PROVISION OF SST AS A SERVICE BASED ON A DATA-STREAM-CENTRIC ARCHITECTURE

S. Müller, C. Kebschull, and J. Radtke

OKAPI:Orbits GmbH @ Institute of Space Systems, TU Braunschweig, 38108 Braunschweig, Germany,
Email: {sven.mueller, christopher.kebschull, jonas.radtke}@okapiorbits.space

ABSTRACT

Space is becoming a domain in which commercial and institutional entities operate side by side. Launch costs are decreasing because of new launch concepts and the miniaturization of satellite hardware. In the US, an effort is made to establish space traffic management as an analogue to air traffic management under the control of Department of Commerce. The private sector grabs opportunities that open up. Its actors plan to put a vast amount of new satellites into orbit which work in tandem. Not only big players are on the field, but many new space companies put their feet into the door ("NewSpace"). All the while, digitalization is progressing, making place for new technologies and paradigms. One of the paradigms emerging is Software as a Service (SaaS), which aims to provide software as something that is connected to on demand. Instead of downloading and installing software on one's own systems, software runs somewhere else and is used via network. Common implementations allow users to connect via web browser and/or allow the user's systems to connect via Representational State Transfer (REST) Application Programming Interface (API). The latter allows an automation of the usage of provided software. This approach yields the possibility for space operators to integrate exactly what they need for their Space Surveillance and Tracking (SST, US pendant is Space-situational Awareness (SSA)) needs into their own ground-based software systems. An algorithm can be provided, but so can a core SST functionality like statistical orbit determination or even a whole catalogue maintenance system. Also, new sensors and catalogues keep coming online pushing the need to be able to cope with Big Data. Scalability and adjustability are key to provide NewSpace actors exactly what they need. This can be achieved by allowing external SST experts to easily adjust a scalable SST processing system which provides SST as a Service (SSTaaS).

This paper presents OKAPI:Orbits' SSTaaS provision based on a Data Stream Management System (DSMS) architecture. DSMS allow the definition of the data flow within the system via a script language. They don't save data permanently, but rather expect continuous data stream input and produce continuous data stream output. DSMS work in RAM only, are suitable for working with uncertain Big Data and are inherently modular. It is shown what SST experts can do with the architecture presented to tailor the service provision to user's needs.

Keywords: SST; DSMS; REST; SaaS.

1. INTRODUCTION

Not only science and exploration, but also Earth observation, reconnaissance and communication activities have been in the focus (of developments of space technologies) since the beginning of the space age in the second half of the 20th century. First, nation states and their administrations sought access to space. Fueled by the cold war, bold risks had been taken to push the boundaries of what was deemed possible and to demonstrate the feasibility of undertakings in space. New technologies had been developed along the way, enabling commercial use of space in the 1960s (TELSTAR 1 was the first commercial satellite, launched 1962 [12]). Since then, access to space has been eased as the prices per kilogram launch mass have dropped ([7] for example gives a reduction in launch cost by a factor of 20 due to the commercialization of space launches) and proper legislation allowing for the development of the commercial space sector (commercial space policy and enabling legislation 1984). The progressing of space technologies has further enabled the development of smaller satellites in the private sector, leading into the "NewSpace" era, where startups and small companies are able to launch and operate satellites – and even satellite constellations.

This new kind of space users works under drastically different requirements than the "classical" space actors, which, in comparison, had unlimited sources in terms of money and personnel. NewSpace' focus is on the application of space systems, without having to spend too much time and money on big soft-

ware products. They need flexibility and automation where possible. This, by the way, also applies to space users from academia and hobbyists (for example hobby astronomers).

Although money and personnel are very restricted for this group, the functionalities needed are comparable to those of the classical space actors. Alongside a large number of different regimes, this also involves the field of Space Surveillance and Tracking (SST). The underlying backbone of every SST system is a very capable and efficient orbit propagator, based on which a large set of functionalities can be built. Among others:

- A satellite pass prediction, interesting for ground station and sensor operators (such as telescopes, laser ranging stations, radars etc.).
- Orbit determination, needed by satellite operators, universities and other research organizations, but also sensor operators.
- Conjunction detection / collision avoidance services including manoeuvre plan recommendations, used by satellite operators, especially constellation operators.

