EPN2020-RI

EUROPLANET2020 Research Infrastructure

H2020-INFRAIA-2014-2015

Grant agreement no: 654208

Deliverable D6.16-Mobile Vespa application with documentation and open source code

Due date of deliverable: 31/01/2019
Actual submission date: 30/01/2019

Start date of project: 01 September 2015 | Duration: 48 months

Responsible WP Leader: Observatoire de Paris, Stephane Erard

Project funded by the European Union's Horizon 2020 research and innovation programme		
Dissemination level		
PU	Public	☐
PP	Restricted to other programme participants (including the Commission Service)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (excluding the Commission Services)	

Project Number	654208
Project Title	EPN2020 - RI
Project Duration	48 months: 01 September 2015 – 31 August 2019

Deliverable Number	D6.16
Contractual Delivery date	31/01/2019
Actual delivery date	30/01/2019
Title of Deliverable	Mobile VESPA application with documentation and open source code
Contributing Work package (s)	WP6
Dissemination level	PU
Author (s)	Carlos Henrique Brandt, Angelo Pio Rossi, Mikhail Minin

Abstract: The VESPA mapping app has been developed as a mobile and desktop web application, allowing exemplary access a selected subset of mapping and imaging VESPA datasets. The app uses Open Source libraries and frameworks and it can run on multiple platforms. The exemplary search function can be extended and preview of data products links to individual granules as external URLs and to VESPA portal query. The Mars footprint and image/cube use case makes use of planetary web mapping basemaps from OpenPlanetaryMap. When available on other planetary bodies, such as the Moon, OPM basemaps will be used as well as geocoding based on feature names.

Table of Contents

List of figures

List of acronyms and abbreviations

Acronym	Explanation
DB	Data Base
GUI	Graphical User Interface
IVOA	International Virtual Observatory Alliance
OPM	OpenPlanetaryMap
UI	User Interface
URL	Uniform Resource Locator

Introduction

The VESPA-App is a web-mobile responsive interface to improve the users experience on exploring planetary data. It is meant to be an extension, simplified version of the main VESPA query portal. While the VESPA portal provides specific options to query VO planetary services (aka EPN-TAP), the VESPA-App provides a higher level, less technical interface to explore the data in those services by offering the data products (images, footprints, spectra) at first hand, and *then* linking the user to the services through the VESPA portal.

Another concept embedded in the App -- which reinforces the its *complementary* aspect to the VESPA portal – is the continuous development through an open community. User demands may and should evolve over time. At the same time, underlying technologies will also evolve to better accomplish the continuous increase of data volume and complexity. To keep our software current to best practices and surrounding updates it is reasonable, cost-effective to make the source code open to the public and motivate people – users and developers – to contribute to their best experience.

The App is meant to present VO data through an interactive interface providing a map viewer and the respective data items geolocated over the field of view on the *map* whenever such information is available (EPN's s_region or c1, c2, c3 fields). When such information is not present -- or for other technical reason a *map* cannot be used --, data products are presented in a list-like interface with the respective information and visualization controls suitable to the items at hand.

App architecture

At the basis of the App structure there is Meteor, a framework that manages the dependencies, server/client communication and the software lifecycle. To structure the App components and reactive to data changes we make use of React.js. Data is managed by MongoDB. Interactive maps are provided by Leaflet. And Bootstrap is user to provide a responsive, elegant interface allowing the user to access the App either a mobile device or a desktop computer.

Figure 1 presents the diagram of the App major components and the flux of data within it; the figure on the right uncovers technical specifications of the conceptual figure from the left

The database

Contrary to the VESPA portal that queries EPN services in real-time, the VESPA-App caches the services data in a serviced database. The main reason for doing that is performance, to provide a better user experience. This decision is supported from the scientific point-of-view since the datasets provided by EPN services are stable enough to consider a daily update of our internal database cache a much reasonable solution.

The database is MongoDB, where each document in the DB has a sub-set of the data provided by the respective VO table, enough to (i) uniquely identify/query in the original table, (ii) geolocate the data item, (iii) provide a description or visualization for the user first-inspection.

In VO/EPN jargon, data items are *granules* (*i.e.*, table records) uniquely identified through the granule_uid and granule_gid fields. Each item has a target_name field where the respective *body* (planet, asteroid) is informed -- an EPN service may provide information of different *targets* (planets, for example) in the same table. Those are the most fundamental information used by the App to organize data through its pages. Other EPN fields and how they are used internally are discussed in the data docs.

