IAC-19-B6.2.5

PROTOTYPING OPERATIONAL AUTONOMY FOR SPACE TRAFFIC MANAGEMENT

Sreeja Nag[1], David D. Murakami[2], Nimesh A. Marker[3], Miles T. Lifson[1], Parimal H. Kopardekar[2]

[1]NASA Ames Research Center and Bay Area Environmental Research Institute, CA USA
[2]NASA Ames Research Center, CA USA
[3]NASA Ames Research Center and Stinger Ghaffarian Technologies, Inc., CA USA
Email: sreeja.nag@nasa.gov, david.d.murakami@nasa.gov

Abstract

Current state of the art in Space Traffic Management (STM) relies on a handful of providers for surveillance and collision prediction, and manual coordination between operators. Neither is scalable to support the expected 10x increase in spacecraft population in less than 10 years, nor does it support automated manuever planning. We present a software prototype of an STM architecture based on open Application Programming Interfaces (APIs), drawing on previous work by NASA to develop an architecture for low-altitude Unmanned Aerial System Traffic Management. The STM architecture is designed to provide structure to the interactions between spacecraft operators, various regulatory bodies, and service suppliers, while maintaining flexibility of these interactions and the ability for new market participants to enter easily. Autonomy is an indispensable part of the proposed architecture in enabling efficient data sharing, coordination between STM participants and safe flight operations. Examples of autonomy within STM include syncing multiple non-authoritative catalogs of resident space objects, or determining which spacecraft maneuvers when preventing impending conjunctions between multiple spacecraft.

The STM prototype is based on modern micro-service architecture adhering to OpenAPI standards and deployed in industry standard Docker containers, facilitating easy communication between different participants or services. The system architecture is designed to facilitate adding and replacing services with minimal disruption. We have implemented some example participant services (e.g. a space situational awareness provider/SSA, a conjunction assessment supplier/CAS, an automated maneuver advisor/AMA) within the prototype. Different services, with creative algorithms folded into then, can fulfil similar functional roles within the STM architecture by flexibly connecting to it using pre-defined APIs and data models, thereby lowering the barrier to entry of new players in the STM marketplace.

We demonstrate the STM prototype on a multiple conjunction scenario with multiple maneuverable spacecraft, where an example CAS and AMA can recommend optimal maneuvers to the spacecraft operators, based on a pre-defined reward function. Such tools can intelligently search the space of potential collision avoidance maneuvers with varying parameters like lead time and propellant usage, optimize a customized reward function, and be implemented as a scheduling service within the STM architecture. The case study shows an example of autonomous maneuver planning is possible using the API-based framework. As satellite populations and predicted conjunctions increase, an STM architecture can facilitate seamless information exchange related to collision prediction and mitigation among various service applications on different platforms and servers. The availability of such an STM network also opens up new research topics on satellite maneuver planning, scheduling and negotiation across disjoint entities.

Acronyms

AI	Artificial Intelligence
AMA	Automated Manuever Advisor
API	Application Programming Interface
ARC	Ames Research Center
CAS	Conjunction Assessment Supplier
CCSDS	Consultative Committee for Space Data Systems
CDM	Conjunction Data Message
COLA	Collision Avoidance
CSpOC	Combined Space Operations Center
FAA	Federal Aviation Administration
HIE	High Interest Event
IADC	Interagency Space Debris Coordination Committee
MDP	Markov Decision Process
MSMA	Multi-spacecraft Maneuver Advisor Algorithm
NORAD	North American Aerospace Defense Command
O/O	Owner/Operator
PoC	Probability of Collision
RSO	Resident Space Objects
S3	Space Service Supplier

SG	STM Gateway
SSA	Space Situational Awareness
STK	Systems Tool Kit
STM	Space Traffic Management
TCA	Time to Closest Approach
TCL	Technology Capability Level
TLE	Two Line Elements
UI	User Interface
UUID	Universally Unique Identifier

1. Introduction to STM Architecture

As outer space becomes increasingly congested with satellites, due to miniaturizing hardware, cheaper launches, more automated operations, entry of emerging economies and proposed megaconstellations, the satellite population in Low Earth Orbit (LEO) is expected to grow from ~1000 to over 16000 in 10-20 years [1], [2]. The increased population will be susceptible to greater risk of physical collision with each other or debris, radio-frequency interference, space weather, lasers and directed energy impacts, and thus create more debris exponentially (Kessler Syndrome). There are more than 19000 resident space objects (RSOs) greater than 10 cm in size, being currently tracked in Earth orbit. With the U.S. Space Fence expected to be operational by the end of 2019[3], surveillance of RSOs down to ~5cm is soon to be possible and will elicit more collision avoidance (COLA) maneuvers.

Current space traffic coordination mainly relies on the U.S. Air Force's Combined Space Operations Center* (CSpOC) to provide state tracking and conjunction prediction emails to operational spacecraft. COLA strategies may be vetted by injecting proposed maneuvers to CSpOC to assess conjunction mitigation. Space agencies of some countries have customized traffic management teams for their own satellites, e.g. NASA's Conjunction Assessment Risk Analysis (CARA) for NASA's non-human spaceflight missions and the French Space Agency CNES's CAESAR. Such Space Traffic Management (STM) services are centralized and cater to very specialized consumers, therefore *not* scalable to 10 or 100x increases in population. Moreover, any *new* STM system will have to account for existing players (CSpOC, CARA, etc.) and players that have supported them. For example, Space Situational Awareness (SSA) data available from the U.S. Strategic Command, the non-profit Space Data Association (SDA) - a consortium of several major spacecraft operators, for-profit companies like LeoLabs, or databases of individual companies like Planet Labs that volunteer their information publicly. STM or SSA data is exchanged using message standards set by the Consultative Committee for Space Data Systems (CCSDS). Even within the U.S., regulatory authority to supervise and license commercial space activities is split across several agencies, e.g. NOAA for remote sensing satellites, FAA's Office of Commercial Space Transportation for launch and re-entry of spacecraft and their interaction with the national airspace. Operators and national regulators also have to account for international law (e.g. the Outer Space Treaty), non-compliant players (since no one body has regulatory authority over space), and orbital debris mitigation compliance with international forums like Interagency Space Debris Coordination Committee (IADC).

While previous studies have discussed different perspectives on the complex landscape of managing global space traffic[4], identified preventive strategies against dangerous overcrowding of low Earth orbit (e.g. slot based allocation of sun synchronous orbits[5]) and active mitigation strategies for preventing collisions against debris (laser-based orbital control[6], [7]), no scalable or modular solution has been prototyped yet. The recently signed Space Policy Directive 3 in the U.S. supports the transition of civil STM responsibilities to a civilian entity and provides additional guidance relevant to the development of an STM system. The International Association for the Advancement of Space Safety STM working group[8] and the AIAA STM working groups have begun to identify the key requirements for such a system, as advised by academic, industry, legal and government stakeholders.

To keep the utilization of outer space safe, sustainable and efficient, to provide structure only when necessary and allow flexibility of operations otherwise, NASA Ames Research Center (ARC) has proposed an STM architecture - visually represented in Figure 1, and described in detail in Reference [9], [10]. The architecture is a set of defined roles, standardized open interfaces (e.g. application programming interfaces/APIs), and data models (with required and optional fields, based on CCSDS and other industry standards), that allow automated and scalable interaction. The *defined roles* are:

- Owner/Operators (O/O), who own and fly satellites participating in the STM architecture.

