Multi-Criteria and Multi-Objective Dynamic Planning by Self-Adaptive Multi-Agent System, Application to Earth Observation Satellite Constellations

Jonathan Bonnet

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *08/06/2017* **par :**

Jonathan BONNET

Multi-Criteria and Multi-Objective Dynamic Planning by Self-Adaptive Multi-Agent System, Application to Earth Observation Satellite Constellations

JURY

Marie-Pierre GLEIZES	Professeur, Université de Toulouse	Directrice
Elsy KADDOUM	Maître de conférences, Université de Toulouse	Co-Encadrante
Serge RAINJONNEAU	Docteur, IRT Saint Exupéry Thales Aliena Space	Co-Encadrant
Flavien BALBO	Professeur, ENS des Mines de Saint-Etienne	Rapporteur
Abderrafia KOUKAM	Professeur, Université de Belfort-Montbéliard	Rapporteur
Virginie GABREL	Maître de conférences, HDR, Université Paris Dauphine	Examinatrice
Grégory FLANDIN	Docteur, IRT Saint Exupéry Airbus Defense & Space	Invité

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse
Directeur(s) de Thèse :
Marie-Pierre GLEIZES, *Elsy KADDOUM* et *Serge RAINJONNEAU*
Rapporteurs :
Flavien BALBO et *Abderrafia KOUKAM*

THESIS

presented at

Université Paul Sabatier - Toulouse III

U.F.R. MATHÉMATIQUES, INFORMATIQUE ET GESTION

to obtain the title of

DOCTEUR DE L'UNIVERSITÉ DE TOULOUSE

delivered by

UNIVERSITÉ PAUL SABATIER - TOULOUSE III

Mention INFORMATIQUE

by

JONATHAN BONNET

Doctoral school: Mathématiques, Informatique, Télécommunications de Toulouse
Laboratory: Institut de Recherche en Informatique de Toulouse
Team: Systèmes Multi-Agents Coopératifs

Multi-Criteria and Multi-Objective Dynamic Planning by Self-Adaptive Multi-Agent System, Application to Earth Observation Satellite Constellations

---

JURY

Abderrafia KOUKAM	*Professor, Université de Belfort-Montbéliard*	(Reviewer)
Flavien BALBO	*Professor, ENS des Mines de Saint-Etienne*	(Reviewer)
Virginie GABREL	*Assistant Professor, HdR, Université Paris Dauphine*	(Examiner)
Grégory FLANDIN	*Doctor, IRT Saint Exupéry, Toulouse*	(Guest)
Marie-Pierre GLEIZES	*Professor, Université de Toulouse*	(Supervisor)
Elsy KADDOUM	*Assistant Professor, Université de Toulouse*	(Co-Supervisor)
Serge RAINJONNEAU	*Doctor, IRT Saint Exupéry, Toulouse*	(Co-Supervisor)

Jonathan Bonnet

PLANIFICATION DYNAMIQUE, MULTI-OBJECTIF ET MULTI-CRITÈRE, PAR SYSTÈME MULTI-AGENT AUTO-ADAPTATIF, APPLICATION AUX CONSTELLATIONS DE SATELLITES D'OBSERVATION DE LA TERRE.

Directrice de thèse : Marie-Pierre Gleizes, Professeur, Université de Toulouse
Co-Directeurs : Elsy Kaddoum, Maître de conférences, Université de Toulouse
Serge Rainjonneau, Docteur, IRT Saint Exupéry

Résumé

Établir le meilleur plan pour l'usinage d'un produit, le meilleur ordonnancement des activités de construction d'un bâtiment ou la meilleure tournée de véhicules pour la livraison des commandes, en prenant en compte diverses contraintes économiques, temporelles, humaines, ou même météorologiques : dans cette diversité d'applications, optimiser la planification est une tâche complexe par le grand nombre d'entités hétérogènes en interaction, la forte dynamique, les objectifs contradictoires à atteindre, *etc*.

La planification de missions pour des constellations de satellites en est un exemple majeur : beaucoup de paramètres et de contraintes, souvent antagonistes, doivent être pris en compte, entraînant une importante combinatoire. Actuellement, en Europe, les plans de missions sont élaborés au sol, juste avant que le satellite ne soit visible par la station d'émission. Les requêtes arrivant durant la planification ne peuvent être traitées, et sont mises en attente. De plus, la complexité de ce problème croit drastiquement : le nombre de constellations et les satellites les composant augmentent, ainsi que le nombre de requêtes journalières. Les approches actuelles montrent leurs limites. Pour pallier à ces inconvénients, de nouveaux systèmes basés sur la décentralisation et la distribution inhérentes à ce genre de problèmes, sont nécessaires.

La théorie des systèmes multi-agents adaptatifs (AMAS) et notamment le modèle AMAS4Opt (AMAS for Optimisation) ont montré leur adéquation pour la résolution de problèmes d'optimisation complexes sous contraintes. Le comportement local et coopératif des agents AMAS permet au système de s'auto-adapter à la forte dynamique et de fournir des solutions adéquates rapidement.

Dans cette thèse, nous adressons la résolution de la planification des missions de satellites par AMAS. Pour cela, nous avons complété et enrichi les modèles d'agents proposés par AMAS4Opt. Nous avons ainsi développé le système de planification dynamique de missions ATLAS. Pour valider ATLAS sur divers critères, nous avons utilisé un grand nombre de données hétérogènes. Enfin, ce travail a été comparé à un système "opérationnel" standard sur des scénarios réels, mettant en valeur les apports de notre système.

Jonathan Bonnet

MULTI-CRITERIA AND MULTI-OBJECTIVE DYNAMIC PLANNING BY SELF-ADAPTIVE MULTI-AGENT SYSTEM, APPLICATION TO EARTH OBSERVATION SATELLITE CONSTELLATIONS.

Supervisor: Marie-Pierre Gleizes, Professor, Université de Toulouse
Co-Supervisor: Elsy Kaddoum, Associate Professor, Université de Toulouse
Serge Rainjonneau, Doctor, IRT Saint Exupéry

Abstract

Building the best plan in product treatment, the best schedule to a building construction or the best route for a salesman in order to visit a maximum of cities in the time allowed while taking into account different constraints (economic, temporal, humans or meteorological): in all of those variety of applications, optimizing the planning is a complex task including a huge number of heterogeneous entities in interaction, the strong dynamics, multiple contradictory objectives, *etc.*

Mission planning for constellations of satellites is a major example: a lot of parameters and constraints, often antagonists must be integrated, leading to an important combinatorial search space. Currently, in Europe, plans are built on ground, just before the satellite is visible by the ground stations. Any request coming during the planning process must wait for the next period. Moreover, the complexity of this problem grows drastically: the number of constellations and satellites increases, as the number of daily requests. Current approaches have shown their limits. To overcome those drawbacks, new systems based on decentralization and distribution inherent to this problem, are needed.

The adaptive multi-agent systems (AMAS) theory and especially the AMAS4Opt (AMAS For Optimization) model have shown their adequacy in complex optimization problems solving. The local and cooperative behavior of agents allows the system to self-adapt to highly dynamic environments and to quickly deliver adequate solutions.

In this thesis, we focus on solving mission planning for satellite constellations using AMAS. Thus, we propose several enhancement for the agent models proposed by AMAS4Opt. Then, we design the ATLAS dynamic mission planning system. To validate ATLAS on several criteria, we rely on huge sets of heterogeneous data. Finally, this work is compared to an operational and standard system on real scenarios, highlighting the value of our system.

Je dédie cette thèse
à la mémoire de mon grand-père.

ICI s'achève une formidable aventure de trois années. Une aventure durant laquelle j'ai eu le plaisir de croiser un grand nombre de personnes avec lesquelles j'ai énormément appris. Je tiens par ces quelques lignes à remercier toutes ces personnes qui via leurs interactions m'ont aidé à produire cette thèse.

Tout d'abord, je tiens à remercier les rapporteurs de cette thèse, les professeurs Flavien Balbo et Abderrafiaa Koukam, d'avoir accepté d'évaluer ma thèse. Merci pour vos remarques et vos apports. Merci aussi à Virginie Gabrel d'avoir accepté de prendre part au jury de cette thèse.

Quatre personnes durant ces trois années ont été particulièrement importantes dans mon travail. Sans mes quatre encadrants, cette thèse n'aurait pu aboutir :

Marie-Pierre, tu m'as fait confiance il y a maintenant un peu plus de trois ans en me confiant ce sujet. Merci pour ton encadrement, ton attention et tes *plocs*. Elsy, pour qui j'étais le premier "doctorant". Merci à toi pour ta gentillesse, ton attention, tes conseils et ton soutien du premier au dernier jour. Tu m'as toujours accompagné, et j'ai toujours pu compter sur toi. Serge, merci d'avoir partagé avec moi ton expérience sur le monde spatial. Merci pour toute ton aide et tous ces conseils durant ces trois ans. Faire partie avec toi de la Secte Apple est un plaisir ! Enfin, celui qui ne veut pas être nommé, mais je prends le risque : Pierre. Ta vision, ta passion, ta pensée m'ont guidé durant ces trois années.

Merci du fond du coeur à vous quatre, merci pour tout ce que vous m'avez apporté et appris, travailler avec vous a été un grand plaisir.

Un grand merci à tous les membres du projet OCE de l'IRT Saint Exupéry. Merci pour votre accueil, et de vos apports : j'ai énormément appris grâce à vous. Mon bureau (élargi) : Alicia, Caro, Guillaume, Matt, Max et Max, je ne sais pas comment vous avez pu me supporter si longtemps ! Un grand merci à Matt pour tes supers graphiques et ton aide, je te passe le flambeau, prends-en soin et fais honneur à la phase 1; toi, le dernier rescapé.

Un merci à tous les membres de SMAC : travailler et partager tant de moments avec vous a été un plaisir. A tous les doctorants et docteurs de SMAC, vous êtes fantastiques, il y a dans ce couloir une cohésion qui permet à chacun d'être vite intégré. Ne perdez jamais cette flamme. Une pensée particulière pour Christine. Merci pour votre passion.

Je ne peux oublier Grégory Flandin, chef du projet OCE-SYNAPSE. Merci Greg de m'avoir accueilli, fait confiance. Merci pour tout ce que tu fais pour toutes les personnes du projet. Par la même occasion, je tiens à remercier l'IRT Saint Exupéry pour le financement de cette thèse, ainsi que toutes les personnes qui m'ont apporté leur aide durant ces trois années.

Papa, Maman, Solène et toute ma famille : merci pour tout. Merci pour votre soutien, merci pour votre amour. Je ne serais rien sans vous.

Et car ces derniers mots marquent pour moi un nouveau commencement, je souhaite commencer ce nouveau chapitre de ma vie en citant Alain, grand philosophe Radical (extrait des saisons de l'esprit - 1937):

> *Il faut croire en soi et espérer ; mais il faut vouloir croire en soi et vouloir espérer. A nous de mieux prendre le tournant, une autre journée commence.*
>
> *En tout temps obscur et difficile, on ne dit pas que l'année sera bonne ; on n'en sait rien ; ce qui arrive nous surprend toujours. Ce qu'on dit, c'est qu'il faut choisir de la penser bonne.*
>
> *Je vous souhaite de penser printemps.*
>
> Jonathan.

Contents

Acronyms

ACO *Ant Colony Optimization*

AMAS *Adaptive Multi-Agent System*

AMAS4Opt *Adaptive Multi-Agent System for Optimization*

AOCS *Attitude Orbital Control System*

ATLAS *Adaptive saTellite pLanning for dynAmic earth obServation*

DCOP *Distributed Constraint Optimization Problem*

DCSP *Distributed Constraint Satisfaction Problem*

DP *Dynamic Programming*

COP *Constraint Optimization Problem*

CP *Constraint Programming*

CSP *Constraint Satisfaction Problem*

EOS *Earth Observation Satellite*

GEO *Geostationary Orbit*

GPS *Global Positioning System*

GTO *Geostationary Transfer Orbit*

ISS *International Space Station*

LEO *Low Earth Orbit*

MAS *Multi-Agent System*

MEO *Medium Earth Orbit*

NCS *Non-Cooperative Situation*

PSO *Particle Swarm Optimization*

SAR *Synthetic Aperture Radar*

SSO *Sun Synchronized Orbit*

General Introduction

Planning is a part of daily life. Dressing, cooking, shopping, studying... Intuitively, we optimize our plans. When we go to the supermarket, we do not randomly cross the rays, but we perform a minimal distance between our needs. If we focus on our daily problems, we can see that they are not easy to solve. Organizing a day implies to take into account many objectives (bring the children to the school, go to the work, to the supermarket...), those objectives could be dynamic and unpredictable: emergency events can happen and must be treated fast turning upside down the established plan. Our life can be considered as a complex problem.

Complexity must not be mistaken with complicate. Where a complicated problem has many different and well-defined parts with well-known behaviors and can be reduced to simpler problems (a puzzle for example can be divided into smaller puzzles before being rather), a complex problem is defined by an important number of interacting little parts and it is their interactions that produce the global behavior. Part of a complex system are guided by simple and individual rules. The behavior of the system cannot be predicted from individual rules: "*the whole is greater than the sum of the parts*".

Complex systems ([Frensch and Funke, 1995] and [Funke, 2010]) are defined by:

- a large number of heterogeneous entities,
- multiple objectives, possibly contradictory,
- a high degree of connectivity between variables: a change in one variable may affect many others (feedback loops and non-linearity),
- a lot of feasible actions, with different effects and consequences that cannot be determined a priori (indeterminism),
- an environment subjected to dynamic evolving, those changes can be spontaneous making the situation less predictable (dynamics).

Those complex problems can be found in plurality of domains and science: from social to neural sciences or manufacturing and production control. Thus, engineers facing optimization problems all day long. Solving such problems is an important subject in computer science. Different solving methods have been developed. They are commonly classified in two categories: exact or inexact methods. First ones guarantee to find (if it

exists) the optimal solution. Still, as they explore the whole search space, they are too slow and cannot be applied to solve complex problems. Approximate methods find a solution using a heuristic to cross the space search. An important drawback of all of those methods is that they cannot manage dynamic, what is one of the characteristics of complex problems.

To overcome this limitation, new methods are currently being studied. Those methods are mainly decentralized. Thus, the exploration of the search space is parallelized and dynamics can be managed. Among those new resolution methods, the SMAC team (Systèmes Multi Agent Coopératifs for Cooperative Multi-Agent System) proposed the Adaptive Multi-Agent Systems (AMAS) [Gleizes, 2012]. In an AMAS, cooperative agents cooperate together to make emerge a solution. Thanks to self-organization mechanism, AMAS naturally provide the flexibility and the robustness necessary to solve complex problems. AMAS are applied with success in many fields, like complex problem solving [Capera et al., 2003]. A generic model of cooperative agents, AMAS4Opt [Kaddoum, 2008], has been proposed and validated on complex problems like manufacturing control. Even if first results are quite good, several lacks exist and must be implemented to apply AMAS4Opt model on new and various complex problems.

In this thesis, we focus on a particular complex problem, for which classical methods show limitations: mission planning for constellations of satellites. Making a mission plan consists in selecting among an important set of requests a subset of requests and temporally organizing them to make a mission plan. This mission plan must respect a set of constraints, often contradictory, like satellite memory and energy, kind of image to acquire, weather, *etc.* Thousand requests are treated to build the mission plan. The planning system must determine, for each selected request, the date on which the acquisition must begin. Indeed, each request can be acquired during a visible window of an average of several minutes, but the acquisition is performed in a few seconds: there are a lot of possible starts for each acquisition, increasing the search space. Thus, all of those constraints and parameters make this problem highly combinatorial: this problem is NP-Hard [Lemaître et al., 2000].

Currently, in Europe, satellite system works on a chronological basis relying on regular "appointments" between satellites and control ground stations. During this appointment, the already computed and validated work plan is uploaded on-board. Before each appointment, the set of clients' requests is taken as input of a planning algorithm. If a request is submitted after the start of the planning process, it is stored for the next programming period. Thus, currently used planning algorithms fail to take into account changes at runtime.

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 Terra Bella*[1] 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.

All of these constraints and new challenges make this problem very important for new space systems. From a scientific point-of-view, this problem is a perfect example of complex

[1] https://terrabella.google.com

problem to solve. In the literature, it is classified as an optimization problem. After studying the existing approaches to tackle it, we highlight that none is able to correctly manage it. In particular, all studied methods lack to manage the dynamic and the important amount of data.

Contributions

This thesis contributes by bringing two main contributions in complex problems solving using multi-agent system:

- Scientific contributions:
  - enhancement of the AMAS4Opt model and definition of new cooperative patterns.
- Applicative contributions:
  - rely on our models to design an adaptive multi-agent system to dynamically plan missions for constellations of Earth observation satellites.

AMAS4Opt has been tested with success on two problems. Facing it to a new one pointed out some limitations and lacks. Those lead to an enhancement of this model and new cooperative patterns for the AMAS theory. This model enhanced has been applied to the mission planning problem. Thus, we design **ATLAS** (Adaptive saTellite pLanning for dynamic eArth obServation), an adaptive multi-agent system to dynamically plan missions of constellations of satellites. Our system has been designed using cooperation rules refinement. Moreover, a mechanism guarantees that delivered plans are coherent with planning constraints, and so does not endanger the satellites. We evaluate our system on a set of experiments, and we confront ATLAS to real scenarios and compare it to the standard planning algorithm. From the analysis of the result, we highlight the advantages and limitations of AMAS4Opt and the proposed patterns.

Manuscript Organization

This manuscript is structured as follows:

- Chapter 1 introduces the **context** of this work and motivations: mission planning characteristics and components of planning systems. This chapter highlights the importance of the planning system and challenges that tomorrow's systems will have to face.
- In Chapter 2, a **state of the art in optimization and planning** approaches is made. We direct this state of the art on approaches studied to answer the mission planning problem. To characterize those methods, criteria are proposed. Then, we focus on exact and inexact methods. After pointing out their major drawbacks, we present methods that can manage dynamics.

- In Chapter 3, we present tools and theory used in this thesis. After an introduction to multi-agent systems, a focus is made on emergence and self-adaptation to define **adaptive multi-agent systems** (AMAS). Finally, we detailed tools to design AMAS to solve optimization problems: **ADELFE** and **AMAS4Opt**.
- In Chapter 4, the **scientific contributions** of this thesis are proposed. From an analysis of the lacks in AMAS4Opt, an enhancement of the model and new pattern to manage complex problems using AMAS are described. Finally, a proof of the coherence of solutions is proposed.
- Chapter 5 relies on AMAS4Opt and scientific contributions to propose the **ATLAS** system. ATLAS is an AMAS to solve mission planning for constellations of Earth observation satellites.
- Chapter 6 shows the different **experiments and tests** we have performed to validate ATLAS on generic and real data. A comparison to the reference algorithm, the Hierarchical Greedy Algorithm, is also made.
- Finally, conclusion and perspectives are made, both on scientific and applicative point-of-view.

PART I

CONTEXT AND STATE OF THE ART

1 Application: Earth Observation Mission

« Le silence éternel de ces espaces infinis m'effraie. »

Blaise Pascal

Contents

1 Introduction

I

In this thesis, we focus on a particular optimization and planning problem under constraints: **mission planning for constellations of Earth observation satellites**. This problem is a good application case of complex problems solving. Firstly, all the components of a complex problem are present. Secondly, from an industrial point of view, this problem is a major drawback for tomorrow's space systems. Indeed, today's approaches will not be able to solve it efficiently: there is a real need for a new way to plan missions for constellations of Earth observation satellites. Finally, in the literature, new ways to solve this problem are now emerging. Still they encounter the same limits (for example, they cannot dynamically take into account new task to plan), making them difficult to apply. All of these reasons make this problem very interesting to study.

A constellation of Earth observation satellites is a fleet of potentially heterogeneous satellites, orbiting around the Earth and using their sensors (optical or radar) to perform acquisitions. Such a fleet allows to cover a large Earth area with a high revisit frequency, while providing different types of picture, or not. Moreover, a constellation guarantees the system's robustness. Mission planning for such constellations is a complex problem raising significant technological challenges for tomorrow's space systems. The large number of customers' requests and their dynamic introduction into the planning system result in a huge combinatorial search space with a highly dynamic evolution environment. The techniques used nowadays have several limitations, in particular, it is impossible to dynamically adapt the plan during its construction even for small modifications, and so to perform a near real-time planning. Indeed, satellites of a constellation are planned one by one, in a hierarchical way instead of a more collective approach.

Today, planning systems follow three steps:

1. requests (and their constraints) are collected, and cut into small geographic areas named as "meshes". A mesh is an area that a satellite can perform as a single acquisition (in one pass),

2. from the set of meshes previously established, the plan is elaborated using a hierarchical greedy algorithm (at least in Europe),

3. finally, the plan is uploaded to the satellite, when the latter is visible from the ground control station. Of course, to ensure the satellite security, the mission plan is validated before the transmission.

Today's operational algorithms show limits to handle the dynamic (new urgent requests, update of the weather forecast, *etc.*) in real time. Thus, the elaboration of the plan is performed at the last minutes before the satellite is visible, in order to consider the most recent information, and so the number of requests stored for the next programming period is minimized.

Earth observation is subject to an important evolution. Nowadays, the number of constellations of satellites drastically increases, like the number of satellites that compose

them. For example, the new *Google Terra Bella*[1] (formerly known as *Skybox Imaging*) constellation project plans to amount to more than 24 satellites in the next years, encouraging client requests to grow drastically. With such a constellation, the requests could be treated much faster than today and each point on the Earth will be acquirable at least once a day. Thus, there is a need for a near real-time planning system. Such a system must dynamically take into account new requests and new constraints during runtime, which is not the case of currently used approaches ([Lemaître et al., 2002] or [Bianchessi et al., 2007]). It is urgent to make mission planning **reactive, dynamic and flexible**.

The objective of this chapter is to introduce the context of this thesis: Earth observation, and more particularly mission planning. In the first part, components of an Earth observation system are described, while the second part analyzes the planning problem addressed in this thesis.

2 Earth Observation Chain

This section presents roughly how an Earth observation chain works, from the client's order to the delivery process. Such a system is decomposed into six steps (Figure 1.1).

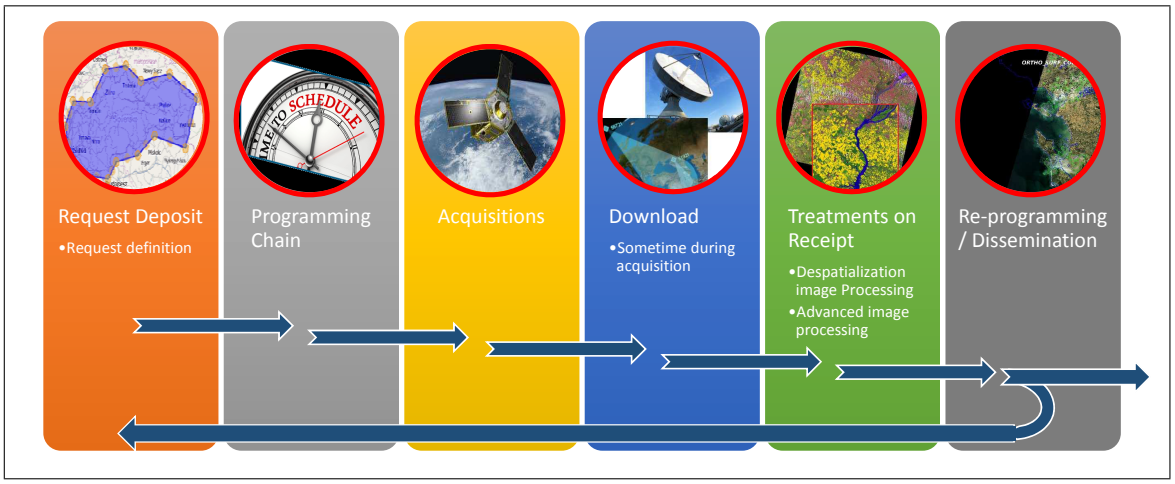


Figure 1.1 — Mission chain and operational concepts

1. **Request deposit**: in the first step of the mission chain, customers order requests: they define the geographic area to acquire, the image type (optical or radar), the acquisition mode: regular (a single acquisition), stereo (two acquisitions of a same point but from different positions in the orbit), *etc.*, and also constraints: the temporal validity range of the request and different geometric and meteorological constraints. Finally, the request is validated by an operator and a priority is adding, depending on the request and the price charged by the customer.

2. **Programming chain**: this key-step is composed of six phases. All of these phases must be accomplished before the visibility access of the satellite on the ground (*i.e.*

[1] https://terrabella.google.com

when the satellite and the ground station can communicate), in order to transmit the whole plan in good conditions.

- Phase 1: **Split**. Once all requests group, they are split into meshes. Indeed, the geographical area to acquire is often larger than the capacities of the satellite sensors. Thus, the request is divided into a set of juxtaposed rectangles, called **meshes** that can be acquired by a satellite in a single pass. There exist different methods to split a request maximizing the surface to acquire ([Liu and Hodgson, 2013] or [Ellis, 2019]). Figure 1.2 illustrates a request and its meshes.
- Phase 2: **Census**. Once all requests collected, it is necessary to determine the ones where a possibility of acquisition exists during the considered period of planning. Finally, for each mesh, a list of accesses and conditions is computed.
- Phase 3: **Ranking**. A priority is associated to each request and by transitivity to each mesh, depending on the customer. The list is then sorted according to these priorities. Some constraints, like the cloud coverage could affect this sort.
- Phase 4: **Planning**. The acquisition and download plans are computed. To produce the acquisition plan, the planning system uses several external modules to respect kinematics compatibility, system constraints and on-board storage capacities. The download plan is computed using those modules and the list of visibility accesses to ground stations.
- Phase 5: **Work plan verification and validation**. This final step verifies that the plan does not violate any constraints of the satellite. In case of failure, the plan is automatically or manually reworked until removal of all violated constraints.
- Phase 6: **Upload**. Once the satellite work plan and the download plan are verified and validated, they are uploaded to the satellite. The download plan is also transmitted to ground stations. Earth observation satellites are orbiting, so information is sent to them when they are in the part of orbital sphere in visibility from the ground station. It is for this reason that plans must be ready before the visibility of the satellite, if not, the satellite will not receive commands for at least one orbit.

3. **Acquisitions**: when the satellite has received the mission plan, it starts to execute it and in particular, it acquires data. For each mesh to acquire, the satellite first performs a pre-acquisition maneuver in order to be in the right *attitude* (*i.e.* the correct position). This maneuver depends, of course, on capacities, for example *Pléiade* can move around its three axis while an old generation of *Spot* satellites can only move their mirrors laterally. Then, when the satellite is in the correct attitude, instruments are activated in the proper mode. The module controlling the satellite movements (the *AOCS for Attitude and Orbit Control System*) is programed to perform the acquisition correctly. Finally, when the acquisition has been realized, data are stored in memory and could be compressed.

4. **Download**: the satellite possesses a download plan, computed on the ground. When it flies over a download ground station, it activates its telemetry payload, and download acquisitions and information about the mission respecting the download plan. A transfer

does not mean an immediate suppression of data. Indeed, an error could occur and deteriorate the transmission. For this reason, data transferred are marked as "downloaded", but the effective suppression will be ordered by the ground control later. Thus, an acquisition can be transferred several times and kept a long time in memory before its removal. Moreover, a picture could be downloaded several times.

5. **Treatments on receipt**: image treatment centers collect all acquisitions to be transformed into final products. Different transformations are applied to raw data (several corrections concerning space deteriorations, terrain geometry corrections and advanced image processing). Several versions of the image could be stored. Indeed, if another customer orders the same request, instead of re-acquiring the image, it is taken from the archives and specific treatments are applied. The on-board removal is ordered when the picture is verified and declared as correct. Then, the data will be archived.

6. **Reprogramming and Dissemination**: once the final image is obtained, a verification is operated to determine whether the image is correct regarding the customer specifications. If yes, the product is distributed to customers. If not, the request (or the mesh) will be re-injected in the planning system, in order to be satisfied during a future mission plan. In such a case, the satellite will be informed which images it can delete.

3 System Components

Previous section describes how the system works. In this section, we present basic concepts and define components of the Earth observation system. Such a satellite system is divided into two "segments": **ground** and **space** segments.

The ground segment is located on the ground, contains the components allowing to manage and exploit the space segment. The space segment is essentially composed of satellites and embedded parts such as the on-board interfaces.

3.1 Ground Segment

The ground segment represents all centers on the ground in charge of the space segment commanding, processing, distributing and archiving of satellite data.

3.1.1 Requests and Meshes

Client requests represent customer orders and are defined by:

- a type (optical or radar),
- a submission date,
- a temporal validity range (from hours to several days or weeks), the request must be acquired, downloaded, treated and disseminated during this range,

I

- a geographic area (the area to acquire, it can be huge like a continent or punctual like a simple target for example),
- a priority negotiated with the customer (the more a customer pays, the more the priority is high),
- some constraints on the image, like the tolerated cloud coverage (for optical acquisitions), or the acceptable incidence angle.

Each request is divided into a set of meshes to acquire. Indeed, request can represent a huge area, and, because of the instrumental limit, satellites cannot necessarily acquire the whole request in a single acquisition. A mesh is the elementary entity that a satellite can acquire in a single acquisition. Each mesh possesses the constraints of its request, but it is defined by a smaller geographic area. Thus, a request is treated if all its constituting meshes are acquired. The decomposition of a request into a set of meshes is outside the scope of the work presented in this thesis, we suppose that this decomposition is already done during the **split** phase. Figure 1.2 illustrates this decomposition.

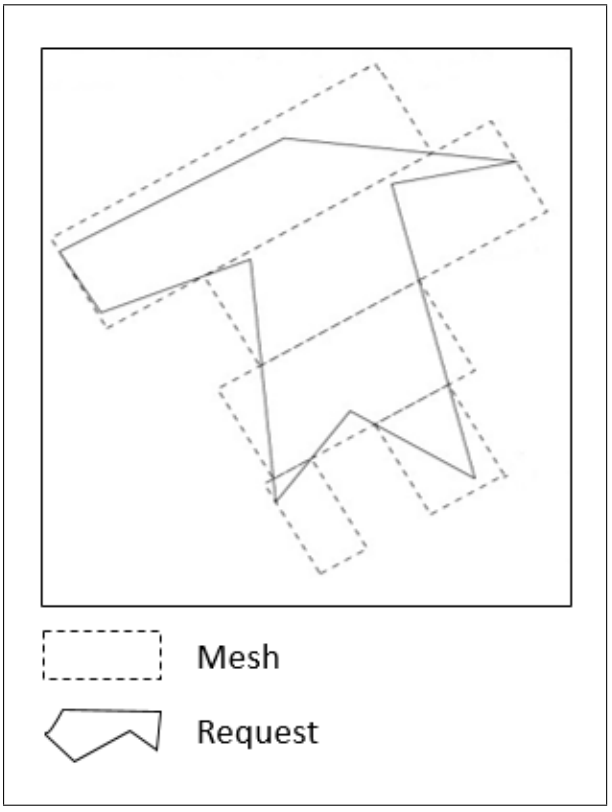


Figure 1.2 — Request decomposition into meshes

For each mesh, one or several visibility access(es) matching to a period where the mesh is visible by the satellite are previously computed, using satellite ephemerides. 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. Figure 1.3 shows an example of a single

visibility access *A* on the mesh m_1^i, by the satellite S_i. An acquisition slot *AS* is necessarily inside access *A*.

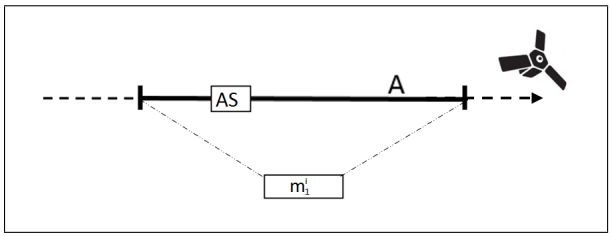


Figure 1.3 — Visibility access and acquisition slot

3.1.2 Ground Stations

The link between both space and ground segments is insured by different ground stations. In those stations, the communication between the ground and the satellite is operated. A satellite can only communicate when it is visible from the station. This visibility period concerns only 5 to 10% of the satellite orbit. This link is very important: missing a transmission implies that the new mission plan is not uploaded and the satellite could have nothing to do during a long period of time, or the data from the satellite cannot be downloaded to the ground, implying a loss of satellite time occupation. To avoid such a situation, uploaded mission plan is computed on a planning horizon greater than at least two successive passes over the station.

3.2 Space Segment

The **Space segment** is composed of the constellation of satellites and the embedded part of the interface between both segments: equipment dedicated to the mission (payload of the satellite) and all the tools needed to allow the satellite to realize its mission (platform).

3.2.1 Satellite

An artificial satellite is an object made by humans, launched into space by a rocket, and orbiting a celestial object (in our case: the Earth). This “Space Robot” ([Lemaitre and Verfaillie, 2006] and [Grasset-Bourdel, 2011]) only has, in theory, a trajectory to maintain. Deflecting a satellite from its trajectory is an expensive maneuver performed in emergency situations to avoid a massive space debris (from a tens of centimeters). Satellites have different functions like GPS, telecommunication, *etc*. In this work, we only consider Earth observation satellites. The mission of such satellites is to perform observations of Earth observation and to download them to the ground control center. Observations (or acquisitions) can be performed by an optical instrument (in the case of Spot satellite for example) or by a radar (for example COSMO-SkyMed[2] satellites). Figure 1.4 details an optical Earth observation satellite: Spot 5. Satellites are composed of two parts: the **payload(s)** and the **platform**. The first refers to all the equipment needed for the mission

[2]http://www.e-geos.it/cosmo-skymed.html

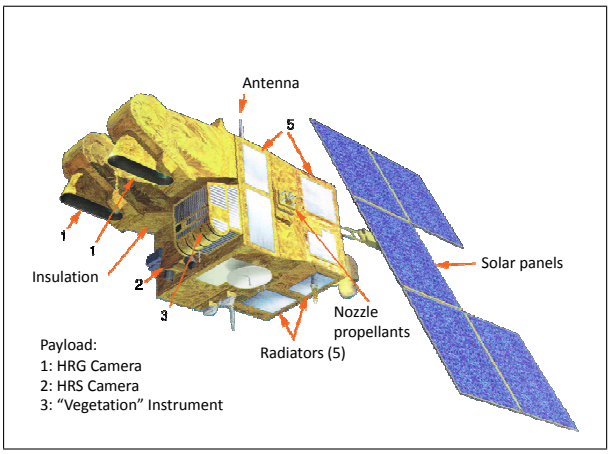


Figure 1.4 — Some of the major components of Spot 5 satellite

(for example: all photo sensors for an optical satellite) while the platform concerns what is helpful to the satellite in order to be able to perform its mission (solar panels, batteries, communication module, reaction wheels or control momentum gyroscopes, *etc.*).

New generation satellites are said **agile**. This characteristic means that they have the ability to maneuver and acquire targets that are not necessarily below them, but in front of/behind them or on their sides. Agile satellites allow to plan much more acquisitions than non-agile ones. But this agility adds a lot of combinatorial in the planning system. Indeed, with a classical observation satellite, like the old versions of Spot satellites, the start of the acquisition occurred when the satellite (on its orbit) is up above the start of the area to acquire perpendicularly to its motion vector. With a total agility, the satellite can move around its three axis: roll, pitch and yaw. Figure 1.5 shows the three axis. The acquisition can start when the satellite can see the target with a maximum front pitch angle (*i.e.* in front of it), up to a maximum near pitch angle (*i.e.* behind it). This implies that the number of possible acquisition starts is much more important. This is a combinatorial factor for planning algorithms that must find the right start, and must compute the correct maneuver to chain acquisitions. Maneuvers capabilities are constraints that cannot be violated, they represent minimal time between acquisitions. Figure 1.6 illustrates the behavior of two satellites, **a** is non-agile while **b** is agile, for the acquisition of two meshes : M_a and M_b.

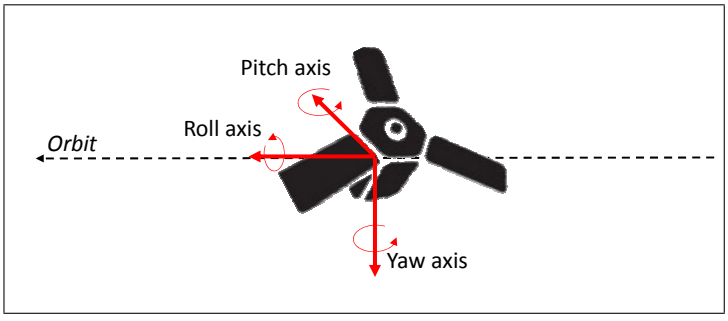


Figure 1.5 — Roll, pitch and yaw examples

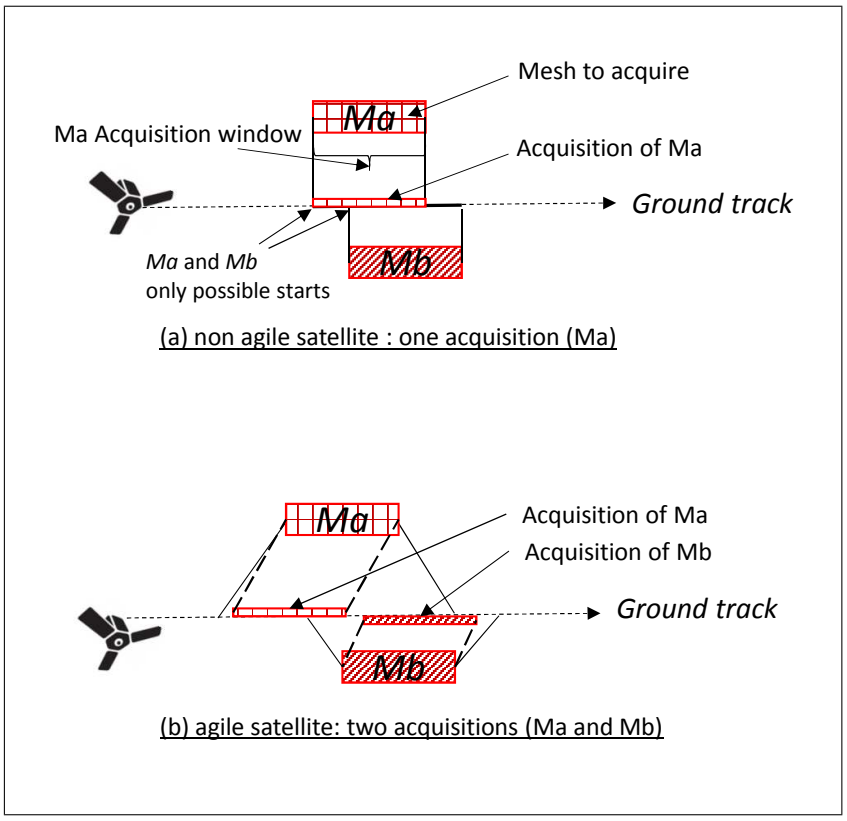


Figure 1.6 — Difference between agile and non-agile satellite

3.2.2 Constellation

A constellation of Earth observation satellites is a fleet, potentially heterogeneous, of satellites (Figure 1.7). Such a fleet allows to cover a large surface of the Earth compared to a single satellite, with a good revisit frequency, ensuring different types of pictures and a certain robustness of the system. Today's satellite constellations are subject to an important evolution (in size and dynamism). For example, the next French constellation will count four ultra-performing satellites to ensure daily revisit of any place on Earth, with a highly reactive system, encouraging near real-time client requests to grow drastically.