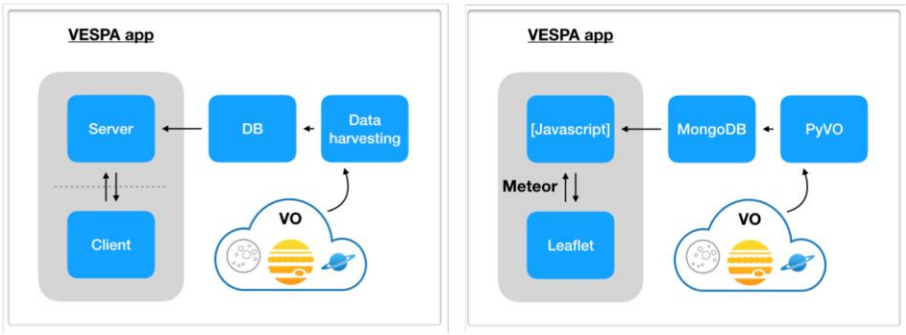

Figure 1: Software components and data workflow relation. In the left figure the conceptual components of the software are depicted, while in the right figure the underlying software technologies and main libraries are presented.

App components

The App is composed by the following components:

- Data collections
- UI
  - App
  - Map
  - List

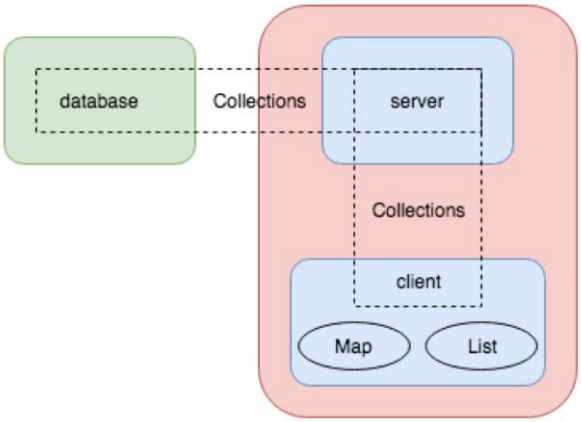

Figure 2: The internal structure of the App components and how data is communicated internally. In the software model used the App is feed by an external database

Data collections

Collections is the way Meteor communicates data between the Client and the Server, and Server with the Database. In VESPA-App the data base is managed by MongoDB, which lies in a third service, independent of Meteor. Collections in MongoDB are part of a database to which we have the freedom to name and is transparent to the App: when we start the (Meteor) Server, the MongoDB database is directly given to it through the MONGO_URL environment variable. Figure 2 is illustrative to understand the relation between the software components and data collections, exclusively managed by the server for better performance and better maintenance.

The way the Client has access to the different collections is by *subscribing* to each of them (from the Client side) and by *publishing* each of them from the Server side. This is done at the files in vespaapp/imports/api/data* (Server and Client), vespaapp/server/main.js (Server only) and vespaapp/ui/App.js (Client only).

To feed the different pages of the App, it is reasonable to ask the Server to do the hard work whenever possible; this increases the communication between Client and Server but reduces the amount of data transmitted to the necessary only. Also, depending on the query performed -- like aggregations or spatial queries -- only the Server can perform, since the Client runs a minimal version of MongoDB API (minimongo).

Geospatial-aware collections

Data sets containing geolocated *granules* -- entries providing non-null s_region or c1,c2,c3 fields -- are stored in a dedicated collection inside MongoDB (currently named CRISM).

MongoDB provides support for spatial queries: to query for points *within* a certain distance, or *intersecting* polygons, for example, over the surface of a planet. To make use of such feature, documents in the data base have to be indexed accordingly using MongoDB's '2dsphere' index. All documents of such data base (*i.e.*, collection) must provide such "geolocation" information.

In such collection of our App database, it was chosen to normalize such information (s_region and c1,c2 polygons) under a field named geometry. In other words, data sets containing geolocated information must be preprocessed before their ingestion into MongoDB. This preprocessing is straightforward though: for each *granule*, move (or copy) the required information from either the s_region or c1,c2 fields and put under geometry.

