VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Programme for the Post-processing and Analysis of Complex Large-Scale Spectroscopic Surveys Using the Virtual Observatory Protocols

Program pro post-processing a analýzu komplexních rozsáhlých spektroskopických prohlídek v rámci protokolů Virtuální observatoře

2016 Bc. David Andrešič

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Diploma Thesis Assignment

Student: **Bc. David Andrešič**

Study Programme: N2647 Information and Communication Technology

Study Branch: 2612T025 Computer Science and Technology

Title: Programme for the Post-processing and Analysis of Complex Large-Scale Spectroscopic Surveys Using the Virtual Observatory Protocols
Program pro post-processing a analýzu komplexních rozsáhlých spektroskopických prohlídek v rámci protokolů Virtuální observatoře

The thesis language: English

Description:

The goal of this Master Thesis is the development of new functionalities as well as fundamental refactoring of current SPLAT-VO programme facilitating the preview, post-processing and advanced analysis of large amount of astronomical spectra obtained from archives of large scale spectra surveys (e.g. SDSS or LAMOST) using protocols of Virtual Observatory. We expect the student to become a member of a wider international development team using standard software methodologies, implementing and testing the required parts of the programme in accordance with priorities and schedule established during regular teleconferences and virtual (e.g. Skype) meetings. The main development language is Java with embedded C/C++ modules and
libraries.

1. The identification of limits and bottlenecks of current version of SPLAT-VO with respect to the processing of large spectra sets (of order of thousands spectra).
2. Refactoring of the main data handling units to allow the simultaneous analysis and processing of pre-configured groups of spectra related by their physical or semantic properties (the containers may be nested).
3. Modification of visualisation capabilities to allow the customisation of output graphics driven by semantic contents (delivered by VO service) as well as user interaction (e.g. save and restore of visualisation parameters for individual groups).
4. Correction of minor errors and inconsistencies of UX design.
5. Implementation of selected algorithms required for efficient quality check as well as final science output. Functions considered here may be related to normalisation of spectra, dynamic quotient and differential spectra, Stokes polarimetry mode, visualisation of echelle, multi-fibre and namely IFU spectroscopy. The simple datacubes analysis is foreseen as well.
6. Customisation of query GUI for experimental visualisation of photometric light curves using the modified SSAP protocol as well as synthetic stellar spectra using TSAP and S3 protocol of IVOA.
7. Preparation of development documentation and user manual and its regular updating - synchronised within all the team of developers.

We expect the (even partial) results to be presented at the IVOA interoperability meetings as well as published in appropriate form (web, arXiv preprint, technical proceedings of or SPIE).

References:

[1] McDowell, Jonathan; Tody, Doug; Budavari, Tamas; Dolensky, Markus; Kamp, Inga; McCusker, Kelly; Protopapas, Pavlos; Rots, Arnold; Thompson, Randy; Valdes, Frank; Skoda, Petr; Rino, Bruno; Derriere, Sebastien; Salgado, Jesus; Laurino, Omar; IVOA Data Access Layer, the; Data Model Working Groups: IVOA Recommendation: Spectrum Data Model 1.1
http://adsabs.harvard.edu/abs/2012arXiv1204.3055M
[2] Škoda, Petr: Common Methods of Stellar Spectra Analysis and their Support in Virtual Observatory. Proceedings of EURO-VO Workshop Astronomical Spectroscopy and Virtual Observatory, ESAC, 21-23 March 2007, Villafranca del Castillo, 97-104. 2011
http://adsabs.harvard.edu/abs/2011arXiv1112.2787S
[3] Škoda, Petr: Multi-line Analysis of Stellar Spectra in the VO Environment. 4 pages, 2 figures; Proceedings of EURO-VO workshop Multiwavelength Astronomy and Virtual Observatory, ESAC, Villafranca del Castillo, Spain, 1-3 December 2008,
http://adsabs.harvard.edu/abs/2011arXiv1112.2788S
[4] Škoda, P.: Optical Spectroscopy with the Technology of Virtual Observatory. Baltic Astronomy, Vol. 20, p. 531-539, 2011
http://adsabs.harvard.edu/abs/2011BaltA..20..531S
[5] SPLAT-VO: Spectral Analysis Tool:
http://www.g-vo.org/pmwiki/About/SPLAT
[6] http://star-www.dur.ac.uk/~pdraper/splat/sun243.htx/sun243.html

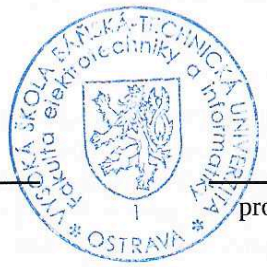
Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. RNDr. Petr Šaloun, Ph.D.**

Consultant: RNDr. Petr Škoda, CSc.

Date of issue: 01.09.2014

Date of submission: 29.04.2016



doc. Dr. Ing. Eduard Sojka
Head of Department

prof. RNDr. Václav Snášel, CSc.
Dean of Faculty

I hereby declare that this master's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, April 29, 2016 ..... Andreši

I hereby agree to the publishing of the master's thesis as per s. 26, ss. 9 of the Study and Examination Regulations for Master's Degree Programmes at VŠB – Technical University of Ostrava.

Ostrava, April 29, 2016 .....

I would like to thank all people that helped me with this thesis, namely, Petr Škoda for his patience, passion and scientific advices, Petr Šaloun for his attitude, patience and publications help and Jiří Nádvorník for his help with time series protocol and data model.

I would also like to thank other members of SPLAT-VO development team: Margarida Castro Neves, Markus Demleitner, Peter W. Draper and Mark Taylor for consulting.

For the assistance on a Nostradamus 2015 conference contribution where part of this thesis was presented, I would like to thank Ivan Zelinka.

And finally, I would like to thank my girlfriend Radka for her never ending support since we know each other.

Abstract

SPLAT-VO is a leading stellar spectra analysis tool that allows displaying, modifying and analysing astronomical spectra. It was developed in 2003 as a part of Starlink project and during its lifecycle, it was extended to include facilities that allows an interoperability with the Virtual Observatory.

At these days, SPLAT-VO also serves as a reference implementation of new Virtual Observatory protocols and data models and is being enhanced in a way of interaction with other tools and collaboration, as well as user experience.

My adjustments were accepted by community enhancing its current capabilitites and will be a part of the next release. Expert community has been partially notified about my results at Nostradamus 2015 conference and will be fully notified at SIMS2016 conference, where I will submit my actual results with deadline after finishing this thesis.

Key Words: Spectrum, time series, astroinformatics, Virtual Observatory, SPLAT-VO, Java, data cubes

Contents

List of Symbols and Abbreviations

VO	–	Virtual Observatory
HTTP	–	Hypertext Transfer Protocol
SOAP	–	Simple Object Access Protocol
WSDL	–	Web Services Description Language
REST	–	Representational state transfer
XML	–	Extensible Markup Language
DNS	–	Domain Name System
ISO/OSI	–	International Organization for Standardization / Open Systems Interconnection
SAMP	–	Simple Application Messaging Protocol
SSAP	–	Simple Spectra Access Protocol
ObsCore	–	Observation Data Model Core
SDM	–	Spectral Data Model
TAP	–	Table Access Protocol
SIAP	–	Simple Image Access Protocol
URL	–	Uniform Resource Locator
ADQL	–	Astronomical Data Query Language
SQL	–	Simple Query Language
GNU/GPL	–	GNU General Public License
FITS	–	Flexible Image Transport System
NASA	–	National Aeronautics and Space Administration
CI	–	Continuous Integration
CVS	–	Code Versioning System
IDE	–	Integrated Development Environment
LDAP	–	Lightweight Directory Access Protocol
OOD	–	Object-oriented Design
OOP	–	Object-oriented Programming
API	–	Application Programming Interface
UML	–	Unified Modeling Language
CPU	–	Central Processing Unit
OLTP	–	On-Line Transaction Processing
OLAP	–	On-Line Analytical Processing
ERP	–	Enterprise Resource Planning
ETL	–	Extract, transform, load
GUI	–	Graphical User Interface
MIME	–	Multipurpose Internet Mail Extensions

WYSIWYG	–	What You See Is What You Get
AI CAS	–	Astronomical Institute of the Czech Academy of Sciences

List of Figures

Listings

1 Introduction

As a result of technical development during last decades, many fields of science were digitized in order to use the growing power of computers, supercomputers and their grids. Since then, instruments and computers produce large amounts of data that increase exponentially and at these days, scientists face a real data avalanche.

