RAPID AND ADAPTIVE MISSION PLANNER FOR MULTI-SATELLITE MISSIONS USING A SELF-ADAPTIVE MULTI-AGENT SYSTEM

Jonathan Bonnet
IRT Saint Exupery, Toulouse, France, jonathan.bonnet@irt-saintexupery.com

Marie-Pierre Gleizes
IRIT, Université de Toulouse, France, gleizes@irit.fr
Elsy Kaddoum
IRIT, Université de Toulouse, France, kaddoum@irit.fr
Serge Rainjonneau
IRT Saint Exupery, Toulouse, France, serge.rainjonneau@irt-saintexupery.com

A constellation of observation satellites allows to cover a large Earth surface, with a good revisit frequency, ensuring different kinds of pictures and the robustness of the system. Planning a mission for a constellation is a complex task: a lot of parameters and constraints, often contradictory, must be taken into account. This huge number of entities make this problem highly combinatorial. Nowadays, the number of constellations of satellites drastically increases, as the number of satellites that compose them (i.e. Google Skybox project). Such a system must dynamically take into account new requests, but this dynamism cannot be taken into account in current approaches.

This paper contributes to this challenge with a new way to plan on-ground the mission of satellites: the ATLAS planning system (Adaptive saTellites pLanning for dynAmic earth obServation). ATLAS is an Adaptive Multi-Agent System, designed to plan missions of constellations of Earth observation satellites. The proposed system brings a major contribution: it is an open and continuous planning system. It has the capability to handle in real-time changes of constraints and/or new request arrivals. ATLAS possesses self-adaptation mechanism in order to locally self-adapt itself according to the dynamic arrival of requests to plan. Thus, ATLAS can dynamically reorganize the mission plan in order to propose a better one (integrating the changes). Because changes are made locally, the whole plan is not challenged and the new plan is provided in a reasonable time. ATLAS can also be stopped at any time and provides a good mission plan. Indeed, the system globally makes the mission plan by local interactions. To enable this capability for real-time adaptation, we use the Adaptive Multi-Agent Systems theory (AMAS). Such systems naturally provide self-adaptation capabilities required to solve this kind of problem. To design our system, we rely on the Adaptive Multi-Agent System For Optimization agent model, providing some design patterns to solve optimization problems using AMAS. In this model, agents are designed as close as possible to the natural description of the problem entities.

Finally, a comparison with a classical greedy algorithm, commonly used in the Europe spatial domain, highlights the advantages of the presented system.

I. INTRODUCTION

A constellation of agile observation satellites allows to cover a large Earth surface, with a good revisit time, ensuring pictures with different characteristics and the robustness of the system. Planning an observation mission for a constellation of agile satellites is a complex task: a lot of parameters and constraints, often contradictory, must be taken into account. This huge number of entities makes this problem highly combinatorial. Currently, in Europe, a satellite system works on a chronological basis relying on regular "appointments" between satellites and control ground stations. During this "appointments", the already computed and validated work plan is uploaded on-board. The clients' requests arriving in the system are stored and before each "appointment", they are taken as input of a planning algorithm. If a request arrives after the start of the planning, it is stored for the next programming period. Thus, currently used planning algorithms, such as greedy algorithms [1], fail to take into account dynamic changes in a short time and to properly manage linked requests (for example requests that require several acquisitions like stereoscopic ones).

Moreover, Earth observation is subject to an important evolution. Nowadays, the number of constellations of satellites drastically increases, as the number of satellites that compose them. For example, the new Google Skybox constellation project plans to amount to more than 24 satellites in 2017, encouraging near real-time client requests to grow drastically. Thus, there is a need for a near real-time planning system. Such a system must dynamically take into account new requests during runtime, which is not the case of currently used approaches ([2], [3] and [1]).

This paper is organized as follows. Section 2 describes the problem and presents its context. Section 3 develops the functioning of the ATLAS system (Adaptive saTellites pLanning for dynAmic earth obServation). Finally, Section 4 presents an evaluation of the ATLAS system for near real-time planning and its comparison to ChronoG, a greedy algorithm.

II. MULTI-SATELLITE SCHEDULING PROBLEM

In this section, a description of the problem and a summary of solving methods currently used are presented.