3.2.3 Orbit

After its launch by a rocket, and in absence of other perturbations, a satellite follows an inertial trajectory around the Earth, which is elliptical: the **orbit**. [Kepler, 1609] proposes a definition in its first law:

> *"An orbit is an ellipse where the Earth occupied one of the two foci."*

Four important points are used to describe the ellipse: **apogee** (point of the orbit where distance to the Earth is maximal), **perigee** (point of the orbit where distance to the Earth is minimal) and two **foci** (for each point of the orbit, the sum of the distances to the two foci is a constant). Using these four points, the ellipse can be described in its orbital plane (*i.e.* the plane where the satellite is moving) using two parameters (Figure 1.8):

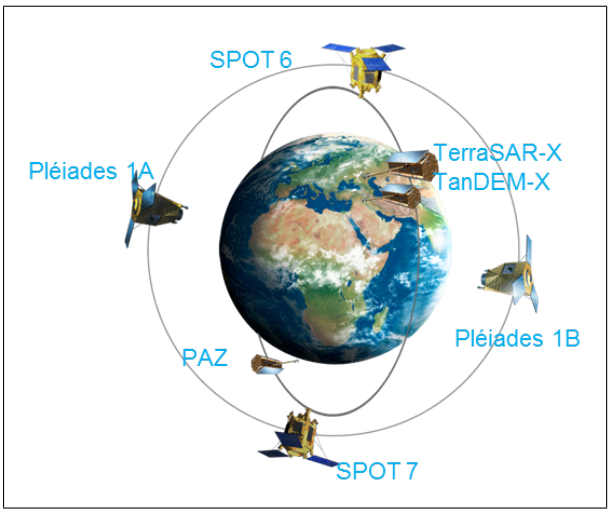


Figure 1.7 — Example of a heterogeneous constellation

- **The semi-major axis** a is one half of the longest diameter of the ellipse. $a = (Rp + Ra)/2$, where Rp is the distance between the Earth center and the perigee and Ra is the distance between the Earth center and the apogee.
- **The eccentricity** e defines the ellipse "flattening": $e = 1 - Rp/a$, where e represents the gap between the ellipse and a circle. Earth observation satellites have quasi-circular ellipse, so e is close to 0.

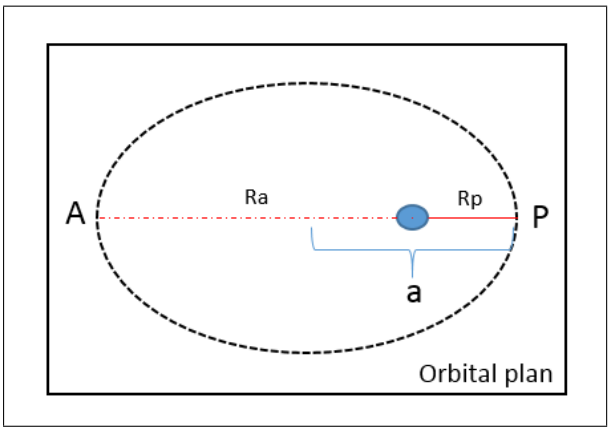


Figure 1.8 — Ellipse parameters

Two additional angular parameters indicate the inclination i which is the inclination the orientation of the orbital plane compared to the equatorial plane, and the orientation of the orbital plane which respect to the origin of the celestial longitudes. Those two angles are given considering the *line of nodes* (the intersection of the equatorial plane and the orbital plane):

- **Right ascension of ascending node** (or **longitude of ascending node)** Ω : the angle from the origin of longitude (on the Earth) to the direction of the ascending node (the point where the orbit of the satellite crosses the equatorial plane)

– **The inclination** i indicates the inclination of the ellipse regarding the equatorial plane.

Figure 1.9 represents an orbit around the Earth and the three orbital parameter (inclination i, the longitude of ascending node Ω and the argument of periapsis ω useful to determine a position on the orbit).

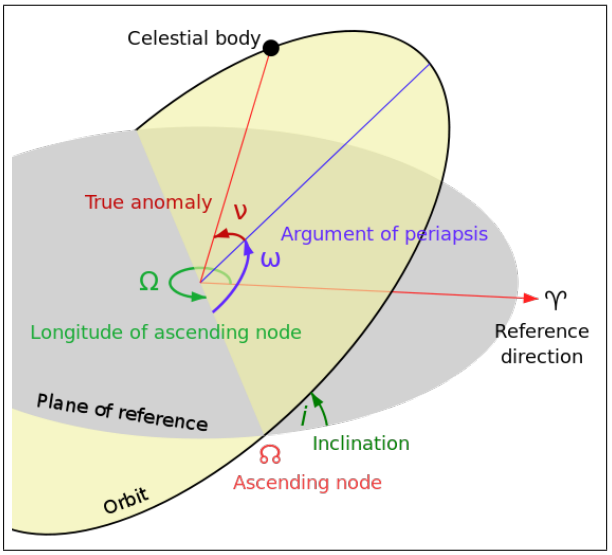


Figure 1.9 — Orbital parameters

The eccentricity of EOS orbits are almost always chosen as quasi-circular. Indeed, for a question of focal distance, it is more interesting to get an almost constant distance. Orbit altitude and inclination varies depending on the satellite mission. Thus, there are several classes of orbits. The Low Earth Orbits (**LEO**) are located between 400 and 1,200 kilometers. The Medium Earth Orbits (**MEO**), from 5,000 to 20,000 kilometers, are used for positioning satellites like Galileo and GPS. The more distant orbits (**Geostationary (GEO)** orbits) are located at 36,000 kilometers to the Earth. GEO are occupied by geosynchronous satellites, which move at the same angular speed as the Earth and are located in the equatorial plane. Another orbit, known as geostationary transfer orbit (**GTO**), is used to transfer a satellite to the GEO orbit. Some orbits are even higher than GEO: Highly Eccentric Orbits (**HEO**) having their apogee over 40,000 km. These orbits are only used for specific telecommunication missions at high latitudes. Figure 1.10 indicates the five different orbits possible for a satellite.

Earth observation satellites possess a **sun-synchronous** low orbit, named **SSO**. Such an orbit allows the satellite always to be over a longitude at the same local hour (typically just before or after midday). Thus, the spotlight of an area is always the same. In addition, such an orbit is *quasi-polar*: Earth observation satellites almost fly over the poles. Figure 1.11 represents the Earth saw from the North Pole and an orbital plane named "*12 a.m. orbit*". The interest of such an orbit is always to have the Sun in the same position relatively to the orbital plane. In reality, chosen orbits are not exactly "12 a.m." but lightly shifted in order to avoid specular effects in the sensor field of view.

Another important celestial body of our solar system to consider is the Sun. Indeed, the Sun is the energy source for both the satellite and imaging (for optical satellites). For

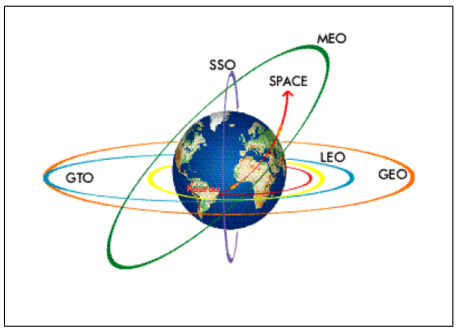


Figure 1.10 — Orbit classes

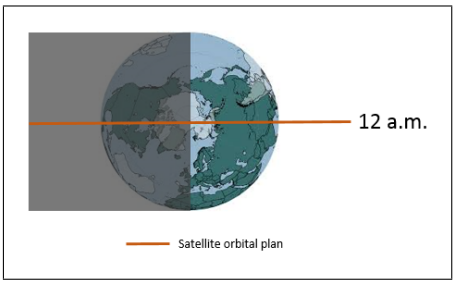


Figure 1.11 — "12a.m. orbit" from the North Pole

these reasons, in planning systems, the Sun and its ephemerides are important entities. Ephemerides are used to determine the position of the Sun regarding the satellite and the area to acquire.

4 Planning a Constellation of Earth Observation Satellites Problem

Based on the work of [Bensana and Verfaillie, 1999] and [Bonnet, 2008], the planning of a constellation 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 attitude and orbital control system, allowing the satellite to control its position and pointing (*towards the area to be observed*).

A set of clients' requests $R = \{r_1, r_2, ..., r_m\}$, where r_i represents a client's request and is defined by:

- its type (*optical or radar*),
- 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 with 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. A request r_i is satisfied if all its meshes are acquired. To each mesh, a set of accesses A are associated.

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,
- maximizing the number of planned meshes or the surface of planned meshes.

5 Problem Complexity

It can be shown that this problem is **NP-Hard** [Lemaître et al., 2000]. The class NP refers to *non-deterministic polynomial time*. Although any given solution to a NP problem can be verified in polynomial time, there is no efficient way to find a solution quickly [Garey and Johnson, 1979]. Problems categorized as NP are often solved through *heuristics*. The mission planning problem is similar to another NP problem, the Knapsack Problem: finding the longest path in a graph with additional constraints. In [Hall and Magazine, 1994] a problem named the "**Space Mission problem**" is introduced and categorized as NP-Hard. The goal of this problem is to select and plan a set of tasks on a machine, among a pool of requests. Each potential task is associated with a fixed duration to place inside a time window. Thus, the aim is to choose a sequence of tasks. This problem is close to our problem description, showing its high complexity. This complexity is a strong limit for current approaches. Indeed, they are based on greedy algorithms and they are not designed to handle dynamics.

6 Thesis Objectives

This thesis is a part of the OCE project (*Observation et Compréhension de l'environnement* French acronym for Observation and Environment Understanding). OCE project takes place in the Institute of Research and Technology Saint-Exupéry[3]. The goal of the project, through its different work packages, is to bring useful data to the final user.

[3] https://www.irt-saintexupery.com

I

A partner of the OCE project is the SMAC[4] (*French acronym for Cooperative Multi-Agent System*) research team in Computer Science Laboratory of Toulouse[5] (*IRIT*). The team studies an approach for the design of complex adaptive systems based on adaptive multi-agent systems and emergence, and has elaborated the AMAS theory (*Adaptive Multi-Agent Systems*) [Gleizes, 2012]. This theory gives local criteria to design *cooperative agents*. The cooperation between agents enables the emergence of a global function. Using AMAS theory, main team challenge concerns the design of complex systems for which the global behavior emerges from the cooperative and local interactions among the agents composing the system. Complex problems solving has been studied by [Kaddoum, 2008], who has proposed general patterns for two roles of cooperative agents to solve optimization problems using cooperative self-adaptive multi-agent systems: **AMAS4Opt** (*Adaptive Multi-Agent System For Optimization*).

The objective of this thesis is to propose a new way to plan a mission of constellations of Earth observation satellites. This system must overcome the difficulties that today systems meet: the increase of the number of requests and satellites, the heterogeneity of constellations, the need of a near real-time planning. To overcome those problems, we propose a new planning system: **ATLAS** (for *Adaptive saTellites pLanning for dynAmic earth obServation*). ATLAS relies on **AMAS4Opt**, and enhances it.

The ATLAS system has to satisfy three major criteria:

1. **scalability**: the system must be able to treat planning with an important number of entities (requests to plan and/or satellites in the constellation),
2. **dynamic**: it must be possible to modify the entities of the problem while the system is running (add or remove request, update constraints, *etc.*), this modification must be quickly taken into account,
3. **more reactivity**: the feedback on the request state must be fast (in order to inform customers about their orders).

In the Earth observation chain presented in Section 2, ATLAS is located in the fourth phase (planning) of the second step (programming chain). It only computes the mission plan. The download plan is not considered yet.

Next chapter presents the optimization and planning algorithm. We propose to focus on methods used or designed for plan missions of Earth observation satellites.

[4] https://www.irit.fr/smac
[5] https://www.irit.fr

2 Planning and Optimizing a Space Observation Mission

« Ce que nous nommons vérité n'est qu'une élimination d'erreurs. »
Georges Clémenceau

Contents

1 Introduction

I

Today, with an increasing number of changes in our society, a highly level of connection between humans or between machines (Internet of Things) and important dynamics, complex problems are omnipresent. Solving such problems is a major challenge for today's applications. Optimization methods could be applied to solve them. Nevertheless, a high level of knowledge on the problem structure must be known and dynamic is very hard to be taken into account.

Optimization consists to find a solution in a space search that satisfies one or several objectives while respecting constraints. A space search can be defined by a set (for example $\mathbb{R}$), but also by equations or models for complex problems. A classic example is the Traveling Salesman Problem (TSP), [Hoffman et al., 2013], [Dorigo and Gambardella, 1997], in which the shortest path on a map that visits all cities (only a single time) and returns to the origin city must be founded. **TSP** problem is a well-known problem and it has been studied many times [Sabry et al., 2014]. Meanwhile, as a NP-Complete problem, there is no algorithm to solve it in a polynomial time.

Two categories of methods can be applied to overcome those problems. Methods of the first category are named **Exact Methods**. Those methods search for the solution into the whole space search. Even they can find the optimal solution, they are very slow, and impractical for NP-Hard problems. The second set corresponds to **Approximate Methods**. Such methods do not explore the whole space search but use functions to cut and optimize the search process. Thus, the method performance highly depends on the chosen function called **heuristic function**. Such a function ranks, at each step, alternatives in the space search and selects the best one based on an evaluation function. Better the design of the heuristic is, better the obtained solution is. Usually, in optimization problems, heuristics combine two important steps: exploration and exploitation. Exploration step consists in considering new solutions while exploitation steps try to improve already found solutions. Good equilibrium between those two steps is essential to reach good solutions [Meignan, 2008].

The key to produce a good approximate method is so to design a good heuristic function: produce a solution in a reasonable time that is good enough. Because of the complexity, the algorithm may not produce the optimal function, but a local optimal.
Mathematically, optimization can be defined by [Intriligator, 1971]:

Given a function $f : D \longrightarrow R$, where D is the space search,
find $x_i \in D$ such that $f(x_i) <= f(x), \forall x \in D$.

In this definition, the objective to optimize is modeled as a global evaluation function to minimize. In complex problems solving, global objective function is usually defined as a sum of weighted objectives. Depending on the weight associated to each objective, the obtained solution improves a subset of the desires objectives. Indeed, it can be impossible to improve all objectives. Such a state is called Pareto optimality [Censor, 1977]. In such a state, it is impossible to improve the satisfaction of a subpart without making at least another worse. Still, in highly dynamic environment where constraints and objectives are subjected

to modifications and evolve during the search process, such global evaluation functions are difficult to define and to evolve in order to handle the dynamics.

Planning is the way to organize a succession of tasks without violating their constraints. A typical example is the construction of a schedule where a group of computer science students must be assigned to an available computer science professor and to an available computer room for a certain date. Optimize the plan is very important: a schedule is better if students do not have long periods without course or if all classrooms have the same occupation rate.

In this thesis, we focus on optimization of mission planning for Earth observation satellites. In this case, build a mission plan is similar to resource allocation problem. Indeed, the objective is to allocate satellite resources in order to answer customers' requests, respecting a set of constraints (problem description is presented in Chapter 1 Section 4).

Optimization methods can be applied to planning problems. In this chapter, a survey of several optimization methods applied to mission planning problems is presented. In the literature, an important number of optimization methods can be found. We focus on the subset of methods currently studied (or already used) and applied to our problem. This chapter is organized as follows: first, a presentation of formalism in optimization methods is presented. In the second section, a set of criteria are defined to discuss the presented methods. Then, classical optimization methods are analyzed given two categories: **Exact Methods** and **Approximate Methods**. Section 5 presents the evolution of **optimization methods to handle dynamic environments**. Finally, section 6 concludes this chapter.

2 Complex Problems Optimization: Description

Complex problems are non-linear, composed of many parameters linked together, it is hard to directly solve them. To remedy this, formalization can be applied.

Optimization problems are often modeled through a **Constraint Satisfaction Problem** (**CSP**). A CSP is defined as a triple $\langle X, D, C \rangle$, where X is a set of variables, D is a set of domain values and C is a set of constraints. Thus, each variable x_i can take a value of the domain d_i respecting constraints. Many methods exist to solve a CSP, for example builds the tree of solutions and explores it to find a solution.

For complex problems, it is hard to find a solution (if it exists), and, moreover, the problem is over-constrained. Thus, constraints are associated to *cost function*. This formalism is the **Constraint Optimization Problem** (**COP**) [Karaboga and Basturk, 2007]. In such a formalism, the set of variables representing problem entities must be assigned a value of a given domain to maximize the objective function.

Finally, **Distributed Constrained Optimization Problems** (**DCOP**) [Maheswaran et al., 2004] is a COP where variables are managed by entities named *agents*. Agents work together to satisfy constraints and optimize a global function. Well-known algorithms developed following this approach organize agents into a hierarchy based on the constraint definition like **ADOPT** (*Asynchronous Distributed Constraint Optimization)* solver [Modi et al., 2005].

3 Analysis Criteria of Planning Systems

I

In the literature, many classical optimization methods have been applied for solving Earth observation planning problem. In order to compare and qualify those optimization methods, we rely on characterization proposed by [Kaddoum et al., 2009] to propose analysis criteria. In Chapter 6, we rely on those criteria to evaluate the developed system and compare it to a standard algorithm, currently used to plan mission of satellites. The proposed criteria are grouped into three categories:

1. **development methodologies**: not all approaches used to solve a complex problem are equal on their development. This criterion qualifies the difficulty to use a methodology to implement a planning system,
2. **characteristics of the system**: the system can be evaluated without having to execute it. This criterion studies the system complexity, and its ability to manage dynamics,
3. **performances**: those criteria allow to evaluate the results obtained by the implemented system. To be adequate to the application of this thesis, we consider general performance criteria, but also performance criteria dedicated to the mission planning problem.

3.1 Development methodology characterization

- **Modeling**: in complex problems, many entities interact together. Thus, each of them has to be modeled and implemented in the system. But not all optimization methods allow the same importance and fineness of the implementation of the different entities. This criterion allows to evaluate the gap with general methods.
- **Ease of implementation**: Depending on the method, the design of each entity could be easy or not. This criterion aims to evaluate the necessary effort required to adapt general concept in order to develop a specialized method.
- **Importance of the heuristic**: Approximate optimization methods use a heuristic to guide the search and to obtain a solution faster. But the algorithm does not necessarily depend only on this function. This criterion allows to determine what is the most important in the algorithm.

3.2 Intrinsic characterization of the system

- **Ability to handle dynamics**: complex problems must manage a dynamic environment. In such systems, new entities or constraints could occur at any-time: the system is indeterministic.
- **Scalability**: complex problems possess a huge number of entities. In the spatial field, the increasing size of constellations implies that a huge number of requests must be planned. This criterion indicates if the system manages this set of entities, and how it is its ability to handle a larger problem.

– **Genericity of the heuristic**: each method has its own heuristic. This one is tuned for the system in which it is used, in order to obtain the best solution. Still, a generic heuristic is tolerant to changes. On the contrary, if the heuristic is not generic, if the problem evolves, the heuristic must be redesigned. This criterion is useful to know if the system is specified to a special type of problem or if can be used to several types.

3.3 Performances Criteria

– **Execution time**: It is important to quickly provide a good solution. Planning systems can be compared according to their speed performance.

– **Number of calls to the guidance function**: The guidance function is the function that determines the satellite maneuver(s) between two acquisitions. This function is very expensive in calculation time. Thus, the number of calls to this function must be minimized.

– **Balance of the solution**: This property concerns the planning of a constellation. Indeed, the interest of a constellation is to balance the load between the satellites. The system must take into account this property otherwise some satellites will be loaded (or not) than others.

4 Optimization Methods Survey

Complex problems solving is a major challenge for today's researchers. For this reason, many approaches have been developed. In the next sections (4.1 and 4.2), we present significant optimization methods, and we compare them using criteria previously defined. For each of them, we focus on their application to the Earth observation planning problem. Figure 2.1 presents a classification of some optimization methods.

4.1 Exact Methods

Exact methods guarantee to find the optimal solution, if it exists. If the problem is complex, and possesses a combinatorial space search, such methods are difficult to apply.

Algorithms belonging to this category explore all the space search. The performance of this method is linked to the size of the space search. This limitation explains why those methods are not used to solve problems with a large space search.

4.1.1 Tree Search, Branch and X

The **Branch and X** family is one of the most popular exact algorithms [Padberg and Rinaldi, 1991]. Those algorithms build a tree representing the space search. By constructing such a tree, the algorithm browses the space search.

For example, **Branch and Bound** algorithms is a classical enumerative method. This algorithm uses a tree to explore the space search, where nodes represent partial solutions.

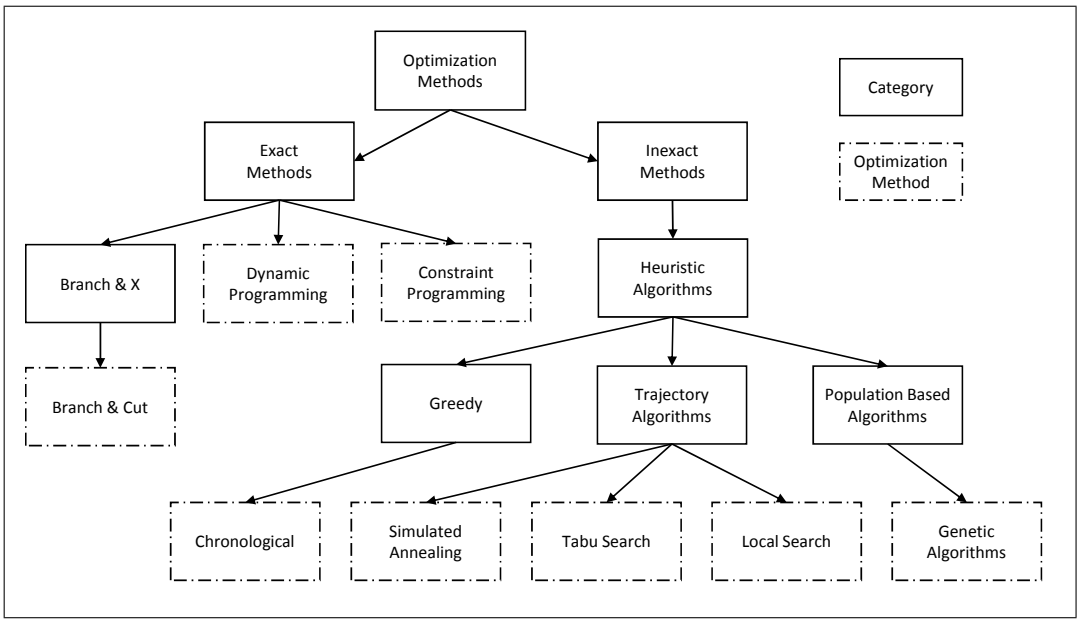


Figure 2.1 — Optimization methods by [Talbi, 2009]

Two steps characterize this algorithm: **branching** and **pruning**. Branching step is a splitting procedure that divides a set of possible solutions. The second step, pruning, as it names suggest cut branches that do not lead to the optimal solution.

Algorithm 2.1: Branch and Bound simple algorithm [Narendra and Fukunaga, 1977]

```
1  List Nodes;
2  Value evaluation ← ∞;
3  Solution s;
4  while Nodes != ∅ do
5  |   n ← takeNode(Nodes);
6  |   if n is a single candidate AND eval(n)< evalution then
7  |   |   evaluation ← eval(n);
8  |   |   s ← n;
9  |   else
10 |   |   Add nodes of n to Nodes;
11 |   end
12 end
```

Such a method allows to cut the search and so to only explore the part where the solution is located. [Bensana et al., 1996] compares the use of two branch and bound algorithms (first using a depth first strategy and second using a first-best strategy). Their results prove that when the size of the problem becomes too large or when the number of constraints is too important, both algorithms fail to find a solution in a reasonable time. [Pemberton, 2000] proposes the same remarks about Branch and X methods, like Branch and Cut algorithm [Padberg and Rinaldi, 1991].

4.1.2 Linear Programming and Constraint Programming

An efficient way to solve a problem modeled by a COP is **linear programming**. In this paradigm, constraints and objective of the optimization is a linear function of the variables. An illustration could be the production of products by a factory to maximize profits, knowing the availability of resources. This problem could be transformed into a linear function to maximize under constraints.

A method usually applied to solve a linear programming optimization problem is the **Simplex Algorithm** [Dantzig et al., 1955]. First step is to construct a polyhedron related to the COP, where vertexes represent values and edge constraints between them and to define a cost function. Simplex explores the edges: if the current edge is not optimal, the Simplex goes to a neighbor where the cost returned is lower. As the number of edges is finite, the optimum is found where no neighbor provides a lower cost. Figure 2.2 shows polyhedra, the start and the path to find the solution.

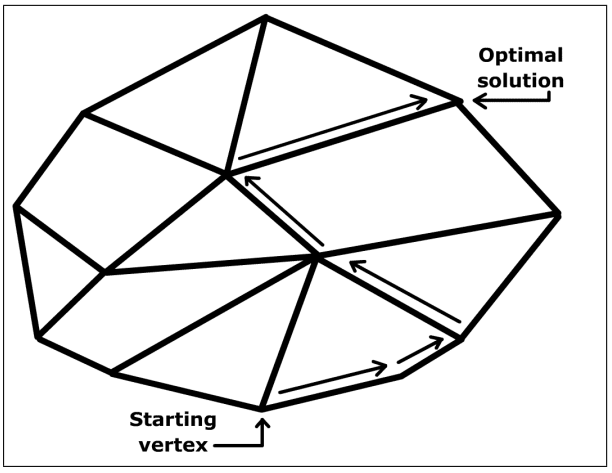


Figure 2.2 — Simplex application

Linear programming allows a more natural formalization and to add a maximum of constraints. In addition, many tools exist (like *CPlex*) to solve such a problem [Bensana et al., 1996]. Still it remains hard to apply it to the planning problem. Indeed, the initial step of modeling the problem as a CSP is very difficult to complex problems. Moreover, because of this huge **combinatorial**, constraints can be linked together. **Non-linearity** implies that it is very difficult to apply linear programming to complex problems, and therefore to Earth observation planning problem.

The important combinatorial is not the only limitation. Indeed, the **dynamics** make the Simplex impracticable. For each change, the polyhedra must be rebuilt. It is difficult to make it evolves in real-time to handle dynamics.

4.1.3 Constraint Programming

A paradigm often used to solve a COP is **Constraint Programming** (**CP**) [Rossi et al., 2006]. From such a COP, CP algorithms use propagation and filtering techniques to eliminate non-feasible solutions. Such methods allow to find a solution (if it exists) and

to prove that a solution does not exist at the end of the search.

The propagation of constraints consists in the reduction of the variable's domain where this part of the domain cannot lead to a solution. All of the constraints must be satisfied, thus if a variable possesses a value that does not satisfy at least one constraint, this value is not in the solution. This reasoning can be applied to each constraint to find values where a constraint is not respecting and so suppress this value.

[Lemaître et al., 2002] propose the use of such a method to schedule a mission of Earth observation satellites. Nevertheless, authors used additional constraints in order to limit the time of the search. For example, requests having very low chances to be selected are ignored. Simplification is also used in [Wang et al., 2016] where several approaches are successively applied, in particular constraint programming and linear programming.

4.1.4 Dynamic Programming

The principle of the **Dynamic Programming** (**DP**) is to recursively decompose the problem into simpler sub-problems. DP is built on the Bellman's Principle of Optimality [Bellman, 1952]: *"an optimal solution can be solved by optimal local sub-problems"*. In this case, a solution S is equivalent to the sum of the solutions of all its decomposition. The difficulty of the approach is the problem decomposition.

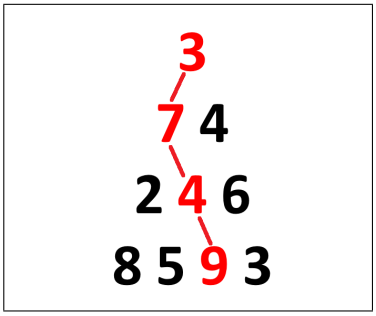


Figure 2.3 — Pyramid Number

A simple example of the Dynamic Programming is a pyramid of numbers where the goal is to find a path composed of numbers (one for each line), where the sum of the numbers of the path is maximal. At each line, a number has as neighbors the one or the two numbers directly in the line up above it.

For example, in Figure 2.3, the solution is **3 + 7 + 4 + 9 = 23**. Here, the solution could be found easily while it exists $2N - 1$ possible paths, where N is the height of the pyramid. In this example, 15 different paths are possible for instance. In the pyramid number problem, Dynamic Programming is adequate as the problem can be easily subdivided. Indeed, the path is built by computing the maximum for each line. Then, by applying the approach from the top to the bottom, each number can be substituted by the maximum sum that leads the top to it. In our example, the result can be obtained in 4 steps (one for each line), and then pull up the result to find the path. For each number of each line (starting from the top), the maximum value path is computed, using the maximum of the previous lines. Finally, when all lines have been treated, the maximal path is the path that leads to the final maximum.

Here, the maximum is 23 and by following these maximum, the path is **3 -> 7 -> 4 -> 9**.

[Lemaître et al., 2002] show that a **simplify version** of the Earth observation planning problem (dependencies between requests are removed) can be stated as an oriented graph where leaves are pairs (i,t) with i an image candidate at the time t, and edges represent possible transitions. On each edge a weight is associated. Thus, the optimal solution to the planning problem is the longest path in the graph. Dynamic Programming is applied to this problem, the longest path (LP) between a and z is $LP(a,z) = max(LP(b,z) + L(a,b))$ where $L(a,b)$ is the maximum length of all paths between a and b. In the literature, other attempts to apply dynamic programming to the space planning problem can be found [Hall and Magazine, 1994], they always face the same problem: non-linearity imposes simplifications.

Even though the Dynamic Programming allows to reduce the exploration, this method suffers from some limitations. Like Constraint Programming, this method cannot be applied to problems that are **highly coupled and complex**. Indeed, more they are linked in the system, less **the whole problem can be decomposed into sub-problems**. A solution could be to add history ([Lemaître et al., 2002]), but this approach could *phagocyte* the memory. Thus a massive complex problem, like the planning problem where many requests are linked (like stereoscopic requests), will need a huge volume of memory to be executed.

4.1.5 Exact Methods Analysis

Even if Exact Methods provide optimal solutions, they cannot be applied to complex optimization problems. In the operational context, where the duration of execution must be minimal, it is also a strong limit. Indeed, as they explore the whole space search (even if some techniques allow to avoid exploring some parts), the large number of constraints and variables make those approaches **extremely slow**. Moreover, **dynamics** cannot be taken into account: any change imposes to rebuild the search tree.

To overcome those problems and to limit the size of the space search, a solution is to simplify the problem. Nevertheless, even if the algorithm can find a solution quickly, it is possible that the global solution is removed from the space search. Another simplification is applied to make the problem compatible with well-known solvers. But here again simplifications can be difficult to proceed and they can ignore some solutions.

4.2 Approximate Methods

To overcome the principal drawback of the exact methods (their slowness), approximate methods are often uses. Different approximate methods exist, they differ in the **heuristic** used to guide the search and the quality of the obtained solution. Indeed, as they do not explore the whole space search, they do not necessarily provide the optimal solution. They are mainly used in space mission planning field where the delay of production can be critical: For example, the plan must be constructed and validated at least 15 minutes before it is uploaded to the satellite.

4.2.1 Greedy Algorithm

A **greedy algorithm** follows a heuristic that consists of making the local optimal choice at each stage with the hope of finding a global optimum. The principle of this algorithm is that when a choice is made, it is never challenged. The most common example is the Knapsack Problem: from a set of items, each with a mass and a price, what is the best combination of objects to fill the knapsack at its maximum while maximizing the total value. To solve this problem using a greedy algorithm, the idea is to first rank the collection by efficiency, for example using a ratio weight/value, and secondly to select most **efficiency** objects until saturation. The Knapsack Problem ([Akçay et al., 2007]) is often used to model the planning problem: *which order to select and add to the plan in order to maximize the number of the treated orders respecting constraints?* Algorithm 2.2 presents the knapsack problem solved by a Greedy Algorithm.

Algorithm 2.2: Greedy algorithm to solve knapsack problem

```
1  List Objects;
2  sort(Objects);
3  weight ← 0;
4  weight_t ← WEIGHT_BAG;
5  for Object o: Objects do
6  |   if weight(o) + weight <= weight_t then
7  |   |   pick(o);
8  |   |   weigh_t+= weight(o);
9  |   end
10 end
```

The Greedy Search proposes a good compromise between computation time and the quality of the solution. It is the most commonly used algorithm in the spatial domain. Moreover, using the knapsack model the problem can be easily formulated. ([Vasquez and Hao, 2001], [Vasquez and Hao, 2003] and [Wolfe and Sorensen, 2000])

In the space planning field, different variations of the Greedy Algorithm exist:

Chronological Greedy Search [Bianchessi et al., 2007]: the greedy search is chronologically performed. At the time t where the search is looking to place an acquisition, the more pertinent acquisition is chosen (generally the more urgent). This one is selected regarding rules of temporal sequence, then the algorithms search for an acquisition to put at the end of the chosen one.

Hierarchical Greedy Search [Wang et al., 2011]: several criteria are used to sort requests: priority, image quality, weather information, *etc.* Then, the list is unstacked and each request is inserted at the best place, respecting constraints.

Stochastic Greedy Search [Frank et al., 2001]: this version is used when the power of the calculus is low or when response time must be weak. From a state of the mission plan called *reference*, a short plan is constructed by randomly choose requests and evaluate the new plan: if it is better, it becomes the *reference*. Generally the algorithm is launched for the period of time that the mission center disposes between satellite's links. At the end of this

period, the *reference* is the final plan.

An important point with greedy algorithm is that choices made so far highly influence the choices to make. Choices are made iteratively, one at a time. Thus, each given problem is divided into a smaller one. A greedy algorithm never reconsiders its choices. By that a greedy algorithm produces a plan rapidly. This is also a limit: **if bad choices are made, they badly impact the plan**.

Moreover, this kind of approach is **not suitable for dynamic environments**. Dynamic insertion of a new request implies to restart the planning system and to ignore all already planned request which is in the plan scheduled after the new one.

The choice of the heuristic used in the greedy algorithm is important. Indeed, **the quality of this function impacts the quality of the solution**. In addition, a heuristic is designed and tuned for a kind of problem. If the problem changes (for example new constraint), it is compulsory to modify the heuristic. Thus, **greedy algorithms are not generic**.

As they are easy to implement, greedy algorithms are often used as a working basis, for example in [Lemaître et al., 2002]. Indeed, the heuristic, the heart of the algorithm, can be used to build other approaches, like local search.

4.2.2 Local Search and Trajectory Algorithm

Local Search algorithms start from an initial solution (which could be empty), and iteratively improve it using different methods. Differences between a solution and the previous are thin: solutions are called **neighbors**. In the literature, we found many different local search procedures [Bräysy and Gendreau, 2005]. Their difference is in the neighborhood exploration. There is no standard algorithm, to produce a high-quality solution, many parameters must be tuned.

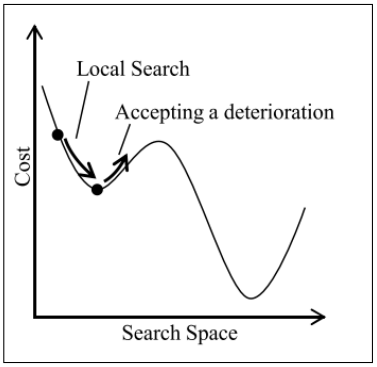


Figure 2.4 — Simulated annealing

Simulated Annealing is inspired by a process used in metalworking industry [Kirkpatrick, 1984]. In this process, alternative slowly cooling and heating phases are applied to a metal in order to minimize its energy, and so produce a high quality metal. The two parameters of this method are the **temperature** and the **energy**, representing the cost of the solution. A new solution can be accepted if it improves the current cost. Another way is using a probability factor based on the temperature. Figure 2.4 represents the evolution of the energy during the solving. The use of the temperature allows to avoid local minimum.

In the Earth observation planning problem, [Wu et al., 2014] used two neighborhoods. The first neighborhood is selected and inserted into the plan. If some conflict appears when the new task is inserted, task in conflict is removed. The second is built using migration. If a scheduled task can be performed at another time, the algorithm uses the free space to try to insert new task. The temperature is calculated using $\lambda_i = \lambda_{min} + p.ln(1 + r_i)$ where λ_{min} is the minimum value of the temperature, p a weight coefficient and r_i the number of consecutive bad moves. This local search approach is also used in [Globus et al., 2004] and [Globus et al., 2003], in their experiment local search use less memory than other approaches.

Tabu Search [Glover and Laguna, 2013] is also a Trajectory Algorithm. The difference between it and the Simulated Annealing is the way it explores the space search. This algorithm uses a list of already explored solutions to avoid re-exploring them. Figure 2.5 details the process. Moreover, tabu search is quite easy to implement, regarding other inexact approach.

There are many declinations of this algorithm, applied to Earth observation planning problem: [Bianchessi et al., 2007], [Lin and Liao, 2004], [Lin et al., 2005] or [Zufferey et al., 2008]. [Vasquez and Hao, 2001] use a Multi-Knapsack Problem formalization to propose a Tabu Search algorithm to solve the planning problem providing good results.

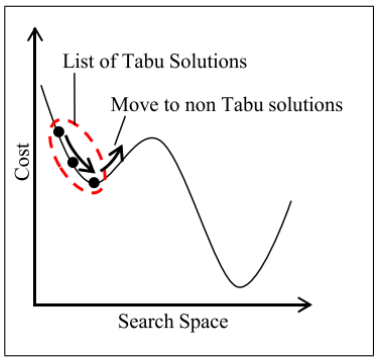


Figure 2.5 — Tabu Search

Local searches possess several strengths. First, those methods are quick enough. The general behavior is easy to implement. They produce good quality solutions and their functioning is intuitive.

First, to produce good results, such methods **must perfectly be tuned to each problem**. The problem must be perfectly analyzed and known to fit the algorithm. This is a weakness for complex optimization problems in dynamic environment like the Earth observation planning problem. Indeed, as a complex problem, it is difficult to perfectly know each entity and all links and dependencies between them. It is also a drawback if the problem evolves, because that implies to tune again the local search algorithm.

Second, local search methods try to modify a solution to improve it. But limit the exploration can block them in **local optimum**, and so never find a better optimum.

Finally, those methods do not handle **dynamics**. Indeed, new requests (or constraints) impose to regenerate the initial solutions.

4.2.3 Genetic Algorithms

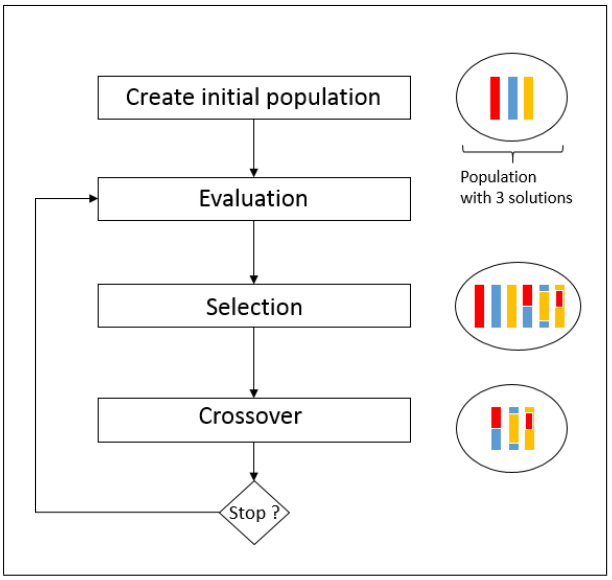


Figure 2.6 — Genetic Algorithm

Genetic Algorithms can be considered as a local search, still their behaviors are quite different. Genetic Algorithms are inspired by Darwin's theory of evolution [Darwin, 1859]. They use nature inspired procedures to make an initial population, representing a set of possible solutions, toward an optimal solution. In the Nature, the weakest individual will disappear while the strongest will engender a new generation that will possess good genes. Randomly, mutations could occur on genes. If a mutation is good for the specie, it will be transmitted to the next generation, else (if it weakens it), the individual has strong chances to be eliminated. This concept is the heart of Genetic Algorithms where genetic operators (**selection, crossover and mutation**) are applied to make the initial population evolve, most of the time randomly chosen. Figure 2.6 describes the algorithm while Table 2.1 shows vocabulary used in genetic algorithms and its comparison to nature.

Nature	Genetic Algorithm
Set of individuals	Population
Individual	Solution
Gene	Part of a solution
Generation	An iteration

Table 2.1 — Comparison between Nature and Genetic Algorithms

Selection. At each generation, a part of the population must be selected to generate the next generation. Each individual is evaluated using a *Fitness* function. This fitness function is problem dependent. If the problem change, the fitness function must be adapted.

Evolutionary Procedures. Now that a part of the population is removed, a new generation can be created. For that, **crossover** and **mutations** are applied. Crossover consists in combining two individuals (*i.e.* the two parents) in order to create a new one (*i.e.* a child) composed by characteristics of both parents. Crossover is repeated until the size of the new population be equal to the initial size. To this operator is also combined a probability

of mutation. Each gene can randomly change. Thus, new properties could emerge and be selected. The mutation factor must be correctly chosen. Indeed, if it is too important, good properties could be erased.

