SpaceOps-2023, ID # 464

A Modern Approach to Visualise Structured and Unstructured Space Missions' Data

Agnese Del Moro[a*], Matthias Dauth[a], Tobias Lesch[a], Clemens Schefels[a], Armin Braun[a], Vlad Filip[b], Tobias Göttfert[a]

[a] Deutsches Zentrum fur Luft- und Raumfahrt, German Space Operations Centre (DLR-GSOC), Münchener Straße 20, 82234 Wessling, Germany
[b] Heavens-Above GmbH , Munich, Germany
* Corresponding Author, agnese.delmoro@dlr.de

Abstract

In this paper the Visualisation and Data Analysis (ViDA) project, currently being developed at the German Space Operations Center (GSOC), is presented. ViDA is a modern, interactive, web-based frontend tool designed to efficiently explore various types of data generated by space missions. It is more than just a telemetry display tool and, as such, includes features from business intelligence, data science and AI tools, while being focused on the multi-spacecraft operations use case. The paper describes how the big data challenges (volume, variety, variability, complexity, value) in the context of spacecraft operations have been addressed and how the adopted solutions have been integrated into ViDA. It also highlights the importance of contextual knowledge as crucial point for the design and implementation of ViDA. The techniques used for creating appropriate visual representations of the data and their relations are described. Such visualisations are specifically designed to deliver interpretable results to the users, thus helping them to quickly extract knowledge from them during their analytical process. Finally, the integration of ViDA into the ground system and its connections to the other tools in the telemetry/telecommand chain are discussed.

Keywords: (ViDA; ATHMoS; spacecraft operations; data visualisation; web application; AI integration)

Acronyms/Abbreviations

Anomaly Report (AR)
Artificial intelligence (AI)
Automated Telemetry Health Monitoring System (ATHMoS)
Business Intelligence (BI)
Database (DB)
Failure Detection, Isolation and Recovery (FDIR)
German Space Operation Center (GSOC)
High Priority (HP)
Hybrid Dynamic Downsampling (HDD)
Intrinsic Dimensionality (ID)
Low Priority (LP)
Machine Learning (ML)
Mission Operations Technology (MBT)
Mission Data Access (MiDA)
Monitoring and Control System (MCS)
On-board Event (OBE)
Out of Limit (OOL)
Outlier Probability via Intrinsic Dimensionality (OPVID)
Telecommand (TC)
Telemetry (TM)
Trending detection (TD)
Visualisation and Data Analysis (ViDA)

1. Introduction

As modern space- or aircraft become increasingly complex, the amount of data they produce has skyrocketed. Larger satellites generate, for instance, thousands of data samples every second just for providing information on its status and health condition. Handling and analysing the vast amount of data on ground is one of the major challenges for space operations in many respects. First of all, the satellite data need to be processed, archived, and made

available to the users, ensuring, at the same time, data security to avoid unauthorised access, corruption, or theft throughout this entire life-cycle. This typically translates in having to resort to on-premises solutions, rather than rely on Cloud resources as many major companies dealing with big data do. Moreover, besides the telemetry (TM) data, many other data products of different formats and nature are generated during operations, e.g., telecommand (TC) logs, on-board event (OBE) logs, anomaly reports (ARs), to name a few. These products end up being scattered in various storage systems and often accessed by diverse tools. Secondly, it is also essential to assist the operation engineers working with these data. As the manual handling and inspection of such large amounts of data are no longer feasible, the automation of telemetry inspection to a high degree is paramount.

Nowadays, many data visualisation and manipulation tools are available on the market to support easy data aggregation and graphics, in line with business intelligence requirements by companies dealing with big data. Although many of the features provided by these "off-the-shelf" software are desirable, these tools are not tailored to space operations and are not easily adaptable to support their specific needs.

To provide the engineers working with spacecraft with a comprehensive, single framework that eases the monitoring and analysis of satellite TM, the Visualisation and Data Analysis (ViDA) tool was conceived and developed at the German Space Operation Center (GSOC). The primary use cases for which ViDA is designed are: i) the support of multi-mission data access for monitoring the satellites' status daily, up to their full life timespan; ii) the integration of the Automated Telemetry Health Monitoring System (ATHMoS; [1]), a set of modern, semi-supervised machine-learning (ML) based algorithms that automatically detect novel behaviour or outliers in the TM data; iii) providing contextual information to the TM data by assembling various data sources, enriching the simple graphical views with context and explainability, to facilitate and speed up the engineers analysis process.

The ViDA framework, presented in this paper, is tailored to respond to the specific needs of spacecraft operations engineers and it is growing hand-in-hand with them, to become a forefront tool, essential for the users in their daily work. The paper is structured as follows: in Section 2 an overview of the ViDA framework is presented, together with the main features of the web interface and the backend. In Section 3 the technologies of choice, and the motivation for said choice are described. Section 4 presents the architecture and infrastructure composing the ViDA framework, while Section 5 describes the data services complementing our main ViDA application. Section 6 gives an overview of the ML-based novelty detection service (ATHMoS), which has been integrated to operations by our group and to which ViDA serves as the main interface to the users. In Section 7, the Mission Data Access (MiDA) library API provided to the users together with JupyterHub to complement the ViDA framework, are briefly described. Finally, the paper concludes with a discussion of future extensions and perspectives in Section 8.

