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

SpaceOps-2023, ID # 414

Innovative Web-Based Technologies for the Next Generation of Flight Dynamics Systems

M. Rasotto[a]*, J. Cruz[b]

[a] Flight Dynamics Engineer, *Terma B.V., Schuttersveld 9, 2316 XG, Leiden, The Netherlands*, mras@terma.com
[b] Software Engineer, *Terma GmbH, Bratustraße 7, 64293 Darmstadt, Germany*, jotc@terma.com
* Corresponding Author

Abstract

The explosion of the commercial space market, with start-ups and space ventures, is one of the most visible trends in space in the last few years. Fuelled by the first initiatives related to enhancing space accessibility and the growing use of small satellites, space activities have attracted an increasing number of players with intrinsically different needs, budgets, and scales, and raised the demand for innovative solutions for ground operations support, and in particular for new commercial ground systems. A key component of every satellite ground system is represented by the Flight Dynamics System (FDS) which often relies on libraries or APIs based on relatively old technologies, complex installation and configuration procedures, high costs and duration for trainings and operations preparation, reduced support to remote operations and a low level of automation. This paper describes some innovative ideas concerning the system architecture and the technology stack used to build a FDS that satisfies the key requirements of flexibility, scalability, and automated operations, and ultimately to meet the new customers' needs.

To better illustrate the technical concepts and the associated benefits, the above-mentioned ideas are contextualized by presenting the new FDS developed by Terma, ORBIT.

Keywords: Flight Dynamics System, Flight Dynamics operations, ground segment software, automation, web application, software technologies.

Acronyms/Abbreviations

Application Programming Interface (API)
Consultative Committee for Space Data Systems (CCSDS)
European Space Agency (ESA)
European Space Operations Centre (ESOC)
Flight Dynamics (FD)
Flight Dynamics System (FDS)
File Transfer Protocol (FTP)
Graphical User Interface (GUI)
Input/Output (I/O)
Improved Inter-Range Vectors (IIRV)
Lightweight Directory Access Protocol (LDAP)
Launch and Early Orbit Phase (LEOP)
Mission Control System (MCS)
Mission Planning System (MPS)
Neural Autonomic Transport System (NATS)
Orbit Ephemeris Message (OEM)
Out of Limit (OOL)
Orbit Parameters Message (OPM)
Pressure Volume Temperature (PVT)
REpresentational State Transfer (REST)
Secure File Transfer Protocol (SFTP)
Satellite Tracking Data Message (STDM)
Telecommand (TC)
Two Line Elements (TLE)
Telemetry (TM)
Tracking, Telemetry and Control (TT&C)
Uniform Resource Locator (URL)

1. Introduction

The typical goal for a FDS is to be able to model and manage the satellite's orbit and attitude during its whole lifetime. This means that it shall provide support to flight dynamics operations throughout all mission phases and therefore be able to perform tasks such as:

- Management of environment model files, e.g., leap second file, ionospheric data, solar activity forecast, ground station meteorological files, etc.
- Raw data pre-processing, to filter raw data coming either from spacecraft TM or external sources (e.g., TT&C stations), apply calibrations and all the required corrections before usage.

- Orbit Determination, to estimate the spacecraft orbit in a high-fidelity model, based on the set of radiometric (range and range-rate) and angular measurements (azimuth/elevation angles) received from TT&C stations during the visibility contact windows or based on the received on-board navigation solutions.
- Orbit Propagation, to generate the orbit ephemeris, i.e., a table of the predicted spacecraft state components as a function of time, based on a fully configurable dynamics model and including Earth gravitational potential, 3-body perturbations, atmospheric drag, solar radiation pressure, solid and ocean tides and Earth reflected radiation pressure, and the most up-to-date Earth Orientation Parameters provided by the International Earth Rotation Service (IERS).
- Attitude Determination, to monitor the current spacecraft attitude based on spacecraft telemetry. A set of attitude sensor biases may be estimated in the monitoring process depending on the space segment design.
- Attitude Propagation, to propagate the satellite attitude at each epoch. Note that the propagation is typically geometrical and not dynamical, meaning that the computations are performed based on the satellite position, a reference attitude mode/law, and a set of biases.
- Propellant Gauging, to estimate the on-board propellant mass and tank pressures based on different algorithms (e.g., PVT, pulse-counting).
- Manoeuvre Planning, to compute the set of manoeuvres required by each spacecraft to maintain its correct orbit within the constellation (routine operations) or to compute the manoeuvres needed to transfer a satellite from its injection point to the final constellation target (LEOP operations). An Orbit Manoeuvre Report is typically generated, collecting the manoeuvre information such as the delta V, the thruster branch to be used, the total time of firing, the calibration factors, the tanks' pressure before the manoeuvre, the fuel mass, etc.
- Product Generation, to generate additional FD products based on the predicted orbit and attitude to be disseminated to other facilities, such as:
    - Events file, with information regarding times of several orbital events such as node crossings, altitude crossings, perigee/apogee passages, eclipses, station visibilities, etc.
    - Station tracking elements, which provides information used by ground stations to track the satellite (e.g., STDM, TLE, IIRV, etc.)
    - Orbit ephemeris files, containing the information regarding the spacecraft orbit (e.g., CCSDS-OPM, CCSDS-OEM, TLE, etc.).
    - Reference ground track and ground track deviation files, containing the deviation of the spacecraft with respect the reference ground track.
    - Time Offset Value (TOV) estimation, containing the time which when applied as a time offset to an input orbit file, minimizes the angular distance with a reference orbit file.
    - Commands parameters file, containing the value of the parameters required by the commands' sequences.
    - Collision Warning report, containing information regarding the results of the conjunction analysis process.

