rinnooy, A.H.G. and P. Brucker; *Complexity of Machine Scheduling Problems*, in Annals of Discrete Mathematics, vol 7, pp. 343-362.

[4] J.K. Lenstra and Kan Rinnooy, A.H.G; *Computational Complexity of Discrete Optimisation Problems*, Annals of Discrete Mathematics, vol 4, pp. 121-140.

[5] A. Aggoun and N. Beldiceanu*, Extending CHIP in order to Solve Complex Scheduling and Placement Problems*, Mathl. Comput. Modelling, vol. 17, no. 7, pp. 57-73, Pergamon Press Ltd., 1993.