The termination could be decided by several factors: a solution that satisfied a maximum of constraints is found, the number of allowed generations is reached, the time of calculation is over, *etc.*

Mutations allow to diversify the population while crossovers generate a new solution melting several solutions. Thus, Genetic Algorithms are quite easy to describe and to understand. That is why, they are often used in optimization. Still, several limitations can be underlined. First, to create a population configurations must be binary encoded. This step is difficult, depending on the problem. Second, the fitness function must be able to determine the distance between the current population and the optimal. Finally, Genetic Algorithms **need a lot of memory and are slow**: evolution is a long process, many cycles and calculation times are necessary to produce a *potentially* good solution.

The main idea of this algorithm has been be applied to planning problems ([Li et al., 2014] or [Soma et al., 2004]). [Mansour and Dessouky, 2010] propose such an algorithm to schedule Spot 5 satellite. In their approach, the population is composed of a gene for each request, and those genes holds 6 alleles, each representing the kind of acquisition. To make their population evolve, authors have tested four selection methods: Tournament, Ranking, Roulette Wheel, and Uniform. Their results prove that on a small instance, a *CPlex* approach is better while a Genetic Algorithm could be a good alternative for soling large scale problems. Note that authors decided to stop both the algorithm after **ten hours** of execution, which is too long regarding operational conditions.

4.2.4 Approximate Methods Analysis

All these algorithms, even if they produce quite good results, have limitations [Bräysy and Gendreau, 2005].

First, 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 be readjusted.

Second, dynamic is hard to be integrated. Indeed, when adding requests during the process of 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.

4.3 Synthesis of Approximate and Exact Methods

In this chapter, we propose an overview of the different optimization approaches, each with its own pros and cons. Those methods have been categorized into 2 categories: exact and approximate methods.

Exact methods guarantee to find the optimal solution (if it exists) by crossing the whole space search. In this thesis, the considered problem is highly combinatorial, so the space search is very large. Exploring it to find a good solution in a reasonable time is impossible. Exact methods cannot be efficiently applied. Table 2.2 summarizes for each exact method studies in this chapter their positions for the defined criteria.

	Exact Methods		
Criteria	Dynamic Programming	Constraint Programming	Branch and Bound
Modeling	- -	+	+
Ease of Implementation	-	+	-
Dynamism	-	-	-
Execution Time	+	-	- -
Scalability	- -	- -	- -
Balance of the Solution	+	+	+

Table 2.2 — Exact methods synthesis

Approximate methods are a credible alternative. Indeed, those methods do not consider the whole space search. Thanks to *heuristic functions* they ignore some parts and find a solution faster. Nevertheless, there is no certitude on the optimality of the solution. Thus, the difficulty is to design a heuristic that allows to find a solution close to the optimal one in a reasonable time. Thus, heuristic-based approaches are highly dependent on the problem: if the problem change (new entities, new constraints) the heuristic must be re-design. Indeed, a generic and efficient heuristic in all cases is impossible to design. This limitation is compliant with the No Free Lunch theorem [Wolpert and Macready, 1997] that said if an algorithm outperforms another on certain types of optimization problems, there must be other classes of problems where this algorithm performs worse. Table 2.3 highlights the positions of the studied approximate methods for the defined criteria.

Finally, a common limitation of all of exact and approximate approaches is that they cannot take into account dynamic events. Still dynamic is an important factor in the mission planning problem, new methods able to manage this property must be studied.

Criteria	Approximate Methods			
	Greedy Algorithm	Simulated Annealing	Tabu Search	Genetic Algorithm
Modeling	-	-	-	- -
Ease of Implementation	+ +	-	-	- -
Dynamism	- -	-	- -	-
Execution Time	+	+	+	- -
Scalability	- -	-	-	-
Balance of the Solution	- -	+	-	+
Heuristic Importance	+ +	+	+	+
Heuristic Genericity	- -	-	-	- -

Table 2.3 — Approximate methods synthesis

5 Optimization in Dynamic Environments

In many complex problems, the environment dynamic evolution must be taken into account. In the previous sections, we exposed that classical approach cannot manage this dynamic. Earth observation satellites planning problems are dynamic optimization problems. All information about the problem is not known at the planning beginning and different perturbations (urgent request, new weather forecast, *etc.*) could occur at any-time. The environment is **dynamic** and highly **non-deterministic**. The three sources of dynamics are:

- A change in request to plan: new requests can be submitted or existing requests can be deleted,
- A change in the constraints: users could modify the constraints they have associated to their orders,
- A change in resources: for example a new weather forecast changes the cloud cover.

Classical optimization approaches fail to manage properly this dynamic. Of course, a first way to handle the dynamic is to reapply the optimization method after each change (and so to restart from scratch). Given sufficient time, this approach produces a solution, but the time between perturbations (and so between re-optimization) can be short. Thus this approach could fail to adapt at each time. To avoid to restart from scratch, a memory can be used: if the new optimum is near from the previous one, the algorithm can quickly reach it. But, because of the non-determinism, it is impossible to predict where the next optimum will be. An important point is to correctly select the information to save.

5.1 Evolutionary Algorithms

The nature, through the evolution, is able to adapt to the dynamics of the environment. Evolutionary algorithms [Eiben and Smith, 2003] are algorithms inspired by natural evolution (Genetic Algorithms are a sub-category of those algorithms). [Jin and Branke, 2005] propose to group Evolutionary Algorithm into four categories:

1. **Generate diversity after a change**: when an event is detected, the diversity is increased and so the new optimum could easily be found.
2. **Maintain diversity**: instead of increasing the diversity when an event is detected, the population is diversified as most as possible. Thus, new constraints or variables can be dynamically taken into account.
3. **Memory-based approaches**: the use of a memory helps the algorithm to get back to previous solutions if the new optimum seems to be close to the previous one.
4. **Multi-population approaches**: instead of seeking for one local optimum, sub-populations allow to explore more regions.

In Section 4.2.3 we present genetic algorithms used in **static environment**. A variation of this method to handle dynamic is to insert new requests to plan (or constraint) after each population evolution ([Bierwirth and Mattfeld, 1999] or [Zakaria and Petrovic, 2012]). Thus, Greedy Algorithm can take into account those new parameters for the new generation. This approach belongs to the memory-based group as not all the genes are deleted. Nevertheless, this method is slow. Indeed, even if the algorithm does not restart from the initial population after each change, it must converge to a new solution.

5.2 Nature Inspired Algorithms

Algorithms of this category are inspired by the collective behavior of social species such as ants, bees, fishes or birds [Dorigo et al., 2008]. The most studied and used nature inspired optimization algorithms are **Ant Colony Optimization** (*ACO*) and **Particle Swarm Optimization** (*PSO*). Such approaches are classified as multi-agent systems. An agent of such a system is defined as follows [Ferber, 1999]:

An **agent** is an entity that:

- is located into a computer system,
- has communication mechanisms,
- possesses local objective(s) and local resources,
- has a local perception of the other agents,
- can interact with other agents,
- follows rules to satisfy its objective while using its resources and knowledge and communicating with other agents, that can help it.

5.2.1 Particle Swarm Optimization

Particle Swarm Optimization algorithms (*PSO*) tackle the problem of a dynamic environment by maintaining a diverse population of solutions. In such methods, agent behavior is inspired by natural organisms like bird or fish schooling [Kennedy, 2011]. In such group, animals in movement keep an optimal distance between them using local perceptions. Thus, behavior of each animal is linked to the behavior of the neighbors: each individual try to optimize its chances to find food, to escape a predator by following trends.

In PSO, each agent, named *particle*, is randomly located into the space search, with a certain speed, and each is a candidate solution for the problem. Each particle can memorize the best solution visited and communicates with others. Thus, a particle is led by its neighbors and its perceptions to find a solution. Thus, the whole particles converge toward the optimal solution. Figure 2.7 illustrates a swarm converging to a solution. PSO have been applied to multi-satellite mission planning [Chen et al., 2012]. In their study, authors randomly generate the initial solution, and they had to well tune the algorithm in order to quickly reach a solution.

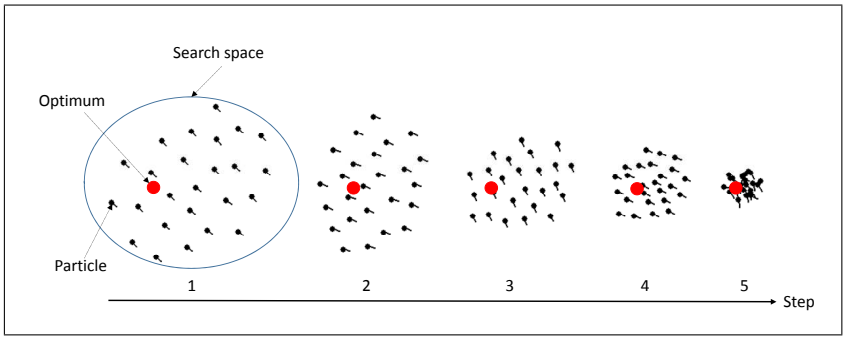


Figure 2.7 — PSO example

5.2.2 Ant Colony Optimization

The Ant Colony Optimization *(ACO)* algorithms are multi-agent systems where each agent imitates the cooperative behavior of ants [Dorigo et al., 2008]. The main idea of ACO is to copy the simple ant behavior using communication mechanisms.

Figure 2.8 illustrates the behavior of an ant colony to handle a perturbation while they keep bringing food to the nest.

- In the nominal state (2.8.A), ants know where the nest is. To find food, they follow a pheromone trail: when an ant carries food, it goes to the nest while putting on the ground pheromones (pheromone is the communication mechanism).
- When a perturbation occurs, the trail is obstructed (2.8.B).
- To continue the mission (bring food to the nest), ants must re-explore the environment to find a new path. Here two paths are possible: one, the shorter, on the top, and the longer on the bottom (2.8.C).

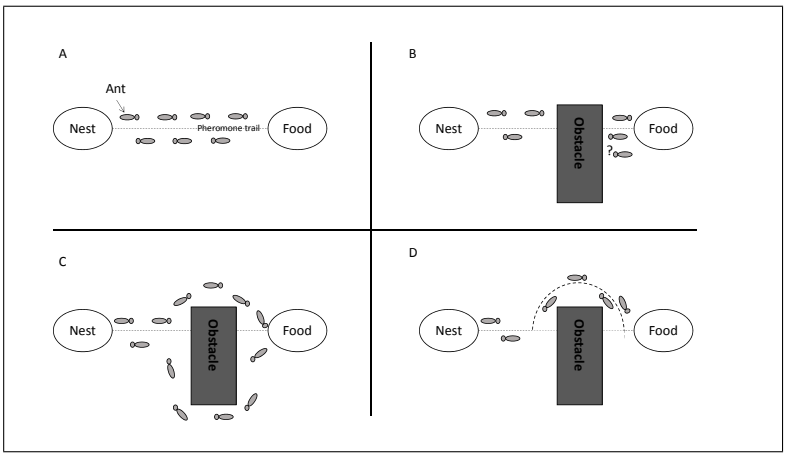


Figure 2.8 — Ant Colony

– Finally, a new path emerges (2.8.D), and it is the shorter.

In an ACO, ants interact indirectly via the environment by *tagging* promising regions, thanks to pheromones. At each step, the ants drop pheromones in the visited position. Pheromones are subject to evaporation and reinforcement processes. Thus, a region explored by a single ant is more likely to be forgotten because pheromones will disappear. Each ant construct a solution from partial solutions. Thus, from the current partial solution, an ant can decide to add a feasible solution component from the set of neighbors. Difficulties in such algorithms are the initialization. Indeed, many parameters have to be tuned: the number of artificial ants, the initial value of the pheromone and the evaporation rate.

[Wei, 2013] use a hybrid ACO to plan mission of radar observation satellites. Computing a mission for a radar satellite is easier than for a full agile satellite. Indeed, request is along the satellite ground trace: there are only pitch agility, and so the number of feasible acquisitions is lower than for an agile satellite. Meanwhile, ACO proposed by Wei converges to a solution in more than 30 minutes. Other attempts can be cited: [Wang et al., 2009], [Li et al., 2014] or [Wu et al., 2013], but authors always face the same problems.

5.3 Hybrid Methods

Instead of applying or adapt existing methods, another approach is to mix several approaches together to design a hybrid algorithm that fit to the problem. Many examples exist in the literature, we focus here on two of them, as they allow a dynamic planning.

First [Wang et al., 2007] propose a rule-based heuristic. The idea is to consider the planning problem as an inserting tasks problems, where tasks can arrive at any-time. To minimize the effect of a new task, the key is to minimize perturbations that it can produce, and so keep stability. For that, an iterative repair heuristic is presented, mixing repair algorithm ([Kramer and Smith, 2003]) and local search. This heuristic generates temporally infeasible solution, with the hope that new tasks will repair it. Even if this hybrid method works well on a small example, it is not suitable for real case scenarios with thousands of requests to plan.

Second hybrid method is proposed in [Wang et al., 2014]. This algorithm is like a greedy

algorithm: requests are sorted in a queue (sorted when new request comes) and are treated one by one. There are three mechanisms to insert a request: directly (if no conflict), with shifting or with deleting. Here again, the algorithm can dynamically treat requests but it is slow, because of the high number of decisions.

5.4 Self-Organized Systems

Recently, self-organizing systems have been used to solve dynamic problems. Indeed, they possess the ability to adapt themselves to their environment. For example, [Picard et al., 2015] propose to use the self-organization and reorganization to adapt Machine-to-Machine infrastructure. In [Kosak et al., 2015], an approach is presented to enable resource allocation in large systems of systems. [Balbo and Pinson, 2005] propose model allowing the synthesis, the evaluation and the update of available information in order to help operators monitoring a public transportation line.

Adaptive multi-agent systems have proved their advantages in several application fields we can for example name [Boes et al., 2013] who develop an adaptive multi-agent system for the dynamic control of heat engine or [Jorquera et al., 2013] who present an application to Multi Disciplinary Optimization problems. 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. For example, [Clair et al., 2008] propose an adaptive multi-agent approach to provide self-regulated manufacturing control.

5.5 Analysis of Optimization in Dynamic Environments

Optimization in dynamic environment tries to overcome problem that classical optimization methods cannot solve especially manage dynamic events.

Most of these approaches are nature inspired. The major drawbacks is the modeling of the *agents* that solve the problem. Nature inspired methods can handle dynamics, but they must be well tuned to correctly solve the problem. Finally, those methods start from an initial solution. This can be randomly generated or obtained through an algorithm. The choice of this first solution is important. If it is too far from the optimum, the algorithm will be slow, on the contrary being too close to it can make the algorithm blocks on a local optimum.

Other approaches are hybrid methods, most of the time based on approximate methods. Even if dynamic events are treated those methods are slow. Indeed, when a new event occurs it could disrupt the whole solution, implying many operations to stabilize it. The necessary time to converge toward a solution or to stabilize it after a perturbation could be important.

6 Conclusion and Analysis

In this chapter, three categories of optimization methods have been presented: exact methods, inexact methods and dynamic environment optimization methods.

Exact methods allow to find the optimum, but they cannot be applied in complex problems solving: they are too slow. Approximate methods cut the search space in order to be faster, but they introduce simplifications. Those one can modify the problem. Finally, design a good heuristic that fits the problem is also a difficult task.

But the major drawback of those two categories is that they cannot manage dynamics. To overcome this major problem, we present optimization methods in dynamic environments. Those approaches are developed to solve complex problems in dynamic environment. Nevertheless, those methods are slow when the problem is composed of many entities, and they must be well tuned to produce a good solution.

This state of the art highlights the fact that no existing method is good enough to solve complex problems under constraint, with dynamic environment and to be applied to the Earth observation planning problem.

3 Multi-Agent System, Self-Organization and Tools for the Study

« One day I will find the right words, and they will be simple. »

Jack Kerouac

Contents

1 Introduction

I

The purpose of this chapter is to introduce the theory and tools used is this work. It is organized as follows: first, multi-agent systems and their principal interests are detailed. Second, the Adaptive Multi-Agent Systems (AMAS) theory is presented. Finally, the last section presents a way to design adaptive multi-agent systems, the ADELFE toolkit, and introduces the AMAS4Opt model, a generic agent model to solve optimization problems using the AMAS theory.

2 Multi Agent System

Multi-Agent Systems (MAS) are computing systems composed of a plurality of entities: the agents. Such systems are *distributed*. Is not an entity knowing everything which performs the whole computation: calculus and knowledge are distributed among the parts of the system. Moreover, MAS are *decentralized*: there is no centralized control. In this section, agents and MAS are defined and their properties are exposed.

2.1 Autonomous Agent

First concept of software agent can be found in the '80s, more precisely in the Distributed Artificial Intelligence (DAI) field. In those systems, the *intelligence* was distributed in different entities: the **agents**. Thereafter, no real definition of *agent* has been proposed, but a consensus on agent properties is clearly admitted. Thus, [Wooldridge and Jennings, 1995] and [Ferber, 1999] define an agent as a software or physical entity that:

- is autonomous,
- is located in an environment and is able to interact with it,
- has communication abilities with other agents,
- has skills.

An agent follows a **life cycle** "Perception - Decision - Action". In the first step, the agent **perceives** information from its environment. In the **decision** step, the agent chooses what action to perform depending on its perceptions. Finally, in the last step, the agent performs **action(s)** previously decided. This life cycle allows agent to change its behavior to take into account dynamic events.

Different agent implementations exist. They differ on where the intelligence is implemented on the agent. We can for example cite reactive or cognitive agents [Gleizes, 2012]. A reactive agent reacts to the changes detected in its environment. Reactions, like reflexes, depend on its perceptions and internal state, and not on its memory. On the contrary, a cognitive agent is more complex. Such an agent possesses an explicit representation of its environment and the other agents of the system. Moreover, a cognitive agent possesses a sophisticated communication strategy. Most of the time, cognitive agents

are chosen for systems with a few agents (~10), while reactive agents composed huge systems (~100).

Other characteristics can be mentioned like located agents, for which perceptions depend on their place in the environment or its contrary communicated agents which can communicate with other agents, without location constraints.

Agents are different of objects (from a software point of view). Indeed, they are autonomous: they decide the action to perform, while objects are an execution unit, they execute what they have been requested for. Thus, agents are **proactive** and objects are **reactive**: they do nothing if no one requests them. In multi-agent system field, objects are implementation techniques.

2.2 Multi-Agent Systems and Their Properties

Now the notion of *agent* has been detailed, we can answer the question of what is a Multi-Agent System (**MAS**). Formally, [Ferber, 1999] proposes to define a MAS as a system that possesses a:

- set of entities E, located into an environment ENV,
- set of agents A such as $A \in E$,
- set of actions ACT, that agents can perform in ENV,
- communication systems.

A MAS is a set of agents, located into an environment, and interacting together to reach their local goal. MAS are interesting to naturally solve **decentralized** problems. Indeed, in such a system, knowledge and competences are **distributed** among the agents.

Moreover, a MAS can be **close** or **open**. A close MAS implies that agents are the same during the whole execution, on the contrary, in an open MAS the agents can appear or disappear from it. In such a MAS, agent can be added by humans or directly created by existing agents. The disappearance could also be caused by external operators or by a *suicide* ability.

Finally, agents of a MAS could be different or not. Thus, MAS could be qualified as **homogeneous**, if all agents are designed on the same pattern, or as **heterogeneous** if agents have different models.

2.3 Environment

As we previously said, a MAS is a system that is located in an **environment**, where agents evolve simultaneously. This environment includes what is external to the system. Thus, agents will interact through it, and manipulate it. For example, if we consider a wolf pack as a MAS, the environment corresponds to the forest, the preys, *etc*. Each agent also possesses its own environment. This one corresponds to the MAS environment and the other agents. In the wolf pack example, the environment of a wolf includes the forest, the

preys and the other wolves. Thus, the system and its environment are coupled together. The system acts on the environment that responds using **feedbacks**. We can distinguish here a feedback loop (Figure 3.1).

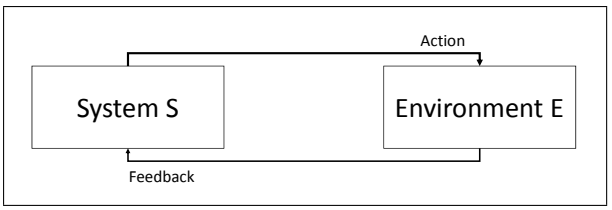


Figure 3.1 — The feedback loop

As the system and the agents, the environment possesses properties. In the literature ([Russell et al., 2003] or [Lind, 2001]), environment is often characterized using the following properties:

- **Accessible** / **Inaccessible**
  - Accessible: system can have complete, update and true information on its environment,
  - Inaccessible: only a partial information is available,
- **Continuous** / **Discrete**
  - Continuous: number of possible actions and perceptions is infinite,
  - Discrete: actions and perceptions are distinct and clearly defined,
- **Deterministic** / **Indeterministic**
  - Deterministic: each action has a single effect on the environment: the current state of the environment determines the next one,
  - Indeterministic: an action has no guarantee effect,
- **Dynamic** / **Static**
  - Dynamic: environment states depend on system actions but also on actions of different processes. External events can produce effects on the environment, those changes cannot be predicted,
  - Static: environment cannot evolve without system actions.

2.4 Interactions

A multi-agent system is a complex system. Indeed, a MAS is composed of many interacting parts, the agents, with simple and local behaviors. Each agent is autonomous and is not controlled from a macro level. Moreover, no entity controls the whole system. Interactions between agents produce the emergent property: the function of the system. Thus, to design multi-agent systems that produce the right function, the key is to find correct local interactions and implement agents with those rules. The Adaptive Multi Agent System

(**AMAS**) theory [Camps et al., 1998], proposes a theoretical framework to design such systems. This approach is based upon interactions between agents, and the **cooperation** notion.

[Jennings, 1999] distinguishes three kinds of interactions: **antinomic**, **neutral** and **cooperative**. An entity has an antinomic interaction if its action disturbs another entity in the accomplishment of its activity. If the action does not disturb but does not favor either, the interaction is neutral. Finally, if the behavior of an entity favor the behavior of another, the interaction between them is cooperative. Systems can also be classified. Thus, if all interactions between the system and its environment are cooperative, the system is in **cooperative** state, else if interactions are neutral or antinomic, the system is in a **non-cooperative** state.

From the agent point of view, cooperation is defined as their ability to work together in order to realize their objectives. Thus, four properties must be satisfied to ensure cooperative interactions [Glize, 2001]:

- **Sincerity**: an agent is sincere, that implies that it never lies,
- **Willingness**: a request is always satisfied if it is coherent with the agent state and if it has skills to perform it.
- **Fairness**: when it is possible, the agent with the lowest level of non-satisfaction degree is favored to be satisfied.
- **Reciprocity**: all agents know those properties and respect them.

2.5 Emergence

Traditionally, the way to tackle a problem or to design a software, is to adopt a bottom-up approach. The system is built from its foundations toward a higher level, each level brings its brick to the global system. A bottom-up approach needs to know the system *finality*: each part bringing something to this finality, this last must be *a priori* known. Such an approach is qualified as **reductionist**: a part can always be subdivided in other parts. [Oppenheim et al., 1958] use reductionism philosophy to classify scientific objects, this classification is shown in the figure 3.2.

This approach is quite good for complicated systems, where the global behavior is well defined. Thus, such systems can be easily decomposed in parts. But reductionism cannot handle complex systems. In these systems, many little parts are interacting and each has its own rules and objective [Ladyman et al., 2013]. Today, complex system study is successfully applied to several fields of science, from biology (shoal of fish) to neuroscience (neuronal network). Top-down approaches cannot be applied to study complex systems, the approach must be reversed: instead of considering the whole, it is necessary to study and design the little parts and their simple behavior and interactions.

In a complex system, the final function cannot be predicted from the knowledge of the local parts. As Aristotle has written: *the whole is greater than the sum of the parts*. This property is called **emergence**. Even if a global definition does not exist for this

I

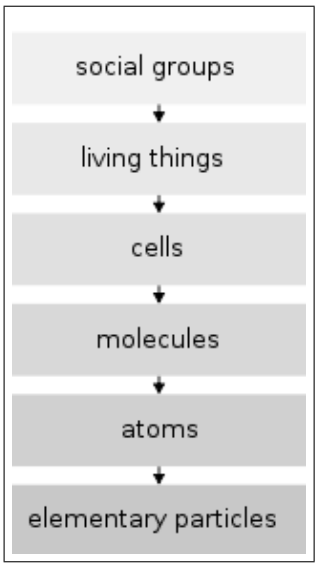


Figure 3.2 — The six levels proposed by [Oppenheim et al., 1958]

notion, an emergent phenomenon is something that is perceptible from a macro level and produced by sub-level interactions (the micro level) ([De Wolf and Holvoet, 2004] and [Di Marzo Serugendo et al., 2011]). In the shoal of fish example, it is the interactions between fishes (the sub-level) that produce the shoal (the macro level), and so the shoal cannot be predicted from the fishes point-of-view. As the emergent property is unpredictable from the micro level, it brings something *new* to the system. As it is new and unpredictable, the emergent property is decentralized: no single part controls it, but interactions from a lower level produce it. Table 3.1 lists some examples of *agents*, their interacting ways, and the emergent property.

Agents	Interactions	Emergent Properties
Social Insects	Phenomenon	Colony
Group of Animals	View, Imitation	Shoal of Fishes, Flock of Birds
Market Consumers	Buying & Selling	Prices

Table 3.1 — Some emergence example

Emergence cannot be only reduced to the interactions of local parts. The emergent property appears while the system is evolving: emergence is linked to dynamic properties. Moreover, the emergent property is not punctual but it remains during a certain period. At first, some fishes interact together and then, when there are enough to swim in the same way, the shoal emerges. Fishes keep this configuration during a long time, and the shoal evolves: new fishes, new forms (to avoid a predator for example), *etc.*

3 AMAS: Adaptive Multi-Agent System

In the previous section, agents and multi-agent systems have been defined. In this section, we define essential notions to present the theory on which this work is based: the Adaptive Multi-Agent System (**AMAS**) theory.

As defined in section 2.5, a complex system has no abilities to observe itself, and so it is impossible from the system point of view to judge if the correct behavior is realized. Thus, the judge role falls to an external observer. Indeed, only him can observe the whole system, and says if the system is **functionally adequate** (*i.e.* if it realizes the function for what it is designed). However, to self-organize, the system must evaluate itself. In the case of self-organizing MAS, agents do not know the global function, but only possesses local perceptions.

[Georgé et al., 2011] presents self-organization as mechanisms allowing the system to adapt to a changing environment. Thus, the system must reach several stable states to react to all perturbations. Self-organization is a mechanism producing those variations, allowing the system to explore different regions of its search space.

So, the key is to find a way to produce a good self-organization, in order to ensure the system to produce an adequate function. The **AMAS** theory proposes the cooperation as the answer. Indeed, the definition of **Functional Adequacy** indicates that a functional adequate system has no antinomic interaction toward its environment. Thus, all systems in cooperative state are functionally adequate. Functional adequacy and cooperation are the key of the theorem of Functional Adequacy.

Theorem. *For any functionally adequate system, there exists at least one cooperative medium system that fulfills an equivalent function in the same environment.*

A cooperative medium system is a system where all parts interact cooperatively. Thus, a MAS where all agents are in a cooperative state (*i.e. a cooperative medium system*) exists for each problem that is calculable. To be in a cooperative state, an agent must: understand the signification of all perceived message, perceive only useful information for its reasoning, and perform actions that only produce information that other agents understand. A demonstration of this theorem can be found in [Glize, 2001]. This demonstration is built on the inclusion of three different classes of systems: Functionally Adequate Systems (FAS), Cooperative Systems (CS) and Cooperative Internal Middle Systems ($CIMS$): $CIMS \subseteq CP$ and $CP \subseteq FAS$. Figure 3.3 illustrates those inclusions.

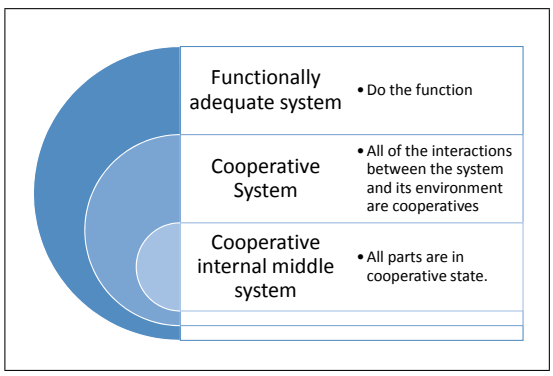


Figure 3.3 — The 3 states of an adequate system

3.1 Architecture of a Cooperative Agent

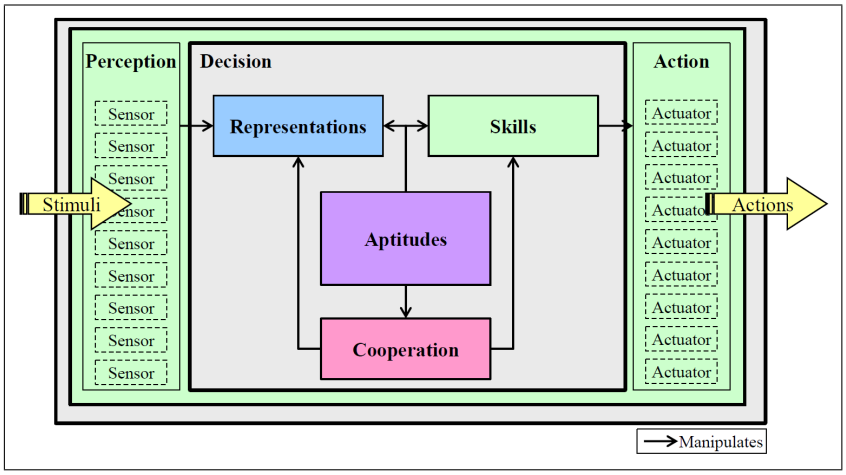


Figure 3.4 — The different modules of a cooperative agent [Bernon et al., 2002]

Cooperative agents are composed of several modules representing a partition of their physical, cognitive and social capacities (Figure 3.4). Each module represents a specific resource for the agent during its life cycle.

- Agent interactions are managed by two modules: perception and action modules,
  - the perception module represents the inputs the agent receives from its environment,
  - the action module represents the output and the way the agent can act on its environment,
- the skill module concerns the knowledge that enables the agents to realize their local function,
- the representation module concerns the beliefs an agent has on its environment and includes the representation of other agents and entities,
- the aptitude module contains generic and external tools an agent needs to accomplish its treatment,
- the cooperation module concerns all cooperative attitudes of the agent. It manipulates the skills and representations modules, in order to anticipate or detect and repair Non-Cooperative Situations.

3.2 Non-Cooperative Situations

Previous section implies that if a system is in cooperative state, the adequate function is produced. Therefore, a system is not permanently in cooperative state. Some situations, known as Non-Cooperative Situations (**NCS**) could occur. Indeed, agents could have concurrent goals or does not understand each other. In [Gleizes, 2012], seven NCSs has been identified.

– **Incomprehension**: the agent does not understand the message it has received,

– **Ambiguity**: a single message can be understood in different ways,

– **Incompetence**: an agent has no skill to treat the information it has perceived,

– **Unproductivity**: even if the agent has understood the message, it cannot get a useful information,

– **Conflict**: the action performed by an agent will be discordant with one performed by another agent,

– **Concurrence**: an action chosen by an agent puts it in concurrence with another one,

– **Uselessness**: the agent produces an action that is not useful for the system.

Figure 3.5 presents those seven situations and their occurrence in the agent life-cycle. Thus, incomprehension and ambiguity are linked to the perception step, incompetence, unproductivity and conflict to the decision one, and finally concurrence and uselessness to the action step.

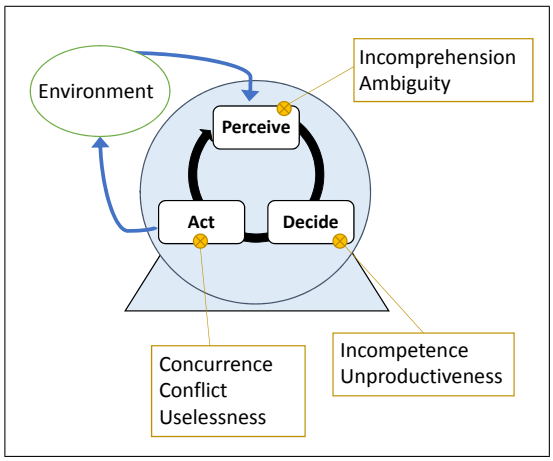


Figure 3.5 — NCS in the agent life-cycle

Thus, in an adaptive multi-agent system, each agent has mechanisms to repair those situations, and so to autonomously adapts its behavior to the context: tuning, reorganization and evolution [Capera, 2005].

– **Tuning**: the agent adjusts its internal state to modify its behavior,

– **Reorganization**: the agent modifies the way it interacts with its neighborhood,

– **Evolution**: the agent can create other agents or self-suppress itself when there is no other agent to produce a functionality or when a functionality is useless.

Local interactions allow the system to self-adapt to perturbations and so to handle dynamics without challenging the already reached solution. Agents always try to be cooperative to avoid NCS. They must possess mechanisms to anticipate and detect them

and to act in consequence. Thanks to those mechanisms, an agent is able to self-adapt its behavior, and so to modify its interactions. Thus, cooperation is the key of the self-organization of the system. The algorithm of an adaptive agent can be described as follows: if a NCS is detected, agent uses the self-adaptation mechanisms to come back to a cooperative state where it performs its nominal behavior. Thus, to solve a problem using an AMAS, the cooperation of all the agents is the key. As a cooperative entity, a cooperative agent spontaneously communicates its relevant information to its neighbors and helps its neighbor that encounter the more difficulties. If all agents are in cooperative state, the emergent function is the adequate function (Figure 3.6).

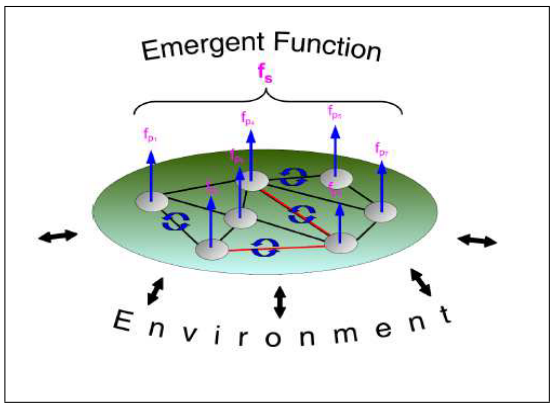


Figure 3.6 — Emergent function

3.3 Analysis

As decentralized and distributed systems, AMAS are relevant to solve complex problems. Where classical optimization approaches must handle the whole complexity, in an AMAS the complexity is treated at a local level by cooperative agents. Agents perform choices in a smaller space search. Distribution and decentralization allow to efficiently manage complex problems.

3.4 Some Applications of AMAS

The AMAS approach has been successfully applied to several application domains. Although it is impossible to dress an exhaustive list of all AMAS, we can quote several of them, illustrating the variety of applications:

- learning from demonstrations applied to robotic [Verstaevel, 2016],
- detection of anomalies in maritime survey [Brax, 2013],
- dynamic analysis of databases [Belghache et al., 2016],
- states evaluation for voltage regulation [Perles et al., 2017].

4 The ADELFE Methodology

Designing a complex system like an AMAS in particular is different from traditional approaches. There is a need to focus on local interactions between the parts of the system, and to use a bottom-up methodology. To help developers design multi-agent systems, different methods have been proposed. For example, **ASPECT** [Cossentino et al., 2010] provides a guide, from requirements to code, allowing the system modeling using a set of refinement methods. Other methodologies exist, we can for example cite **INGENIAS** [Pavón and Gómez-Sanz, 2003] or **PASSI** [Burrafato and Cossentino, 2002].

In this thesis, we rely on **ADELFE** [Bernon et al., 2002] (French acronym for *Atelier de DEveloppement de Logiciels a Fonctionnalite Emergente*, Toolkit for Designing Software with Emergent Functionalities). ADELFE is based on the Rational Unified Process (*RUP*) [Kruchten, 2004] development process and enhances it to design software with emergent properties [Bonjean et al., 2014]. The goal of this methodology is to guide the development of adaptive multi-agent systems, through five work definitions, from preliminary requirements to design and fast prototyping. While the first two work definitions are common toward other development methodologies, the last three are relative to an adaptive multi-agent system implementation. The five work packages of ADELFE are:

WP1 **Preliminary requirements**: in this first phase, a description of specifications between customers, users and designers is made. This description contains the inputs and the outputs of the system, in addition its limitations and constraints.

WP2 **Final requirements**: in this work definition, the preliminary requirements are transformed in a use case model. The requirements (functional or not) and their priorities are organized and managed.

WP3 **Analysis**: Once all requirements are well defined, the analysis begins. Identification and definition of entities is performed. An entity can be **active** or **passive**. An active entity is an entity possessing a certain autonomy, allowing it to change its state without external interactions. On the contrary, passive entities have no autonomy, there are resources or data. A cooperative agent is an active entity with cooperative behavior. The next step is the definition of the **cooperative agents**. Active entities are studied in detail: if it possesses autonomy, a local goal and interactions, such entities become cooperative agents. The analysis phase defines an understanding view of the system, its structure and identifies if the AMAS theory is required.

WP4 **Design**: the fourth work definition details the system architecture in terms of modules, objects and agents. These activities are important from a multi-agent point of view: agent behaviors and the NCSs are defined. At this point, the characterization of the multi-agent system is achieved.

WP5 **Implementation**: finally, the implementation of the framework and agent behaviors can be realized.

5 A Generic Pattern for Solving Optimization Problems: AMAS4Opt

Usually, complex optimization problems are formalized as a set of entities that must satisfy a set of constraints. In multi-agent systems, entities are commonly represented using agents. Depending on interactions between the system and its environment, the agent organization emerges and constitutes the problem solution.

ADELFE methodology helps to design software with emergent functionality. This methodology is not dedicated to an application domain but ADELFE stays at a high level of abstraction. This is a strength, ADELFE can be applied to several kinds of applications, but this is also a limit, an important effort is necessary to adapt it to a particular domain. To help the design of cooperative agents able to solve optimization problems, the **AMAS4Opt** model has been proposed by [Kaddoum, 2011]. This model proposes generic design patterns of two roles of cooperative agents: **"Constrained Role"** and **"Service Role"**.

Agents under "Constrained Role" are cooperative agents which, at a certain time, have a problem and need the help from other agents: those under the "Service Role". Thus agents under the "Constrained Role" are considered as activators of the resolution. Indeed, they bring the problem, and the solution can be reached only with the help of agents under the "Service Role". In optimization problems, several agents can request the help of the same agent under the "Service Role". As a cooperative agent, this last help the most critical one. This **criticality**, defined as the agent distance between its current state and its satisfaction state, helps the cooperation and enables the system to find a solution.

To ensure negotiations and interactions between agents, AMAS4Opt proposes four majors types of messages:

- **Request For Service**: this message is sent by agents under the "Constrained Role" to request a service from agents under the "Service Role". The message is specified using the knowledge the agent possesses. The agent also adds its criticality in the message, so the agent under the "Service Role" knows the criticality of the emitter,
- **Request For Information**: both roles can exchange this type of message, when they need supplementary information,
- **Answer To Request**: after the treatment of the request for service, agents under the "Service Role" reply using this type of message,
- **Information Message**: information judged useful by agents can be exchanged through this type of message.

The interaction diagram presented in figure 3.7 illustrates the generic behavior of the AMAS4Opt model with two agents: Constrained Agent (under the "Constrained Role") and Service Agent (under the "Service Role").

