Multi-satellite mission planning using a self-adaptive multi-agent system

Jonathan Bonnet, Marie-Pierre Gleizes, Elsy Kaddoum, Serge Rainjonneau, Grégory Flandin

Multi-satellite mission planning using a self-adaptive multi-agent system

Jonathan Bonnet*†, Marie-Pierre Gleizes†, Elsy Kaddoum†, Serge Rainjonneau*, Gregory Flandin*
*Institut de Recherche Technologique Saint Exupéry, Toulouse, France
Email: {surname.name}@irt-saintexupery.com
†Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier, Toulouse, France
Email: {Surname.Name}@irit.fr

***Abstract*—Mission planning for a constellation of Earth observation satellites is a complex problem raising significant technological challenges for tomorrow's space systems. The large numbers of customers requests and their dynamic introduction in the planning system result in a huge combinatorial search space with a potentially highly dynamical evolution requirements. The techniques used nowadays have several limitations, in particular, it is impossible to dynamically adapt the plan during its construction even for small modifications. Satellites of a constellation are planned in a chronological way instead of a more collective planning which can provide additional load balancing.**

In this paper, we propose to solve this difficult and highly dynamic problem using adaptive multi-agent systems, taking advantage from their self-adaptation and self-organization mechanisms. In the proposed system, the agents, through their local interactions, allow to dynamically reach a good solution, while ensuring a controlled distribution of tasks within the constellation of satellites. Finally, a comparison with a classical chronological greedy algorithm, commonly used in the spatial domain, highlights the advantages of the presented system.

***Keywords*—adaptive multi-agent system; planning; multi-satellite;**

I. Introduction

A constellation of Earth observation satellites is a fleet, potentially heterogeneous, of satellites. Such a fleet allows to cover a large Earth surface, with a good revisit frequency, ensuring different kinds of pictures and the robustness of the system. Planning an observation mission for a constellation is a complex task. Indeed, a lot of parameters and constraints, often contradictory, must be taken into account as for example, the number of satellites and their characteristics, the volume of customers requests and their associated constraints (like the kind of acquisition, the priority of the customer, the temporal validity range), and external constraints (such as the cloud coverage for optical satellites or matching the ground stations duty cycle). In addition to this, today's satellite systems are subject to an important evolution (in size and dynamism). For example the new Google *Skybox* constellation project will amount to more than 24 satellites, encouraging near real time client requests to grow drastically.

Currently, a satellite system works on a chronological basis relying on regular "appointments" between satellites and control ground segment when the work plan is uploaded onboard. The clients' requests arriving in the system are stored in the input pool and before each "appointment", they are taken as input of a planning algorithm. Requests arriving after the start of this planning process are stored for the next programming period. Indeed, currently used planning algorithms, such as greedy algorithm [16], fails to manage properly linked acquisitions (as for example stereoscopic acquisition where a request must be acquired several times) and to take into account dynamic changes in a relatively short time. To improve this, new approaches using self-adaptation and self-organization mechanisms are required. Indeed, those mechanisms bring more flexibility, robustness and real time adaptation. Thus, mission plans can be dynamically computed.

Self-adaptive multi-agent systems as defined in [11] with their ability to take into account a large number of entities and constraints, naturally provide self-adaptation and self-organization capabilities required to solve this kind of problem. To build our system, we rely on AMAS4Opt (*Adaptive Multi-Agent System For Optimization*) [14], a generic agent model which provides design patterns to solve optimization problems using cooperative self-adaptive multi-agent systems as defined in the Adaptive Multi-Agent System (*AMAS*) Theory [11]. In this model, agents are designed as close as possible to the natural description if the entities of the problem.

This paper is organized as follows. Section II describes the problem and presents its context. Section III develops the functioning of the ATLAS system (*Adaptive saTellites pLanning for dynAmic earth obServation*). Finally, Section IV presents an evaluation of the ATLAS system 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.

A. Description of the Problem

Based on the work of [2] and [6], the considered problem can be described as follows:

A set of satellites $Sat = \{s_1, s_2, ..., s_n\}$, in which each satellite s_i 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 client's requests $R = \{r_1, r_2, ..., r_m\}$, where r_i represents a client request and is defined by:

- its kind (*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,
- w, the tolerated cloud coverage, $0 <= w <= 1$,
- a set of meshes, $M^i = \{m_1^i, m_2^i, ..., m_p^i\}$, associated to the request r_i, called *sisters*.

A set of meshes to acquire $M = \{M^i, M^j, ...\}$ each M^i is associated to a request r_i and decomposed into a set of elementary meshes $M^i = \{m_1^i, m_2^i, ...\}$. Indeed, request can represent a huge area, and, because of the instrumental limit, satellites can not necessarily acquire the whole request in a single shot. So requests are divided into a set of meshes: a mesh m_k^i is the elementary entity that a satellite can acquire. Each mesh possesses the constraints of its request, but it is defined by a smaller geographic area. Thus, a request r_i is satisfied if all its meshes are acquired. To each mesh, one or several **access(es)** matching to a period where the mesh is visible by the satellite are defined. 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. Fig. 1 shows an access *A* on the mesh m_1^i, by a satellite. An acquisition slot *S* is inside the access *A*.

A set of hard and soft constraints must be taken into account. The set C_h 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), C_s can be released.

The objective of the problem is to provide a work plan where the maximum number of meshes possesses an acquisition date and is affected to a given satellite. The satellites must ensure the coverage while:

- satisfying all the hard constraints C_h,
- maximizing the number of satisfied soft constraint C_s,

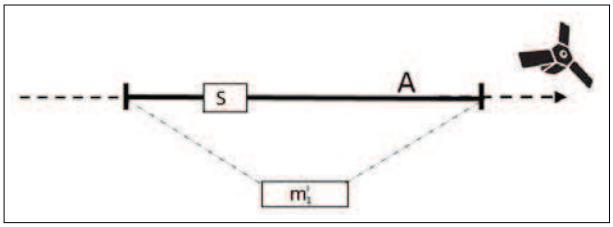

Figure 1. Visibility access and slot of realization

- maximizing the number of planned meshes.

