BACARDI: A SYSTEM TO TRACK SPACE DEBRIS

M. Stoffers[1], M. Meinel[2], M. Weigel[3], M. Siggel[4], H. Fiedler[5], K. Rack[6], and Y. Wasser[7]

[1]*German Aerospace Center, Simulation and Software Technology, Cologne, Germany, Email: martin.stoffers@dlr.de*
[2]*German Aerospace Center, Simulation and Software Technology, Berlin, Germany, Email: michael.meinel@dlr.de*
[3]*German Aerospace Center, Space Operations and Astronaut Training, Oberpfaffenhofen, Germany, Email: martin.weigel@dlr.de*
[4]*German Aerospace Center, Simulation and Software Technology, Cologne, Germany, Email: martin.siggel@dlr.de*
[5]*German Aerospace Center, Space Operations and Astronaut Training, Oberpfaffenhofen, Germany, Email: hauke.fiedler@dlr.de*
[6]*German Aerospace Center, Simulation and Software Technology, Cologne, Germany, Email: Kathrin.Rack@dlr.de*
[7]*German Aerospace Center, Space Operations and Astronaut Training, Oberpfaffenhofen, Germany, Email: yi.wasser@dlr.de*

ABSTRACT

We present the "Backbone Catalogue of Relational Debris Information" (BACARDI) as an effort of the German Aerospace Center (DLR) to keep track of cooperative and uncooperative orbital objects. Key features of BACARDI are a database storing information about all known orbital objects, and a set of processing services that produce orbit information and different products like collision warnings.

Keywords: space situational awareness, space debris, orbit database, workflow system.

1. INTRODUCTION

The „Backbone Catalogue of Relational Debris Information" (BACARDI) has been developed in a joint approach of the German Space Operation Center (GSOC) and DLR Simulation and Software Technology. It's aim is to provide an unified database that contains orbit information of active satellites and space debris in Earth's orbit and related data products.

This paper describes the technical approaches taken to develop the BACARDI system:

- First we give a broad overview of the system as a whole in Section 2.
- The main part of this paper describes the software of the BACARDI system in Section 3. After an architectural overview we focus on technical details libraries and frameworks and how they are integrated to build-up the system.
- Towards the end we present some existing and planned services that will be provided by BACARDI in Section 4
- Finally we close with our conclusions in Section 5.

2. BACARDI OVERVIEW

BACARDI itself is a large-scale Python-based software platform to register and track orbital objects like space debris and satellites. The system supports different representations of orbital objects like TLE, osculating orbits, or ephemerides. Thus it allows for keeping track of objects from different sources with the aim to compile a database with highest completeness of known objects orbiting Earth and achieve precise orbit accuracy. To achieve this, data is collected from external databases as well as sensor networks all around the globe. Especially, SMARTnet^{TM} [11] provides tracking data from a network of ground-based telescopes operated by DLR, AIUB (Astronomical Institute of the University of Bern) and ADS (Applied Defence Solution). For computation, the platform integrates the Fortran flight dynamic libraries of the German Space Operation Center (GSOC) and combines its capabilities with the database.

BACARDI is designed to be scalable. At the core of the software a database with the orbital and meta information of all known objects is maintained. New tracklets, which are time series of measurements of the same object within a certain time span, are correlated with known objects to allow for orbit improvements of already recorded orbits, detection of manoeuvres, and in the case of negative correlation new objects can be detected. Also a regular collision detection for all known objects will be performed in the near future.