2. The ViDA Framework and Features

One of the primary objectives of ViDA is to aid satellite operations engineers in inspecting a large number of TM data sets on a weekly, monthly and yearly basis, in order to assess the health status of the satellite. ViDA has several features designed to ease the inspection process. As an example, it provides highly customisable and user-specific telemetry overviews, which can contain the visualisations of a user-defined selection of important TM parameters. To further assist engineers, the overview integrates automated highlighting mechanisms to easily find outliers, novel behaviour or trends in the data sets. To perform the analysis required for the highlighting, data-mining methods such as statistical analyses and automated outliers or novelty detection methods are embedded in the ViDA framework. The latter include results from the ML-based novelty detection algorithms of ATHMoS [1], which is specifically developed for analysing large quantities of satellite TM data. Finding outliers or novelties is usually not the end of the investigation process. For satellite operators, it is essential to gain a detailed understanding of what causes a particular behaviour in the TM data to detect signs of hardware degradation as early as possible, thus preventing, or at least, getting enough knowledge to be able to act quickly on critical situations. To get a better understanding of the causes of outliers and TM in general, ViDA presents contextual information to the users. Contextual information about the spacecraft system state is built by aggregating data from various sources, such as the history of telecommands (TCs) executed, on-board events (OBEs) and out-of-limit (OOL) violations of TM parameter values.

An important feature of ViDA is the visualisation of data from different satellites, which allows the users to perform different types of comparisons, e.g. between similar satellites or fleet members. An example is stacked plots showing the same telemetry parameter for different satellites in the same time range. Another key aspect of ViDA is the support of life-time analysis for the spacecraft, being able to plot TM data over the entire mission lifetime almost instantaneously. Through the ViDA framework, users will have access to commonly used regression methods for data extrapolation. For extensive investigations related to life-time predictions or novelty analyses, which go beyond the pre-built analysis functionality of ViDA, the framework offers access to a powerful centralised data analysis environment, which is based on JupyterLab (see Sect. 7). To simplify the analyses, the JupyterLab instance delivered within the ViDA framework provides an API to easily access the TM and other data sources, called MiDA.

The ViDA framework consists of a central web-based user interface and underlying micro-services supplemented by the custom analysis application programmable interface. It is based on a modular service-oriented architecture, where each component is designed as an independent, controllable, scalable and robust micro-service with its own specific scope.

2.1 ViDA Web Interface

The user can conveniently access the application through a web browser from all machines that have access to the server on which the ViDA backend (Sect. 2.2) is running. By logging in with the user's credentials, each user can have access to the full functionality of ViDA. A personalised dashboard is the entry point to the application (Sect. 2.1.1) and gives an overview of the ViDA features. From the dashboard, as well as from the side bar, users can navigate to all components, such as the satellite-specific dashboard (Sect. 2.1.2), the daily/weekly telemetry overview (Autoplots, see Sect. 2.1.3), or the in-depth analysis space (Workspace, see Sect. 2.1.4). A detailed view of all the ViDA web application features available, or under development, is provided in the next sub-sections.

2.1.1 User's Dashboard

The user's dashboard is the first view a user is presented with after logging into the application (Fig.1). This feature is focused on the multi-mission aspect of operations. Indeed, the dashboard contains an overview of the status of all the satellites the user is interested in or is operating, including the time range for which the TM data are available in the database, the total number of high-priority (HP) and trending (TD) novelties detected in the last day or week by ATHMoS (see Sect. 6) and the general status of each subsystem. From this page the user can navigate to the satellite-specific dashboard, for a more detailed view of the detected novelties in each satellite and subsystem, access directly the "Autoplots" page of the selected satellite to inspect the daily TM overview, or access external data sources containing other useful operational information (e.g., ARs, TC logs, etc.). The basic structure and implementation of the user's dashboard are already in place, while the aggregated data information planned to be displayed as the status overview of each satellite is still under development.

Fig. 1. Design of the ViDA user's dashboard (showing dummy data).

2.1.2 Satellite-specific Dashboards

The satellite-specific dashboard (Fig.2) provides a more detailed overview of the status of the satellite and its subsystems, including a list of all the novelties detected in the selected time range, and a network graph linking all the TM parameters that are in close correlation and clustering them. Such graph allows the users to visually connect all parameters showing novel behaviour, in order to easily identify where the problem could be. This information can be enriched by a table view of all the events, system logs and TCs sent to the satellite in the same time range, and/or other user's specified views that can help him/her in performing a rapid inspection of the status of each subsystem. From this dashboard, by clicking on specific novelty detections listed in the table, the users can navigate to their

personal workspace (Sect 2.1.4) where plots of the selected parameters will be automatically loaded, and they can further investigate the novel behaviour to understand its origin and decide on the necessary course of action to fix the problem.