*CSpOC was called Joint Space Operations Center (JSpOC) until July 18, 2018

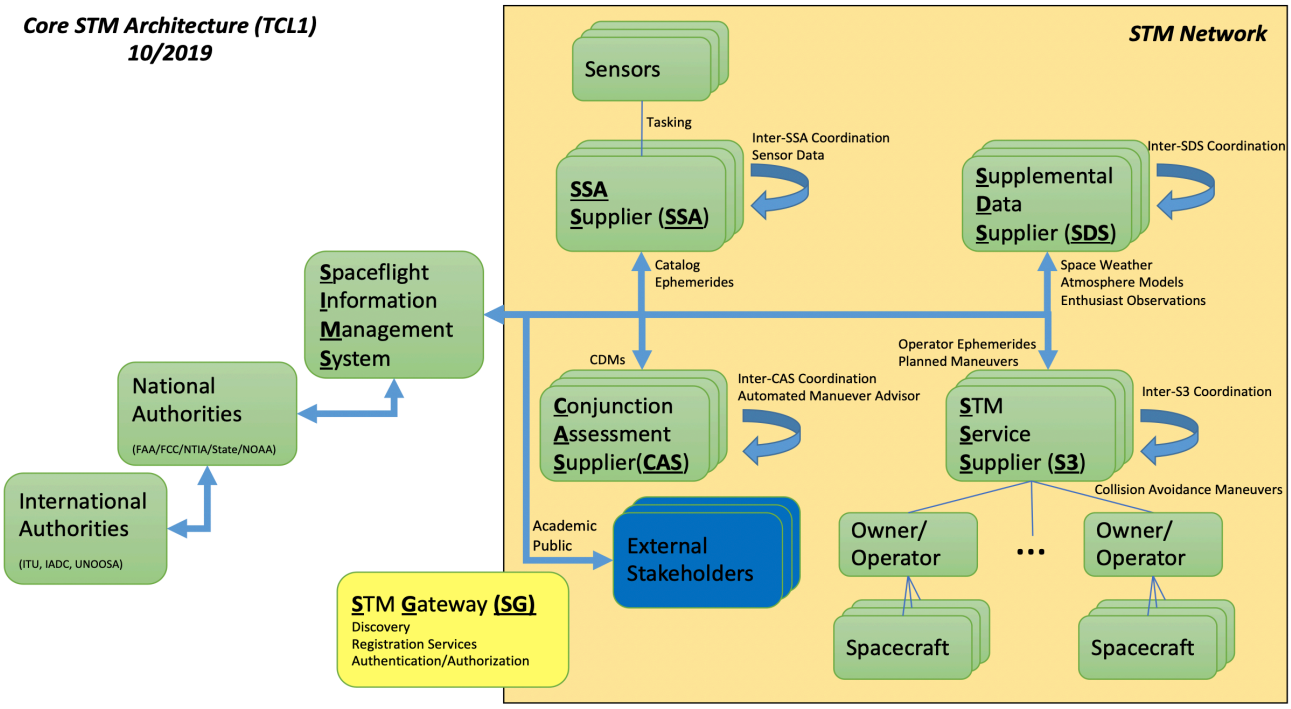


Figure 1: Space Traffic Management architecture proposed by NASA Ames Research Center, as detailed in [9], [10]. The Concept of Operations (ConOps) is based on NASA's successful UAS Traffic Management architecture that has been developed in partnership with the FAA.

- STM Service Suppliers (S3), who act as a concierge providing STM & compliance services to O/Os, serve as a link to the broader STM ecosystem, and procure services on behalf of O/Os.
- Space Situational Awareness Suppliers (SSA), who are responsible for acquiring and fusing sensor observations and cooperative tracking data from O/Os to generate and maintain a catalog of space objects.
- Conjunction Assessment Suppliers (CAS), who are responsible for screening objects against SSA catalogs for potential conjunctions, as well as verifying collision avoidance maneuvers proposed by S3s.
- Supplemental Data Suppliers (SDS), who provide other relevant data and services. Examples might include precision atmospheric modelling data to reduce errors in spacecraft orbit propagation or space weather warnings.

The roles and responsibilities are described in functional terms because the STM architecture makes no assumptions about the type of actor who would provide a given service (government, non-profit, or commercial) or regarding the separation of roles across actors. Multiple roles might be fulfilled by a single conceptual or legal entity. For example, a large O/O who flies many satellites might choose to act as its own S3 or contract with an outside provider.

As space traffic gets more complex, new functions may be developed within an existing CAS or S3, or they may contract with an outside provider. For example, if a CAS has informed an S3 of an imminent conjunction and the S3 needs guidance on maneuver options and trade-offs, they may use their in-house planning tools on CAS-provided data, request the CAS perform the analysis, or request an external service (e.g. Automated Maneuver Advisor (AMA)) to use their custom planning tools on S3-provided CAS data related to the conjunction.

New services added to the STM network may even be hardware services by providers that do not fall into any of the above categories (neither operators, brokers, software nor data suppliers). For example, external and active de-orbit services may be supplied to operators of non-manueverable spacecraft by external providers of such mechanisms (e.g. rendezvous chasers, tethers, robotics arms, lasers, aerodynamic decelerators).

The STM Gateway is a *management role* within the proposed architecture that will handle certain basic functions like registration, discovery, authentication of

participants, and auditable tracking of data provenance and integrity. This open-access architecture allows any user to join the system, after reliable authentication, and be discoverable as a new participant by a centralized registry of participating entities or various decentralized discovery techniques. We can thus accommodate users ranging from small academic Cubesats to proposed mega-constellations with thousands of satellites, while addressing their customized needs for communications, interoperability, regulation, and protection of proprietary data. The APIs and data models present a low barrier to entry for new or existing STM actors and/or services. As long as these interfaces are followed, STM is expected to serve as a non-hierarchical marketplace for services that can accommodate future regulatory requirements, and integrate data from multiple sources, providing information to those who need it to behave responsibly, while being responsive to source-imposed restrictions on sharing. Decentralization also implies that new nodes can be added *scalably*, and common standards allow software developed for one supplier to be *reused* and *interoperate* with another.

We have deployed a research platform within a simulation lab at NASA Ames. The goal is to visualize, assess, and validate our proposed STM architecture and performance under increasingly complex use cases, grouped into technical capability levels (TCL). The TCLs provide an evolutionary path to implement a software prototype of the proposed STM architecture and its application to groups of use cases. This paper describes the prototype and its application to on-orbit operations for a collision avoidance and maneuver planning use case, per civil/commercial catalogs. Once development is complete, the intention is to open source the platform code to enable use and further development by all stakeholders interested in participating in an STM ecosystem, as well as eventual transition to the appropriate regulatory agency for use in operations.

2. Software Prototype of the STM Architecture

The STM software prototype implements the ConOps of our proposed architecture (Figure 1) in successive TCLs, similar to the manner in which NASA's UTM software prototype[11] was developed and is currently being handed to the FAA for operational use via the NASA UTM Research Transition Team that was formed to "collaboratively explore concepts, develop prototypes, and demonstrate a possible future UTM system to enable large-scale low altitude UAS operations".

We used the iterative Software Development Life Cycle (SDLC) process to develop the STM prototype. SDLC's phases included creating a detailed plan for our requirements analysis, definition, product design, architecture, development, testing and deployment. This section presents our implementation of the system software based on the proposed STM architecture (each role is an 'application') and describes the functional elements and their interactions using activity diagrams and API structures.

The STM architecture requires applications (S3, STM Gateway, CAS, SSA) to interact as loosely coupled services, making the microservices architectural pattern a strong fit for STM implementation. Microservices allow the construction of small independently versioned and scalable customer focused services with specific business goals which communicate with each other over standard protocols with well-defined interfaces. As the services are independently deployable and scalable, each service also provides a firm module boundary. Other benefits of microservices include [12], [13]:

1. Complexity localization — Services are self-contained, independent applications. The development team for each service (S3, CAS, etc.) is only concerned with understanding the complexities of their service. Other teams only need to know what capabilities are being provided by the other services; they don't need to know how they work internally.
2. Cross-cutting business functionality — Eliminates the need to reinvent standard pieces of functionality used across the organization multiple times; for example, authentication and user management.
3. Increased resiliency — Since a number of services communicate simultaneously, when one fails, the client should be designed to allow its neighbors to continue functioning as it steps out as elegantly as possible. Improved fault isolation means an uninterrupted user experience.
4. Better scaling, efficient system optimization and organization — Scaling decisions can be made at a more granular level.
5. Output flexibility — Simplified data extraction.
6. Real-time processing support — The publish-subscribe framework facilitates data processing in real time.
7. Support for best technology selection — One is not limited to a single technology set for the overall project. Each microservice can be developed using the most appropriate programming language and data storage technology for its function.

8. Scalability — High level of code and data reuse, making it faster and easier to deploy additional services to address new use cases.
9. Security flexibility — Allows applications to segment off and outsource their non-core business functions without disclosing core services.
10. Experimentation flexibility — Ability to try out a new technology stack on an individual service. Compared to monolithic designs, any dependency concerns will be smaller and rolling back changes is simpler. It also eliminates any long-term commitment to a single technology stack.
11. Coordination — Uses event-streaming technologies to enable easy integration compared to the heavyweight inter-process communications protocols.

2.1. STM Software Stack

The STM software stack is an implementation of our proposed STM architecture, intended to provide the basic *functions* (yellow box in Figure 1) and serve as a portal to allow seamless interaction of *roles* and *services* (green boxes in Figure 1). It was designed to be simple, highly extensible and modular, while utilizing as much open source technology as possible. The public software developer ecosystem was studied to choose a stack which would be easy to implement, and is well supported by a reputable open source community.

User Interface (UI) webpages have been developed for the STM gateway (SG) and S3. The SG UI currently supports basic functions of registration and discovery, providing capabilities for STM managers to manage and verify interfaces between S3-CAS-SSA. The current S3 UI supports O/O registration, setting up screening requests to the CAS, and displaying results from the CAS. We have selected a modern web development framework to provide capabilities such as notifications, integration with web exchange protocols, and is modular and extensible. The following web frameworks were considered:

- Angular 2+: Popular framework, large support community, MIT license, maintained by Google.
- React: Supported by Facebook, large developer support community.
- Django: Python web-framework, high scalability, offers high security, provides rapid development.

Angular 5 was chosen as the web development framework due to its small package size, extensibility, support of simple progressive web application, material design, in-build code optimization, extensive capabilities to interface with APIs and our familiarity with it.

Application Program Interfaces (APIs) are an effective interface mechanism to share functionality amongst independent applications/programs while maintaining code separation. STM APIs have been developed to provide interfaces to CAS, SSA, S3 and SG, using simple http protocols. Our requirements for API were to develop a door/inlet into each service (CAS/SSA/S3/SG), allowing other programs to securely interact with it without the need to share any code.