In order to solve this, an appropriate data and computational structures were created. In astroinformatics, we have Virtual Observatory that with its data archives, server-side processing capabilities, and specialized protocols, as well as client applications, allows to handle these amounts of data efficiently.

One of these client applications is SPLAT-VO that was (at the beginning) designed for stellar spectra analysis and that is also a primary subject of this thesis. Since the beginning, it is one of the best tools for stellar spectra analysis that can use the power of Virtual Observatory. Several years ago, its development was basically took over by Heidelberg University and Czech Academy of Sciences, under which we are attempting to improve it in a way of user experience, handling large amount of data, adding support for time series and making it a reference implementation of new Virtual Observatory protocols.

In the near future, we plan to finalize the work on time series protocol and data model implementation, general refactoring of user interface that should allow more organized work with SPLAT-VO and to improve loading large amounts of spectra and time series, as well as other minor tweaks of user interface.

2 Astroinformatics and Virtual Observatory

When informatics reached borders of other fields of science, new disciplines emerged. During last decades, we newly recognized bioinformatics, geoinformatics, and basically any other X-informatics (where X stands for any other science) [1].

In case of astronomy, the astroinformatics helps to handle large amounts of data produced by observational instrumentations (reaching petabytes per each observing night) and supercomputer simulations that are impossible to process on a single local machine [2]. For example, these are numbers for the several famous projects:

- Sloan Digital Sky Survey (SDSS) Data Release (DR) 12: 116 Terabytes[1];
- Large Synoptic Survey Telescope (LSST): 15 Terabytes every night[2];
- Pan-STARRS: several terabytes every night[3].

As we can see later, the Sloan Digital Sky Survey is only one possible data source of many.

In this sense, using the power of supercomputer grids and specialized communication protocols, astroinformatics helps to traditional astronomy with data-to-knowledge transformations, information visualization, knowledge extraction, sky-based and catalog-based indexing techniques, data mining and knowledge discovery, data-intensive computing, and astrostatistics [1].

Basically, we can say that astroinformatics is the new data-oriented paradigm for 21^{st} century astronomy research [1].

2.1 Virtual Observatory

Virtual Observatory (VO) is the vision that astronomical data sets and other resources should work as a seamless whole [3]. It is a collection of interoperating data archives and software tools which utilize the internet to form a scientific research environment in which astronomical research programs can be conducted [4].

For an astronomer, there is a great overview of Virtual Observatory benefits at GAVO[4] Wiki site[5].

VO is being guided and standardized by IVOA[6], a world-wide organization that shelters individual country organisations (e.g. GAVO). Except the work on new standards and protocols, IVOA also organises Virtual Observatory conferences and regular inter-operability meetings all around the World, where our results were also published.

[1] Taken from SDSS homepage: `http://www.sdss.org/dr12/data_access/volume/`.

[2] Taken from LSST homepage:`http://www.lsst.org/`.

[3] Taken from Pan-STARRS homepage: `http://pan-starrs.ifa.hawaii.edu/public/design-features/data-handling.html`.

[4] German Astrophysical Virtual Observatory. Homepage: `http://www.g-vo.org/`

[5] `http://www.g-vo.org/pmwiki/About/About`

[6] International Virtual Observatory Alliance. Homepage: `http://ivoa.net/`

2.1.1 Architecture

From technical point of view, VO is a world-wide ecosystem of mutually compatible data sets, resources, services, and software tools which use a common set of technologies and a common set of standards [3]. Many of the VO systems communicate with each other via custom application-layer[7] protocols (using especially HTTP, SOAP/WSDL or REST) stated by IVOA that heavily use XML format for transferred messages and data. The VO infrastructure is based on a concept of [3]:

- *Resources* that represents databases of any kind with some standardized metadata about themselves;
- *Services* as processing nodes (e.g. data service querying a database behind and/or processing obtained data, communicating with other systems via specialized and standardized protocols, and publishing metadata about itself).

On the side of user interaction, VO acts according to a typical client-server model: client sends its request to a server via a common communication protocol and server responds appropriately. To hide the complexity of VO infrastructure from an user, there is a built-in mechanism in VO for discovering resources. This mechanism is called VO Registries[8] and it acts like a common internet DNS: it holds all identifiers and metadata of all known VO resources. Every resource that is supposed to be available in VO needs to be registered in at least one publishing registry (this is done by XML record with unique identifier and other resource metadata). Other registries share these metadata among themselves (similar to DNS) via web services. When a client application wants to query some resource, it discoveres available resources via searchable registries (again, via web service).

For a better idea about positioning of resources, services, protocols, data models and client applications in VO architecture, see Fig. 1 from *IVOA Architecture* specification [5].

2.1.2 Protocols

As said before, Virtual Observatory elements heavily communicates with each other via specialized protocols. In this section, most commonly used protocols called by client applications will be shortly presented.

SAMP - Simple Application Messaging Protocol is used as a part of *XML-based, event-driven publish/subscribe messaging system*. It allows to *publish* some generic weak-typed data in XML format to all registered *subscribers* that are interested in the published *message type*.

[7] Of ISO/OSI model - see for example `http://www.cisco1900router.com/what-is-ios-model-the-overall-explanation-of-ios-7-layers.html`

[8] For full specification of VO Registry, see `http://www.ivoa.net/documents/RegistryInterface/20091104/REC-RegistryInterface-1.0.pdf`

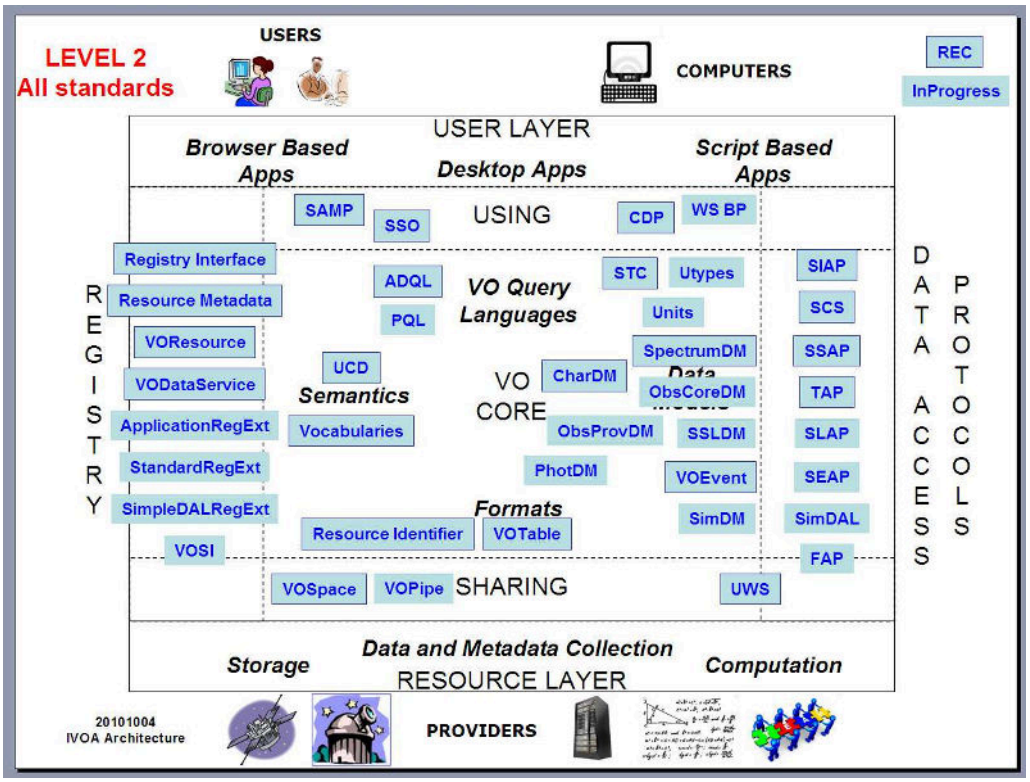


Figure 1: IVOA Architecture Level 2. Source: [5].