Of course, the propagation might also be needed without added functionalities.

One obvious way to have SST capabilities as easily accessible and usable as possible is to just release the algorithms involved. This allows users to translate it into the programming language they happen to use and to run it. However, the example of the NASA Breakup model [14] shows that such an approach can result in many different-behaviouring algorithm implementations. Also, there is the question of validating and maintaining ones own algorithm implementations. It's the responsibility of the user to ensure the correctness of the code. Additionally, implementing capabilities which comprise of multiple algorithms can be cumbersome and needs a fitting software architecture to manage the capabilities.

Another possibility is to release the source code of implemented algorithms. This way, users can copy the code into their own software and do modifications as necessary. Although this is a very fine approach for fostering collaboration, we see that there are users who don't want to be bothered with downloading the code, getting all necessary dependencies with the correct versions, integrating the code (which may need glue code to cross programming languages), modifying it, testing it and maintaining it.

Another obvious way is to release a ready-to-use software package in the form of libraries. This approach is a lot better for users in terms of easy access. However, the dependencies need to be managed in any case and the libraries need to be accessible from the programming language of the user. This means either that the provider of the software packages needs to distribute its software for as many different systems as possible or that users need, again, to cross language boundaries.

Then, of course, the algorithms can be made available in the form of executables. However, providers, again, need to take multiple software platforms into account and using the tools may encompass writing of input files, calling the tool on the command-line (if possible), reading output files and so on. Another disadvantage lies in this approach's limits in terms of automation and flexibility. gpredict, for example, is a wonderful tool to predict passes. However, it lacks automation and software integration capabilities. Users also may have to build what they need based on multiple tools bundled together. Another example is OreKit which takes into account different usage platforms by being written in Java. However, integration with other software still is not that easy and the tool is rather complex and extensive which can make it difficult to just use one simple function.

Also, with the IT world rapidly evolving, it becomes clear that software, in the long run, will more often communicate with other software via web-based techniques. This decoupling of software programs, i.e. the separation of software programs and/or components in terms of location and technical dependencies, makes it possible for the used software to be hosted and maintained on a different machine which may even be located on the other side of Earth. The means of web-based communication are well-standardized and widely available. Although the decoupling, of course, has disadvantages that need to be taken into account, its ease of use lets this so-called Software as a Service (SaaS) increase in popularity.

Apart from the above-mentioned challenges, a problem that arose during our work as project scientists was software scalability. Whenever we finished the development of simulation software and/or its configuration, the time came to execute many simulations, e.g. Monte-Carlo runs. There always was the question of where to execute them. Normally, many cores were needed as well as lots of RAM and persistent storage. For this purpose, servers were bought, set up, installed, maintained. Of course, servers can also be rent and/or servers that are available for research purposes be used. But getting, installing, setting up and integrating external software components is still a hassle. Also, there are still boundaries to what can be done, because of the resource limits of the server. If more parallization is used, more RAM is necessary for execution of, e.g., numerical full-force orbit propagation and scripts need to be built to distribute the work load on many servers. Or, maybe there already exists a mechanism for distributing execution on multiple machines which needs to be understood

and configured. This is the reason that the IT world currently thinks about so-called serverless computing, or, in other words, Function as a Service (FaaS) [9] where the cloud provider adds server resources automatically when necessary. To be concrete, if numerical, full-force orbit propagation was available via SaaS in a FaaS cloud, users could request the execution of such orbit propagation without worrying about the technical details of server resource allocations.

In the space domain, SaaS does exist at some places. One example is LeoLabs, who provide access to their platform via `https://platform.leolabs.space/`. Given the nature of the company, the services focus on sensor data handling. Through either a web-based graphical user interface, a python command-line tool or a REST API, the data as well as data products can be retrieved. Very similar to that are the interfaces to the CSpOC catalogue, provided via `https://www.space-track.org/`.