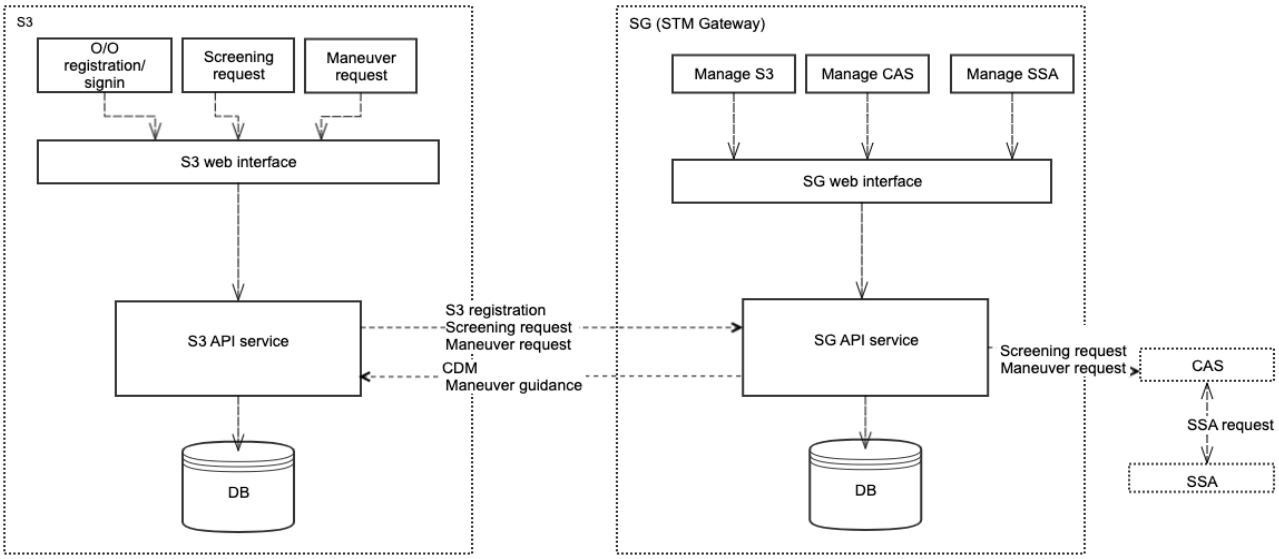


Figure 2: Current STM Testbed showing the functions for SG and one S3 instance, interacting with one CAS and one SSA. The architecture supports multiple instances of services (S3, CAS, SSA, etc.)

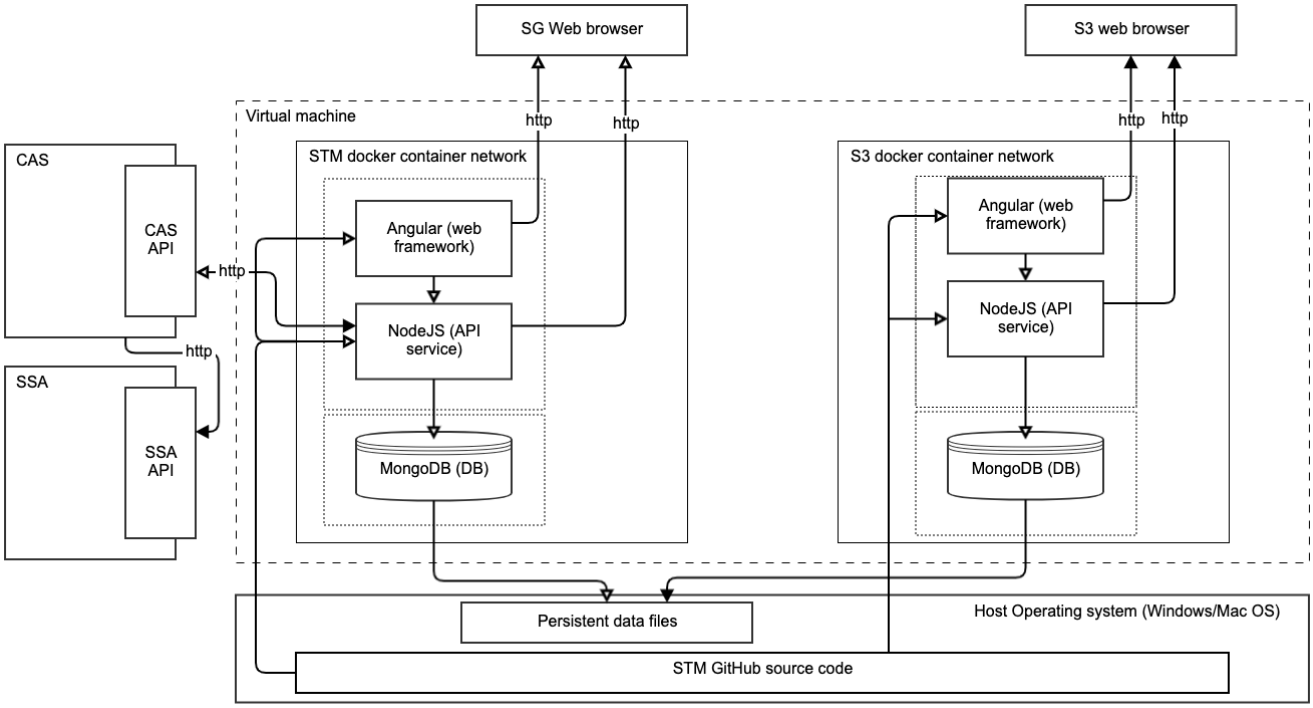


Figure 3: Current deployment structure of the STM stack. Arrows represent flow of information. Only one instance of services (S3, CAS, SSA) is shown, however multiple are possible.

The following API frameworks were considered:

- Node.JS: Javascript-based runtime environment providing non-blocking, event driven servers due to its single-threaded nature; remains lightweight and efficient in the face of data-intensive real time applications that run across distributed devices.
- Flask: Python-based lightweight web application framework designed to be quick and easy, with the ability to scale up to complex applications.

Node.JS was chosen as the preferred framework. We use SwaggerHub, an API development platform that leverages the core capabilities of the open source Swagger framework to build, document, manage, and deploy the STM APIs.

Databases (DB) were created for storing user authentication and authorization, storing requests, responses and configuration details. Our requirements from a DB were a simple setup, to be Docker friendly (small foot print), allow JSON-like binary data points (BSON). The following database tools were considered:

- MongoDB: Open source document-based database management tool that stores data in JSON formats; Highly scalable, flexible and distributed NoSQL database; Schema-less, no complex joins, ease of scale out, conversion/mapping of application objects to database objects not needed.
- Couch-base: Open-source NoSQL, multi-model, document oriented DB management system that store JSON documents.
- PostgresSQL: Object oriented relational DB with an emphasis on extensibility and standards compliance; functions and operators can be used with JSON and JSONB.

MongoDB was chosen due to its docker friendly images, ability to store JSON, capability to change streams which enables pushing updates across and from various service providers and STM roles.

Express was chosen as the web server due to its extensive routing features including routing, separating handlers (put, get, post, etc), static file serving, and a framework that many popular template engines can plug into.

To summarize the prototype implementation of *functions* and *roles* within our proposed STM architecture: SG and S3 have been developed in Angular (UI) and Express (web server). Their sequence diagrams and functions will be explained further via the use case in Section 3. The example CAS has been developed in MATLAB and Analytical Graphics, Inc's Systems Tool Kit (STK) software, and its structure is described in Reference [14]. Its utilization for complex

maneuver planning will also be explained further via the use case in Section 3. The example SSA draws primarily from the space-track.org database that lists the most recent Two-Line Element (TLE) ephemerides, as provided by CSpOC as a free service to the public and satellite operators. We are currently testing UT Austin's integrated database†[8] of various SSA sources (including CSpOC) as an alternative SSA example. The S3, CAS, and SSA are hosted on separate servers to mimic disjoint entities within the STM network. Currently, we have implemented an example AMA in Python inside the CAS application, on the same server. However, since it exchanges information with CAS using pre-defined APIs, the AMA and CAS can be separate entities as well. Figure 2 shows the STM test bed in terms of functions within the prototype implementation, while Figure 3 shows the chosen technologies. Deployment and information flow sequence will be described in Section 2.2 and 2.3, respectively.

2.2. STM Stack Deployment

The STM stack was developed and deployed into Docker containers, per best practices in the micro-services architecture. Docker containers are lightweight compared to virtual machines making them resource effective. They provide uniformity across development and production environments, making deployment agnostic to the operating system. They are suitable for continuous integration and deployment, with consistency across multiple development and release cycles, and have an excellent support base with several official supported containers. The Dockerized, and thereby standardized, STM environment has allowed repeatable development, build, test and production.