II.I Problem Description

As the planning problem is a complex and difficult problem, we propose a description that helps us to design our system. Based on the work presented in [4], the considered problem can be described as follows:

A set of Earth observation satellites called **constellation** Sat = {S1,S2,..,Sn} where each satellite Si has its own characteristics:

- a quasi-circular orbit around the Earth,
- an energy management module,
- a storage capacity,
- a payload, in this case observation instruments (optical or radar), and their characteristics,
- an orbital and attitude control system, allowing the satellite to control its position and pointing (towards the area to be observed).

A set of clients' **requests**. Each defined by:

- the type of the request (optical or radar for example),
- a submission date,
- a temporal validity range (from hours to several days or weeks),
- a geographic area,
- a priority given by the customer,
- the tolerated cloud coverage, w, $(0 <= w <= 1)$,
- a set of meshes, associated to the request.

The geographic area defines the earth zone requested by the client. This area can be large: a country or a continent, for example. Satellites cannot necessarily acquire the whole request in a single shot. Thus, requests are divided into a set of meshes.

A **mesh** is the elementary entity that a satellite can acquire in a single shot. Each mesh possesses the constraints of its request, but it is defined by a smaller geographic area. Thus, a request is satisfied if all its meshes are acquired. The decomposition of a request on a set of meshes is outside the scope of the work presented in this paper, we suppose that this decomposition is already made.

To each mesh, one or several **access(es)** matching to a period where the mesh is visible by the satellite are defined in among, using satellite ephemerids. It is during one of these accesses that the acquisition must be performed. The duration of the acquisition slot is usually drastically smaller than the duration of the access. Those accesses are computed by an external module.

To those entities, **a set of hard and soft constraints** must be taken into account. The set of hard constraints (validity range of the request for example) must be satisfied. The set of soft constraints (constraints whose operator can tolerate degradation like the cloud coverage), can be released.

The objective of the problem is to provide a mission plan where the maximum number of meshes are affected to satellites. The satellites must ensure the coverage while:

1. satisfying all the hard constraints,
2. maximizing the satisfied soft constraints,
3. maximizing the number of planned meshes.
4. taking dynamically into account new requests or constraints.

The problem is dynamic. Thus, the whole set of requests to plan is not known at the beginning: requests (and so meshes) arrive during runtime. In addition to this, constraints can be dynamically changed (or added or removed). For example, new weather forecasts are taken into account during planning.

In this work, the problem is solved using a decentralized and distributed multi-agent system where agents cooperate locally to reach their own goals. The global objective of the problem is not known by the agents, it emerges from their local interactions.

II.II Mission Planning: a Survey

Its important dynamic, its large number of entities and constraints interacting together make so that we consider here a complex optimization problem, with a high combinatorial level. In [1], the problem is categorized as **NP-Hard**. In literature, several studies investigate classical optimization approaches to solve it. Most commonly used or studied approaches are exposed here and compared with their capacity to provide a mission plan while handling dynamic perturbations, for example insertion of new requests to plan.

As described in Section II.I, the planning problem can be formalized as a constraint satisfaction problem (**CSP**). Nevertheless, and because of its complexity, it is hard to propose a correct mathematical modeling for the entire problem. Thus, it is necessary to simplify it [1]. To perform this simplification some constraints can be relaxed: the duration of the planning horizon can be shortened or the modeling of the satellite's agility can be lightened. Even if the simplified problem is correct, the obtained solutions remain sub-optimal. Once modeled as a CSP, it is possible to solve the problem using **exact resolution methods**. In several papers [4], [5] and [6], efficient tools like *ILOG Solver* have been used. [7] studied **Dynamic Programming** to decompose the problem in simpler sub-problems. Given the difficulty to decompose recursively the problem, the usage of those algorithms is limited. Moreover, in case of requests requiring an history, the algorithm cannot be applied. This problem occurs for example for stereoscopic acquisitions: several acquisitions of a single target are linked, preventing to decompose the problem. In [8], the same critics are established to **Branch and Bound algorithms**. Indeed, even if all solutions are not explored, algorithms based on Branch and Bound technique use problem properties to guide the search, consume a lot of memory, and require potentially high execution time to reach a solution.

