17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 517

Automated Planning and Scheduling System for a Heterogeneous Spacecraft Constellation

Rémy Derollez[a]*, Robin Petitdemange[b], Lucas Brémond[c]

[a] *Loft Orbital Technologies, 45 Boulevard de Strasbourg, Toulouse, 31000, France, remy@loftorbital.com*
[b] *Loft Orbital Technologies, 45 Boulevard de Strasbourg, Toulouse, 31000, France, robin@loftorbital.com*
[c] *Loft Orbital Solutions, 321 11th Street, San Francisco, California 94103, USA, lucas@loftorbital.com*
* Corresponding Author

Abstract

Loft Orbital deploys space infrastructure as a service. With the advent of smallsat technology and the fast growth of the space-as-a-service market, this infrastructure aims at supporting a broad range of applications for a variety of customers, involving a multiplicity of spacecraft platforms and payload types.

In order to operate these heterogeneous space systems, while providing high controllability and monitoring capacity to its customers, Loft has developed a fully-automated, web-based mission control system (MCS), called Cockpit. Designed to grant access and control of on-orbit assets to operators as well as customers, Cockpit can be used by end-users to send requests to task their payloads. Predicated on a set of temporal and/or geospatial constraints and service-level agreement (SLA) based policies, these requests can be formulated either programmatically, leveraging Cockpit's application programming interface (API), or interactively, using its graphical user interface (GUI). This entity is thus responsible for coordinating and automating end-to-end mission planning and scheduling, from end-user requests ingestion and processing, schedule management and optimization, to tasking and execution monitoring.

The diversity of space systems being tasked, the complexity of predicting and simulating their state over long time horizons, the flexibility and the deconfliction capability required by an open requesting engine, and the curse of dimensionality of the scheduling problem pose unique mission planning and scheduling challenges.

In order to respond to these challenges, Cockpit's mission management services implement abstractions based on simple and generic requesting and scheduling concepts. This makes for a mission-agnostic simulation, processing and tasking core, combined with mission-specific configurations and resolvers, fostering software scalability and maintainability. In addition, and without compromising with observability of the space systems being tasked, these abstractions promote computational tractability of the scheduling problem — two prerequisites for performing complex simulation, resource allocation and planning optimization.

This paper presents some of the key aspects and tradeoffs of the aforementioned technical challenges and how Cockpit's mission planning system is built to achieve automated planning and scheduling of Loft's heterogeneous constellation.

Keywords: satellite scheduling, heterogeneous constellation, abstraction, automation, optimization.

Acronyms/Abbreviations

ADCS	Attitude Determination and Control System	MCS	Mission Control System
AOI	Area of Interest	MILP	Mixed-Integer Linear Programming
API	Application Programming Interface	SDK	Software Development Kit
GUI	Graphical User Interface	SLA	Service-Level Agreement
LEO	Low-Earth Orbit	YAM	Yet Another Mission

1. Introduction

Loft Orbital is a provider of space infrastructure as a service, accommodating and operating a variety of customer payloads, from remote sensing or communication payloads, to software-only "virtual" payloads, and technology demonstrations. Typical Loft missions can be either rideshare missions with multiple payloads onboard, or dedicated missions, deployed onto either a single or multiple spacecraft. As of March 2023, its fleet consists of three satellites in LEO; YAM-2, YAM-3 and YAM-5, hosting multiple payloads. Twenty-five more satellites are manifested for launch through 2025, including two dedicated subconstellations.

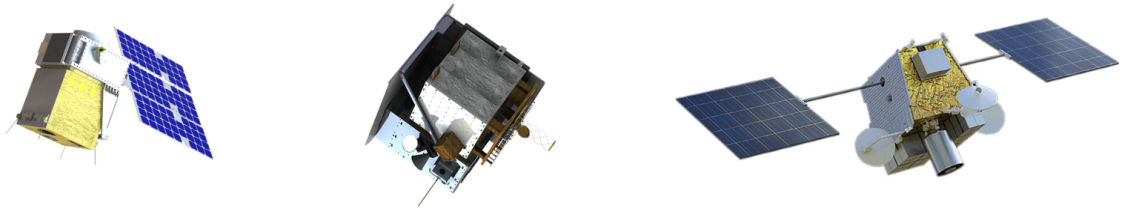

Fig. 1. Various satellite platforms compose Loft Orbital's heterogenous space infrastructure. This illustration includes flying and planned missions.

As the constellation grows, both in number and diversity, standardization and automation of spacecraft operations is critical to the company scaling and ultimately achieving its business goals. Loft is thus continuously developing Cockpit, its platform and payload-agnostic MCS, to orchestrate mission planning and fully automate operations of its heterogeneous constellation [1,2]. With Cockpit, end-users — both internal, i.e., SatDevOps[1] operators, and external, i.e., customers — are granted with high controllability and monitoring capabilities over their on-orbit assets. The MCS is responsible for processing submitted requests, based on a set of temporal and/or geospatial constraints (e.g., AOIs) and/or other geometrical constraints, along with asset-specific settings. Once feasible and safe opportunities are identified, depending on simulation output and SLA considerations, these must be translated into actual operational plans to be executed onboard the space asset(s) involved, while being coordinated with ground segment resources.

Mission management, requesting and scheduling services of Cockpit are in charge of processing these requests, and performing schedules management and optimization. Operational plans are based on tasks, efficiently assigned to relevant sub-systems, with the goal of eventually maximizing data collection throughput. The multiplicity and complexity of the space systems being tasked pose unique resource allocation and scheduling challenges. These space systems are frequently oversubscribed, meaning that the number of requests chronically exceeds the feasible number of jobs that these systems can satisfy [5], which is even more true in the case of rideshare missions hosting multiple payloads, generally competing for common resources. The satellite tasking problem, that is, arranging execution times of a group of tasks given a varied set of request-defined and resource constraints and objectives, is essentially an NP-hard (nondeterministic polynomial-time hard) constraint satisfaction problem [6,7]. Request-defined constraints include limited visibility time windows with the target area, Sun elevation angle, off-nadir angle, or even cloud coverage. Resource constraints comprise energy capacity, finite agility, data storage capacity, communication bandwidth, sub-system availability and task precedence relationships. Objectives fundamentally consist of maximizing data collection throughput, i.e., satisfying the largest possible number of requests.