The current STM stack as deployed is illustrated in Figure 3. While it can support multiple S3/CAS/SSA or other services, per the STM architecture and marketplace described in Section 1, only one instance of each has been shown. The stack was verified on Windows 10, Mac OS 10+, and Ubuntu. NASA's stringent security requirements prevent admin access on any host machine. To work within NASA security parameters, which is perhaps representative of siloed application restrictions that STM is expected to operate in, the containers are deployed inside a Virtual Machine (VM). We used Ubuntu 14 as the VM to deploy the application containers. SG and S3 applications were available over http in a web browser, and will be extended to https for future implementations. SG and S3 have two containers each: MongoDB, Express + Node. The hosted Express servers which rendered the Node.JS API and the compiled Angular app were rendered as HTML + JS pages. Data persistence is achieved by configuring the MongoDB to allow the SG and S3 to store the data files on the host machines.

2.3. APIs and Interactions

Since STM's core benefit is the availability of common APIs and data models that roles can use to interact with each other seamlessly, we defined them on SwaggerHub. Every piece of information exchanged between two containers or servers in Figure 3 would invoke a Swagger request that adhered to defined APIs. For example, the OpenAPI model of the SSA response to an orbit data request is provided below. In the example, the SsaOrbitDataMessage model consists of the SsaRSO ("Resident Space Object") field, which specifies the catalog used and identifier within the catalog, and the SsaODM ("Orbit Data Message") field, which contains the ephemeris data for the object. This information is transmitted as a JSON object. All of the interactions between service providers and with SG described in this section occur through such APIs.

```
SsaOrbitDataMessage{
    SsaRSO*     RsoID{
                    description:
                                        Resident space object identifier and catalog
                    catalog             string
                                        example: NORAD
                                        Enum:
                                        Array [ 3 ]
                    identifier          string
                                        example: 27386
                }
    SsaODM*     RsoOrbitData{
                    description:
                                        CCSDS-like Orbit Data Messsage (state vector based)
                    epoch*              TimestampUTC string($date-time)
                                        ISO8601 (UTC) timestamp string.
                    frame*              string
                                        example: EME2000
                    position*           [...]
                    velocity*           [...]
                    covarianceFV        [...]
                }
}
```

The SG manages the basic STM functions (e.g. registration, discovery) that allow interactions between S3, CAS and SSA. Each S3 can register with the SG using a self-generated UUID identifying it uniquely. It stores CAS, SSA and S3 configuration settings via the S3 + SG API in MongoDB (Figure 3). STM's core customers are the spacecraft O/O's, and any S3 supports O/O registration and O/O sign-in. The S3 login page is shown below.

† AstriaGraph page accessed on October 2, 2019: http://astria.tacc.utexas.edu/AstriaGraph

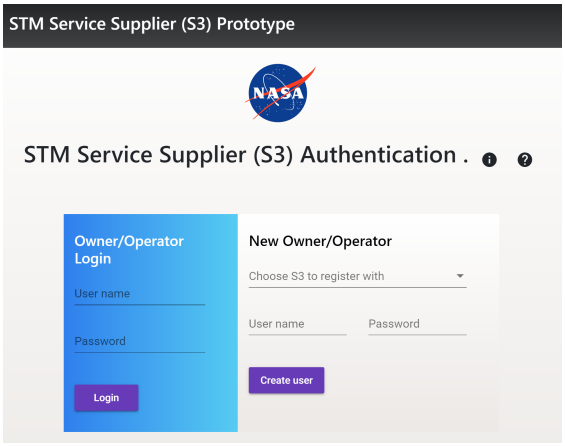
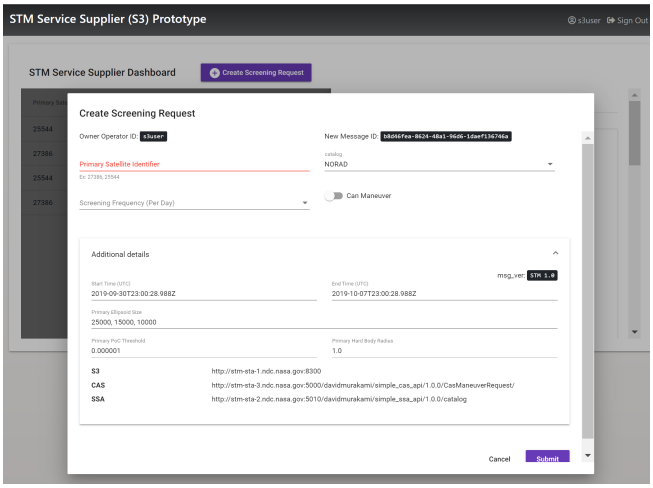
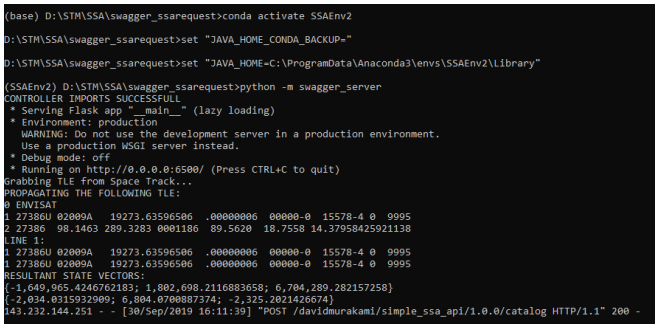


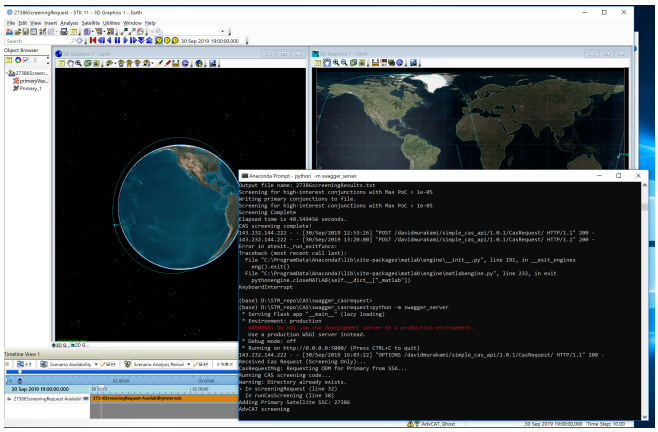
Figure 4 is a sequence diagram of typical interactions between the STM service providers for screening and collision avoidance services. S3 may invoke a screening request for the O/O's spacecraft at a pre-determined frequency or on an as-necessary basis, as seen in the UI screenshot below. The request specifies the O/O's unique ID and properties. The UI also allows the S3 to enlist their choice of SSA and CAS provider for the request. The screenshot below also shows the S3 discovery of CAS and SSA registries via APIs. These registries are published by the SG.



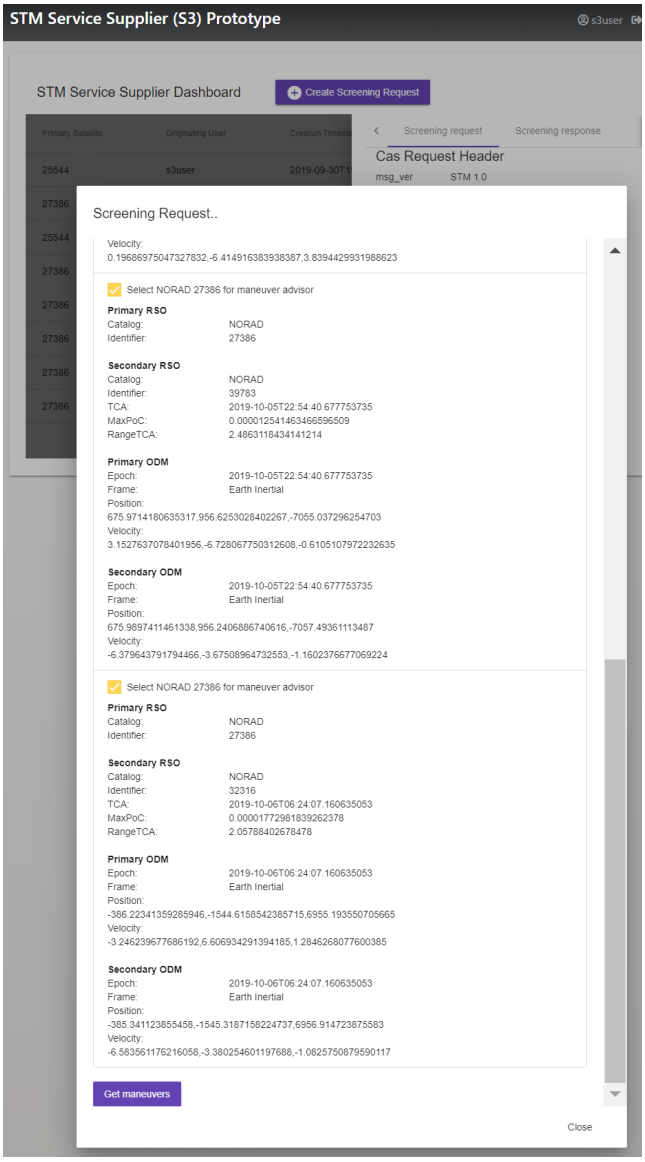
Once the screening request is submitted by S3, it invokes the S3+SG API for user authentication, authorization, storing and retrieving screening requests and responses, and then invokes the CAS API. These APIs are exposed securely by the SG. The CAS server, upon receiving the request, invokes the SSA server using the SSA API to retrieve the most updated TLE and state information on all spacecraft. The SSA server log is seen below. Note that this workflow assumes that the CAS, SSA, CAS-AMA servers are set up (running and ready to receive) before the S3 launches its requests.