Our long-term goal is to provide a REST API with which scalable SST services can be executed without a hassle from anywhere. In the long run, the services are planned to comprise:

- Pass Prediction
- Numerical Full-Force Orbit Propagation
- Initial Orbit Determination
- Orbit Improvement
- Orbit Correlation
- Collision Avoidance Manoeuvre Plan Generation

Currently, Okapi version 0.10 beta is running with the provision of pass prediction and numerical full-force orbit propagation. It is still no version 1.0 and in beta state. Also, the FaaS aspect is not realized yet, since our focus lies on the provision of more services and their validation through own tests and as much feedback from as many people as possible. This paper's structure is roughly oriented on the software layers (cf. Figure 1). Section 2 gives an overview of SST algorithms for which we are currently working on a service-based provision. In Section 3, the database-like technology used for a performant and modular provision is described, before the service-oriented access via REST is the topic of Section 4. Because of the modular approach, a tailoring of the services to a user's needs is possible. How this can be done is shown in Section 5, with a conclusion given in Section 6.

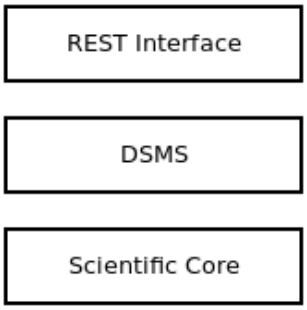

Figure 1. Software layers of SST service provision.

2. SCIENTIFIC CORE

The scientific software, which is planned to be made accessible via web services, originates from research and development conducted at Institute of Space Systems (IRAS) of Technische Universität Braunchweig (TUBS), Germany. A selection of the algorithms used for the services is described in the following.

2.1. Pass prediction

The current pass prediction method is based on the SGP4 implementation provided from David Vallado [13]. Generally, it requires a TLE as input, the definition of a ground station in WGS-84 as well as a time window, for which passes are to be predicted. The times should always be close to the epoch of the TLE. At first, the TLE element is propagated over the required time frame. Following, the TEME positions returned by SGP4 are converted into the correct SEZ coordinate frame, from which range, azimuth and elevation can be computed. All times, at which the elevation is negative, are neglected, others are aggregated as passes. For each pass, next to the position of the object in SEZ, a time stamp, the range-rate, illumination conditions (in-sunlight or not), and signal delay time are calculated. A typical pass being output by a pass prediction is given in Figure 2.

As long as the input provided are TLE, the accuracy of this implementation is deemed sufficient. For high accuracy applications, a future pass prediction based on a more precise numerical propagation (cf. the following Section) will be released.

2.2. State and covariance extrapolation

Since 2012 a numerical propagator, called NPI Ephemeris Propagation Tool with Uncertainty Extrapolation (NEPTUNE) as part of the ESA Networking/Partnering Initiative (NPI) programme [4]

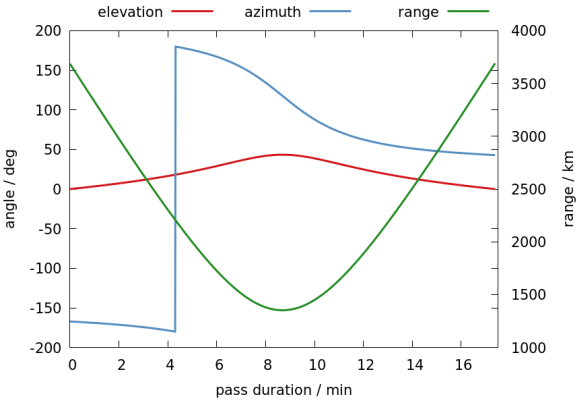

Figure 2. A typical pass prediction.

has been developed at IRAS. The core of the propagator is the optimized Störmer-Cowell integration routine [3]. It is a variable and multi-step, double integration algorithm, with support for shadow boundary transits. NEPTUNE incorporates a number of relevant force models for highly accurate state extrapolation [5]:

- EIGEN-GL04C, EGM96 and EGM2008 gravity models,
- NRLMSISE-00 and HWM07 to estimate the drag,
- Sun and Moon third-body perturbations,
- Solar radiation pressure,
- Visible and infrared albedo,
- IERS solid and ocean tides,
- and the IAU 2006/2000A (GCRF/ITRF) transformations regarding Earth Orientation Parameters (EOPs).

