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

SpaceOps-2023, ID # 263

ADAPTING GROUND STATION NETWORKS FOR AUTOMATIC M2M OPERATIONS

Arne Nylund[a]*, Torkil Rein Gustavsen[b]

[a] *Kongsberg Satellite Services AS, Prestvannveien 38, 9011 Tromsø, Norway*, mailto:arne@ksat.no
[b] *Kongsberg Satellite Services AS, Prestvannveien 38, 9011 Tromsø, Norway*, mailto:torkilrg@ksat.no
* Corresponding Author

Abstract

As more satellite missions and constellations emerge, operations based on Machine-to-Machine interfaces (M2M) has become increasingly important to satellite operators to exploit the full capabilities of the satellites and to utilize existing Ground Stations (GS) for optimal coverage.

Traditionally, the interaction between satellite owners and GS providers has been semi-synchronous, i.e., exchanging files (typically XML) with possible long delays between request and response. This has made it challenging to support a more dynamic world with changing needs.

Therefore, a demand for well-defined interfaces that allow for faster turnaround time, which are reliable, responsive, and easy to implement has emerged. Providing lean interfaces that are easy to implement and leveraging open standards can be of great benefit to the industry, enabling new ways to operate and do business. This paper explores how ground networks can be implemented as REpresentational State Transfer (REST) Application Programming Interface (API) services described in OpenAPI-specifications, which benefits these yields for ground station providers and satellite operators, and the trade-offs involved.

Although REST services are well suited for planning and reporting purposes, an automated M2M interface between satellite owners and GS providers also must support passing information of a more asynchronous nature (e.g., unpredictable, or unplanned events that need the other party's immediate attention), in addition to providing streaming capabilities (e.g., performance metrics during contacts). One answer to this is using the Advanced Messaging Queuing Protocol (AMQP). AMQP provides a two-way secure and persistent architecture, which is flexible and easy to implement.

This paper will address how M2M interface can be implemented from a GS perspective, along with deployment considerations for supporting multiple satellite owners utilizing the same basic services in a consistent and reliable manner. The paper will also demonstrate how this can be used between satellite owners and GS providers, fulfilling the needs of all parties for fully automated operations, removing the need for manual interaction. Finally, the paper will address lessons learned in implementing this interface in the KSAT Lite global ground station network.

Keywords: Automation, API, M2M, REST, AMQP

Acronyms/Abbreviations

Acquisition Of Signal (AOS)
Advanced Message Queuing Protocol (AMQP)
Application Programming Interface (API)
Consultative Committee for Space Data Systems (CCSDS)
Concept of Operations (ConOps)
Extensible Markup Language (XML)
Ground Station (GS)
Hypertext Transfer Protocol (HTTP)
Interface Control Document (ICD)
Just In Time (JIT)
Kongsberg Satellite Services (KSAT)
Machine-to-Machine (M2M)
Mission Operation Center (MOC)
Network Operation Center (NOC)
OpenAPI Specification (OAS)
REpresentational State Transfer (REST)
Service Level Agreement (SLA)
Software Defined Radio (SDR)
Yet Another Markup Language (YAML)

1 Introduction

Planning spacecraft operations is dependent of the purpose of the mission, ranging from science or mapping missions serving a long-term objective to missions where the objective may change rapidly on short-term notice. Going a few years back, with relatively few active missions, coordination between satellite operators and ground station providers was mostly set up to support more long-term planning. Interaction was based on file exchange and operational support was done by mechanisms such as email or even voice. Planning and scheduling of satellites was often done days or weeks ahead in time.

This approach has over time served all parties well, and is still true for missions that does not need more flexible operational modus, i.e., the ability to change satellite operation on short-term notice. However, as both the space industry and user expectations have evolved over the last years, we have seen a growing need for a more flexible approach that remove the manual processes involved traditionally.

Being a ground station provider for many years, we have observed a significant change over the last years – new spacefaring nations have entered the arena along with new commercial spacecraft operators. The number of launched spacecrafts/constellations have especially increased the last couple of years, and this tendency is foreseen to continue for both communication and observation of the Earth [1][2][3].