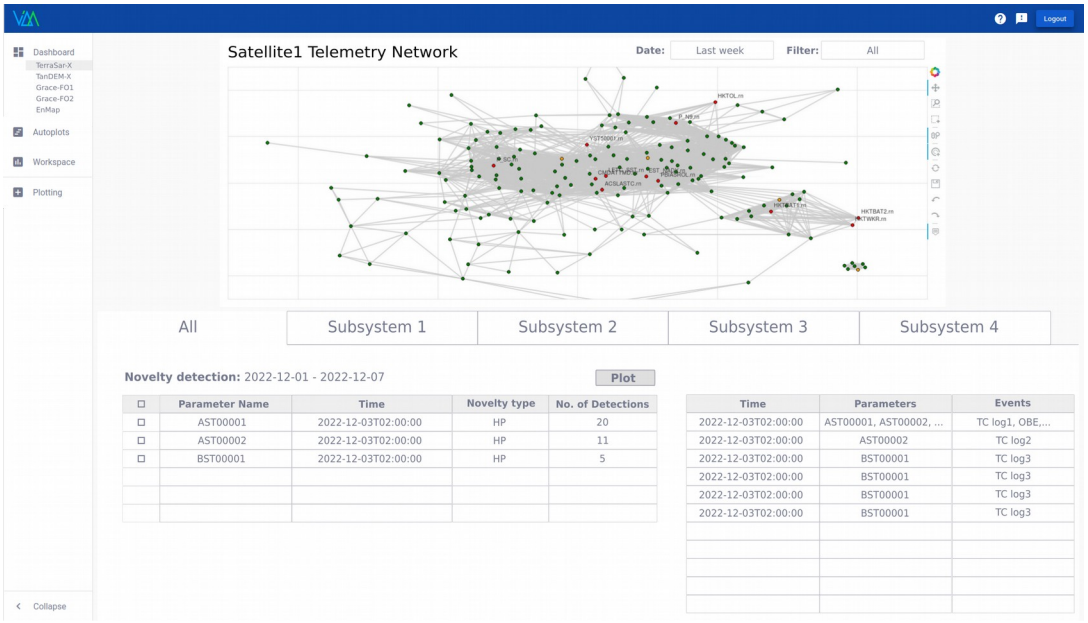

Fig. 2. Design concept of the ViDA satellite-specific dashboard (showing dummy data).

The satellite-specific dashboards are not yet available as a feature in ViDA as they are currently being developed. However, it is planned for them to be included in a new software release in the next 2-3 months.

2.1.3 Autoplots

The Autoplot page displays a set of predefined plots, typically showing the last day's, week's or cycle's data (Fig. 3), that the operation engineers inspect daily for their routine system checks. These plots are selectable per subsystem and can be navigated through in seconds for quick inspections. On the other hand, they are all interactive and can be rescaled and modified for a deeper visual inspection, if needed. The results of our ML novelty detection service are also displayed in the autoplots to ease the engineers job in identifying possible problems in the satellite's subsystems. The autoplot feature is already fully functional and provided to the users in the current release of ViDA.

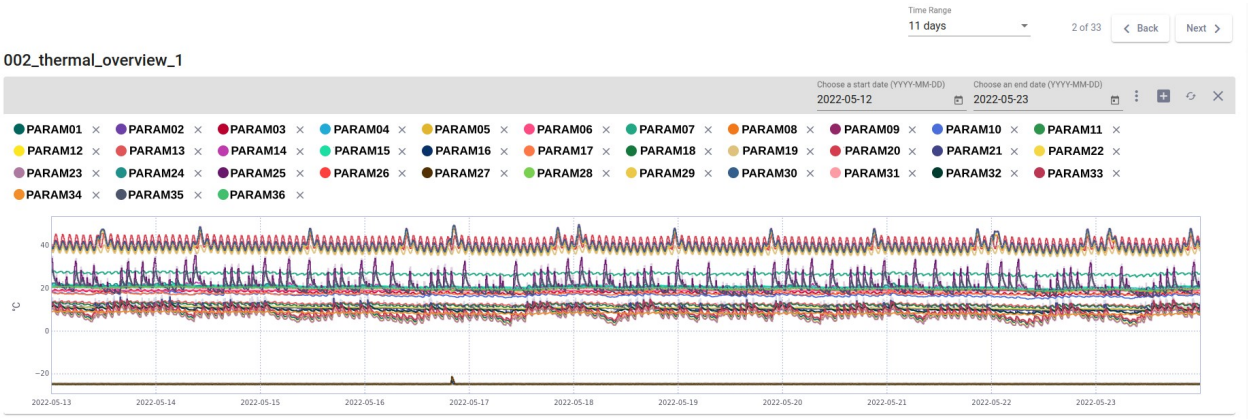

Fig. 3. Example screenshot of a ViDA autoplot with multiple parameters and relative legend displayed on top of the plot.

2.1.4 Workspaces

The workspace is a dedicated page where users can select and fully customise their TM plots to perform specific analyses on the data. Currently, single- and multi-parameter time-series plots are available as selectable views, but

the library of plot types will be extended in the near future. It is important to stress that the key features of ViDA are the possibility to display multi-mission and long-term TM, offering the users the ability to perform in-depth analyses over the whole satellites' lifetime. For example, inspecting novelties detected in previous years, or comparing the behaviour of the same parameters in different satellites (e.g., members of a fleet) are common use cases.

