JSatOrb: ISAE-Supaero's open-source software tool for teaching classical orbital calculations

Julio HERNANZ GONZALEZ*, Thibault GATEAU*,
Lucien SENANEUCH *, Theo KOUDLANSKY *, and Patrice LABEDAN*
* ISAE-SUPAERO, Université de Toulouse, FRANCE
Email: firstname.lastname@isae-supaero.fr

***Abstract*—JSatOrb is an ISAE-Supaero's software tool dedicated to orbital calculation and designed for pedagogical purposes, with professional level features outputs. It has been initiated to find a soft which would fill the gap between local teachers developed tools and professional tools, exploiting state of the arts algorithms concerning space mechanics calculus. Even if current provided open source libraries are not fully compliant with our pedagogical requirements (simplicity, flexibility, multi-plateform and ergonomics), they provide complete and accurate calculus methods that are dedicated to a professional use. However, GUI part is not the main concern when used by space engineers which only require API access.**

Concretely, JSatorb project is open-source (MIT license) and under development. It is inspired from current full-stack implementation methods. Ergonomic and intuitivity are at stack concerning the front-end, which is mainly based on Angular (https://angular.io/) and Cesium (cesiumjs.org). Efficiency and correctness on calculus are provided by the back-end part, which relies on Orekit (https://www.orekit.org/). Developed in Java, Orekit is a space dynamics open source library. It depends only on the Java Standard Edition version 8 and Hipparchus (https://hipparchus.org/) version 1.0 libraries at runtime. It is widely used by ESA and CNES.

CONTENTS

I. INTRODUCTION

Historically ISAE-SUPAERO developped a software tool dedicated to orbital calculation and designed for pedagogical purposes: Satorb. It has been widely used for more than fifteen years in teaching master level students. It was closed software, based on VB.NET (https://code.msdn.microsoft.com/Official-Visual-Studio-f48134ec). SatOrb was designed to easily set satellites in motion and analyse its effects, manoeuvres, ephemeris... JSatOrb can be seen as an evolution of Satorb, open-source and modernised.It designed to be ergonomic, easily understantable, intuitive as a student-focused learning tool. But it should be comparable to professional software for researchers and engineers.

It is inspired from current full-stack implementation methods. Ergonomic and intuitivity are at stack concerning the front-end, which is mainly based on Angular (https://angular.io/) and Cesium (cesiumjs.org). Efficiency and correctness on calculus are provided by the back-end part, which relies on Orekit (https://www.orekit.org/).

First of all, we describe main features implemented in original Satorb software (Section II). we looked in litterature if altenative exist (Section III-A) and which astrodynamics library could be used (Section III-B). The decision on how to handle the user interface has also been taken into account (Section III-D). Afterward, we propose an implementation closer to current web developpments (Section III-D), based on a front-end providing all functionnalities to the end-user, and a back-end handling all calculus to the astrodynamics library.

II. JSATORB SPECIFICATION: AS GOOD AS OUR CURRENT TEACHING TOOL, SATORB

SatOrb is an orbit and multi-satellite, multi-station mission analysis software that allows a 2D or 3D display of satellites and constellations as well as multiple analysis of access, visibility, coverage and reports on the radio links. SatOrb was developed by C. Colongo at ISAE-SUPAERO. Other institutions have contributed to the development such as the German university TU München, [18].

The main available features of this software are summed up in the following list:

1) Create satellites which can be given very specific energetic, orbital or behavioural parameters;

2) Create constellations to group various satellites;
3) Create ground stations and upward, downward or inter-satellite links;
4) Obtain reports on the evolution of the created elements.

A further detailed list of SatOrb's main functionalities is now presented.

A. Main display

The main display presents five menus on top, each of them with their own settings. There is also the main proper display (view of the Earth and our satellites, ground stations...) Below it we have four menus to create elements and a numerical display on the right. An image of the display can be seen in figure 1.

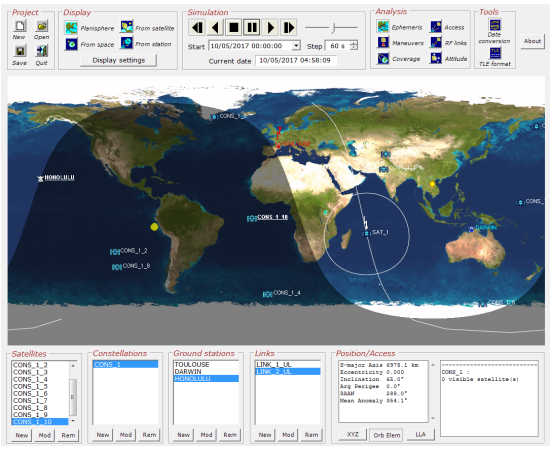

Figure 1. SatOrb's main display. Source: ISAE-Supaero.