The functional architecture of a FDS is therefore summarized in Fig. 1.

Despite the previous list of FD tasks being well known, intrinsically different needs, budgets, and scales of new space companies, lead to different missions' configurations, involving one or multiple satellites, different orbital regimes, different sensors/actuators, and different performance requirements. A modern commercial FDS should therefore be flexible enough to account for all these and be able to scale up to support satellite constellations. In these situations, parallelization is surely important to minimize the overall run time. However, even with parallel execution, operations for satellite constellations could require significant effort, if they require operator intervention. For these reasons, together with parallel task execution support, the automation of routine processes for the generation of products, the analysis of the results and the detection of deviations from the expected satellite behaviour is another crucial aspect for the next generation of FDS.

In the rest of the paper, we will first review and identify the software technology stack that could be used to build such a system (Section 2), followed by the description of the proposed system design and decomposition which ensures the satisfaction of the key requirements of flexibility, scalability, and automated operations, previously mentioned. For the sake of clarity, the FDS developed by Terma, ORBIT, is used as example in Section 4. Finally,

we will summarize the lessons learned and report the foreseen possible improvements and future development activities (Section 5).

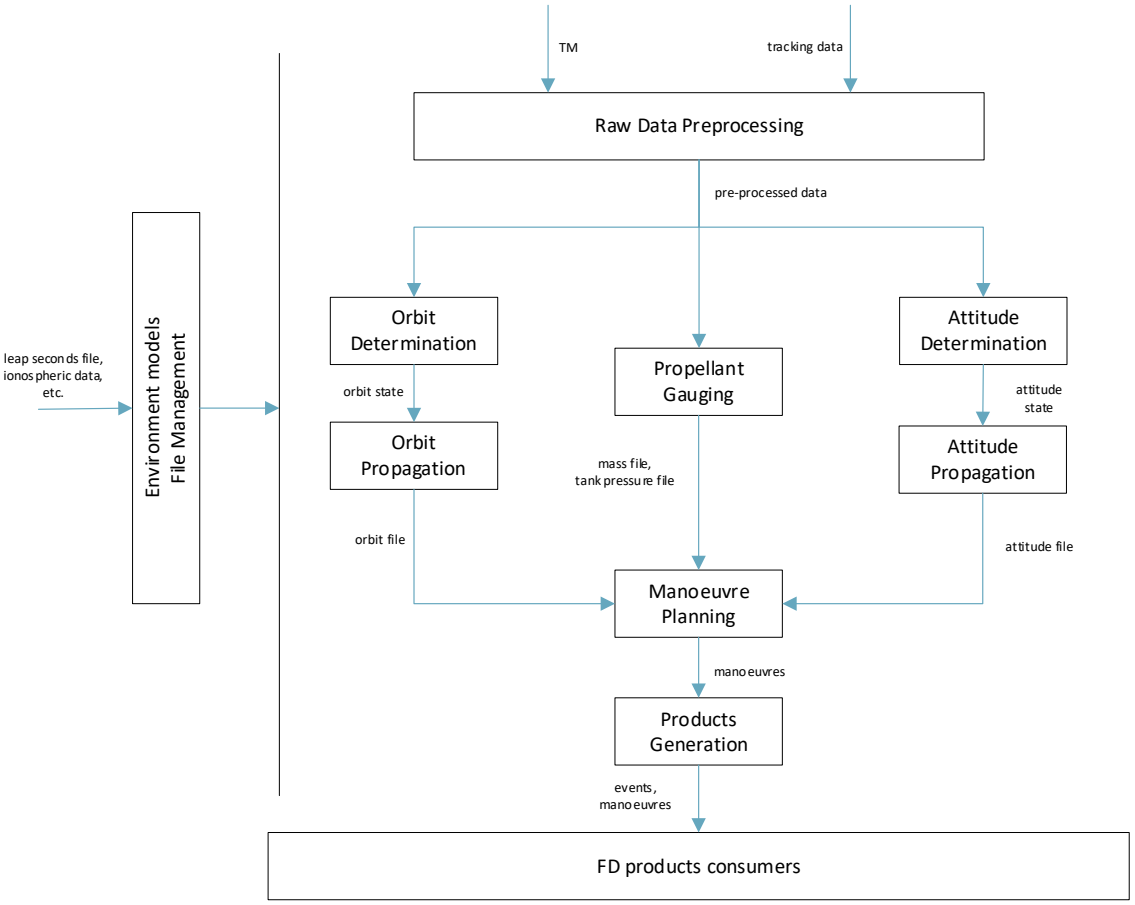

Fig. 1. FDS functional architecture.

2. Software Technologies

2.1 Astrodynamics Libraries/Tools

As described in Section 1, the FDS shall be able to perform several tasks that involve astrodynamics computations (e.g., reference frame transformation, orbit propagation, etc.). It is therefore clear that a FDS must rely on an astrodynamics library/tool, which implements all these low-level calculations. The main requirements here are: performance and computational efficiency, robustness, portability, and completeness. Several astrodynamics libraries are available as summarized below:

- **Orekit** [1]. Developed in Java (cross-platform), Orekit is a space dynamics open source library. It depends only on the Java Standard Edition version 8 (or above) and Hipparchus [2]. Orekit aims at providing accurate and efficient low-level components for the development of flight dynamics applications. It is designed to be easily used in very different contexts, from quick studies up to critical operations. Its main features are time frame conversions, reference frames conversions, orbit determination (batch least-squares and sequential filters), orbit propagation (analytical, numerical, or semi-analytical methods), event handling, attitude, manoeuvres and force models. Orekit is widely adopted and extensively used and validated [3, 4].