NEPTUNE is optimized for high performance computing within SST systems to extrapolate ECI state vectors and covariances. It is applied within statistical orbit determination processes as well as conjunction, re-entry and pass prediction functions. NEPTUNE is modular and can be used directly as a Fortran library, via a common Orbital Propagation Interface (OPI) [10] or as a stand-alone application.

2.3. Orbit Determination

As part of the Radar System Simulator (RSS) project initiated at IRAS in 2015, multiple orbit determination algorithms have been implemented and tested [8]. The Initial Orbit Determination (IOD) capabilities for radar tracklets are achieved through different methods:

- Gibbs,
- Herrick-Gibbs and
- Preliminary orbit determination from the Goddard Trajectory Determination System (GTDS).

The latter is an implementation of the algorithm described in the GTDS report [2]. It uses multiple consecutive position vectors to derive a full state vector, estimating the velocity vector in an iterative manner. As a starting point of the iterative process a circular orbit is assumed. For objects with a-priori knowledge multiple statistical orbit determination algorithms have been implemented and are available:

- Weighted Least Squares,
- Extended Kalman Filter,
- Unscented Kalman Filter, and
- Ensemble Kalman Filter.

They produce an updated state vector as well as a covariance to express the uncertainty from the measurement and process noises. Within RSS project, a cataloguing system has been developed, holding and updating the states of all objects in a PostgreSQL database. The simulation system was designed to hold large populations and has been used for sensitivity analyses with the MASTER-2009 population [6] as the starting point.

2.4. Conjunction Analysis

One of the simulation applications with larger populations is the analysis of conjunctions. For this purpose the algorithm by Alfriend & Akella as described in [1] has been implemented.

3. DATA STREAM MANAGEMENT SYSTEM (DSMS) ARCHITECTURE

In order to achieve modularity and scalability, a Data Stream Management System (DSMS) is utilized. In comparison to conventional database systems, DSMS can be described as management systems for data streams instead of data tables. While queries that are sent to conventional data base systems create and manipulate data tables, queries to DSMS create and manipulate data streams. Data streams are potentially infinite sequences of data sets that stream in or out of a place of interest. In a so-called relational DSMS, a data set is the equivalent of one row that

can be found in tables of conventional databases – together with one of two markers: insertion or deletion. Insertion data sets tell the DSMS that the data set shall be inserted into the data table, Deletion data sets tell it to delete it from the table. As a result, a data stream can be seen as a data table that is updated continuously by data sets streaming in. A broader explanation of this can be found in [11].

Separate functions are bundled as so-called operators, self-contained modules with input and output (cf. Figure 3). Two special kinds are data sinks and data sources. A data source (operator) accepts incoming data streams, while a data sink (operator) channels data streams out of the DSMS. By querying the DSMS, the operators and their connections are created to produce an operator graph which determines the data flow in the DSMS. There are operators for the selection, aggregation, branching and merging of data. What's important is that a DSMS works continuously on a data-set-per-data-set basis. Each data set is processed and its results are produced, (if any), before the next incoming data set is looked at. This way, big bulks of data are not processed in one batch, but rather one data point at a time. Also, the CPU time is distributed over all operators by a scheduler. Because of this, a quasi-parallelization and a conveyor-belt-like processing is realized leading to responsive and scalable processing. Since all functionalities are incorporated by one or multiple operators, DSMS are inherently modular, their internal data flow can be modified quickly by changing queries saved in a script file.

4. SOFTWARE-AS-A-SERVICE PROVISION VIA REST

4.1. REST API Provision

SaaS can normally be accessed by other software via an Application Programming Interface (API), which is often either SOAP- (Simple Object Access Protocol) or REST-based (Representational State Transfer). As to REST-based APIs, in many cases, using such an API requires only a few lines of code - or a Firefox add-on - or two commands on linux command-line (one for the installation of, say, *wget*). It's using the same technology and standards as the WWW - HTTP or HTTPS requests are sent, an HTTP or HTTPS response is delivered. Requests and responses are HTML documents. Instead of being interpreted to render a web page, they carry bare information and/or data, ready to be read and used by other programs and software.