1. firstly, Constrained Agent searches for a service and estimates its criticality (actions 1 and 2), and then requests service to the relevant agent (here Service Agent),

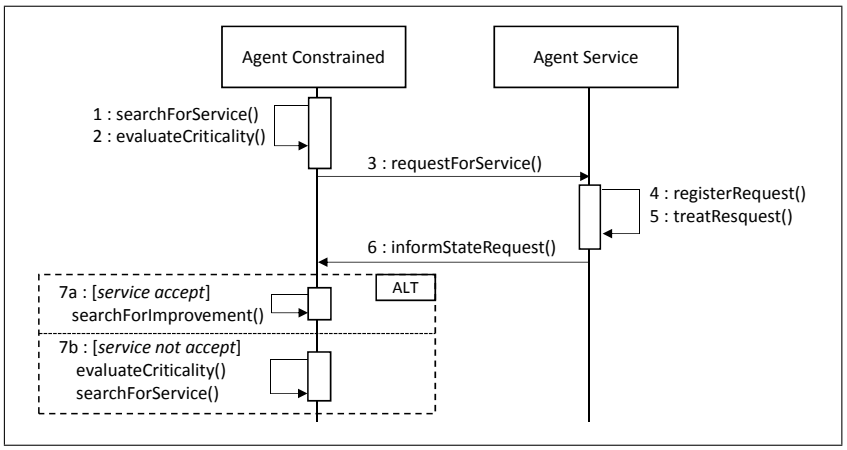


Figure 3.7 — AMAS4Opt simple interaction diagram

2. at the reception, Service Agent registers the request (4) and treats it (5). It then answers Constrained Agent (6),
3. if the service has been accepted, Constrained Agent searches for improvements (7a),
4. else it updates its criticality and searches for another service (7b).

AMAS4Opt model has been used to solve the manufacturing control scheduling problem [Clair et al., 2008], and for the design of complex products [Kaddoum and Georgé, 2012]. In those two problems, the AMAS4Opt model has shown adequacy and advantages. Nevertheless, agents under the “Constrained Role” cannot *a priori* select a service. This is a limit when several agents under the “Service Role” can help it. Another limits is that agents under the “Service Role”’ can have a complex behavior as they are the only ones that can satisfy agents under the “Constrained Role’.

6 Conclusion and Analysis

In this chapter, the AMAS theory, on which this thesis is based, has been presented. The decentralization and the distribution of AMAS make them a good approach to tackle the mission planning problem.

To design AMAS, the ADELFE methodology can be used. To overcome the genericity of this method, AMAS4Opt model has been detailed. But as we showed, the AMAS theory and the ADELFE methodology stay at a high level of abstraction, and the design of an AMAS is a difficult task. The AMAS4Opt model is a first step considering optimization problems, but it can be enhanced too.

The next part presents the contributions of this thesis. From a scientific point of view, with an enhancement of the AMAS4Opt model and patterns to manage complex problems using AMAS (Chapter 4). Then, relying on our contributions, the ATLAS planning system is presented (Chapter 5).

PART II

CONTRIBUTIONS

4 Models to Solve Complex Problems

« Sometimes, it is the people no one imagines anything of, who do the things that no one can imagine »

Alan Turing

Contents

1 Introduction

Classical optimization algorithms, like exact or approximate methods presented in Chapter 2, are not suitable to face the growing complexity of today's application. On the contrary, self-organizing systems, have shown their abilities to tackle such problems. Among those systems, Adaptive Multi-Agent Systems (**AMAS**) allow to naturally take into account a large number of entities and constraints, and provide self-organization mechanisms to allow the system to self-adapt to dynamics.

In this thesis, we rely on the **AMAS4Opt** model, proposed by [Kaddoum, 2011]. Using the AMAS theory, this model provides two roles of cooperative agents (Service and Constrained) and generic rules to solve Non-Cooperative Situations (**NCSs**). Moreover, the model fills the gap between the ADELFE development methodology and specific solving algorithms, providing good results on the tested uses cases.

This chapter aims to bring the four main contributions of this thesis:

1. **An enhancement of the Constrained Role**. To be satisfied, an agent under this Role must select the relevant agent under the "Service Role". But, if it exists several choices, the agent cannot take a decision *a priori*. There is a lack of information. To overcome it, and complete the knowledge of the agent, we propose a new cooperation mechanism that enables agents under the "Constrained Role" to perform a cooperative selection: the **cost**.

2. The **service delegation** for agent under "Service Role". In the AMAS4Opt model, agents under the "Service Role" can have a complex behavior: they are the only ones able to satisfy agents under the "Constrained Role". This complexity leads slowness and is not coherent with the AMAS theory where behaviors must be simple. We rely on ADELFE methodology to implement a pattern improving agents under the "Service Role" with service delegations.

3. **Constraint relaxation through the neighborhood**. In adaptive multi-agent systems, agents take local decisions based on their partial knowledge of the environment. To perform their task, they could need help from other agents, and so relax constraints to them. We propose a pattern for cooperatively relax constraints from neighbor to neighbor.

4. An implementation of **stop mechanism for adaptive multi-agent systems and its proof of coherence**. AMAS are systems that run continuously. At any-time, the organization of the agents constitutes a solution. Agents adapt their organization to the dynamic of the environment. When an external observer, like an operator, requires a solution, the organization could be in transition between two coherent states. Thus, it could be necessary to let the system finalize the adaptation before proposing a solution. To ensure that the proposed solution is coherent with the system constraints, we propose a proof of coherence and its implementation.

2 Enhancements of the AMAS4Opt Model

The **AMAS4Opt** model defined by [Kaddoum, 2011] and presented in Chapter 3 specialized the usage of the AMAS theory to solve complex optimization problems. The model is easy to implement, and defined the processes to follow to design agents. In addition, the model proposes precise rules to avoid or anticipate Non-Cooperative Situations.

Still, this model requires some adjustments and some features need to be tested. Firstly, an agent under the "Constrained Role" cannot *a priori* choose the relevant service. Their partial knowledge implies that agents need more information to select the best service. Secondly, agents under the "Service Role" can have a complex behavior causing a slowdown of the system. Moreover, this complexity is not coherent with the AMAS theory, where behavior must be simple. In this section we propose two enhancements of the two roles proposed by the model.

2.1 Choice of Service: Agent Cost

In an adaptive multi-agent system, the criticality represents the local measure of dissatisfaction of an agent. It helps to enable the cooperative attitude: cooperative agents always try to help the most critical agents. In the AMAS4Opt model, the criticality represents the difficulty for an agent to reach its goal. When requesting for a service, an agent under the "Constrained Role" computes its criticality. Using this measure, an agent under the "Service Role" selects the most critical agents to help.

As defined in the AMAS4Opt model, the criticality does not allow full cooperation between agents, this measure only allows agents with the "Service Role" to satisfy more dissatisfied agents, and by that to be cooperative. An agent under the "Constrained Role" cannot *a priori* cooperatively select an agent with the "Service Role". Agents under the "Constrained Role" do not know the state of all the agents under the "Service role" and the difficulty for them to perform the service. This lack of information prevents agents under the "Constrained Role" to take a cooperative decision, producing a Non-Cooperation Situation: there is **concurrence** between agents. Moreover, services can be partially known, an agent under the "Constrained Role" can discover a new agent under the "Service Role" later, implying to update its knowledge.

To avoid and to anticipate this problem, we propose the **cost**. The cost represents the difficulty for an agent under the "Service Role" to perform its activity and satisfy an agent under the "Constrained Role". The cost is computed by agents using their knowledge and representations. It combines different measures such as the number of solicitations, the amount of interaction links or the need to change organization of agents. Formally, the cost is represented as : $COST = \langle C_1, ..., C_n \rangle$, where each C_i is one of the cost measures.

In the same manner as the criticality measure definition in AMAS4Opt, the cost cannot be reduced to a single value. It is represented by an ordered list of measures. When an agent has to compare two (or more) costs, it makes the comparison by unstacking the list, element by elements. Thus, the agent does not compare the whole list: the comparison

stops whenever a difference is perceived. For example, if we consider two costs $C_1 = \langle 10, 5, 1 \rangle$ and $C_2 = \langle 10, 4, 1 \rangle$ with a natural order comparator for each measure, C_2 is lower than C_1. The comparison is processed as follows: first measure equal $(10 = 10)$ so the second measure is compared $5 > 4$, so $C_2 < C_1$. Of course, the comparator depends of the application, each measure can have its comparator and threshold can be applied, preventing strictly comparisons.

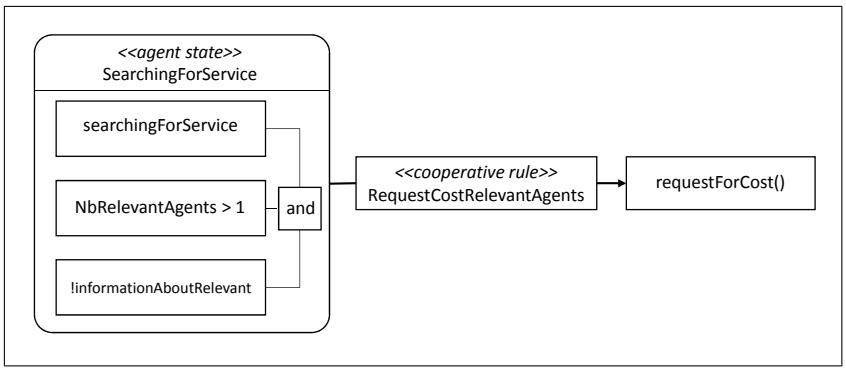


Figure 4.1 — NCS concurrence diagram rule

The cost allows to resolve **concurrence** Non-Cooperative Situations. Figure 4.1 presents the concurrence NCS rules diagram. This diagram highlights the conditions in which the NCS occurs, the cooperative rule and the action to perform to avoid the NCS. When an agent under the "Constrained Role" is searching for a service, detecting several agents that can help it while it has no information to favor a service among the others, the cooperative rule is to request the cost of all relevant agents by sending to each of them a *Request For Cost* message. At the reception of this messages, the addressed agent estimates the cost and sends it back. Using the received costs, the agent cooperatively selects the relevant service and sends it a *Request For Service* message. This cooperative choice is application dependent.

2.2 Enhancement of the Service Role

As presented in Chapter 4, the ADELFE methodology helps to design software with emergent functionality. ADELFE is composed of five Work Definitions (WD), each divided into activities. The WD3, the analysis phase, aims at identifying the system structure (entities and cooperative agents) and justifying AMAS adequacy at global and local level. This local adequacy is verified in the activity 13. In this step, the developer studies each cooperative agent and its behavior to determine if the agent must itself be implemented as an AMAS. Three criteria are proposed to confirm or not the need of decomposition:

1. **view:** the entity only access to a part of the data,
2. **granularity levels:** the entity is brought to have a lot of interactions and reasoning,
3. **behavior evolution:** the entity needs to adapt its behavior to the change.

Nevertheless, this decomposition was not addressed in AMAS4Opt, while it can be interesting to specify it for the "Service Role". Agents under this role are defined in order to

help the agents under the “Constrained Role” to satisfy their constraints. In certain cases, agents under the “Service Role” can have a complex behavior validating the three points addressed by the activity 13 of ADELFE.

Agent handling such a complex service must itself be decomposed into an AMAS. In this case, as it involves the definition of a new AMAS, this decomposition is not easy. We propose to remedy this situation using **service delegation**. Thus, a service agent can create new cooperative agent(s) and delegates services toward them. To identify them, we propose two criteria:

1. the possibility to divide the view of the agent into smaller ones,
2. the independence of the proposed services.

The new cooperative agents interact and negotiate together to answer the initial request. Thus, they can switch from “Service Role” to “Constrained Role”. If the initial agent under the “Service Role” perceives a request for which it already has created a delegate agent, it directly transfers the request to it. Moreover, as a cooperative agent, the initial agent keeps its autonomy: it may perform some treatments to the request before the transmission.

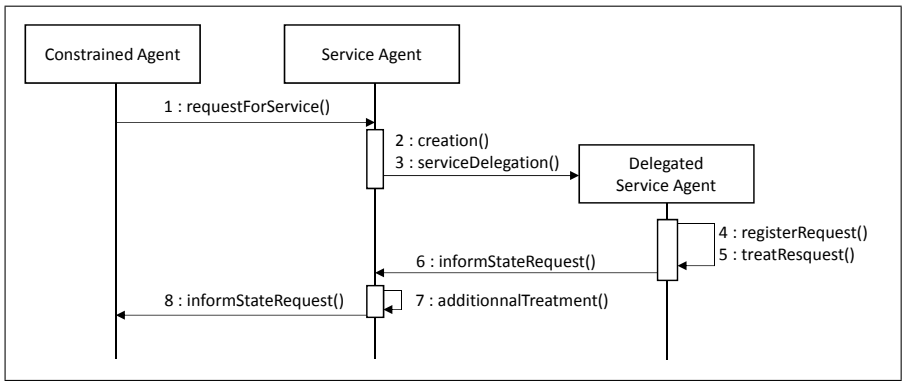


Figure 4.2 — Interaction diagram of service decomposition

Interaction diagram on figure 4.2 illustrates the service decomposition.

– At the reception of a *request for service* (action 1), Service Agent verifies if it has no delegated service agent to this service. Else, it creates one (2) and delegates the service (3),

– when it is created, the agent registers the request (4), treats it (5) and informs the Service Agent (6),

– the Service Agent can perform additional treatments (7) before informing the Agent Constrained (8).

2.3 AMAS4Opt Enhancements: Synthesis

We summarize the enhancements of the AMAS4Opt agent model in generic algorithms. The algorithm 4.1 presents the agent under the “Constrained Role”, and the use of the cost

to select the relevant service. Figure 4.3 illustrates the algorithm and the different Non-Cooperative Situations the agent can detect. First, the agent sends Request For Cost messages. Then, when it has enough information, the agent cooperatively selects the relevant service by sending a Request For Service message. If the service is accepted, the agent will search for an improvement to avoid partial uselessness Non-Cooperative Situation. Else, it will search for more information to find another service.

Algorithm 4.1: Agent under the Constrained Role

```
1  while not satisfied do
2      evaluate criticality;
3      if nb relevant service agent > 1 AND no information then
4          request for cost;
5      end
6      else if informations ok then
7          select lowest cost;
8          request for service;
9          if service accepted then
10             search for improvement;
11         end
12     end
13     else
14         request for information;
15     end
16 end
```

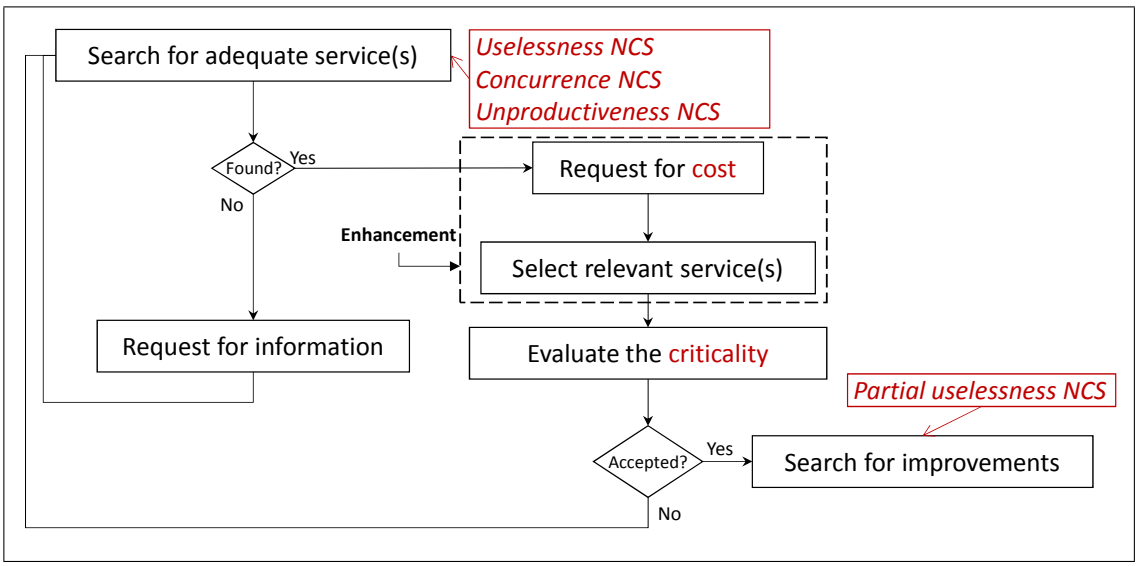


Figure 4.3 — Constraint Role behavior

The algorithm 4.2 presents the agent under the "Service Role" with treatment of **Requests For Cost** and the **delegation** implementation. Figure 4.4 illustrates the algorithm, the **service delegation** is highlighted. First, the received requests are analyzed: depending

on its knowledge, the agent can delegate those request. Then, the agent treats the requests depending on their type: Request For Cost, Information or Service. The different Non-Cooperative Situations and the states where agent is faced with them are indicated too.

Algorithm 4.2: Agent under the Service Role

```
get all requests;
while untreated request from constrained agent do
    if decomposition needed AND not delegate agent then
        create delegate agent;
        delegate service;
    end
    else if request for service then
        register request;
    end
    else if request for cost then
        compute cost;
    end
end
while untreated request from delegated agent do
    register request;
end
while registered request do
    select and treat the less critical;
end
```

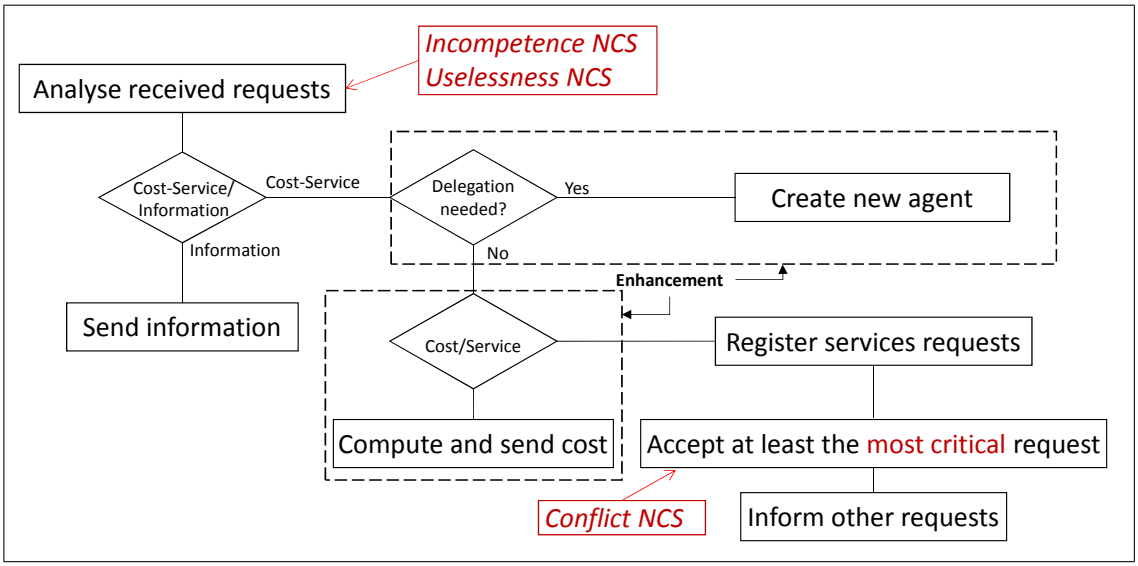


Figure 4.4 — Service Role behavior

2.4 Cost and Service Delegation: Illustration

In this section, the cost and the service delegation has been proposed as an enhancement of the AMAS4Opt agent model. The **cost** allows agents under the "Constrained Role" to cooperatively select the relevant service. **Service delegation** improves the behavior of a complex agent under the "Service Role".

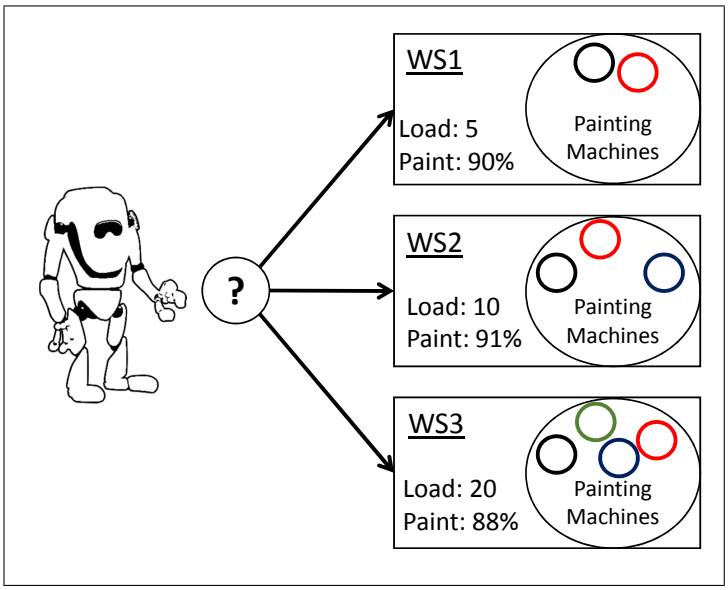


Figure 4.5 — Cost and service delegation illustration: robots painting factory

The following problem illustrates both of the proposed enhancements. We consider a robot R needing to be painted in red in a painting factory. This factory is composed of several workshops, each possessing a set of painting machines. Robots must select the relevant workshop, depending on the requested color. In our illustration, three workshops (WS) possess the red color, and so suits to R. Figure 4.5 shows the initial situation: WS1 can paint in black or red, WS2 in black, red or blue and WS3 in black, red, green or blue. The paint levels of the workshops are close (around 90%).

1. To select the more relevant, R requests the cost to the three workshops,
2. when it perceives the request, workshop estimates the difficulty to paint the robot, depending on its load and paint level. The computed cost is represented by: $\langle Load, Paint \rangle$, where $Load$ is the occupation rate of the workshop and $Paint$ is the level of available paints.
3. when all costs have been received, R perceives that all workshops have enough paint but one is less loaded than the others two. It selects it and request the painting, here, WS1 is selected.

The first three steps illustrate the cost mechanism to cooperatively select the relevant workshop. R is an agent under the "Constrained Role" who cannot select the relevant agent under the "Service Role": a workshop, and uses the cost to perform its selection.

Now that R has selected the relevant workshop, the painting could be planned. A workshop has an important complexity: several painting machines to manage and a lot of requests to treat. Thus, delegation mechanisms are applied. Painting machines receive requests from their workshop and negotiate together to paint robots.

4. when WS1 receives a request to paint, it delegates it to its painting machines,
5. painting machines negotiate and cooperate to paint R as fast as possible,
6. finally R is painted with red !

This simple illustration explains the bring of the cost and the delegation. The robot can select a workshop where it knows that it can be quickly painted with the correct color. Each workshop has a complex behavior, the **delegation** allows to simplify it. Thus, a workshop only computes the cost based on its knowledge, and delegates the painting to the correct machine that makes the service requested by the robot: to be painted.

3 A Cooperative Agent Pattern for Constraints Relaxation through the Neighborhood

In adaptive multi-agent systems, as presented in the Section 2 of Chapter 3, cooperative agents are defined as local entities. They possess local goals and interact locally. Moreover, there is no global entity that supervises the system: the adequate function emerges from the local interactions.

AMAS have proven their adequacy for complex problems solving. Still, an agent can satisfy its constraints but does not have the full competencies to do it alone. To overcome this situation, we propose a simple pattern where agents find help in their neighborhood by relaxing their constraints on their neighbors.

The proposing pattern is based on the agents local perceptions and cooperation. Instead of sends a request to all the agents of the system, an agent who cannot fully satisfy its constraint solicited agents with which it has acquaintances: its **neighbors**. Agent A_1 is a direct neighbor of A_2 if A_1 has an interaction link with A_2.

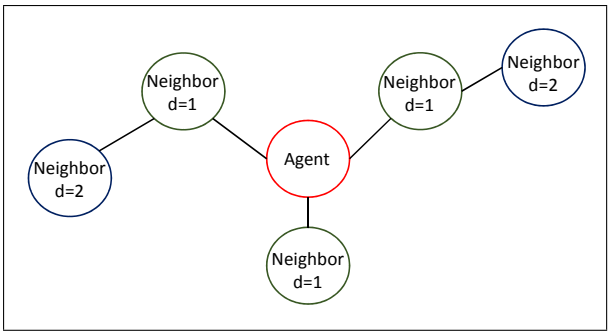


Figure 4.6 — The neighborhood

Figure 4.6 shows an agent, its 3 direct neighbors (with whom it has direct interactions) and 2 neighbors of its neighbors. When a neighbor agent perceives a request for help, three situations can occur:

– if it cant can fully help, it answers **OK** to the agent which has requested its service and helps it ($AG3$ and $AG4$ in Figure 4.6),

– else, if the help can be partly proposed, the agent **helps the best it can** and it also switches its role to Constrained and transfers the request to its neighbor. In its message, it subtracts the part of services it proposes, like $AG2$ in Figure 4.6. The neighbor performs the same reasoning,

– finally, if the solicited agent cannot help, it becomes Constrained and **transfers the whole request**.

To avoid infinite transmissions, a timer is decreased at each transmission. If an agent perceives the message with a null timer, it does not transfer the message, even if the help cannot be totally proposed.

Diagram 4.7 explains the pattern:

1. Agent 1 cannot reach its goal, it *requests help* to its neighbor, Agent 2 (action 1),
2. Agent 2 computes the help (2), it answers *OK* (3a) if it can help or transfers the request for help to its neighbor, Agent 3 (3b),
3. Agent 3 performs the same algorithm (4, 5a and 5b).

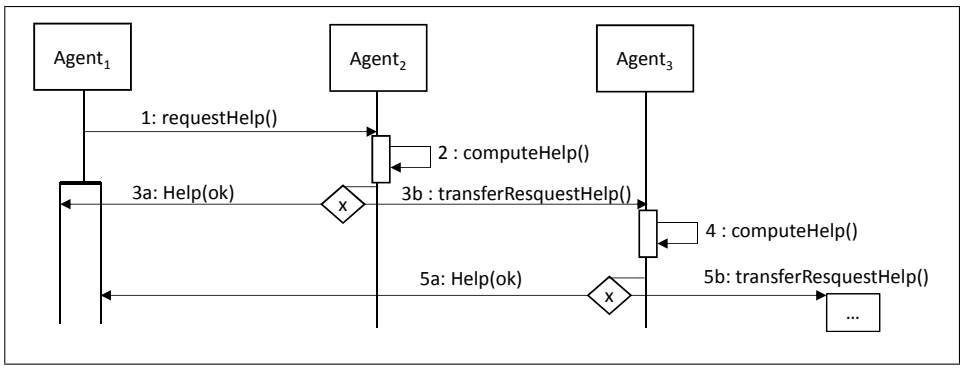


Figure 4.7 — Interaction diagram of the neighbor relaxation pattern

An **illustration of the constraint relaxation pattern** is proposed in Figure 4.8 where a fleet of robots is clearing out a building after a collapsing. There is no supervisor, each robot does its job and know which robots are near to it. In this example, the robot C perceives a huge debris but cannot remove it alone. It requests help for its two neighbors (B and D). B is too busy, it transfers the request to A, while D can help C but its battery is almost empty so it transfers the request to E, and starts helping C with its limited capacity. Finally, three robots are helping C, the debris can be removed. Figure 4.9 shows the final situation.

This example highlights the bring of local perceptions and partial knowledge: all robots can focus on their job, there is no need for a specialized one that would manage the fleet. Simple neighbors relaxation rules overcome the lack of supervision.

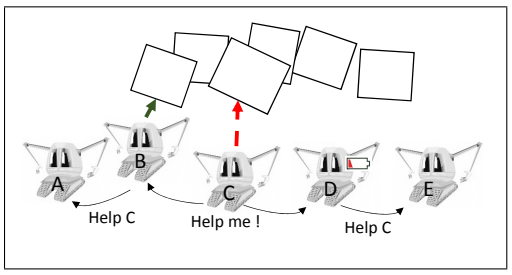


Figure 4.8 — Initial situation: robot C requests help

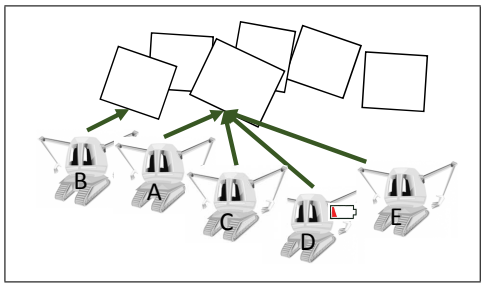


Figure 4.9 — Final Situation: A, B and E help C

4 Stopping a Self-Adaptive System: Rules and Proof of Coherence

In adaptive multi-agent systems, the agent organization emerges and constitutes the problem solution. This organization depends on interactions between the system and its environment. As the environment is dynamic and unpredictable, the organization evolves during the resolution: agents self-organization produces the adequate function. Moreover, as presented in Chapter 3, an adaptive multi-agent system is characterized by its lack of a global entity and control. Cooperative agents possess a partial knowledge of the system, and their behaviors are based on local interactions rules.

Those characterizations show differences with classical systems. Such systems are *open*, and not dedicated to be stopped, but always to adapt to events and always to propose a coherent solution. A solution is characterized as coherent if all treated constraints are respected. Indeed, the presence of a global function allows to qualify or not the proposed solution. On the contrary, in an adaptive multi-agent system, constraints are distributed among the agents. A solution is coherent if in the organization, all agents are in a coherent state *i.e.* if each agent respects its constraints. Those constraints depend on local interactions, to be coherent an agent must know that all agent interacting with it are in coherent states too. Moreover, the system continuously self-adapts to the dynamic of the environment, this self-adaptation is not immediate. When the system must propose a solution, for example when a human operator requests for it, it is impossible to be certain if the current organization is coherent, a certain delay must be respected to be sure of the organization coherence.

To respect the stop request, the delay between its reception and the real stop must be as short as possible. When the stop is requested, an agent a can be in coherent state ($C(a)$) or not ($\neg C(a)$). Thus, the solution of the system S is coherent if $\forall a \in S, C(a)$. An agent a is in a coherent state if all agents, A, in interaction with it knows its state, and if a knows the state of each agent of A. Formally, $C(a) = True \Leftrightarrow \forall ag \in A, K(a,ag) \wedge K(ag,a)$, where $K(a_1,a_2) = True$ if a_1 knows the state of a_2.

A cooperative agent can interact with two kinds of messages: ones that can modify the organization and ones that do not. The last kind does not impact the coherence, it can be ignored while the first must be treated and implies an answer.

To overcome the lack of global knowledge, we introduce a new **active entity** called **Manager**. The Manager is the interface between the system and the human operator. This entity knows all the agents and their state and it is automatically updated when an agent enters or leaves the system. When the operator asks for a stop, the Manager broadcasts the message to all agents, and waits for the answers, to indicate the real stop. The Manager does not affect the functioning of the system. Its role is only to be the interface between operator and the system: **it is not a cooperative agent as it has no goal**.

Using the Manager and knowing the kind of messages that can impact the coherence, behavior to perform in case of a request for stopping can be deduced. The interaction diagram in Figure 4.10 illustrates it:

1. When the Manager receives the stop request, it transfers it to all the agents and waits for all the "*OK Stop*" messages from them (action 1 and 2).
2. Those one can be in any step of their life-cycle (*perceive, decide or act*), they have to finish their current life-cycle n (3).
3. In the next life-cycle $(n+1)$, an agent can have two behaviors, depending if it has sent messages that can change the organization in cycle n (4):
   (a) if yes, the agent waits for the answers and treats them (5a and 6).
   (b) else (5b), the agent performs its stop behavior: only messages that change the organization are treated, the others are ignored,
4. To be sure that all messages are treated, the agents wait for a second life-cycle, $n+2$, (7).
5. After the life-cycle $n+2$, the agent is in a coherent state and so sends an "*OK Stop*" message to the Manager (8 and 9).
6. When all "*OK Stop*" messages have been received by the Manager, it must wait for a $2t$ delay before exposing the solution, where t is the maximal duration of a transmission between two agents (10). Indeed, a message can be delayed, the Manager must wait for this eventuality. If a message is perceived during the waiting (11), the Manager waits for another $2t$ delay (12).
7. Finally, the coherent solution is exposed, where all the agents are in a coherent state (13).

Even if the moment of real stop does not correspond to the moment of the stop request, the duration between those two instants is short. The duration of a life-cycle is brief: agents perform local interactions and they have a simple behavior. For example, on a standard laptop, the duration of a life-cycle is of the order of the millisecond. Moreover, the maximal duration of a transmission between two agents, t is also short: it depends on the middleware

but even for highly decentralized applications, the transmission of a message can be ignored. On a standard laptop, this duration is negligible.

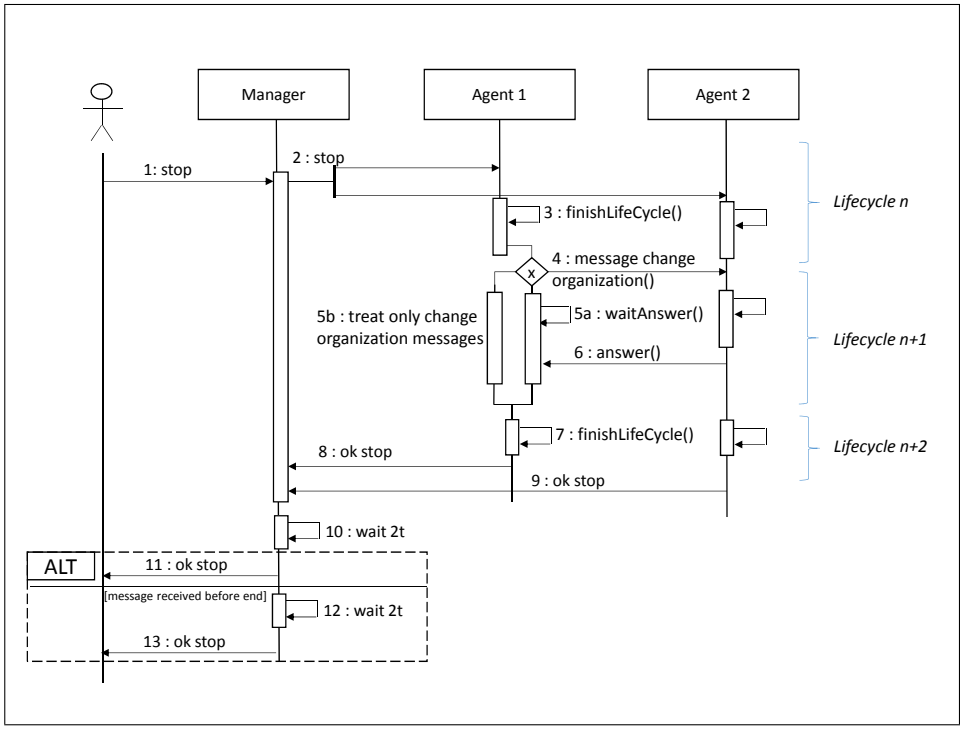


Figure 4.10 — Interaction diagram of the coherence proof

5 Conclusion and Analysis

In this chapter, two enhancements of the AMAS4Opt model, a pattern for cooperative agents and mechanisms to ensure a coherent solution when stopping an AMAS has been proposed.

– The cost brings cooperation in the selection of service by agents under the “Constrained Role”.

– A refining of agents under the “Service Role” has also been proposed. This new behavior allows agents to be able to deal with an important number of constraints. The enhancement of the AMAS4Opt allows to apply the AMAS theory to new categories of complex optimization problems.

– To optimize the resolution, we also bring a new pattern to relax constraints. Thus, cooperative agents rely on their direct neighbors to look for help, without exploring the whole system.

– Finally, a stop mechanism has been introduced. This mechanism allows to stop the system while proposing a coherent solution.

In the following chapter, we rely on those propositions to tackle a complex problem: mission planning for constellations of Earth observation satellites.

5 ATLAS: Adaptative saTellite pLanning for dynAmic earth obServation

« To improve is to change; to be perfect is to change often. »

Winston Churchill

Contents

1 Introduction

Mission planning for a constellation of Earth observation satellites is a complex problem raising significant technological challenges for tomorrow's space systems. The increasing numbers of customer requests, their introduction during runtime and their dynamic evolution during the planning result in a huge combinatorial search space. Today's systems are in front of several limits. As highlighted in Chapter 2, it is impossible to dynamically adapt the plan during its construction, and satellites are chronologically treated, minimizing the advantages of a constellation.

To sum up, to overcome those limits we need to build a system that:

- is able to manage a huge set of entities and their constraints, while providing a high quality plan for the whole constellation,
- can manage new requests (and or constraints) during its execution,
- provides a fast feedback on customer requests state.

To answer those questions, we rely on the *AMAS* theory and the enhanced version of the AMAS4Opt agent model to propose the ***A**daptative sa**T**ellite p**L**anning for dyn**A**mic earth ob**S**ervation* system (**ATLAS**). ATLAS produces a mission plan for a constellation of satellites, from a list of client's requests (those can be dynamically added into the system without interruption). Moreover, as ATLAS works continuously operators can have a quick feedback on the plan construction. The objective of this chapter is to present the ATLAS system. After justifying the adequation of the AMAS approach, we first describe the general composition and behavior of ATLAS before to detail each cooperative agent.

1.1 The Environment of ATLAS

As an Adaptive Multi-Agent System, ATLAS evolves in an environment, and interacts with and through it. This environment is composed of the cooperative agent, and all the entities implied in the planning. The description of its environment can be made using the characteristics defined by [Russell et al., 2003], thus environment is:

- **non-accessible**: only a partial information on the environment is available,
- **discrete**: the number of possible actions each agent can produce on the environment is clearly defined, and are instantiations of AMAS4Opt functions,
- **deterministic**: each action an agent can produce has a single effect on the environment: the current state of the environment determines the next one.
- **dynamic**: external events can produce effects on the environment, those changes cannot be predicted: new entities or constraints can be added while running. For example, new requests can be added at any-time.

Self-organizing software allows natural decentralization and distribution. Those properties are useful to tackle this problem. Thus, this characterization comforts us in the choice of the use of an adaptive multi-agent system. In the next sections, we focus on the cooperative agents of ATLAS, their objectives and interactions between them and the entities.

1.2 The Entities of ATLAS

1.2.1 Entities

The ADELFE methodology presented in Chapter 3 guides engineers during the design of AMAS. Among its five successive steps, called *Work Definition*, the third consists in analyzing the requirements and specification to identify the **entities** of the system. Two kinds are distinguished: **active** (they possess an autonomy and can change their state) and **passive** (they have no autonomy, there are resources). Following this activity, nine entities have been identified:

- six entities are **active**:
  1. requests want to be covered and interact with their meshes,
  2. meshes want to be planned, they interact with requests and satellites,
  3. satellites are in charge of satisfying meshes and building plans,
  4. weather prediction module is able to inform all entities about new weather update,
  5. solar ephemeris module is able to update all entities,
  6. ground stations can change their state and so refuse some interactions,
- three are **passive**, they are used by satellite Agents to perform computations during the planning decision:
  1. satellite memory module,
  2. energy management module,
  3. the trajectory calculus library is only a calculus module.

In the current version of ATLAS, solar ephemeris module and ground stations are not yet implemented.

1.2.2 Cooperative Agents

After the identification of the entities, the ADELFE methodology guides the characterization of the **cooperative agents**. Among the active entities, three possess a local goal and must interact with other entities to reach this goal. To achieve their behavior there are autonomous: they decide by them-self with witch entities or agents they negotiate and they can change their inner-state themselves, with no external control. Those three entities are: **satellite**, **request** and **mesh**: they are defined as **cooperative agents**. The use of the AMAS4Opt model allows to define their role. Request and mesh Agents must be

satisfied: they are agents under the “Constrained Role”. On the contrary, satellite Agents provide solutions: build mission plans by planning meshes, and so satisfy mesh Agent. Those agents are under the “Service Role”. Those agents will be detailed in Section 2.

The last step in the characterization of the agents is the study of their behavior to determine the adequacy of AMAS at the agent level. The three criteria to study, proposed by ADELFE, are:

1. **view:** the entity only has access to a part of the data,
2. **granularity levels:** the entity is brought to have a lot of interactions and reasoning,
3. **behavior evolution:** the entity needs to adapt its behavior to the change.

Request and mesh Agents have a simple perception (only their meshes for request Agents, their request and satellites for the mesh Agents) and granularity (they only negotiate with a small number of agents). They do not need to be decomposed.