Cockpit is built with the intention of addressing this problem, in the context of a heterogeneous spacecraft constellation. The MCS is eventually orchestrating end-to-end mission planning, scheduling and operations, from ingestion of end-user requests via its API or its GUI, their processing and the optimized tasking of the infrastructure

[1] SatDevOps designates Loft Orbital's methodology to dispatch satellite operations (SatOps) knowledge and responsibilities across the various engineering teams, rather than having a dedicated SatOps team [3,4].

assets (e.g. spacecraft, ground stations, networks, servers) [1,2]. Its mission management services implement requesting and scheduling abstractions, combined with mission-specific configurations and resolvers, that promote not only software scalability and maintainability, but also tractability of the scheduling problem. Generic requesting concepts help describe this problem and derive potential opportunities. Simple scheduling abstractions and a tasking framework, relying on a constraints-based configuration language, foster their translation into operational scenarios, and eventually the construction of a refined schedule, leveraging optimization techniques.

This paper is organized as follows. Section 2 introduces the satellite scheduling problem in more detail and provides an overview of previous research, as well as some of their applications to state-of-the-art operational mission planning systems. Common optimization techniques, as well as examples of mission planning tools are presented, while highlighting the specificities of the problem in Loft Orbital's case of heterogeneous space systems. In this context, the abstraction approach inherent to Cockpit is defined. Section 3 focuses on Cockpit's requesting and scheduling concepts, its request processing approach, and its schedule generation mechanisms. Some more aspects on simulation, on the enforcement of SLA-based policies, and on the philosophy of combining abstractions and configurations to solve these problems are depicted. Ways of further automation of data collection planning, namely via event-driven requesting, are also discussed. Finally, some conclusions and ways forward are presented in section 4.

2. Background

2.1 The Satellite Scheduling Problem

As listed by Potin [8], the Earth observation satellite tasking problem is an example of an oversubscription scheduling problem that is characterized by a number of constraints:

- Revisit limitations: visibility time windows between a satellite and a given observation target are limited, although predictable.
- Time required to take each image.
- Cloud coverage.
- Energy capacity and thermal control.
- Finite agility: slewing, i.e., transitioning between different look angles, requires non-zero time.
- Pointing angle: the highest resolution images are usually obtained by nadir pointing.
- Limited onboard data storage capacity.
- Ground station availability: visibility time windows are restricted as with any other target, and communication bandwidth is limited.
- Coordination of multiple satellites and/or stereo pair acquisition of the same target (either by multiple sensors, or a single sensor several times).

In some cases, such as the Loft Orbital constellation case to some degree, these constraints are complexified by additional factors, such as the need to balance between the competing objectives of observation and data downlinking [9] — some satellites undertaking only one of the two at a time —, or heterogeneous properties of satellites and sensors [10]. For some subsets of spacecraft, station-keeping and maneuvering constraints might also need to be included.

Arranging execution times of a set of tasks while satisfying the aforementioned constraints is an NP-hard constraints satisfaction problem [6,7]. A large amount of research exists on this problem. Historically, these researches initially focused on single satellite scheduling problems. Lemaître *et al.* [11] investigated a simple sequence-based greedy algorithm, dynamic programming and constraint programming approaches, as well as local search. Heuristic algorithms became effective methods for solving combinatorial optimization problems [5], but with some drawbacks, such as inherent non-determinism (no guarantee of optimality), and difficulty of tuning parameters [11]. Genetic algorithm approaches were tried by Parish [12] for requests ordering and schedule generation based on simple rules, and investigated in many others in the context of satellite scheduling. Globus *et al.* [5,13] compared several stochastic algorithms applied to realistic satellite scheduling problems, including variants of

the genetic algorithm, hill climbing, simulated annealing, squeaky wheel optimization and iterated sampling. They concluded that simulated annealing outperformed other studied techniques.

In following studies, the problem was extended to multiple satellites. It has been approached by heuristic algorithms too, such as genetic or priority-based [14] algorithms. Wang *et al.* [14] presented a nonlinear model of the satellite constellation scheduling problem with a priority-based heuristic. Mixed integer linear programming (MILP) formulations were then provided to overcome limitations of heuristic algorithms, for instance by Monmousseau [15], in combination to a simulated annealing approach, or Kennedy *et al.* [16]. Terra Bella (formerly known as Skybox Imaging and acquired by Planet Labs) used MILP approaches in combination with dynamic programming heuristics for competing imaging scheduling and data downlink scheduling of their constellation [9]. Cho *et al.* [10,17] also used MILP formulations, considering heterogeneous properties of satellites and/or including both energy and data capacity constraints. Liu *et al.* [7] also proposed a MILP model, including task switching time, energy and data capacity constraints, in combination with a heuristic search algorithm based on a symmetric recurrent neural network. Eddy and Kochenderfer [18] first formulated the satellite task scheduling problem as a semi-Markov decision process (semi-MDP). They explored the efficiency of forward search and Monte-Carlo tree search algorithms compared with MILP, graph search and rule-based strategies. These strategies were applied to multi-objectives Earth imaging satellite problems with a single agent, taking into account a wide range of spacecraft tasking types. Later, they used graph theory to further interpret the satellite tasking problem as an undirected sparse graph [19], showing that the problem can be posed as a maximum independent set problem, for which the ReduMIS algorithm can be leveraged in a multi-agent context. This approach showed good results with respect to traditional MILP and seemed well suited for large constellations and large search-space problems. Hadj-Salah *et al.* [20] also leveraged the MDP formulation but focused on reinforcement learning and deep neural networks to find optimal policies that an agent can learn for optimally imaging large areas with a constellation in a highly dynamic environment (e.g., considering cloud coverage). This approach is all-the more interesting that their intent is to target a heterogeneous spacecraft constellation to satisfy optical imaging requests. Finally, Liu *et al.* [21] suggested a non-centralized approach to mission planning by modeling a distributed Earth observation satellite as a multi-agent system. Advantages of game theory solution algorithms applied to these systems include self-organization, scalability and autonomy properties. This approach seems particularly relevant in the case of ever-growing constellations, composed of space systems from different generations, and thus with disparate properties. Adaptive particle swarm optimization and tabu search were applied to a constellation of spacecraft involving different kinds of payloads, such as synthetic aperture radars (SARs) and optical imagers.

2.2 Automated Mission Planning Systems

Automation of mission planning is critical to operating and scaling a heterogeneous spacecraft constellation such as Loft Orbital's.

The 2018 CCSDS informational report on mission planning and scheduling [22] listed some of the main characteristics of fully automated planning systems:

- Operator involvement should be limited to monitoring the planning and execution processes, and, ultimately, only to intervening in case of anomalies.
- Service-oriented architectures allow for the automated interaction between different planning entities.
- Web-based services allow external users to interact with the mission planning system.
- These services must function autonomously and run the planning process, which essentially consists of optimizing the priority of tasks among the pool of received requests, versus resources and constraints.