Our REST API utilizes HTML body text formatted in Java Script Object Notation (JSON). JSON libraries are widely available and JSON text is human-readable, even more readable than XML. It is also a structured language meaning that it is possible to express structures containing values and/or other structures. The current workflow for using the services is explained using the pass prediction. First, users request service execution:

```
{
  "tle": string,
  "groundLocation": {
    "longitude": float,
    "latitude": float,
    "altitude": float
  },
  "timeWindow": {
    "start": ISO-8601 timestamp,
    "end": ISO-8601 timestamp
  }
}
```

The request indicates the use of TLE data and expects the two lines of a TLE as a string. Following is a sensor definition using longitude, latitude and the altitude of the sensor. The JSON formatted configuration concludes with the definition of the start and end time of the time window for which the request is placed. The response to this HTTP request contains a request ID, which is put into the address of the second request for retrieving the service execution results. This separation of requesting execution and requesting execution results is to prevent blocking in software that uses a service. Depending on the extensiveness of the request, service execution could take some time which is why the software that made the execution request should be free to do something else, while the service execution can run in parallel. Currently, two different aggregations of the pass prediction output are provided: The first one includes an overview of characteristics of the passes by giving the values for each parameter at the beginning, mid, and end of each pass as well as the minima and maxima:

```
{
  "azimuth": {
    "atEnd": float,
    "atMid": float,
    "atStart": float,
    "max": float,
    "min": float
  },
  "elevation": {
    ...
  },
  "timeWindow":{
    "end": ISO-8601 timestamp,
    "start": ISO-8601 timestamp
  }
  ...
}
```

The second possibility is to retrieve a complete tracking file, which includes all values as time series, cur-

rently fixed to one value per second.

The full propagation is currently kept very simple. A service execution request contains only the state, epoch, object characteristics, and the time span of the propagation:

```
{
  "initialState": {
    "area": float,
    "mass": float,
    "x": float,
    "y": float,
    "z": float,
    "xDot": float,
    "yDot": float,
    "zDot": float,
    "mjd": float
  },
  "request": {
    "propagationDuration": float
  }
}
```

The execution results response simply contains the new state, propagated over the required time frame. For example:

```
{
    "mjd_of_state": 57599.02210648,
    "output_state_dotx":
        -0.04964944547075876,
    "output_state_doty":
        -7.292983108675719,
    "output_state_dotz":
        1.636736163248116,
    "output_state_x":
        -431.5592424195601,
    "output_state_y":
        1527.4145096469522,
    "output_state_z":
        6937.437009491794
}
```

The current implementation serves evaluation purposes, as important parameters such as the reference frame are not defined. Future releases will implement CCSDS standards to fill this gap.

4.2. Access from Programming Languages

Here, the example of sending a service request and retrieving the result is given, using the REST API from Matlab. The first request is sent as a so-called POST request, for which Matlab provides the following function:

```
response = webwrite(url, request_body
    , options);
```

The url contains the complete address to the resource, to which the request is posted. The request body contains the actual request (cf. previous Subsection), for example a state to be propagated, or an object and ground-station combination, for which a pass prediction is to be made. The options need to be defined to fit the server. For Okapi, the following options are needed:

```
options = weboptions('RequestMethod
    ','post','MediaType','application/
    json','HeaderFields',{ 'Accept' '
    application/json';'access_token'
    access_token; 'expires_in'
    string_with_time_to_expiration;'
    token_type' string_with_token_type
    });
```

In short, the HTTP method is defined (POST), the media-type which is to be used to exchange the data, as well as optional header fields, for which the most important is an access token, which has to be retrieved beforehand. Once the request is sent, the response contains a request ID which is used to identify the results. Retrieving the result works analogous, but as the request is retrieved using the GET method instead of POST, webread is used:

```
result = webread(url,options);
```

The options have to be adapted by changing the RequestMethod to 'get'. The result then contains the solution to the previously sent request, thus a propagated state or the prediction of a satellite passing over a ground station.