While the amount of recorded objects grows, additional

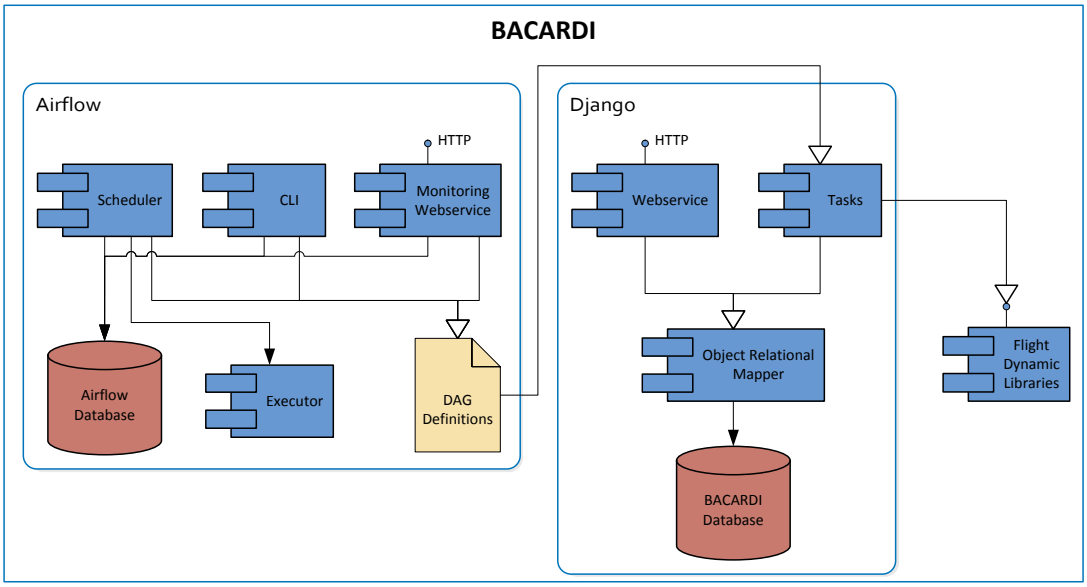

Figure 1. Architectural Overview of BACARDI.

processing power can be added by providing additional computing hardware. This enables us to keep the system growing while the data set grows from approximately 26,000 known objects today to an estimated number of about 250,000 objects once the new space fence of the USA is operational. As a first estimate, we are projecting a system that is able to process about 10 new incoming measurements per second. Besides our scalable architecture this is achieved by tuning the Fortran codes for use on many-core systems and accelerator hardware.

3. TECHNOLOGY

The following subsections provide a more detailed insight as to how we designed the architecture to support the implementation of BACARDI.

- First we explain the architectural overview and associated goals in Subsection 3.1.
- Next we describe the core „Django" package in Subsection 3.2.
- The GSOC Flight Dynamic libraries are topic of Subsection 3.3. How they are made available for Python code using F2x is topic of Subsection 3.4.
- The workflow management with the „Airflow" package as well as scheduling and execution of tasks is handled in Subsection 3.5.

3.1. Overview of the Architecture

In BACARDI, we aim for a loosely coupled software architecture with strictly separated packages to address maintainability, test-ability and extensibility. Figure 1 shows an outline of BACARDI. The architecture is divided into the three main packages labelled „Airflow", „Django", and „BACARDI". While BACARDI does not have a layered architecture in a classical sense, these packages build upon each other in a hierarchical way. Well defined functional constraints and interfaces of the packages are the base for a decoupled and extensible design. We especially pay high attention to keep the core „Django" package decoupled in a way that it is still usable without the parts of the „Airflow" package.

3.2. Django Core Package

This package consists of three components that make up the core functionality of BACARDI.

First, it contains the main data model, that is defined using the Django Object Relational Mapper (ORM). The ORM allows for defining Python classes that are describing the data. These classes are automatically mapped to tables in a relational database. By providing an object-oriented interface, the ORM makes it easier to query object instances from the database without the need to use a query language. It also enables us to replace the underlying database technology if necessary. The ORM also validates data written into or retrieved from the database against the model. This reduces the risk of incomplete database entries significantly.