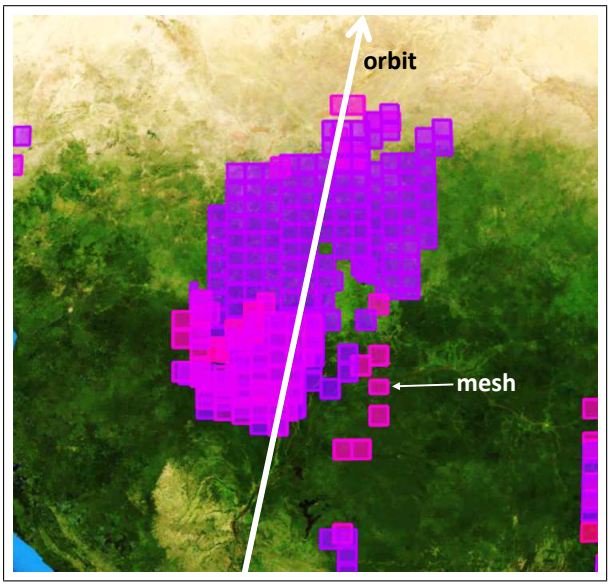


Figure 5.1 — An hot-spot

As presented in Chapter 1, in *hot-spots*, hundred meshes are in conflict in a little portion of the satellite orbits: there is many solutions to plan each of those meshes, this result in a high combinatorial search space that satellite Agents must handle.

Figure 5.1 shows this complexity: the 200 meshes of this small area can be acquired in a same visibility windows (around 5 minutes), and it is impossible to plan all the meshes (the average acquisition duration is 10 seconds). The ideal combination would be composed of only 25 acquisitions. Thus, there is $\binom{200}{25} = 4.52 \times 10^{31}$ combinations possible. Moreover, each acquisition can be started at any moment in the window. For example, if this window is discretized into steps of 10 milliseconds, there are 3 000 possible starts for a single acquisition. This makes the search space more combinatorial: it is impossible to let satellite Agents reorganize their plan in a reasonable time.

Given this example and regarding the three criteria proposed by ADELFE, the satellite Agent must be decomposed. In this thesis, we decompose the behavior of the satellite Agent into an **intern AMAS in charge of the satellite mission plan**.

1.2.3 Satellite Agent Decomposition

To identify the agents of the intern AMAS of satellite Agent, we focus on the description of the intern planning problem of a satellite. Inside the plan of a satellite, the **acquisitions** are in conflict to build the mission plan. The shift of an acquisition can make enough place to allow the satellite Agent to plan another mesh. This situation is illustrated by Figure 5.2: there is not free enough space to plan $Acquisition_1$, but if $Acquisition_2$ starts a few times later, it makes enough place for $Acquisition_1$.

This illustration allows to identify a new cooperative agent: the **acquisition Agent**. Those agents, located inside each satellite Agent, must negotiate together to produce the satellite mission plan. The analyze shows that those agents have both AMAS4Opt agent roles: they have to find a place to be planned, they have the “Constrained Role”, but they can negotiate together to free additional areas, they also have the “Service Role”. To ease the decomposition of satellite Agents, we rely on the service delegation proposed in Section 2 of Chapter 4.

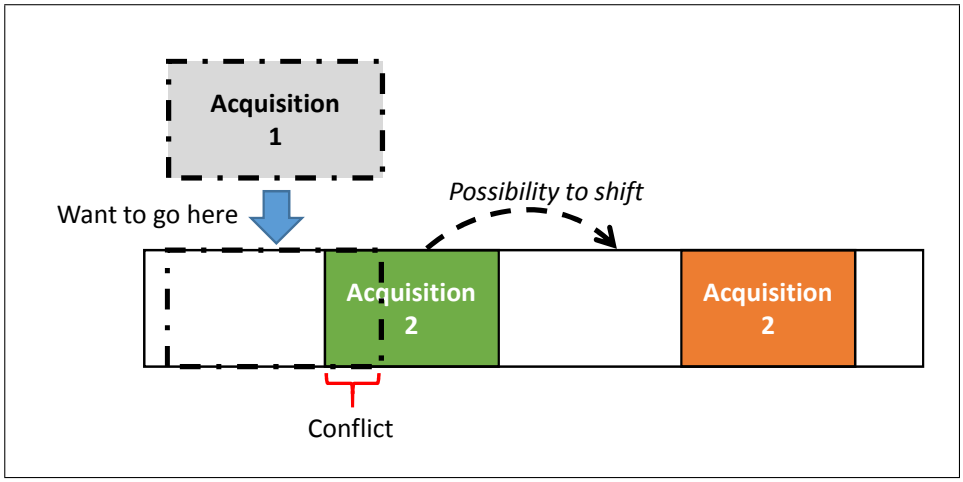


Figure 5.2 — Example of the shift behavior

1.2.4 Synthesis of the Two Levels of ATLAS

This analysis highlights two levels in the ATLAS system:

1. the first is in charge of the distribution of meshes toward satellites: **request, mesh and satellite Agents**,
2. the second concerns the building of the mission plan for each satellite: **satellite and acquisition Agents**.

In the following section, we present the global description of the ATLAS system. Then, each cooperative agent is detailed.

1.3 System Description: Interactions Diagrams

Before detailing the design of each cooperative agent, we present in this section the global functioning with two interaction diagrams. The different messages and computed values are detailed in the next sections. The first diagram (Figure 5.3) illustrates the behavior of the first ATLAS level, with request Agent, mesh Agent and satellite Agent. Figure 5.4 focuses on the second level: how acquisition Agents cooperative interact to organize the planning inside a satellite Agent.

The first diagram (Figure 5.3) highlights how the mesh Agents cooperatively organize their distribution between satellite Agent, but the building of the mission plan is not considered. In this example, the mesh Agent can be planned by two satellite Agents.

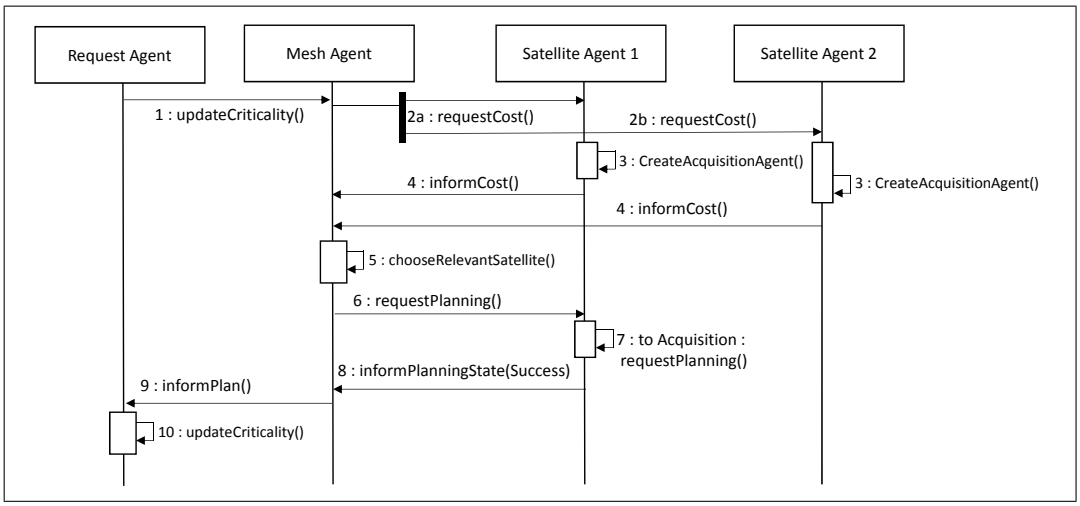


Figure 5.3 — Interaction diagram: first level of ATLAS, mesh Agent negotiates with two satellite Agents

– first (action 1), the request Agent informs the mesh Agent to update its criticality. This measure corresponds to the non-satisfaction degree of an agent. For a request Agent this factor is related to the number of mesh Agents not yet planned. When a mesh Agent is planned, the satisfaction degree of the request Agent increases and its criticality decreases, section 2.2 detailed the criticality of request Agents,

– when a not planned mesh Agent receives such a message, it increases its criticality and requests the planning cost to all satellites that can acquire it (2a and 2b). This cost represents the planning difficulty: more it is difficult to plan a mesh, more the cost is important. The cost and the measures that compose it are detailed in section 2.4,

– at the reception, each satellite Agent creates an acquisition Agent to compute the cost (3). The cost is then returns to the mesh Agent (4),

– as a cooperative agent, the mesh Agent selects the satellite Agent with the lowest cost (5) and requests it the planning (6),

– the satellite Agent plans the mesh Agent (7) and informs it (8),

– finally, the request Agent is informed of the planning (9) and update its criticality (10).

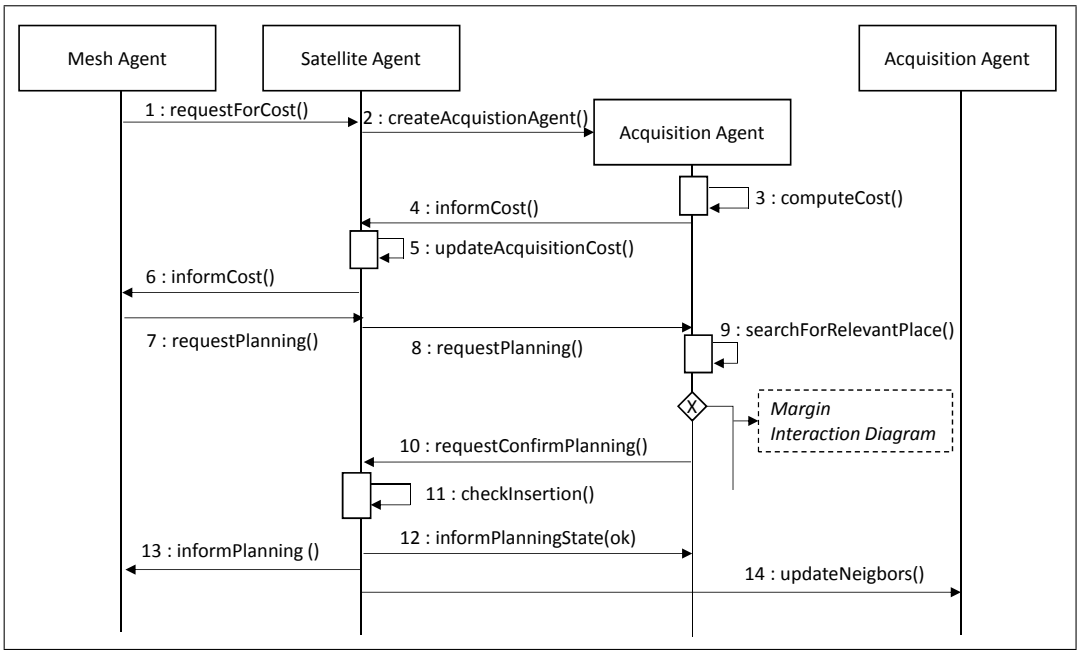


Figure 5.4 — Interaction diagram: second level of ATLAS, acquisition Agents planning

The second diagram (Figure 5.4) shows how acquisition Agents, inside satellite Agents, interact to build the mission plan.

– when a satellite Agent receives a *requestForCost* message from a mesh Agent, it creates an acquisition Agent (actions 1 and 2). This one locally computes its cost considering only its visibility accesses (3) and sends back to the satellite Agent its access with the lowest cost (4); this cost is completed by the satellite Agent before it is returned to the mesh Agent (5 and 6),

– when a mesh Agent has chosen a satellite Agent (7), this last asks for the planning to the corresponding acquisition Agent (8),

– now the acquisition Agent must place itself (9):

  1. first, the agent updates its cost and selects an access,
  2. in this access, the agent highlights potential intervals where it can be placed,
  3. if at least one interval exists and there is no conflict with its neighbors, the agent estimates the booking and requests the confirmation to the satellite Agent (10),
  4. else, the agent uses the margin mechanisms presented in section 2.5.3,

– the satellite Agent checks the planning (11) and confirms to the acquisition Agent and to the mesh Agent (12 and 13), moreover, it informs relevant acquisition Agents of their new neighbor (14).

Those two diagrams show the two levels of ATLAS. In the next section, the design of each cooperative agent is detailed.

II

2 Cooperative Agents: Request, Mesh, Satellite and Acquisition

In the first section, the agents involved in the planning have been identified. In this section, we firstly present the messages that agents can exchange. Then, each agent is detailed: its composition, its behavior and the different non-cooperative situations the agents must detect and repair.

2.1 Agent Interactions and Communications

In the ATLAS system, the communication relays on messages exchange. Table 5.1 lists the different messages agents can exchange in the first ATLAS level while Table 5.2 details message used by acquisition Agents. Following the AMAS4Opt model, messages can be: request for cost (the difficulty to perform a service), request for service, answer to request (information about the success or failure in the request treatment), simple information message or request for information (for example a change in the neighborhood important for the knowledge of the agent). Request for information type of messages is not represented in the two following tables.

From	To	Message	request for cost	request for service	answer to request	information message
Request Agent	Mesh Agent	*updateCriticality*				✓
		informCancel				✓
Mesh Agent	Request Agent	*informPlan*				✓
		informCancel				✓
Mesh Agent	Satellite Agent	*informCancel*				✓
		requestCost	✓			
		requestPlan		✓		
Satellite Agent	Mesh Agent	*informCost*			✓	
		informPlanningState				✓
		informCancel				✓

Table 5.1 — Messages in the first ATLAS level

Table 5.1 details messages used by request, mesh and satellite Agents:

– Request Agent only sends information messages to mesh Agents. An *updateCriticality* message is sent if the criticality of the request Agent has changed, so its mesh Agents must update their criticality. In case of *informCancel*, the mesh Agent cancels its behavior, this can occur if the validation date of the request Agent is over,

– mesh Agents interact with request Agent and satellite Agent:
  – messages to request Agent are sent if the satellite Agent cancels the mesh (*informCancel*), or if the mesh is successfully plans (*informPlan*).
  – If the request Agent decides to cancel its mesh Agents, for example if it has estimated it is impossible to be planned in a reasonable time, the mesh Agents perceiving *informCancel* messages have to stop their behavior and send an *informCancel* message to their satellite Agents, to select the relevant agent Satellite, the mesh Agent requests the cost of planning (*requestCost*) and, when the satellite Agent is chosen, it requests the plan (*requestPlan*).

– Satellite Agent answers to requests for cost from mesh Agents (*informCost*) or planning requests (*informPlanningState*). If it cancels a mesh Agent it informs it (*informCancel*).

From	To	message	request for service	answer to request	information message
Satellite Agent	Acquisition Agent	*requestForPlanning*	✓		
		informCancel			✓
		informNeighbors			✓
		informPlanningCost		✓	
		informPlanningState			✓
Acquisition Agent	Satellite Agent	*informCost*		✓	
		requestConfirmPlanning	✓		
		informCancel			✓
Acquisition Agent	Acquisition Agent	*margin*	✓		
		computeMargin		✓	

Table 5.2 — Messages of the acquisition Agents

Table 5.2 details the messages exchanged in the second AMAS level, between satellite and acquisition Agents:

– If the Satellite Agents is selected by the mesh Agent, it requires the acquisition Agent to compute its planning with a *requestForPlanning* message. It can also transfers *informCancel* messages from mesh Agents to their acquisition Agents and *informPlanningCost* message to inform them of the mesh Agent selected cost. Satellite Agent also sends information messages: *informNeighbors* to inform the acquisition Agent of new neighbors and potential conflicts to take into account in its planning behavior, and *informPlanningState* to confirm or not the planning of an acquisition Agent,

– acquisition Agents request for service to satellite Agents: after the choice of a planning date, a *requestConfirmPlanning* message is sent to the satellite Agent. The satellite Agent is informed of the cost *informCost* message, and of the cancellation with *informCancel* message,

– acquisition Agents negotiate with each other with *computeMargin* messages. When perceiving this message, an acquisition Agent computes its available area around its planning date and answers with *margin* message.

In the following sections, we rely on those messages to detail the cooperative behavior of the agents.

2.2 Request Agent

A request is a customer demand corresponding to a geographic area, composed of a set of meshes. As a request can be submitted by customers at any time, request Agents are created at the submission to represent a customer's request. The request Agent knows the different constraints defined by customers, and its relative mesh Agents. During its reasoning, it negotiates with that last in order to be satisfied. Thus, its local objective is a constraint: have all its meshes planned, the request Agent has the "Constrained Role", and possesses a criticality, corresponding to its non-satisfaction degree. This criticality is composed of an ordered list of measures: the percentage of its mesh Agents planned (updated when the agent receives a message informing the planning or the canceling) and its end validation date. The request Agent is characterized as follows:

- **Local goal**: have all its meshes planned.
- **Type**: "Constrained Role"
- **Criticality**: $C_r = \langle NP, RD \rangle$.
  1. NP represents the percentage of its meshes that are Not Planned,
  2. RD is the number of Remaining Days in the validity range,
- **Negotiate with**: its mesh Agents.

2.2.1 Request Agent Nominal Behavior

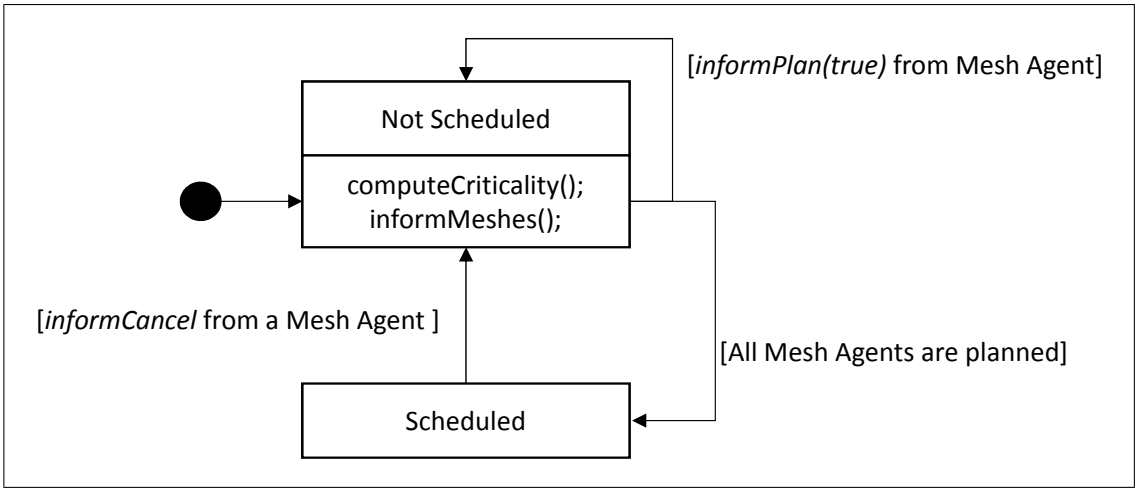


Figure 5.5 — Request Agent states

Figure 5.5 illustrates the nominal behavior of a request Agent, and its interactions with mesh Agents. A request Agent can be in two states: **Not Scheduled** and **Scheduled**.

In the **Not Scheduled** state, the request Agent tries to have all its meshes planned. For that, the agent negotiates with its mesh Agents to inform them about the planning progress. Its criticality C_r is increased in two cases:

1. one of its mesh Agent is canceled (perception of an *informCancel* message),
2. when the limit validity date is approaching.

If C_r changes, the request Agent informs its mesh Agents. When all its mesh Agents are planned, the request Agent becomes **Scheduled**.

In the **Scheduled** state, the request Agent is satisfied. Nevertheless, it can perceive an *informCancel* message from one of its mesh Agents and becomes **Not Scheduled** again. As the request Agent is cooperative and benevolent, it accepts to yield its scheduled to help another agent.

This agent is the interface with the operators. When a plan is requested, the mission plan is established using information of the request Agents. Nevertheless, the request Agents stay in ATLAS until operators remove them. As presented in Chapter 1, there are considered as done after the final picture validation: if the picture is not conformed with client's constraints, the request is re-planned.

During its nominal behavior, the request Agent can be in a abnormal situations: the non-cooperative situations. The agent is able to detect and anticipate those situations and to self-adapt to come back to its nominal behavior.

2.2.2 Non-Cooperative Situation

Request Agent can face one Non-Cooperative Situation (NCS): request Agents **Uselessness**. This can occur in the **Not Scheduled** state. More the end validation date is near, more a **Not Scheduled** request Agent is uselessness.

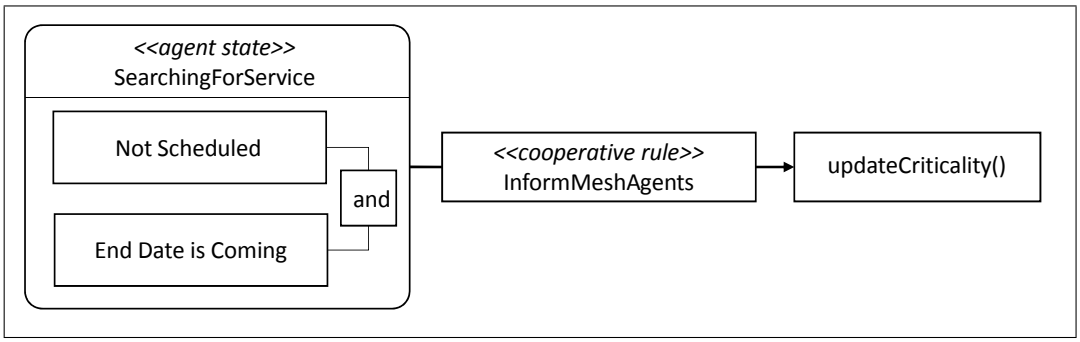


Figure 5.6 — Uselessness NCS: Update criticality when the end date is coming

To avoid this NCS, the agent informs the mesh Agents, using *updateCriticality* messages, to update their criticality. Thus, the mesh Agents increase their criticality and will have a higher priority when they request the planning. Figure 5.6 illustrates this NCS: the situation

in which the NCS can occur and the action to do. It is a cooperative rule that agents must follow until condition of the rule are satisfied.

Figure 5.7 presents in an *AMAS-ML* diagram the different modules (those different modules are described in section 3.1 of Chapter 3) of a request Agent:

- The first modules correspond to the *knowledge* of the agent: skill, characteristic, representation and aptitude modules:
  - the *skill* of the request Agent is its ability to compute its criticality in function of its perceptions,
  - its *characteristics* correspond to the composition of the agent (its id, mesh Agent and accesses, its dates of validation and priority), they are defined by the customer and the operator when the agent Request is created.
  - the *representations* are the knowledge of the agent that evolves during the planning process: initially the plan is empty, all its meshes are not planned and the request Agent is in Not Scheduled state,
  - the *aptitude* module can be instantiated by specific application tools, useful for the treatment agent. It is empty for the request Agent,
- the *perception* module enables the agent to perceive its environment. It contains all messages that the agent can perceive,
- the *action* module allows the agent to act on its environment. It refers to the message the agent can send.

The diagram (Figure 5.7) highlights the agents the request Agent negotiates with: mesh Agents.

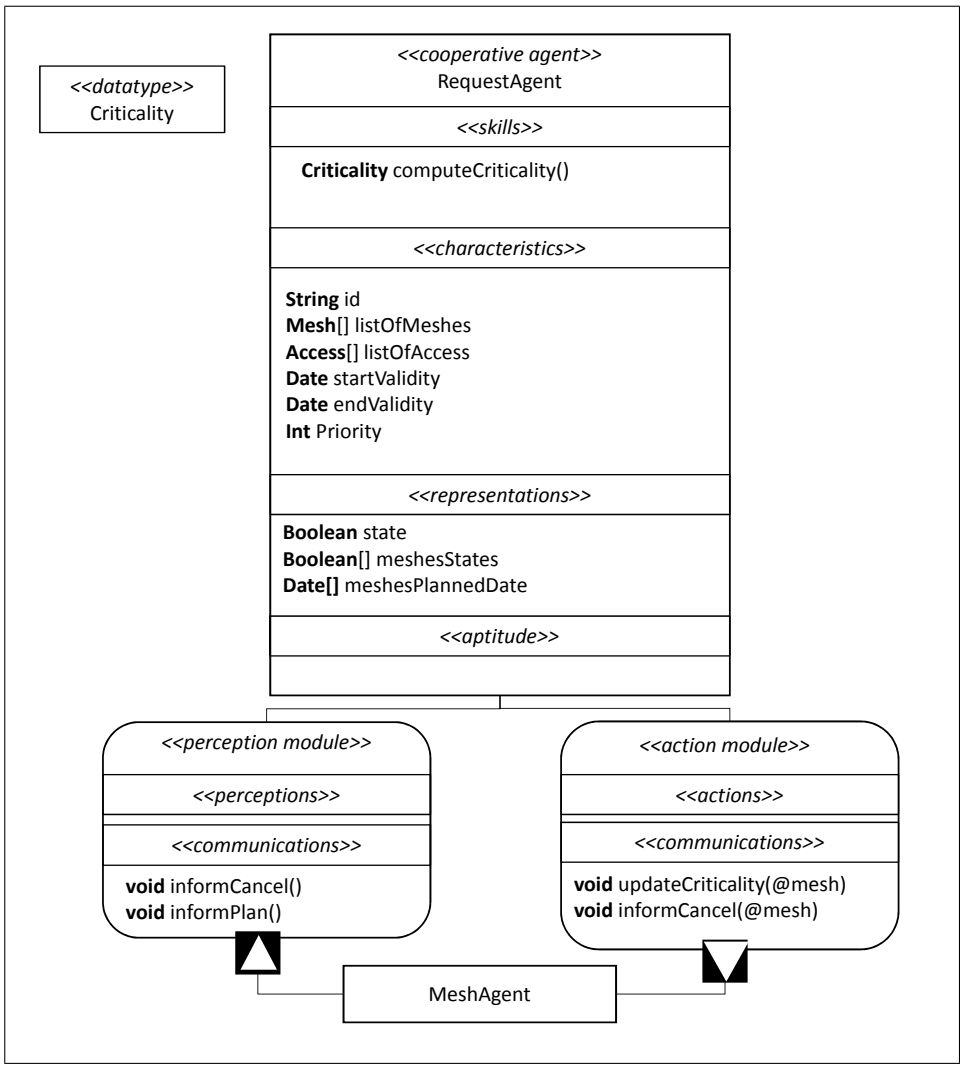


Figure 5.7 — Request Agent diagram AMAS-ML

2.3 Mesh Agent

A mesh is a part of a request, having the same constraints. It has to be planned by one of the satellites which possesses a visibility access towards it. The mesh Agents are created when their request Agent is created, and they know it. Moreover, the request Agent delegates to its mesh Agents its constraints (tolerated weather, acquisition type and angle constraints for example). Each mesh Agent must be planned respecting them. The local objective of this agent is a constraint, the mesh Agent is under the “Constrained Role”. To reach its goal, the mesh Agent negotiates with request and satellite Agents. The criticality of the mesh Agent is composed by the criticality of its request Agent, the client priority and the number of *informCancel* messages the agent has received from satellite Agents. The agent updates those parameters using its perceptions and knowledge: more its request Agent is critical or more the agent is canceled, more it is critical. The criticality is used by the mesh Agent to compare is criticality degree to other mesh Agents in case of conflict when it wants to be planned by a satellite Agent. The mesh Agent is characterized as follows:

– **Local goal**: be planned.

– **Type**: “Constrained Role”

– **Criticality**: $Cm = \langle Cr, P, R \rangle$

1. Cr is the request Criticality,
2. P the Priority,
3. R the number of Rejects,

– **Negotiate with**: its request Agent and satellite Agents that can acquire it.

2.3.1 Mesh Agent Nominal Behavior

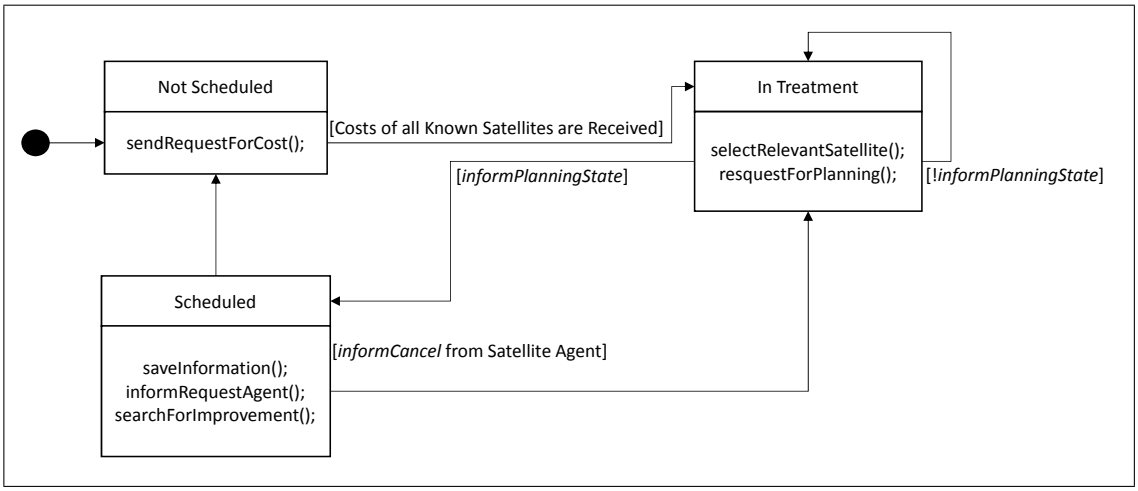


Figure 5.8 — Mesh Agent states

Figure 5.8 illustrates the nominal behavior of a mesh Agent, and its interactions with its request Agent and with the satellite Agents that can plan it. The nominal behavior is described as follows:

In the **Not Scheduled** state, a mesh Agent is looking for the relevant satellite Agent to request the planning. To select it, the mesh Agent uses the **cost**, defined in Chapter 4: a *requestCost* message is sent to all satellite Agents that can plan it. Satellite Agents estimate the difficulty to plan the mesh Agent and return the cost measure. When all the answers are received, mesh Agent becomes **In Treatment**.

In the **In Treatment** state, the mesh Agent selects the relevant satellite Agent in order to request the planning. To cooperatively select a satellite Agent the cost is used. Once the satellite Agent with the lowest cost is chosen, a *requestPlanning* message is sent. The mesh Agent waits for the answer. If the satellite Agent answers favorably (*informPlanningState*), the mesh Agent becomes **Scheduled**, else it stays **In Treatment** and selects another satellite Agent.

When a mesh Agent becomes **Scheduled**, it is satisfied. Information relative to the planning date is saved, and the request Agent is informed (*informPlan*). Thus, the mesh

Agent informs all its satellite Agents of its planning and the planning cost. If a satellite Agent perceives that its cost is lower than the planning cost, it informs the mesh Agent. When the mesh Agent perceives such an information, it updates its knowledge and informs its satellite Agent. At anytime, the request Agent can decide to cancel its mesh Agents and send them *informCancel* message. In this case, mesh Agent stops its behavior.

During its nominal behavior in the **Not Scheduled** and **In Treatment** states, the mesh Agent must detect and anticipate two non-cooperative situations: concurrence and conflict.

2.3.2 Non-Cooperative Situations

The two Non-Cooperative Situations mesh Agents must detect and anticipate to come back to its nominal behavior are **Satellite Agent Concurrence** and **Mesh Agent Conflict**. Figures 5.9 and Figure 5.10 illustrate the rules which express the NCSs.

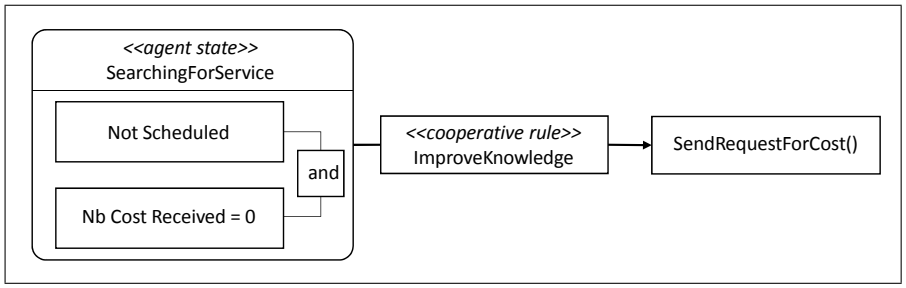


Figure 5.9 — Concurrence NCS: request the cost to update the knowledge

Satellite Agent Concurrence. In the **Not Scheduled** state, mesh Agents have potentially not enough knowledge to choose a relevant satellite Agent because it has no cost information. There is a **Concurrence** NCS choice between the satellite Agents (Figure 5.9). To overcome this lack of information, the mesh Agent must improve its knowledge: it sends *request for cost* messages. At the reception of the answers (the cost), the choice of the relevant agent can be performed: the mesh Agent compares the received costs, by comparing the measures composing the cost one by one. The satellite Agent cost is presented in Section 2.4.

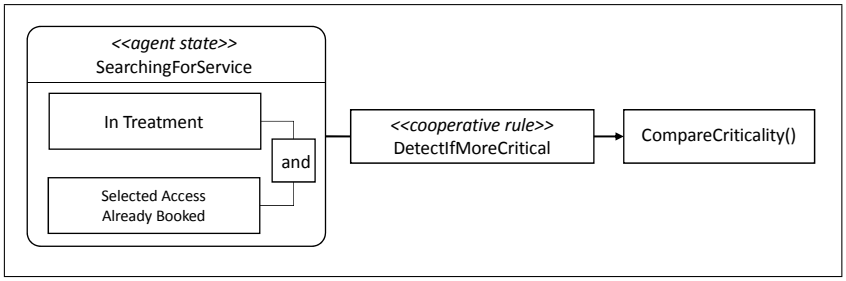


Figure 5.10 — Conflict NCS: the chosen access is already booked, compare the criticality

Conflict. In the **In Treatment** state, the mesh Agent can detect a potential future **Conflict** with another already planned mesh Agent (Figure 5.10). Indeed, the cost is only an estimation of the difficulty to plan a mesh Agent by the satellite Agent. If the satellite Agent

cannot plan the mesh Agent in the chosen access, it indicates this conflict in the cost. If the mesh Agent detects that the selected access is already booked, it must detect if it is more critical. Thus, the mesh Agent compares its **criticality** to the ones indicated in the cost, corresponding to the selected access, and decides to confirm to satellite Agent or not. To compare criticality, the agent compares the criteria one by one, from the request criticality (Cr) to the number of rejects (R). The comparison stops when a criteria is different.

Figure 5.11 presents the *AMAS-ML* diagram of the mesh Agent. The cost, representing the difficulty for a satellite Agent to plan a mesh Agent is presented in the next section.

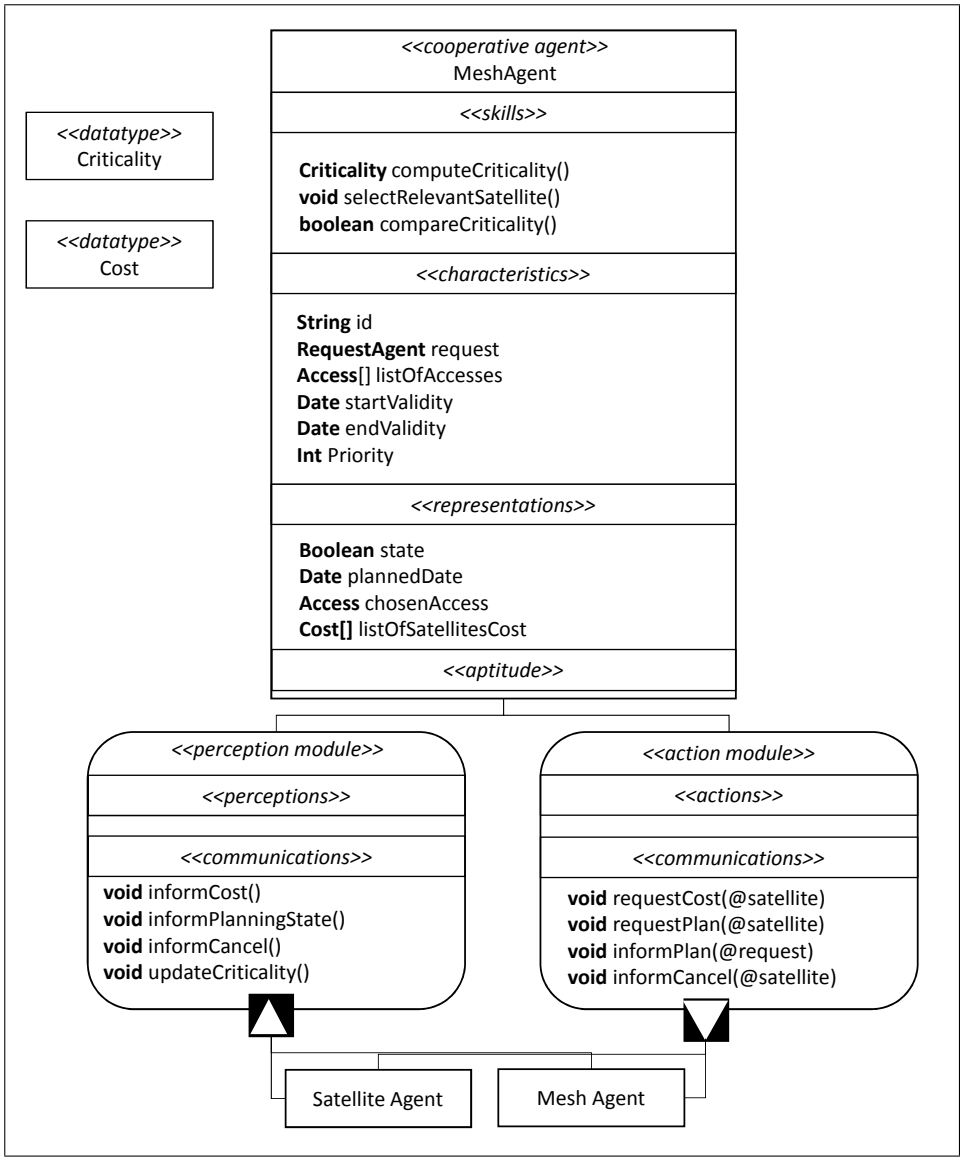


Figure 5.11 — Mesh Agent diagram AMAS-ML

2.4 Satellite Agent

The satellite Agent is an agent that helps the mesh Agents. Its local objective is not a constraint, satellite Agent has solutions (plan the mesh Agents), it is under the

"Service Role". As defined by the AMAS4Opt agent model, the satellite Agent does not have a criticality, but it computes a cost, representing its difficulty to plan a mesh. This indicator is an ordered list composed of several independent measures, computing by the satellite Agent using its knowledge:

1. the criticality of the potential mesh Agents in conflict with the mesh Agent which requests the planning,
2. the number of already mesh planned by the satellite Agent,
3. the occupation of the agent in the current life cycle,
4. the computed cost by the acquisition Agent.

When a mesh Agent compares costs to select the relevant satellite Agent, cost parameters are compared one by one, respecting the order of the list.

As presented in Section 1, the satellite Agent interacts with the two levels of AMAS in ATLAS. The first level corresponds to the treatment of mesh Agents requests and the distribution of meshes in the constellation. The second level corresponds to satellite mission planning: to plan the mesh Agents that have cooperatively selected it, the satellite Agent delegates its planning to an internal AMAS, composed of acquisition Agents (this agent is detailed in Section 2.5). Thus, it has mechanisms to create acquisition Agents and to transfer requests toward them. It also completes the cost estimated by its acquisition Agents. The satellite Agent is characterized as follows:

- **Local goal**: help mesh Agents to be planned
- **Type**: "Service Role"
- **Cost**: $Cost = \langle Cr_m, L, M, C_A \rangle$
  1. Cr_m is the criticality of the mesh Agent in conflict (if any),
  2. L the Load of the satellite,
  3. M the number of Messages received, in the current life cycle,
  4. C_A, the cost from the acquisition Agent (explained in Section 2.5).
- **Negotiate with**: mesh Agents and acquisition Agents.

Figure 5.12 shows the two AMAS levels of ATLAS and how the satellite Agents is positioned between them. Satellite Agent can receive three types of request from mesh Agents:

1. when perceiving *requestCost* message, the satellite Agent must estimate its difficulty to plan the mesh Agent. For that, it creates (if it does not exist) the acquisition Agent. This one computes the cost. The satellite Agent updates it by adding information using its knowledge, its plan load and the number of solicitations in the current life-cycle.

II

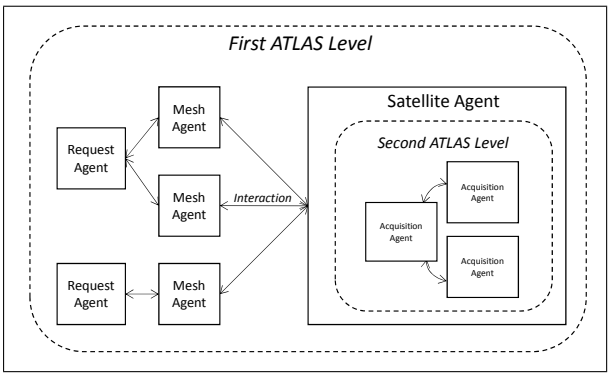


Figure 5.12 — The satellite Agent and the two AMAS levels

2. if it receives a *requestPlan* message, the satellite Agent delegates the planning to the acquisition Agent,

3. finally, if the message is an *informCancel*, the satellite Agent cancels the planning.