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

B. State of the Art

Different papers compare the problem of mission planning of a constellation to optimization problems such as the **Multidimensional Knapsack Problem** [15], [13], or the **Traveling Salesman Problem** [17]. In the first one the knapsack represents the memory capacity of the satellite, while in the second, the path to find between the cities models the sequence between acquisitions. Many algorithms exist to solve these problems, and methods of spatial planning are inspired by them. We gather here the most commonly used approaches in the field, separating them between the exact methods and the approximate methods.

Exact Methods guarantee to find the optimal solution, if it exists. **Dynamic Programming** is used to decompose the problem into simpler sub-problems. The difficulty of such algorithms comes from the ability to recursively cut the initial problem. As shown in [16], when applying the algorithm linearly, the execution will quickly use most of the memory resources making the approach inapplicable in case of complex linked requests, as it is the case of stereoscopic imaging. It requires several acquisitions of the same mesh with an angular distance which creates links between sub-problems making the decomposition impossible. [18] emit the same criticism of algorithms built on **Branch and Bound**. Indeed, even if all the solutions are not explored as the algorithm uses the properties of the problem to avoid some branches, the use of such methods is extremely expensive in exploration time and required memory space. These methods are therefore not suitable for problems with very large search spaces, like the problem treated in this paper.

Approximate Methods can be applied to overcome these drawbacks. They help to find a good quality solution in a reasonable time. They differ in the heuristic used to guide the search and the quality of the obtained solution. The most commonly used algorithm in the spatial domain, due to a compromise between computation time and the quality of the solution, is **the Greedy Search**. Its principle is simple, it chronologically schedules requests, and in case of a conflict it applies a local heuristic. This algorithm is often used as a working basis, for example in [16]. Different other methods are adapted and applied to the problem of mission planning, like **Simulated Annealing** [19], **Tabu Search** [3], or **Genetic Algorithms** [12], [18].

All these algorithms, even if they produce quite good results, have limitations. Firstly, they are highly dependent on the heuristic that guides the search. Generally, such heuristics use a set of parameters (for example size of genes in Genetic Algorithm) that must be adjusted in order to reach good solutions. This requires a good knowledge of the problem and whenever something changes (size of the constellation, new constraints, etc.), the heuristic must often be re-adjusted. Also, when adding requests during the process of the planning, the

process must be resumed to the first opportunity of the added request. Finally, those algorithms are generally designed to chronologically plan the mission of a single satellite. Even if the problem remains similar when considering a constellation, they ignore new constraints such as the load balancing. In addition to this, they are not suitable to handle dynamics. Indeed, operational systems work step by step: during the building of the plan, no request can be added.

Recently, self-organizing systems such as adaptive multi-agent systems have proven valuable for solving dynamic problems, thanks to their ability to adapt themselves to their environment. For example, [9] propose an adaptive multi-agent approach to provide self-regulated manufacturing control. A dynamic resource allocation problem is also treated thanks to a trust and cooperation-based algorithm by [1]. Today, such systems are applied in a multitude of domains, we can for example name [4] who develop an adaptive multi-agent system for the dynamic control of heat engine or [10] who present an application to smart grids and management of electric vehicles. Being close to a natural description of the problem and designing the agents as close as possible to the entities of the problem, allow this approach to define more intuitive and rich solving algorithms which are then more robust to dynamics, flexible, open and provide a relevant level of adaptation in real-time.

To the best of our knowledge, multi-agent systems have not yet been applied to plan missions for satellite constellations **on-ground**. However, multi-agent systems have been used to plan on-board mission of a constellation of Earth observation satellites like [7]. In their approach, the satellites are defined as communicating agents through Inter-Satellite Links (ISL). The purpose is not the planning algorithm itself, but the distribution of tasks between satellites. For that, they negotiate among themselves the distribution of demands received from the ground to maximize the number of satisfied requests. Because of the Inter-Satellite Links, the applicability of this method requires a number of conditions such as the size of the constellation or the distance between satellites.

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

III. ATLAS

In the *Adaptive Multi-Agent System* Theory [11], agents are defined as autonomous, adaptive and cooperative entities that possess local objective. The local cooperation between agents allows the system to self-adapt in order to realize the function for which it is designed. Cooperation is defined as the ability agents have to work together in order to realize a common global goal. It implies that the activities of the agents are supplementary, and dependency links and solidarity exist between them. To deal with dynamic environments, agents possess mechanisms enabling them to autonomously modify their organization.

In order to identify the agents of the ATLAS system, we followed the ADELFE toolkit (French acronym for *"Toolkit to develop software with emergent functionality" - Atelier de Développement de Logiciels à Fonctionnalité Emergente*). ADELFE allows to develop software with emergent functionality [5]. The goal of this methodology is to guide the development of adaptive multi-agent systems, through several work definitions, from preliminary requirements to design and fast prototyping. Thus, ADELFE allows to qualify the environment of the system, the exchanged messages and to identify the system entities.

To ease the design of agents behavior and interactions for solving optimization problems based on the AMAS Theory, the AMAS4Opt agent model has been proposed [14]. 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" manages 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.

In this section, the identified agents, their behavior, interactions and the criticality measure of agent under "constrained role" are detailed. Then, the extension of the AMAS4Opt model, the cost measure, representing the criticality of agents under the "service role" is presented.

A. Agents

Given the problem description and using the ADELFE toolkit, 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* because they do not interact with others entities, they are resources.