The field geometry is composed by two sub-fields coordinates and type which specify the array of coordinates and type of geometry being represented (point, line, polygon). Details of the structure of these fields are documented in data/.

App

The vespaapp/imports/ui/App.js component is the entry point of data and initialization of pages of the App. In a React.js software data always go (naturally) down in the components structure. It is in App.js then that data is queried and loaded from the (Meteor/DB) collections; then passed as props to Map/List components.

Routes

When a client starts the App, routes to the multiple pages are initialized. Such routes are defined in vespaapp/imports/startup/client/routes.js and coupled to the targets defined in the homepage at vespaapp/import/ui/Home.js.

Map

The vespaapp/imports/ui/Map.js component is responsible for displaying the *map-canvas* using Leaflet. Leaflet manages DOM nodes independently of React, which requires a slightly different design of the Map component when compared to a pure React class/function. To make them work in harmony, React Component's lifecycle has to be observed; For instance, one will notice that the instantiation of Leaflet's layers (map, markers, etc.) is done inside componentDidMount(), while render()was left with a placeholder. An analogous situation happens with the Slider.js component, where jQuery is required to control the widget.

The Map component is connected to the (data) collection (crism) interface indirectly through the Meteor.Session global storage structure to allow for geo/spatial queries. As the user moves the map boundaries (either by zooming or shifting the basemap), the database is queried for the features (*e.g.*, footprints) *intersecting* the site area. Meteor Session objects are reactive components, which means that updates to any variable(s) stored in it will promptly trigger a reaction on *any* component listening the respective variable(s).

In the Map component a Session.set() call will perform the update of the global variable bbox with the updated boundaries of the map after a user interaction. Consequently, in the App.js component where there is a Session.get(bbox) statement, a new query (or *subscription* to be more precise) to the crism collection will request for new data from the server, which will update the list of *granules* presented to the user.

List

In its simplest, the App will display a list of data items. If the user, for example, ask for data from Jupiter all we can display is a list of images, for instance.

The vespaapp/imports/ui/List.js component, though, implements a somewhat sophisticated system for rendering items on demand, only the items that fit in the screen/viewport are rendered. That behavior provides an optimization regarding the user/client resources (namely memory and bandwidth), guarantees stability of the software, and ultimately improves the user experience. Notice that the number of documents (*i.e.*, *granules*) retrieved from the database may go up to the thousands, if not properly handled (as it *has* been done) the client may simply go out of resources and crash.

Database management system

To manage the data sets used by the app we are using MongoDB.

Data sets

From the point-of-view of user interaction -- and so the way data is presented -- there are two major groups of our data sets:

- those with geolocated information (EPN-Core's s_region)
- those without geolocated information (empty s_region field)

The s_region field

The s_region field in EPN/TAP services provide geometric information about a data record (measurement, topography), typically over a planet, in ground coordinates -- Longitude, Latitude -- of the respective body. In general, s_region is an array of coordinates (lon,lat) representing a polygon (may as well be any other geometry -- line, point).

The difference from the database point-of-view is in the index. For efficiency and consistency in operating geometric queries (*e.g.*, "intersect"), the geometric figures are indexed accordingly in the database.

MongoDB knows about geometric objects and provide support for geometric queries. Collections of objects with geometric information may be indexed with a 2dsphere index -- given the documents provide a type field next to coordinates.

Documents suitable to geometric queries will have the following fields ("location" may be substituted by any other string, *e.g.* "loc"):

```
{

  "location": {
    "coordinates": <array> ,
    "type": <string>
  },

}
```

Where coordinates are an array of [lon,lat] coordinates as explained in the GeoJSON reference page, and type must be one of GeoJSON types:

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon

VESPA data download

Here you will find the list of services suitable for the app, and the files (script, config, schema, etc.) to download data.

- The selected services with data suitable to our app are in:
  - services.json.
- Those services are a selection (of suitable *schemas*) from:
  - virtualRegistry.json
- The columns (*i.e.*, data) we want to download are defined in:
  - service_columns.json
- The (python) script responsible for downloading data is:
  - download_data.py
    - Besides the columns in service_columns.json, an extra field schema_epn_core is added by the script to the output (JSON) file
- If using Anaconda Python distro, an environment ("vo") is in:
  - conda_environment.yml
  - In any case, all we need to use the download script is:
    - python (current at 3.7.1)
    - pyvo (current at 0.9.2)
    - pandas (current at 0.23.4)

App usability

Examples of the app usability on a small-screen device, such as a smartphone, are provided in Fig 3-6. Larger devices, such as desktop-based browser can be used in a similar way. While the App provides a lightweight and interactive interface to the users, when the users find the data resource of interest, they are redirected to the respective data service providers for further data specific analysis.