4.3. Programming Language Connectors

To ease the use of the provided API, it made sense to provide a connector to the API, which takes over the communication with the REST API and makes it accessible as any other Matlab function:

```
% perform login, initialize all
    options
PicardLogin = OkapiInit(url_to_server
    ,user_name,password);

% send a request to the server
requests =
    OkapiSendPropagationRequest(
    PicardLogin, request_body);

% retrieve a result
result = OkapiGetResult(PicardLogin,
    requests);
```

5. TAILORING

The described architecture makes it rather easy, to adapt the services to different users or changed user needs. As an example, the functionality to create pass predictions can be used. A general overview of the pass prediction implemented in Picard is given in Section 2.1. Based on a user requirement, the first iteration of the pass prediction created an overview of key characteristics of a pass over a certain ground station. Therefore, in the first iteration, a pass prediction operator was written that delivers exactly these kind of outputs as result. Following, for a different application, a complete tracking file of each satellite pass was needed. Thus, instead of providing characteristics per pass, azimuth, elevation, etc. had to be provided continuously (i.e. in one-second steps) over the complete duration of the pass. To achieve this, a second iteration of the pass prediction was performed, by doubling the pass prediction operator and its corresponding REST source and sink operators plus the adaption of some of their outputs. While this is a working solution, it is not very efficient and furthermore doubles large amounts of the code. Therefore, the third iteration is going to be the reversion to one pass prediction operator chain, which provides the full output of each pass (thus a tracking file), and forward this output to further operators, which perform filtering tasks, such as building averages, minima, maximum, etc. The great advantage of this approach is that once filter operators have been written, they can be used for any kind of input from any kind of other operators. All that needs to be done is to specify the path the data has to take in a script file. Thus, now, any changes to the output of the pass prediction can be performed by simply adding and/or adapting filters. If the filters are available, this is achievable without the need of doing code changes. Also, changes to the data flow are planned to be possible during run-time which achieves even more flexibility. The concept of the described iterations is shown in Figure 3.

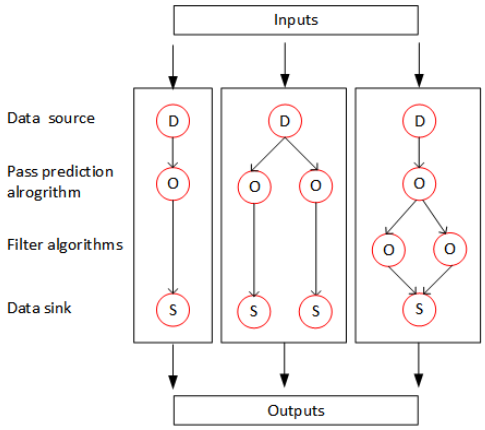

Figure 3. Schematic approach of iteratively tailoring.

This example highlights the great advantage of using a DSMS: once the complex functionalities have been implemented, it is very easy to tailor the output in regard to the user's requirements or also a user's access rights. A second example, which is yet to be fully implemented, is the propagation of objects. Depending on the clearance level, one user might be eligible to orbital data with a low accuracy, while another user might be eligible to data with very high accuracy. Using Chebyshev interpolation, as described in [5], the ephemerides are created once, but aggregation operators post-process the results according to user clearance.

Just as it is very straight forward to exchange the outputs provided, it is, from the architectural point of view, easy to add different operators with analogous functionalities. Going back to the example of the pass prediction, the current implementation provides a basic solution. To upgrade it with an advanced solution, only the processing operator, which actually performs the pass prediction, needs to be exchanged. The remaining process stays exactly the same. Which type of pass prediction to choose can be defined either in the script file or be defined based on criteria contained in the input data.