1.1 Motivation

When starting to plan for the projected growth back in 2018, contact planning was often batch-oriented, where multiple contacts were sent to the ground station days or even weeks in advance. We soon realized that the way the industry was heading, expectations from spacecraft operators were going to be different than before. Two of the identified factors were directly related to the operational concept:

- Expectation of lower service cost
- Higher flexibility for contact scheduling

As the industry has evolved over the latest years, there has been a decrease in cost for both manufacturing of spacecrafts and launches [4][5]. This has created a demand for lower ground station service cost [4]. Based on the foreseen future variety of spacecrafts, along with an expected increased load, only adding assets would scale poorly from a cost perspective. From a ground station provider perspective, maximizing asset utilization (the antenna being the main contributing factor) and minimizing human interaction would be key factors to realize a service offering at a lower price point.

In general, ground stations can provide a wide number of services to spacecraft operators, ranging from data acquisition and alignment, through processing to data distribution. The expectation for lower service cost could therefore, in theory, impact several parts of the service chain – from interaction with the spacecraft operators to how assets are utilized. So, to achieve a multi-mission approach[i] with a high degree of self-service requires automation from spacecraft operator planning, schedule exchange and automated activation of assets.

Based on the foreseen increase in constellations, an operational change was expected where a more holistic approach was needed for planning management. In such a scenario, automation and software solutions are fundamental factors, both for commanding of individual spacecrafts and for handling downlinks from available assets. By providing an always-on interface that could be used by spacecraft operators at any time, flexible planning could be supported with little human interaction needed.

In addition to the two factors identified above, there is a need for providing insight into the ground station service. The fundamental architecture for this was based on an internal need to streamline our own operations, by providing a uniform monitoring and control interface to ease the work for our operational staff. However, it soon became evident that this could also be amended to provide operational insight to spacecraft operators. With more constellations being launched and more automated planning solutions on their side, a need for more insight into the activities taking place at the ground station surfaced. During a spacecraft contact, the ground station equipment involved provides several performance metrics. This information needs to be promptly available at the receiving end to give maximum value. An example is downlink quality or error rate to adjust downlink parameters.

[i] *Assets, e.g., antennas, are used to serve several compatible spacecrafts*

1.2 The way ahead

Based on the above, we realized that changes were needed to account for current development and trends within the space industry. These changes were categorized within the following areas:

- Foreseen growth and increased activity
    - Multi-mission support throughout the service chain
    - A more flexible planning interface, with loose coupling between the parties and statelessness
    - A uniform interface that could serve all spacecraft operators
- A change in ConOps towards more rapid tasking and interaction
    - Closer planning loops with more timely feedback
    - Minimal manual interaction
    - More insight into ongoing activities

Software and automation were going to be important factors. Ground station providers play a crucial role in ensuring safe, reliable, and predictable reception of downlinked data. Provided interfaces must be available on a 24/7 basis, generating correct and timely feedback on both requests and statuses.

2 Designing For Growth

Our distributed monolithic architecture worked well for the problems we had faced in the past and was heavily influenced by how the business operated until then. As described in 1.2, we started planning for major changes in the industry which would impact our business. Therefore, we started designing and building towards a modern ground network solution that:

- automatically processed and quickly responded to customer requests without human intervention
- could be used by many missions "out of the box"
- supported continuous integration deployment, enabling us to release changes more frequently
- enabled cloud-native design to allow for scalability and security enhancements
- allowed us to utilize public cloud infrastructure when feasible, and
- could be deployed in an interoperable way with our existing systems and services

Some of the challenges and design goals required not only technical solutions, but also a change in business model, concept of operations and infrastructure. We have chosen to limit the scope of this paper to the technical changes we conducted over the course of two years, and their ConOps implications.

To address the technical challenges and design goals described, we opted to augment our existing architecture with software services using the microservices architectural style [6]. A microservice architecture is a variant of service-oriented architecture in which the application is comprised of a collection of loosely coupled services communicating through secure, lightweight protocols. Unlike modules in a monolithic architecture each service is self-contained and enforces that all interaction happens through well-defined, public interfaces.

The software services encapsulate a discrete slice of the capabilities we offered to external and internal users:

- Schedule & assets management
- Ephemeris management
- Reporting
- Monitoring
- Authentication & Authorization

Each individual service was based on either REST (see section 2.1) or the publish-subscribe (see section 2.2) architectural styles. Figure 1 shows an overview of the individual services and their interactions.