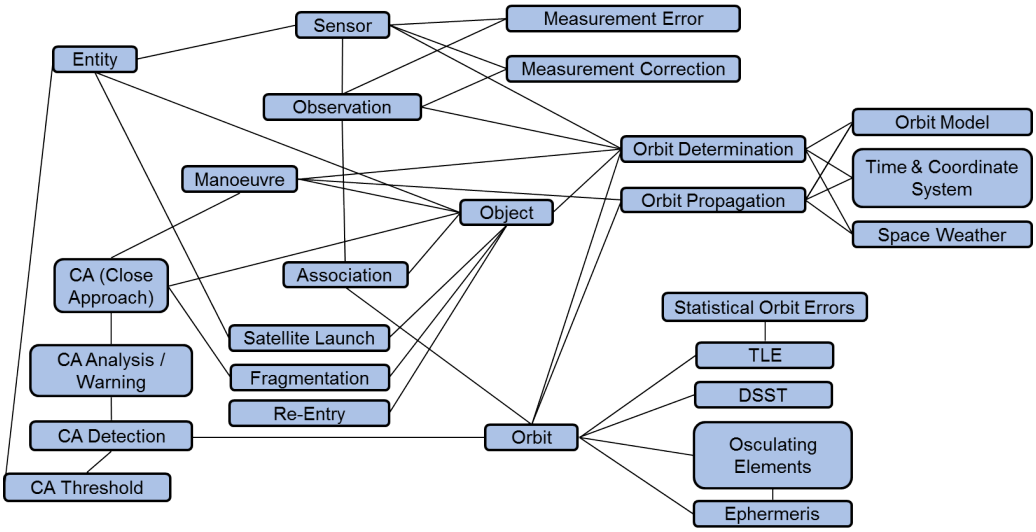

Figure 2. Excerpt of BACARDI database tables.

As an addition the Django allows to track changes and version of the ORM using an integrated migration concept. This ensures a reliable and stable database schema during the lifetime of the software.

The most important database tables and data relations implemented are outlined in Figure 2. As depicted, for each sensor observation the applicable measurement corrections are related, as well as measurement error records to be used for measurement weighting during observation correlation and orbit determination. Correlation results are stored, which establish the association between sensor tracks, orbits, and objects.

Different representations of an orbit are handled. So far, the Simplified General Perturbations theory for propagation of Two-Line Elements is implemented as well a numerical orbit integrator. Future version will include interpolation of orbit ephemeris and the Draper Semi-Analytical Satellite Theory (DSST). For each orbit record, a set of auxiliary data, like solar flux and geomagnetic indices, or the selected force model settings, can be stored. This allows for orbit propagation with identical input data.

The object history will be established with database records on fragmentation events, like explosions or on-orbit collisions, satellite launch and re-entry into the Earth's atmosphere. More tables dedicated to derived data products exit. As an example, for close approach (CA) detection thresholds on collision probability and on the CA geometry can be set for individual objects, depending on the needs of different satellite missions. For each CA event, updates for the screening results are stored.

The next core component is a collection of tasks. To enforce a separation between the database access layer and other parts of the software, common tasks are placed within the „Django" package. These tasks can be seen as atomic activities that interact with the data model. They act as building blocks for more complex services. Tasks are split into the three categories named „importer", „exporter", and „processor".

- Importer: An importer is defined as a task that imports data from a source into the BACARDI database. It can utilise external libraries to fetch and transform data into valid database objects.
- Exporter: An exporter is defined as a task that exports or serialises data from the BACARDI database into a specific format. It can utilise external libraries to store or send the result at or to different locations.
- Processor: A processor is defined as a task that executes computations on the data within the BACARDI database. E.g. associate tracklets with collected tracks or determine new orbits from associated tracklets and tracks. It can utilise external libraries like FD libraries to perform such computations.

Lastly, there is a „Webservices" component that uses Django to provide access to model data and tasks using a simple web-based API. This component might be used in a future version of BACARDI to exchange data with partners.

3.3. Flight Dynamic Libraries

At GSOC, there is a long history of software development and application to space operations. Therefore, already existing software related to orbital mechanics could be integrated into BACARDI as another package. Building on Fortran libraries from the flight dynamics group, a mixed language programming approach is adopted. Existing and newly developed Fortran code is wrapped and

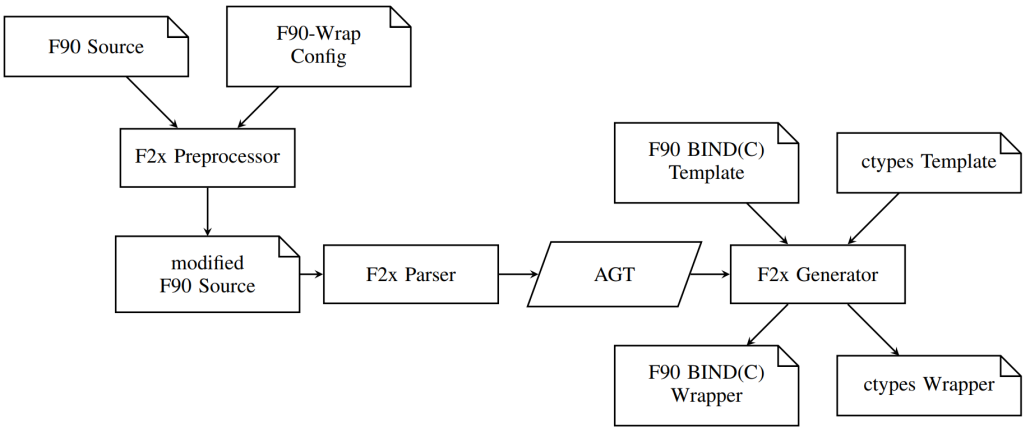

Figure 3. Internal workflow of F2x to wrap Fortran sources to Python.