The AMAS4Opt model allows to design the behavior and interactions between agents. In our case and given the AMAS4Opt agent roles definition, the satellite agents play the "service role", and the mesh agents and request agents play the "constrained role".

Below, the three types of cooperative agents indicating their local objective, the entities of the system with which they interact during the solving process and agents with whom they have to negotiate (exchange of information, service requests, etc.) to reach their objective, are presented.

Satellite agent type (service role)

- goal: to acquire requested data,
- interacts with:

- attitude and orbit computation module,
- cloud coverage,
- battery,
- memory,
- solar ephemeris,

- negotiates with:
  - mesh agents.

Request agent type (constrained role)

- goal: to have all its meshes planned,
- negotiates with:
  - mesh agents.

Mesh agent type (constrained role)

- goal: to be planned,
- negotiates with:
  - satellite agents,
  - request agents.

In the current version of the ATLAS system, the decomposition of requests into meshes is not taken into account because of the lack of real data. Thus, interactions between mesh agents and request agents is not currently implemented. Note that the interactions will only affect the criticality of the mesh agent. Indeed, whenever once of its mesh is planned, the criticality of the other meshes (*sister's*) must increase.

B. Self-Adaptive Multi-Satellite Planning

In the ATLAS system and based on the agent behavior definitions given by AMAS4Opt, the planning of the constellation is provided by the cooperation between its agents. This cooperation is ensured through the exchange of messages between agents and is guided by two indicators: **the criticality and the cost**. It is this cooperation that allows to produce a mission plan maximizing the number of scheduled requests and balancing the load within the constellation (this balanced load can be considered as a constraint). The life cycle of each agent is decomposed into three steps: "Perceive - Decide - Act". During phases of perception and action, agents receive and send messages (these exchanged messages are shown in table I). The phase of decision is the key step. Indeed, according to its perception and its state, the agent chooses which action it has to perform.

The cycle "Perceive - Decide - Act" is repeated until the system provides a solution. As the agents only have partial perceptions, they do not know if a global solution is reached. In order to detect that a good and coherent solution is obtained, and stop the agents, we introduce an **Observer**: as its name suggests, its function is only to observe the evolution of the system. Whenever this one is stable: agents do not change their state, the Observer asks all the agents to stop and exposes the solution. It is very important to note that the Observer has no interaction with agents during the solving, it only observes and stops them: it does not belong to the solving process.

The sequence diagram presented in Fig. 2 describes an example of interaction between a mesh agent and two satellites agents. In this example, a mesh agent can be planned by two satellite agent. The diagram can be described as follows:

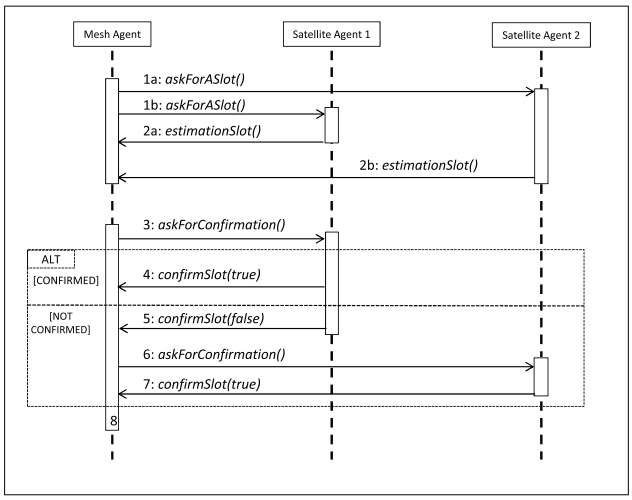

Figure 2. Sequence diagram

- First, the mesh agent sends an *askForASlot()* message to the both satellite agents (actions 1a and 1b).
- At the reception of this ask, both satellite agents compute the difficulty to schedule the mesh and answer with *estimationSlot()* message (actions 2a and 2b).
- When all satellite agents answered, the mesh agent chooses the satellite with the lowest cost (in order to be as cooperative as possible) and sends it a *askForConfirmation()* (action 3). The mesh agent keeps in memory all other answers. In this example, the chosen satellite agent is the number 1.
- The satellite agents can confirm or not (*confirmSlot()*). If confirmed (action 4), the mesh agent is planned.
- If the chosen satellite agents does not confirm (action 5), the mesh agent sends a *askForConfirmation()* (action 6) to the second satellite agent.
- The satellite agent confirms (action 7).
- Finally, the mesh agent is scheduled (point 8).

As we mentioned earlier, an important constraint of this problem is that the arrival of requests is dynamic. Customers can make an order at any time. Current systems, because of their chronological approach, cannot completely handle this constraint. It is not a problem for ATLAS, requests can be injected during the execution of the system, regardless of their accessibility date and corresponding agents are created at runtime. The autonomous cooperative behavior of the agents and the self-adaptation and self-organization mechanisms make the system self-organized, and able to dynamically plan requests. Thus, we can qualify ATLAS as a **dynamic continued resources allocation system**.