In the case of missions hosting multiple payloads, such as some rideshare YAMs conducted by Loft, planning is even more complex — resources and constraints should be globally defined, or an explicit part of requests.

Modern mission planning tools naturally tend to be designed in order to follow the aforementioned criteria. As for the latter criterion specifically, some schedule optimization techniques mentioned in the previous section 2.1 may be implemented as part of these systems. Some examples are presented in the rest of this section.

As foreseen by Frank *et al.* [23], it is indeed not viable to task satellites of a fleet individually and focus should be placed on service-oriented requesting systems, leveraging optimization techniques to automate planning and operations. The authors provided an early modelization of the problem of tasking heterogeneous satellites, sometimes with multiple payloads onboard, using a constraints-based approach. They applied a stochastic greedy search algorithm to solve it with the intent to streamline NASA's Earth imaging operations. In the context of the DEIMOS-2 mission, ESA/ESTEC built a tool to generate feasible acquisition sequences [24]. The tool consisted of a mission timelines generator, responsible for analyzing the orbital geometry, finding observation opportunities from a set of user-requested AOIs and building optimized schedules satisfying the constraints, and of a timelines refinement module "re-playing" best-performing timelines with higher fidelity simulation. Generation of feasible schedules follows a greedy strategy, and schedule optimization is performed by means of a genetic algorithm, as described by Globus *et al.* [5]. As part of their research, Cho *et al.* [10,17] built a user request - visibility time window generator tool, taking in a target location and some constellation parameters as inputs, identifying opportunities and eventually returning schedules and associated metrics for each satellite. Scheduling strategy relied on a MILP formulation, including energy and data capacity constraints, or some heterogeneous properties of satellites. Earth observation missions of Airbus Defence and Space (SPOT, Pleiades and Pleiades Neo [25]) have been supported by multiple planning and scheduling techniques over the different generations of spacecraft. Constraints programming and local search methods have been investigated [11]. Operators at CNES and ONERA then discussed on-board decision making capabilities to handle cloud detection, leveraging an additional onboard sensor [26]. Finally, some more recent work contemplated the usage of modern reinforcement learning modeling and deep neural networks to obtain action policies for planning purposes [20]. Bianchessi *et al.* [27] described the constructive deterministic planning and scheduling algorithm for the first generation of the COSMO-SkyMed SAR constellation. This algorithm was capable of handling multiple horizons, taking into account capture and ground segment opportunities, quotas per customer, requesting deadlines, payload operational profiles and data storage constraints, while required to run under a given time threshold. The implementation constraints led to a sequential decision sequence strategy allowing back-tracking. Terra Bella developed automated scheduling tooling for their constellation, allowing operator interaction [9]. Optimal scheduling relied on a MILP approach, enhanced with dynamic programming heuristic to better balance competing objectives of data collection and downlinking. Capella Space relied on a batch-optimized task scheduling formulation for their constellation of SAR satellites [28]. Planet Labs also addressed scheduling of its Dove cubesat fleet by dichotomizing data collection and downlinking [29]. They split the problem into two sequential parts, first solving the ground contact selection problem using a series of MILP, before providing the solution as an input to the imaging allocation problem, solved using a grid optimization algorithm.

While interoperable interfaces of automated mission planning systems are usually still "defined on a per-mission basis" [22], Loft Orbital's MCS, Cockpit, aims at automating mission planning and operations of heterogeneous space infrastructure at scale.

2.3 Cockpit

2.3.1 Overview

Cockpit is Loft Orbital's MCS. It is built in order to reduce the complexity of spacecraft operations, while providing high controllability and monitoring capabilities to its end-users:

- It is designed for full automation of operations, limiting operator involvement to mere monitoring or off-nominal interventions, thus allowing Loft's fleet to scale up.
- It is leveraging a microservice architecture [30], structuring a collection of loosely coupled, independently deployable services interacting with each other.
- It is cloud-based. End-users are able to interact with Cockpit either programmatically, via its API, or interactively, via its GUI exposing a collection of views, mapping API calls.
- Its mission management, requesting and scheduling services are dedicated to autonomously run the planning process.

2.3.2 Abstraction Approach

Loft's space infrastructure is meant to support a broad range of applications: it is thus built upon a stack of heterogeneous systems — satellites buses from multiple space systems vendors, hosting a multiplicity of payloads of various kinds, in combination with ground sites potentially from multiple ground segment networks. This is why Cockpit implements space and ground segment abstractions to expose a universal set of interfaces and help unify heterogeneous systems.

For mission planning in particular, abstractions may also promote tractability of the scheduling problem. Cockpit's mission management, requesting and scheduling services model simple requesting concepts that help describe the problem constraints and objectives, and simple, granular tasking concepts to translate derived opportunities into operational plans that may be optimized. Section 3 describes some of the key aspects of the requesting and scheduling concepts and of schedule generation and schedule optimization mechanisms in Cockpit.

Additionally, these core abstractions, used in combination with mission-specific configurations and/or resolvers, also largely promote code scalability and maintainability.

Agreed-upon definitions and semantics are an important part of the standardization and democratization process for automated planning and scheduling [22]. By fulfilling criteria mentioned in the CCSDS green book and applying them operationally and successfully to a fleet of satellites providing a various range of applications, Loft Orbital:

- shows its compliance and agreement with those suggested standardized concepts,
- suggests one way to implement them for further automation and scalability,
- and demonstrates its applicability to operating a constellation of heterogeneous spacecraft, eventually enabling space infrastructure as a service.

3. Automated Planning and Scheduling in Cockpit

3.1 Overview

Mission management, requesting, scheduling services form a subset of microservices in Cockpit dedicated to mission planning and scheduling.

These are working in tight collaboration with other services of Cockpit, including:

- flight dynamics services, providing orbital geometry data — enabling the use of geospatial constraints as part of requests —, or presenting maneuvering plans for constellation geometry management or collision avoidance,
- simulation services, providing system-level data for safe and proper resource allocation,
- a weather forecast service, enabling the use of cloud coverage constraints as part of requests.

Cockpit services are essentially federated by GraphQL APIs [2,31]. Because of this kind of heavy interconnection, a graph data structure is particularly relevant. Section 3.2.2 presents requesting interfaces in more detail.

3.2 Requesting

Requesting is the primary mechanism that end-users leverage to task their on-orbit assets. Cockpit's requesting system typically implements goal-based planning, utilizing objective-oriented requests, as opposed to activity-based planning. In goal-based planning, an objective (rather than a predefined activity) is given as input to the planning system, which is then left to come up with defining activities to achieve an objective [22].