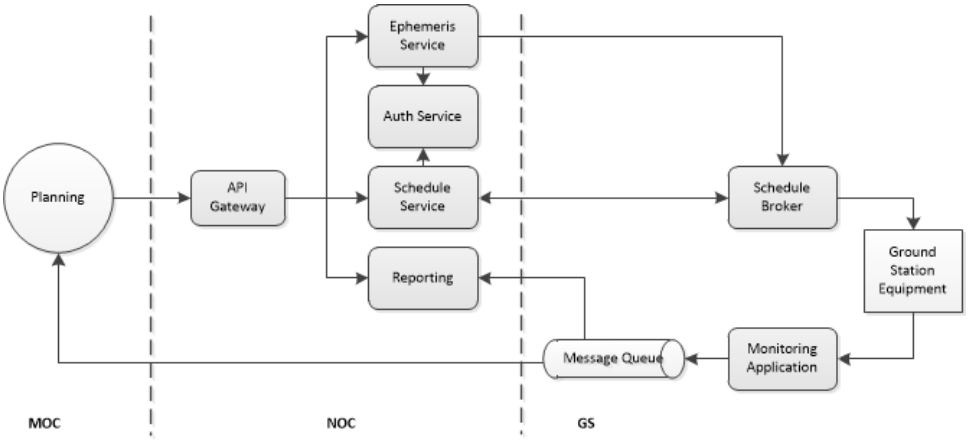

Figure 1 High-level architecture

Clients integrated with our system can query for available apertures, manage schedules, and retrieve historical reports of our performance. During a contact the system provides near-real time insight into how ground equipment is performing which is useful for operators internally and externally.

Services and their function shown in Figure 1 are described in Table 1.

Service	**Responsibility**
API Gateway	Filter and proxy incoming requests to the correct backend service
Ephemeris Service	Manage ephemeris files and provide visibility predictions and tracking files
Auth Service	Authentication and authorization service validating credentials
Schedule Service	Manage assets and contact schedules
Reporting	Provide post-pass reports and real-time monitoring information across stations in the network
Schedule Broker	Handles interaction with Schedule Service to retrieve schedules, configurations, ephemeris. Schedules contacts on ground stations
Monitoring Application	Continuously monitors the various Ground Station Equipment. Every monitoring parameter is normalized and published to a Message Queue
Message Queue	Durable, first-in, first-out event-bus enabling clients to asynchronously consume messages published by Monitoring Application
Ground Station Equipment	Antenna and base-/wide-band equipment supporting a spacecraft contact

Table 1 Service Description

2.1 Representational State Transfer

Many of the microservices are built using the REST architectural style [7]. REST is a set of design principles for building web services and is widely used to implement network-based systems. Applications that adhere to the design constraints REST defines are sometimes described as *RESTful*.

In REST, *resources* represent the entities or data that the application manages and exposes to clients through a uniform interface. Resources are representations of domain-specific objects such as *spacecrafts, apertures, contacts, ephemeris,* etc, formatted as JSON or XML.

RESTful applications typically employ the request-response pattern. A web server makes available resources clients can operate on through a set of standard operations. These operations are typically mapped to HTTPS methods as shown in Table 2.

HTTPS Method	**Operation**
POST	Create
GET	Read
PUT	Update
DELETE	Delete

Table 2 REST operations

The set of standard operations allows RESTful applications to provide a consistent and predictable way for clients to interact with *resources.*

2.2 Publish-subscribe

Publish-subscribe (PubSub) is an architectural design pattern for asynchronously exchanging messages between publishers and subscribers [8]. In PubSub-applications a server produces messages for clients to consume. Because of the loose coupling, publishers do not need to know the identity or location of the receivers. The use of topics[ii] allows for the efficient filtering and routing of messages, enabling targeted delivery of information to specific groups of subscribers.

There are various implementations of the publish-subscribe pattern, such as message queues, event buses, and data streams. These different implementations vary in terms of features, performance, and scalability. However, the core principles of decoupling and topic-based routing remain consistent across all implementations. A common protocol used in publish-subscribe software is the open standard AMQP (Advanced Message Queuing Protocol) [9]. It is a binary protocol based on TCP and can efficiently support a variety of communication patterns.