The five menus on top are: Project, Display, Simulation, Analysis and Tools. The four boxes on the bottom allow the user to create either a satellite, a constellation, a ground station, or a link. They will be analysed in II-B. There is also a fifth box further to the left named Position/Access to visualize data. The simpler menus: Project, Display, Simulation and Tools are presented below, leaving Analysis to be further scrutinized in II-C.

1) **Project**. Simple and quick access to basic project actions: create, open, save or quit.
2) **Display**. The display mode provides four different views: Planisphere, From space, From satellite and From station. Satellite and station views will only be selectable if an element has been created. Further settings for satellite and ground station views are:
   - Set vectors to Satellite/Stations, Earth centre, Sun, and/or Moon.
   - Set the attitude of the sphere orientation.
   - Show: other satellites and/or the sphere of attitude.
   - Set traces on attitude sphere: Earth horizon, Sun, Moon, Satellite X, Y and/or Z.
   - Illuminate satellite behind Earth.
   - Fit satellite to attitude sphere.

   For planisphere a minimum elevation from ground for footprint drawing can be set and for both planisphere and Earth a choice between three Earth images is presented (Earth image, Earth with less ice and Earth by night). Day/night and clouds can be on or off.
3) **Simulation**. It allows to run the simulation: play, stop, pause, forward or backward. The start date and the time step (simulation speed) can be selected as well.
4) **Tools**. Quick access to Date Conversion and TLE format. The Date Conversion tool allows to convert a date into the Julian calendar, the CNES Julian day, the modified Julian day and the NORAD epoch. The TLE Format shows information on how the NORAD Two-Line Element Set Format functions.

B. Creation boxes

The four creation boxes on the lower part of the main display allow to create: a satellite, a constellation, a ground station, or a link. The user can create a new element (New), modify an existing element (Mod) or delete an element (Rem).

1) **Satellites**. There are four modifiable parameters.
   - *Orbit*, where the satellite is named and the orbit entered (a NORAD TLE set can also be loaded). The orbit can be:
     - Geostationary
     - Sun synchronous: altitude or inclination will be demanded.
     - Critically inclined, Sun synchronous
     - Repeating ground trace
     - Molniya
     - Critically inclined
     - Circular
     - Other
   - *Orbit parameters*. The propagator (Kepler, Brouwer, Mosaif, SGP4/SDP4), its coordinate system, and the date are chosen. Main orbital parameters are to be set too.
   - *Attitude*. Primary and secondary directions of the satellite are selected.
   - *Graphics*. Colour, the visualization of the leading or trailing track, the satellite model (cube or cylinder) and its accessories (solar panels of various sizes) are chosen.
2) **Constellations** A constellation can be created either by entering its parameters, from TLEs database or by grouping the existing satellites.
   The parameters to be introduced are: altitude, inclination, first orbit plane RAA, number of satellites (t), number of orbit planes (p) and relative spacing between satellites in adjacent planes (f). A Graphics sub-menu allows to customize the colour of the constellation or of each orbit plane.
3) **Ground stations**. An index of possible locations on Earth with the latitude, longitude and altitude data is already provided. These parameters can be manually added too. Coverage parameters (minimal and maximal

elevation and maximum range) are also to be added and a colour is to be chosen.

4) **Links**. Links may be: upward, downward or inter-satellite. The user will be asked to fill in information on the transmitting entity (power, line loss and antenna) and on the receiving entity (line loss, noise temperatures and antenna) which can be satellites or ground stations.
   For up and downlinks we must also provide the carrier (frequency, bandwidth, data rate, spectrum utilization, modulation type, and bit error rate), link losses and atmospheric losses. For inter-satellite links the link type and a fast or accurate method for Earth loss is to be selected. Antennas can be: Circular Parabolic, Horn, Helix, Dipole 1/2 Wave Patch, Vertical Whip 1/4 Wave, Yagi or Manual. Each of them with its own further settings.

C. Analysis

Analysis allows to calculate and obtain information on the behaviour of created elements.

1) **Ephemeris**. An analyse for a period of time and a time step is provided. Ephemeris can be: Position/Velocity, Keplerian Ephemeris, Longitude, Latitude and Altitude Ephemeris and Eclipse Times. Output may be a numeric report or a graph.
2) **Manoeuvres**. Create or remove manoeuvres. When created anew the final orbit or the generic speed gains may be specified as well as the time of the manoeuvre. If final orbit is chosen, the manoeuvre can be coplanar (Hohmann transfer or One-Tangent Burn), non coplanar (variation of inclination, RAAN or both), or generic. Target orbit ought to be specified too. If ΔV is to be specified it can be perpendicular to the orbit plane, tangent to the speed vector or generic (gains in all three axes).
3) **Coverage**. The statistics of coverage and figure of merit or the ground trace are presented for a satellite or constellation and for a period of time and a specific region of the Earth (or the entire globe).
4) **Access**. Access, azimuth, elevation, and range of a satellite as well as the number of visible satellites of a constellation for a selected time interval.
5) **RF Links**. A report of the link budget, the link margin, signal to noise density, the Doppler shift, Earth warmth loss, atmospheric loss or the depointing error for one of the created links and for a selected period of time.
6) **Attitude**. The latitude/longitude variations for a satellite with respect of another satellite, ground station or a third element like the Sun or the Moon for a period of time.