The functioning of ATLAS is based on the exchange of messages (Table I) between agents and their processing. In order to illustrate how agents interact between them, we present two case studies. The fist one (Fig. 3) is a simple scenario with two mesh agents (AM1 and AM2) to plan with two satellite

TABLE I. AGENTS AND MESSAGES

Agent	Message	Description
Satellite	estimationSlot(Cost c)	Inform the mesh of the cost to reserve a slot
	confirmSlot(Boolean b)	Confirm (or not) the planning of a mesh
	cancel(Access a)	Cancel the planning of a mesh
Mesh	askForASlot(Access a)	Ask a satellite for planning
	askForConfirmation(Access a)	Ask a satellite for confirmation
	informRequest()	Transmit new information to its request
Request	informMesh()	Transmit new information to a mesh

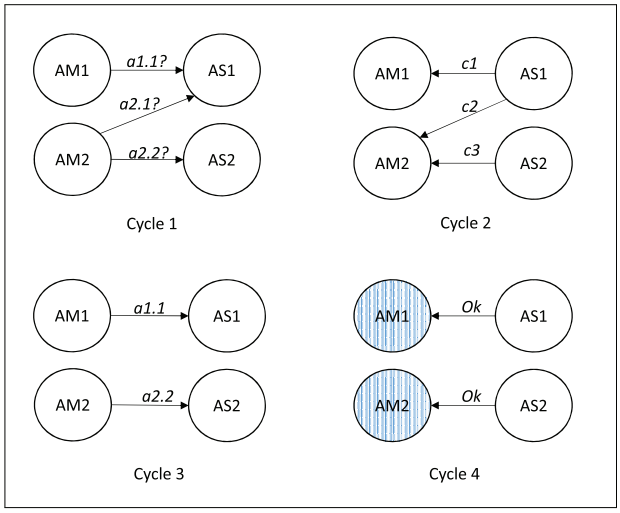

Figure 3. Simple case study

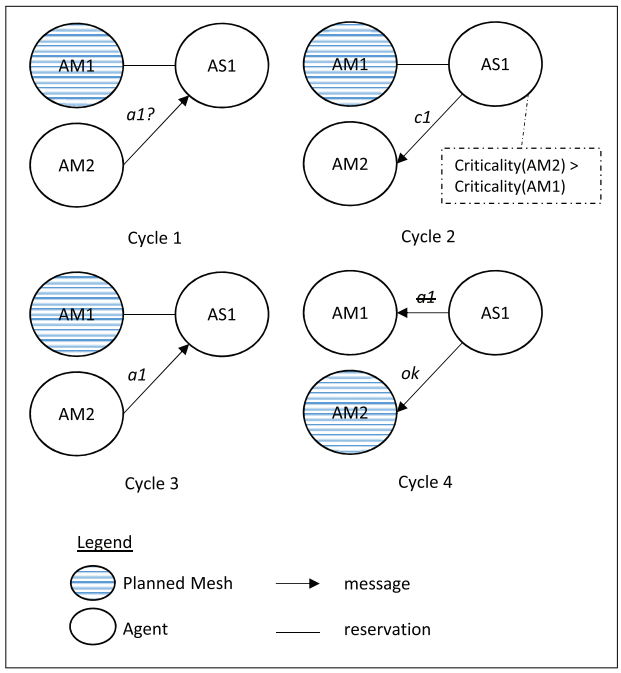

Figure 4. Case study with self-adaptation

agents (AS1 and AS2). The second (Fig. 4) presents a dynamic scenario to highlight how the ATLAS adaptive mechanisms work. A new mesh agent (AM2) is introduced into ATLAS and it only can be acquired by a satellite agent (AS1) at an already booked access (a1). The situations exposed in the two figures are not linked.

Fig. 3 exemplifies how two mesh agents (AM1 and AM2) negotiate with two satellite agents (AS1 and AS2), in order to be planned. AM1 can be planned by AS1 (access a1.1) while AM2 by AS1 and AS2 (the two accesses a2.1 and a2.2). The solving follows these cycles:

- cycle 1: for all their access, AM1 and AM2 ask for coverage to satellite agents: three demands *askForASlot()* are sent,
- cycle 2: AS1 and AS2 compute costs of planning (c1, c2 and c3) and return these costs to the mesh agents thanks to *estimationSlot()* messages,
- cycle 3: AM1 and AM2 select the lowest costs and ask confirmation to the satellite agents (*askForConfirmation()*),
- cycle 4: AS1 and AS2 validate the slot of reservation: they inform the mesh agents (*confirmSlot()*) and book the slot in their plan.

The scenario (Fig. 4) illustrates a dynamic situation where a new mesh agent (AM2) is introduced in ATLAS. AM2 is very urgent and can only be acquired by the satellite agent (AS1). In addition to this, the access that AM2 asks for (a1) is already attributed to another mesh agent: AM1. The four following solving cycles show how the three agents interact between them and how AS1 self-adapts its mission plan to dynamically book the more critical mesh agent:

- cycle 1: AM1 is planned on the access a1 with AS1, AM2 sends *askForASlot()* to AS1 for the same access a1,
- cycle 2: AS1 starts by comparing the criticality of AM1 and AM2. As AM2 is more critical, AS1 computes the cost to plan this mesh agent at access a1 and answers it (*estimationSlot()*),
- cycle 3: AM2 sends a *askForConfirmation()* message

to AS1,

- cycle 4: AS2 plans AM2 (*confirmSlot()*) and cancels AM1. This agent (AM1) restarts its life cycle (step 5 in the flowchart description).

C. Criticality and Cost Measures

Criticality: in ATLAS, the dissatisfaction of mesh agents is represented by their criticality degree. [8] defines the criticality measure as "*the distance between the current state [of the agent] and the state in which its local objective is achieved*".

The rules of local behavior of satellite agents are defined to encourage the planning of the most *critical* mesh agents. This criticality is represented by a set of ordered values. In the current version of the system, this measure includes **two criteria**. The **first criterion** concerns the priority to plan the mesh P. This is defined by the customer when sending its request. In case of equality of this first criterion, the satellite agent uses the number of remaining access Ra (**second criterion**). Thus, a mesh agent, that is visible only once by the satellites, has the priority over other meshes visible several times. Indeed, if the satellite agent favors a mesh agent with multiple access instead of a mesh with a single access, this one cannot be planned. Finally, the mesh agents change their criticality according to their status and the responses to their messages sent to satellite agents. The criticality of a mesh agent is represented as a set of value $Cm = \langle P, Ra \rangle$.