2.3 OpenAPI

The OpenAPI Specification (OAS) is used to describe the capabilities of a RESTful web service. OAS is a standard for defining an Interface Control Document (ICD) that features a machine-readable format that can be used to generate documentation, client libraries and other tools that make it easier to work with the API. OpenAPI documents are created in either YAML or JSON and describes an API, its endpoints, message formats, error codes and security mechanisms.

2.4 Architectural Design

Basing our microservices on the REST and PubSub architectural styles has yielded several benefits. Both architectural styles incentivize loose coupling between services and their collaborators. This makes possible a high degree of flexibility and scalability, as client and server can evolve independently. Features or resources can be added to services independent of changes to client code until the client decides to support them. This has helped avoid a lot of complex coordination with external users but has also had a positive effect between different development teams at the company.

Each microservice is self-contained, meaning it includes everything it needs to operate and fulfil its role, which has also made it easier to change things over time. Especially software using shared infrastructure resources such as filesystems, databases, and load balancers can accumulate technical debt over time since making changes can require complex coordination between teams.

Despite its advantages, a microservices architecture is far from free. The most immediate effect is the increased overall complexity. Reliable, distributed systems are hard to build and the introduction of a network boundary between services opens a host of new failure modes compared to in-memory communication.

Testing and debugging microservices by themselves is often straight-forward, if not easier than monolithic applications. Each individual service is usually smaller in scope and has fewer internal dependencies but testing the complete microservices system is hard. Reproducing issues locally is not always possible when factoring in differences in simulated and real-world usage patterns. To account for this, we have invested a lot in instrumentation and observability to better understand how every service performs during outages and correlate events across service logs.

Software security is critical and must be designed into an architecture, as it is very hard to secure an insecure design. Realizing a new architecture also meant designing for security in depth and zero-trust. Communications between microservices must be encrypted in-transit, and both AMQP and HTTP have secure equivalents based on the Transport Layer Security (TLS) protocol. Using role-based access control (RBAC)

A key factor in our journey towards a microservice architecture has been managing interface-changes. For services to be reliable, their public interface must be consistent even if implementation details underneath change. By ensuring that the only integration point available outside of a service is its public API, developers gain confidence in making changes so long as the interface does not change. Describing each service's interface using OAS helped us communicate what a service was capable of and how to use it. A machine-readable ICD allows for

[ii] *A logical grouping of events consumers is able to subscribe to.*

automating interface generation based on the OAS. By including interface generation as a compile-time step developers can get immediate feedback on discrepancies between API specification and implementation. This may prevent subtle interface changes before deployment. Overall OAS is a useful tool for defining contracts between services and communicating capabilities between development teams internally and externally.

Interfaces sometimes need to be changed. To be able to continuously improve our offering and provide new features we employed a versioning scheme to allow us to rectify errors from the past or create new features. Every endpoint made available is prefixed with a version to allow existing integrations to keep operating until all clients have migrated to the new version. An example API endpoint broken into components is shown in Figure 2.

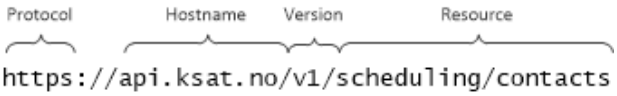

Figure 2 Example API endpoint

By differentiating between breaking and non-breaking changes we established a set of ground rules to clarify across development teams what constituted a breaking change versus a non-breaking change. Generally, adding new parameters or endpoints is possible within the context of an existing version but removing or updating parameters are considered a breaking change.

3 Lessons learned

When embarking on this effort some years back, we underestimated the complexity and effort required. Development took longer than initially anticipated, as it usually does. Looking back, part of it is attributable to short-sighted technical decisions we made early on, whilst others were due to misconceptions of the number of new missions and their expectations.

Despite fairly accurate projections the industry grew faster than we expected. Developing a service itself did not necessarily take a long time, but integration and deployment was time-consuming all the while we were scrambling to support an unprecedented number of new missions at the same time.

3.1 Expectation Management

Cost-control was very important going in – we had planned to offer a more standardized service at a lower price point. Investing in automation and removing humans from the loop did reduce cost, and we were able to support more spacecraft operators and missions with the same automated services. However, the more important benefit was the increased performance and reliability impact this has had on our offering.