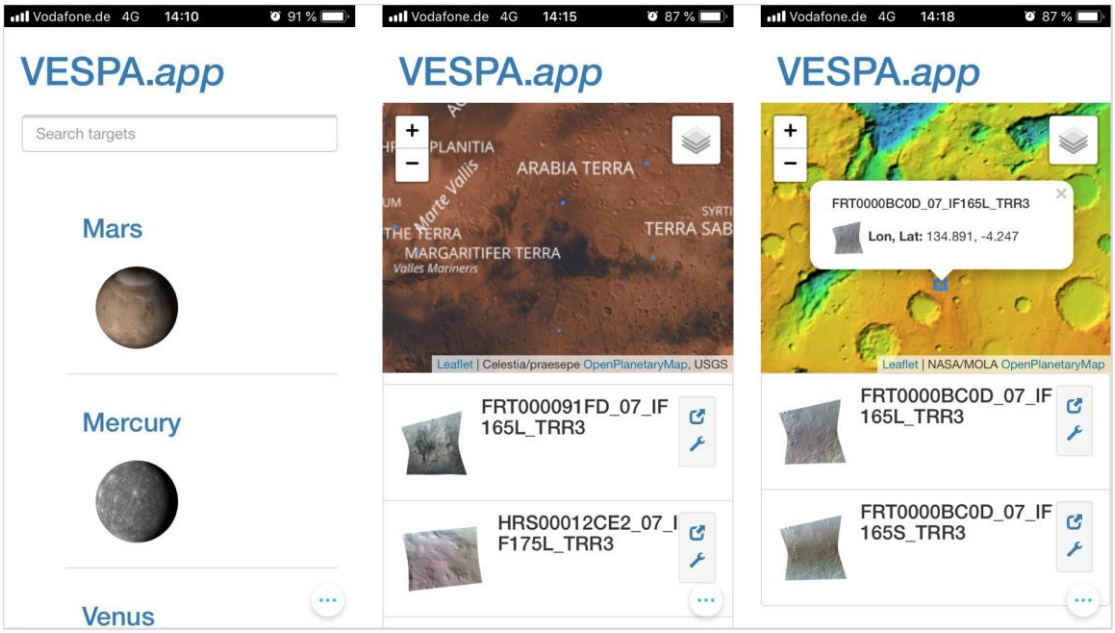

Figure 3: Examples of VESPA web app GUI for Mars. Left: Entry page. Center: Mars OPM-based map and data product preview, with links to VESPA granule pages. Right: OPM-based browsable map view and footprint identification and different background basemaps.