3.2.1 Requesting Concepts

3.2.1.1 Requests

Requests in Cockpit are thus based on a constraint-based formulation, which can be grouped into the following categories: *what*, *where*, *when*, and *how*.

- *What* defines the targeted operational asset. In Loft Orbital's case, it can be of different kinds: a physical payload (e.g., a thermal infrared imager), a "virtual" one (e.g., an algorithm running on onboard hardware), or a "swarm" of several identical payloads to be orchestrated, potentially across multiple satellites.

- *Where* defines geospatial constraints: a certain ground target, or a set of AOIs.
- *When* defines temporal constraints: around a specific instant, or within a given time interval, and/or a range of local times.
- *How* can be different additional constraints placed on top: acceptable ranges of Sun elevation, off-nadir angle, forecasted cloud coverage, …

This formulation of these request constraints is agnostic to the target asset, following Cockpit's abstraction approach. Additionally, these constraints may always be completed by asset-specific settings, provided via unstructured interfaces and/or attached files, predicating this asset's concept of operations (e.g., exposure time settings for an imager, parameters for an algorithm and/or a script to be uplinked, *etc*.).

Some examples of natural language equivalents of requests could then be:

- Take a picture using *My Thermal Infrared Imager* over *Dubai* sometime *around March 10th, 2023*, only during *daytime*, if the expected *cloud coverage is less than 20%,* using given exposure time and gain.
- Activate *My Processing Payload* whenever flying over *metropolitan France*, using the attached file.
- Collect RF samples using *My SDR Swarm* over *California* either *on Tuesday or on Friday*, only within an *elevation angle greater than 40 degrees*.
- *etc*.

End-users can also assign a priority level to their request, which will be leveraged for global deconfliction. In certain use cases, priorities can also be assigned on a per-AOI basis within a single request, to prioritize the coverage of a certain area over other ones.

3.2.1.2 Operational Scenarios

Once a request is submitted to Cockpit's requesting system, leveraging interfaces described in the following section 3.2.2, it is analyzed and processed as presented in 3.2.3, in order to identify feasible and safe operational scenarios according to simulation (section 3.4), and falling under SLAs (section 3.5).

These abstractions are the basis of the planning and orchestration to satisfy the associated request. In the case of swarms, the execution of a given request may be fragmented and dispatched onto multiple satellites (as exemplified in the third example above). Naturally, only feasible and safe operational scenarios are exposed, so that users cannot interfere with each other (e.g., for missions deployed on the same satellite, or generally speaking, competing for shared resources).

Operational scenarios returned to the user may be ranked using different metrics. Exposing multiple options balancing request input (e.g., time to completion versus mean cloud coverage, mean off-nadir angle versus mean Sun elevation angle, …) lets the user explore Pareto fronts, as illustrated by figure 2, and decide which constraint is paramount. This allows to increase flexibility in scheduling and to keep the system generally agnostic to selection criteria.

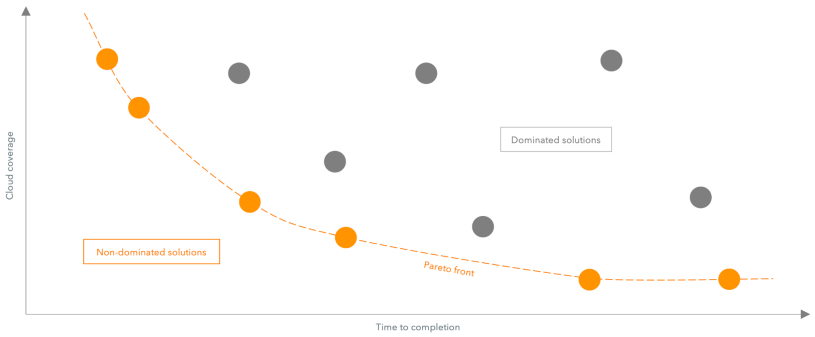

Fig. 2. Pareto front visualization for operational scenarios.

While the next paragraph focuses on interfaces for submitting and monitoring requests, their processing and the generation of operational scenarios is presented in greater detail in the following sections.

3.2.2 Requesting Interfaces

Cockpit exposes two types of interfaces: APIs, for programmatic access or machine-to-machine communication, and a GUI providing a set of views sitting on top of API-exposed features.

3.2.2.1 Programmatic Interface

Cockpit services, including mission management services, expose GraphQL APIs [31]. Interactions with the requesting API can be turned into native Python, leveraging a Python software development kit (SDK) developed by Loft. Figure 3 presents an example snippet of code for request submission via the API, leveraging the Python SDK. The provided request example follows the first natural language instance from section 3.2.1.1.

Python

```python
from cockpit.asset import Payload
from cockpit.mission import Request


# Create request
request = Request.create(
    target=Request.Target(
        type=Request.Target.Type.Payload,
        destination=Payload.get(name="My Thermal Infrared Imager")
    ),
    settings=Request.Settings(
        target_settings={"exposure_time": 1000, "gain": 4},  # Payload-specific settings
    ),
    constraints=Request.Constraints(
        spatial=Request.Constraints.Spatial(
            type=Request.Constraints.Spatial.Type.Point,
            set="SRID=4326;POINT Z (55.2962 25.2770 0.0)",  # Spatial constraint: over Dubai
        ),
        temporal=Request.Constraints.Temporal(
            type=Request.Constraints.Temporal.Type.NearInstant,
            set="2023-03-10T00:00:00Z",  # Temporal constraint: around specific instant
        ),
        additional=Request.Constraints.Additional(
            sun_elevation_angular_range=(0.0, 90.0),  # Sun elevation constraint: daylight
            cloud_coverage_range=(0.0, 20.0),  # Cloud coverage constraint: less than 20%
        ),
    ),
    comments="Collecting thermal infrared data over Dubai.",
)

# Submit request for processing
await request.process()

# Select and confirm first available operational scenario
await request.confirm(request.operational_scenarios.first())
```

Fig. 3. Example snippet of code for request submission via Cockpit's API, leveraging a Python SDK mapping GraphQL API calls to native Python.

3.2.2.2 Graphical Interface

The Cockpit GUI ("web app") displays a collection of views that can be used to perform many of the same operations which are available via the API. It is designed for both kinds of end-users: customers as well as

SatDevOps users, with varying degrees of access privileges, so that high-level requesting and fine-grained control, respectively, can co-exist in the same application.

Submitting and monitoring requests and schedules is one of the web app's main flows. Both generic forms and views, allowing to specify request constraints, and asset-specific forms depending on the target being requested, are served.