The large number of constraints and parameters make these approaches extremely slow. In addition to this, any modification like the introduction of new requests, imposes to re-establish the search tree. These methods are not suitable to handle the high dynamic level of the studied problem.

To overcome these drawbacks, **approximate methods** can be applied. A good solution can be produced in a reasonable time thanks to the use of a heuristic function.

The most used algorithm, in Europe, is the **Greedy Algorithm**. This last allows a fast execution and its implementation is quite easy depending on the implemented variant. Several versions exist but the principle is the same: the algorithm crosses the requests and when it makes a choice, this one is never challenged. Thus, variations concern the way to look over the requests pool. The most used families are:

Chronological Greedy Search [2]: the plan is chronologically constructed. At each step, the algorithm searches for the more pertinent mesh (acquisition) to schedule (generally the one with the highest priority) at the end of the last scheduled one respecting precedence constraints.

Hierarchical Greedy Search [9]: the first step consists to sort the pool of meshes using several criteria: priority, image quality, weather information, etc. Then, the list is unstacked: each mesh is inserted at the best place, respecting its constraints.

Genetic algorithms as presented in [10] or [8] can also be used. The main underlined limits of these approaches are the modeling and the slowness of their execution (several hours), that is prejudicial in order to quickly produce a good solution.

All these algorithms, even if they produce quite good results, have limitations. When adding new requests during runtime, the process must be resumed to the first possible visibility access of the meshes of the added request. Indeed, operational systems work step by step: during the building of the plan, no request can be added. In addition to this, those algorithms are generally designed to plan the mission of a single satellite. Thus, they don't profit very well of the advantages of a constellation in order to increase the planning capacities.

Recently, **self-organizing systems** such as adaptive multi-agent systems have proven valuable for solving highly dynamic problems, thanks to their ability to adapt themselves to their environment. For example, [11] proposes an adaptive multi-agent approach to provide self-regulated manufacturing control. Today, such systems are applied in a multitude of domains, like smart grids and management of electric vehicles [12] or self-control of heat engines [13]. As it is close to the natural description of the problem, this approach allows to define solving algorithms which are then more robust to dynamics, flexible, open and provide a relevant level of adaptation in real-time.

In this work, we focus on the usage of self-adaptive multi-agent systems as defined by the **AMAS Theory** (Section III.I), in order to plan on-ground the mission of a constellation of satellites.

III ATLAS PLANNING SYSTEM

In the presented work, we propose to use adaptive multi-agent systems to plan missions for a constellation of Earth observation satellites in near real-time. This approach allows to naturally consider a large number of entities and constraints while providing self-organization mechanisms to handle the problem dynamics. In this section, we present the AMAS approach and the contribution of this article, the ATLAS system.

III.I AMAS Approach

In the AMAS theory (**A**daptive **M**ulti-**A**gent **S**ystems) [14], agents are considered as autonomous and cooperative entities, having a partial knowledge on their environment and searching to reach a local objective. Agents of these systems interact locally in a **cooperative** manner producing partial functions. Cooperation is defined as the capacity of the agents to work together in order to reach a common objective. Thus, any activity between agents is complementary and solidarity links exist between them. Using cooperation, the system self-adapts to stay in cooperative state. The cooperation of all parts of the system makes the adequate function the system was designed for "*emerges*" (Figure 1).

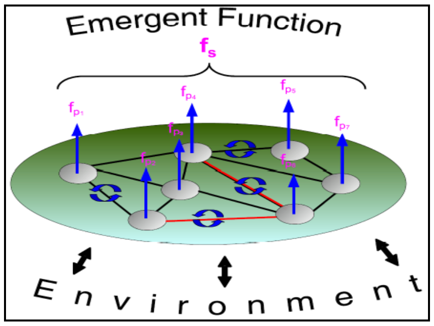

Figure 1 Emergent Function

Local interactions allow the system to self-adapt to perturbations and so to handle dynamics without challenging the already reached solution. Perturbations produce "**Non Cooperative Situations**". To repair those situations, agents possess mechanisms to autonomously adapt their behavior to the context [14]:

- **Tuning**: the agent adjusts its internal state to modify its behavior,
- **Reorganization**: the agent modifies the way it interacts with its neighborhoods,
- **Evolution**: the agent can create other agents or self-suppress when there is no other agent to produce a functionality or when a functionality is useless.