III. Existing Solutions

A. State of the Art - SatOrb Equivalent Softwares

A list with the main similar software packages found is presented below.

STK/AGI [11]: PC. Free and commercial versions. Systems Tool Kit (STK) is the foundation of AGI's product line. It provides four-dimensional modelling, simulation, and analysis of objects from land, sea, air, and space in order to evaluate system performance.

GMAT [15]: Cross platform, Free. NASA open source product. The General Mission Analysis Tool (GMAT) is an open-source space mission design tool developed by a team of NASA, private industry, and public and private contributors. It is used for real-world engineering studies, as a tool for education and public engagement, and to fly operational spacecraft.

FreeFlyer [10]: PC/Linux, commercial. FreeFlyer is software for space mission design, analysis and operations. It provides an astrodynamics functionality for missions analysis.

ORSA [19]: Linux/Mac/PC, free. Started in summer 2001 just as a simple collection of celestial mechanics C++ classes, the ORSA project now collects many general classes, a graphical interface running under Linux/Unix, Mac OS X and Windows, and numerous of tutorial programs.

B. State of the Art - Astrodynamic Libraries

In this section four libraries are presented: Orekit, JAT (Java Astrodynamics Toolbox), TUDAT (TU Delft Astrodynamics Toolbox), and CelestLab.

Orekit [5]. Developed in Java (cross-platform), Orekit is a space dynamics open source library. It depends only on the Java Standard Edition version 8 (or above) and Hipparchus [1] version 1.0 (or above) libraries at runtime. Its main package structure is illustrated in figure 2.

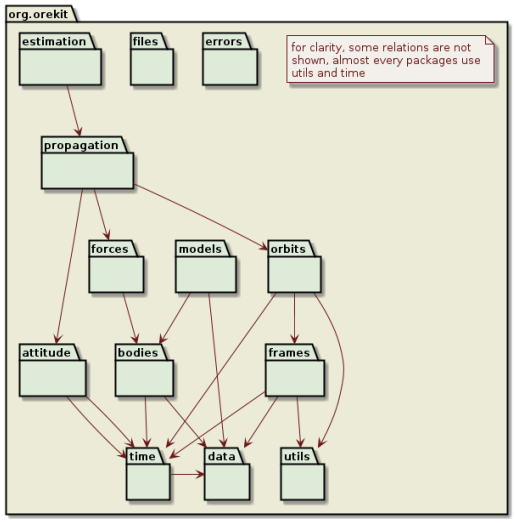

Figure 2. Orekit package structure. Source: [5]

Orekit aims at providing accurate and efficient low level components for the development of flight dynamics applications. It is designed to be easily used in very different contexts, from quick studies up to critical operations. Orbit

propagation is carried on by automatic differentiation, as analysed in [2].

Its **features** are: Time, Geometry (frames, IERS conventions...), Spacecraft State, Manoeuvres, Propagation (analytical, numerical or semi-analytical, predefined events...), Attitude, Orbit determination and file handling and various Earth Models. It also allows customizable data loading and presents several languages.

Orekit is highly recognised internationally and is currently being used by the ESA (as in [7]), or by the CNES [6], which has selected it as the basis for its space dynamics systems since 2011. It has also been used for studies by industrial actors such as EUMETSAT [8].

JAT [3]. A cross platform and open source library to help users create their own application programs to solve problems in astrodynamics, mission design, spacecraft navigation, guidance and control using Java or Matlab. Thanks to the research, it has been found out that JAT makes use of Orekit.

TUDAT [4]. A set of C++ software libraries, developed and maintained by staff and students in the Astrodynamics & Space Missions research group at the Faculty of Aerospace Engineering, Delft University of Technology, in the Netherlands. This toolbox is intended to provide users with functionalities to be able to simulate various astrodynamic applications.

CelestLab [14]. A space flight dynamics functions library developed and maintained by CNES for trajectory analysis and orbit design for various missions. It is written in Scilab language. It includes more than 200 functions, such as: orbit propagation, attitude computation, elementary manoeuvre computation, change of coordinate systems, or three body orbit analysis.

C. Synthesis

Ensuring that the chosen library fulfills all the available modules of SatOrb will justify our choice. The comparison is presented in table I. Orekit generally satisfies the needs of SatOrb and even provides new features. It lacks the visualization segment as well as the creation of links and constellations of satellites. Nevertheless, the growing importance of this library and its rising international recognition allows us to choose Orekit. Even though other JAT seem to also fulfill all the required functionalities, Orekit seems a more sensible option as it provides support and far better usability.

Regarding the absence of the visualization segment it is indeed what we were looking for as it allows us to clearly differentiate between the user interface part of the project and the calculations.