Fig. 4 Example screenshot of a Workspace with two multi-parameter plots, one of them also highlighting time periods where novelties were detected, as yellow (TD) and red (HP) vertical bars.

The workspaces, with all the custom plots currently displayed, can then be named, saved and stored in the database for the users to re-load and continue their analyses at a later time. Each saved workspace is only available to its owner. In the future it is foreseen to have a sharing functionality, for a saved workspace to be accessed by multiple users, when desired.

2.2 ViDA Backend

The ViDA backend is responsible for accessing the databases which hold satellite data and user preference data. To keep the frontend as slim as possible, the ViDA backend shoulders the resource demanding processing and serves preprocessed data to the frontend.

The backend provides a dedicated http endpoint to serve the web-frontend application using a GSOC public GraphQL endpoint (see detailed description in Section 3). Most parts of the GraphQL API are however restricted to authenticated and authorised users. Authorization, which is based on JSON Web Tokens sent in the web request headers, is also handled by the backend via the GraphQL API. A dedicated GSOC internal server is used for validating user credentials and retrieving the roles of the user. For each protected query, the backend filters the returned data according to the user's role or access rights.

Currently, the ViDA backend provides the connection to several satellite's DB servers, containing TM data, metadata information of the TM parameters and the novelty detection results from ATHMoS for each satellite. In the future it will also be enhanced to connect to further data sources to read out all necessary contextual input data (e.g., OBE, OOL, FDIR, TCs, etc.) from there.

3. Adopted Technologies

The ViDA web application is the central user interface to interact with the ViDA framework. In order to provide a smooth and convenient user-experience, the ViDA web application is based on state-of-the-art technologies, which are widely used and supported. ViDA is designed as a distributed web application with a server (backend) serving multiple clients (using the frontend of the ViDA application) to partition tasks and workload.

The frontend is based on the Angular framework [2], a development platform built on Typescript, which supports developing an interactive and reactive single-page application according to newest standards [3]. Using this foundation also gives a lot of flexibility for the design of the application. This freedom can be used to develop ViDA

in close cooperation with the spacecraft operations engineers, to tailor ViDA and its *"look and feel"* specifically to their needs.

To create the data visualisations available in ViDA, the D3.js library was employed. This is a JavaScript library, free and open source, which is very powerful and customisable to create a large variety of charts and plot types. Its capacity to support large datasets and dynamic behaviours for interaction and animation [4], makes D3.js a suitable choice for visualising satellite's TM data, in a fast and interactive manner.

For the server side, scalability, robustness and a reactive design were the drivers for choosing Scala [5] and the Akka [6] framework as the foundation of the backend. In fact, Akka provides good abstractions in Scala for concurrent, asynchronous, and distributed programming. The primary benefit of Akka comes from using the Akka actor systems. Actors are in principle isolated processing units that communicate with each other asynchronously via messages. These messages can, for example, contain commands to read specific data from a database and send the result to a certain recipient. Actors have a controlled life cycle, i.e., they are allowed to fail, but are always restarted by a managing actor higher up in the actor systems hierarchy. This is especially beneficial for encapsulating IO actions that do not block other activities of the backend. Furthermore, actors can be spawned on demand making the system scalable horizontally, e.g., if several users request data simultaneously.

Communication between both endpoints is set up via a GraphQL API. In short GraphQL is an open source query language designed to lift some of the restrictions of REST APIs. One of the most notable benefits of GraphQL is that clients have total flexibility in setting up queries and thus full control of the information sent and requested. Furthermore, GraphQL only needs one server endpoint to which a query is sent for server-side execution. A query can, for example, contain requests pointing to several separated data sources that might even be dependent on each other. Yet, GraphQL lets the client decide which data it needs (avoiding over-fetching by specifying only relevant fields) while only making a single request. Besides data queries, ViDA also makes use of GraphQL mutations to persist user's data. Each GraphQL operation that is sent to the ViDA backend is validated against a GraphQL schema that is generated from the backend Scala code directly. Just like Scala, GraphQL is strongly typed to make it less error prone.

3.1 Database

For the first version of ViDA the NoSQL database *Apache Cassandra* was used. The insertion and query performance were satisfying and the sharding of data by using multiple database servers worked flawlessly. However, the missing flexibility in querying data from the DB was a major drawback. Specifying a range of dates for retrieving TM data from the DB was not straightforward. Therefore it has been decided to look for an alternative solution to store a lot of data, combined with high insert rate and good query performance. Also important, but not top priority, was the amount of disk usage by the database. Not only the database schema plays an important role there, but also the possibility to compress data. With satellite TM containing a lot of repetitions of values over time, the potential for good compression ratio is high. After some prototyping and benchmarks the database of choice was TimescaleDB [7]. It is an extension to PostgreSQL and therefore is based on an open source object-relational database system. It provides full SQL syntax for flexible queries and delivers even better performance for insertion and retrieving data than the solution with Cassandra did.