The algorithm of an adaptive agent can be described as follow: if a Non Cooperative Situation is detected, agent uses one or more self-adaptation mechanisms to come back to a cooperative state where it performs its nominal behavior.

To ease the design of agents' behavior and interactions for solving optimization problems under constraints based on the AMAS Theory, the **AMAS4Opt** agent model has been proposed [15]. This model provides design patterns for two cooperative agent roles: "**constrained role**" and "**service role**". One agent can have one or both roles and switches at runtime between them depending on the situation it faces. The agents having the "constrained role" manage the constraints and must be satisfied, while the agents having the "service role" are skilled to help the agents under the "constrained role". This model uses the notion of so-called criticality of agents with the "constrained role" as an engine for the cooperation between agents. We have used and extended this model to design the agents and the general architecture of the ATLAS system.

III.II ATLAS Architecture and Functioning

Given the problem description, we identified nine kinds of entities:

- three types of cooperative agents: the request agent, the mesh agent and the satellite agent (their behaviors are detailed later in this paper),
- three types of active entities: the cloud coverage, the solar ephemeris and the downloading station. These entities do not have a goal to satisfy, but they still interact and influence agent activities and decisions.
- three types of passive entities: the memory of the satellites, their battery and the module in charge of attitude and orbit computation. These entities are considered passive as their state is modified by other entities. For example, the satellite modifies the remaining available space of its memory when it performs an acquisition.

The AMAS4Opt model provides patterns to define the agents' behavior and interactions. Mesh and request agents possess constraints to satisfy: they must be planned under certain constraint, they have the "constrained role". On the contrary, the satellite agents can help them to be satisfied, they have solution(s), and thus the "service role". From these roles interactions between agents and entities are defined:
Request agents interact with their mesh agents,
Mesh agents interact with their request agent and with satellite agents that can acquire them,
Satellite agents interact with mesh agents requesting their help,
All of the agents use passive and active entities to improve their knowledge. The satellite agents interact with the trajectory module, cloud coverage, the ephemeris and both energetic and memory modules, while mesh agents interact with cloud coverage and the ephemerids.
In ATLAS system, and based on the agent behavior definitions given by the AMAS4Opt model, the planning of the constellation is provided by the local cooperative interactions among the agents of the system. This cooperation is ensured through the exchange of messages between agents and is guided by two indicators: **the criticality and the cost**.
The dissatisfaction of mesh agents is represented by their **criticality** degree. More the agent is closed to the state where its objective is reached, more its criticality is low. In the current version of ATLAS, two criteria are used to represent criticality: the priority given by the customer and the number of accesses existing to plan the mesh. Requests also influence the criticality of their mesh agents. When the validity range is going to expire or when only a mesh agent from its set of meshes is not planned, the request increases the criticality of the unplanned meshes.

The **cost** is an indicator of the difficulty for the satellite agent to take into account and plan a mesh agent. Indeed, the criticality does not allow full cooperation between agents: it only allows agents with the “service role” (i.e. satellites) to know which agents with the “constrained role” (i.e. meshes and requests) have the priority and by that to be cooperative. An agent with the “constrained role” cannot favor an agent with the “service role” among another and so cannot make a cooperative choice. The cost is here introduced to solve this problem.
To favor cooperation, a mesh agent chooses the satellite agent answering with the lowest cost. Different elements increase the cost:

- the need to adapt the plan by moving scheduled mesh, in this case the cost contains the criticality of the mesh agent to cancel,
- an important memory load (many meshes are already planned by the satellite),
- a large number of demands received.

A satellite agent answers with a low cost if, for example, it has no mesh scheduled or if it has received a few messages.
Cooperation is the engine of self-organization. It is ensured thanks to cost and criticality. Thus, ATLAS functioning can be described as follows:

In a first step, mesh agent asks for coverage to all satellite agents they have an access to it.

Satellite estimates the difficulty to plan each mesh and answers with the cost.

Mesh agent chooses the lowest cost and asks confirmation to the satellite that confirms or not.
Of course, as ATLAS handles dynamic, if a new mesh asks for a place that is already attributed to another mesh, the satellite also favors the most critical. At any moment, the request agent can increase the criticality of its mesh agent.