[1]Checked on GoogleScholar on 3 May 2017, duplicates included. References have also been searched on IEEE Xplore Digital Library resulting in one reference for Orekit and zero for SatOrb and the other libraries. We agree that is not a perfect metric, but that give an idea of how living is the project.

D. First Prototyping Attempts

Once we have made the choice of Orekit as the library that will engage with all the calculations it is crucial to decide how to handle the user interface. There are many available options. A couple of them have been tried and will be here presented. Keep in mind that we wanted to keep ahigh degree of modularity between the calculations and the user interface.

One of the options was language R with RStudio. The existence of a wide range of different packages available on-line makes R a suitable candidate. Furthermore, Shiny [12] for R allows to easily develop user interface applications for the web. Nevertheless, interconnection between R and Java isn't that easily handled

A simpler thing would be to keep all the development in the same programming language. When looking towards user interface development in Java a first attempt was made with Swing [16]. Swing in Java provides useful tools to create an application with a display, buttons and other elements. But being JavaFX a much newer and easier GUI widget tool kit in Java, a change to JavaFX was quickly made. In addition, JavaFX intends to replace Swing. It is also worth mentioning the use of SceneBuilder [9] to help the creation of the various elements of the interface such as buttons, menus, panes... Regarding the use of JavaFX, it must be said that there are plenty of tutorials on-line and the use of SceneBuilder facilitates the process very much. Among the many tutorials used to learn and understand how to work with GUIs are [17] or [13].

Some prototyping has therefor been decided (figure 3.

However, this solution remained as a heavy client application running, and was lacking of modularity. Dependency to java was high, and we wanted to go further in the separation between GUI part and core code. This way, we avoid to be totally relying on the interface part, or the calculus part. For all these reasons, we decided to drastically change our architecture.

IV. Webservice Architecture using a REST implementation

A. Architectural view

Nowadays more and more software propose a service approach (google drive, microsoft office, dropbox, slack). The applications running on the user side are only a graphical representation of distributed models and calculus. On a model modification, or a calculus request answer, the user interface is updated in order to always have the last informations displayed. JSatorb architecutre is build on the same idea. We have a front-end which manage user interactions and interfaces and a backend which manage data and calculus request made by the users (Fig. 4).

B. Front-end

Like explained above, front-end application is the interface which the user will interact with (Fig. 5). That mean that after the basic element of the page has been load on the browser, the web page will act like a standalone application

Feature type	Software / Features	SatOrb	OreKit	JAT	TUDAT	CelestLab
Visualization	Terminal	✓	✓	✓	✓	✓
	2D (planisphere)	✓	-	✓	-	✓
	3D (Earth)	✓	-	✓	-	-
Ephemeris	Position/Velocity	✓	✓	✓	✓	✓
	Keplerian Parameters	✓	✓	✓	✓	✓
	Eclipses	✓	✓	✓	✓	✓
Manoeuvres	Impulse	✓	✓	✓	✓	✓
	Continuous	-	✓	✓	✓	✓
Time	UTC	✓	✓	✓	✓	✓
	TAI	-	✓	✓	✓	✓
	Julian	✓	✓	✓	✓	✓
	NORAD	✓	-	-	-	-
Propagators	Kepler	✓	✓	✓	✓	✓
	Brouwer	✓	-	-	-	-
	Mosaif	✓	-	-	-	-
	SGP4/SDP4	✓	✓	✓	-	-
	Central	-	✓	✓	-	✓
	Lyddane	-	-	-	-	✓
	Eckstein-Hescher	-	✓	✓	-	✓
	Cohessy Wiltshire	-	-	✓	✓	✓
Others	TLE Format	✓	✓	✓	✓	✓
	Coverage	✓	✓	✓	-	✓
	Constellations	✓	-	-	-	-
	Ground stations	✓	✓	✓	✓	✓
	Links	✓	-	-	-	-
	Language	VisualBasic	Java	Java	C++	Scilab
	References[1]	74	88	75	52	13

Table I
SOFTWARE COMPARISON.