Once a request is submitted and processed, various metrics are exposed for comparison of the different proposed operational scenarios. Requested AOIs, corresponding sensor projections and coverage distribution in terms of sensors and satellites can then be tracked via different views, as illustrated by figure 4. As far as the user's access privileges allow (i.e., limited to the asset(s) they have authorization for), the schedule and tasks can be thoroughly introspected via Gantt charts and ground track projections, as shown by figure 7.

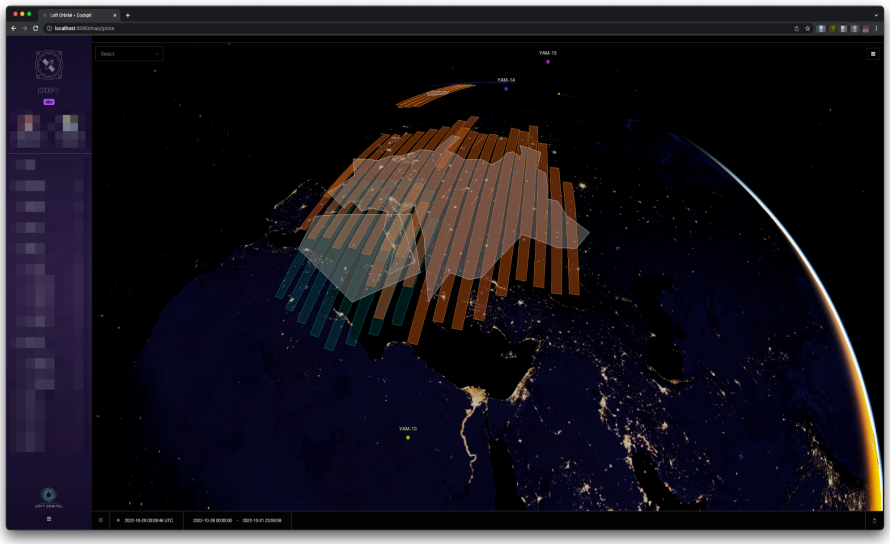

Fig. 4. Requested AOIs and sensor projections displayed in Cockpit GUI.

3.2.3 Request Processing

3.2.3.1 Request Processing Flow

Cockpit mission management and requesting services handle the processing of requests. Processing load may be dispatched to multiple workers if need be.

Multiple layers of analysis are performed to identify opportunities, based on different input and constraints. The output of request processing is a set of operational scenarios from which the end-user can choose the one that best fits their need. Request processing and scheduling essentially consists of the following high-level steps:

- Spatial and temporal constraints, along with optional additional constraints (e.g., acceptable ranges of Sun elevation, off-nadir angle, forecasted cloud coverage, …) and asset-specific settings are evaluated to derive time windows of visibility with the target point or AOIs, from the simulated orbital geometry and attitude of involved satellites.
- For each window, navigation profiles, coverage and other metrics are computed, and an initial set of tasks is derived, forming an operational scenario.
- Operational scenarios are confronted to the existing schedule for deconfliction with already committed activities, and simulated to check for safety, ensuring that hardware constraints are respected.
- Operational scenarios are refined for optimality of the global schedule.

At various levels in the requesting and scheduling flow, policies are enforced to assess whether a given request and/or a resulting operational scenario is acceptable from an SLA standpoint, and prune the ones that are not.

Note that the aforementioned steps are not executed in a linear manner. Instead, the flow is more similar to a graph. Various layers, especially when it comes to policies, simulation and/or schedule optimization (detailed in the next sections) may be applied iteratively to check the global feasibility, safety, and optimality of the scenarios.

In this context, the need for efficient fundamental routines (namely, around numerical propagation) rapidly arises. Cockpit mission management, requesting, scheduling and simulation services make use of the Open Space Toolkit open-source C++ libraries [32].

Fig. 5. High-level schematic of various steps in the requesting and scheduling flow.

3.2.3.2 Request Lifecycle

Depending on the outcome of request processing, a request may either be rejected straightaway, or accepted if at least one feasible scenario is available. In a second time, upon selection of an operational scenario and confirmation of the request by the end-user, this scenario is reprocessed by the scheduling and simulation services. This is to prevent any race condition from occurring; the schedule may have been altered since the first processing step, and adding the selected scenario as-is would potentially be introducing either a conflict, or suboptimal resource allocation. If the scenario can still be handled, the request can be confirmed.

The request is effectively scheduled once, for each of the involved satellites, their individual schedule has been successfully translated into commands and uplinked for execution. The request can then either be deemed completed, if onboard execution was successful, based on telemetry and feedback, or failed, in off-nominal situations where its execution would have been unsuccessful. Nominally, a request's end of life corresponds to the delivery of the associated data to the end-user.

Cockpit API and GUI offer great observability into the request lifecycle. They allow to monitor statuses of requests, introspect their associated operational scenarios, and access output data processing and delivery items.

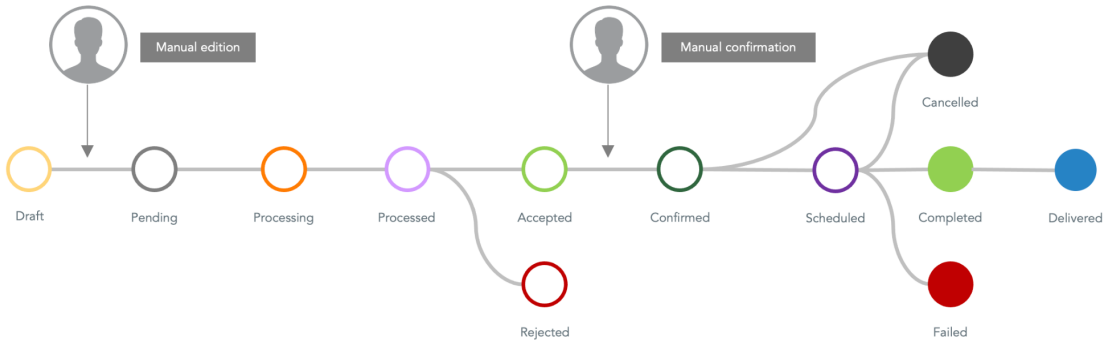

Fig. 6. Request lifecycle, from submission to data delivery.

3.3 Scheduling

3.3.1 Scheduling Concepts

In Cockpit, simple tasking abstractions are used as building blocks of operational scenarios and spacecraft schedules.