A satellite Agent can be face of one non-cooperative situation during its behavior: a conflict non-cooperative situation.

2.4.1 Non-Cooperative Situation

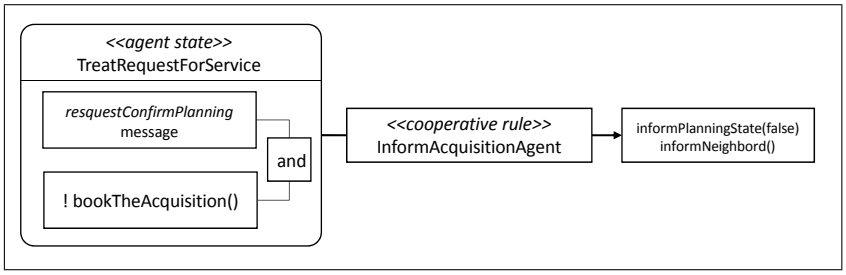


Figure 5.13 — Conflict NCS: the require slot is already booked, indicate the failure

Figure 5.13 illustrates the **Conflict** non-cooperative situation a satellite Agent can perceive. This conflict is detected by a *requestConfirmPlanning* message from an acquisition Agent that request a planning but the satellite Agent cannot book the acquisition Agent because the require slot is already booked. The satellite Agent informs the acquisition Agent of the failure and the acquisition Agents in conflict with it (*informPlanningState* and *informNeighbors* messages). Using this information, the acquisition Agent updates its knowledge by adding in its neighborhood the acquisition Agents in conflict, and then finds another slot.

Figure 5.14 presents the *AMAS-ML* diagram of a satellite Agent. The behaviors relative to the internal planning are highlighted. The satellite Agent has aptitudes: it relies on external libraries to compute the guidances (the duration of an acquisition) and the maneuvers.

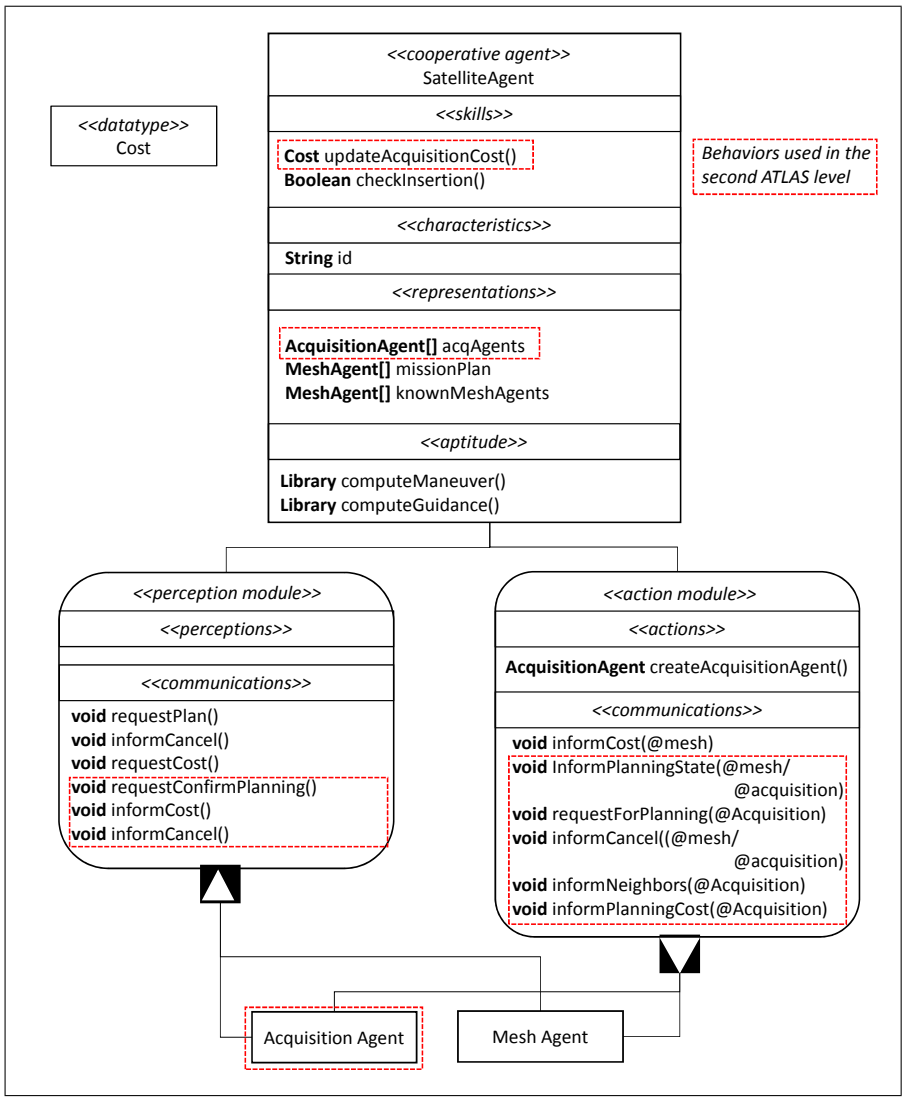


Figure 5.14 — Satellite Agent diagram AMAS-ML

2.5 Acquisition Agent

An acquisition Agent represents all the opportunities of a mesh Agent to be planned by one satellite Agent. For example, if a mesh Agent can be planned in four different visibility accesses, two on satellite Agent a, and two on satellite Agent b, two acquisition Agents will be associated with this mesh Agent: one by satellite Agent a, the other by satellite Agent b.

The acquisition Agents are created at runtime by satellite Agents when they perceive a *requestCost* message from the mesh Agents. The mesh Agents never interact with them. The set of acquisition Agents possessed by a satellite Agent represents an internal multi-agent system that aims to manage the plan of this satellite.

To be satisfied, an acquisition Agent has to find a place inside one of the accesses of the satellite Agent mission plan. From this knowledge, the acquisition Agent estimates its difficulty to find a place: the **cost**, and transmits it to its satellite Agent. If the mesh Agent selects this satellite Agent, this one transfers a *requestForPlanning* message to the

corresponding acquisition Agent, which will try to be planned in one of the visibility accesses. In this behavior, the acquisition Agent is under the “Service Role” as it helps mesh Agents to be planned.

Acquisition Agent must be planned inside visibility accesses. Still those accesses are common to several acquisition Agents, when the acquisition Agent has to plan itself, there are potential conflicts, and it has to take into account those conflicts to find a place. The set of acquisition Agents in potential conflict are called the neighborhood. The acquisition Agent is informed of such agents by its satellite Agent, when this one plans acquisitions. Figure 5.15 illustrates the neighborhood: AA_1 is a **neighbor** of AA_2 because AA_1 is planned in an access that intersects one access of AA_2. If the acquisition Agent cannot find enough free space to be planned, it negotiates with its neighbors to know if they can move and so create free space. During this resolution, the acquisition Agent acts as agent under the “Constrained Role”. Indeed, in order to satisfy the mesh request for planning, it requires the help of its neighbors to free more space. This behavior is detailed in section 2.5.3.

During its reasoning, the acquisition Agent switches between both the roles according to its perceptions.

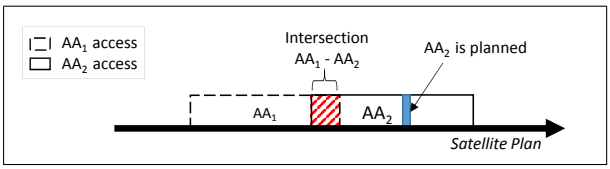


Figure 5.15 — Illustration of the neighborhood: AA_2 is a neighbor of AA_1: it is planned in an access with an intersection with a AA_1 ones

As it switches between both the roles, an acquisition Agent possesses a **criticality** and a **cost**. Its criticality is composed of the priority of the mesh Agent and the amount of time the agent is rejected. This last measure is automatically updated when the agent perceives a *cancel* message. The cost is computed using the neighborhood. Thus, we have identified criteria implying an important (or not) difficulty to be planned in the chosen accesses:

- the presence of other acquisition Agents,
- for each priority, the number of neighbor acquisition Agents
- and the distribution of free areas in its accesses.

Those criteria impact the difficulty of planning: more there are neighbor agents, more there are conflicts and conversely, more there are large free areas, more it is easy for the agent to be planned. The last two criteria (access duration and the occupation of the middle of the access) help to differentiate two or more accesses. The acquisition Agent is defined as follows:

- **Local goal**: find the best plan position
- **Type**: “Constrained Role” and “Service Role”
- **Criticality**: $Ca = \langle P, R \rangle$

1. P the priority of the represented mesh Agent,
2. R the number of cumulated rejects by the acquisition Agent,

– **Cost**: $Cost = \langle P, L_P, L_N, D, M \rangle$

1. P, a boolean, $P = True$ if the agent can be Planned without conflict,
2. $L_P = [P_{prMAX}, ..., P_{prMIN}]$, a sorted list, where each P represents the number of acquisitions of Priority pr in the neighborhood (MAX is the highest priority and MIN the lowest one),
3. $L_N = [H, M, S]$, a sorted list, where each element represents the Number of free areas in the access, decomposed according to the length of the area: Huge (H), Medium (M) and Small (S),
4. D the access Duration,
5. M a boolean representing if the Middle of the access is occupied or not.

– **Negotiate with**: its neighbors and the satellite Agent responsible for it.

2.5.1 Acquisition Agent Nominal Behavior

Figure 5.16 illustrates the behavior of an acquisition Agent, its interactions with its satellite Agent and its neighbors.

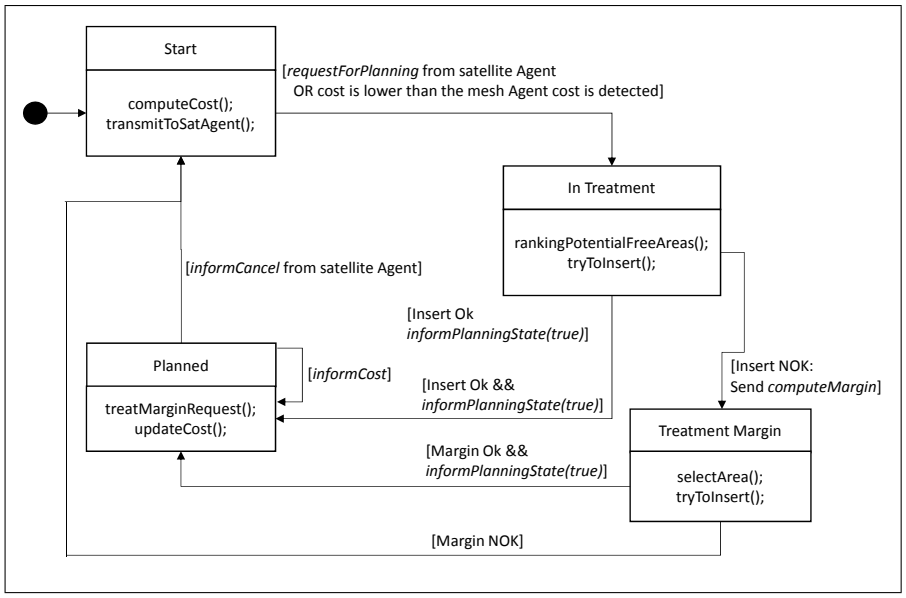


Figure 5.16 — Acquisition Agent states

When it is created by a satellite Agent, the acquisition Agent is in the **Start** state. In this initial state, the satellite Agent has delegated to the acquisition Agent the service of computing the cost. After computation, it is transmitted to the satellite Agent (*informCost* message). Then, the acquisition Agent expects a *requestForPlanning* message from the satellite Agent to become **In Treatment**.

The **Start** state is not a passive state. Indeed, during its wait, the acquisition Agent updates its cost according to the evolution of its neighborhood: new or canceled acquisition Agents impacting the number of neighbor agents and the number (and size) of free areas. Moreover, if the mesh Agent has selected another satellite with a lower cost, the acquisition Agent is informed of this cost. Thus, if the current cost becomes lower (for example more free areas or less high priority neighbors) that this perceived cost, the acquisition Agent can autonomously decide to be planned and goes to the **In Treatment** state.

In the **In Treatment** state, the acquisition Agent has to select a place to be planned and request confirmation to the satellite Agent. For that, in the access with the lowest cost, the acquisition Agent ranks all the free areas (*rankingPotentialFreeAreas()*). For all ones where the acquisition Agent can be planned in, it tries to be inserted (*tryToInsert()*), starting from the more interesting one. Three cases are possible:

- If an area is relevant, a *requestConfirmPlanning* message is sent to the satellite Agent to confirm the planning,
- Then, if the satellite Agent confirms (with *informPlanningState* message), the acquisition Agent goes to **Planned** state.
- Else if the satellite Agent answers negatively or if no area is relevant, the acquisition Agent sends a *computeMargin* messages to all its neighbors and goes to the **Treatment Margin** state.

In the **Treatment Margin** state, the acquisition Agent is waiting for *margin* messages to its neighbors. When all *margin* messages have been received, the acquisition Agent cooperatively selects the best place to be planned and sends a *requestConfirmPlanning* message to the satellite Agent. This behavior is precisely defined in the section 2.5.3. If no margin is available, or if the satellite Agent answers negatively, the acquisition Agent goes to **Start** state where it updates its cost and waits for a decrease. Else, the acquisition Agent goes to **Planned** state.

In the **Planned** state, the acquisition Agent continues to search for a better place, by updating its cost. If it decreases significantly, it informs the satellite Agent. The acquisition Agent treats *margin* messages too. Moreover, at anytime, it can be canceled by the satellite Agent and so goes back to **Start** state.

During its nominal behavior, the request Agent can be in three non-cooperative situations. The acquisition Agent is able to detect and anticipate them to come back to its nominal behavior.

2.5.2 Non-Cooperative Situations

The three non-cooperative situations the acquisition Agent can face during its different reasoning states are: **Concurrence** and **Conflict** in the **In Treatment** state, and partial **Uselessness** in the **Planned** and **Start** states. Figures 5.17, 5.18 and 5.19 illustrate those three situations and local rules agents follow to back to a nominal situation.

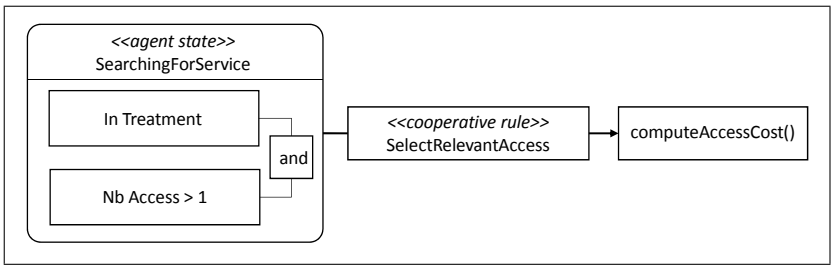


Figure 5.17 — Concurrence NCS: several accesses available, compute the access cost

Concurrence. In the **In Treatment** state, Acquisition Agent has the choice between several accesses where it can be planned. To select the relevant access, an acquisition Agent must solve this **Concurrence** NCS (Figure 5.17). As for a mesh Agent, an acquisition Agent uses the **cost** to select an access. For each access, the agent computes the cost (*i.e.* the difficulty to be planned inside this access). The cost its computing using its knowledge and depends on the:

1. access occupation (*i.e.* the number neighbors Acquisition Agent),
2. the criticality of the neighbors Acquisition Agent,
3. the ground distance between the agent and its neighbors (a short distance implies short transition maneuvers and so the possibilities to plan more acquisitions).

Thus, for each access a list of measures is computing by the acquisition Agent. Then, the lists are compared element by element, from the access occupation to the ground distance between the agent and its neighbors.

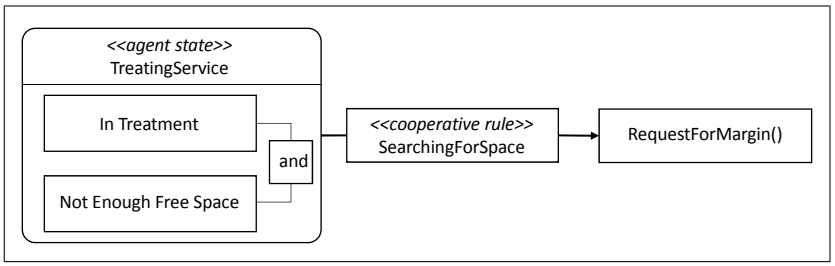


Figure 5.18 — Conflict NCS: cannot book, use the margin

Conflict. In the **In Treatment** state, even if the concurrence is solved, it is possible that the acquisition Agent cannot be planned because of **Conflict** with its neighbors (Figure 5.18): it has not enough free space. To solve this NCS, the agent searches for space using the margin mechanisms defined in the next section.

Uselessness. Even if it is in the **Planned** state, acquisition Agent is searching for improvement, to avoid partial **Uselessness** (Figure 5.19), corresponding to a suboptimal planning. To overcome it, the acquisition Agent updates its cost whenever neighbors evolve. Thus, if its new cost is lower than the cost when it was planned, the acquisition Agent tries to be planned in and sends a *requestConfirmPlanning* message to the satellite Agent.

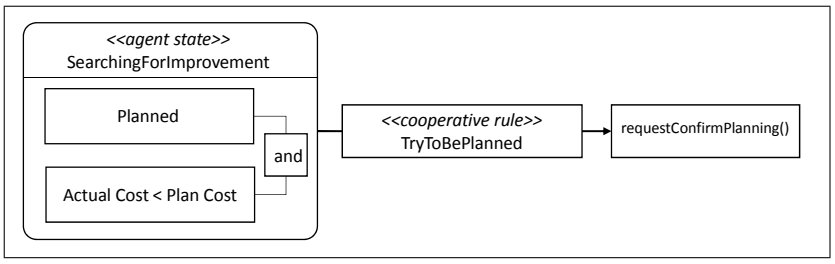


Figure 5.19 — Uselessness NCS: improvement is possible

Figure 5.20 represents the *AMAS-ML* diagram of an acquisition Agent. The acquisition Agent has the same aptitudes as its satellite Agent to compute its maneuvers and duration.

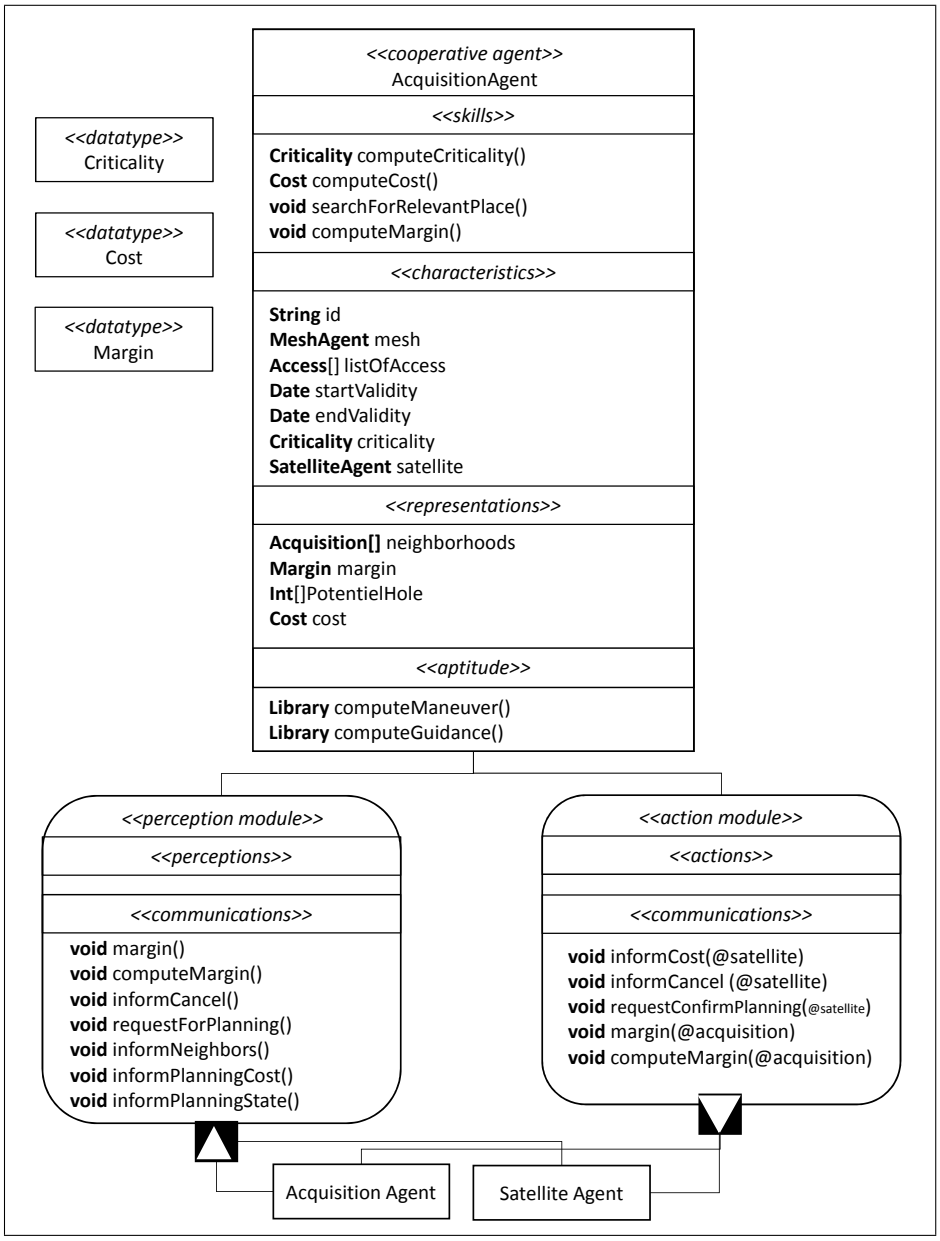


Figure 5.20 — Acquisition Agent diagram AMAS-ML

2.5.3 Negotiation between Acquisition Agents: the Margin Cooperation Mechanism

As previously highlighted, in *hot-spots* an important number of acquisitions are in conflict in a short period, resulting in a high combinatorial problem. During its reasoning, the acquisition Agent of such areas are in conflict with the acquisition Agents of their neighborhood. Those conflicts imply an important difficulty for each acquisition Agent to find a place. To overcome those situations, we implement the **Neighborhood Relaxation** discussed in Chapter 4. The idea is simple, and it is based on cooperation using the cost concept. The relaxation follows three steps:

1. for each potential area, the acquisition Agent request the potential shift of an acquisition: *margin*, of the already planned acquisition Agents by sending *computeMargin* message. In the message, the needed space is indicated,

2. when perceiving such a message, the acquisition Agent compares the need to its margin: the available margin can be greater (or equal) or smaller than the request. If the margin is enough, the acquisition Agent informs the emitter with a *margin* message. In this message the total available margin and the criticality of the agent are indicated. Else, it transfers the *computeMargin* message to its first neighbor.

3. finally, when the answers for all potential areas have been perceived, the acquisition Agent cooperatively selects an area. For that, the areas are ordered by criticality. The criticality of an area is an ordered list composed of the possible space, the number of acquisition Agents in conflict and the higher criticality of the agents in conflict. When the less critical area is chosen, the acquisition Agent requests the planning to satellite Agent. At the reception, the satellite Agent cancels the acquisition Agents in conflict.

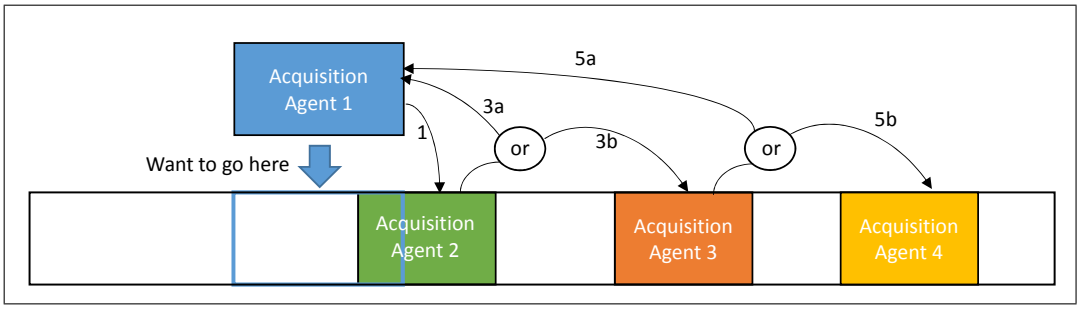


Figure 5.21 — Illustration of the relaxation

Figure 5.21 illustrates the functioning of the relaxation: $Acquisition\ Agent_1$ wants to go in the rectangle below it, but there is a conflict with $Acquisition\ Agent_2$. The negotiations are described in the interaction diagram, Figure 5.22:

– $Acquisition\ Agent_1$ sends a *computeMargin* message to the acquisition Agent in conflict, here $Acquisition\ Agent_2$ (action 1),

– at the reception, $Acquisition\ Agent_2$ computes its margin and:

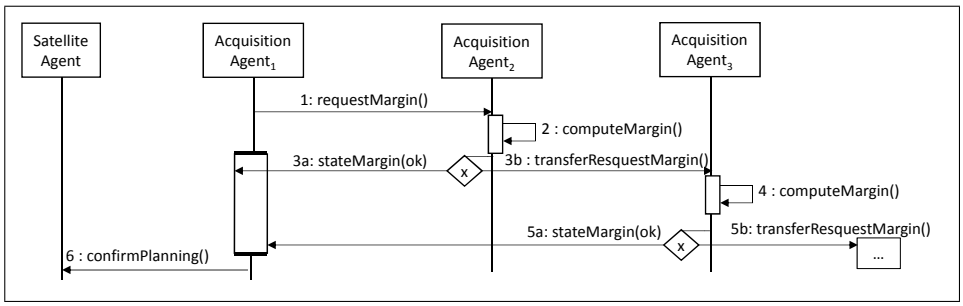


Figure 5.22 — Interaction diagram: the margin

  - if the acquisition Agent has enough margin, it answers OK with a *margin* message to $Acquisition\ Agent_1$ (3a),
  - else the request is transferred to the next neighbor, $Acquisition\ Agent_3$, (action 3b). The agent subtracts its margin to the request,

- finally, each agent performs the same algorithm (enough margin or not) until the necessary requested margin was founded or the request has been transferred to a certain amount of agent (typically a hundred), to avoid infinite transfers.

The **margin** brings more cooperation in the system. Acquisition Agents cannot find the best place in their first planning decision, they produce a plan with a lot a free areas. There is no rule to avoid those free areas. Using the margin, the acquisition Agents can produce free space. Thus more acquisition Agents can be planned, and the free areas in the plan are collapsed.

3 Illustration of the Behaviors

To sum up the behavior of the agents presented in this chapter, we propose a simple example, by focusing only on mesh and satellite Agents. We consider a constellation of two satellites $C = \{Sat1, Sat2\}$. The set of three meshes to acquire $M = \{M_T, M_M, M_A\}$. On the map, M_T corresponds to Toulouse, very urgent, M_M corresponds to Montauban, not urgent, and M_A corresponds to Albi, urgent. All meshes possess visibility accesses defined as: $a_{mesh/satellite} = [startdate; enddate]$. This scenario implies five agents in the system. We consider is this scenario that the maneuver and the acquisition duration are fixed. In real case, the system relies on external libraries. We suppose that computations of maneuvers and acquisition duration do not impact on the resolution. Figure 5.23 illustrates our scenario on a map. We can see the satellite trajectories, the three meshes and their access.

- **Satellite Agents**

  1. **Satellite Agent 1:** SA_1
     - Satellite: Sat_1
     - Can plan: MA_T and MA_M
     - Maneuver duration between two meshes: 3

2. **Satellite Agent 2:** SA_2
   - Satellite: Sat_2
   - Can plan: MA_M and MA_A
   - Maneuver duration between two meshes: 3

– **Mesh Agents**

3. **Mesh Agent Toulouse:** MA_T
   - Geographic area: M_T
   - Visibility access: $a_{T/1} = [10;30]$
   - Priority: very urgent
   - Acquisition duration: 7
4. **Mesh Agent Montauban:** MA_M
   - Geographic area: M_M
   - Visibility accesses: $a_{M/1} = [3;33]$ and $a_{M/2} = [10;30]$
   - Priority: not urgent
   - Acquisition duration: 7
5. **Mesh Agent Albi:** MA_A
   - Geographic area:M_A
   - Visibility access: $a_{A/2} = [0;20]$
   - Priority: urgent
   - Acquisition duration: 7

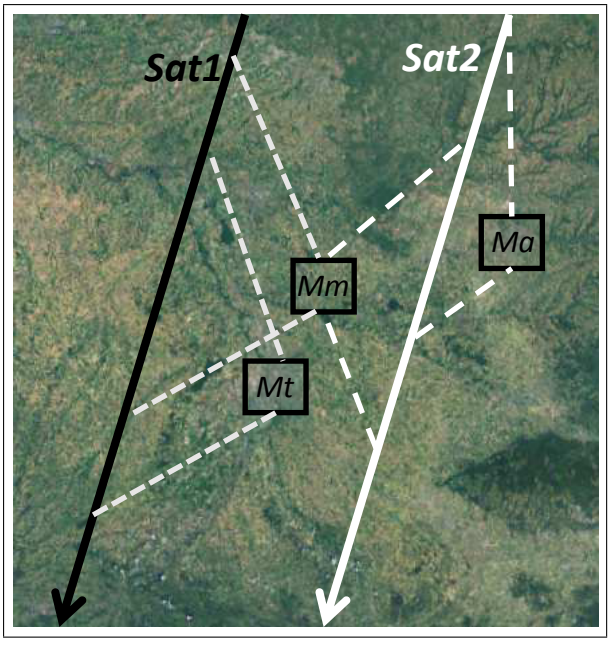


Figure 5.23 — Illustration of the behaviors: the three meshes are represented by black rectangles; the satellite trajectories by the two arrows; each access by dotted line

We consider that MA_M is already planned by AS_1 at $t = 13$ for 3 units of times. The criticality Cr is represented by the mesh priority, thus $Cr_{MA_T} > Cr_{MA_A}$, $Cr_{MA_T} > Cr_{MA_M}$ and $Cr_{MA_A} > Cr_{MA_M}$. The cost C is represented by the criticality of potential meshes in conflict Cr, the number of messages received M and the number of already planned meshes P: $C = \langle Cr, M, P \rangle$.

1. MA_T and MA_A have no cost information. Thus, they request the cost for their accesses to both satellite Agents: SA_1 for MA_T and SA_2 for MA_A.
2. In this step, satellite Agents create acquisition Agents. Those last estimate the cost. On SA_2, one has been received, the cost returned to MA_A is $\langle null, 1, 0 \rangle$. On SA_2 only one request has been received, but the acquisition Agent perceives a conflict with the one corresponding to MA_M. The returned cost to MA_T is $\langle Cr_{MA_M}, 1, 1 \rangle$.
3. MA_T and MA_A have received the cost, they can cooperatively select a satellite. MA_T perceives the conflict with MA_M, it compares the cost and perceives it is more critical: it confirms to SA_1. MA_A detects no conflict, it confirms to SA_2.
4. Both the satellite Agents inform their acquisition Agents to plan: from $t = 13$ to $t = 20$ for MA_T, and from $t = 0$ to $t = 7$ for MA_A. SA_1 cancels the acquisition Agent corresponding to MA_M and sends a cancel message to MA_M.
5. MA_T and MA_M are planned. MA_M receives the cancel information. It remembers that SA_2 can plan it. Thus, it requests planning to it.
6. SA_2 can plan MA_M form $t = 20$ to $t = 27$, after a maneuver from form $t = 17$ to $t = 20$.
7. The three meshes are planned.

This simple example shows the interest of the criticality and the cost. In a few interactions, the new meshes are planned, and the canceled ones find new places.

4 Conclusion

In this chapter, we have presented the ATLAS system. ATLAS is an adaptive multi-agent system to dynamically plan constellations of Earth observation satellites. To design ATLAS and the four different types of cooperative agents, we rely on the AMAS4Opt model and the proposed enhancements the cost, the service delegation, and the pattern to delegate service to neighbors (presented in Chapter 4). In the ATLAS system, cooperation between agents is guided by criticality and cost. The cost allows the mesh Agents to cooperatively select the relevant satellite Agent and the acquisition Agents to select the best access to be planned in. The criticality guarantees the respect of the different constraints (customer priority, validation date, *etc.*). Criticality and cost are the engines of the self-adaptation.

The design of ATLAS gives it major advantages toward today's planning system:

– On the contrary as today's planning systems, ATLAS can dynamically take into account new requests. Indeed, in ATLAS, the cooperative agents are able to self-adapt to

perturbations. Thus, when the operator submits a request from a customer, a new request Agent and its mesh Agents are created. Those new cooperative agents will interact with the other existing agents to find a place to be planned. If this new request is an emergency, its criticality will be very high and the request will be quickly planned: the already planned agents will let their place because of the cooperation.

- As ATLAS runs continuously, it provides a quick feedback on the planning progress of the customer request. Indeed, it is directly taken into account by ATLAS, so the operator has a feedback on its progress.
- In ATLAS, computations are distributed. The satellite Agents delegate their planning to acquisition Agents. Those agents locally interact to find a place, so the system is more reactive.
- The architecture of the ATLAS system is generic: it can be used to other planning system. Moreover, this architecture has been patented.

In the next part, we present the evaluations of the ATLAS system.

Part III

Experimentations and Discussions

6 Experimentations and Discussions

« Stay hungry, stay foolish. »

Steve Jobs

Contents

1 Introduction

In the previous sections, we present our contributions to solve complex problems using the AMAS theory. We have enhanced and applied the AMAS4Opt model to solve a major problem that space industries are facing: **mission planning for constellations of Earth observation satellites**. The applicative contribution of this thesis, the **ATLAS** planning system, aims to solve it.

In Section 2, we present our scenarios generator. This generic data generator allows us to test ATLAS in situations unreachable with real scenarios but interesting to test as scalability or real-time management. We rely on those scenarios in Sections 3 and 4.

The Section 3 highlights the bring of the cooperation to design ATLAS.

In this chapter, experiments to test and validate ATLAS are detailed. Those experiments aims to answer today's limitation highlighted in Chapters 1 and 2. We evaluate ATLAS considering scalability, dynamism and reactivity issues. Those experiments are detailed in Section 4.

The ATLAS system is a generic system, it can be used with different satellite planning systems. In the context of this thesis, ATLAS has been tested on a real scenario from Spot and Pléiades satellites, and scenarios derived from this real scenario by domain experts, guaranteeing the respect of maneuver constraints. Section 5 presents results and comparisons to a standard planning algorithm, the Hierarchical Greedy Algorithm, used nowadays in Europe to plan missions for satellites.

2 Scenario Generator

In traditional software, case studies allow to define a way to use the system and to globally describe its functional requirements. During the design of adaptive multi-agent systems, case studies are essential to engineers to study a large variety of applications and so identify all non-cooperative situations. Thus, case studies must be generalized in synthetic environments. Such environments deliver automatically an important and various corpus of use cases to validate system behaviors.

ATLAS is a planning system. Case studies correspond to planning scenarios. A scenario is characterized by the number of satellites (the constellation size), the number of meshes to plan and the average number of accesses per mesh. To test and validate ATLAS, scenario must represent:

1. **scalability**: scenario with varying number of satellites in the constellation and scenario with varying number of requests to plan,
2. **dynamic**: scenario where requests are dynamically submitted,

To validate a system, an important number of scenarios is required. It is impossible to manually create this corpus, to overcome it, we implement and use a **generic generator of scenarios**.

Sat \ Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sat 1	1	1	1	2	2	3	3	4	4	5	5	6	6	6	6	7	7
Sat 2	91	91	91	91	92	92	92	93	93	94	94	95	95	95	96	96	96
Sat 3	180	180	180	181	181	182	182	183	183	183	183	184	184	185	185	185	186

Figure 6.1 — Complete mission plan for 3 satellites

The generator starts by constructing a **complete mission plan**, where each satellite is fully occupied by the acquisition of several meshes. In this first step, acquisitions are randomly generated to fill all the available time of each satellite. The generator ignores maneuvers between and during acquisitions. We consider that the acquisition duration given by the generator includes both the maneuvers and acquisition duration. This abstraction allows to make our scenario more generic and the problem remains the same: find the correct position of a slot inside a pool of accesses where the duration is longer than the slot one.

Figure 6.1 shows an example of the complete mission plan for a 3-satellite constellation and for 16 time steps. Acquisition slots are in yellow and green, the number representing their mesh id. For example, Satellite 1 acquires mesh 1 from $t = 0$ to $t = 2$, then mesh 2 from $t = 2$ to $t = 4$.

After the construction of the complete mission plan, the acquisitions are transformed into a potential visibility access for a mesh. Meshes are then completed with 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 of each mesh and so the complexity. This factor indicates the maximum number of satellites that can acquire the mesh. Thus, each mesh possesses several accesses and can be acquired by different satellites of the constellation. A begin date is also added to meshes, to make scenario dynamics. This start date can be random or fixed, bringing more scenario diversity.

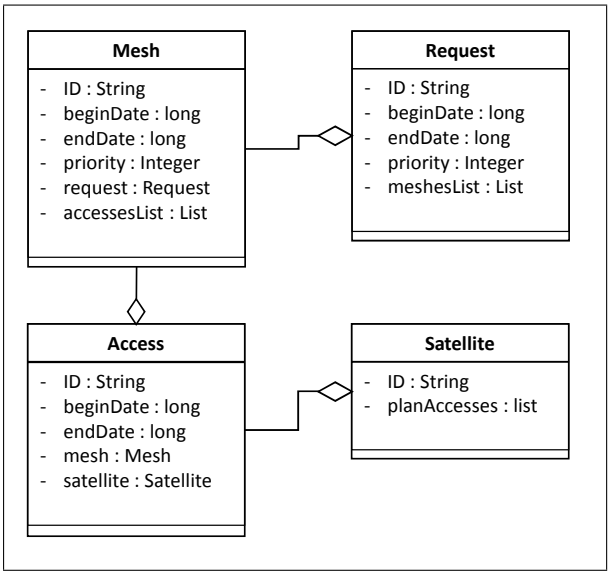


Figure 6.2 — Scenario class diagram

Finally, the generator assigns a priority to each mesh. This priority corresponds to the priority given by the customer and represents the first criterion of the mesh Agent criticality measurement. We decide to use three classes of priorities: routine (the lowest one), normal and urgent. The generator randomly associates a priority to each mesh respecting a distribution decided by users. For our experiments, we fixed the distribution as following: 50% of routine (the lowest priority), 35% of normal and 15% of urgent (the higher priority).

At the end of the generation, a list of meshes to acquire is delivered. Each mesh can be acquired in several accesses by one or several satellites. At least one optimal solution (regarding the number of planned meshes) exists: the complete plan. From this generator, we can produce an important number of scenarios, each different and each with different complexity, representative of the combinatorial of the problem. This generation process has been validated by experts of the space field.

Figure 6.2 presents the class diagram of a scenario. Based on the problem description presented on Chapter 2, requests are composed of a list of meshes. Those must be planned in one of their accesses by the relevant satellite.

2.1 Scenarios Details

In the following Sections (Sections 3 and 4), tests are made using scenarios produced by the generator. Table 6.1 presents all used scenarios. For each scenario, all meshes have 5 visibility accesses toward one or more satellites of the constellation (randomly chosen). Moreover, all satellites have the same number of meshes to plan. The distribution of priority is 50% of routine (the lowest priority), 35% of normal and 15% of urgent.