Another lesson learned was that providing a uniform interface that would serve all spacecraft operators at the same time added another level of operational challenges. When early adopters of the new interface went operational, it became more difficult to apply changes to support new actors with different needs. Providing an ever-evolving service to many organizationally diverse clients does ask questions of the Change Management-practices.

Around launches we would typically be expected to enter periods of change freezes. This was tough for us to accommodate all the while we had yet other spacecraft operators dependent on us making changes to support their missions. We were able to resolve this over time by clearly stating up-front our policy on being able to make changes, while also improving our internal versioning practices to prevent us from mistakenly breaking a client integration. Encouraging teams to state up-front what collaborators can expect in terms of Service Level Agreement (SLA) and change management practices in context of their services is something we endorse.

3.2 Interfaces Are Forever

Great care should be taken in designing service interfaces. This is especially true across organizational boundaries. Once an interface is in use it can be time-consuming to discontinue. The schedule interface we initially made available closely mimicked the original file-based formats already in production. Suffice to say it worked better in a file-exchange context and its semantics have turned out not to be a good fit. Despite publishing new and more ergonomic interfaces, we still support our initial interface alongside newer ones and expect to continue supporting them for several years.

Ensuring backwards compatibility is crucial in a multi-mission context and having stable and long-lived interfaces helps. Hence, there is a need for standards to emerge and be widely adopted. The Consultative Committee for Space Data Systems (CCSDS) has fostered several standards that have made cross support easier

and ensured compatibility between parties. The CCSDS 902.1-B-1 Blue Book published May 2018 describes the Simple Schedule Format (SSF) format designed to facilitate exchanges of aperture schedule information between space agencies and/or commercial or governmental spacecraft operators. It is intended to be used throughout a mission's lifetime and was derived from analysis of schedule formats used by various space agencies [10]. We were aware of but chose not to implement the CCSDS 902.1-B-1 standard for our message formats. Our justification at the time was that it would cause interoperability issues which would be time-consuming to resolve.

With more and more constellations coming online we believe the needs and ergonomics of scheduling interfaces will change. Capacity utilization constraints and radio-frequency interference are examples of challenges that may necessitate ground operators to help spacecraft operators plan more efficiently.

3.3 ConOps Changes

The introduction of a new always-on interface also had implications on our concept of operations. Moving from a semi-automated world, where humans validated and accepted plans, to an automated scenario effectively transferred responsibility and control to the users. As the volume of spacecraft contacts has steadily grown and our software has improved over the past few years, our operations run better with less human interaction.

Another aspect of this that impacted our ConOps was shifting from predictable contact plans on a weekly basis, to more dynamic and less predictable scheduling. This necessitated a change in how we conducted our engineering work and maintenance activities. In this new world, any activity impacting a ground station must be put into the same overall planning system and inform users. Improving the breadth and scope of our operational notifications, delivered via a publish-subscribe interface is an ongoing effort to better support spacecraft operators and their planning.

A side effect of allowing more frequent changes to schedule plans is that responsibility of successfully getting plans into the system has transferred to the user. This underlines the importance of having reliable services on both ends and has worked well, albeit with periods of friction due to outages or ground station capacity constraints. For single-spacecraft mission with predictable needs for support the increased flexibility may not seem worth the extra responsibility.

4 Discussion

As stated in section 1.2, realization of the new architecture had to be done in parallel with on-going 24/7 operations. The new architecture was gradually implemented from 2020, at the same time as activities were increasing along several axes; more spacecraft operators, more spacecrafts, and as a consequence more contacts to support (as shown in Figure 3, Figure 4 and Figure 5). This turned out to be quite challenging, particularly in the beginning, as services were still under development.

Even if we had put lots of effort into making the coupling between spacecraft operators and the ground station as lightweight as possible, it is still a coupling. And great care needs to be taken to avoid breaking client integration as the server side evolves. Communicating and synchronizing changes between client and server is possible, but experience has shown that versioning and depreciation regimes are more effective.