III.III Near Real Consistent Solution Proof

Mission plan is obtained through local interactions between agents. In the ATLAS system, those interactions are messages that agents exchange among themselves. Agents can be considered as distributed entities without global knowledge on the system: no agent knows the state of the whole planning. Thus, if a stop is asked by an operator (for example to establish a plan), we must be certain that each agent is in a consistent state. To be certain of this global consistency, some messages and actions must be realized before the real stop of the system.
A mesh agent m can be in two states: planned (PL(m)) or not (¬PL(m)). If a mesh agent is planned, that must be by one and only one satellite agent and both the mesh agent and the satellite agent knows each other. If it is not planned, no satellite agent has it in its plan. We note P_s the mission plan of the satellite **s,** Book(m, s_i) the fact that the mesh m knows that it is planned by the satellite si and M the set of all meshes to plan. Formally we can write:

(1) $PL(m^i) \Leftrightarrow \exists! s \in Book(m^i,s) \wedge m^i \in P_s$

(2) $\neg PL(m^i) \Leftrightarrow \nexists s \in Book(m^i,s) \wedge m^i \notin P_s$

Thus, we deduce that ATLAS is in a consistent state if and only if:

(3) $\forall m^i \in M, PL(m^i)" \oplus " \neg PL(m^i)$

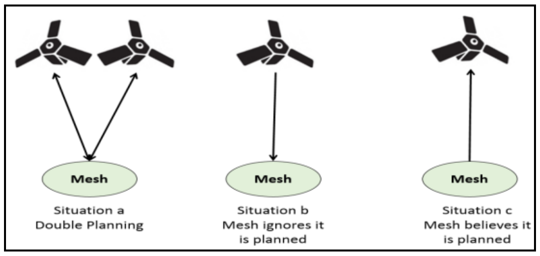

Figure 2: Inconsistent Situations

In the ATLAS system, three situations of inconsistency have been identified (Figure 2)
Figure 2.a) a mesh agent is planned by more than one satellite:

(4) $Book(m^i,s1) \wedge m^i \in P_{s1} \wedge Book(m^i,s2) \wedge m^i \in P_{s2}$

Figure 2.b) a mesh agent ignores that it is planned:

(6) $\neg Book(m^i,s) \wedge m^i \in P_s$

Figure 2.c) a mesh agent believes is it planned:

(5) $Book(m^i,s) \wedge m^i \notin P_s$

We introduce a new active entity called “**Manager**”. The Manager is the interface between ATLAS and the human operator. This entity knows all the mesh agents and their state, the mesh agents also know this entity. When the operator asks for a stop, the Manager broadcasts the message to all the Satellite agents, and waits for the answers, to indicate the real stop. The following sequence diagram (Figure 3) describes the behavior of the Manager: At the reception of the “ask for stop” (action 2), mesh agents must finish their cycle (action 3). Then, three behaviors are possible, depending of the last agent’s action.

1. If a message “ask for confirmation” has been sent before the stop order was received, the mesh agent must wait and treat the satellite answer, to avoid the situation **b** (action 6).
2. Else if a message for cancel has been received before the stop order was received, it must be treated to avoid situation **c** (action 7).
3. Else, the mesh agent sends a message *OK* to the manager (action 8).

When all the *OK* messages have been received by the Manager (action 9), ATLAS may not be in a consistent state yet. Indeed, messages from the satellite agents may have a latency. The transmission duration of message between two agents is *t*, *2t* are necessary to ensure that the message sent by the satellite agent, is received by the mesh agent and the answer is sent back to satellite agent. Thus, the Manager must wait for *2t* after the reception of all of the messages to indicate the real stop (action 11). If a message is received during the wait (action 10), the *2t* must be waited again. In the ATLAS system, agents are located on the same computer, so the t duration is very short, that is why the delay between the order to stop and the real stop is short.