- **GODOT** [5]. GODOT is a collection of libraries written in C++11 by the ESA/ESOC flight dynamics team aimed at performing a wide range of orbit-related flight dynamics tasks, such as orbit propagation, orbit estimation and trajectory optimisation, in support to mission operations for different types of space missions. GODOT C++ libraries and classes are complemented by Python wrappers, which can be used to create Python applications as well.

- **TUDAT** [6]. A set of C++ software libraries, developed and maintained by staff and students in the Astrodynamics & Space Missions research group at the Faculty of Aerospace Engineering, Delft University of Technology, in the Netherlands. This toolbox is intended to provide users with functionalities to be able to simulate various astrodynamics applications.

In addition to these, other tools/software packages are available such as:

- **Satellite Tool Kit (STK)**: developed by Ansys, it provides four-dimensional modelling, simulation, and analysis of objects from land, sea, air, and space in order to evaluate system performance (see [7]).
- **General Mission Analysis Tool (GMAT)**: developed by a team of NASA, private industry, and public and private contributors, GMAT is an open-source space mission design tool used for real-world engineering studies, as a tool for education and public engagement, and to fly operational spacecraft (see [8]).
- **FreeFlyer**, developed by A.I. Solutions, FreeFlyer is software for space mission design, analysis and operations. It provides an astrodynamics functionality for mission analysis (see [9]).

The main drawback of these tools is that they require a commercial license (STK and FreeFlyer) which might have a non-negligible impact on the final cost of the FDS. GMAT on the other hand, although open source and compatible with commercial use, seems to provide only prototype APIs, as they lack online documentation and are characterized by apparent inconsistencies.

The choice is thus restricted to the first three libraries. Being based on compiled programming language, GODOT and TUDAT are typically characterized by higher computational performances. Orekit, on the other hand, provides cross platform compatibility, is very mature and robust, also thanks to a well-established community of developers and users, which ensures fast replies and bug fixes implementation. Knowing this, the choice might also depend on other factors, such as the availability of in-house experience and background. In our case, the more sensible option was Orekit, also because the low level of maturity of GODOT at the time the implementation started.