The evolution of the criticality of a mesh agent increases if the number of its accesses is reduced and when the end of its validity range comes closer. It is equal to zero when the acquisition of the mesh agent is planned.

Cost: the criticality, as defined in the AMAS4Opt model does not allow full cooperation between agents. Indeed, this measure only allows agents with the "service role" to know which agents with the "constrained role" have the priority and by that to be cooperative. An agent with the "constrained role" cannot favor an agent with the "service role" against another and so cannot make a cooperative choice. To solve this problem, we add the cost measure.

The cost is an indicator of the difficulty for the satellite agent to take into account and plan a mesh agent. It is computed by the satellite agent and returned to the mesh agent that sends acquisition demand. The mesh agent will therefore receive a cost (*estimationSlot()*) from each solicited satellite agent. To favor cooperation, a mesh agent chooses the satellite agent answering with the lowest cost. Here are some elements that 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 Cm,
- an important memory load (many meshes are already planned by the satellite) Ml,
- a large number of demands received D,

The cost is represented by the triplet $C = \langle Cm, Ml, D \rangle$. A satellite agent answers with a low cost if, for example, it has no mesh scheduled or if it has received a few messages.

IV. Scenarios Generation

In this section, the process of scenario generation is detailed as well as solution quality indicators. These different scenarios are used in Section V and VI in order to test ATLAS and compare it with a greedy algorithm, commonly used to plan satellites mission.

A. Generator of Scenarios

Before building mission plans, each customer's request is converted into a list of corresponding meshes to acquire. This preprocessing does not concern ATLAS. Indeed, ATLAS is designed to build a mission plan. It takes as an entry the list of meshes to plan and their characteristics, and provides a mission plan.

In order to test our system, we implement and use a generator of scenarios. 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 planned mesh a validity range and a random number of accesses, each performed by a satellite randomly chosen from the constellation. The random number of access 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, the generator defines a list of several meshes, each possessing several accesses and can be acquired from different satellites of the constellation.

In addition, the generator assigns a priority to each mesh. This priority corresponds to the priority given by the customer and represents the first criteria of the mesh agent criticality measurement. The generator randomly associates a priority to each mesh respecting the following distribution:

- 50% of "routine" priority, the lowest priority;
- 30% of "normal" priority;
- 20% of "urgent" priority.

As previously presented, customer's requests do not arrive in the same time. To introduce this dynamic, 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.

B. Experimental Setup

Nine different scenarios have been produced by the generator, to illustrate increasingly large constellations (from 2 to 10 satellites). Table II presents their characteristics. In these scenarios, we progressively increase the factor of accessibility of meshes by satellites. The larger is the constellation, the more the meshes are visible by different satellites. Thus, when

TABLE II. DESCRIPTION OF THE SCENARIOS

S	Number of Satellites	Number of Meshes	Number of Access	Accessibility Factor (%)
1	2	173	481	50
2	3	263	773	55
3	4	341	1041	60
4	5	424	1545	65
5	6	514	2157	70
6	7	580	2661	75
7	8	677	3748	80
8	9	777	5128	90
9	10	861	5677	100

increasing the number of possible accesses, the number of possible solutions increases as well as the size of search space.

Finally, to assess the ability of ATLAS and ChronoG (Section VI) to handle a large volume of low priority meshes ("routine"), a special scenario with 5 satellites was created, this scenario is named *S_special*. To create this scenario, we start from the generated scenario number 4 in Table II (424 meshes) and duplicate all the meshes to obtain a set of 848 meshes. The mesh priority was then fixed according to the following distribution: 10% of "urgent", 15% of "normal" and 75% of "routine". This scenario is *over-constrained* as it is impossible to plan the whole set of the meshes (the number of meshes was doubled from the scenario number 4).

C. Quality Solution Indicators

In the following sections, three indicators are used to define the quality of solutions produced by ATLAS and ChronoG.

- The percentage of planned meshes P. This percentage is computed using the number of meshes to plan M (the entry of the system) and the number of planned meshes M' (the system output), $P = \frac{M' * 100}{M}$.
- The global criticality level Gc. This indicator allows to observe the system execution. The fewer there are meshes that are satisfied and the more the global criticality is high. Thus, a good system that planned a maximum of meshes should have a weak global criticality at the end of this execution.
- The load balancing L indicates the occupation percentage of each satellite, in order to see if the constellation is entirely used.
- The scalability S. This last quality solution indicator allows to test a system with a huge set of requests.

V. ATLAS RESULTS

In this section, the results obtained with ATLAS to plan acquisitions over a constellation of satellites are presented. Following experiments are made using scenarios given by the generator presented in the Section IV. As ATLAS is a dynamic distributed approach, all of the presented results are an average over a hundred executions of ATLAS, the variance being lower than 2%.

TABLE III. DYNAMIC CONSIDERATION WITH ATLAS 1 AND 2

	ATLAS 1	ATLAS 2
Stability	17	24
% Routine	68	91
% Normal	92	95
% Urgent	89	95
% Total	87	94

A. Percentage of Planned Meshes

First experimentation compares two versions of ATLAS. The first one (named ATLAS 1) is a version where the self-adaptation mechanisms of the agents are not implemented. Indeed, in ATLAS 1, when a mesh agent is planned by a satellite agent, it cannot be canceled. At the contrary, in ATLAS 2 all self-adaption mechanisms are implemented. Thus, in ATLAS 2, satellite agents are fully cooperative with mesh agents: they can cancel or move less critical mesh to release space in order to plan critical mesh agents. ATLAS 1 and 2 are tested using the scenario number 4 of Table II. However, in order to underline self-adaptive mechanisms, the arrival of meshes into both systems is controlled. Indeed, all "normal" and "routine" priorities are available at the start of the execution, but "urgent" ones are inserted at cycle 10. Table III exposes the results. ATLAS 2 plans more "urgent" meshes (95%) than ATLAS 1 (only 89%) and the percentage of planned meshes is more important for ATLAS 2 (94%) than for ATLAS 1 (87%). The stability (the cycle when the Observer decides to expose the solution) is reached at cycle 17 in the case of ATLAS 1 rather than at cycle 24 for ATLAS 2. That is explained by the fact that when "urgent" requests arrive, most of slots are occupied in both systems, but satellite agents of ATLAS 2, thanks to their self-adaptation mechanisms, can re-organize their plan to accommodate "urgent" requests, while ATLAS 1 cannot free space. This experiment shows the importance and the contribution of the self-adaptation mechanisms: more customers requests can be planned, and "urgent" priorities are dynamically taken into account and planned. Moreover, the number of cycles needed to reach a stability is small, so ATLAS 2 quickly adapts itself.