3.3.1.1 Tasks

An atomic action within the satellite schedule is modeled by a task to be executed by a given target sub-system. As such, tasks represent the allocation of a given resource — either exclusively accessed (e.g., ADCS system) or shareable (e.g., onboard storage) —, for an estimated time span, for a given activity. In other words, tasks, placed on "swimlanes" in the schedule, indicate business (or partial business) of the associated sub-system. Some tasks may then be translated into a set of actual commands to be executed onboard. Tasks can be grouped into logical sets achieving a common goal, modeling as such high-level precedence or side-effect relationships between constituting tasks.

3.3.1.2 Sub-Schedules

Multiple tasks together form a portion of schedule, which may or may not be selected for execution eventually. In Cockpit, those are referred to as sub-schedules. Software version control systems such as Git are built to support non-linear development: branching allows to add commits to parallel branches to the main line of development, and branches may then be merged onto others [33]. Cockpit's tasking and scheduling system is inspired from this flow: task sets can be added to a sub-schedule that is parallel to the "main" version of the spacecraft schedule, and may or may not be merged onto it, i.e., actually selected for execution. Symmetrically, sub-schedules may be reverted from the main schedule.

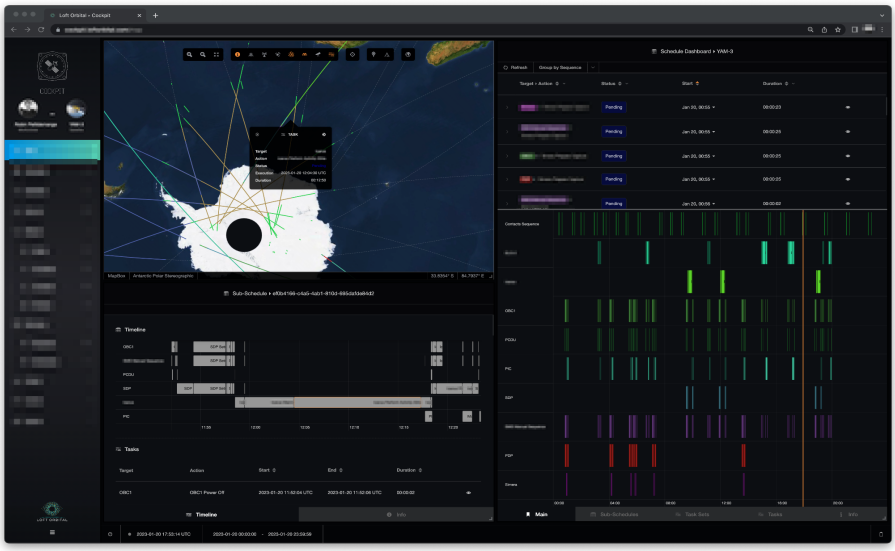

Fig. 7. Sub-schedules and tasks displayed in Cockpit GUI.

As such, sub-schedules are, for example, constitutive of operational scenarios resulting from requests. When an operational scenario is selected for execution, the underlying sub-schedule is merged onto the main schedule, for each of the involved spacecraft. Should the associated request be canceled, those sub-schedules would be reverted.

Besides requests, other flows, described in more detail as part of section 3.7, may trigger the generation of sub-schedules and their merging onto the main schedule:

- either automated, e.g., downlinking and/or maintenance tasks, maneuvering tasks, *etc*.,

- or manual, i.e., ad-hoc operator-induced tasking, namely during early orbit and commissioning phases, or in case of anomalies.

3.3.2 Sub-Schedule Generation

Concepts of operations are generally consistent and predictable to some extent. As a result, repetitive patterns of tasks can generally be identified for most activities. For instance:

- Executing a request for an imager will, in most cases, involve powering on and setting up that imager, slewing to and/or tracking the target, taking the image, and going back to an idle state after forwarding collected data to a processing unit.
- Ground station passes generally involve setting up communication systems while slewing to track a ground station antenna, receiving/sending data, before returning to a nominal state for attitude and communication systems.
- *etc.*

As such, it is possible to use a template-based approach to yield first-order approximation sub-schedules. Those can be refined in a second time, while being confronted against the main schedule for potential merging:

- trying to improve optimality, by rearranging tasks,
- via simulation, checking for feasibility and safety of that schedule branch.

Those two topics are the objects of the following subsections.

Sub-schedule templates essentially define reusable, semi-dynamic skeletons of tasks. While their general arrangement is fixed, they are parameterized by temporal input (fixing task execution times, at first-order) and/or by any kind of additional input required to fully resolve the template into a concrete sub-schedule. This input is generally the result of computation (e.g., outcome of request processing, ground station visibility or maneuvering analysis), or may be manually provided by an operator in ad-hoc situations.

```
Unset
_adcsTrackingTaskSet: #TaskSet & {
        tasks: [
        {
                action:   _targets.obc.actions.adcs_tracking
                timeSpan: #TimeSpan & {
                        start:    _previousTaskSet.tasks[0].timeSpan.end
                        duration: 1.0 // second
                }
                parameters: target: _analysis.adcs_tracking.target
        },
        {
                action:   _targets.adcs.actions.tracking
                timeSpan: #TimeSpan & {
                        start:    _adcsTrackingTaskSet.tasks[0].timeSpan.end
                        duration: _analysis.adcs_tracking.duration
                }
        },
}
```

Fig. 8. Illustration of templating tasks using CUE [34].

Such templates may be conveniently written using a constraints-based configuration language such as CUE [34]. CUE is an open-source spin-off of GCL, the General Configuration Language developed by Google. One of the key principles of CUE is that complex constraints can easily be defined by merging types and values into a single construct. A typical field declaration in CUE represents a set of nodes to which apply a constraint acting as data validator. Because declaration order does not matter, this encourages a clear separation of concerns between

computation and configuration, e.g., by keeping data that needs to be computed outside of CUE and injecting it to be mixed in. These principles proved to be particularly relevant for our application, where temporal and general input data is generated externally and injected into a template while being validated against underlying constraints. A practical example is provided by figure 8.

3.3.3 Schedule Optimization

Confronted to the main schedule for potential merging, first-order approximation sub-schedules generated from templates may undergo a refinement process. Ultimately, the branch stemming from the main schedule onto which that sub-schedule would be merged should be optimized. While development of related capabilities in Cockpit is on-going work, various approaches seem worth considering for this purpose, which essentially consist of optimally rearranging this branch of schedule, starting from this first-order approximation as an initial feasible solution. The state-of-the-art presented in section 2.1 suggests that combinations of MILP formulations and heuristic approaches are worth considering.