2.2 Web-based Technologies

Nowadays more and more software are built using web based technologies and in general adopting a web-based approach, in which the application program is stored on a remote server and delivered through a browser interface. Some common benefits of web applications include:

- Faster and easier development process, since they are built once for all the platforms
- Simpler updates management, as only updates to the web server that runs the web app are required
- Support for multiple users
- Easier software deployment, no need to install and configure the application on each single workstation
- Platform and operative system independent, as they require only a web browser
- Intrinsic support for remote or even cloud operations
- Support for multiple monitors, through the usage of browsers tabs
- Better information sharing, since the centralization of data and calculations intrinsic in this architecture allows a new product produced and validated by an operator to be immediately available to all operators working on the same mission and with the proper visibility rights

Web applications are often separated in two parts: the backend and the frontend. The first one is typically the part in which the requested tasks and operations are performed. The frontend instead is what users see and interact with such as the GUI. In the following subsections we present the main technologies and frameworks for backend and frontend, respectively.

2.2.1 Backend Frameworks

The main backend frameworks are:

- **Spring Boot**: Spring Boot is an open-source Java-based framework developed by Pivotal Team and used to build stand-alone and production ready spring applications. It provides support to REST APIs and includes an embedded Tomcat server and an SQL database.

- **Django**: Django is a high-level, open-source, Python web framework. Django has been gaining popularity for its simplicity, ease of use, pragmatic design, yet fully featured compared to many other frameworks. It is also compatible with most major databases and inherits all of Python's benefits, such as great support, productivity boost, and advanced development speed.
- **Node.js***:* Node.js is an open-source, cross-platform, back-end, JavaScript runtime environment for writing server-side applications using JavaScript. Node.js is usually used for non-blocking, event-driven servers for traditional websites and back-end API services.
- **Flask:** Flask is a Python-based micro web framework that does not require specific libraries and tools. This backend does not have form validation, a database abstraction layer, or components that require functions from external sources.

While any of the above-mentioned options can be used, it is natural to select a framework that is written in the same language as the selected low level astrodynamics library. This is particularly important in order to avoid the typical overheads that occur when using different programming languages. Therefore, in our case, having selected Orekit, and therefore Java, the wiser option was represented by Spring Boot.

2.2.2 Frontend Frameworks

The main frontend frameworks are:

- **Angular**: Angular is a TypeScript-based, open-source, front-end mobile and web application framework developed by Google in 2010. Angular is a platform and framework for building dynamic single-page client applications through HTML and TypeScript.
- **React**: React is an open-source, front-end JavaScript library for creating interactive UIs. React is developed and maintained by Facebook and a large community of dedicated developers. React can also be used as a base for a single-page or mobile application.
- **Vue.js**: Vue.js is an open-source JavaScript framework for building web UIs and single-page applications. Vue.js can be used for both desktop and mobile app development.
- **Blazor**: Blazor is a free, powerful, open-source C# web framework for creating web apps using C# and HTML, developed by Microsoft. Blazor runs C# code directly in the browser using WebAssembly and gives you access to .NET on the client-side for fast and rich web applications without the need for JavaScript.
- **WebAssembly**: WebAssembly is a new type of code that can be run in web browsers – a low-level assembly-like language that is a portable binary-code format for executable programs. In short, WebAssembly is a new extension for your web browser that lets you run precompiled code fast. WebAssembly lets you write code in any programming language and lets other people run that code on any platform without installing anything.

After an initial review, both Angular and React were considered good candidates, whereas the others have been discarded because of their maturity level, lack of popularity, or complexity. Angular was finally selected despite its steeper learning curve since it allows producing feature-rich enterprise-level solutions. Again, the available in-house expertise also played an important role in the selection process.

3. System Design

3.1 System Architecture

The system can be decomposed as presented in Fig. 2. The main components are:

- Server, the backend module based on Spring Boot and providing support for REST APIs. The backend provides all the astrodynamics functionalities and is organized in FD activities, each covering one of the tasks reported in Section 1. This ensures software modularity and flexibility, thus simplifying the implementation process of new functions (e.g., mission specific FD computations) and reducing the cost for maintaining those already existing. The backend also embeds an internal SQL database, used to store configurations, input files, as well as all the generated FD products. Thanks to the usage of Spring Boot, the backend is also equipped with an internal Tomcat server, which is used to serve the REST endpoints and to trigger activities execution or download products. An authentication module provides the means to

configure access to the application over the LDAP protocol. A scheduler tool is also available and used for automating procedures. Finally, the server includes support for scripting language (e.g. Python), allowing users to generate plots as well as define their own activities and generate additional FD products based on the orbit and attitude information contained in the database.