The publishing can be *synchronous*[9] as well as *asynchronous*[10] and the *message* can be broadcasted to all subscribers (interested in the published *message type*) or sent to an individual subscriber only.

As shown on Fig. 2, the entire communication is centralized via *SAMP hub*. An SAMP-interested application uses a *discovery mechanism* to look for a *hub* on the same network (and the *hub* itself using the same mechanism guarantees that there will be only one running *hub* on the network). If no *SAMP hub* is discovered, the application launches its own one. Otherwise, it *subscribes* to the discovered *hub*, providing its *name* and *message types* that it is interested in and asks the *hub* for information about other registered applications. When a *publisher publishes* a *message*, the *hub* will decide (based on the interested *message types* of *subscribers*) which *subscribers* will be notified. A notified *subscriber* then asks *hub* for the *message* from the *publisher*. Details about SAMP protocol can be found in [6].

SSAP - Simple Spectra Access Protocol is used to remotely discover and access one-dimensional spectra [7]. To access the *spectra data sets*:

1. The *client* queries the *global resource registry* (see 2.1.1) in order to find *services* of interest [7].

[9] Waiting for a response from subscriber.
[10] Not waiting for any response from subscriber

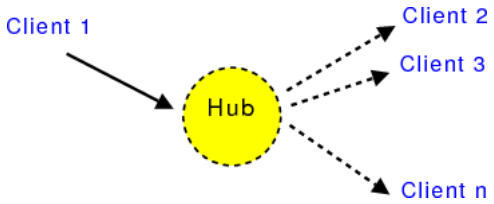


Figure 2: The SAMP hub architecture. Source: [6].

2. The *client* makes a *data discovery query* to selected *services*.
3. The *service* returns a *VOTable* (see 3.1.4) with result *data sets metadata*.
4. The *client* retrieves selected *data sets* via *access reference* in *metadata* [7].

The query for selected *services* is in the following format [7]:

```
http://www.myvo.org/ssa?REQUEST=queryData&POS=22.438,-17.2&SIZE=0.02
```

As it can be seen, the query may be parametrized with interested data set format, position, size, band, time and some optional parameters [7].

The VOTable returned by queried service may contain some significant columns identified by *Spectral Data Model (SDM)*: a part of SSAP that attempts to unify the meaning (identified by *UTYPE* [7] and *UCD* [7]) of results from various resources.

There is a lot of other query parameters and resulting VOTable details behind SSAP protocol that are beyond the scope of this thesis. If interested, please refer to full SSAP specification in [7].

ObsCore - Observation Data Model Core is used as a data discovery and data access protocol for observation data of basically any kind (so unlike, for example, SSAP, ObsCore does not focus on spectra only). Nevertheless, it aims at providing a simple model easy to understand and to implement by data providers that wish to publish their data in the Virtual Observatory [8]. To do so, it defines lists of mandatory table names etc.

Other IVOA developed many protocols for different purposes. Their complete list is beyond the scope of this thesis[11], anyway, the following list summarizes other most used and significant of them:

SIAP - Simple Image Access Protocol is used for image data access from a variety of astronomical repositories through a uniform and reasonably simple interface [9]. Candidate images are retrieved as a list in *VOTable* format (see 3.1.4) with *access reference URLs* based on query defining a rectangular region on the sky [9]. Images themselves can be in various formats, including FITS (see 3.1.3).

[11] More complete list can be found at: `http://www.ivoa.net/documents/`

TAP - Table Access Protocol is a protocol for accessing general table data, including data catalogs as well as general database tables (including metadata) [10]. TAP supports multiple query languages, including *ADQL (Astronomical Data Query Language)*[12] - a specific query language for astronomical purposes derived from standard *SQL* by IVOA. Results are returned as VOTable (see section 3.1.4) as usual in IVOA protocols. TAP also supports simple spatial cross-matching [10].

ConeSearch is a simple query protocol for retrieving records from a catalog of astronomical sources [11]. The query describes sky position and an angular distance, defining a cone on the sky and response returns a list of astronomical sources from the catalog whose positions lie within the cone, formatted as a VOTable (see 3.1.4) [11].

2.2 Tools

This section covers a list of most used and most popular tools compatible with VO, especially those that immediately relates with the aim of this thesis. The complete list of VO applications can be found at IVOA's *VO Applications for Astronomers* page[13].

2.2.1 SPLAT-VO

SPLAT-VO is a spectral analysis tool, that received a VO support several years ago. For more information, please see section 4 that is dedicated to it, since SPLAT-VO and its enhancements are the aim of this thesis.

2.2.2 TOPCAT

TOPCAT is a shortcut for *Tool for OPerations on Catalogues And Tables* [12] and it is a powerful interactive graphical viewer and editor for (not just astronomical) tabular data [12]. It supports data visualization and various data formats and protocols including SAMP which makes it a perfect cooperative tool for use cases that are not covered by TOPCAT itself.

It is a part of *Starlink* and its *Starjava* project as well as SPLAT-VO (see section 4) and is available under GNU/GPL license[14].

2.2.3 Aladin

Aladin Sky Atlas is a tool for interactive accessing to astronomical images with connection to *Simbad*[15] and *VizieR*[16] services. It is developed by the *Centre de Données astronomiques*

[12] ADQL specification: `http://www.ivoa.net/documents/latest/ADQL.html`
[13] `http://www.ivoa.net/astronomers/applications.html`
[14] GNU General Public License - see `http://www.gnu.org/licenses/gpl-3.0.en.html`
[15] `http://simbad.u-strasbg.fr/simbad/`
[16] `http://vizier.u-strasbg.fr/viz-bin/VizieR`

de Strasbourg[17], written in Java and available under GNU/GPL license. It has a Lite version that can run directly from web browser. Same as TOPCAT or SPLAT-VO, Aladin supports SAMP protocol which makes it a great cooperative tool.

[17] `http://aladin.u-strasbg.fr/aladin.gml`

3 Basic Terms and Concepts

This section covers basic terms and basic concepts from astronomy, astroinformatics and software engineering necessary for understanding of following sections.

3.1 Astronomy and Astroinformatics

Full description of basic astronomical and astroinformatical terms would of course be beyond the scope of this thesis. This section therefore covers a minimum in a form of entities and data formats required for understanding to following sections, where modifications of SPLAT-VO tool will be described.

3.1.1 Spectrum

Probably everyone remembers the old primary school physics experiment with sunlight and prism, where sunlight coming through a prism is dispersed to individual colors, or knows a rainbow, where the sunlight is dispersed in the same way on rain drops acting like a prism. This is possible due to a fact that the sunlight — as a form of electromagnetic radiation that consists of electromagnetic waves visible by naked human eye — is composed from several electromagnetic waves with different frequencies (a.k.a. wavelengths) [13]. And this composition is visible as a result of prism dispersion and known as a sunlight spectrum.