Scenario Number	Satellites Number	Meshes Number	Section
I	5	1 667	3
II	5	717	4.2
III	10	1 500	4.3.1 and 4.4
IV	20	3 000	4.3.1 and 4.4
V	30	5 500	4.3.1 and 4.4
VI	50	7 500	4.3.1 and 4.4
VII	100	15 000	4.3.1 and 4.4
VIII	1 000	150 000	4.3.1
IX	10	1 500	4.3.2
X	10	3 000	4.3.2
XI	10	6 000	4.3.2
XII	10	8 000	4.3.2
XIII	10	10 000	4.3.2

Table 6.1 — Detail of the scenarios

3 Value of Cooperation

The development of an adaptive multi-agent system is driven by the ADELFE methodology, presented in Chapter 3. During agents identification (activity 12), a step aims to identify cooperation failures interaction between entities. The list of cooperation failures allows to define Non-Cooperative Situations (**NCSs**), and to implement rules to repair and/or avoid them.

Nevertheless, it is complex to correctly list the whole cooperation failures at first. As they can be linked, a rule can produce a new cooperation failure. Thus, developing an AMAS relies on the refinement of its cooperation rules. To guide the development, we used the cooperation to refine the behavior of agents. At each step of the development, experiments are made to highlight one (or more) of the 7 NCS defined by the AMAS Theory, presented in Chapter 3. Once the NCS identified, a cooperation rule is proposed, implemented and the system is tested. The iterations are made while NCSs are identified.

In this section, **the bring of cooperation** is illustrated on ATLAS. The scenario used is the scenario number I of Table 6.1. The objective is to plan the maximum of meshes and for each priority. Moreover, the order between the priorities must be respected. All results highlighted in this section are an average after 10 runs, the deviation is below than 2%.

3.1 Initial Behavior

A first basic solution for the planning is that the mesh Agents randomly select a satellite Agent and send it a request to be planned. The satellite Agents try to plan the mesh Agents and inform about the success or failure. This first approach is characterized by:

- Agents: mesh and satellite,
- Rules: requests for plan, plan success/failure,
- NCS identified: **concurrence** between satellite Agents which can plan the mesh Agent.

Table 6.2 shows the initial results. Only 83.98% of the meshes are planned. The priority constraint is not respected: the more set planned is the routine one, and the distribution is uniform.

% Planned	% Urgent	% Normal	% Routine
83.98	83.09	82.01	85.59

Table 6.2 — Firsts results

3.2 First Iteration: the Cost

To overcome the first failure and to solve the identified concurrence NCS, it is necessary to add a rule for mesh Agents making them do the correct choice, in the best of their knowledge. Thus, we add the cost mechanism. Using it, mesh Agents can cooperatively select the relevant satellite Agent.

A NCS is identified in the satellite Agents behavior. When the satellite Agent perceives request for plan, it cannot favor a mesh Agent from another, because the criticality is never used. We identify here a conflict NCS. The system is characterized by:

- Agents: mesh and satellite,
- Rules: requests for cost,
- NCS identified: **conflict** in the satellite Agent behavior, when performing service.

Table 6.4 shows the results obtained with the implementation of the first rule. Results are improved, 86.80% of total planned, the cost allows to plan the same proportion of each priority. Nevertheless, high priority meshes are not fully satisfied.

% Planned	% Urgent	% Normal	% Routine
86.80	87.77	86.15	86.91

Table 6.3 — Results after the first iteration

3.3 Second Iteration: Cancel Using Criticality

To solve the conflict NCS previously identified, we implement a behavior using the criticality. Satellite Agents favor more critical mesh Agents. Moreover, in case of conflict, satellite Agents inform of the criticality of the already planned mesh Agents. Thus, mesh Agents can confirm their choice if they are more critical.

In this iteration, no NCSs are identified. Nevertheless, as presented in Chapter II, the behavior of the satellite Agent is highly complex. The agent must treat a high number of conflicts. It is necessary to implement delegation mechanisms.

- Agents: mesh and satellite,
- Rules: favor more critical agents, indicate in the cost the criticality of mesh Agent in conflict,

Table 6.4 shows the results. Results are well better, 93.22% of the meshes are planned, and the urgent meshes are almost all planned (98.56%).

% Planned	% Urgent	% Normal	% Routine
93.22	98.56	95.68	89.80

Table 6.4 — Results using second iteration

3.4 Third Iteration: Acquisition Agent

The service delegation pattern, presented in Chapter II, is applied. Satellite Agents create and delegate the planning to acquisition Agents. Thus, acquisition Agents can

negotiate together to optimize their planning, using the neighbor relaxation: the margin. The system is characterized as follows:

- Agents: mesh, satellite and acquisition,
- Rules: delegate service to acquisition Agents,
- NCS identified: partial **uselessness** of acquisition Agents.

Table 6.5 shows the results obtained. With the new type of agent and the margin rule optimize the mission plan. But we can see that some agents are not planned. We identify here partial uselessness NCS: when an agent Acquisition is planned, it never searches for an improvement.

% Planned	% Urgent	% Normal	% Routine
96.88	98.92	97.12	96.04

Table 6.5 — Results after the third iteration

3.5 Final Iteration: Search for Improvement

The last cooperation rule aims to solve the partial uselessness of acquisition Agents. These rules give behavior to acquisition Agents in order to always search for improvement. The cost is updated at each change in the neighborhood of the acquisition Agent. Thus, if it is lower than the planned cost, the acquisition Agent decides to change its planning and informs its mesh Agent. This change produces new free areas for not planned agents.

The final system is characterized as follows:

- Agents: mesh, satellite and acquisition,
- Rules: update the cost and try to be planned in access where the cost decrease.
- NCS identified: $\emptyset$

Table 6.6 shows the final results. More than 98% of the meshes are planned, and all the urgent meshes are planned.

% Planned	% Urgent	% Normal	% Routine
98.56	100	98.92	97.84

Table 6.6 — Final results

Figure 6.3 illustrates the results for the five experiments. Each rule improves the results. The most important gap can be seen between the second and the third rule, cancel using criticality. Using the criticality, the dynamic is locally managed, a mesh can be inserted without the system reconsiders the whole solution. Moreover, this rule allows to plan more urgent priorities than fewer priorities, and more normal priorities than routine priorities. The rules have allowed to increase the results of 15% on the total of planned meshes.

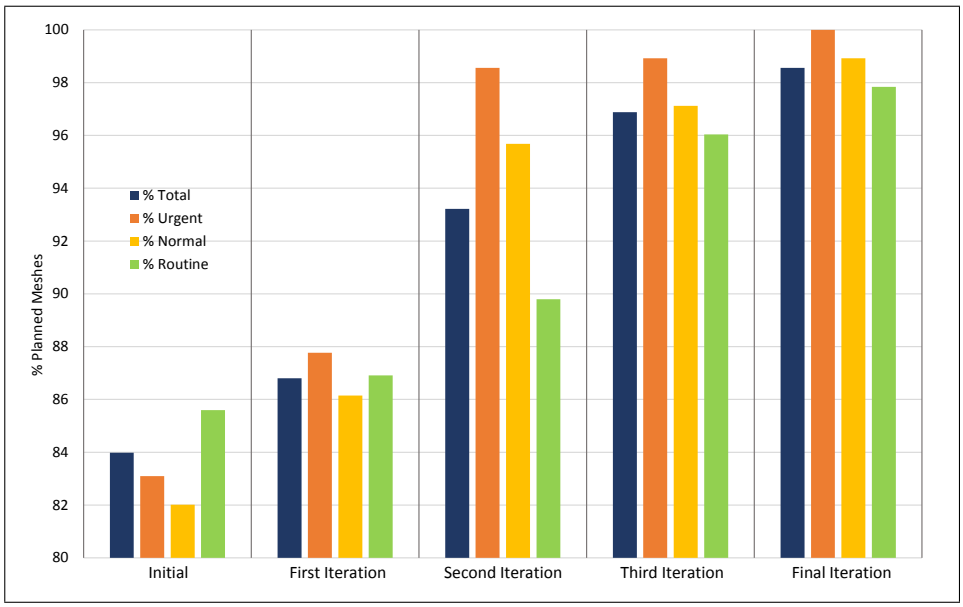


Figure 6.3 — Percentage of total planned meshes and per priority

3.6 Discussion on the Added Value of Cooperation

In this section, the **bring of cooperation** and its application is illustrated using the enhancement of ATLAS. At each step, the identification of the non-cooperative situation and rules to prevent it, allow the designer to enhance the system.

With ADELFE to identify the system entities (passive, active and cooperative agents), the *Cooperation Driven Development* allows to easily improve the AMAS to reach good solutions. This development methodology is a good tool to guide the design of agents behavior. The cooperation is not domain-dependent and is generic enough to complete the ADELFE methodology.

Finally, all the implemented rules remain at a local level. Thus, it is easier to develop than a global rule. Moreover, local cooperation allows a better management of the complexity and of the dynamic.

4 ATLAS: Generic Tests and Validation

After a presentation of our validation criteria, we detail in this section different experiments we have performed to test and validate ATLAS.

4.1 Validation Criteria

From the generator, several scenarios have been made in order to test and validate ATLAS on main categories of criteria defined in Chapter 1: **scalability, dynamism and reactivity**.

To qualify ATLAS on those three categories, we defined six criteria.

1. The dynamic management. The system must be able to manage requests whenever they are submitted.

2. The average delay before treatment. To perform near-real time planning, requests must be treated as fast as possible.

3. 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), $P = \frac{M' * 100}{M}$. Scenarios producing through the generator possess at least a solution with all the meshes planned (the complete plan), P allows to show the gap between the proposed solution and the complete plan.

4. For each type of priority, the percentage of planned meshes $P_{PRIORITY}$. Each request has a priority given by customers, this priority is a hard constraint. Comparison of the percentage of each type allow to validate that ATLAS respects this constraint.

5. The scalability. This quality solution indicator allows to test a system with a huge set of requests.

6. The load balancing indicates the occupation percentage of each satellite, in order to see if the constellation is entirely used.

In the following sections, we propose several experiments to evaluate ATLAS regarding those criteria:

1. dynamic management and average delay before treatment criteria are studied in section 4.2,

2. scalability, percentage of total planned meshes and per priority are considered in section 4.3,

3. finally, section 4.4 studies load balancing criteria.

4.2 Dynamic Management

4.2.1 Dynamic Submission

To assess the ATLAS abilities to handle dynamic, we rely on scenario number II of Table 6.1. In order to underline self-adaptive mechanisms, the arrival of meshes into the ATLAS system is controlled. All routine priorities are available at the start of the execution, the normal ones are inserted when ATLAS stabilizes itself, and finally, when ATLAS has handled both normal and routine meshes, urgent ones are inserted.

Figure 6.4 exposes the results. The 3 curves illustrate the percentage of all kinds of priorities during the execution. A, B and C correspond to the insertion of routine, normal and urgent polls of requests. 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 reorganize their plan to accommodate meshes with a higher priority.

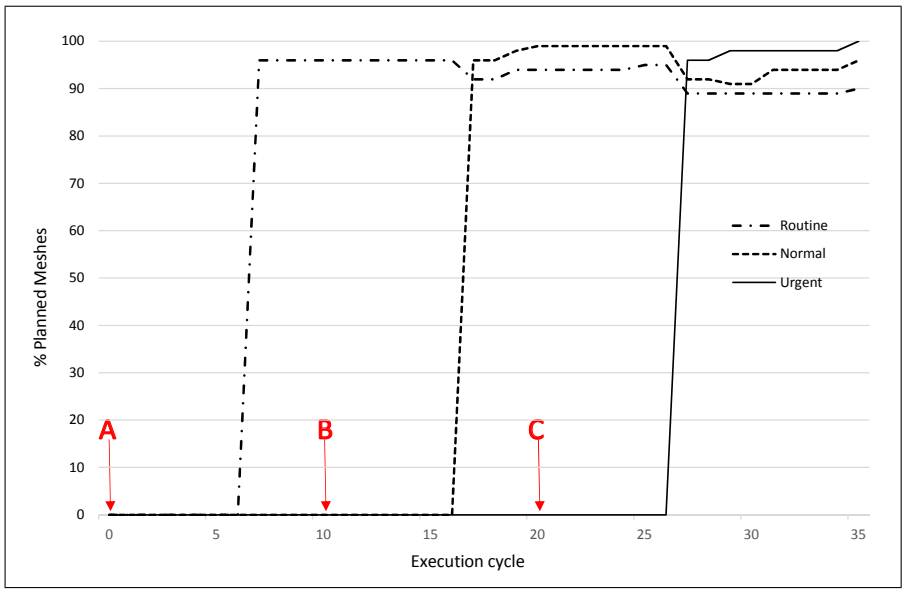


Figure 6.4 — Dynamic management (% for each priority)

Finally, 96% of the whole requests are planned (100% of urgent, 98% of normal and 94% of routine).

Interesting point shown by this experiment is that even if a mesh is canceled in order to free some place to a more urgent one, the canceled one could find another space in the planning. This is visible with the normal and routine polls: the percentage of planning meshes for those polls decreases before increasing again when a new poll is inserted. This is explained by the fact that low priority agents let their place in favor of more priority ones before finding new places. Thanks to local self-adaptation a request previously canceled can be planned again, without restarting the planning process from scratch and without perturbing the whole plan.

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, ATLAS takes into account requests without reconsidering its whole planning, in a short delay. Thus, near real-time planning can be performed.

4.2.2 Delay Average Time

Dynamic planning allows to produce near-real time plans. To make such a planning process possible, the system must be able to treat a new request at any moment. For operational reasons, a request must be taken into account and planned as fast as possible. Indeed, one of the major advantages of near-real time planning is the possibility to submit urgent requests that can be taken into account very fast.

To illustrate the delay of planning, we proposed to analyze on the previous experiment the time to adaptation when a perturbation, corresponding to the submission of a set of requests. Figure 6.5 represents the percentage of not planned meshes during the execution.

The two perturbations (1 and 2) correspond to the submission of the set of normal and urgent requests.

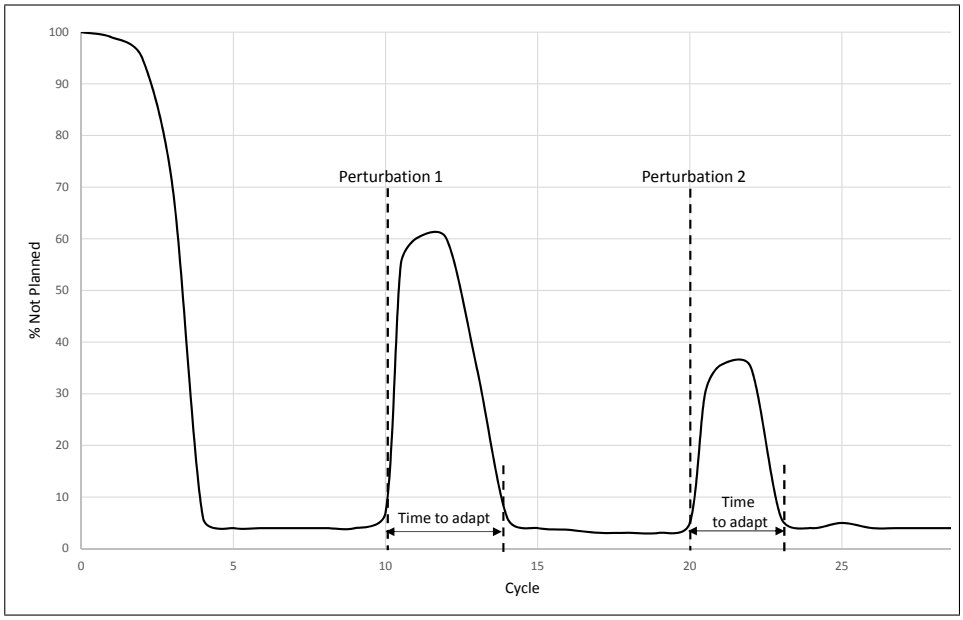


Figure 6.5 — How ATLAS adapts itself to the perturbation

As the curve illustrates it, ATLAS quickly adapts itself to take into account new requests to plan. After the submission of the normal set, it takes 4 cycles to retrieve the same percentage of not planned meshes and recover from the perturbation. The urgent requests are taken into account after 3 cycles. Even if they only represent 15% of the whole request, 96% of the requests are already planned when they are submitted. This quick adaptation is explained by the criticality. As they are urgent, the other agents let their place to the new corresponding mesh Agents. We can also see that the other agents quickly find a new place. The curve also highlights the fact that the system stabilizes itself regardless the number of meshes dynamically submitted. This experiment represents the worst case because requests were submitted by set and ATLAS has to adapt to a huge number of perturbations. If requests were submitted all along the planning time, they will cause smaller perturbations.

4.2.3 Synthesis on Dynamic Management

Those experiments prove that ATLAS is able to dynamically treat requests. Thanks to self-adaptation mechanisms, agents quickly adapt their organization to favor high priority requests.

Moreover, more the mesh has a high priority, more quickly it is planned. The criticality measure guides the cooperation. Thus, agents with a lower priority possess a lower criticality than urgent ones, and they let their initial place before finding a new one. Finally, the whole plan is not reconsidered. The agents locally cooperate.

4.3 Scalability

The number of daily requests to treat is increasing, as the number of satellites to operate. Be able to deal with an important number of entities, in a reasonable execution time, is more than ever important to space systems. Thus, ATLAS must be scalable toward the number of the satellites and the number of meshes to plan. As ATLAS manages the whole constellation as the same time, it always treats more requests than a standard system operating one satellite at a time. Scalability is so an important evaluation criteria. The ATLAS scalability is measured with two experiments: the impact of the constellation size and the behavior of ATLAS when the number of requests to treat increases.

4.3.1 Impact of the Constellation Size

III

To study the impact of **constellation size**, we rely on 7 different scenarios produced by the generator and detailed in Section 2.1: scenario number from III to VIII of Table 6.1.

Figure 6.6 illustrates the results for the 7 tested scenarios. Each scenario has been tested 10 times, the graph shows the minimal and maximal percentage of planned meshes obtained during those runs.

The increasing number of satellites does not penalize ATLAS. Results are constant, the minimums planned meshes vary between 94.8% and 95.3% (a range of 0.5%) while the maximum has a range of 0.49% (between 96.01% and 95.52%). Moreover, the results dispersion per scenario is also short, the maximal difference is obtained for the 10-satellite scenario with a deviation of 1.11%.

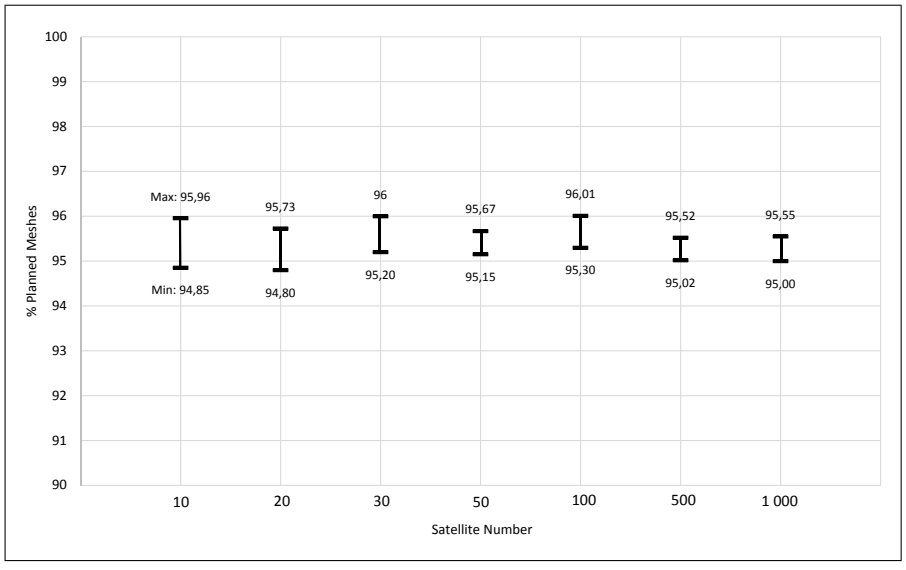


Figure 6.6 — Results for the 7 constellations

Figure 6.7 gives the percentage of high priority planned meshes for each scenario. For the 10, 20, 30 and 50-satellite constellations, all of the high priority meshes are planned. One mesh is missing for the 100-satellite constellation. The last two constellations do not reach the 100%, but more than 96.5% of the high priority meshes are planned. For the three

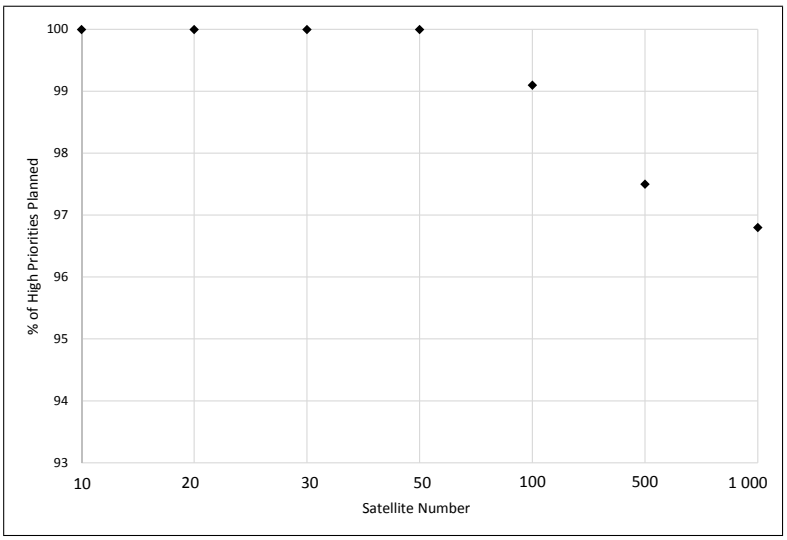


Figure 6.7 — Percentage of high priorities planned for the 7 constellations

scenario this loss is explained by the scenario complexity. Indeed, thousands of meshes can be planned in similar visibility access, and high priority meshes are in conflict together. Regarding the important number of meshes to plan, this loss is minimal.

The first experiment has been repeated using dynamics. In this version, meshes are randomly inserted into ATLAS. The results are coherent: for the 7 constellations, all the percentage of planned requests are located in the same range given by Figure 6.6. Those results show again that ATLAS can perfectly manage the dynamic. The first experiment illustrates the adequacy of ATLAS to manage huge constellations.

4.3.2 Impact of the Number of Meshes to Plan

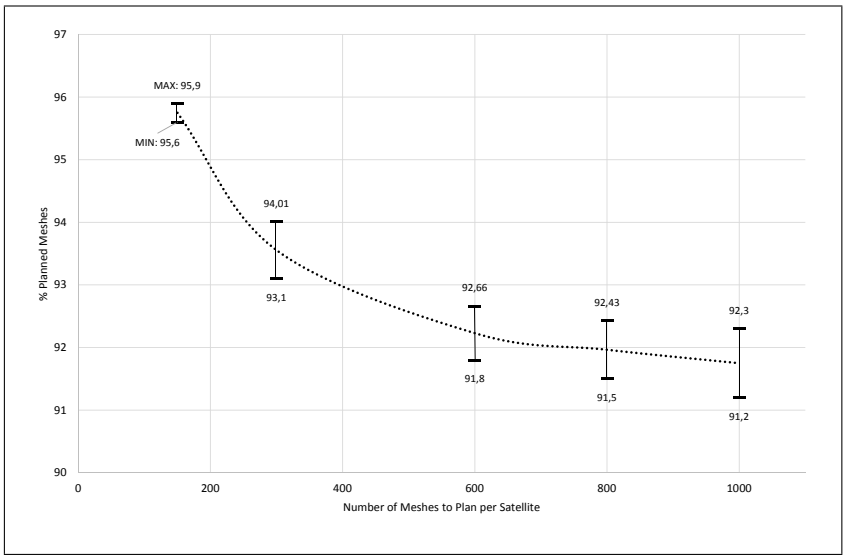


Figure 6.8 — Evolution of the percentage of planned meshes when the number of meshes per satellite increases

The second kind scalability to test is the scalability toward **the number of meshes** to plan. For the second experiment, the number of satellite is constant but the number of meshes to plan by satellite is increasing. Scenarios used in this experiment are scenarios number from IX to XIII of Table 6.1. Because each mesh can be planned by more than one satellite, each satellite must manage more than the number of meshes initially affected to it. Those scenarios are highly complex: the high number of meshes to plan is located into a short time window, causing a lot of conflicts between meshes.

Figure 6.8 shows the results. For each scenario, the minimum and maximum percentages of planned meshes are illustrated on the graph. The curve represents the average trend. Results highlight that ATLAS can manage huge number of meshes to plan per satellite. The trend shows that the percentage aims to stabilize between 91% and 93%.

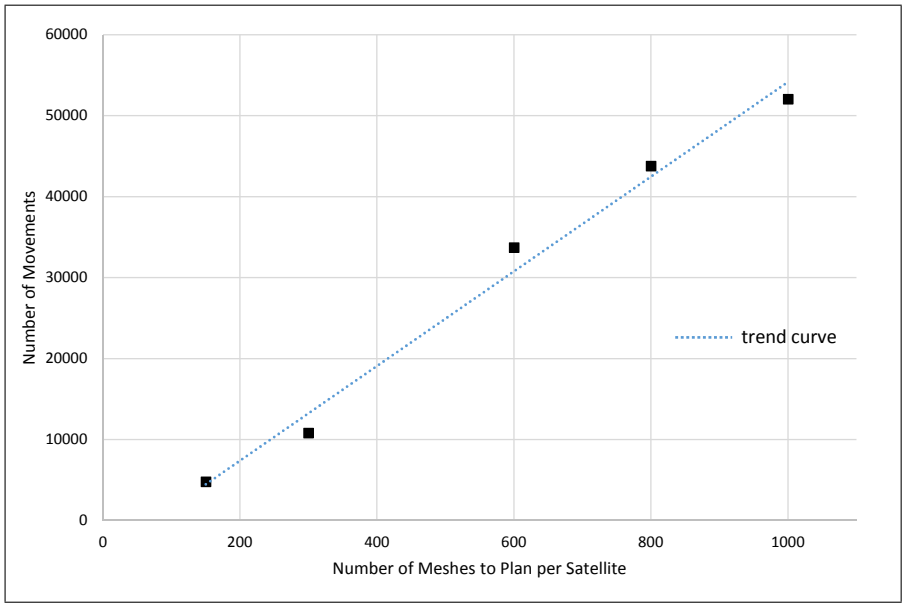


Figure 6.9 — Evolution of the number of mesh movements regarding the number of meshes per satellite

Figure 6.9 shows the evolution of the movements of the mesh Agents (a movement corresponds to a plan/cancel transition) when the number of meshes to plan increases. Increasing the number of meshes to plan increasing the complexity of the scenario. More meshes imply more hot-spots with huge combinatorial because those new meshes have accesses at several places in the scenario and so increase the number of conflicts. Nevertheless, the Figure 6.9 shows the evolution is linear. ATLAS is able to deal complex scenarios.

4.3.3 Discussion on Scalability

From the experiments on scalability management, several points can be characterized. On the constellation size scalability, ATLAS is not impacted by the increase of the number of satellites to manage, when the average accesses per meshes and the number of meshes per satellite is constant, even if the requests are randomly submitted to ATLAS. The system also respects the priorities by planning a maximum of high priority requests. The scalability

is also respected when the number of requests to plan per satellite increases. Finally, even if the number of meshes to plan and the number of potential solutions increases, we show that the evolution of the movements in ATLAS is linear.

4.4 Load Balancing

An advantage of using a constellation is to be able to fair-distribute the load among it. Thus, it ensures resilience on the constellation (no satellite is more important than another), and if a critical request is submitted, the satellite has more chance to possibly treats it without conflict because it is not overloaded.

To highlight how ATLAS addresses this issue, we focus scenario number from III to VII of Table 6.1. Figure 6.10 shows the results of distribution percentages in a box plot, one for each scenario. The bottom corresponds to the satellite with the minimal load, the top to the satellite with maximal load. The box represents the first and the third quartiles (respectively the bottom and the top of the box).

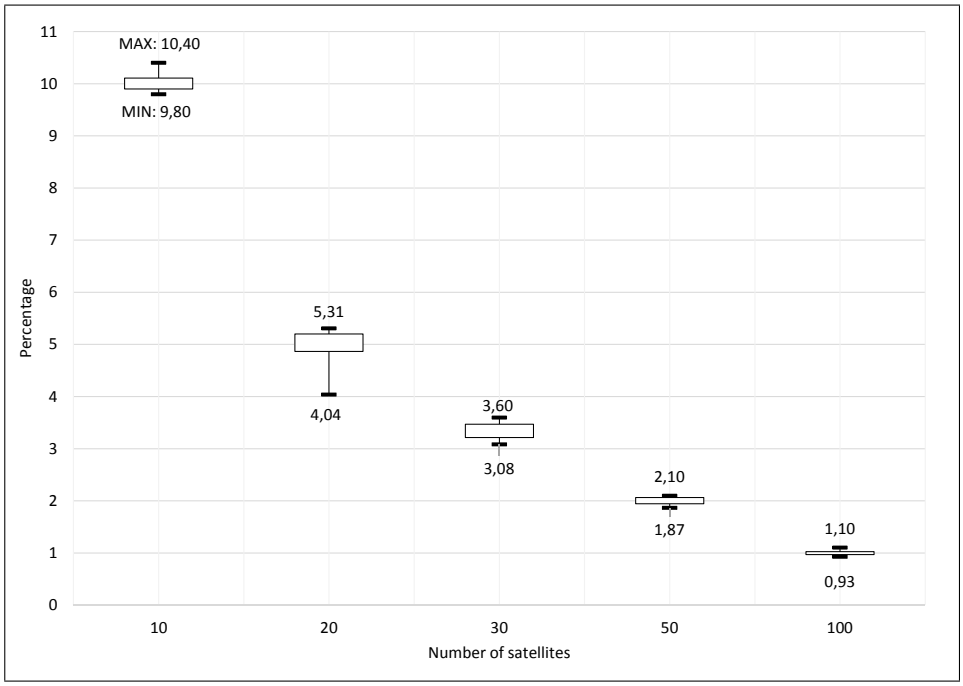


Figure 6.10 — Load balancing for 5 constellations

The plots show that the load is fairly distributed inside the constellation. The average load (respectively 10%, 5%, 3.33%, 2% and 1%) are always between the first and the third quartile, illustrating a good repartition. The maximal range are for the 20-satellites (1.31%) and the 10-satellites (0.60%) constellation, but that range is explained by the small number of satellites: a difference of few meshes implies a high percentage. More the constellation size increases, more the range decreases: only 0.17% of difference between the more and the less loaded satellites in the 100-satellite constellation.

4.4.1 Discussion on Load Balancing

An interest in using a constellation is to have the possibility to fairly distribute the load among the satellites. In ATLAS, the cost represents the difficulty for satellite Agent to plan a mesh, it is composed of a list of several measures.

There is no global rule imposing a balancing, the load emerges among the constellation. Mesh Agents use cost criteria to naturally favor less loaded satellite Agents. Of course, if satellites do not have the same number of meshes to plan, the load is not equal.

4.5 Discussion on Generic Validation

In this section, we present different experiments to test ATLAS on its dynamic management, capacity of scalability and the balance of the solution.

- Self-adaptation and criticality allow to dynamically plan meshes while ATLAS is running. Moreover, the delay to take into account such perturbations is short, in particular for urgent meshes.
- ATLAS can be used to plan both huge constellations and huge set of meshes. The experiments show the scalability of ATLAS.
- Finally, ATLAS produces plan where the meshes are fairly distributed among the constellation.

In the next section, we compare ATLAS to an algorithm currently used in Europe: the Hierarchical Greedy Algorithm.

5 Comparison to an Operational Algorithm

In this section, the standard algorithm used in Europe to plan mission of satellites is first introduced. Then, the experiments made on ATLAS to compare it to this standard algorithm are presented. Those experiments rely on an operational mission scenario given by partners of the OCE project, in which this thesis has been made.

5.1 Operational Algorithm: Hierarchical Greedy Algorithm

In Chapter 2, a state of the art of current optimization approaches is proposed. Among them, in the approximate methods class, we found the algorithm the most used today in operation centers: **Hierarchical Greedy Algorithm**. To ease the reading, in the following of this chapter we simply name it **HGreedy Algorithm**.

Basically, HGreedy Algorithm follows a heuristic that consists of making the local optimal choice at each stage with the hope of finding a global optimum. In mission planning, the Greedy Algorithm follows two steps:

1. First, a **note** is given to each mesh, using several criteria like the priority, image quality or weather information. Using those notes, accesses are sorted. The note is similar to the criticality: more it is high, more the access is critical.

2. Then, the list is unstacked, starting from the more critical, an **insertion** into the mission is tried for each access. If a mesh cannot be inserted, it is ignored: the plan is never reconsidered.

5.2 Test on Spots Constellation Missions

5.2.1 Experimental Protocol

In the first section, tests were made on generic scenario. In those scenarios, satellite maneuvers are not considered. The acquisition slot given by the generator is composed by the acquisition duration and a standard maneuver duration. This abstraction is not a simplification, the problem remains the same: find the right begin of the acquisition in the access interval.

In real scenario, satellites perform maneuvers during acquisitions (*the guidance*) in order to always have their acquisition instruments oriented toward the mesh, and maneuvers between acquisitions to be correctly positioned before the next acquisition start. Figure 6.11 illustrates the acquisition of 2 meshes and the maneuver between them.

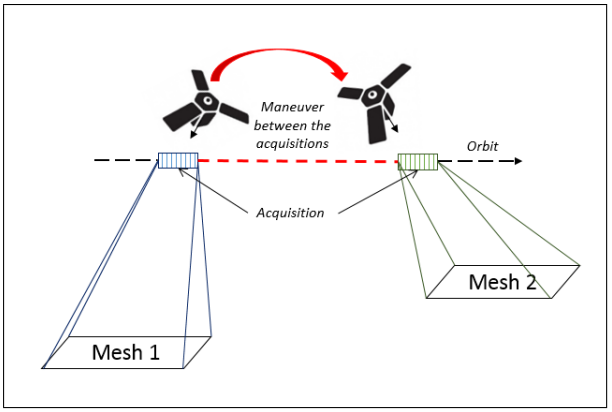


Figure 6.11 — Two acquisitions and a maneuver

Those maneuvers are computed by an external library. During their reasoning, cooperative satellite Agents call functions to compute maneuvers and guidance. In order to perform fair validation, the external library is the same for ATLAS and the HGreedy Algorithm.

Spot 6	836
Spot 7	733
Total	1 569
Common Accesses	47

Table 6.7 — Spot 6/7 scenario details on accesses

In this section, we propose to compare plans obtained with ATLAS and the Greedy

Algorithm on a real Spot mission scenario. This scenario represents a 4-hour mission, with 1 182 meshes to plan. The constellation is composed of 2 satellites: Spot 6 and 7. The satellites are on the same orbit, with 180° of phase difference. Thus, few number of meshes possess an access on both the satellites.

The table 6.7 details the accesses of the scenario, the total is higher than the number of meshes because some meshes can be acquired by the two satellites. Moreover, 47 accesses are common to the two satellites: they can be acquired by the two satellites.

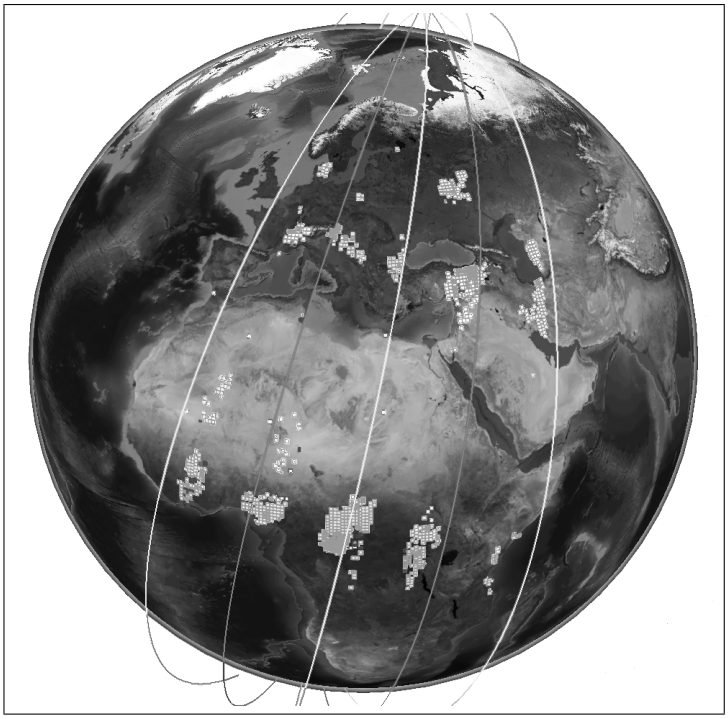


Figure 6.12 — Spot 6/7 scenario visualization through XML-Visu

Figure 6.12 shows the display of the scenario using *XML-Visu* software visualization, provided by Airbus Intelligence[1]. On this figure, we can note the fact that meshes are not present all along the satellites orbits, but they are concentrated into areas of interest called *hot-spots*. Because of the important concurrence between meshes on certain *hot-spots*, it is impossible to plan the whole meshes. When two or more meshes are in concurrency (*i.e.* when they are overlapped or very close), their accesses are in concurrence too: during certain period of its orbit, the satellite can acquire several meshes. This cover is a factor of concurrency.

This concurrency is highlighted using the average percentage of cover in Figure 6.13, where percentage are grouped by class. At least 50% of the accesses possess less than 20% of cover, but 25% have between 20 and 40% and 8% are in high concurrency areas, with at least 80% of cover.

This analysis shows that in this scenario, the concurrency between the satellites is not an important factor: only 47 accesses are common on both the satellites. This is explained by the fact that Spot 6 and 7 have the same orbit, and there are enough shift to not visit same

[1] http://www.intelligence-airbusds.com/

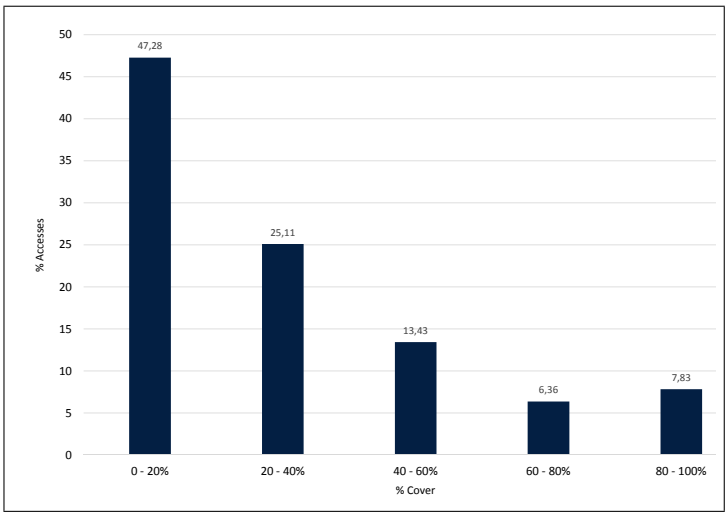


Figure 6.13 — Spot 6/7 scenario: percentage of cover between accesses

areas. Moreover, this is only a 4-hour mission, with only 5 orbits. Still, the analysis of the access covering shows that there are important concurrence areas. In those, the complexity is very important, so it is impossible to plan all the meshes.

In the following section, two additional scenarios have been studied. Those scenarios are not real but were built using real data:

- **4-Satellites**, based on the original Spot mission (1 724 accessible meshes), where 2 additional satellites are added. Their orbits are constructed to add more conflicts: they cross the original Spot 6 and 7 orbits above the main areas of interest.
- **feasible**, counts 8 satellites and 1 209 meshes to acquire. It is constructed using the same pattern as the scenario generator: all of the 1 209 meshes can be planned.

5.2.2 Results on Spot Mission

From an operational point-of-view, several criteria are considered to judge a mission plan. In this study, we only focus on the percentage planned meshes with high priorities, the number of planned meshes in the mission and the number of calls of maneuvers and guidance computation library (those computations are costly in terms of CPU and memory usage).

	ATLAS	Greedy
Planned Acquisitions	220	209
High Priorities (%)	100	100
Guidance	33 500	21 800
Maneuvers	64 200	94 500

Table 6.8 — Spot 6/7 mission results

Table 6.8 shows the results for the plans obtained with ATLAS and HGreedy Algorithm.

ATLAS plans more acquisitions (220 versus 209), but the percentage of high priorities are the same (100%), the difference between the two plans is on low priorities. ATLAS calls more often the guidance API. This high number is caused by the self-organization process. Indeed, HGreedy never reconsiders a choice: when an acquisition is planned, it is never canceled. On the contrary in ATLAS, the agents can be planned, canceled and then re-planned many times, depending on criticality evolution, causing more calls to guidance function. Nevertheless, HGreedy performs more calls to the maneuvers function. This can be explained by the fact that after booking an acquisition, the algorithm tries to minimize the maneuver with previous acquisition.