Upon receiving the SSA response via API, the CAS server initiates the screening process using an example conjunction assessment application running on MATLAB and STK.



The CAS app processes the screening results received from MATLAB and STK, then invokes an API to return Conjunction Data Messages (CDM) to the S3. Each potential conjunction is selectable on the UI, so that the S3 can identify High Interest Events (HIE) among the CDMs that warrant more analysis and re-submit to CAS to re-run with more accurate SSA sources and to observe how the event evolves over time. The current prototype allows the S3 to select HIEs from the list of CDMs that they would like to plan maneuvers for. Submitting the planning request ('Get manuevers') invokes the CAS API, which then calls the AMA tool (currently housed inside CAS).



Once the manuever request is submitted by S3, it invokes the CAS-AMA API which starts the AMA application and runs the manuever planner. An example scenario will be described in Section 3 showing the iterative nature of interactions (specifically in Section 3.3) between AMA, CAS and SSA in scheduling manuevers for multiple satellites involved in frequent conjunctions. After completion, the AMA responses are currently stored on the CAS server and may returned to the S3 in the form of graphs and data files characterizing the 'proposed manuevers'. If more than one manuever sequence is proposed, the S3 may choose one 'accepted manuever' in consultation with the O/Os, make the orbit correction, and push the updated orbit ephemeris to the SSA. In turn, the SSA makes the information available to the wider STM network so that every other operator now has access to updated information. Throughout the process, security between S3 and SG is maintained with a combination of token and UUID for each S3. MongoDB stores the user(s), screening request and response, system logs, and interacts with the other applications using the S3 + SG API to MongoDB (Figure 3).

3. Application to Sequential Conjunctions and Planning Collision Avoidance

The STM prototype was used to test conjunction assessment and COLA use cases that represent complex, cascading behaviour. Current state of the art is that spacecraft operators get an email from CSpOC or another automated SSA/CAS service informing them of future conjunctions involving their spacecraft, and if they have maneuvering capabilities, they execute a COLA maneuver. There is rarely coordinated planning (or even communication exchange) between operators, and at the current orbital population, rarely a series of consecutive conjunctions expected for the same satellite in a few days. However, rare occurrences of miscommunication have lead to catastrophic and near-permanent damage in space. For example, the Iridium 33 collision with the defunct Kosmos 2251 in 2009, resulting in >2000 pieces of debris >10 cm, was caused due to imprecise SSA data, and inadequate methods of orbital updates or data sharing. More recently, ESA's Aeolus satellite maneuvered to mitigate a conjunction with a SpaceX satellite in September 2019‡. As SpaceX increases its Starlink constellation size to 7500+ satellites[15], such conjunctions are expected to become more frequent, and COLA maneuvers may have imminent, sequential impact on following conjunctions in a 'cascading' fashion.

Our proposed STM architecture allows participants to pull from continuously updated SSA sources and to publish O/O provided information automatically to an SSA (start and end of sequence in Figure 4). We have developed an example CAS [14], that is capable of automated *(A) conjunction screening*, i.e. 1-vs-all or N-vs-M conjunction predictions, *(B) encounter identification* to flag HIEs from the conjunction list based on pre-determined threshold probabilities of collisions (PoC), covariance errors allowed, etc., *(C) manuever generation* to list possible COLA strategies to avoid the HIE, and *(D) maneuver screening*, i.e.

‡ Space News, accessed on Oct 4, 2019 https://spacenews.com/esa-spacecraft-dodges-potential-collision-with-starlink-satellite/

conjunction assessment for the resultant satellite orbit if the O/O were to execute any of the COLA strategies. CAS interacts with S3 via standard APIs to provide CDMs; S3 chooses HIEs among the CDMs and submits a maneuver planning request to CAS (Figure 4). Reference [14] describes the example CAS, the algorithms used to enable the above A-D steps, and the trade-offs between the COLA maneuvers generated for consideration. This paper extends the CAS capability to introduce planning and scheduling a sequence of efficient maneuvers to clear conjunctions by maximizing a reward function that factors in system-wide conjunction risk among an arbitrary set of satellites. The maneuver planner is currently internal to the example CAS ('generate proposed maneuver' in Figure 4), however it can be a separate service that uses data exchanged between the CAS and SSA. The CAS returns a list of proposed maneuvers to S3, which selects a maneuver and implements it in consultation with the O/O, after which the new TLEs are updated in STM (Figure 4).

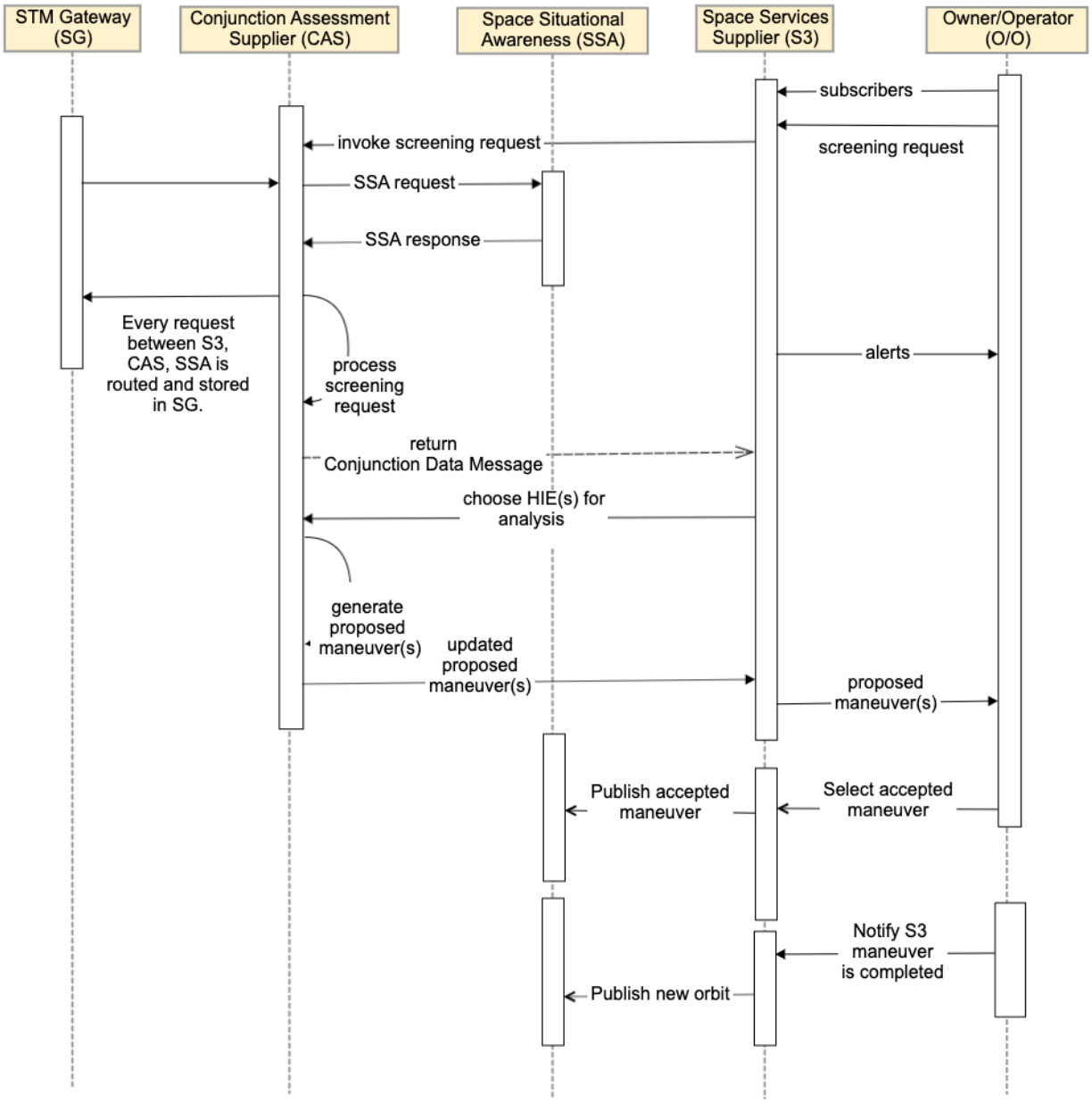


Figure 4: Sequence Diagram for interactions between the STM Gateway and various roles/providers in the STM network for the screening and conjunction assessment workflow

To demonstrate the capability of the STM prototype in a COLA sequence, we conducted a case study involving sequential conjunctions by six satellites over a one-week period, overseen by an S3. Current state of the art maneuver planning executes steps A-D in sequence and is sufficient for infrequent COLA decision-making by a single satellite against debris or non-cooperative satellites. Maneuver planning for multiple controllable agents with frequent conjunctions is more complex; it requires steps A-D to be executed iteratively as an algorithm searches through the planning horizon for an optimum sequence of maneuvers. This case study demonstrates the STM prototype's ability to seamlessly exchange information among various service applications on different platforms and servers, in keeping with process flow in Section 2.3, to enable *continuous* and *iterative* collision prediction and control. While the example planner is a system-wide reward-maximizing, greedy scheduler that outputs a schedule for COLA actions by each satellite, it paves the path for new research in satellite maneuver planning, scheduling and negotiation across disjoint entities, enabled by the availability of an STM network.