In more general way, spectrum is a graph of relative intensity vs. frequency of any wave phenomena [13]. The spectrum is a "fingerprint" of its source. In case of astronomy, lets imagine a hot star (composed from superhot plasma) or diffuse gas that emits light on different wavelengths. What wavelength it is, depends on a original chemical element emitting the light. These elements then can be identified in the spectrum using their wavelengths. The comparison of photographic and intensity plot spectra of a star is shown in Fig. 3. We recognize 3 basic types of spectra [14]:

Continuum spectrum A dense hot object (such as the core of a star) acts like a black body radiator. If we were able to view the light from this source directly without any intervening matter then the resultant spectrum would appear to be a continuum as shown in Fig. 4, left [14].

Absorption spectrum Most stars are surrounded by outer layers of gas that are less dense than the core. Photons of specific frequency emitted by a star can be absorbed by electrons in the diffuse outer layer of gas, causing the electron to change energy levels and emitting a new photon of specific frequency. The direction of this re-emission however is random. By this, the intensity of light at the wavelength of that photon will be less in the direction of an observer and the spectrum will show dark absorption lines or a decrease in intensity as shown on Fig. 4. Stellar spectra typically look like this [14].

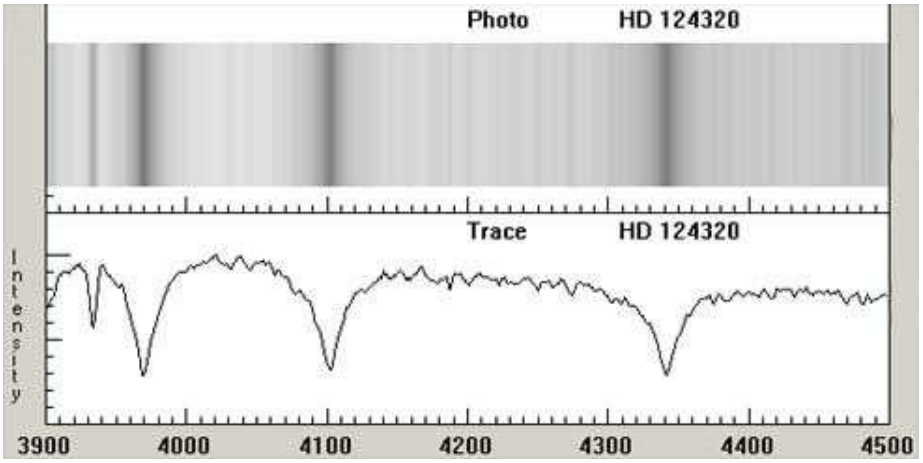


Figure 3: Comparison of photographic and intensity plot spectra for a star. Source: [14].

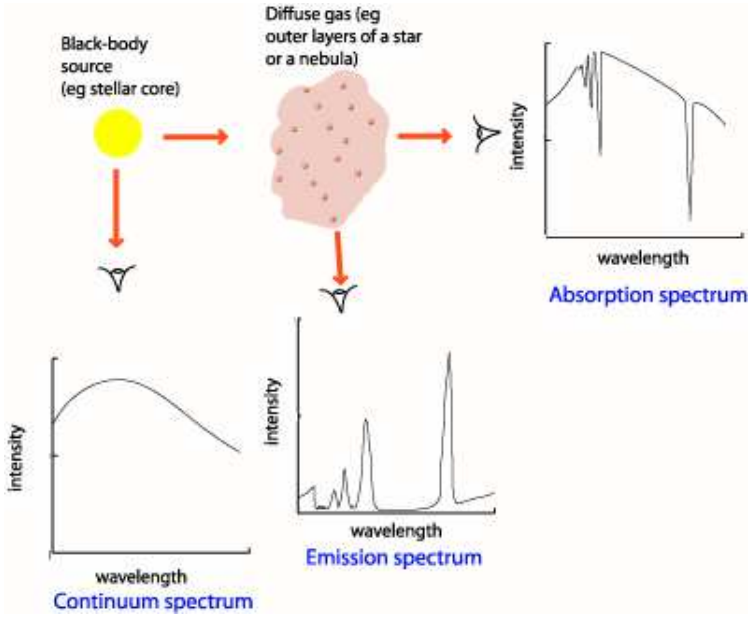


Figure 4: Continuous, emission and absorption spectra and their sources. Source: [14].

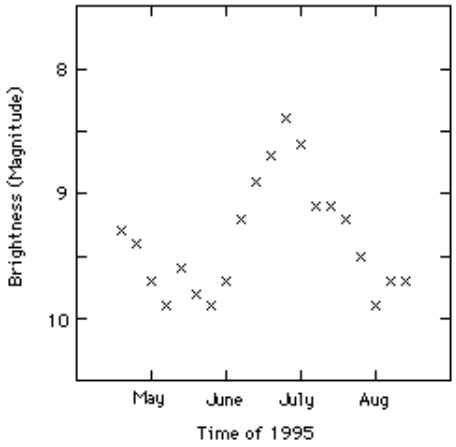


Figure 5: A simple light curve. Source: [15].

Emission spectrum It occurs if an observer is not looking directly at a hot black body source but instead at a diffuse cloud of gas that is not a black body. If this cloud can be excited by a nearby source of energy such as hot, young stars or an active galactic nucleus then the electrons in atoms of the gas cloud can get excited. When they de-excite they emit photons of specific frequency and wavelength. As these photons can be re-emitted in any direction an external observer will detect light at these wavelengths. The spectrum formed is an emission or bright line spectrum, as shown in Fig. 4 [14].

3.1.2 Time Series / Light Curves

Astronomical time series (a.k.a. light curves) are graphs that show the brightness of an object over a period of time [15] as shown in Fig. 5.

Light curves are useful in case of objects which change brightness in time, such as novae, supernovae and variable stars [15]. Since we know generally what light curves look like for a set of objects [15], we can compare it and understand an observed phenomena.

3.1.3 Data Format FITS

Stands for *Flexible Image Transport System* and is a specialized data format developed by NASA[18] and the International Astronomical Union[19] for the transport, analysis, and archival storage of scientific data sets [16]:

- Multi-dimensional arrays: 1D spectra, 2D images, 3D+ data cubes [16]
- Tables containing rows and columns of information [16]
- Header keywords provide descriptive information about the data [16]

[18]National Aeronautics and Space Administration - see `https://www.nasa.gov/`
[19]`http://www.iau.org`

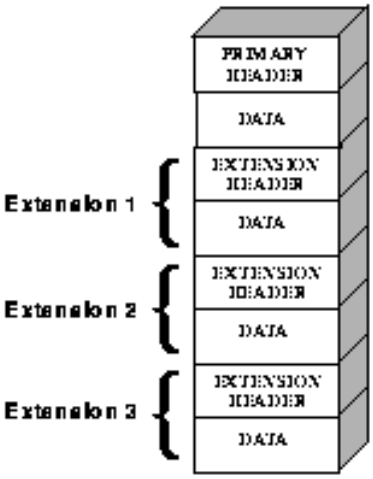


Figure 6: The structure of FITS file. Source: [17].

FITS has been exclusively used for almost all astronomical data storage for more than 30 years. It allows to store various data types inside a single file. For example, it can contain an image of a star with its spectra in different wave bands. Each item is then stored in a form of so-called extension of the respective type.

As shown on Fig. 6, FITS consists of items called *Header Data Unit (HDU)*. There is a mandatory *primary HDU* that contains some mandatory metadata in a form of key/value pair (keys are world-wide standardized) in ASCII header. There may also be other *HDU*s called *extensions*. Each *extension* has its own set of metadata in header an we recognize following standard types of extensions [18]:

IMAGE This extension type provides a means of storing a multidimensional array similar to that of the FITS primary header and data unit [18].