- Client, the frontend module based on Angular and providing support to the operator thanks to the generation of a modern, intuitive, and dynamic interface, the ingestion, preparation and forwarding of the user's inputs to the backend side and the visualization of the generated results. It can be easily reached via any web browser, enabling remote or even on-cloud operations and the possibility to interface with the backend from any type of device (i.e., desktop, laptop, tablet or even smartphone) or operating system. The interface with the backend module is managed via REST APIs.

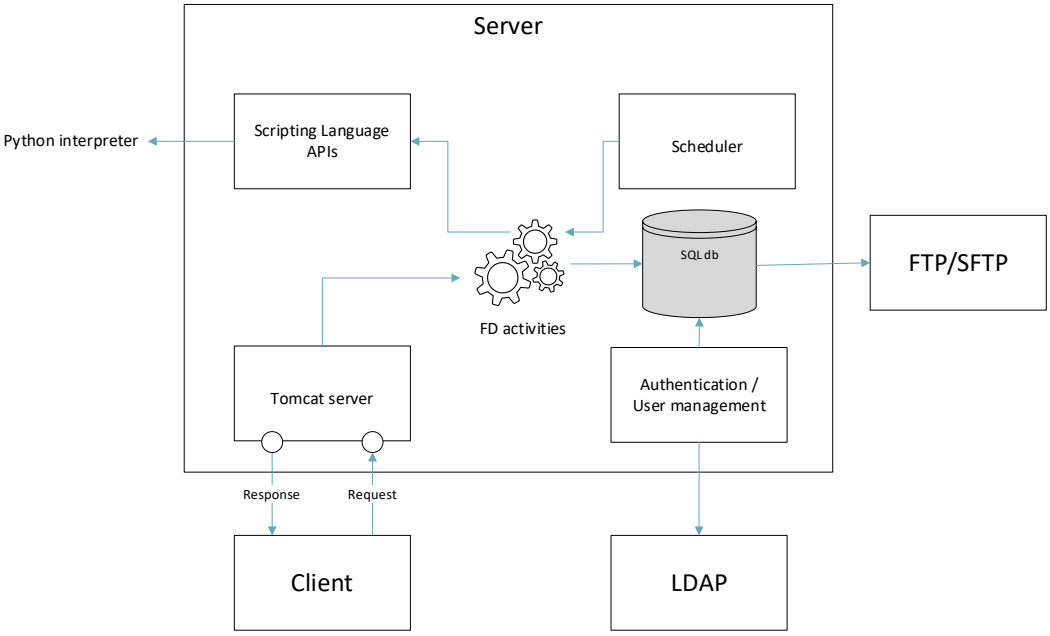

Fig. 2. FDS physical architecture.

As shown, the system relies on some additional dependencies:

- LDAP, to manage operators' identities via LDAP protocol.
- FTP/SFTP, to manage connections towards external facilities via FTP/SFTP protocol.

For both cases, client libraries are included in the system, but the server is not included and must be installed separately.

3.2 General FDS workflow

As briefly described above, the execution of any FD activity is controlled via REST protocol. The main advantage of this approach is that the REST protocol is based on HTTP and, as such, is fully compatible with web browsers, and largely used in web development. Also, unlike other protocols, there is no third-party library needed, but it is fully supported by Spring Boot.

Fig. 3 reports the sequence diagram illustrating the execution workflow of a generic FD activity. The execution can be initiated manually (via a web-based UI or via terminal/command line), or automatically (using a scheduling tool). An HTTP request is created with a predefined URL and with the required configuration and input files as payload. A REST controller, which exposes the predefined REST endpoint/URL is thus triggered, which in turns starts the execution of the activity in a dedicated thread. Note that in case multiple satellites are selected, the activity execution is scaled and distributed across multiple threads. The execution usually starts with a query to the SQL database to retrieve the information needed (e.g., configuration files and input products). Once the execution is complete, the generated products are stored locally on the server storage system and the corresponding file path is

added to the database to ensure future accessibility. A response is also created and passed back to the controller and finally to the UI, providing feedback to the user on the status of the activity. In order to maintain the record of the executed activities and their status, the response is also stored in the database.