3.1. Conjunction Screening and Encounter Identification

For a one week period starting May 20 2010, we simulate an S3 that is responsible for 6 spacecraft (NORAD ID # 10676, 25419, 25477, 41918, 42809, 42961) belonging to one or many operators, henceforth called *primary* spacecraft. When sending the screening request to the CAS, the S3 sets the "do-not-violate" thresholds on PoC to be 1e-6 and the spacecraft error ellipse to be 20 km × 10 km × 5 km, in the along track, cross track and radial directions. While this PoC is more conservative and the error ellipse far larger than typical, we set it so that several sequential conjunctions are detected and collision avoidance maneuvers have a good chance of inducing new additional high-risk conjunctions over the study period, without having to forcefully increase the satellite population beyond the current database. The aim is to mimic a scenario when the population is much larger and sequential conjunctions are commonplace.

While true PoC is a more accurate metric to assess risks, it is computed by integrating a three-dimensional probability density function that quantifies the uncertainty of the relative position between any two satellites. Since we did not have access to the covariances of the 6 primary satellites, an alternative measure of risk called maximum PoC (Max PoC) was used, which is the worst-case PoC possible for a given conjunction geometry. The CAS performed the request by executing MATLAB code that automated an instance of AGI STK and AdvCAT, an add-on conjunction analysis tool. A one-versus-all screening is carried out, i.e. the orbit of each of the 6 primaries is compared to the orbits of all remaining RSOs over a week of simulation. For simplicity, STK's HPOP propagator was used with TLE-derived state vectors as initial conditions.

Based on these parameters, ~50 conjunctions or encounters were returned as CDMs by CAS to S3, and are referred to as *primary conjunctions*. Encounters identified are those whose Max PoCs violate the PoC threshold during the scenario time period.

3.2. Maneuver Generation and Screening

Candidate COLA maneuvers are generated to mitigate the primary conjunctions using the analytical formulation of Bombardelli and Hernando-Ayuso[16], implemented on MATLAB. Given an impulse budget ΔV and maneuver location $\Delta\theta$ before the Time to Closest Approach (TCA), this algorithm finds the optimal impulsive ΔV vector orientation that maximizes the miss distance between two objects at TCA. Future work may optimize the in-track, radial, cross-track directions of thrusting such that COLA actions may be incorporated into regular station-keeping (e.g., prograde in-track burns are almost free for a circular orbit). As with other elements of the CAS, note that it can replaced by another algorithm, should it be more appropriate.

The goodness of a maneuver is determined (screened) not only by the extent to which the primary conjunction was mitigated, but also by additional conjunctions that the maneuver causes. A *secondary conjunction* is a post-maneuver conjunction between the primary satellite and the same secondary RSO that resulted in the HIE, which motivated the maneuver. A *tertiary conjunction* is a post-maneuver conjunction between the primary satellite and a completely different RSO.

The maneuver planner is informed by insights from several case studies, three of which are documented in [14] - a HIE between an active satellite and debris object, an active satellite against four others, a head-on conjunction between two satellites in very similar orbits. The results show that primary conjunction mitigation and minimizing new secondary/tertiary conjunctions are conflicting objectives even for a single maneuverable satellite to avoid a single HIE (Fig 5, 7, 8 in [14]). Therefore, an inclusive objective function is required for optimum planning. For multiple sequential conjunctions, at any given point in time before TCA, there may be multiple maneuver choices by a satellite,

depending on which conjunction it is trying to avoid. Between 22 and 26 hours after the screening epoch in Figure 5, NORAD #15333 may choose to execute the red or green maneuvers, depending on avoiding NORAD #41343 or #36155 respectively. Therefore, each time step in the planning horizon can be considered a Markov Decision Process (MDP). In the figure, screening epoch is defined as the simulation start time. The original Max PoC of each event is shown as dashed lines; the TCA for each event corresponds to the point where the dashed and solid lines intersect.

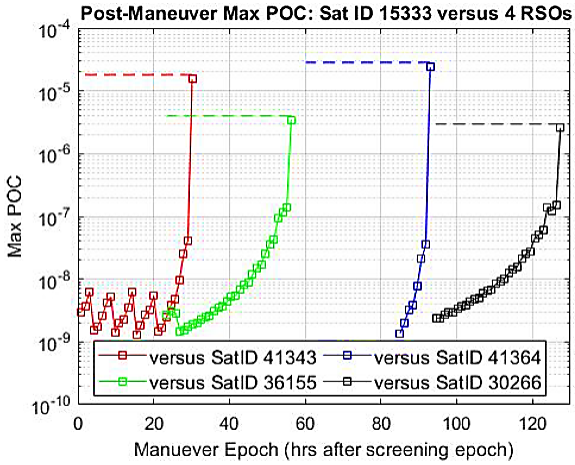


Figure 5: Example of sequential conjunctions and the maneuver options (varying Δθ only) generated for COSMOS 1603/NORAD ID: 15333 to mitigate them[14]. Maneuver performance is quantified by the post-maneuver screening, in terms of Max PoCs against the four spacecraft that caused the primary conjunctions. Each square is a 1m/s maneuver (ΔV) arranged by execution epoch (Δθ)

While this example shows the importance of multi-objective optimization (reward proposed in Equation 1 may be used) to select a maneuver or a combination, the controllable, decision-making agent is still a single S3 managing a single spacecraft. For generality, the case study in this paper shows sequential conjunctions between multiple *controllable* spacecraft and involves a maneuver tradespace for every spacecraft, and every conjunction. The O/O(s), mediated by their S3(s) and informed of the tradespace by their CAS, are expected to consult, plan and decide which maneuver is to be selected.

3.3. Maneuver Planning

An example system-wide reward function for manuever planning is proposed in Equation 1. In the case study example, it is evaluated for a set of potential collision avoidance maneuvers on a per-HIE basis, but this function could also be cumulatively added over time-steps to support alternative planning algorithm designs. It maximizes mitigation benefit minus cost across all controllable spacecraft, so that the space ecosystem is safer and more efficient as a whole. The process of *implementing* it across disjoint entities is the credit of the STM architecture and its software prototype, per the process in Section 2.3.

$$
\begin{aligned}
r = {} & w_1 * s_1 * [new\_prim\_PoC \\
& * \{(new\_prim\_PoC - th * 10^{-2}) \\
& > 0\} - old\_prim\_PoC] \\
& - w_2 * s_2 * \Delta V \\
& - w_3 * s_3 * \sum sec\_PoC \\
& * \{(\text{sec }\_PoC - th * 10^{-2}) > 0\} \\
& - w_4 * s_4 * \sum tert\_PoC \\
& * \{(tert\_PoC - th * 10^{-2}) > 0\} \\
& * \frac{1}{\log\left(time_{to_{conjunct}}\right)} \\
& - w_5 - P
\end{aligned}
$$

Equation 1

$$
\mathrm{P} = \begin{cases} -\infty \; if \; PoC(t + \Delta t) > th \\ 0, otherwise \end{cases}
$$

Equation 2

The first term captures the extent of mitigation of the primary conjunction, ignoring conjunctions with max PoC two orders of magnitude or more below the HIE threshold PoC. The second term captures the required delta-V of the maneuver in m/s. The third term captures the total PoC of secondary conjunctions introduced due to the maneuver, again disregarding conjunctions with max PoC two orders of magnitude or more below the HIE threshold. If a maneuver changes the time of the primary conjunction but does not eliminate it, this will be recorded as a secondary conjunction. The fourth term captures the total PoC of tertiary conjunctions introduced due to the maneuver (those with an RSO not involved in the mitigated conjunction). These conjunctions are also filtered to remove any more than two orders of magnitude below the threshold max PoC and additionally weighted by the time until the tertiary conjunction, so that conjunctions with the entire space ecosystem that are far into the future do not overshadow immediate ones. A log is used to reduce the severity of fall-off in the value of future conjunctions. The fifth term is a constant cost to plan any manuever, so that a single maneuver is preferred to multiple ones to mitigate potential impact to the primary mission, all else being equal. This term is irrelevant for the implemented greedy per-conjunction algorithm, but may be important for future state-space search algorithms. The sixth term

(P) is a policy term that is set to –infinity if any PoC (new primary PoC, or any secondary or tertiary PoC) is greater than the threshold PoC that flags an HIE, with a pre-defined Δt, because any maneuver that creates an new conjunction within an unacceptably short horizon, is deemed unacceptable. For the greedy algorithm, no ΔT was used, simply requiring all HIEs to be cleared before they occur. The weights w_x are set by the S3, in keeping with their customer/operator's priorities of the above described parameters. All were set to unity for this case study. The scaling factor s_x for every term is automatically determined by the planner, based on the available set of maneuvers and their corresponding PoC, so that each term is of similar order of magnitude. Δt was set to two orbital periods. The reward function can take positive or negative values, with higher reward being better.