TABLE This ASCII table extension type contains rows and columns of data entries expressed as ASCII characters [18].

BINTABLE This binary table extension type provides a more flexible and efficient means of storing data structures than is provided by the **TABLE** extension type. The table rows may contain a mixture of numerical, logical and character data entries. In addition, each entry is allowed to be a single dimensioned array. Numeric data are kept in binary formats [18].

3.1.4 Data Format VOTable

The VOTable format is an XML standard for the interchange of data represented as a set of tables [19] endorsed by IVOA. It is an unordered set of rows, each of a uniform structure, as specified in the table description (the table metadata). Each row in a table is a sequence

of table cells, and each of these contains either a primitive data type, or an array of such primitives [19]. The data in VOTable may be expressed in one of the following formats:

TABLEDATA Pure XML format that can be used for small tables [19].

FITS Encapsulated or re-encoded FITS (see 3.1.3) [19].

BINARY and BINARY2 Ease of programming and support for streaming [19].

In order to describe the semantics of the contained data, VOTable uses UCD[20], Utypes[21], Units[22] and STC[23] [19].

An example of VOTable follows:

```
<?xml version="1.0"?>
<VOTABLE version="1.3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.ivoa.net/xml/VOTable/v1.3"
 xmlns:stc="http://www.ivoa.net/xml/STC/v1.30" >
  <RESOURCE name="myFavouriteGalaxies">
    <TABLE name="results">
      <DESCRIPTION>Velocities and Distance estimations</DESCRIPTION>
      <GROUP utype="stc:CatalogEntryLocation">
        <PARAM name="href" datatype="char" arraysize="*"
               utype="stc:AstroCoordSystem.href" value="ivo://STClib/CoordSys#
                  UTC-ICRS-TOPO"/>
        <PARAM name="URI" datatype="char" arraysize="*"
               utype="stc:DataModel.URI" value="http://www.ivoa.net/xml/STC/stc-
                  v1.30.xsd"/>
        <FIELDref utype="stc:AstroCoords.Position2D.Value2.C1" ref="col1"/>
        <FIELDref utype="stc:AstroCoords.Position2D.Value2.C2" ref="col2"/>
      </GROUP>
      <PARAM name="Telescope" datatype="float" ucd="phys.size;instr.tel"
             unit="m" value="3.6"/>
      <FIELD name="RA" ID="col1" ucd="pos.eq.ra;meta.main"
             datatype="float" width="6" precision="2" unit="deg"/>
      <FIELD name="Dec" ID="col2" ucd="pos.eq.dec;meta.main"
             datatype="float" width="6" precision="2" unit="deg"/>
      <FIELD name="Name" ID="col3" ucd="meta.id;meta.main"
             datatype="char" arraysize="8*"/>
```