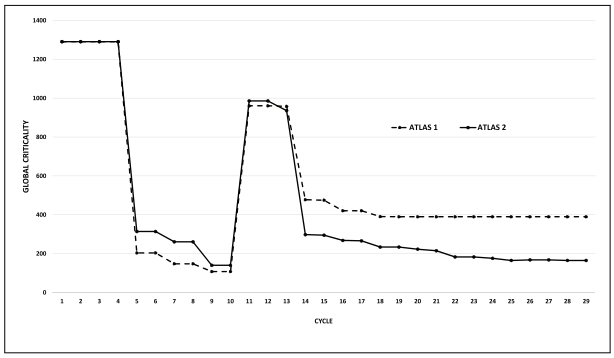

Figure 5. Evolution of the global criticality

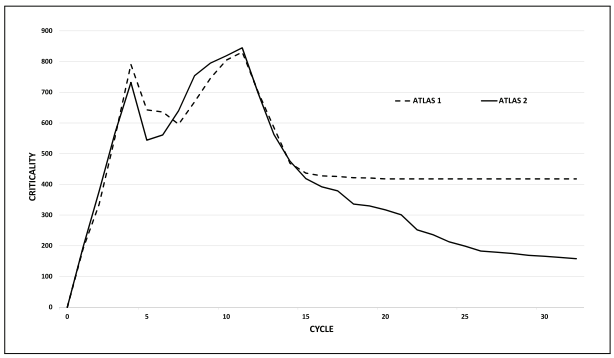

Figure 6. Evolution of the global criticality during the solving

B. Evolution of the Global Criticality

Fig. 5 shows the evolution of the global criticality for executions of ATLAS 1 and 2 in the case the arrival of "urgent" meshes is controlled. This global criticality is not used during the solving, it is just an indicator computed by summing the client priority of not planned mesh agents at each cycle of execution: $Gc_i = \sum_{j \in M} P_j$, where i is the cycle, M the set of unplanned mesh agents and P_j is the client priority of the mesh agent j. The increase at cycle 11 is caused by the arrival of the "urgent requests". The stabilization is reached by ATLAS 1 at cycle 17 while ATLAS 2 continues to decrease thanks to its self-adaption mechanisms, as it can plan more requests.

Fig. 6 compares the evolution of the global criticality in the case where the arrival of all meshes are not controlled. This situation is more realistic because clients requests can arrive at anytime. At first, we see that for both systems the criticality increases. This fast increase is explained by the fact that meshes just arrive into the system. From cycle 4, first meshes are planned so the criticality decreases for two cycles. Between cycle 7 and 15, the criticality is rising again because meshes continue to arrive along time. In the case of ATLAS 2, the criticality is higher because the system cancels meshes to free some space for more critical meshes. Finally, from cycle 15, the global criticality does not evolve for ATLAS 1 (the system can no longer plan meshes), while it continues to decrease for ATLAS 2. Indeed, starting cycle 15, ATLAS 1 can no longer plan meshes because satellites have no more free space. Here again, ATLAS 2 is more efficient. Thanks to their self-adaptation and self-organizing mechanisms, agents reorganize their plans and so schedule a maximum of meshes decreasing by that the global criticality of the system.

VI. Comparison to a Chronological Greedy Algorithm: ChronoG

A. ChronoG

To analyze the quality of solutions provided by ATLAS, we compare it to the commonly used algorithm in the spatial domain: the chronological greedy algorithm, named here **ChronoG**. ChronoG treats the constellation satellite by satellite. At each step of time 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 t.
- If several meshes can be set, ChronoG uses an heuristic to select the best one (see Fig. 7).
- In case of equality, ChronoG selects the mesh with the smallest difference between the end of its access and the current time t.

Finally, ChronoG plans the chosen mesh by booking its duration.

Fig. 7 explains the heuristic used by ChronoG: the three accesses A, B and C, respectively belonging to the meshes m1, m2 and m3, (m1 and m2 have "normal" priority and m3 has "urgent" one) can be scheduled at t. C will be chosen because it belongs to the most critical mesh: m3 has no more available access and its priority is "urgent". Note that ChronoG is not able to manage informations at runtime. All informations about dynamic events are available at the beginning.

B. Comparison to ChronoG

1) Percentage of Planned Meshes and Scalability: The nine scenarios presented in the Section IV are used to compare the performances of ATLAS and ChronoG on different case studies. Fig. 8 shows the global percentage of planned meshes. ATLAS is always better by more than 10% than ChronoG. ATLAS self-organization mechanisms allow it to go back on decisions, unlike ChronoG which schedules chronologically.

Table IV compares the percentage of each kind of priority planned by ATLAS and ChronoG. Gray background represented best percentage. We see that for the "urgent" meshes ChronoG is always better than ATLAS, it is because ChronoG plans urgent meshes first. But it books slots that could be suited for lower priority meshes. Indeed, we can see that ATLAS reached a stable behavior for the three kinds of priorities but ChronoG plans less than 60% of routine priority meshes decreasing by that the global percentage of planned meshes.