Running a full factorial of maneuvers for each of the 6 satellites with varying executing satellite, maneuver execution time (corresponding to $\Delta\theta$), and ΔV is computationally unrealistic because it entails re-computing Equation 1 (i.e. re-propagating RSO orbits, and doing a computationally expensive search for conjunctions) thousands of times. Simulated annealing (a traditional global optimization method) with the above reward took 3-12 hours to find the optimum maneuver sequence for sat #25419 alone, with ΔV = 1-10 m/s, a week of temporal search space and various combinations of tuning parameters. Algorithm complexity scales with the power of the number of satellites. Therefore it was deemed unfeasible for exploring multiple controllable satellites, primarily owing to the computational load of the propagation and AdvCAT screening steps.

Instead, we devised a Multi-Spacecraft Maneuver Advisor Algorithm (MSMA) that informs the example AMA. The MDP is formulated as a search space for every spacecraft, similarly to that in Ref [17] Fig 4 wherein the decision variable is a maneuver option instead of instrument pointing direction, and has been implemented in Python and is housed within the CAS. MSMA is a greedy algorithm to clear all conjunctions involving at least one of S3's 6 spacecraft over the scenario time period. For each HIE in chronological order, the algorithm (1) screens for potential conjunctions above an S3 or operator-specified risk threshold per Section 3.1, (2) generates a tradespace of COLA maneuvers for all S3 spacecraft, per Section 3.2 by varying ΔV and $\Delta\theta$ between an HIE-clearing maneuver and next HIE; (3) calculates the reward for each candidate COLA maneuver per Equation 1 with PoCs computed per Section 3.2; (4) selects and simulates the COLA maneuver with the highest reward; (5) screens for HIEs and repeats this process until the system is free of HIEs from the start to end of the case study time period.

Running the MSMA for the case study duration of 1 week identified and cleared 13 HIE/encounters that included at least one of the 6 S3's satellites and warranted a COLA action. Some of these conjunctions were present in the original scenario and some new ones were induced by the proposed COLA maneuvers. Nonetheless, MSMA in the example AMA implementation reduced the total number of HIEs from ~50 (screened in Section 3.1) to 13, and cleared all of them by distributing the COLA maneuvers over all 6 spacecraft. Figure 6 shows the spread of the 13 mitigated HIEs by the 6 selected satellites as vertical lines of varying colors over the evaluated week. One maneuver option (ID# 10676, ΔV = 7.5m/s, last $\Delta\theta$) has been removed from the plot since its extremely low reward of -14.45 biases the greyscale.

For every HIE from left to right, MSMA generates M*N maneuvers; across M propellant options and N epochs ($\Delta\theta$) evenly spanning the time-space between the maneuver selected to avoid the previous HIE (colored circles in Figure 6-top and bottom) and the next HIE (Figure 6-vertical lines). While N=10, M=3 in the results presented, they can be easily changed to improve fidelity at the cost of increased runtimes. The scheduler chooses the maneuver that clears the most imminent HIE with the highest reward, before moving to the next. Figure 6 shows the full tradespace of maneuvers considered (top) arranged in specification (Y-axis) and time (X-axis), the reward possible per potential maneuver (bottom). While none of the 13 conjunctions cleared were between the 6 S3 satellites, if such an HIE were to be screened, the MSMA would choose the highest system-wide reward between the maneuver tradespace for *both* spacecraft, i.e. another stack of 30 circles for that HIE in Figure 6-top, and another 30 markers of the same color but different shape in Figure 6-middle. Only one satellite with the higher reward would manuever, i.e. no difference in Figure 6-bottom. This behaviour has verified on single events, however difficult to replicate in a cascading conjunction case study unless it is the first encountered HIE (clearing one HIE often clears other HIEs between two target satellites).

Per-conjunction reward maximization via MSMA does not necessarily lead to global optima (global maximum reward), but it is significantly more tractable and provides a reasonable baseline against which other more sophisticated solutions can be compared. More importantly, this simple example demonstrates the need

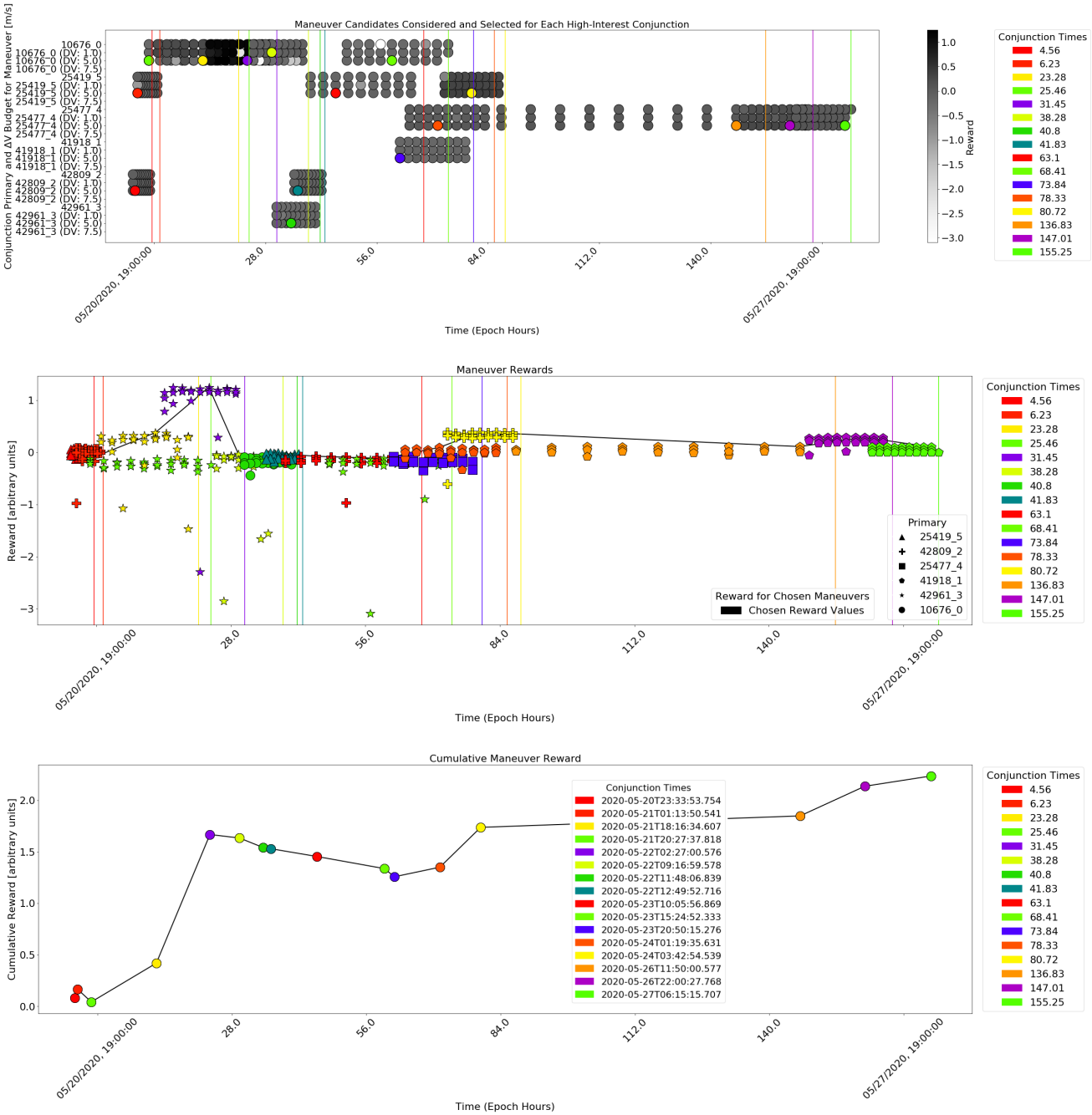


Figure 6: [top] Full tradespace of maneuvers generated by the planner, as a function of time of execution time (X-axis) and executing 6 satellites per the listed ΔV in km/s (Y-axis). The 13 HIE/conjunctions mitigated are indicated by vertical colored lines at TCA (label in bottom). The candidate maneuvers are shaded in greyscale in proportion to their reward. The chosen maneuver to mitigate a conjunction is shaded in the same color as the conjunction vertical it mitigates. [middle] Maneuver tradespace with their reward on Y-axis. The chosen maneuvers are joined with a black line. [bottom] Cumulative reward (benefit of the planner by integrating Equation 1) obtained by adding up the reward associated with each maneuver to mitigate each conjunction. The manuever markers and conjunction lines are color-matched throughout.

of automated and distributed planning in a crowded future space environment where sequential conjunctions and cascading consequences of maneuvers to avoid them will be common. The STM architecture is expected to open research in these areas, and the STM prototype is expected to enable testing them.