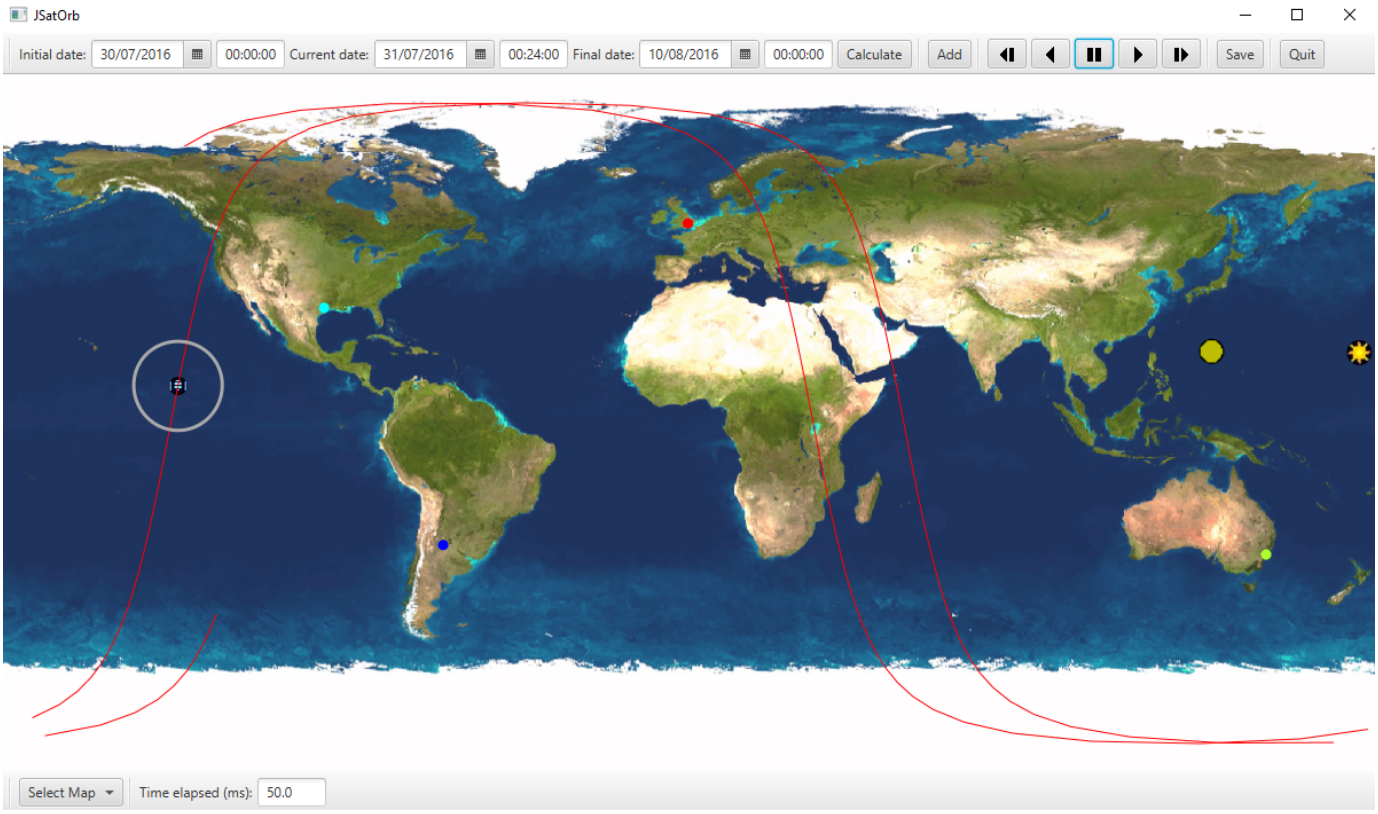

Figure 3. JavaFX JSatOrb's example view. We can see ground stations located in London, Cordoba, Houston, and Sydney. The orbit represented in this demo is a sun-synchronous orbit. The sun synchronous orbit is a circular retrograde orbit. It can be appreciated how the circle of visibility stretches and deforms itself when reaching the poles.