Two features of TimescaleDB make it even more worthy of being a suitable database for ViDA: i) automatic downsampling, i.e. data reduction of long time-series data, and ii) native compression. The downsampling functionality helps increasing the query performance when retrieving TM data for large time spans by using continuous aggregates. Instead of querying the full-resolution data available, only a subset is retrieved. This subset is materialised in the DB continuously, fully automated, and the average, maximum and minimum values within a certain time frame, called bucket, are provided (see Sect. 5.2). The native compression of TimescaleDB saves already 98% of disk space by converting the data from a row-based format into a hybrid row-columnar one and applying type-specific compression algorithms. The database system is deployed on a RAID10 setup and runs flawlessly up to now.

4. Architecture and Infrastructure

As the ViDA framework is not one single application and is intended to allow for a multi-mission and fleet-capable solution, its architecture has to be reliable and able to scale as the demand grows. The system shall also allow for deployment in various configurations, if security or data locality requirements dictate so. The services comprising the whole system are running on Linux servers, using virtualisation and containerisation technology. For the beginning, a simpler setup has been chosen with just Docker containers running the software on a few servers, and regular instances of PostgreSQL databases as the basis for TimescaleDB. Later on, with growing computational

and storage demand, the components are portable to a cluster/orchestration solution (like Kubernetes) and database sharding could be enabled, once this becomes necessary.

The components that need to be hosted are currently the following:

- The ViDA backend service (Sect. 2.2), delivering requested data to the web clients via the GraphQL API. Currently a single containerised instance suffices for the number of concurrent users. In the future, this container could be hosted on a cluster and load-balanced in case the performance of the single instance is no longer sufficient.
- A number of TimescaleDB instances. As the ViDA backend is flexible from where to fetch the data, multiple database instances can be (and are) used to host the data of several missions, in line with available server resources and expected load. As the compression of TimescaleDB is really good, regular (non-sharded) instances of PostgreSQL/TimescaleDB are able to fit the expected total amount of data. Current estimates foresee that a single satellite with 30,000 parameters produces less than 2 TB of data on disk for its TM over a lifespan of 15 years. Some additional space is needed to host the remaining data (e.g. TC logs, ATHMoS novelty detection results, etc.), but still common hard disk and RAID sizes will be able to hold this amount without the need for database sharding. So far, the DB performance has been sufficient, however, in the future, with more data analysis workloads accessing the data, further performance improvements could become relevant.
- Several instances of the Data Ingestion service (Sect. 5.1), to continuously fill the databases with up-to-date TM data from the monitoring and control system. The data ingestion currently happens in a batched fashion, producing short periods (~20 min) of burst load on the database, predominantly during night times, while producing no database load for the rest of the day. In the future, the data ingestion service will be extended to also allow continuous streaming of incoming TM data. At that point, the data rates will need to be considered when assessing the database performance requirements.
- An instance of JupyterHub (external open-source software) with our own MiDA library baked in (Sect. 7), for easy data access for the users. This is a candidate for later scaling and making use of a cluster, as each logged-in user spawns a new session container for his/her interactive web-based Python environment. With more users and their increased proficiency with the system, computational demands will most likely rise.
- All containers constituting the ATHMoS infrastructure. ATHMoS (Sect. 6) was recently refactored to use a framework for ML workflows; its architecture is described in more detail in [8]. The ATHMoS services are also good candidates for a migration to a cluster-based system, to cope with the increased resource requirements when adding more and more satellites to the processing.

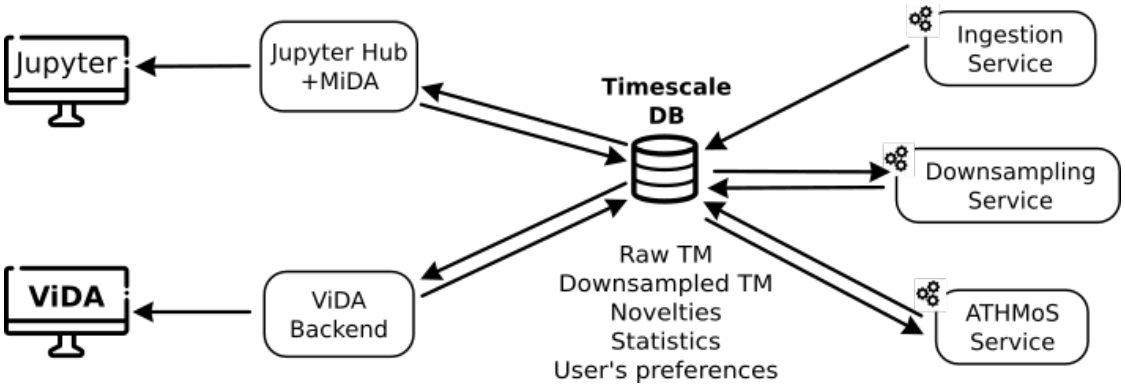

Fig. 5 Schematic infrastructure view of the ViDA framework.

A schematic view or the current ViDA framework infrastructure is shown in Fig. 5. As the development of the ViDA system progresses, further services will need to be added and, at some point, the migration to a cluster/orchestration solution will be performed. The services that are currently foreseen to be developed are additional data ingestion services for non-telemetry data (TC logs, etc.) and micro-services for processing tasks that the ViDA backend can trigger (e.g. for fitting/regression analysis, pattern search, etc.). Those new services will then also require a message broker like, e.g., Apache Kafka.