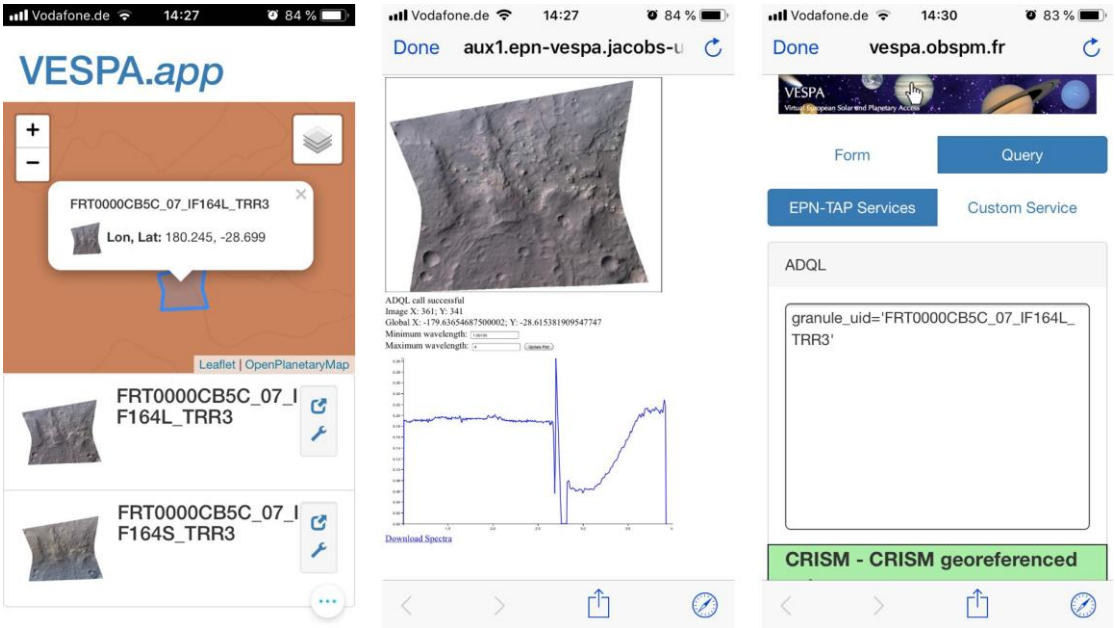

Figure 4: Examples of VESPA web app GUI for Mars and linked pages. Left: VESPA app with exemplary footprints loaded. Center: JacobsUni service with access to individual spectra. Right: link to VESPA portal, usable also via mobile

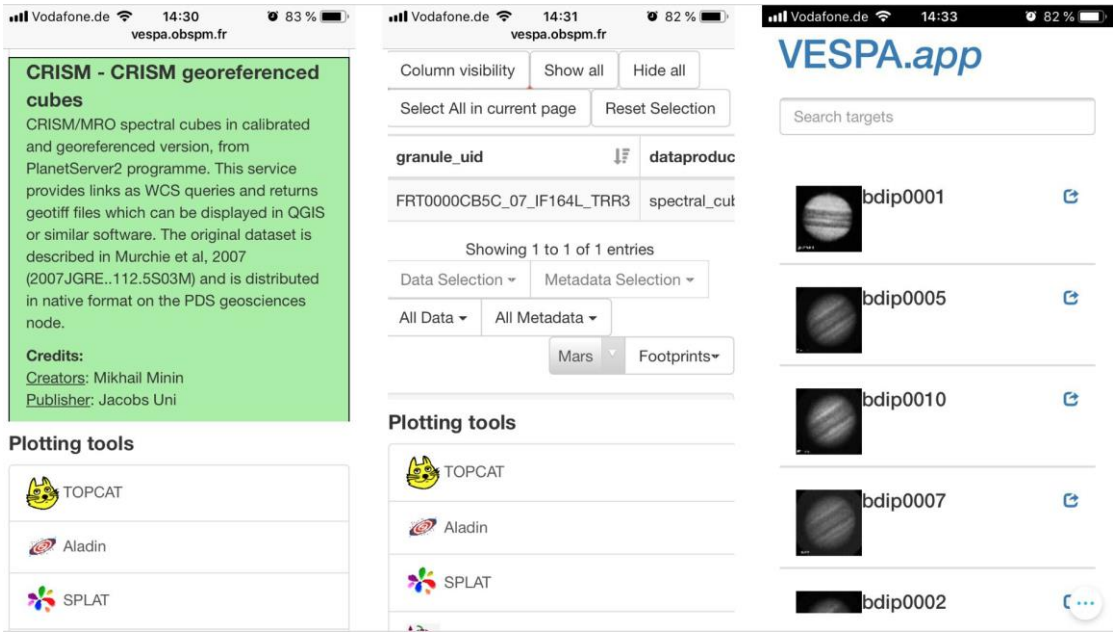

Figure 5: Left, center: Examples of VESPA linked granule/data product web pages. Right: Jupyter full disk images previewed in the VESPA app