Using the scenario *S_special*, the ability of ATLAS and ChronoG to manage high priority meshes while planning a maximum number of meshes is analyzed. Table V presents the percentage of "urgent" and "normal" meshes planned and the global percentage of the planned meshes. ChronoG plans all the "urgent" meshes, and 90% of the "normal" priority ones,

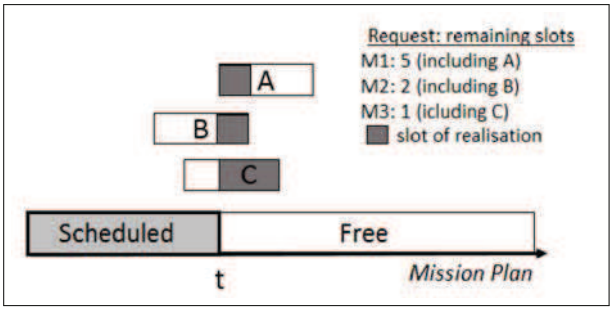

Figure 7. Heuristic of ChronoG

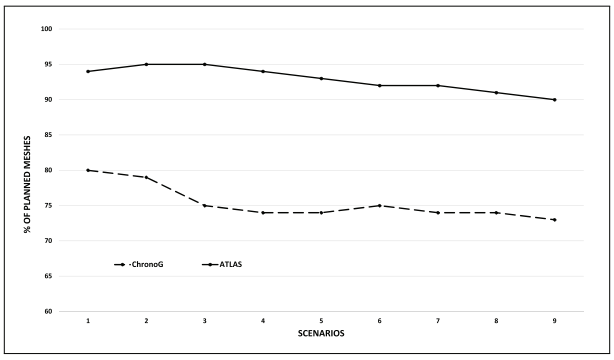

Figure 8. Comparison of ATLAS and ChronoG

TABLE IV. PERCENTAGE OF EACH KIND OF PLANNED MESH FOR ATLAS (A) AND CHRONOG (CG)

	% Urgent		% Normal		% Routine		% Total	
S	A	CG	A	CG	A	CG	A	CG
1	94	100	94	98	94	60	94	80
2	97	100	95	91	95	63	95	79
3	96	98	94	92	96	55	95	75
4	93	98	95	89	94	55	94	74
5	94	100	93	89	92	57	93	74
6	92	100	91	85	93	60	92	75
7	92	99	92	88	92	56	92	74
8	93	96	90	85	91	58	91	74
9	92	99	91	88	92	55	90	73

but only 45% of whole set of meshes are planned: only 29% of "routine" meshes are present in the solution. ATLAS plans less "urgent" meshes than ChronoG (80%) but ensures 81% of the whole set of meshes. The construction of the plan by ATLAS is not disturbed by the scalability.

These results can be explained by the fact that the heuristic of ChronoG 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.

TABLE V. PERCENTAGE OF PLANNED MESH ACCORDING TO PRIORITY

	ATLAS	ChronoG
% Urgent	80	100
% Normal	82	90
% Routine	80	29
% Total	81	45

2) Load Balancing: As we previously said, another important criterion when establishing a mission plan, is load balancing. To illustrate load balancing, a portion of the mission plan is presented (see Fig. 9 and 10). In this case, two identical *agile* satellites (*A* and *B*) are following each other, their *agility* allows them to acquire meshes parallel to their moving direction but also in front or behind them. In addition, each mesh can be acquired by the two satellites. Fig. 9 and 10 represent tasks that satellites have to execute during their flight. The central axis represents the ground track, *ie* the travel of the satellites seen from the ground. On the top of the central axis (black boxes) are the tasks of satellite B, the tasks of the A are on its bottom (grid boxes). Boxes are colored in function of the attributed satellite, they are empty if the mesh is not affected.

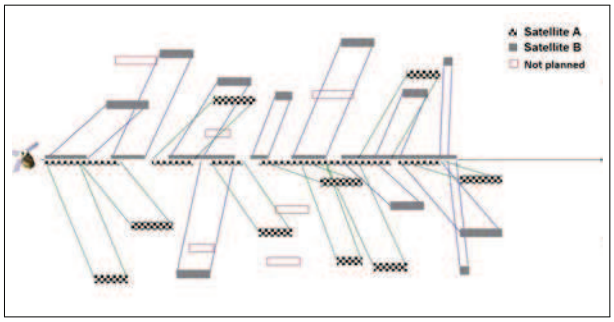

Fig. 9. Mission planned with ChronoG

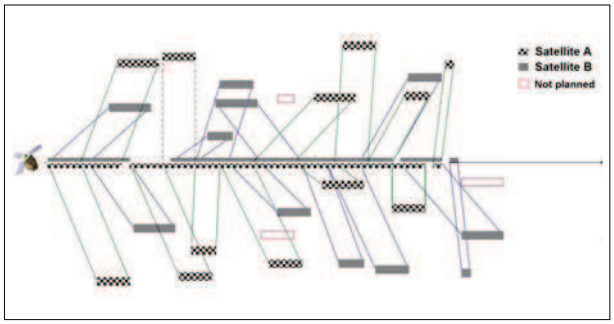

Fig. 10. Mission planned with ATLAS

In the plan produced by ChronoG (Fig. 9), satellite A has only nine tasks scheduled while satellite B eleven. Fig. 10 shows the plan produces by ATLAS where twelve meshes are acquired for A and eleven for B. Also note that the ChronoG plan counts more inactivity periods than the plan produced by ATLAS. This is explained by the fact that ChronoG is a chronological approach. Because of that and because there is no cooperation between meshes and satellite, first urgent tasks are processed and planned before others. Thus, when a mesh is planned, it cannot free its slot to favor another. On the contrary, thanks to its self-adaptation and cost mechanisms, ATLAS can efficiently balance the load on the different satellites of the constellation.