Fundamentally, the optimization process runs every time an operation is conducted on the main schedule, either a sub-schedule is merged onto it, or reverted. Because of that symmetry, it is interesting to track unitary optimization operations (rearrangements) that have been previously performed at sub-schedule level. This also promotes observability into the optimization process.

3.4 Simulation

In order to assess feasibility, safety and optimality of the schedule, a certain degree of modelization of the spacecraft must be considered, in order to simulate, for instance, ADCS maneuvers, power and data states, and various subsystem states.

Simulation is required to identify the side-effects that each task would impart; typically, what is the subsystem involved in the execution of this task, for how long it will be busy (or partially busy), and/or what percentage of the available resources will be used. In addition, simulation must ensure that all the sub-schedules on the main schedule are compatible with one another at the hardware level (e.g., can those two ADCS maneuvers required by those two concurrent activities onboard be performed safely and efficiently?).

Exposed operational scenarios are guaranteed to satisfy those criteria of feasibility and safety by the simulator, whose role is to “play” a branch of the schedule with the candidate operational scenario, and to prune it if violating resource constraints.

3.5 Policies

Throughout the requesting and scheduling flow, policies are enforced to ensure that requests and/or resulting operational scenarios fall under mission SLAs, agreed upon between the constellation operator, Loft Orbital, and the payload or asset owner. Mission SLAs and thus policies can take many forms:

- Requests may be restricted to specific regions of the globe: policies must then be enforced to reject prohibited requests (e.g., in some cases, requests over the South Atlantic Anomaly).
- Requests and operational scenarios may be subject to deadlines or lead time limitations: policies prevent requests from being processed and/or confirmed with an operational scenario after a given deadline.
- Requests may be limited to a given number, a given amount of collected data, or a given amount of operational time, over some rolling time period.
- *etc*.

Given the complexity and highly dynamic nature of the requesting and scheduling flow, and the diversity of policies, they may be evaluated at various stages in the process.

3.6 Configuring and Resolving Abstractions

As the reader can appreciate, while abstractions can greatly help approach the requesting and scheduling problem, some payload-, platform- and/or mission-specific considerations remain. This is the case for instance of the

definition and processing of the asset-specific settings on a request, the proper simulation of subsystems with intrinsic properties, the translation of their concept of operations into tasks, or any SLA considerations (policies).

This ambivalence between an abstracted core flow and specific configurations (or resolvers) is inherent to the satellite tasking problem applied to heterogeneous space systems. Loft's strategy is to implement abstractions and a general-purpose request processing, scheduling and simulation flow, which is fed with configuration and used in combination with mission-specific resolvers (e.g., sub-schedule templates described in section 3.3.2).

3.7 A Scheduling Ecosystem

As previously mentioned, the presented tasking abstractions and frameworks are not limited to supporting the requesting flow, but rather at the basis of a scheduling ecosystem. Other flows, either automated, or manual, may trigger the generation of sub-schedules and writing on the main schedule.

This is the case, for instance, of downlinking tasks to operate ground station contacts, scheduled in an automated fashion. Eventually, various maintenance plans, such as maneuvers, station-keeping or collision avoidance tasks, can be derived and automatically scheduled, with limited to none operator interaction. These sub-schedules may or may not take precedence over any pre-existing tasks on schedule, depending on nature and priority.

In any case, manual tasking and overwriting by SatDevOps operators is always a possibility, to support any ad-hoc activities such as commissioning phases, anomalies, or specific maneuvering. Operator-induced tasking leverages the same ecosystem of concepts, via procedures, calling the Cockpit API and written using its Python SDK, and wrappers around the previously described sub-schedule generation CUE framework.

3.8 Further Automation: Event-Driven Requesting

In the previous sections, requesting was presented through the prism of user-induced, imperative goal-based planning. In this paradigm, requesting is initiated by an end-user, who must be intentional about formulating and submitting a request at a given time, and who must manually define the underlying goal through constraints. While the presented abstractions and interfaces largely facilitate this process and enable the automated and optimized scheduling of resources, the end-user is still involved in the decision process, assessing the proposed operational scenarios and eventually selecting one of them for execution.

Cockpit mission management services implement supplemental abstractions to further streamline this flow and set up event-driven requesting. Request pipelines can be created to autonomously handle the constraints definition, request submission and operational scenario selection on behalf of the user. An external event source may trigger a pipeline resolver that will autonomously populate constraints and submit a request for processing. In a second time, once the request is processed, another resolver will rank operational scenarios and pick the most relevant one according to heuristics defined as part of the pipeline. For example, a pipeline could be defined with a resolver deriving events such as natural disasters (e.g., wildfires) from news websites, social media and/or various APIs, and autonomously submit and confirm requests.

This paradigm may be used to further increase data collection autonomy and reduce latency, by providing an end-to-end collection plan with no human decision-maker in the loop.

4. Conclusion

Automating the planning and scheduling of operations for a heterogeneous spacecraft fleet such as Loft Orbital's poses some unprecedented challenges. Loft's infrastructure aims at supporting various missions, either rideshare or dedicated ones, hosting a diversity of physical or "virtual" payloads, which generally compete for onboard resources. While existing mission planning and control systems have been addressing some of these challenges, Loft Orbital's case calls for a different kind of answer. Cockpit, Loft's fully-automated, web-based MCS, approaches the heterogeneous satellite tasking problem by embracing abstractions to simplify requesting of diverse data collections and to promote the tractability of the resulting resource allocation and schedule optimization problem.

Building upon fundamental principles from the CCSDS standard, Cockpit promotes a mission-agnostic approach to planning and scheduling and lays the foundations of a rationalized space infrastructure as a service, by eventually enabling end-to-end automated data collection planning for a multiplicity of space applications.

As the infrastructure grows and as new ways of interacting with this infrastructure emerge, additional questions arise and use cases appear. Some of them may challenge the use of fully centralized mission planning instances and bring into question what the right balance between on-board and on-ground decision making capabilities should be. This topic is part of ongoing work at Loft Orbital as both ground systems and onboard interfaces continuously evolve to fit the needs of end-users. Finally, for Loft Orbital, the need for scalability exceeds mission planning and operations; mission analysis and design phases would also greatly benefit from automation and abstraction. By leveraging common tools — with varying degrees of modeling fidelities —, rationalization and simplification could be brought into mission development phases, eventually further reducing access time to Loft's space infrastructure for end-users.

References

[1] P.-D. Vaujour and L. Brémond, System and Method for Providing Spacecraft-Based Services, U.S. Patent No. 10,981,678, Washington, DC: U.S. Patent and Trademark Office, 2019.