Regarding those criteria, plans delivered by ATLAS are similar to those delivered by the HGreedy Algorithm. On the total of planned meshes criteria, ATLAS is better by 11 acquisitions.

III

5.2.3 Results on 4-Satellites & Feasible

	4-Satellites		Feasible	
	ATLAS	Greedy	ATLAS	Greedy
Planned Acquisitions	359	331	1 180	1 175
High Priorities (%)	100	100	95	95
Guidance	162 000	75 690	105 000	109 000

Table 6.9 — 4-Satellite and *feasible* mission results

Table 6.9 shows the results for 4-Satellites and Feasible. In the first scenario, ATLAS plans 8% more acquisitions but the same percentage of high priorities are planned by the 2 systems. On *feasible*, ATLAS plans few additional acquisitions, but here again both the systems plan 95% of the urgent priorities.

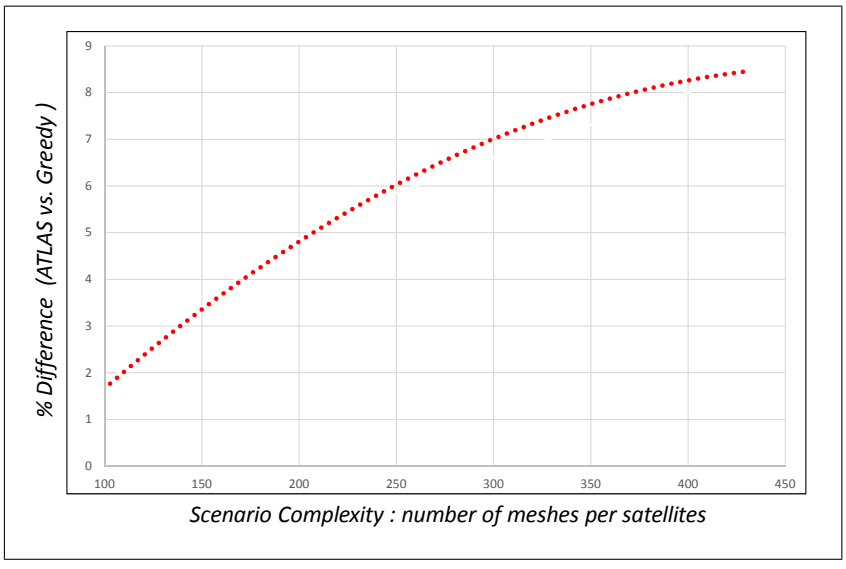


Figure 6.14 — Difference between ATLAS and the Greedy system

All those scenarios proved that ATLAS can manage real data and produce plans with better results quality than the operational reference Algorithm.

Figure 6.14 illustrates the differences between both the systems when the number of meshes to acquire per satellite increases. This curve illustrates the advantage of ATLAS: more the satellites have meshes to acquire, more the percentage of planned meshes is important for ATLAS.

5.3 Mission Chronology

As presented in Chapter 1, today's mission planning systems follow three steps. Firstly, requests are stored. Then, before the satellite is visible from the ground control, the plan is required, so the poll of requests is submitted to the planning algorithm. Finally, when the plan is ready and the satellite is in visibility the transmission is processed.

This way of work brings the advantage to only start the algorithm a few times a day, but it is incompatible with near-real time planning. Requests are treated one poll at a time. If a request arrives while the system is planning, it is stored for the next planning period, for example 4 hours later. In the case of urgent requests, for example look for an airplane crash, this delay could be harmful.

The following experiment illustrates two mission chronologies, one with ATLAS and one with the HGreedy Algorithm, on the same scenario, representing a typical programming day: requests are submitted all along the day before the plan is required. In this scenario, we consider 10 sets of mesh submissions.

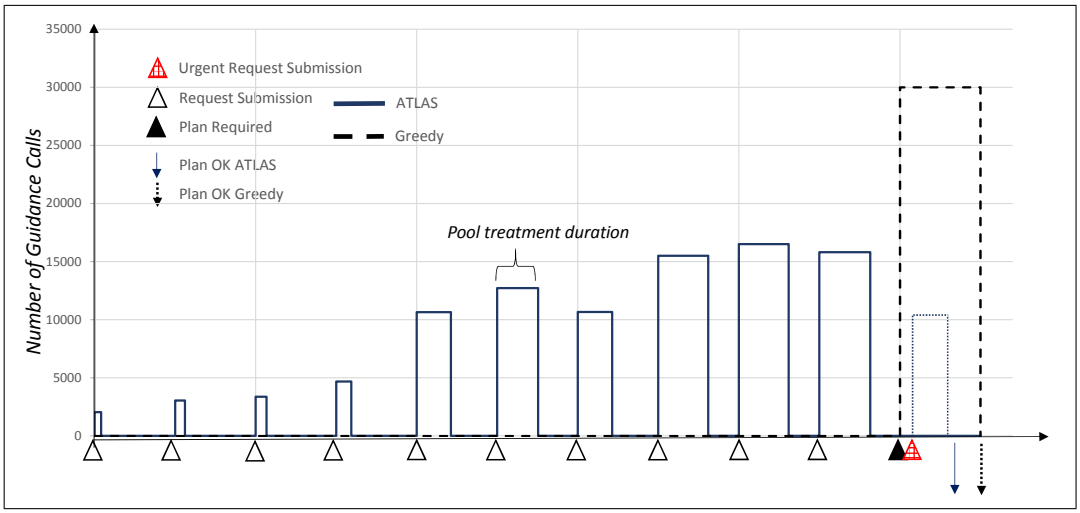


Figure 6.15 — Mission chronology: urgent submission after the plan request

The Figure 6.15 shows the results. The 10 submissions are distributed all along the day before the plan request. The height corresponds to the number of calls to the guidance API to adsorb the submission. Thanks to its capacities to handle dynamic, ATLAS treats requests at their submissions. When the plan is required, ATLAS performs no computation: the plan is ready. On the contrary, the HGreedy Algorithm does nothing before the request for a plan. At this time, the plan processing is performed, causing a lot of computations and

a certain delay before that the plan . Moreover, even if ATLAS performs more call to the guidance function, there are distributed among the programming period.

Figure 6.15 also contains a variation: a set of highly urgent requests is submitted just after the request for the plan. HGreedy cannot take this submission into account, because the planning process is already launched. ATLAS, thanks to its self-adaptation can manage them. The plan is reorganized and delivered before the HGreedy's one, and contains the urgent requests. Moreover, with the Stop mechanisms (presented in Chapter 4), the exposed plan guarantee coherence.

This experiment highlights the advantages of near-real time planning. With this planning paradigm, requests are treated at their submission and the plan can be delivered faster. If an emergency request is submitted just after the plan is requested, it can be treated, on the contrary of Greedy Algorithm that does not allow dynamics. Moreover, ATLAS quickly adapts its plan to new request. Even if adding requests leads to more conflicts and so more computations, agents cooperation allows a fast adaptation.

5.4 Discussion on Operational Validation

This section focuses on an operational point-of-view. We compare ATLAS to the HGreedy Algorithm, the standard algorithm currently used in Europe mission centers. Tests were performed using real missions from Spot constellations, and scenarios build using Spot satellite specifications. To be fair in our evaluations, the same platform has been used. Thus, both the algorithms rely on the same satellite maneuvers control computation library, and modules to receive requests and to deliver plans.

On the four operational criteria, results are roughly the same, but the number of planned acquisitions is higher with ATLAS. The HGreedy Algorithm performs fewer calls to the maneuver computations library. This is due to the fact that in the ATLAS system, the agents work in cooperation, they all try to be planned and cooperate in case of conflict or to optimize their placement. On the contrary HGreedy never reconsiders a decision, and plans the requests one at a time.

In standard way of planning, this important number of calls could be a drawback. Nevertheless, self-adaptation allows to produce near-real time planning. Thus, ATLAS works continuously, the calls are performed all along the run, and not when a plan is requested. Dynamic abilities is a major breakthrough. Requests can be dynamically treated, making the system more reactive.

6 General Synthesis

This chapter presents the different experiments realized to highlight ATLAS properties. The first Part points that current approaches cannot manage dynamics. Furthermore, the trend is to an increasing number of requests and satellites. New systems must be able to deal with this scalability. Experiments we have made focus on those points.

Using scenarios build thanks to the scenarios generator, ATLAS has been tested with

various, generic and complex data. Those scenarios help us to characterize ATLAS on three points:

1. **scalability**: ATLAS is able to manage constellations with an important number of satellites and/or to treat a huge number of requests to plan,
2. **dynamic**: on the contrary to standard algorithm, ATLAS runs continuously and can take into account requests when they are submitted. Moreover, experiments have shown that perturbations caused by such submissions are quickly regulated, and high priority requests are quickly treated,
3. **load balancing**: plans produced by ATLAS take advantage of the constellation, the mission load is fair distributed among the satellite.

On an operational point of view, ATLAS has been tested and validated on a **platform** integrating the standard algorithm (Hierarchical Greedy Algorithm). Those tests have shown the ATLAS adequacy on real missions. Our system can be used with real data and satellite management systems. Results prove that ATLAS delivers high quality plans. Moreover, ATLAS allows to perform near-real time planning. With this way of managing satellite systems, emergencies can be quickly treated, and computations are made all along the programming run, and not only when a plan is required.

Further experiments would allow to test ATLAS on long missions (for example a week of planning) and using scenarios with more conflicts between meshes.

Table 6.10 summarizes the considered and not considered criteria, regarding criteria defined in Section 2.

Criteria	Considered	Not Considered	Remarks
Development Methodology	✓		cooperation ease the development
Ease of the Implementation	✓		
Importance of the Heuristic		X	ATLAS is not based on a heuristic
Genericity of the Heuristic		X	
Dynamic Management	✓		
Execution Time		X	AMAS middleware makes not easy the measure
Guidance Function Call	✓		
Load Balancing	✓		

Table 6.10 — Summary of the considered or not criteria, regarding criteria defined in Chapter 2

Part IV

Conclusion and Perspectives

7 Conclusion and Perspectives

« Avoir tort le premier jour et raison le second, voilà l'histoire de tous les grands apporteurs de vérités. »

Victor Hugo

This thesis started from the analysis of complex problems and the difficulties to efficiently solve them. The huge number of entities composing them, their non-linearity and the important dynamics imply that today's approaches reach hard limits.

In Chapter 2, we see that common optimization methods (exact or inexact approaches) fail to manage complex problems. New methods like *ACO* or *PSO* allow to take into account dynamics but they show limits: they are slow and not suited to manage a huge set of entities.

Self-organized systems seem promising to solve complex problems. Their natural abilities of openness and self-adaptation make them able to dynamically take into account new events. Among those systems, we focus on Adaptive Multi-Agent System (AMAS), Chapter 4. AMAS are based on the local cooperation of entities, *cooperative agents*. The local interactions and negotiations made *emerge* the adequate function. AMAS4Opt is a generic agent model specifying the AMAS Theory for solving optimization problems. This model has shown its efficiency to ease the design of adaptive multi-agent systems for several problems. Still, some enhancements concerning cooperative behavior, were required (Chapter 4).

Among the complex problem, this thesis focuses on a particular one: mission planning for constellations of Earth observation satellites. It is a complex problem raising significant technological challenges for tomorrow's space systems. The large and heterogeneous numbers of customer requests and their dynamic submission in the planning system result in a huge combinatorial search space with a potentially highly dynamic evolution requirements. Those evolution and characteristics made this problem a very interesting complex problem to study, with immediate repercussions.

In this thesis, we rely on AMAS and the enhancements of AMAS4Opt to design ATLAS (presented in Chapter 5), an adaptive multi-agent system to solve mission planning problems. Chapter 6 highlights several experiments made on ATLAS in order to test and validate it.

Scientific Contributions

This thesis focuses on complex problems solving using Adaptive Multi-Agent System. [Kaddoum, 2011] proposes a generic agent model to solve complex problems: AMAS4Opt. In this thesis, we bring two enhancements for this model. The first enhancement is the on choice of service for agent under the "Constrained Role". The **cost** allows agents under the "Constrained Role" to cooperatively select the relevant service. The second enhancement aims to reduce the behavior complexity **by enhancing the "Service Role"**. The propose pattern help designer to apply the 13^{th} activity of the ADELFE development methodology [Bonjean et al., 2014] to AMAS4Opt.

Our two other contributions concern AMAS more generally, and do not explicitly focus on optimization problem.

In an adaptive multi-agent system, the agents collective behavior makes emerge the global function. Alone, an agent may not possess the whole behavior to solve a problem and must require help from other agents. For that, we propose a **design pattern to cooperatively relax constraints through the agent neighborhood**. In this pattern, this search for help is locally made, by a transmission from neighbor to neighbor.

AMAS are designed to run continuously. The system self-adapts to dynamic changes, and the agent organization provides a solution. Thus, the stop question remains: how to be sure that when a solution is required, the system is not in a transition (and potentially incoherent) state? This thesis details **rules of conception** to quickly stop an AMAS, while proposing a **coherent solution**. Those rules and the associated coherence proof are a first step toward formal proof of AMAS.

The experiments done have shown the bring of AMAS for complex problem solving. Moreover, we highlight the advantage of cooperation on the design and improvement of agent behavior process. The main limit of this theory stays its high level of abstraction, but the new AMAS4Opt model simplifies its use by non-experts.

Applicative Contribution

In this work, we choose a relevant problem applying our scientific contributions: mission planning for constellations of Earth observation satellites. While, nowadays approaches reach hard limits, the space fields are evolving, requiring new methods to plan missions.

ATLAS System

From the scientific contributions proposed in this thesis, we design the ATLAS system. Interactions between agents allow ATLAS to plan missions for constellations of different types of satellites. Moreover, ATLAS can, at any time, take into account new requests, which are treated without reconsidering the whole plan, by acting locally (where conflicts appear).

Scenarios Generator

Atlas has been tested on generic and real scenarios. The generic one has been generated using a generator, validated by experts of the domain. To compare ATLAS on real scenarios, a standard algorithm has been developed, based on definition and model of a system currently used to plan missions.

Generic tests prove that ATLAS has relevant properties:

1. it is **scalable**, it can treat mission planning with an important number of entities (requests to plan and/or satellites in the constellation),
2. it handles the **dynamism**, it is possible to add or remove requests, update constraints, while the system is running, and those modifications are quickly taken into account,
3. it is **reactive**, it provided feedback on request state.

Operational Validation

On the different tests made to compare it to an operational Hierarchical Greedy Algorithm, ATLAS is at least as efficient regarding the number of planned meshes. Nevertheless, thanks to its dynamic properties, ATLAS delivers plans faster. This capacity of taking into account dynamic event is a major breakthrough. Today's operational systems, at least in Europe, cannot manage dynamics, they must rebuild the plan to make changes. This paradigm begins to be limited with the increasing number of requests and satellites, and the need to plan mission faster.

In operational systems, plan computation is started when a request for plan is operated. Thus, the plan is not ready when it is requested. On the contrary, as ATLAS works continuously, computations are made when customer's requests are submitted. Thus, the plan is finalized more quickly when a request for plan is sent.

Perspectives...

This work opens new directions in optimization, AMAS and mission planning systems.

From a Scientific Point of View

In the current version of the AMAS4Opt model, criticality and cost are defined by domain experts. They are composed of several ordered criteria. Using the recent researches made by the SMAC team on context learning [Boes et al., 2015], the system can find by itself the order between criteria, depending on the context. Thus, the role of experts would be only to list the criteria, the order and the importance of each would emerge.

A step further would be to let the system find the composition of the cost and criticality, without domain experts specifications. Thus, AMAS4Opt would become a generic system, able to manage any optimization problem.

Finally, in this thesis the service delegation and neighbors relaxation have been introduced and applied. Both those mechanisms must be applied to other applications to perform more tests.

From an Applicative Point of View

The comparison of ATLAS to an operational system have underlined it efficiency. Still, some constraints must be implemented to make ATLAS ready to be used in operational context. For example, the use of a real battery management module or weather module could fill the gap between ATLAS and today's systems.

Today, satellites only follow their mission plan. Studies are made to give them more autonomy. Thus, we could imagine overloaded plans on ground and satellites taken on-board decisions to select acquisitions to perform, depending on their memory or the weather, for example. In this model, ATLAS could be used to build the overloaded plans. An embedded version of ATLAS could also be designed to be executed by satellites, on-board.

As ATLAS remains at a generic level, it could be used to plan other systems, like U.A.V. (Unmanned Aerial Vehicle), traffic management system or management product line. In the OCE project, where this work takes place, other problems linked to mission planning are treated, like the large coverage problem. The purpose of this problem is to optimize to cutting of large request (continent, country, *etc.*) into meshes, taking into account satellite orbits, capacities and weather. ATLAS could be tested to this problem and provide solutions. Thus, ATLAS will treat the cutting and the planning phases of the mission chain.

ATLAS could be used to manage a lot of ground centers. As it is distributed, a central station could be the reception center and dispatches the requests toward sub-centers only in charge of the plan of a satellite or a sub-system. ATLAS is open, so the sub-centers would not necessarily use AMAS system, but manage the plan as they want as they are able to send to the control center a cost of planning. In this configuration, each sub-center would work as a satellite Agent: mesh Agents (in the ground center) does not know the functioning of each sub-center. Such an architecture would highlight the service delegation.

About Emergence, Final Words

To understand *complexity*, it is necessary to understand the word itself. Edgar Morin uses the Latin etymology to define it: *complexus*, literally *"what is woven together"* [Morin, 1995]. This definition strongly highlights the non-linearity of complex systems. It is also useful to distinguish what is complex to what is complicated/difficult.

A complicated system can be assimilated to a machine composed of small parts, each having its precise role to produce a well-defined function. On the contrary, a complex system possesses non-linearities, a huge number of entities and it is their interactions that produce the global behavior, but this behavior cannot be predicted from the individual rules. The quote of Aristotle *"the whole is greater than the sum of its parts"* sums up one of the key concepts of complex systems: *emergence*.

The bottom-up approaches producing emergent functionalities are fascinating. They take the exact opposite of reductionist approaches. Instead of considering the whole system, bottom-up required to find entities and simple interaction rules from which complexity emerges.

I found with the SMAC team[1] and the AMAS theory a very interesting way to study complexity. Researchers and members of the SMAC team, performed a lot of works on different application fields, underlining concrete realizations of emergent artificial systems.

To produce emergence, the AMAS theory proposes the use of the cooperation between entities. Focus on local cooperation rules allows to get around the complexity. At a local level, it is easy to define simple rules. Using cooperation, entities can adapt their behavior and dynamically change their organization and produce a different, but adequate, function.

For me, this thinking about emergence is fantastic. Entities of complex systems do not know there are a part of whole. Like fishes or birds does not know they constitute a flock. If we consider our brain as a complex system, as many recent researches suggest it, what is the emergent function? May our conscience be this emergent property [Searle and Tiercelin, 1995]? I am sure that emergence and the thought of the complex will take an important role on the theories of tomorrow.

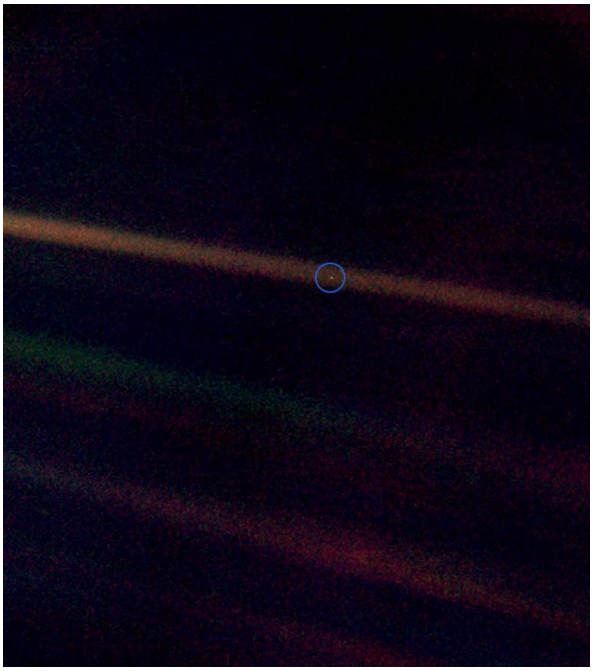
This work would not have been possible without the kindness of my supervisors, and each member of my two teams. I would like to conclude this thesis by sharing with them a reflection made the author and astronomer Carl Sagan in [Sagan, 1994] about the photography *A Pale Blue Dot*. This picture of our solar system has been taken by the probe Voyager 1[2] in 1991, at 6,4 billion kilometers from our planet, near to the Pluto orbit. In a ray of light, we can see a tiny dot: the Earth. It is fantastic to think that here, in this little pixel, somewhere in the darkness of the Universe, billions of entities live and interact together, producing life.

"We succeeded in taking that picture, and, if you look at it, you see a dot. That's here. That's home. That's us. On it, everyone you ever heard of, every human being who ever lived, lived out their lives. The aggregate of all our joys and sufferings, thousands of confident religions, ideologies and economic doctrines, every hunter and forager, every hero and coward, every creator and destroyer of civilizations, every king and peasant, every young couple in love, every hopeful child, every mother and father, every inventor and explorer, every teacher of morals, every corrupt politician, every superstar, every supreme leader, every saint and sinner in the history of our species, lived there - on a mote of dust, suspended in a sunbeam."

— Carl Sagan, A Pale Blue Dot, 1994.

[1] http://www.irit.fr/smac
[2] http://voyager.jpl.nasa.gov

IV



"You are here."

Personal Bibliography

National Journal Paper

- Bonnet Jonathan, Gleizes Marie-Pierre, Kaddoum Elsy and Rainjonneau Serge, *ATLAS : planification multi-satellite dynamique en temps réel,* In Revue d'Intelligence Artificielle, Spécial JFSMA 2015/2016 (RIA), volume 30, n 1-2/2016, p. 35-59.

International Conference Papers

- Bonnet Jonathan, Gleizes Marie-Pierre, Kaddoum Elsy, Rainjonneau Serge and Flandin Grégory, *Multi-satellite mission planning using a self-adaptive multi-agent system,* In 9^{th} IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'15), Boston, United States, p. 11-20.
- Bonnet Jonathan, Gleizes Marie-Pierre, Kaddoum Elsy and Rainjonneau Serge, *Rapid and adaptive mission planner for multi-satellite missions using a self-adaptive multi-agent system,* In 67^{th} International Astronautical Congress (IAC), Guadalajara, Mexico.

National Conference Paper

- Bonnet Jonathan, Gleizes Marie-Pierre, Kaddoum Elsy and Rainjonneau Serge, *Planification de missions multi-satellites par système multi-agent cooperatif,* In 23^{eme} Journees Francophones sur les Systemes Multi-Agents (JFSMA'15), Rennes, France, p. 113-122.

Patent

- Gleizes Marie-Pierre, Kaddoum Elsy, Bonnet Jonathan, Rainjonneau Serge and Flandin Grégory, *Procédé et dispositif d'élaboration d'un programme d'acquisition de données complexes par une constellation de dispositifs d'acquisition de données élémentaires,FR3038089 - 2016-12-30 (BOPI 2016-52)*.

Bibliography

AKÇAY, Y., LI, H., AND XU, S. H. 2007. Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research 150,* 1, 17.

BALBO, F. AND PINSON, S. 2005. Dynamic modeling of a disturbance in a multi-agent system for traffic regulation. *Decision Support Systems 41,* 1, 131 – 146.

BELGHACHE, E., GEORGÉ, J.-P., AND GLEIZES, M.-P. 2016. Towards an adaptive multi-agent system for dynamic big data analytics (short paper). In *The IEEE International Conference on Cloud and Big Data Computing (CBDCom), TOULOUSE, 18/07/2016-21/07/2016*. IEEE Computer Society - Conference Publishing Services.

BELLMAN, R. 1952. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences 38,* 8, 716–719.

BENSANA, E. AND VERFAILLIE, G. 1999. Earth Observation Satellite Management. In *Constraints*. Vol. 299. 293–299.

BENSANA, E., VERFAILLIE, G., AGNÈSE, J., BATAILLE, N., AND BLUMSTEIN, D. 1996. Exact and inexact methods for the daily managament of an Earth observation satellite.

BENSANA, E., VERFAILLIE, G., AGNESE, J., BATAILLE, N., AND BLUMSTEIN, D. 1996. Exact &inexact methods for daily management of earth observation satellite. In *SpaceOps' 96*. Vol. 394. 507.

BERNON, C., GLEIZES, M.-P., PEYRUQUEOU, S., AND PICARD, G. 2002. Adelfe: a methodology for adaptive multi-agent systems engineering. In *International Workshop on Engineering Societies in the Agents World*. Springer, 156–169.

BIANCHESSI, N., CORDEAU, J. F., DESROSIERS, J., LAPORTE, G., AND RAYMOND, V. 2007. A heuristic for multi-satellite, multi-orbit and multi-user management of Earth observation satellites. *European Journal of Operational Research 177,* 2 (Mar.), 750–762.

BIERWIRTH, C. AND MATTFELD, D. C. 1999. Production scheduling and rescheduling with genetic algorithms. *Evolutionary computation 7,* 1, 1–17.

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*. 243–250.

BOES, J., NIGON, J., VERSTAEVEL, N., GLEIZES, M.-P., AND MIGEON, F. 2015. The Self-Adaptive Context Learning Pattern: Overview and Proposal. In *CONTEXT 2015*. Cyprus.

BONJEAN, N., MEFTEH, W., GLEIZES, M.-P., MAUREL, C., AND MIGEON, F. 2014. Adelfe 2.0. In *Handbook on Agent-Oriented Design Processes*. 19–63.

BONNET, G. 2008. Coopération au sein d'une constellation de satellites. Ph.D. thesis.

BRAX, N. 2013. Self-Adaptive Multi-Agent System for Aided Decision-Making: An Application to Maritime Surveillance. Ph.D. thesis, Université Paul Sabatier, Toulouse, France.

BRÄYSY, O. AND GENDREAU, M. 2005. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science 39,* 1, 104–118.

BURRAFATO, P. AND COSSENTINO, M. 2002. Designing a multi-agent solution for a bookstore with the passi methodology. In *AOIS@ CAiSE*.

CAMPS, V., GLEIZES, M.-P., AND GLIZE, P. 1998. A self-organization process based on cooperation theory for adaptive artificial systems. In *1st International Conference on Philosophy and Computer Science "Processes of evolution in real and Virtual Systems", Krakow, Poland, 02/12/1998-04/12/1998*. Pages de la publication : ..

CAPERA, D. 2005. Systèmes multi-agents adaptatifs pour la résolution de problÃ¨mes application à la conception de mécanismes. Ph.D. thesis, Université de Toulouse, Toulouse, France.

CAPERA, D., GEORGE, J.-P., GLEIZES, M.-P., AND GLIZE, P. 2003. The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In *International Workshop on Theory And Practice of Open Computational Systems (TAPOCS at IEEE WETICE 2003 (TAPOCS - WETICE), Linz, Austria, 09/06/2003-11/06/2003*. IEEE Computer Society, http://www.computer.org, 389–394. Pages de la publication : –.

CENSOR, Y. 1977. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization 4,* 1, 41–59.

CHEN, Y., ZHANG, D., ZHOU, M., AND ZOU, H. 2012. Multi-satellite observation scheduling algorithm based on hybrid genetic particle swarm optimization. In *Advances in Information Technology and Industry Applications*. Springer, 441–448.

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), Venice Italy, 20/10/2008-24/10/2008*. IEEE Computer Society, 107–116.

COSSENTINO, M., GAUD, N., HILAIRE, V., GALLAND, S., AND KOUKAM, A. 2010. Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems 20,* 2, 260–304.

IV

DANTZIG, G. B., ORDEN, A., WOLFE, P., ET AL. 1955. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics 5,* 2, 183–195.

DARWIN, C. 1859. The origin of species. *London: Murray*.

DE WOLF, T. AND HOLVOET, T. 2004. Emergence and self-organisation: a statement of similarities and differences. *Engineering Self-Organising Systems 3464*, 1–15.

DI MARZO SERUGENDO, G., GLEIZES, M.-P., AND KARAGEORGOS, A. 2011. Self-organising systems. Natural Computing Series.

DORIGO, M., BIRATTARI, M., BLUM, C., CLERC, M., STÜTZLE, T., AND WINFIELD, A. 2008. *Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008, Brussels, Belgium, September 22-24, 2008, Proceedings*. Vol. 5217. Springer.

DORIGO, M. AND GAMBARDELLA, L. M. 1997. Ant colonies for the travelling salesman problem. *BioSystems 43,* 2, 73–81.

EIBEN, A. E. AND SMITH, J. E. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

ELLIS, G. 2019. The next generation system worldview-2 for cwrs. Presented as the 15th GeoCAP Annual Conference, Taormina, Italy.

FERBER, J. 1999. *Multi-agent systems: an introduction to distributed artificial intelligence*. Vol. 1. Addison-Wesley Reading.

FRANK, J., ARI, J., MORRIS, R., SMITH, D. E., AND FIELD, M. 2001. Planning and Scheduling for Fleets of Earth Observing Satellites.

FRENSCH, P. A. AND FUNKE, J. 1995. Definitions, traditions, and a general framework for understanding complex problem solving. In *In P. A. Frensch & J. Funke (Eds.), Complex problem solving: The European perspective*. Hillsdale, NJ: Lawrence Erlbaum, 3–25.

FUNKE, J. 2010. Complex problem solving: a case for complex cognition? *Cognitive Processing 11,* 2, 133–142.

GAREY, M. R. AND JOHNSON, D. S. 1979. A guide to the theory of np-completeness. *WH Freemann, New York*.

GEORGÉ, J.-P., GLEIZES, M.-P., AND CAMPS, V. 2011. Cooperation. In *Self-organising Software*, G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, Eds. Natural Computing Series. Springer, http://www.springerlink.com, 193–226.

GLEIZES, M.-P. 2012. Self-adaptive complex systems (regular paper). In *European Workshop on Multi-Agent Systems, Maastricht, The Netherlands*. Vol. 7541. 114–128.

GLIZE, P. 2001. L'adaptation des systèmes à fonctionnalité émergente par auto-organisation coopérative. Ph.D. thesis, Université Paul Sabatier.

IV

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*.

GLOBUS, A., CRAWFORD, J., LOHN, J., AND PRYOR, A. 2004. A comparison of techniques for scheduling earth observing satellites. In *AAAI*. 836–843.

GLOVER, F. AND LAGUNA, M. 2013. *Tabu Search*. Springer.

GRASSET-BOURDEL, R. 2011. Planification dynamique et réactive pour des satellites agiles d'observation de la Terre. Ph.D. thesis, Onera.

HALL, N. G. AND MAGAZINE, M. J. 1994. Maximizing the value of a space mission. *European journal of operational research 78,* 2, 224–241.

HOFFMAN, K. L., PADBERG, M., AND RINALDI, G. 2013. Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science*. Springer, 1573–1578.

INTRILIGATOR, M. D. 1971. *Mathematical optimization and economic theory*. Vol. 39. Siam.

JENNINGS, N. R. 1999. Agent-oriented software engineering. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 4–10.

JIN, Y. AND BRANKE, J. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on evolutionary computation 9,* 3, 303–317.

JORQUERA, T., GEORGÉ, J.-P., GLEIZES, M.-P., AND RÉGIS, C. 2013. A Self-Adaptive Multi-Agent Algorithm for Interactive Continuous Optimization (regular paper). In *International Conference on Computational Collective Intelligence Technologies and Applications (ICCCI), Craiova, Romania, 11/09/2013-13/09/2013*. Number 8083 in LNAI. Springer, http://www.springerlink.com, 437–456.

KADDOUM, E. 2008. Auto-régulation du contrôle manufacturier par système multi-agent auto-organisateur. Ph.D. thesis.

KADDOUM, E. 2011. Optimization under Constraints of Distributed Complex Problems using Cooperative Self-Organization. Ph.D. thesis.

KADDOUM, E. AND GEORGÉ, J.-P. 2012. Collective self-tuning for complex product design (short paper). In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Lyon - France, 10/09/2012-14/09/2012*. CPS (Conference Publishing Services), (electronic medium).

KADDOUM, E., GLEIZES, M.-P., GEORGÉ, J.-P., AND PICARD, G. 2009. Characterizing and evaluating problem solving self-* systems. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD'09. Computation World:*. IEEE, 137–145.

KARABOGA, D. AND BASTURK, B. 2007. Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In *International Fuzzy Systems Association World Congress*. Springer, 789–798.

KENNEDY, J. 2011. Particle swarm optimization. In *Encyclopedia of machine learning*. Springer, 760–766.

KEPLER, J. 1609. *Astronomia nova*. Vol. 1.

KIRKPATRICK, S. 1984. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics 34,* 5-6, 975–986.

KOSAK, O., ANDERS, G., SIEFERT, F., AND REIF, W. 2015. An approach to robust resource allocation in large-scale systems of systems. In *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 1–10.

KRAMER, L. A. AND SMITH, S. F. 2003. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *IJCAI*. 1218–1223.

KRUCHTEN, P. 2004. *The rational unified process: an introduction*. Addison-Wesley Professional.

LADYMAN, J., LAMBERT, J., AND WIESNER, K. 2013. What is a complex system? *European Journal for Philosophy of Science 3,* 1, 33–67.

LEMAITRE, M. AND VERFAILLIE, G. 2006. Tutorial on Planning activities for Earth watching and observation satellites and constellation. *ICAPS*.

LEMAÎTRE, M., VERFAILLIE, G., AND JOUHAUD, F. 2000. How to Manage the New Generation of Agile Earth Observation Satellites ? 2 Scheduling an Agile Earth Observation Satellite. *Proceedings of the International Symposium on AI*, 1–8.

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.

LI, J., CHEN, H., AND JING, N. 2014. A data transmission scheduling algorithm for rapid-response earth-observing operations. *Chinese Journal of Aeronautics 27,* 2, 349–364.

LI, Y., WANG, R., AND XU, M. 2014. Rescheduling of observing spacecraft using fuzzy neural network and ant colony algorithm. *Chinese Journal of Aeronautics 27,* 3, 678–687.

LIN, W.-C. AND LIAO, D.-Y. 2004. A tabu search algorithm for satellite imaging scheduling. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Vol. 2. IEEE, 1601–1606.

LIN, W.-C., LIAO, D.-Y., LIU, C.-Y., AND LEE, Y.-Y. 2005. Daily imaging scheduling of an earth observation satellite. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 35,* 2, 213–223.

IV

LIND, J. 2001. *Iterative software engineering for multiagent systems: the MASSIVE method*. Springer-Verlag.

LIU, S. AND HODGSON, M. E. 2013. Optimizing large area coverage from multiple satellite-sensors. *GIScience & Remote Sensing 50,* 6, 652–666.

MAHESWARAN, R. T., PEARCE, J. P., AND TAMBE, M. 2004. Distributed algorithms for dcop: A graphical-game-based approach. In *ISCA PDCS*. Citeseer, 432–439.

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.

MEIGNAN, D. 2008. Une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques, application aux problèmes d'optimisation de réseaux de transports. Ph.D. thesis, Université de Technologie de Belfort-Montbeliard.

MODI, P. J., SHEN, W.-M., TAMBE, M., AND YOKOO, M. 2005. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence 161,* 1, 149–180.

MORIN, E. 1995. La stratégie de reliance pour l'intelligence de la complexité. *Revue internationale de systémique 9,* 2, 133–165.

NARENDRA, P. M. AND FUKUNAGA, K. 1977. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers 100,* 9, 917–922.

OPPENHEIM, P., PUTNAM, H., ET AL. 1958. *Unity of science as a working hypothesis*. na.

PADBERG, M. AND RINALDI, G. 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review 33,* 1, 60–100.

PAVÓN, J. AND GÓMEZ-SANZ, J. 2003. Agent oriented software engineering with ingenias. In *International Central and Eastern European Conference on Multi-Agent Systems*. Springer, 394–403.

PEMBERTON, J. 2000. Towards scheduling over-constrained remote sensing satellites. In *Proceedings of the 2d International Workshop on Planning and Scheduling for Space*.

PERLES, A., CAMILLERI, G., AND GLEIZES, M.-P. 2017. Self-adaptive distribution system state estimation (regular paper). In *European Workshop on Multi-Agent Systems (EUMAS), Valencia, Spain, 15/12/16-16/12/16*.

PICARD, G., PERSSON, C., BOISSIER, O., AND RAMPARANY, F. 2015. Multi-agent self-organization and reorganization to adapt m2m infrastructures. In *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 91–100.

ROSSI, F., VAN BEEK, P., AND WALSH, T. 2006. *Handbook of constraint programming*. Elsevier.

RUSSELL, S. J., NORVIG, P., CANNY, J. F., MALIK, J. M., AND EDWARDS, D. D. 2003. *Artificial intelligence: a modern approach*. Vol. 2. Prentice hall Upper Saddle River.

SABRY, A., BACHA, A., AND BENHRA, J. 2014. A contribution to solving the traveling salesman problem using ant colony optimization and web mapping platforms application to logistics in a urban context. *Codit'14*.

SAGAN, C. 1994. *Pale blue dot: a vision of the human future in space*. Random House.

SEARLE, J. R. AND TIERCELIN, C. 1995. *La redécouverte de l'esprit*. Gallimard.

SOMA, P., VENKATESWARLU, S., SANTHALAKSHMI, S., BAGCHI, T., AND KUMAR, S. 2004. Multi-satellite scheduling using genetic algorithms. *ISTRAC/ISRO, SpaceOps*.

TALBI, E.-G. 2009. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons.

VASQUEZ, M. AND HAO, J.-K. 2001. A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications 20,* 2, 137–157.

VASQUEZ, M. AND HAO, J.-K. 2003. Upper bounds for the spot 5 daily photograph scheduling problem. *Journal of Combinatorial Optimization 7,* 1, 87–103.

VERSTAEVEL, N. 2016. Thèse de doctorat. Ph.D. thesis, Université de Toulouse, Toulouse, France.

WANG, H., XU, M., WANG, R., AND LI, Y. 2009. Scheduling earth observing satellites with hybrid ant colony optimization algorithm. In *Artificial Intelligence and Computational Intelligence, 2009. AICI'09. International Conference on*. Vol. 2. IEEE, 245–249.

WANG, J., DEMEULEMEESTER, E., AND QIU, D. 2016. A pure proactive scheduling algorithm for multiple earth observation satellites under uncertainties of clouds. *Computers & Operations Research 74*, 1–13.

WANG, J.-M., LI, J.-F., AND TAN, Y.-J. 2007. Study on heuristic algorithm for dynamic scheduling problem of earth observing satellites. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*. Vol. 1. IEEE, 9–14.

WANG, M., DAI, G., AND VASILE, M. 2014. Heuristic scheduling algorithm oriented dynamic tasks for imaging satellites. *Mathematical Problems in Engineering 2014*.

WANG, P., REINELT, G., GAO, P., AND TAN, Y. 2011. A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation. *Computers & Industrial Engineering 61,* 2, 322–335.

WEI, J. 2013. The mission planning model and improved ant colony solving algorithm for networking sar satellites. In *Management Science and Engineering (ICMSE), 2013 International Conference on*. IEEE, 14–19.

WOLFE, W. AND SORENSEN, S. 2000. Three Scheduling Algorithms. *Inform Journal on Management Science 46*.

WOLPERT, D. H. AND MACREADY, W. G. 1997. No Free Lunch Theorems for Optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION 1,* 1, 67–82.

WOOLDRIDGE, M. AND JENNINGS, N. R. 1995. Intelligent agents: Theory and practice. *The knowledge engineering review 10,* 02, 115–152.

WU, G., LIU, J., MA, M., AND QIU, D. 2013. A two-phase scheduling method with the consideration of task clustering for earth observing satellites. *Computers & Operations Research 40,* 7, 1884–1894.

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.

ZAKARIA, Z. AND PETROVIC, S. 2012. Genetic algorithms for match-up rescheduling of the flexible manufacturing systems. *Computers and Industrial Engineering 62,* 2, 670–686.

ZUFFEREY, N., AMSTUTZ, P., AND GIACCARI, P. 2008. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling 11,* 4, 263–277.

IV

List of Figures

List of Tables