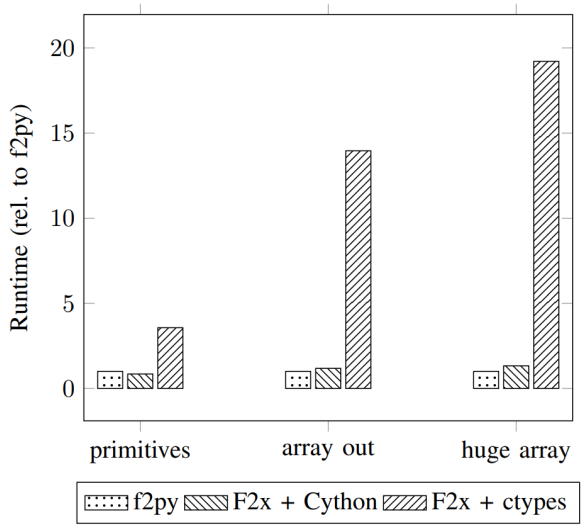

Figure 4. Benchmarks results to compare different F2x templates with f2py.

called from Python, the main programming language of BACARDI.

The flight dynamics libraries provide many useful algorithms, e.g. time and coordinate system transformation, various orbit perturbations or modelling of sensor observations [8]. These functions serve as building blocks for new Python modules or Fortran extensions. This reduces time and effort needed to implement new algorithms.

Subroutines for complex and computational intensive calculations like numerical orbit determination and propagation [10] can be executed in Fortran. We achieved run times much faster compared to a pure Python implementation.

The FD libraries have been carefully developed, tested and are flight-proven by many satellite missions. Existing test cases have been reproduced after code wrapping and automated as Python unit tests.

3.4. F2x

To integrate the Fortran code of the FD Libraries into BACARDI, we use *F2x* [6]. This tool was initially developed in the context of the BACARDI project and is meant as a more powerful replacement for *f2py* [5].

F2x combines modern approaches to overcome some of the restrictions of f2py. Especially, it allows for using derived data types. It also abstracts from the underlying Fortran compiler by applying the standardised BIND(C) interfacing option. The parsing of the Fortran source is performed utilising a full Fortran grammar imported from the Open Fortran Parser project. The wrapping layers are created with template based code generation techniques and thus allow flexible adaptation. The representation of the code that is used as model for code generation is a simplified version of an abstract syntax tree that is used to abstract the generation from the actual parser in use.

Figure 3 shows the internal process that F2x applies to get from Fortran source to a usable Python module. All these steps are implemented as an automated build process with distutils. It is based an the *NumPy* infrastructure to support a wide range of Fortran compilers.

There are currently two implementations: One is a feature-complete implementation that uses Python's *ctypes* library to access the BIND(C)-exported Fortran methods, the other implementation uses *Cython* to generate the required Python API. This implementation is preliminary and lacks a lots of features supported by ctypes. However, some benchmarks have been made to compare both implementations with each other as well as with f2py. The results are shown in Figure 4. One can clearly see that the ctypes implementation is a lot slower than f2py, which was expected. In contrast, the Cython implementation shows much faster results that are in the range of f2py.

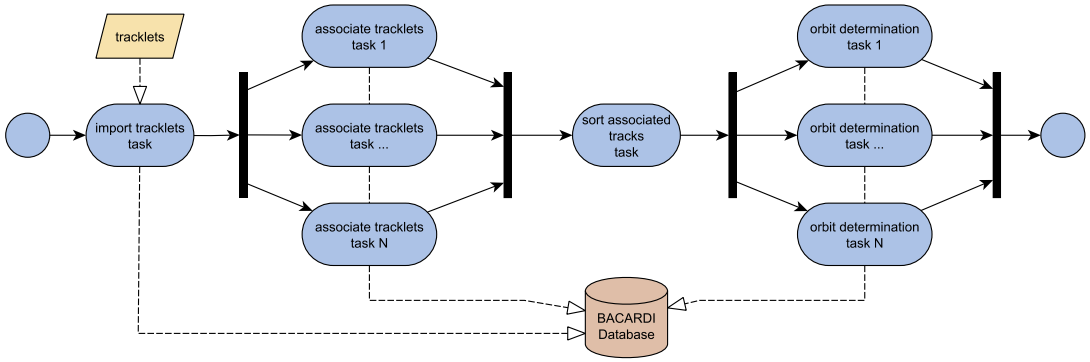

Figure 5. DAG for tracklet to track correlation and orbit determination.