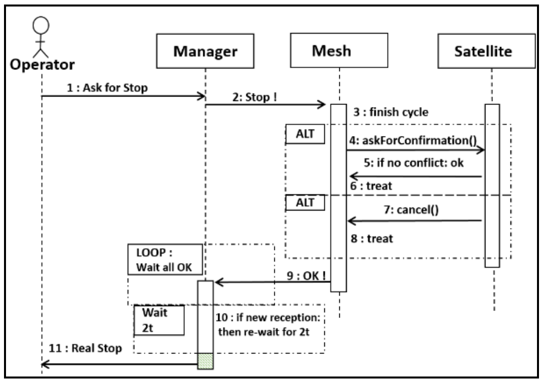

Figure 3: Stop Sequence Diagram

IV EXPERIMENTS

In this section, we present experiments we carried out to test and validate our system. Firstly, use case scenarios and the way we produce them are presented. Then, two experiments are detailed. These experiments validate our approach.

IV.I Experimental Setup

In order to test our system, we implement and use a **generic generator of scenarios**. This generator provides, for certain number of satellites, a list of meshes to acquire. For each mesh, several accesses are defined. ATLAS must select an access and schedule the acquisition inside.

The generator starts by constructing a complete mission plan, where each satellite is fully occupied by the acquisition of several meshes. The meshes and their characteristics are randomly generated in order to fill all the available time of each satellite. Once this complete mission plan is generated, the generator adds to each mesh a validity range and a random number of accesses, each being randomly associated to a satellite chosen into the constellation. The random number of accesses is linked to an accessibility factor defined by the user in order to increase (or not) the accessibility. This factor allows to generate the maximum number of satellites that can acquire the mesh. Thus, each mesh possesses several accesses and can be acquired from different satellites of the constellation. Finally, a priority is assigned to each mesh corresponding to the priority given by the customer. The generator randomly associates a priority to each mesh respecting the following distribution:

- 50% of “routine” priority, the lowest priority;
- 35% of “normal” priority;
- 15% of “urgent” priority.

As previously presented, customer's requests do not arrive at the same time. To introduce this dynamics, all meshes are not available at the start of the system execution, but arrive during solving. Thus, we can show how the system **self-adapts at runtime**. This generator allows to produce a significant number of scenarios representative of the combinatorial of the problem. This generation process has been validated by experts of the space field. Those different scenarios enable to test the validity, the robustness and the scalability of ATLAS.

IV.II Adaptation at Runtime

First experimentation illustrates the ATLAS’s ability to dynamically handle arrival of new meshes to plan. For this experiment, we use a scenario with 5 satellites, 717 meshes to plan, the ratio accesses per mesh is 3.65. Each mesh has a priority, respecting the distribution given in Section IV.I. In order to underline self-adaptive mechanisms, the arrival of meshes into the ATLAS system is controlled. Indeed, all “routine” priorities are available at the start of the execution, the “normal” ones are inserted at cycle 10, when ATLAS is converging to a solution. Finally, when ATLAS has handled both “normal” and “routine” at cycle 17, “urgent” meshes are inserted.

Figure 4 exposes the results. The 3 curves illustrate the percentage of all kinds of priorities during the execution. At each insertion, the number of planned meshes of lower priority decreases. This is explained by the fact that thanks to their self-adaptation mechanisms, Satellite agents can re-organize their plan to accommodate meshes with a higher priority. This experiment shows the importance and the contribution of the self-adaptation mechanisms: more meshes can be planned at runtime: “urgent” priorities are dynamically taken into account and planned. Moreover, the number of cycles needed to reach a stability is small, so ATLAS quickly adapts itself.

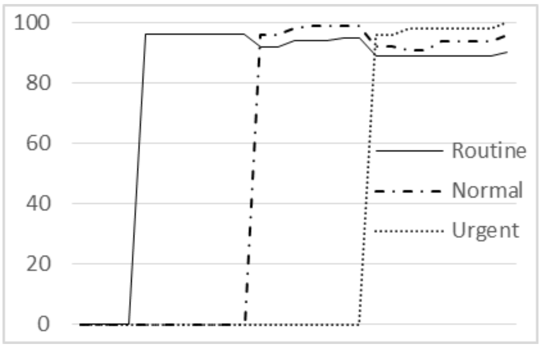

Figure 4 Dynamic Handling (% for each priority)

Another interesting point shown by this experiment is that even if a mesh is cancelled in order to free some place to a more urgent one, the cancelled one could find another space in the planning. Here again ATLAS self-adapts, and a mesh previously canceled can be planned.