3.4. Maneuver Negotiation

The planning example we presented involved a conjunction between multiple maneuverable satellites, all controlled by the same S3. If multiple maneuverable satellites within an CDM are controlled by *different* S3s, maneuver negotiation may be needed before accepting and publishing a maneuver. Currently in the absence of STM, such negotiations are rare and occur by email (if at all). More commonly, an operator simply informs CSpOC of an intended manuever, evaluates screening results, and executes the maneuver without negotiation. This process has functioned thus far because conjunctions are rare—most conjunctions involve only one maneuverable satellite (the other being propulsion-less, e.g. CubeSat or debris)—and the overhead of negotiation and legal implications has been perceived to outweigh the benefits.

As conjunctions become more common and manuevers result in cascading secondary or tertiary conjunctions, manuever negotiation is envisioned when CDMs involve single or multiple manueverable satellites by different S3s. Consensus may be established by S3s communicating through the STM network and APIs, or a brokering service that is an SDS within the system. A future version of the STM prototype will define such APIs and implement example S3s (and brokers) to demonstrate the utility of such standardization.

Who maneuvers may be decided by rule-based systems like those used by sailboats, Coasian payments based systems where one would pay the other to maneuver and determine a price cheaper than its own cost to maneuver (e.g., a currency quantification of Equation 1), auction-based systems where such payment prices are bid and counter-bid between the negotiators, resource-based systems which choose the higher reward or lower cost operator (e.g., evaluated using Equation 1), dual-manuever implicit cost split where both manuever with equitable cost (e.g., by optimizing the total reward across both operator satellites using Equation 1), or "space chicken" where it is assumed that one operator will eventually decide to move to avoid an imminent HIE. These are discussed in detail in Ref [9]. Research on such topics will need to take into account rules for public sharing of information, disparity in reward definitions or weights across operators, commercial incentive to over-report mission impact or under-report efficient manuevers to avoid action, fairness over time so that the onus of action is equitable (an operator is in some way credited for previous maneuvers when considering the next required maneuver), etc.

4. Conclusions and Future Work

NASA Ames has a small scale STM lab with workstations, servers and a hyperwall, with NASA and AGI software suites, and is leveraging the UTM experience and codebase for STM development. Figure 7 shows our development process from left to right: we have proposed an STM architecture in the past, summarized in Figure 1[9], [10]. In the current paper and associated references[14], we have developed a software prototype using example service providers (e.g. CAS, AMA, SSA) and in-house sample models or publicly available algorithms for internal functions of the services. We are now involving early partners in industry, academia, and government who will be potential operators and higher fidelity service providers, i.e. potential customers of the STM network. The modular, containerized architecture ensures that they will interact only through APIs and data models, without having to share any proprietary internals of their software. The presented prototype of the STM architecture is for an initial version of TCL1 (on-orbit operations with civil catalogs). The architecture and prototype will be matured over the TCLs defined in Ref [10] Fig. 8, and as we improve our use case portfolio and increase partners. The process in Figure 7 is expected for every TCL.

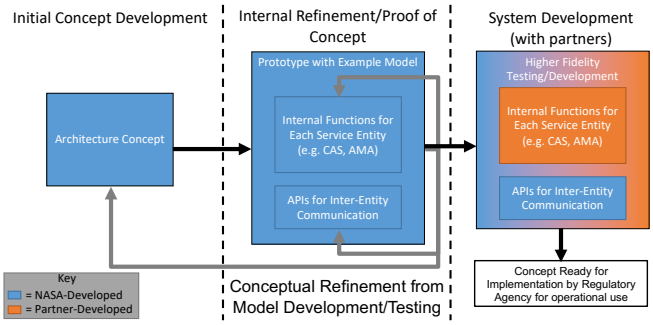


Figure 7: Development cycle of the STM prototype from the proposed architecture on the left (past work), to a software prototype which uses sample models or examples developed in-house as service entities in the middle (current state), to a higher fidelity software prototype which uses partners to represent service entities (future work)

The STM architecture and standardization of interaction between entities paves the way for a research ecosystem similar to other AI/autonomy fields, such as UAS or tele-operated cars. Some planning and scheduling scenarios are listed below in terms of increasing complexity:

1. Single maneuverable satellite to avoid a conjunction with an uncooperative target

2. Single maneuverable satellite to avoid conjunction with a single maneuverable satellite, both controlled by the same S3
3. Multiple maneuverable satellites to avoid sequential conjunctions with multiple maneuverable satellites, all controlled by the same S3
4. Multiple maneuverable satellites to avoid sequential conjunctions with multiple maneuverable satellites, controlled by different S3s

We applied the STM prototype to a multi-satellite scenario that required sequential COLA actions to prevent frequent 'cascading' conjunctions (#3). The scheduling algorithm in our example AMA is capable of handling scenarios #1, #2 and #3, and we discuss the negotiation implications required to extend to #4. We also present an example reward function and an example planning algorithm to schedule the COLA maneuvers across multiple satellites. These simple examples are expected to serve as a strawman for future, high fidelity STM services. While this paper focused on CAS and COLA related applications of autonomy within the STM , there is more scope for AI/autonomy development for other aspects of STM such as validating and merging SSA catalogs, decision-making as a function of uncertainty in data unavailability, and COLA as a function of varying levels of data sharing (Ref [9] lists five levels).

Acknowledgments

This project was sponsored by NASA Aeronautics Research Mission Directorate. The authors would like to thank Dr. Joseph Rios and Priya Venkatesh at NASA ARC, and Prof. Mykel Kochenderfer and his student, Nolan Johnson, at Stanford University for useful discussions on the topics presented.

References

[1] W. Ailor, G. Peterson, J. Womack, and M. Youngs, "Effect of large constellations on lifetime of satellites in low earth orbits," *J. Space Saf. Eng.*, vol. 4, no. 3, pp. 117–123, Sep. 2017.

[2] G. Peterson, M. Sorge, and W. Ailor, "New Space Activities and Implications for Space Traffic Management and Conjunction Assessment," Aerospace Corporation, 2018.

[3] J. A. Haimerl and G. P. Fonder, "Space fence system overview," in *Proceedings of the Advanced Maui Optical and Space Surveillance Technology Conference*, Maui, Hawaii, USA, 2015.

[4] C. Contant-Jorgenson, P. Lála, and K.-U. Schrogl, "The IAA cosmic study on space traffic management," *Space Policy*, vol. 22, no. 4, pp. 283–288, 2006.

[5] K. Bilimoria and R. Krieger, "Slot architecture for separating satellites in sun-synchronous orbits," in *AIAA SPACE 2011 Conference & Exposition*, 2011, p. 7184.

[6] C. R. Phipps and C. Bonnal, "A spaceborne, pulsed UV laser system for re-entering or nudging LEO debris, and re-orbiting GEO debris," *Acta Astronaut.*, vol. 118, pp. 224–236, 2016.

[7] J. Mason, J. Stupl, W. Marshall, and C. Levit, "Orbital debris–debris collision avoidance," *Adv. Space Res.*, vol. 48, no. 10, pp. 1643–1655, 2011.

[8] M. A. Skinner, M. K. Jah, D. McKnight, D. Howard, D. Murakami, and K.-U. Schrogl, "Results of the international association for the advancement of space safety space traffic management working group," *J. Space Saf. Eng.*, 2019.

[9] S. Nag, D. Murakami, M. Lifson, and P. Kopardekar, "System Autonomy for Space Traffic Management," in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, London, United Kingdom, 2018, pp. 1–10.

[10] D. D. Murakami, S. Nag, M. Lifson, and P. H. Kopardekar, "Space Traffic Management with a NASA UAS Traffic Management (UTM) Inspired Architecture," in *AIAA Scitech 2019 Forum*, San Diego CA, 2019, p. 2004.

[11] P. Kopardekar, J. Rios, T. Prevot, M. Johnson, J. Jung, and J. Robinson, "Unmanned aircraft system traffic management (utm) concept of operations," in *16th AIAA Aviation Technology, Integration, and Operations Conference, AIAA Aviation*, 2016.

[12] J. Thönes, "Microservices," *IEEE Softw.*, vol. 32, no. 1, pp. 116–116, 2015.

[13] N. Dragoni *et al.*, "Microservices: yesterday, today, and tomorrow," in *Present and ulterior software engineering*, Springer, 2017, pp. 195–216.

[14] J. V. Cabrera, S. Nag, and D. D. Murakami, "An Initial Analysis of Automating Conjunction Assessment and Collision Avoidance Planning in Space Traffic Management," presented at the AAAS Spaceflight Mechanics Conference, Maui, Hawaii, USA, 2019.

[15] Federal Communications Commission, "Space Exploration Holdings, LLC Request for Modification of the Authorization for the SpaceX NGSO Satellite System." 26-Apr-2019.

[16] C. Bombardelli and J. Hernando-Ayuso, "Optimal impulsive collision avoidance in low earth orbit," *J. Guid. Control Dyn.*, vol. 38, no. 2, pp. 217–225, 2015.

[17] S. Nag *et al.*, "Autonomous Scheduling of Agile Spacecraft Constellations with Delay Tolerant Networking for Reactive Imaging," in *International Conference on Planning and Scheduling (ICAPS) SPARK Applications Workshop*, Berkeley, California, U.S.A., 2019.