The total resources available to the ViDA framework are currently 448 GB RAM, 40 CPUs and 36 TB of disk space, which is good enough to cater for the 5 satellites that are currently operationally processed with it. These resources are estimated to be suitable for supporting up to 10 satellites over their full lifespan (10-20 years) with the

currently available features (i.e., ATHMoS novelty detection service, long-term TM visualisation and moderate requirements stemming from user's analysis scripts).

5. The Data Services

The data services in the background of the ViDA framework are developed as micro-services according to the reactive manifesto [3]. It assures good and scalable performance, maximum stability and predictive behaviour in case of errors, with a minimum of user interaction. The heavy-duty services are implemented in Scala, whereas prototyping and calculation-heavy services may initially be developed in Python. Once stability of the method has been reached and more performance is needed, the decision to orchestrate the existing code or to convert into a Scala service can be taken.

5.1 Data Ingestion Service

The ViDA data ingestion service imports necessary input data (TM data and other operational products) into the central database. It is based on the same technologies/libraries than the ViDA backend, namely, Scala and Akka. The ingestor receives as input the offline housekeeping TM data from the satellite already pre-processed into a row (time) – columnar (parameter values) format in plain text files. The input is converted into the data model of the relational database table (one row per parameter and timestamp) and inserted. This is done via resilient and parallel data streams that scale linearly with the number of processing cores until the disk performance is the limiting factor. The Akka stream library helps setting up those streams with the plain text input files as source and the DB table as sink. At last, the slick connector provided by Akka allows for batch insertion, which significantly speeds up the ingestion. The ingestor runs in multiple containerised instances that are distributed on several nodes, and each instance serves one mission. Together those instances insert over 700 million rows per day. One of those missions generates about 230 million rows of new data, which are ingested in 15 minutes. This instance runs on a 4-core machine with 32 GB of memory. It is located on the same machine as the DB itself. The ingestion speed could still be improved by using more processing cores and SSDs as disks, but for our current needs these results are good enough.

The next steps will be to add different data formats and sources, e.g., to also be able to retrieve telemetry data from the archive of upcoming missions that use EGS-CC [9, 10] as their monitoring and control system (MCS). The possibility to insert data in real time by becoming a streaming client to such MCS is also envisioned.

5.2 Downsampling Service

One of the key features of ViDA is the possibility to display long-term TM time series, spanning the full lifetime of a satellite, very fast. This is achieved through the downsampling service, which pre-processes the data and downsamples them into hierarchical levels of detail, which are then displayed instead of the full-resolution data.

The concept behind this service comes from the fact that it would be impossible for the user to clearly visualise millions of data points into the space available in a standard computer screen without overdrawing. Overdrawing occurs when the screen resolution is lower than the number of data points to visualise, resulting in the data points overlapping. This overlap can become so severe that it is impossible to perceive the number of points in a given region of the plot, thus obscuring outliers, hiding data distributions, and making the relationships among subgroups of data difficult to discern. To make matters worse, retrieving such amounts of data directly from the database would put high load on the network infrastructure, charge a lot of computational power, and, in the end, slow down the system.

The downsampling service chunks the full-resolution data into fixed time windows (or buckets), calculates meaningful aggregated values for each bucket and stores them in the DB. For some common aggregation functions the "continuous aggregate" functionality of TimescaleDB can be used (Sect. 3.1) and no dedicated external service code needs to be implemented. Currently, a standard minimum/maximum/average aggregation is performed, which will be soon augmented with a LTTB downsampling method. The LTTB algorithm [11] is designed to achieve the best visual approximation of the original full resolution data with less data points. However, in its standard form, the LTTB algorithm chunks by the number of data points and not by fixed time ranges, making it more suitable for data with constant sampling rate. In the case of our satellite TM, the algorithm will loose more detail in the time ranges with a lower sampling rate than those with a higher sampling rate. Thus, further aggregation methods, e.g., a modified version of the LTTB, that are suitable for the type of data in the satellite domain and the visualisations desired by the spacecraft operations engineers, shall be implemented.

Given the huge number of data points for each TM parameter, multiple hierarchical levels of detail are calculated, each providing a higher-level summary of the level below it. Based on the user's selected time range and the resolution of the screen, the appropriate level of detail is selected and retrieved from the DB by the ViDA backend to be shown in the plots. As the user zooms into the plot, finer levels of detail are retrieved and displayed, in a fast and

efficient way: first the pre-aggregated levels, later on the full-resolution data aggregated "on-the-fly" to optimally fit the screen resolution, and finally the full-resolution data without any aggregation.