[2] L. Brémond, B. Poon and G. Damien, Future-Proof Mission Control Systems: Leveraging Agnostic Design for Autonomous and Event-Driven Satellite Operations, 73rd International Astronautical Congress, Paris, France, 2022, 18–22 September.

[3] B. Poon, L. Brémond, C. MacLachlan and L. Stepan, SatDevOps: A Novel Automated Satellite Operations Methodology, SpaceOps 2023, Dubai, United Arab Emirates, 2023, 6–10 March.

[4] Loft Orbital Solutions Inc., SatDevOps™, 2023.

[5] A. Globus, J. Crawford, J. Lohn and A. Pryor, A Comparison of Techniques for Scheduling Earth Observing Satellites, Proceedings of the 16th Conference on Innovative Applications of Artificial Intelligence, 2004.

[6] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Macmillan Higher Education, 1979.

[7] S. Liu and J. Yang, A Satellite Task Planning Algorithm Based on a Symmetric Recurrent Neural Network. Symmetry, 2019; 11(11):1373. https://doi.org/10.3390/sym11111373.

[8] P. Potin, End-to-End Planning Approach for Earth Observation Mission Exploitation, SpaceOps 1998, Tokyo, Japan, 1998, 1–5 June.

[9] S. Augenstein, A. Estanislao, E. Guere and S. Blaes, Optimal Scheduling of a Constellation of Earth-Imaging Satellites, for Maximal Data Throughput and Efficient Human Management, Proceedings of the International Conference on Automated Planning and Scheduling, 2016.

[10] D.-H. Cho, H.-Y. Kim and H.-L. Choi, Optimal Continuous-Time Job Scheduling for Multiple Low Earth Orbit Satellites, 2016; 10.2514/6.2016-2107.

[11] M. Lemaître, G. Verfaillie, F. Jouhaud, J.-M. Lachiver and N. Bataille, Selecting and Scheduling Observations of Agile Satellites, Aerospace Science and Technology, Volume 6, Issue 5, 2002, pp. 367–381, ISSN 1270-9638, https://doi.org/10.1016/S1270-9638(02)01173-2.

[12] D. A. Parish, A Genetic Algorithm Approach to Automating Satellite Range Scheduling, Air Force Institute of Technology Rept. AFIT/GOR/ENS/94M-10, Wright-Patterson AFB OH, 1994.

[13] A. Globus, J. Crawford, J. Lohn and A. Pryor, Scheduling Earth Observing Satellites with Evolutionary Algorithms, Proceedings of the International Conference on Space Mission Challenges for Information Technology, 2003.

[14] P. Wang, G. Reinelt, P. Gao and Y. Tan, A Model, a Heuristic and a Decision Support System to Solve the Scheduling Problem of an Earth Observing Satellite Constellation, Computers & Industrial Engineering, Volume 61, Issue 2, 2011, pp. 322–335, ISSN 0360-8352, https://doi.org/10.1016/j.cie.2011.02.015.

[15] P. Monmousseau, Scheduling of a Constellation of Satellites: Improving a Simulated Annealing Model by Creating a Mixed-Integer Linear Model, KTH Royal Institute of Technology, Stockholm, Sweden, 2015.
[16] A.K. Kennedy and K.L. Cahoy, Performance Analysis of Algorithms for Coordination of Earth Observation by CubeSat Constellations, Journal of Aerospace Information Systems, Vol. 14, No. 8, 2017, pp. 451–471.
[17] D.-H. Cho, H.-Y. Kim and H.-L. Choi, Optimization-Based Scheduling Method for Agile Earth-Observing Satellite Constellation, Journal of Aerospace Information Systems, Vol. 15, No. 11, 2018, 10.2514/1.I010620.
[18] D. Eddy and M. J. Kochenderfer, Markov Decision Processes for Multi-Objective Satellite Task Planning, IEEE Aerospace Conference, 2020.
[19] D. Eddy and M. J. Kochenderfer, A Maximum Independent Set Method for Scheduling Earth Observing Satellite Constellations," CoRR, vol. abs/2008.08446, 2020, https://arxiv.org/abs/2008.08446.
[20] A. Hadj-Salah, R. Verdier, C. Caron, M. Picard and M. Capelle, Schedule Earth Observation Satellites with Deep Reinforcement Learning, 2019, https://arxiv.org/abs/1911.05696.
[21] L. Liu, Z. Dong, H. Su and D. Yu, A Study of Distributed Earth Observation Satellites Mission Scheduling Method Based on Game-Negotiation Mechanism. Sensors, 2021.
[22] The Consultative Committee for Space Data Systems (CCSDS), Report Concerning Space Data System Standard – Mission Planning and Scheduling, Informational Report CCSDS 529.0-G-1, Green Book, 2018.
[23] J. Frank, A. Jonsson, R. Morris and D. E. Smith, Planning and Scheduling for Fleets of Earth Observing Satellites, Proceedings of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space, 2001.
[24] S. Tonetti, S. Cornara, A. Heritier and F. Pirondini, Fully Automated Mission Planning and Capacity Analysis Tool for the DEIMOS-2 Agile Satellite, Workshop on Simulation for European Space Programmes (SESP), ESA-ESTEC, Noordwijk, The Netherlands, 2015, 24–26 March.
[25] Airbus, 2022. OneAtlas Data - Tasking, https://api.oneatlas.airbus.com/guides/oneatlas-data/g-tasking.
[26] G. Beaumet, G. Verfaillie and M.-C. Charmeau, Autonomous Planning for an Agile Earth-Observing Satellite, Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), 2008.
[27] N. Bianchessi and G. Righini, Planning and Scheduling Algorithms for the COSMO-SkyMed Constellation, Aerospace Science and Technology, Volume 12, Issue 7, 2008, pp. 535–544, ISSN 1270-9638, https://doi.org/10.1016/j.ast.2008.01.001.
[28] Capella Space, 2022. Tasking API, https://docs.capellaspace.com/api/tasking.
[29] V. Shah, V. Vittaldev, L. Stepan and C. Foster, Scheduling the World's Largest Earth-Observing Fleet of Medium-Resolution Imaging Satellites, Proceedings of the 11th International Workshop on Planning and Scheduling for Space, 2019.
[30] C. Richardson, Microservices, 2019. Microservice Architecture Patterns, https://microservices.io/patterns/microservices.html.
[31] The GraphQL Foundation, 2023. GraphQL, https://graphql.org.
[32] Open Space Collective, 2023. Open Space Toolkit, https://github.com/open-space-collective.
[33] S. Chacon and B. Straub, Pro Git (2nd edition), Apress, USA, 2014.
[34] CUE Language, 2023. About CUE, https://cuelang.org/docs/about.