IV.III Comparison to a Greedy Algorithm

To analyze the quality of solutions provided by ATLAS, we compare it to the commonly used algorithm in the spatial domain: a chronological greedy algorithm, named here ChronoG. ChronoG treats the constellation satellite by satellite. At each time step t of the planning of each satellite, ChronoG checks if the slot is free or if a mesh is already scheduled.

- If a mesh is scheduled, ChronoG goes to the next step, else ChronoG checks if a mesh can be set at this step, by determining whether the mesh possesses an access encompassing the current time.
- If several meshes can be set, ChronoG uses a heuristic to select the best one.

The heuristic used by ChronoG is close to the criteria used to define the criticality in ATLAS. To select the mesh to plan, the function first considers the client priority and then (only if several meshes have the same priority) the number of remaining accesses (the less a mesh has accesses, the more it is critical). Note that ChronoG is not able to manage information at runtime. All information about dynamic events are available at the beginning.

The objective of this second experiment is to demonstrate that an adaptive system is better than a classical approach to provide near real-time planning. For that, we use the same scenario as in Section 4.2. This scenario is executed on ChronoG too. Because ChronoG cannot handle dynamics, we stop it when it converges to a solution, add to the list of meshes the new ones to plan and relaunch it. The duration times are not significant to be considered: with the whole set of meshes, ATLAS converge to a solution in 34 milliseconds and ChronoG in 32 milliseconds. Thus, we compare percentage of planned set after a convergence for each algorithm. Table 1 shows the results we obtain. With the first set, both the algorithms planned 96% of the “routine” meshes. This is explained because there are only 50% of the whole meshes, thus it is easy for both the systems to obtain a good solution. But a gap can be observed with the insertion of “normal” meshes: ATLAS still planned 96% but ChronoG only 82%. Remember that ChronoG, as a classical algorithm cannot handle the dynamics and need to be relaunched whereas agents of ATLAS use local rules to cooperate so while running, new meshes can be added.

	ChronoG	**ATLAS**
%Routine	96	96
%Routine and Normal	82	96
% Routine, Normal and Urgent	75	95

Table 1 Percentage after insertions

Table 2 shows the percentage of planned meshes for each kind of category at the end of the execution. The 98% of “urgent” planned by ChronoG is explained by the fact that the heuristic of this algorithm favors high priority meshes and has not a cooperative behavior. Thus, high priority meshes are always booked over others. On the contrary, ATLAS is cooperative and used a continuous approach. Thus, mesh agents choose less costly answers from satellite agents that begin by booking more critical meshes. Cost and criticality allow the system to be cooperative and to plan a maximum of meshes.

	ATLAS	**ChronoG**
% Total Final	96	75
%Routine Final	94	55
% Normal Final	94	89
% Normal Final	98	95

Table 2 Percentage of Final Planned Requests

This second experiment illustrates that our system, ATLAS, can take into account requests without reconsidered its whole planning. Thus, near real-time planning can be performed. ChronoG, because it cannot handle the dynamic have to rebuild the whole plan in case of changes, so new *good* plans cannot be generated faster than ATLAS.

IV.III Integration of Guidance and Maneuvers API

As presented in Section IV.I, the first version of our generator provided to us generic data. Indeed, in those first scenarios, no real information about satellites where provided: in the complete plans, accesses duration include a fixed maneuver duration. Of course, in real planning this duration depends on the maneuver the satellite has to perform to acquire two meshes successively, and so some maneuvers cannot be realized. The acquisition duration also varies, according to the satellite position. In order to compute both these durations (acquisition and maneuver), we rely on an industrial guidance and maneuvers library allowing to compute realistic durations even for most recent satellites. This library provides data for start and end of maneuvers and acquisition too. Thus, those information permit us to provide acquisitions plans that can be analyzed with mission visualizer tools.

Moreover, we made the integration of these library as generic as possible, thus ATLAS can work with a lot of different satellite maneuvers computation systems.