6. ATHMoS Novelty Detection Service

The ATHMoS service is specifically developed for analysing large quantities of satellite TM data from a huge number of parameters. The ATHMoS service is based on semi-supervised ML and, broadly speaking, classifies whether the TM time-series data behave in a nominal or in a novel (anomalous) way. At its core, ATHMoS uses a novel algorithm for statistical outlier detection, which makes use of the so-called Intrinsic Dimensionality (ID; e.g., [12], [13]) of a data set. Using an ID measure as the core data-mining technique allows us not only to run ATHMoS on a parameter by parameter basis, but also monitor and flag anomalies for multi-parameter interactions, without a-priori manual training or parameter-specific tuning. The method analyses the past 12 months of TM data of every parameter as the training dataset, extracting various statistical features, and using a semi-supervised machine learning algorithm to distinguish between nominal and anomalous (novel) behaviour. Newly received TM data are then compared to the cluster of nominal data, and the "Outlier Probability via Intrinsic Dimensionality" (OPVID) algorithm [1] is applied to assign anomaly probabilities, based on the density of the cluster and the distance of the new data points to those of the nominal cluster. From the comparison with "historic" (past 1 year of data) and "recent" nominal clusters (which include only 1 month of data), and the resulting anomaly probabilities, the novelties are then classified as "high priority" (HP) or "low priority" (LP). Trends in the data, which might lead to a more severe anomaly in the near future, are also detected and classified as "trending detections" (TD). We refer to [8] for further details.

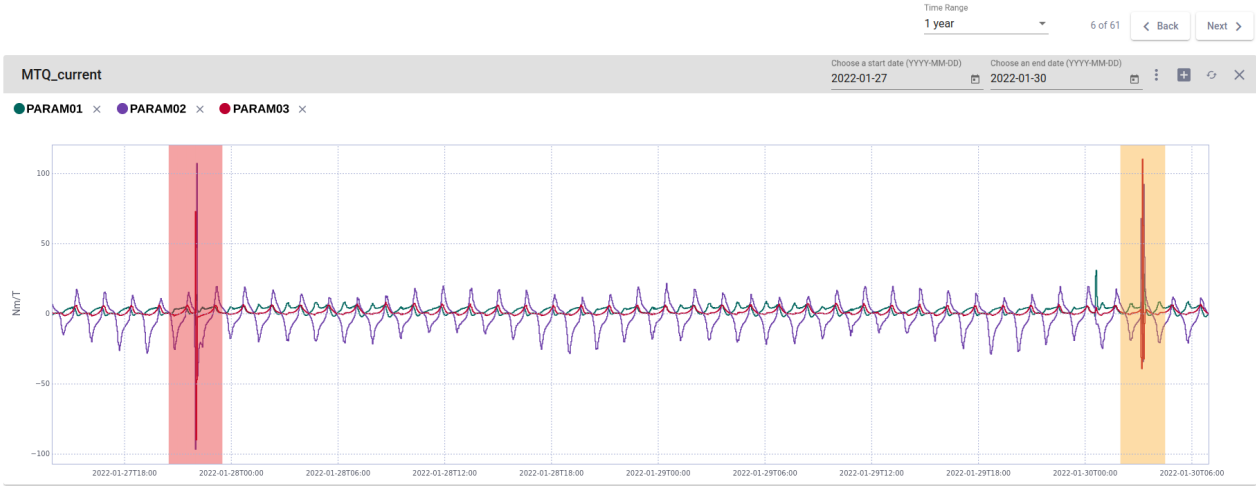

Fig. 6 Example screenshot of two novelties detected by ATHMoS for parameters in an autoplot (zoomed-in) and displayed in ViDA as vertical bars highlighting the time ranges of the detections (HP in red and TD in yellow).

This service is designed to run daily in the background, analysing newly received TM from every parameter to identify potential anomalies. The results from the ATHMoS analysis are stored in the DB and can be displayed in plots and visualisations (e.g., Fig. 4 and Fig. 6), on the ViDA dashboards (Fig. 1), or serve as a basis of automated alerts in our ViDA web application (Sect. 2). A key principle of the method is that any false-positive (or false-negative) detection can be corrected and relabelled by the user. These new labels will be considered during future data processing; thus, the detections will become more accurate and reliable over time. The relabelling functionality will be made available to the users through the ViDA interface (see Sect. 8). Moreover, by performing a periodic retraining of the models using past and recent TM data, the method adapts to changes in the nominal behaviour that can occur over a mission's duration.

7. Custom User Data Analysis

Whereas the ViDA frontend provides a wide variety of means to display and analyse stored data, special use cases remain, that cannot be foreseen by the developers. Giving the user the freedom to access the data via an API allows for custom analysis and integration of the raw and preprocessed data from the various ViDA services into their own routine workflows. This API interface is encapsulated into the MiDA library, which can be installed locally or, without the need for a local development setup, be used via a JupyterHub environment that integrates well

with the ViDA universe. The JupyterHub environment is a well-established multi-language development platform, providing a highly scalable per-user notebook application using the resources of the ViDA hardware cluster (Sect. 4). This especially enables the user to overcome the limits of local resources for big amounts of data.

The ViDA JupyterHub environment comes equipped with a variety of language kernels, e.g., Scala, Python, R, C#, C++, etc., where the user can develop his/her own tools for data processing. MiDA is developed in Python, but with the power of multi-language Jupyter notebooks, interaction with user's code written in other languages is easily possible within the same code block.

MiDA provides a number of generic methods to access data from the various ViDA services for its in-memory analysis, as well as generic transform-reduce operations to extract results from long time periods, thus freeing the user from the load-analyse-dismiss cycle of memory compatible chunks of data. In principle, once the methods and results are validated, they can eventually be integrated into new data derivation services and offered to the user in the ViDA web interface.