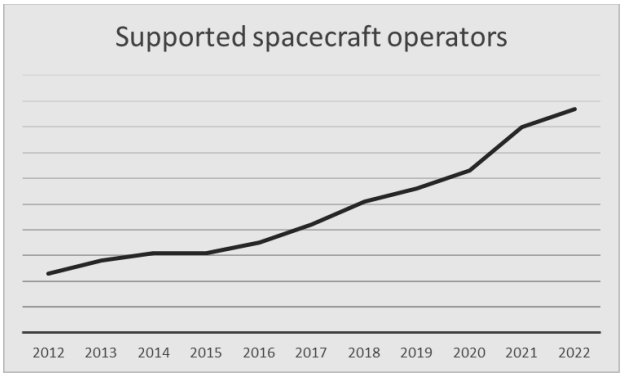

Figure 3 Number of spacecraft operators supported (2012 - 2022)

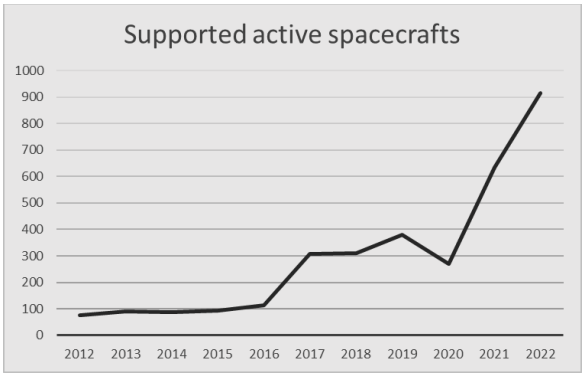

Figure 4 Number of active spacecrafts supported (2012 - 2022)

As mentioned earlier, the introduction of a new and more flexible architecture, where automation is a central part, also had implications on our ConOps. Moving from a semi-automated world, where operating personnel manually adjusted the planning outcome, to a more agile world, where spacecraft operators themselves adjusted their plans, took some time to get used to.

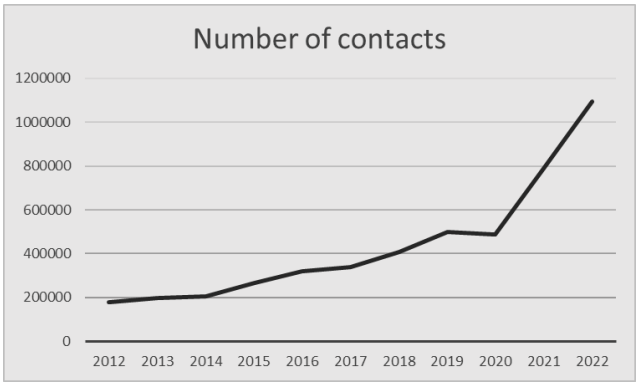

Figure 5 Number of contacts supported (2012 - 2022)

In the original architecture, spacecraft operators could plan for long timeframes, and leave to the ground station provider to solve this. By moving to a self-service, microservice architecture, supporting smaller and more frequent requests, responsibility is moved to the spacecraft operators to "keep track" and adjust. However, it is our understanding that the benefits outweigh the disadvantages, and that the new interface has been well received by the community.

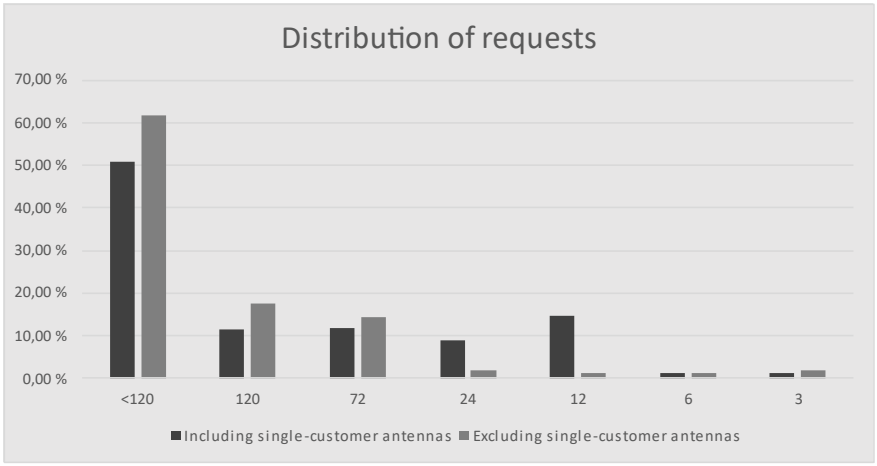

Figure 6 Distributions of requests based on hours prior to AOS over the last year

Before introducing this new architecture, requests were typically sent one week ahead, with a response the day after. Based on our experience so far, providing a new and more flexible architecture for planning and monitoring between spacecraft operators and the ground station has been well received. Although not conclusive, the distribution of requests prior to AOS (see Figure 6) indicates that spacecraft owners are taking advantage of the ability to change their schedules closer to contact start.