V CONCLUSION

In this paper, we have presented the ATLAS system based on self-adaptive multi-agent approach to dynamically plan a constellation of Earth observation satellites and provide near real-time solutions. The autonomous cooperative behaviors of the designed agents increase the self-adaptation and self-organization of the system. Thus, a mission plan can be computed insuring more flexibility, robustness and real-time adaptation. ATLAS can dynamically take into account a large number of high priority requests.

From the conducted experiments, we underline the fact that the considered self-adaptive approach delivers better results than commonly used methods. Mission plans generated by ATLAS are more optimized than mission plans delivered by ChronoG. In addition to this, ATLAS is not affected by dynamic and can produce near real-time plan: autonomous behaviors of the agents allow the system to adapt itself in order to plan a maximum of meshes, while respecting the constraints.

Future works will focus on adding another level of adaptation to improve solutions. We plan on testing ATLAS on real scenarios given by *CNES* (French spatial agency) and *Airbus Defence & Space - GEO-Information* who are in charge of the *Spot* and *Pléiades* constellations. These scenarios will allow us to compare our results with actual mission plans.

ACKNOWLEDGEMENT

These results have been reached in the frame of the OCE research project held in the Saint-Exupéry Technology Institute (IRT). We would like to thank industrial and academic members of the IRT which support the presented activity through their bringing, both in terms of proper knowledge and financials as well: Airbus Defence and Space, IRIT, Thales Alenia Space, Telespazio.

This project also took benefit of the Future Investment Plan (PIA) set up by the French state, steered by the *Commissariat Général aux Investissements*, and managed by the Research National Agency (ANR). Authors associate theirs thanks to those institutions.

REFERENCES

[1] Lemaître, M and al. (2002). Selecting and scheduling observations of agile satellites.

In: Aerospace Science and Technology.

[2] Bianchessi, N. and al. (2007). A heuristic for the multi-satellite, multi-orbit and multiuser management of Earth observation satellites. In: European Journal of Operational Research, 177(2):750–762.

[3] Junzi S and Marcel Quintana C. (2010). Optimising Ground Stations Scheduling with a Genetic Algorithm. In I-SAIRAS 2010, Sapporo, Japan.

[4] Bensana, E. and Verfaillie, G. (1999). Earth Observation Satellite Management. In: Constraints, volume 299, pages 293–299.

[5] Dago P. (1997). Extension d'algorithmes dans le cadre des problèmes de satisfaction de contraintes valués. PhD Thesis, Université de Toulouse, France.

[6] Bouveret S., Lemaître M. (2014). Un algorithme de programmation par contraintes pour la recherche d'allocations leximin-optimales. In: JFPC, 2014.

[7] Grasset-Bourdel R., Flipo A., Verfaillie G. (2011). Planning and replanning for a constellation of agile Earth observation satellites.

[8] Mansour, M. A. and Dessouky, M. M. (2010). A genetic algorithm approach for solving the daily photograph selection problem of the spot5 satellite. In: Computers & Industrial Engineering.

[9] Wang P., Reinelt G., Gao P., Tan Y. (2011). A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation. In: Computers & Industrial Engineering, vol. 61, no 2, p. 322–335.

[10] Yuan Z., Chen Y., He R. (2014). Agile earth observing satellites mission planning using genetic algorithm based on high quality initial solutions. In: 2014 IEEE congress on Evolutionary computation.

[11] Clair, G and al. (2008). Self-Regulation in Self-Organising Multi-Agent Systems for Adaptive and

Intelligent Manufacturing Control. In: IEEE International Conference on Self-Adaptive and Self-Organizing Systems 2008, pages 107–116.

[12] de O Ramos, G., Rial, J. C. B., and Bazzan, A. L. (2013). Self-adapting coalition formation among

electric vehicles in smart grids. In: Self-Adaptive and Self-Organizing Systems 2013.

[13] Boes, J. and al. (2013). Self-Organizing Agents for an Adaptive Control of Heat Engines (short paper). In: ICINCO, 2013, pages 243–250.

[14] Gleizes, M.-P. (2012). Self-adaptive Complex Systems. In: European Workshop on Multi-Agent Systems, Maastricht, The Netherlands, p114–128.

[15] Kaddoum, E. (2011). Optimization under Constraints of Distributed Complex Problems using Cooperative Self-Organization. PhD Thesis, Université de Toulouse, France.