[20]The UCD1+ controlled vocabulary - see http://www.ivoa.net/documents/latest/UCDlist.html
[21]See http://wiki.ivoa.net/bin/view/IVOA/Utypes
[22]See http://www.ivoa.net/documents/VOUnits/
[23]Space-Time Coordinate Metadata - see Space-TimeCoordinateMetadata

```
      <FIELD name="RVel" ID="col4" ucd="spect.dopplerVeloc" datatype="int"
             width="5" unit="km/s"/>
      <FIELD name="e_RVel" ID="col5" ucd="stat.error;spect.dopplerVeloc"
             datatype="int" width="3" unit="km/s"/>
      <FIELD name="R" ID="col6" ucd="pos.distance;pos.heliocentric"
             datatype="float" width="4" precision="1" unit="Mpc">
        <DESCRIPTION>Distance of Galaxy, assuming H=75km/s/Mpc</DESCRIPTION>
      </FIELD>
      <DATA>
        <TABLEDATA>
        <TR>
          <TD>010.68</TD><TD>+41.27</TD><TD>N 224</TD><TD>-297</TD><TD>5</TD><TD
              >0.7</TD>
        </TR>
        <TR>
          <TD>287.43</TD><TD>-63.85</TD><TD>N 6744</TD><TD>839</TD><TD>6</TD><TD
              >10.4</TD>
        </TR>
        <TR>
          <TD>023.48</TD><TD>+30.66</TD><TD>N 598</TD><TD>-182</TD><TD>3</TD><TD
              >0.7</TD>
        </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

Listing 1: VOTable example. Source: [19].

As it can be seen from VOTable above, it consists of *Metadata* (*Parameters* + *Infos* + *Descriptions* + *Links* + *Fields* + *Groups*) and *Table data* (stream of *Rows* that contains *Cells* of *Primitives* - integers, chars, etc.) arranged as a set of *Tables* (list of *Fields* + *TableData*) [19].

To better understand the example, there is one *Table* called `results` under a *Resource* `myFavouriteGalaxies`. This table has some *Parameters* identifying the telescope used and coordinate system (based on STC). This *Table* has six columns with meaning described in *Fields*, each with reference identificator `ID`. The data itself are contained within *Table data* section. There are three *Rows* with *Cells* ordered in the same order, as *Fields* in *Metadata*, which gives them a meaning. For example: the second galaxy in the table has declination -63.85 degrees.

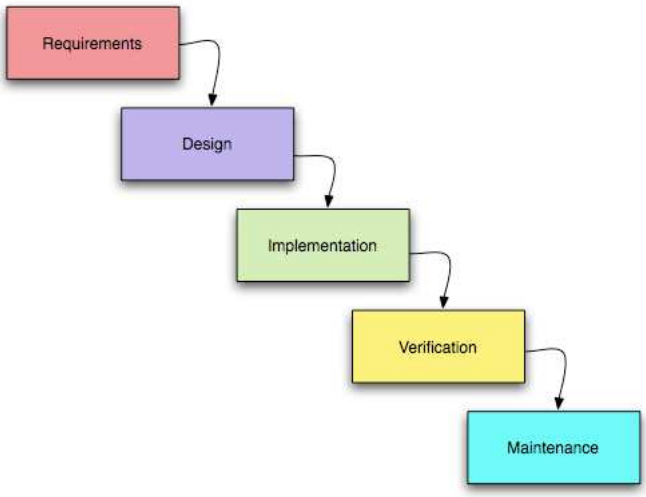


Figure 7: A waterfall model of software developmnet. Source: [21].

Figure 8: An iterative model of software development. Source: [22].

3.2 Software Engineering

As well as in case of the previous section, full description of all major software engineering terms and concepts would be beyond the scope of this thesis. Therefore, only those terms and concepts neccessary for understanding the following sections are described in this section.

3.2.1 Basic Methodics and Development Processes

During a history of software development, several processes evolved. From a most basic *waterfall model* (see Fig. 7), that suffered from several drawbacks (long period from requirements collection to a final product, a significant risk of delivering a system with misunderstandings, etc.), the software process moved to an *iterative way of software development* (see Fig. 8) [20], that attempts to solve major drawbacks of *waterfall model* by segmentation of the whole software process to several "small waterfalls" called *iterations*.

In each iteration then, we recognize the following actions that are realized in greater or lesser degree (based on the development phase) [20]:

- Business modeling
- Collecting requirements
- Analysis and prioritization of requirements
- Development
- Testing
- Deployment
- Feedback and Change management
- Project management

There are several other methodologies in software development, that builds on the waterfall and iterative software development (RUP, SCRUM, etc.). Their description is behind the scope of this thesis, so for more information, please refer to [20] or [22].

3.2.2 Continuous Integration and Jenkins CI

Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible [23].

This leads to significant elimination of many integration problem, such as breaking the existing code, wrong use of API among development teams (or inside a development team).

CI can be used in cooperation with code versioning systems (see 3.2.3), system scripting, build tools (such as Ant, Maven, etc.), specific software development tools for checking code coverage (by tests), copy/paste detection, automated bug finders etc. It can therefore be used as a tool for a complete *Continuous Delivery* (delivering the system to production use at every moment).

Jenkins CI Is one of the best tools for Continuous Integration with large user base. It is written in Java with web user interface (see Fig. 9), so it can be deployed to production on every *Java Application Server* or *Servlet Container*. Jenkins CI[24] can be used for automated builds, testing, deploying and (also in cooperation with many plugins) for basically every automated action one can think. Individual automated actions (called *build*) can be triggered by multiple events, such as period, commit to repository, manual, by other *upstream build* etc.

[24] Jenkins CI homepage: `https://jenkins-ci.org/`

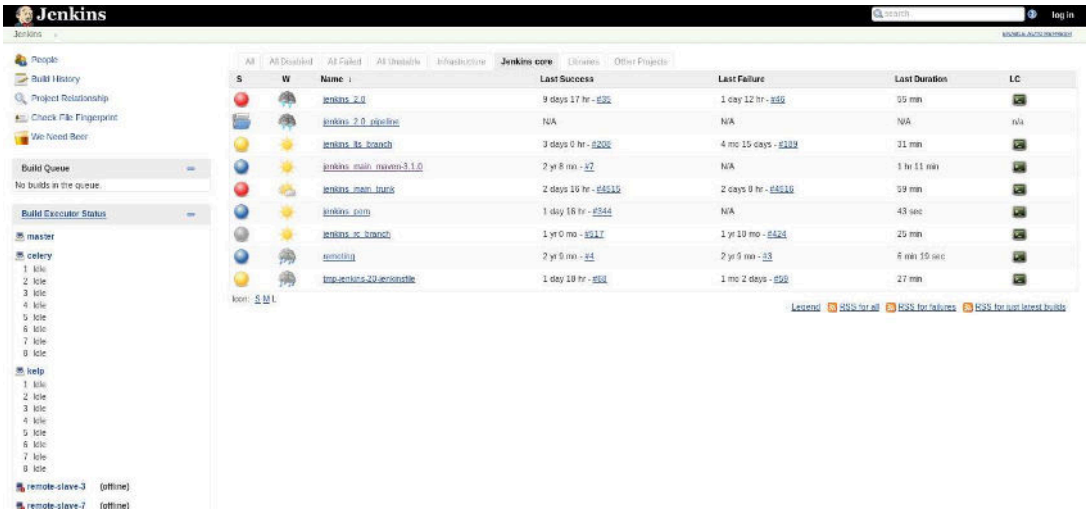


Figure 9: Jenkins CI - example of UI.

3.2.3 Code Versioning Systems

Code Versioning Systems (a.k.a. CVS) are important in software development for several reasons:

- It allows to track history of the project on file level, including traversing it (reverting changes, etc.).
- It makes team cooperation much faster and easier. CVS automatically merges files modified by several members of the development team.
- It keeps the source code at one place.

There are several main CVS:

CVS The oldest (1986 [24]) and most primitive CVS still in use (especially for small projects.).

SVN A.k.a. *Subversion* is probably the version control system with the widest adoption [24]. Many large and well-known projects still uses it. It also has a large support in IDEs and many user clients.

Git Initially developed by Linux kernel author Linus Torvalds, *Git* is currently one of the best, fastest and most used CVS, especially in large projects. It is considered to be a *Swiss Army Knife* of the code versioning - it is capable of almost every imaginable use case, altough the way of achieving it might not be as user-friendly, as in case of other CVS. In contrast with other CVS, Git uses a distributed version control system, so there is not one centralized code base to pull the code from [24].

Mercurial Designed for larger projects, Mercurial is extremely fast and capable of all usual use cases with ease, which is something, that makes it different from Git (ease of use vs. count of features).

Bazaar Calls itself "Version control for human beings." [24] One of the main features of Bazaar is the fine-grained control over the setup [24]. Otherwise it is very similar to other modern CVS, like Git or Mercurial.