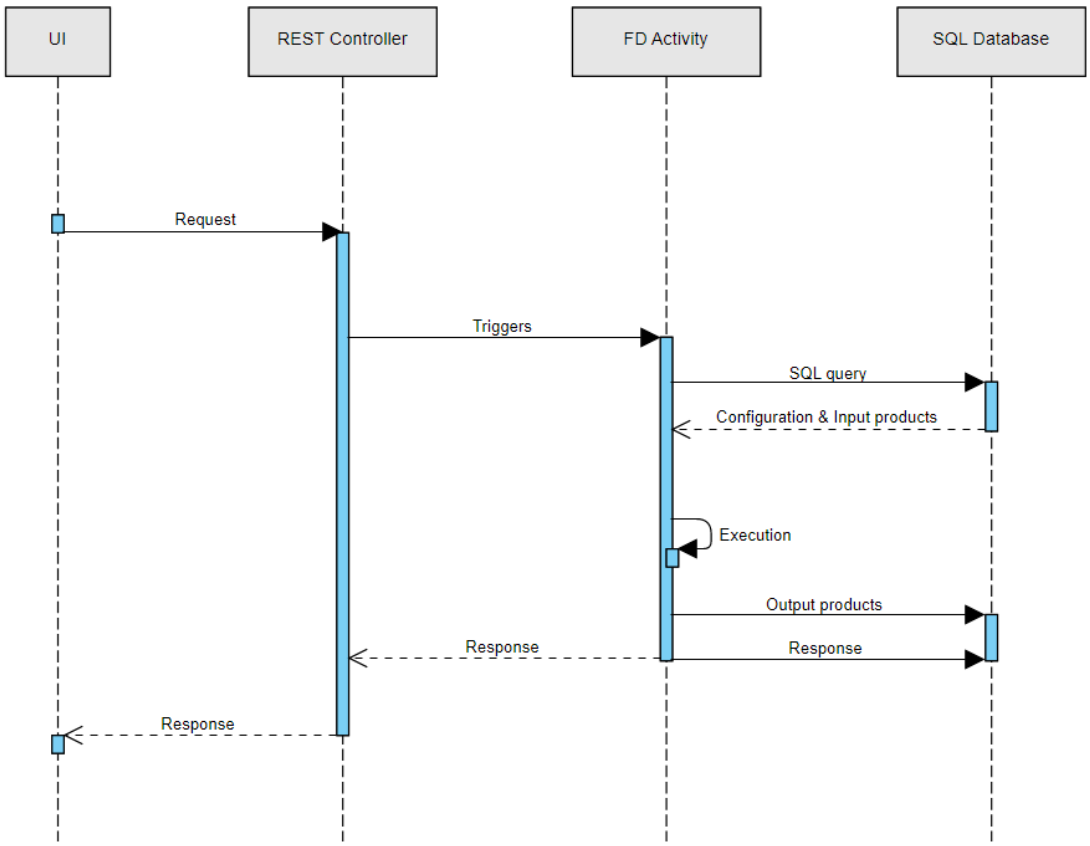

Fig. 3. Sequence diagram for FD activities execution.

It is worth noting how this approach allows:

- End-to-end FD operations to be covered by sequentially executing a set of activities.
- Execution of multiple FD activities by multiple users.
- Support to multiple satellites.

3.3 Interfaces

As part of the ground segment software, the system has a number of interfaces, as shown in Fig. 4:

- The MCS, which provides the spacecraft telemetry, both offline (via SQL queries in the archive database) or in real-time (e.g., via NATS). The FDS in turn provides the MCS with the Propagation Delay File and the Task Parameter Files with the computed TC parameters (e.g., manoeuvre command parameters, parameters to update the on-board orbit propagator, sensor inhibition commanding parameters, etc.).
- The TT&C stations, which provide the results of the ranging activities. The FDS generates the tracking files (e.g., STDM files) with information needed by the stations for the ranging session (typically orbit vector or tracking angles). The ranging sessions are initiated by the MCS, which sends the request with the detailed parameters provided by the FDS. Once the ranging campaign is completed, the TT&C stations send to the FDS the obtained measurements, calibration and meteo data (e.g., CCSDS TDM files).

- The MPS, which receives the following data:
  - Orbit and attitude events, including eclipse & sensor inhibition times, station visibility times.
  - Planning requests containing information for the requested task (e.g., manoeuvre execution, sensor inhibition etc.).

  The MPS generates a request response and uploads to the FDS via REST APIs.
- Third party servers, from which environment model files are received (e.g., leap seconds file, ionospheric data files, IERS bulletin, etc.). Note that the file transfer occurs over FTP/SFTP protocol and nominal and backup server can be configured.
- The Collision Service provider, which provides collision warnings in terms of Collision Data Messages (CDMs), with information regarding the closest approach and the probability of collision. Upon ingestion, the FDS evaluates the risks and, if needed, plans for collision avoidance manoeuvres.