3.5. Airflow

We use the Python-based Open Source framework Apache Airflow to handle task scheduling and processing in BACARDI. The „Airflow" package consists of four main components: a task scheduler, an executor running tasks, a command line interface (CLI), and a web interface for management and monitoring.

To combine simple tasks into more complex services or workflows, Airflow utilises the concept of Direct Acylic Graphs (DAGs). These tasks – being the basic building blocks – are imported from the „Django" package. This approach enables us to loosely couple BACARDI core functionalities to Airflow while still being able to easily replace Airflow. The separation also avoids complex DAG scripts and keeps BACARDI core functionalities testable and adaptable for future modules.

As shown in Figure 1, DAGs are explicitly configured in DAG definition files. These configuration files define schedule times for DAGs as well as which queue or hardware pool is used within a specific task. It is also possible to set a priority to each task if needed.

A typical DAG is shown in Figure 5. One can see different tasks that are partially executed in parallel where each task is maintaining its own connection to the BACARDI database, if necessary.

The main purpose of the Airflow scheduler component in BACARDI is to schedule DAGs at a predefined point in time. Therefore, the scheduler constantly monitors the DAG definitions and updates the database, if changes were made or DAGs are due to schedule. For this reason, Airflow is maintaining its own database to store metadata about scheduling and execution of DAGs. Next to state and runtime information about upcoming, running and finished DAG runs, the database is also used as communication channel between different Airflow components as well as between separate tasks. By using its own database it is ensured that the main BACARDI database is kept free from metadata of the scheduling. To execute unscheduled processes, the web services and the command line interface provided by Airflow can be used.

Beside similar solutions like Luigi [7], Apache Airflow is designed to be adjustable in all of its components. To scale-out the system the executor component, which is responsible for the actual task execution, can be transparently replaced with another executor component. For Example, the default local parallel executor can be replaced with a celery executor [3] to distribute and load-balancing tasks over multiple hardware resources by utilising message queuing systems. This allows us to scale the system with the expected increase in load caused by a growing dataset and additional sensors from various networks constantly delivering new observations.

4. EXISTING AND PLANNED SERVICES

The Django web framework used in BACARDI allows for exposing functionality over a ReST Api. By reusing the exporter functions from the task package, data such as TLEs can be queried from a web resource. This way we enable other software to make use of our data. A first example is implemented with the BACARDI Viewer. The viewer was inspired by the *Stuff in Space* project [12] and is a WebGL based software that visualises the orbital objects of the database inside the web browser. The BACARDI Viewer propagates the objects in real time based on TLE data using the *satellite.js* library [9]. One of its objectives is to visually illustrate the challenge of space debris to a broad public. Therefore, it is possible to interactively discover the space around the Earth, jump to objects with a mouse click, or use several filters to reduce the amount of data displayed. We also included many ways to filter objects: object name, time of start, data source, altitude, perigee, apogee and any combination thereof. In addition to the current position data, it is planned to also visualise paths with increased collision probability in the future. A screenshot of the BACARDI Viewer software is shown in Figure 6.

Future version of BACARDI will allow to directly import and export datasets or trigger new calculations from an attached dataset in Apache Airflow utilising a ReST Api.