Ground network providers have better insight into their capability, capacity and scheduled contacts compared to individual clients. There may be utilization benefits of allowing the ground network provider to use the data it has to optimize schedules instead of each mission reserving contacts individually.

In addition to the more technical work needed to realize the new interface, another challenge needed to be addressed - migration. Spacecraft operators heavily rely on the services provided by ground stations, and new functionality had to be an addition to the existing operations without causing negative impact. However, as we expected current spacecraft operators to transition to the new services, a migration path needed to be established along with test environments. Although an important aspect, it is not addressed in this paper.

5 Conclusions

When we decided to take on this challenge some years back, the NewSpace/SmallSat industry was in its early phase – expectations were high, but activity was still fairly low. However, the growth experienced over the last years has exceeded our expectations, but to our advantage the new solution was introduced at the right time, enabling it to grow with the industry.

In hindsight, even being challenging at times, supporting this growth without this new solution would have been very hard (or even impossible). The industry has grown substantially, and a large portion of this growth has been supported by this new and flexible architecture described in this document.

However, we do not regard the new architecture as finished – as with other software it will be modified and enhanced to best suit spacecraft operators and ourselves. While moving towards better solutions in a fast-developing industry, it is our opinion that automation and self-service are key factors going hand-in-hand. Building software services based on the REST and PubSub architectural styles has been successful in our experience. It provides a good foundation for further service-level enhancements.

Adapting ground station networks for automatic M2M operations covers several other areas: antenna configuration and setup, wide- and base-band equipment configuration and setup, the use of Software Defined Radios (SDR) for maximum flexibility, data processing and distribution, etc. Although the focus in this paper is on a new and more flexible scheduling architecture, this does not imply that other areas deserve less attention.

References

[1] Alan Burkitt-Gray 23, 'Fourfold increase' in satellites over the next 10 years to 17,000. December 10, 2021, https://www.capacitymedia.com/article/29tguvmaqy9h43clx4mio/fourfold-increase-in-satellites-over-the-next-10-years-to-17-000, (accessed 12.01.2023).

[2] Stephen Young, The Number of Active Satellites in Space Skyrockets . . . Literally. July 27, 2021, http://www.gizmag.com/spaceport-america-hits-nags/36200/, (accessed 14.01.2023).

[3] Sita Sonty, Cameron Scott, Troy Thomas and Sarah Zhou, Op-ed | Herding rockets: Improved Space Traffic Management will accelerate industry growth. May 11, 2022, https://www.capacitymedia.com/article/29tguvmaqy9h43clx4mio/fourfold-increase-in-satellites-over-the-next-10-years-to-17-000, (accessed 14.1.2023).

[4] Leandra Bernstein, Affordability Is Transforming the Satellite Industry: How Each Segment Is Keeping Up with the Pace of Change". July 12, 2022, https://www.kratosdefense.com/constellations/articles/affordability-is-transforming-the-satellite-industry, (accessed 12.01.2023).

[5] Jeff Matthews, The decline of commercial launch cost, https://www2.deloitte.com/us/en/pages/public-sector/articles/commercial-space-launch-cost.html, (accessed 14.1.2023).

[6] Thönes, J. (Jan.-Feb. 2015). Microservices. *IEEE Software, vol. 32, no. 1, pp. 116-116*, 116-116.

[7] Fielding, R. T. (2000). REST: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*

[8] Eugster, P. T. ((2003)). The many faces of publish/subscribe. *ACM computing surveys (CSUR) 35.2*, 114-131.

[9] O'Hara, J. (2007). Toward a Commodity Enterprise Middleware: Can AMQP enable a new era in messaging middleware? A look inside standards-based messaging with AMQP. *ACM*, 48-55.

[10] CCSDS (2018). CROSS SUPPORT SERVICE MANAGEMENT— SIMPLE SCHEDULE FORMAT SPECIFICATION, https://public.ccsds.org/Pubs/902x1b1c1.pdf