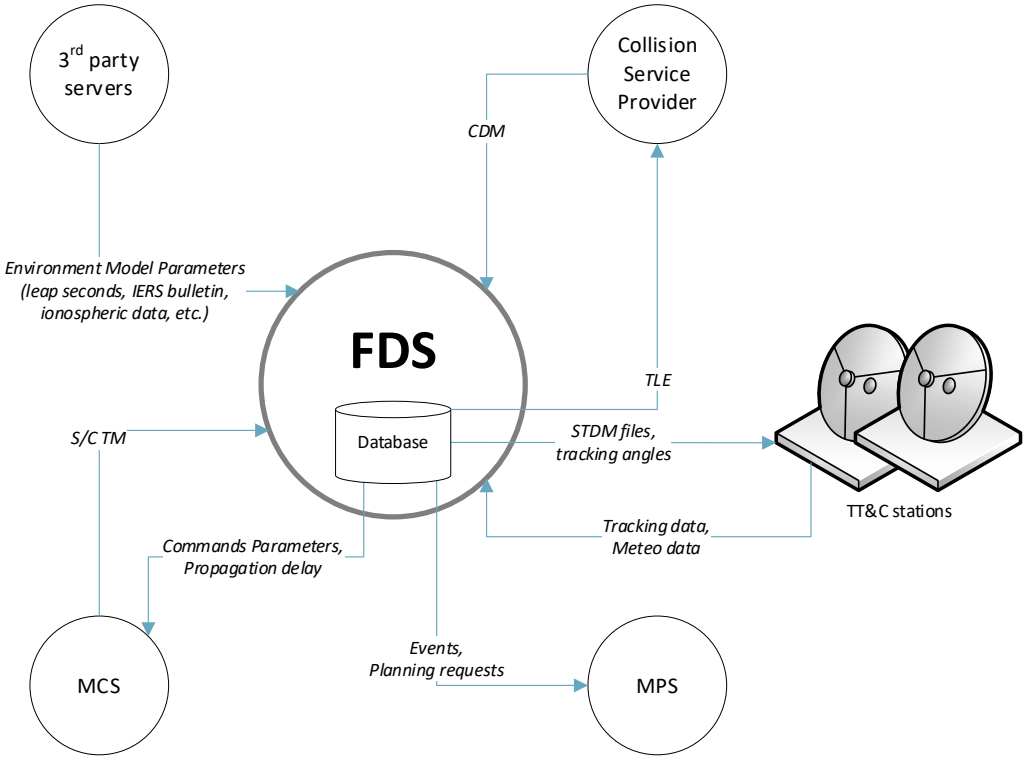

Fig. 4. FDS Interfaces.

3.4 Automation

Thanks to the usage of REST APIs, automated procedures can easily be implemented (e.g., via Python scripts or any other language supporting REST APIs) for the generation of routine products, for analysing the results as soon as they become available and for detecting when a spacecraft deviates from its expected behaviour. Examples of automatic validations may include, checksums, algorithm convergence checks and OOL checks.

The periodic update of environment files (from third party servers) can also be automated, for instance via the embedded scheduling tool by setting the required update frequency (e.g., daily, monthly, etc.) in the application configuration.

Finally, automation between modules of the ground segment suite is also achieved using monitoring services that detect whenever certain files in a configurable system location has been updated/created (e.g., a new CCSDS TDM file has been sent by the TT&C stations).

4 Terma FDS: ORBIT

The above-mentioned concepts have successfully been applied by Terma in the development of ORBIT, the flight dynamics software package of the Terma Ground Segment Suite (TGSS). ORBIT is a web-based modern implementation of a classical FDS, providing the general functionalities such as orbit determination, orbit propagation, event prediction, manoeuvre planning and fuel bookkeeping, but also some more advanced features such as the support to electric and hybrid propulsion. As explained in the previous sections, it uses web-based technologies and a modular architecture that makes it innovative, portable, secured, and extensible. It is designed to support operations for LEO missions with multiple spacecraft and automated operations scenarios.

Fig. 5 illustrates the ORBIT's home page.