8. Discussion and Future

Up to now, the foundations of the ViDA framework have been laid out, to provide the users with a reliable and user-friendly application to ease and support their daily work. On top of these foundations, ViDA will be enriched with the integration of several more features and services, which will make this framework a unique and forefront tool for spacecraft operations.

As mentioned in the previous sections, one of the main use-cases for which ViDA was developed is the inclusion of contextual information to the TM data and to the ATHMoS detection results, such as the TCs, OBEs, ARs, OOL, flight procedures, etc. This information is crucial to help the user painting a wider picture of what is happening on the satellite at a certain time, thus promptly identifying the root-cause of anomalous behaviours. The information might not be limited to the direct by-products coming from the spacecraft, but in the future also include external data, such as space weather or space debris maps, which might significantly affect the status of a satellite.

Other important features that are planned as part of the ViDA roadmap are pattern searching [14] across the TM time series and data labelling. These features will allow the users to identify recurrent patterns and behaviours occurring in the lifetime of a satellite, whether due to TCs or specific manoeuvres and label them. The user will also have the possibility to label the novelties detected by our ATHMoS service, e.g., in case of false-positive (or false-negative) detections, and store the labels in the database. These labels will then be taken into account by our ML algorithms to improve the data models and thus the classification of the upcoming data, making the algorithms more and more accurate over time (Sect. 6).

While currently ViDA is designed to be an "off-line" data monitoring and analysis tool, i.e., handling data that are at least 1-day old, new developments that are envisioned for the ViDA/ATHMoS framework are in the direction of real-time (or near real-time) data analysis and predictive maintenance. For the real-time feature, a prototype of the ATHMoS novelty detection algorithms has been realised and tested by our research team, by using a continuous data stream on which the processing is done on-the-fly. Further implementation and developments are needed to realise a fully functional service, however the initial results are promising. This service could be setup directly in the control room and assist the engineers monitoring the real-time data stream available during every satellite pass, allowing prompt reaction in the event of any spotted problem. On the same line, a predictive maintenance service is foreseen to be included in ViDA, by using regression models and extrapolation into the future behaviour of the TM parameters for identifying and predicting trends that might lead to anomalies and unhealthy systems. Such service is a very desirable feature in space operations, as an early identification of a problem and a prompt reaction can be vital for avoiding major issues on the spacecraft.

Acknowledgements

The authors are thankful to Dr. Martin Wickler, Dr. Edith Maurer, all members of the MBT-Team and Heavens-Above GmbH for their valuable support and fruitful discussions.

References

[1] C. O'Meara, L. Schlag, L. Faltenbacher, M. Wickler, ATHMoS: Automated Telemetry Health Monitoring System at GSOC using Outlier Detection and Supervised Machine Learning, 14th International Conference on Space Operations, pp. 2347, 2016.

[2] Angular, https://angular.io/, (accessed 20.01.23).

[3] J. Bonér, D. Farley, R. Kuhn, M. Thompson, The Reactive Manifesto, 16 September 2014, https://www.reactivemanifesto.org/, (accessed 20.01.23).

[4] M. Bostock, D3.js, 2021, https://d3js.org/, (accessed 20.01.23).

[5] Scala, https://www.scala-lang.org/, (accessed 20.01.23).
[6] Akka, https://akka.io/, (accessed 20.01.23).
[7] Timescale, Inc., TimescaleDB, 2023, https://www.timescale.com/, (accessed 20.01.23).
[8] C. Schefels, L. Schlag, A. Del Moro, K. Halmsauer, T. Lesch, T. Göttfert, Bringing a Machine-Learning Based Novelty Detection Software Tool from Research to Production, 17th International Conference on Space Operations, Dubai, United Arab Emirates, 2023, 6 - 10 March.
[9] A. Walsh, J. M. Carranza, W. Bothmer, P.-Y. Schmerber, J. Rueting, P. Parmentier, P. Chiroli, M.-C.Charmeau, M. Geyer, The European Ground Systems - Common Core (EGS-CC) Initiative, American Institute of Aeronautics and Astronautics, SpaceOps, 2012, May.
[10] M. P. Geyer, A. Braun, S. Gärtner, P. Hamacher, L. Schlag, A.-K. Schroeder-Lanz, C. Stangl, N. Trebbin, Migrating from GSOC's SCOS derivate GECCOS to a distributed EGS-CC operations environment based on CCSDS MO/MAL, SESP 2017: Simulation and EGSE facilities for Space Programmes, Noordwijk, 2017.
[11] S. Steinarsson, Downsampling Time Series for Visual Representation, 2013.
[12] L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. E. Houle, and M. Nett, Estimating Continuous Intrinsic Dimensionality, National Institute of Informatics Technical Report, Tokyo, Japan, March 2014.
[13] M. E. Houle, Dimensionality, Discriminability, Density and Distance Distributions, IEEE 13th International Conference on Data Mining Workshops, December 2013, pp. 468-473.
[14] C. Schefels, L. Schlag, L. Dreier, L. Lincoln, and M. Steinbach, To Catch Them All: A Generic Approach for Pattern Detection in Time Series Satellite Telemetry Data, 16th International Conference on Space Operations, Cape Town, South Africa, 2021, 3 - 5 May.