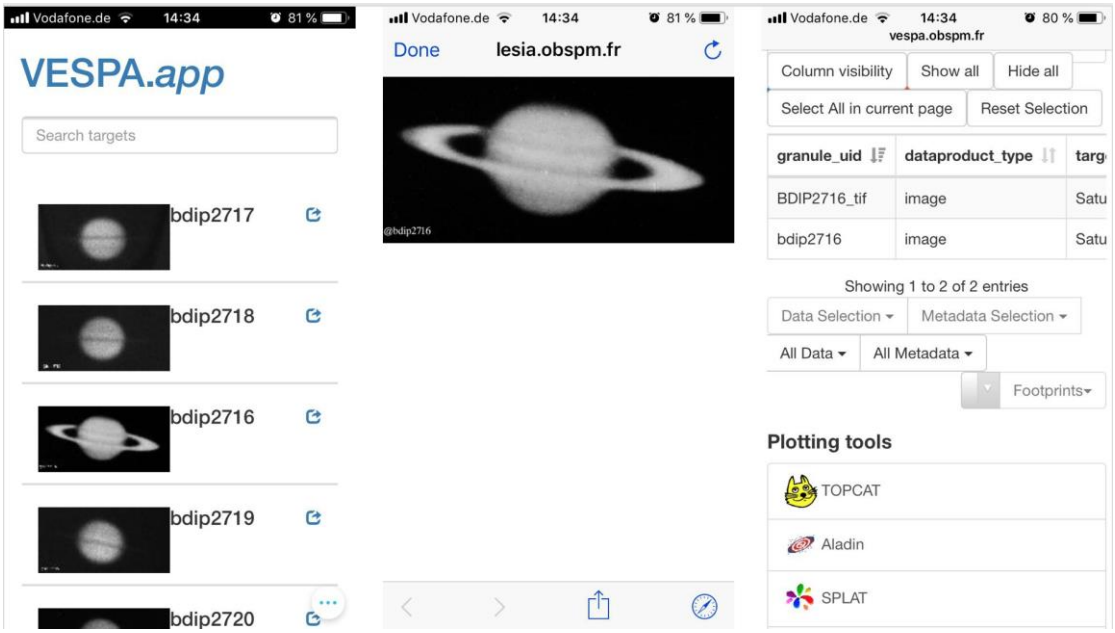

Figure 6: Left: Saturn full disk astronomic observations. Center: link to the individual data product page/browse. Right: VESPA portal entry for the data.

Future applications

The VESPA app showcases some features to interactively discovery, search and preview of VESPA data. Suitable for mapping datasets, such as CRISM

footprints, it can also support non-geospatial mapping via simple browsing e.g. of full disk telescopic imagery. The feature of granule filtering by spatial range has been added and it will be further developed. The support for geospatially-enabled FITS (see D11.10; Marmo et al., 2019) is also going to be evaluated and tested, in order to further enable interporability across GIS and VO (See Minin et al., 2019).

Code repository

The code is publicly available through the GitHub platform at https://github.com/epn-vespa/vespaapp, under the dedicated EPN-VESPA repositories directory. The repository is organized at the top level as follows:

- data/
  - Necessary files to build the data base: utility software for downloading and ingesting data into the database, as well as the documentation to accomplish that.
- docker/
  - Optional documentation, setup files and script to run a test environment through Docker containers.
- docs/
  - Documentation meant for developers and technical users.
- vespaapp/
  - The App source code

Reference libraries

- CRISM, http://epn1.epn-vespa.jacobs-university.de/tableinfo/crism.epn_core
- BDIP, http://lesia.obspm.fr/BDIP/
- International Virtual Observatory Alliance, http://www.ivoa.net/
- OpenPlanetary project, https://www.openplanetary.org/
- Leaflet, https://leafletjs.com/
- Meteor, https://www.meteor.com/
- React, https://reactjs.org/
- MongoDB, https://www.mongodb.com/
- Docker, https://www.docker.com/

References cited

Marmo, C., Hare, T., Erard, S., Minin, M., Pineau, F.-X., Zinzi, A., Cecconi, B., Rossi, A.P. (2019) FITS format for planetary surfaces: definitions, applications and best practice, Earth and Space Science, DOI:10.1029/2018EA000388

Minin M., Rossi, A. P., Marco Figuera, R., Unnithan, V., Marmo, C., Walter, S., Demleitner, W., Le Sidaner, P., Cecconi, B., Erard, S., Hare, T. M. (2019) Bridging the gap between Geographical Information Systems and Planetary Virtual Observatory. Submitted to Earth and Space Science (special section Planetary Mapping: Methods, Tools for Scientific Analysis and Exploration), DOI:10.1029/2018EA000405.

Rossi, A. P., Erard, S., Marmo, C., Minin, M., Brandt, C. H., Fernique, P. (2018) VO-GIS interface and potential application to space data archives. EuroPlanet H202 RI deliverable D11.10, availavle online on http://www.europlanet-2020-ri.eu/research-infrastructure/public-deliverables