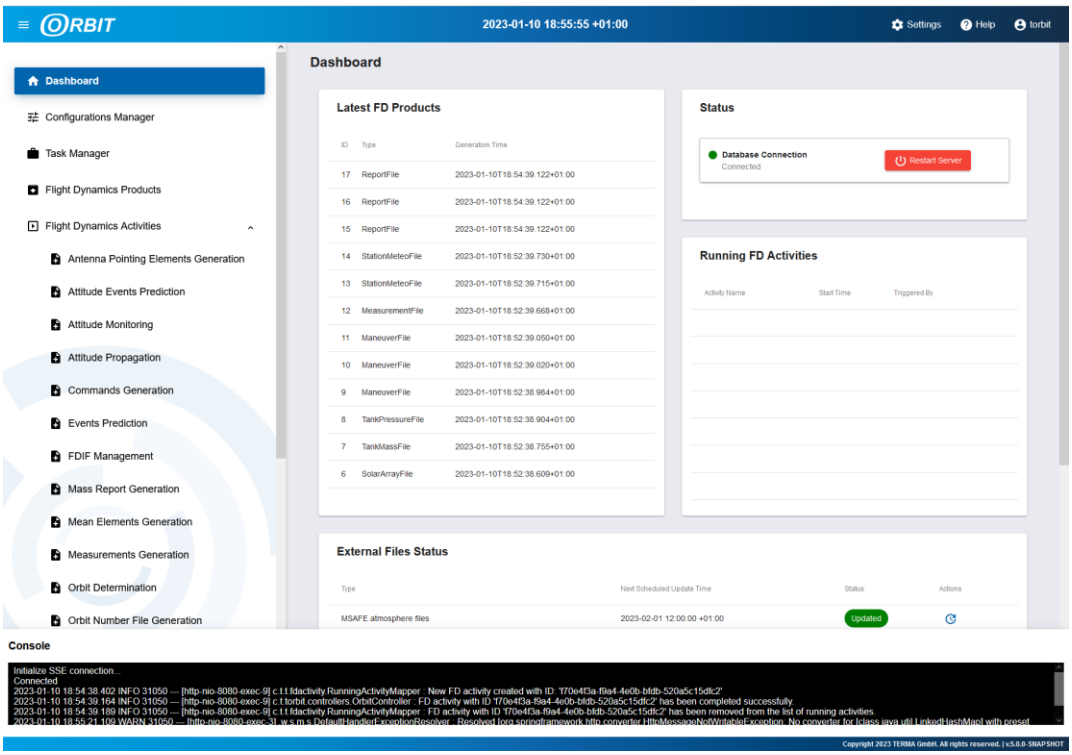

Fig. 5. ORBIT user interface.

5 Conclusions

The technology stack and the system architecture that could be used to build a modern FDS have been presented. In particular the choice of using an open-source library such as Orekit for the astrodynamics computations, ensures compatibility with commercial applications, portability to any operating system and great robustness, thanks to the large community of users and developers.

The adoption of a web-based architecture proved to be essential to simplify and speed up the development, guarantee portability to any platform and enable remote or even cloud operations. The chosen backend and frontend frameworks, Spring Boot and Angular, respectively, include all the necessary tools to implement enterprise applications and embed essential features such as the authentication layer, the internal SQL database, and the scheduling tool for automatic operations.

Moreover, the split of the functionalities in separate FD activities ensures modularity, thus reducing the cost of maintenance and enabling the extension and integration of new functions (e.g., covering mission specific needs).

These concepts have successfully been implemented by Terma in ORBIT, a modern FDS, part of the Terma Ground Segment Suite (TGSS). ORBIT is now supporting Low Earth's Orbits mission and satellites with a chemical, electrical or hybrid propulsion systems. In the future, support to geostationary satellites will be added.

Acknowledgements

The authors would like to thank Ahmed Elmaghraby, Gaetan Geffroy, Francesco Lupi, Giuseppe Russo and Omiros Papadatos Vasilakis for their invaluable contributions to the development of ORBIT.

References

[1] CS Systemes d'Information et al. Orekit. https://www.orekit.org/, (accessed: 2023-01-09).
[2] Hipparchus site. https://hipparchus.org/, (accessed: 2023-01-09).
[3] SOCIS the ESA Summer of Code in Space. https://www.esa.int/Enabling_Support/Space_Engineering_Technology/SOCIS_The_ESA_Summer_of_Code_in_Space, (accessed: 2023-01-10).
[4] Bernard, N., Maisonobe, L., Barbulescu, L., Scortan, S., Cefola, P. J., Casasco, M., and Merz, K., "Validating Short Periodics Contributions in a Draper Semi-Analytical Satellite Theory Implementation: the Orekit Example," ISSFD, 2015.
[5] ESA. GODOT site. https://godot.io.esa.int/docs/1.0.0/, (accessed: 2023-01-10)
[6] TU Delft. TU Delft Astrodynamics Toolbox site. https://docs.tudat.space/en/stable/, (accessed: 2023-01-09).
[7] Ansys. Systems Tool Kit (STK) site. https://www.ansys.com/products/missions/ansys-stk, (accessed: 2023-01-10).
[8] NASA. GMAT site. https://sourceforge.net/projects/gmat/, (accessed 2023-01-10).
[9] A.I. Solutions. FreeFlyer site. https://ai-solutions.com/freeflyer-astrodynamic-software/, (accessed 2023-01-10).