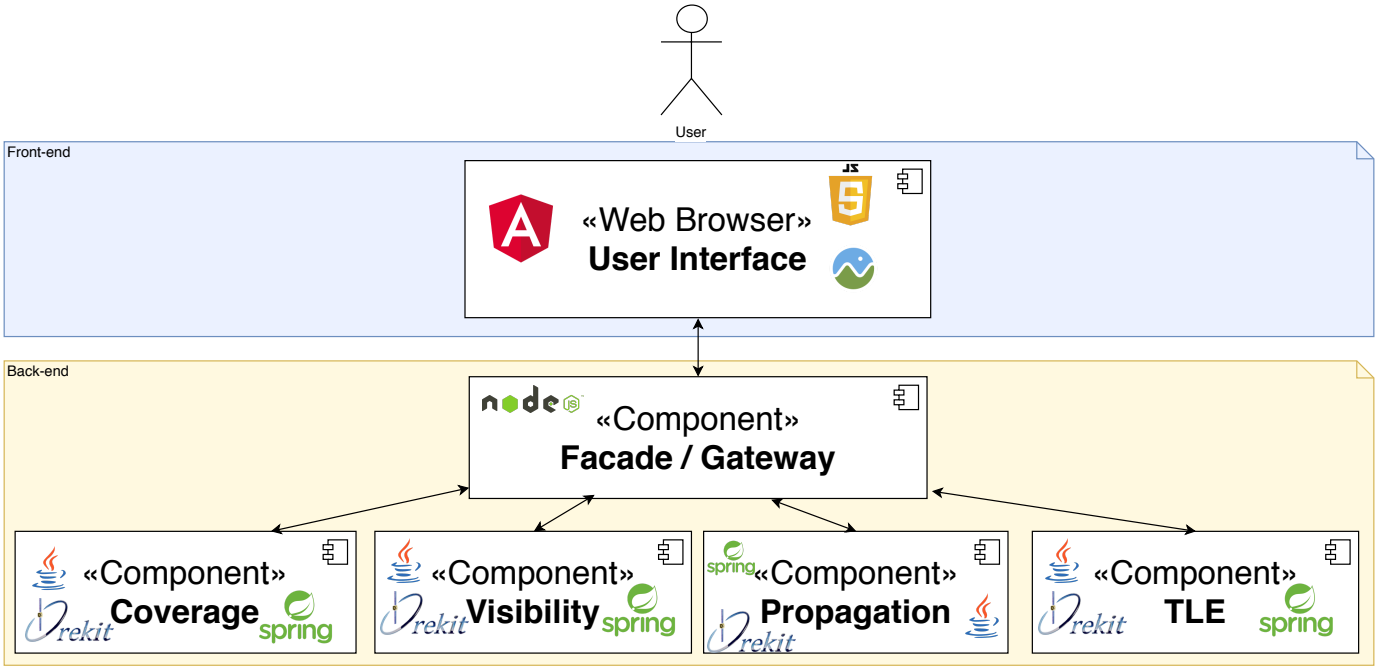

Figure 4. An Architecture representation of JSatorb and his web services

by make itself request to others components. This is called a web application or web app. The application is structure in several part re-usable called front-end components.

AngularJs has been used to develop JSatorb to facilitate the integration of the different part of the applications. It is a javascript framework which is use to separate the application in several part which are using services to communicate. In addition to the parts used to interact with the user, Jsatorb allows the visualization of orbits. Several views are available, including the planisphere view and the 3D view of the earth. The 3D view is implemented with cesium.js an open source library using graphical acceleration in the browser (WebGl).

C. Back-end

The backend is composed of multiple services components. Each one of them is used to manage one core domain. That mean that for example one service is looking after storing data, an other is calculating propagations, an other for eclipses ... Each services can be deployed in your local computer or on an other material on the same network. Component connection and communication are cornerstones of the application a protocol is needed to called distributed calculations. There are many protocols and implementations that can handle this type of task (RMI, Corba, SOAP, REST, ...). In an other hand we would like to have a cross platform application available throw a web browser. In order to make it possible, the protocol chosen need to be compatible with a web browser.

REST (REpresentational State Transfer) is based on HTTP protocol. It is fully compatible with web browser, and actually largely used in web development. Unlike SOAP protocol, there is no third party library, or tool-kit needed. As a result of this, select REST protocol will reduce coupling between components and it is easy to understand for user/developer. For example to use the propagation service to make a calculation, an HTTP Post Request has to be send to the service address with a pre-defined url. The request will include required inputs parameters to make the calculus.

```
POST myServiceAddress/propagation
{
    "initialDate": "2004-01-25T22:32:00",
    "duration": 1200 ,
    "stepT": 60,
    "a": 24396159,
    "e": 0.72831215,
    "i": 7,
    "omega":180,
    "raan": 261,
    "lM": 0
}
```

We also made the choice to use a component as a gateway between the front-end and all the others backend services. This gateway will receive request from a user interface to transfer them in one or several others services (Fig. 6). This is a facade design pattern and the user interface don't have to know every service he is using. This will simplify the interaction of the user interface.

V. Conclusions and Further Development

JSatOrb is a attempt to provide to user a software tool dedicated to orbital calculation and designed for pedagogical purposes, with professional level features outputs. It must