Microsoft Visual SourceSafe Microsoft's attempt to create a CVS to use in cooperation with its Visual Studio. After a criticism, it was discontinued and is no longer distributed with current versions.

3.2.4 Issue Tracking

Issue tracking system allows an organization to register, watch, prioritize and generally organize every issue related to some project. It allows to track the history of every issue, modify its states, add comments or attachments etc.

Modern issue tracking systems also supports project management use cases, such as making time estimates for each issue, creating Gantt's charts, network chart and critical path, etc.

Among other requirements, we can present the capability to integrate the issue tracking system with company's LDAP system, ability to integrate with company's database infrastructure and runtime environment in general, web user interface etc. Among the mostly used issue trackers, we recognize:

Projectlibre Local machine Java application supporting issue tracking and several charts.

Redmine One of the most used issue trackers, written in Ruby on Rails, highly configurable with many plugins and charts support.

Trac and Apache Bloodhound Another mostly used issue tracker and its Apache spin-off. *Trac* is written in Python and stands on using many plugins in order to meet general requirements on issue tracking system.

Phabricator Is written in PHP and developed by Facebook, currently also in use by many significant projects, such as Mediawiki, Blender, Dropbox, etc.

Atlassian JIRA Mostly commercial, but very favorite issue tracking system with its own ecosystem. Many signifact projects uses it (e.g. Spring, Hibernate, etc.).

Github Issues Favorite issue tracker of Github[25].

[25]Github homepage: `https://github.com/`

3.2.5 Project Documentation

Project documentation is a desirable part of every software project. It describes the software architecture, its *API*, used technologies and software libraries, build process etc.

We can look at the project documentation from the following perspectives:

Software API Usually written by programmers themselves as a part of source code, that is during compilation transformed to a HTML, PDF or other user-friendly form. Typical example of this is *Javadoc*.

Used technologies, architecture and software libraries Usually listed in master project documentation - a *Wiki page* or a set of documents. Maintained by programmers and software architects.

Project development organization Usually listed in master project documentation and maintained by project manager and/or team leader.

Project build information Usually listed in master project documentation and maintained by programmers and software architects.

Other information - project history etc. Usually listed in master project documentation.

Based on these requirements, we can divide project documentation to the following:

Source code documentation *Javadoc* in Java, XML documentation in .NET etc.

Master project documentation Can be in a form of a single document (appropriate for small projects), multiple documents (typical for corporations), Wiki-like systems (used in general) or their combination. It should contain an overview of software project architecture (including UML diagrams), used technologies and 3rd party software libraries, development team organization and processes, information about obtaining source code and building the project, project history etc. In modern age, it should also support collaborative work for multiple simultaneous authors, revision history, adding comments and discussion etc.

3.2.6 Virtualization, Containers and Docker

The power of modern computers allows to run multiple services on a single host machine. At these days, it is only a question of security, required/reserved computational power and organization structure requirements what of the following approaches will be used:

No virtualization This is the most basic approach for running multiple services on a single machine: directly on the host machine. It is the most effective in sense of speed and system resources, but also most vulnerable to attacks (if the attacker takes control over a service, it also gains its privileges inside the entire host system) and difficult for system administration (security, library conflicts etc.).

Containers Uses a single kernel and resources from the host system. Virtual machines running inside a container have a direct access to host machine hardware via its shared kernel. This leads to very small overhead and high performance with a great level of isolation. Typical example of container is Linux `chroot`, FreeBSD Jail or *Docker*.

Paravirtualization (Hypervisors) Uses a modified host kernel that contains a *hypervisor*, that allows guest machines to access host hardware (via hypervisor calls). The guest knows that it is running inside a virtual machine, yet the level of isolation is much higher than in case of containers (which is on the other hand redeemed by higher usage of host system resources). A typical example is *XEN*.

Full virtualization Simply emulates all hardware except CPU. The guest has no idea that it is running inside a virtual machine, so the level of isolation on the same CPU architecture is the highest possible (as well as host resources consumption). A typical examples are *VirtualBox*, *VMWare* or even *XEN* configured in full virtualization mode.

Emulation Simply emulates all hardware including CPU. This allows to run for example old *Amiga* applications on a modern *x86-64* computer, so the level of isolation (and host system resources consumption) is the highest possible. This leads to poor performance, yet it has its use cases. A typical example is *QEMU*.

3.2.7 Datacubes and Data Warehouses

To understand this section, we need to describe a difference between OLTP and and OLAP:

OLTP *On-Line Transaction Processing* - transactional systems for real-time data collecting and storage (ERP, accounting software etc.). Thousands of transactions are being made within a minute. It is considered to be a primary data source. These systems contain large amounts of data and their analysis causes a significant delays in primary usage [25].

OLAP *On-Line Analytical Processing* is based on a multidimensional database concept. Its basis is multidimensional table (*datacube*), that allows to analyze the data from many points of view (*dimensions*) directly by the user (so they do not need to work just with preprepared views). They are designed for data analysis of periodically (not real-time) updated data obtained from OLTP [25].

Data Cube As tables in relational databases, data cubes are used for storage in OLAP databases [25]. Data cube is a multidimensional extension of database table (2-dimensional, 3-dimensional and n-dimensional in general) [25]. An example of data cube is shown on Fig. 10. This data cube has 3 dimensions (*Part*, *Store Location* and *Customer*).