VII. DISCUSSION

Current systems are based on a global chronological heuristic. With the evolution of today's constellations and the increase of clients' requests, this chronological approach is facing strong limitations.

Firstly, adding or deleting a mesh (modifying the list of meshes to plan) during the construction of the plan is difficult and costly. This is especially the case for the greedy algorithm:

meshes arriving after the start of the processing are stored. If a mesh imposes a modification (because of its priority for example), the plan must be completely re-computed since the date of the modified mesh. That is why meshes that arrive during the generation of the mission plan are processed several hours after. On the contrary, ATLAS is an open dynamic system. Adding or deleting a mesh during the execution is possible. These modifications are local perturbations and agents self-adapt and change their organization to converge toward a new solution.

Secondly, classical systems consider first high priority requests and book available slots of the plan. Thus, the beginning of the mission plan is full and compact, while the end is rather empty. In addition to this, those systems are not adapted to constellations: load balancing is not taken into account, and some satellites have more acquisitions to perform than others. In ATLAS, agents use cost and criticality measures to cooperate and provide a balance load within the constellation.

Finally, space planning systems belong to the category of complex systems. Such systems are composed of many entities in interaction (agents, passive and active entities in our problem). Thus, another limitation is the way methods are used today to plan constellations. Presently, satellites of a constellation are most of the time planned one at a time, so the plan does not take advantage of the benefits of the constellation. On the contrary, the decentralized decision-making and distributed nature of ATLAS make it easier to consider all the satellites of the constellation at once.

We have presented the interest of an adaptive multi-agent approach to produce high quality mission plans for constellations of Earth observation satellites: dynamic planning, high percentage of customers request planned, a fairness distribution of task within the constellation and a the system scalability.

VIII. Conclusion and Perspectives

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. 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 and balance the load of the constellation.

From the different 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 balanced and optimized than mission plans delivered by ChronoG. In addition to this, ATLAS is less affected by the increase of the volume of meshes: autonomous behaviors of the agents allows 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 reach better solutions. The remaining behaviors of request agents and *sister* meshes will be implemented, and the criticality measure of mesh agents will take into account this influence. Finally, we plan on testing ATLAS on real scenarios given by *CNES* (*Centre National d'Etudes Spatiales*) 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.

Acknowledgments

Authors would like to thanks Institut de Recherche Technologique Saint Exupéry for funding this research.

References

[1] Anders, G., Steghofer, J.-P., Siefert, F., and Reif, W. (2013). A trust- and cooperation-based solution of a dynamic resource allocation problem. In *Self-Adaptive and Self-Organizing Systems 2013*, pages 1–10.

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

[3] Bianchessi, N., Cordeau, J. F., Desrosiers, J., Laporte, G., and Raymond, V. (2007). A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites. *European Journal of Operational Research*, 177(2):750–762.

[4] Boes, J., Migeon, F., and Gatto, F. (2013). Self-Organizing Agents for an Adaptive Control of Heat Engines (short paper). In *International Conference on Informatics in Control, Automation and Robotics (ICINCO), 2013*, pages 243–250.

[5] Bonjean, N., Mefteh, W., Gleizes, M.-P., Maurel, C., and Migeon, F. (2014). Adelfe 2.0. In *Handbook on Agent-Oriented Design Processes*, pages 19–63.

[6] Bonnet, G. (2008). *Coopération au sein d'une constellation de satellites.* PhD thesis.

[7] Bonnet, G. and Tessier, C. (2009). Multi-agent collaboration: A satellite constellation case. *Frontiers in AI and Applications*, 179.

[8] Bouziat, T., Combettes, S., Camps, V., and Glize, P. (2014). La criticité comme moteur de la coopération dans les systèmes multi-agents adaptatifs (short paper). In *Journées Francophones sur les Systèmes Multi-Agents, 2014*, pages 149–158.

[9] Clair, G., Kaddoum, E., Gleizes, M.-P., and Picard, G. (2008). Self-Regulation in Self-Organising Multi-Agent Systems for Adaptive and Intelligent Manufacturing Control (regular paper). In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO) 2008*, pages 107–116.

[10] 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*, pages 11–20. IEEE.

[11] Gleizes, M.-P. (2012). Self-adaptive Complex Systems (regular paper). In *European Workshop on Multi-Agent Systems, Maastricht, The Netherlands*, volume 7541, pages 114–128.

[12] Globus, A., Crawford, J., Lohn, J., and Pryor, A. (2003). Scheduling earth observing satellites with evolutionary algorithms. In *Conference on Space Mission Challenges for Information Technology*.

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

[14] Kaddoum, E. (2011). *Optimization under Constraints of Distributed Complex Problems using Cooperative Self-Organization*. PhD thesis.

[15] Lemaitre, M. and Verfaillie, G. (2006). Tutorial on Planning activities for Earth watching and observation satellites and constellation. *ICAPS*.

[16] Lemaître, M., Verfaillie, G., Jouhaud, F., Lachiver, J. M., and Bataille, N. (2002). Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, 6(5):367–381.

[17] Mancel, C. (2004). *Modélisation et résolution de problèmes d'optimisation combinatoire issus d'application spatiales*. PhD thesis.

[18] Mansour, M. A. and Dessouky, M. M. (2010). A genetic algorithm approach for solving the daily photograph selection problem of the spot5 satellite. *Computers & Industrial Engineering*, 58(3):509–520.

[19] Wu, G., Wang, H., Li, H., Pedrycz, W., Qiu, D., Ma, M., and Liu, J. (2014). An adaptive Simulated Annealing-based satellite observation scheduling method combined with a dynamic task clustering strategy. *Computing Research Repository*, abs/1401.6098:23.