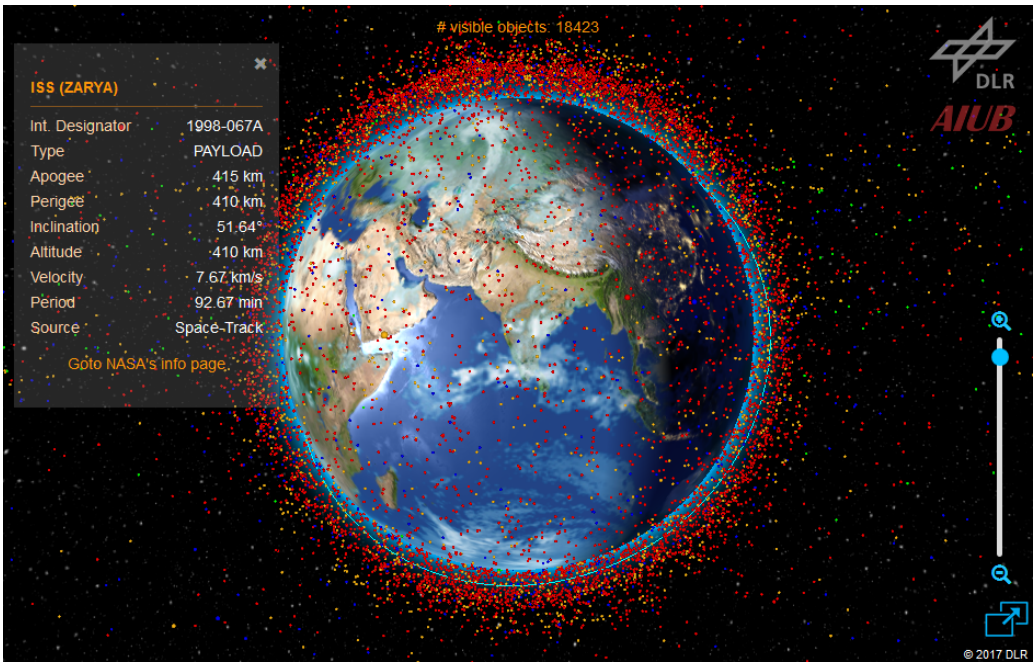

Figure 6. Screenshot of the BACARDI Viewer

5. SUMMARY

We presented the „Backbone Catalogue of Relational Debris Information" (BACARDI) that collect orbit information, determines orbits from sensor observations and generates derived data products. Especially, we described the architecture of the whole system and how we approached the implementation using popular Python frameworks like Django and Airflow. We briefly introduced how we integrated the flight dynamic libraries written in Fortran by using F2x. As some first results we also introduced the BACARDI Viewer which is one of the first applications using BACARDI.

Currently a first version of BACARDI is set up to evaluate current implementations. All of the infrastructure is ready for high availability production use. Interfaces for data import are constantly fed by partner catalogues and data from SMARTnet^{TM} so a growing number of associated observations and recorded orbital objects is generated. First results for processing optical telescope observations can be found in [2] and [11].

REFERENCES

1. Apache Software Foundation (ASF), Apache Airflow (incubating), `https://airflow.apache.org/`
2. M. Weigel, M. Meinel and H. Fiedler, "Processing of Optical Telescope Observations with the Space Object Catalogue BACARDI", 25th International Symposium on Space Flight Dynamics (ISSFD), Munich, 2015.
3. Solem A., and contributors, Celery: Distributed Task Queue, `http://celeryproject.org/`
4. Django Software Foundation and individual contributors, Djangp Web Framework: The web framework for perfectionists with deadlines, `https://www.djangoproject.com/`
5. Peterson P., (2009). F2PY: a tool for connecting Fortran and Python programs, *International Journal of Computational Science and Engineering*, **4**(4), 296–305
6. Meinel, M. (2018). F2x: A versatile, template-based Fortran wrapper written in Python, (Version 0.1), *Zenodo*, `https://doi.org/10.5281/zenodo.1405956`
7. Spotify AB, Luigi: Python module to build complex pipelines of batch jobs, `https://github.com/spotify/luigi`
8. V. Chobotov, "Orbital Mechanics", Third Edition, American Institute of Aeronautics and Astronautics, 2002.
9. Kandadai S. et al., satellite.js: Modular set of functions for SGP4 and SDP4 propagation of TLEs, `https://github.com/shashwatak/satellite-js`
10. O. Montenbruck adn E. Gill, "Satellite Orbits - Models, Methods and Applications", Springer, 2000
11. H. Fiedler, J. Herzog, M. Prohaska, T. Schildknecht and M. Weigel, "SMARTnet(TM) - Status and Statistics", International Astronautical Congress 2017 (IAC), Adelaide, 2017
12. Yoder J., Stuff in Space: A real-time interactive WebGL visualisation of objects in Earth orbit, `http://stuffin.space/`