6. CONCLUSION

For certain types of SST users like NewSpace companies, hobby astronomers and research institutions, the wish for easy, automatable, flexible and scalable access to SST services seems to exist. Acquisition of huge, complex software products is often not an option due to budget constraints. Integration of open-source software is often possible, but can come with a lot of work for which not all users can find the time. Freely-available programs sometimes don't have all the capabilities needed. At places where these types of concerns can be found, a Software-as-a-Service-based usage of SST capabilities can be a viable option.

With pass prediction and numerical full-force propagation already available as beta, a possibility to try out the algorithms described via the DSMS and the REST interface described, is possible and very welcome. The authors are happy to get in contact via mail or other means. Additionally, through co-creation agreements, we currently seek to collect as much feedback as possible.

We feel committed to the idea of fostering collaboration through open-source software which is why we evaluate the possibility for the numerical, full-force orbit propagator NEPTUNE can become open-source. Such a development would begin with a light" version of NEPTUNE being augmented with new functionalities one after another. Also, we want to make open-source SST which is already available

to also become available via SaaS. For this to happen, we intend to work together with open-source projects.

ACKNOWLEDGMENTS

The authors would like to thank DLR and ESA, who supported foundations for this work under DLR contract AZA 50LZ1404 "Entwicklung eines Radar-System-Simulators" and ESA contract 4000103850/11/D/JR "Network Partnering Initiative: Definition of Orbit State, Orbit Ephemerides and Orbit Covariance Formats". Also, they would like to thank Institute of Space Systems of Technische Unversität Braunschweig for providing all the opportunities. The project OKAPI (03EGSNI203) is funded by the Federal Ministry of Economics and Energy and the European Social Fund as part of the EXIST programme.

REFERENCES

1. Alfriend K., Akella M., Lee D., Frisbee J., and Foster J., (1999). Probability of Collision Error Analysis, *Space Debris*, **1**(1), 21–35.
2. Anon., (2008). *Goddard Trajectory Determination System (GTDS), Release 2008.01*, `https://software.nasa.gov/software/GSC-15539-1`, retrieved 2018-08-16
3. Berry M., (2004). *A Variable-Step Double-Integration Multi-Step Integrator*, Virginia State University
4. Braun V., Horstmann A, (2015). *Networking/-Partnering Initiative TU-BS/ESOC*, TU Braunschweig
5. Braun V., (2016). *Providing Orbit Information with Predetermined Bounded Accuracy*, TU Braunschweig
6. Flegel S., (2011). *Final Report – Maintenance of the ESA MASTER Model*, Institute of Aerospace Systems
7. Jones H. W., (2018). *The Recent Large Reduction in Space Launch Cost*, 48th International Conference on Environmental Sciences, 8 - 12 July 2018, Albuquerque, New Mexiko, USA
8. Kebschull C., Reichstein L., Stoll E., (2017). *A Simulation Environment to Determine the Performance of SSA Systems*, Advanced Maui Optical and Space Surveillance Technologies Conference, 19.09 - 22.09 2017, Kihei, USA
9. Loschwitz, Martin, (2018). IT ohne Server – der Weg vom Cloud zum Serverless Computing, *Linux-Magazin*, **2018**(11), 20–23
10. Möckel M., (2016). *High Performance Propagation of Large Object Populations in Earth Orbits*, TU Braunschweig, `http://dx.doi.org/10.5281/zenodo.48180`
11. Müller, S., Stoll, E., (2017). *Data Stream-Centric SST System Architecture Enhancement*, 1st IAA Conference on Space Situational Awareness (ICSSA), Orlando, FL, USA, 2017, Paper IAA-ICSSA-17-00-01
12. NASA, (2012). *July 12, 1962: The Day Information Went Global*, *NASA History*, NASA Content Administrator, edited 2018, `https://www.nasa.gov/topics/technology/features/telstar.html`, retrieved 2019-01-13
13. Vallado D., (2013). *Fundamental of Astrodynamics and Applications*, Fourth Edition, Springer
14. Johnson N. L., Krisko P. H., Liou J.-C., Am-Meador P. D. (2001), *NASA's New Breakup Model of Evolve 4.0*, Advances in Space Research Vol. 28, 1377–1384