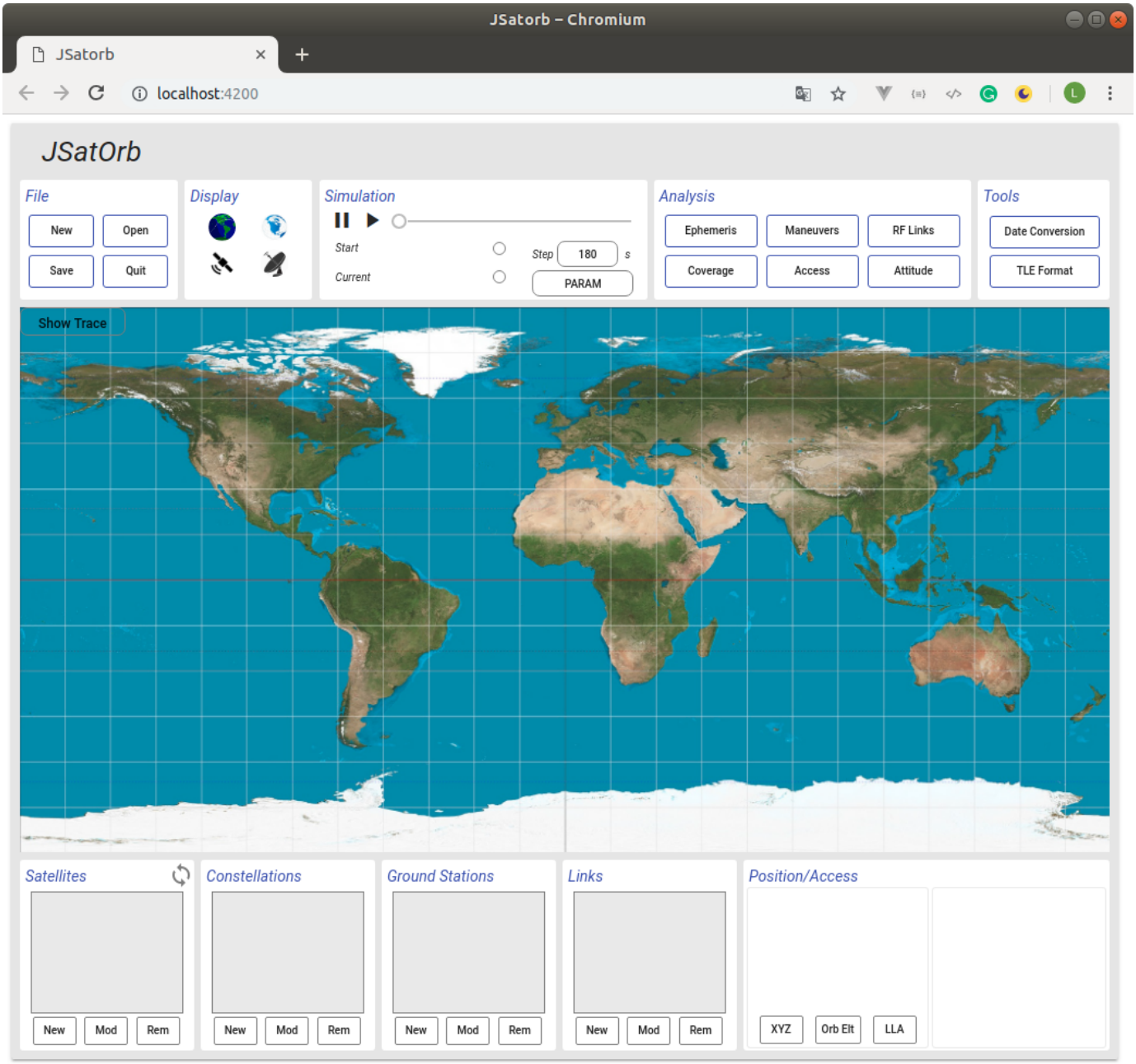

Figure 5. JSatorb user interface (front end)

be open source, cross platform, modular, with a clear differenciation between user interface and calculus code core. Litterature review allow us to select Orekit as our core calculus library. First tests confort us is orienting the development into a current web design application: a front-end for the GUI, and a back-end with call to calculus code core.

Concretely, JSatorb project is now an open-source (MIT license) and under development https://sourceforge.isae.fr/projects/jsatorb/repository. A first version is available (devellopped with Angular, based with a home-made server calling Orekit libraries but not developped as a RESTfull application yet).

We have recently discovered also Astropy http://www.astropy.org/ which could be a python alternative to Orekit. Robustness of the approach we choose allow us the possibility to switch to it with little development cost. It doesn't appear in the comparison table as we didn't know it at the period this work was achieve.

This is only the beggining of the development of JSatorb and we hope to have feedbacks from developpers and users.

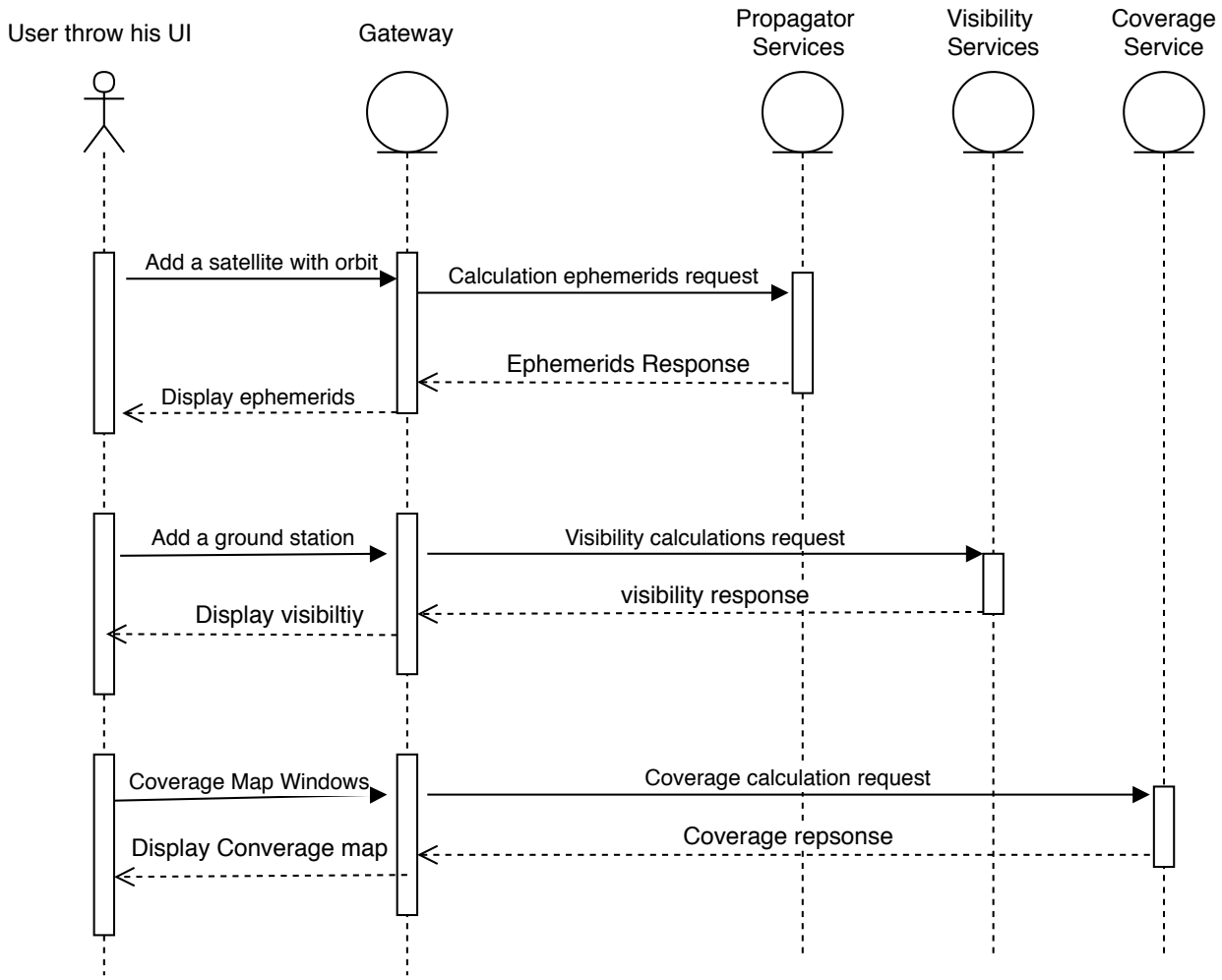

Figure 6. A sequence diagram of components interactions with satorb

REFERENCES

[1] Hipparchus site. https://hipparchus.org/. Accessed: 2017-05-02.

[2] Antolino Andrea and Luc Maisonobe. Automatic differentiation for propagation of orbit uncertainties. In *Final Stardust Conference*, November 2016.

[3] Tobias Berthold. Java Astrodynamics Toolkit. http://jat.sourceforge.net/. Accessed: 2017-03-28.

[4] TU Delft. Tu Delft Astrodynamics Toolbox. http://tudat.tudelft.nl/projects/tudat/wiki. Accessed: 2017-03-28.

[5] CS Systèmes d'Information et al. Orekit. https://www.orekit.org/. Accessed: 2017-04-18.

[6] Centre National d'Études Spatiales. Cnes. https://cnes.fr/fr. Accessed: 2017-05-02.

[7] ESA. Socis - the esa summer of code in space. http://www.esa.int/Our_Activities/Space_Engineering_Technology/SOCIS_The_ESA_Summer_of_Code_in_Space/(print). Accessed: 2017-05-02.

[8] European Organisation for the Exploitation of Meteorological Satellites. Eumetsat. http://www.eumetsat.int/website/home/index.html. Accessed: 2017-05-02.

[9] Gluon. Scene Builder. http://gluonhq.com/products/scene-builder/. Accessed: 2017-06-19.

[10] a.i. solutions Inc. FreeFlyer. https://ai-solutions.com/freeflyer/. Accessed: 2017-03-28.

[11] Analytical Graphics (AGI) Inc. Systems tool kit (STK). https://www.agi.com/products/stk/. Accessed: 2017-03-28.

[12] RStudio Inc. Shiny. https://shiny.rstudio.com/. Accessed: 2017-06-19.

[13] Marco Jakob. code.makery: JavaFX8 tutorial. http://code.makery.ch/library/javafx-8-tutorial/. Accessed: 2017-06-19.

[14] Alain Lamy, Thierry Martin, and Guillaume Azema. CelestLab. https://forge.scilab.org/index.php/p/celestlab/page/CelestLab/. Accessed: 2017-04-08.

[15] NASA. GMAT. http://gmatcentral.org/. Accessed: 2017-03-28.

[16] Oracle. The java tutorials. getting started with swing. https://docs.oracle.com/javase/tutorial/uiswing/start/. Accessed: 2017-06-19.

[17] Oracle. Java Documentation. http://docs.oracle.com/javase/8/javase-clienttechnologies.htm. Accessed: 2017-06-19.

[18] Prof. Dr. rer. nat. U. Walter. *SatOrb 2004*. TU München, 2004.

[19] Pasquale Tricarico. ORSA. http://orsa.sourceforge.net/. Accessed: 2017-03-28.