Dimension Elements can be a name of product, its price, size, etc. In case of Fig. 10 and *Part* dimension, it is *P1 - P5*.

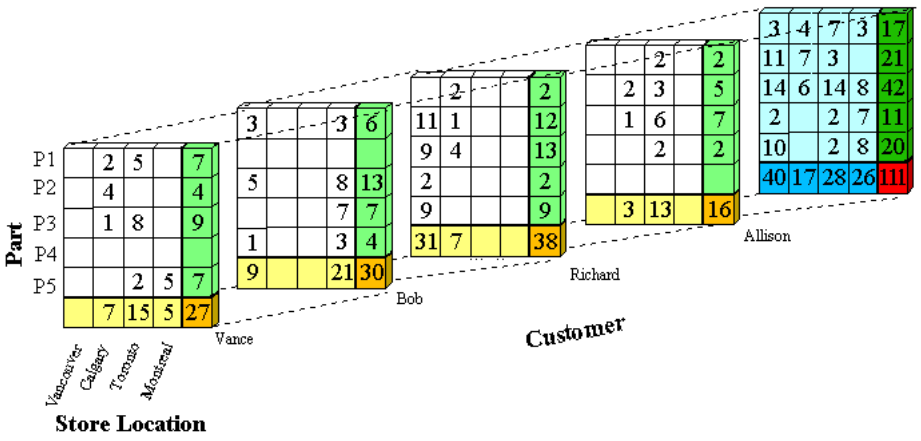


Figure 10: An example of data cube (with totals). Source: [26].

Hierarchy of Elements can be created on a dimension level for groups of relating elements. For Fig. 10, let's imagine a geographical hierarchy of *Parts* based on where they have been produced:

- *Parts*
  - *EU*
    - *P1*
    - *P2*
  - *USA*
    - *P3*
    - *P4*
    - *P5*

Basic operations with Data Cubes There are several operations that can be performed on a data cube. Those most basic are:

Slicing selects one particular dimension from a given cube and provides a new sub-cube [27].

Dicing selects two or more dimensions from a given cube and provides a new sub-cube [27].

Roll-up performs aggregation on a data cube in a way of climbing up a concept hierarchy for a dimension or dimension reduction [27].

Drill-up is the reverse operation of roll-up. It is performed either by stepping down a concept hierarchy for a dimension or introducing a new dimension [27].

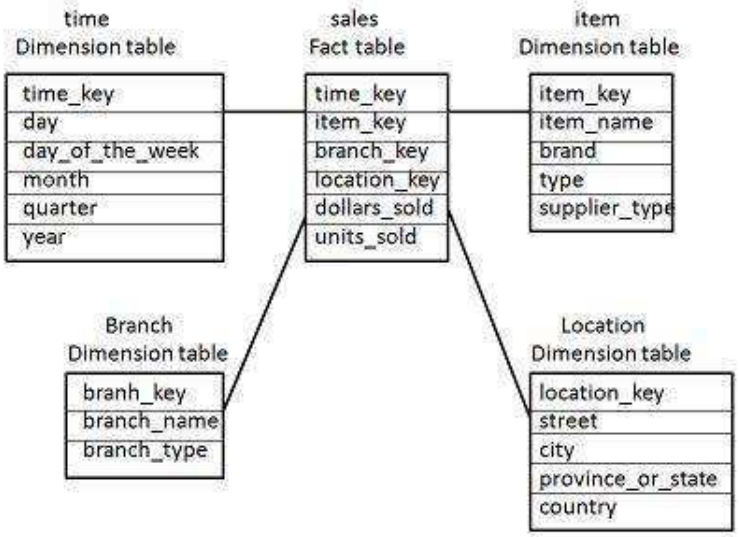


Figure 11: An example of star data warehouse schema. Source: [29].

Pivoting is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data [27].

A more complete description of data cube operations including simple, easy-to-understand illustrations can be found in [27].

Data Warehouses A data warehouse is a relational database that is designed for query and analysis rather than for transaction processing [28]. It usually contains historical data derived from transaction data and separates analysis workload from transaction workload [28].

In addition to a relational database, a data warehouse environment includes an extraction, transportation, transformation, and loading (*ETL*) solution, an online analytical processing (OLAP) engine, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users [28].

Star and Snowflake Schemas In data warehousing, we recognize two basic database schemas (similar to traditional database schemas). In center of both of them, we can find a *fact table* - table of business facts, measurements etc with foreign keys to surrounding *dimension tables*, that contains descriptive metadata used for filtering etc.

Star schema Each dimension in a star schema is represented with only one-dimension table, that contains the set of attributes (see an example on Fig. 11) [29].

Snowflake schema Some dimension tables in the Snowflake schema are normalized, which causes the data to be splitted up into additional tables (see an example on Fig. 12) [29].

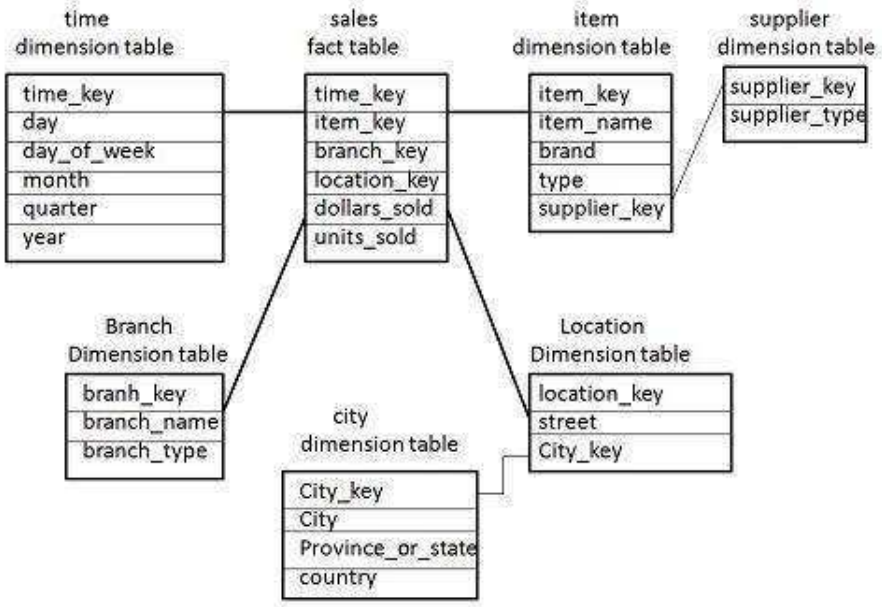


Figure 12: An example of snow flake data warehouse schema. Source: [29].

In my work, the international project SPLAT-VO (see section 4) is ready for star schema mainly. There is also work on enabling basic data cubes operations within it (see section 6.1). For a more complete description and other data warehouse schemas, please refer to [29].

4 SPLAT-VO

SPLAT-VO and its modifications is the main subject of this thesis. Its name (SPLAT) is a shortcut for SPectraL Analysis Tool [30] and it is a software application for displaying, modifying and analysing astronomical spectra [31] (see Fig. 13 and Fig. 14 respectively, for example).

4.1 History

SPLAT was originally developed in 2003 as a part of Starlink (and its STARJAVA package) project [31]. During its development, the original SPLAT was extended to include facilities that allowed an interoperability with the Virtual Observatory (see Sec. 2.1) [32], which resulted in a *VO* suffix in its name in 2005.

SPLAT was also a leader project for moving future developments into the Java language, with the obvious benefits of improved portability, modern language capabilities (OOD, OOP) and core-level support for features like UIs and Internet protocols and services. This work eventually led to the formation of what became the StarJava project, containing also the well-known table processor TOPCAT [4] (see section 2.2.2).

The original Starlink project was eventually shut down in 2005, yet its development continued until now under *Joint Astronomy Centre*[26] (that released some parts under GNU/GPL licence) and since 2015 by *East Asian Observatory*[27] [33].

4.2 Team and Development Organization

SPLAT-VO development was took over in 2012 by GAVO[28] (German Astrophysical Virtual Observatory), Astronomical Institute of the Czech Academy of Sciences[29] and VŠB - Technical University of Ostrava, Faculty of Electrical Engineering and Computer Science[30]. Its current development is focused on new and experimental IVOA standards implementation, handling large data sets and other data types and data models as well as improving user interface.

4.2.1 Current Development Team

As mentioned before, SPLAT-VO is currently developed by several organizations:

Astronomical Institute of the Czech Academy of Sciences Fričova 298, 251 65 Ondřejov, Czech Republic.

Web: `http://www.asu.cas.cz/`

[26] Joint Astronomy Centre homepage: `http://www.jach.hawaii.edu/`
[27] East Asian Observatory homepage: `http://www.eaobservatory.org/`
[28] German Astrophysical Virtual Observatory homepage: `www.g-vo.org`
[29] `http://www.asu.cas.cz/`
[30] `http://www.fei.vsb.cz/`

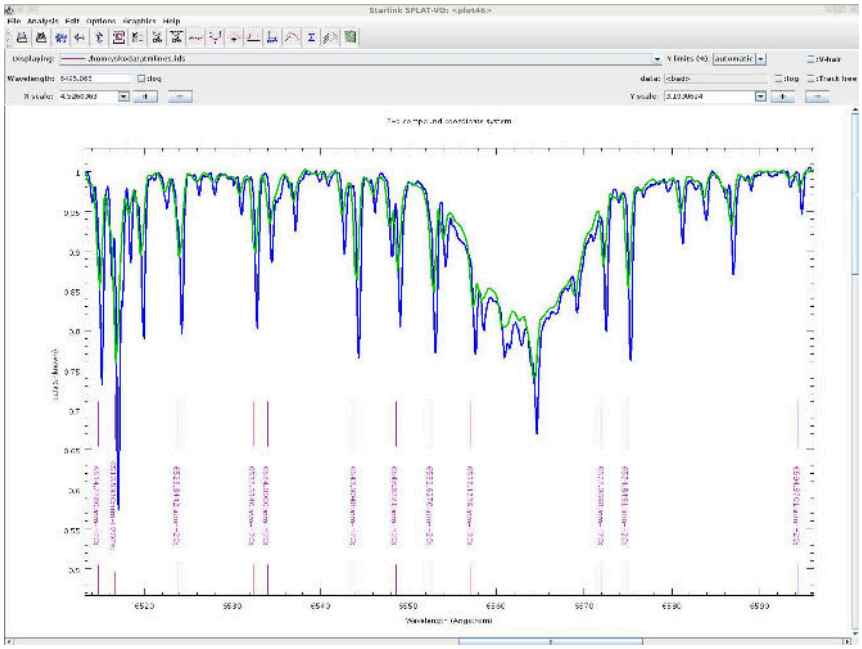


Figure 13: SPLAT-VO at work. Source: [30].

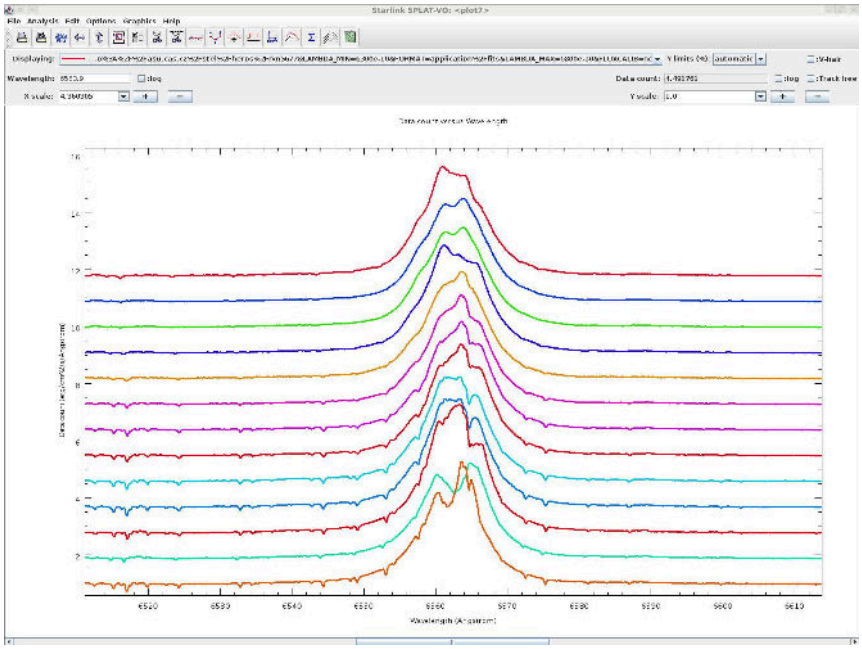


Figure 14: Many spectra in SPLAT-VO. Source: [30].

- **Petr Škoda** Scientific advisor and coordinator, IVOA Interoperability Meetings presenter.
  E-mail: skoda@sunstel.asu.cas.cz.

Heidelberg University Astronomisches Rechen-Institut (ARI) Zentrum für Astronomie, Universität Heidelberg (ZAH) Mönchhofstr. 12 - 14, 69120 Heidelberg, Germany.
Web: http://www.uni-heidelberg.de/

- **Margarida Castro Neves** Maintainer and Java developer for SSA, ObsCore and DataLink protocol-related features.
  E-mail: mcneves@ari.uni-heidelberg.de.
- **Markus Demleitner** Server side implemetation of VO protocols and data models, GAVO coordinator.
  E-mail: msdemlei@ari.uni-heidelberg.de.

VŠB - Technical University of Ostrava Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VŠB - Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic.
Web: http://www.cs.vsb.cz/

- **David Andrešič** Java developer for spectra visualization, data export, time series implementation, refactoring suggestions and general tweaking.
  E-mail: david.andresic.st@vsb.cz.

4.2.2 Previous Members of Development Team

The main developers of the original SPLAT-VO are:

- **Peter W. Draper** SPLAT architecture and main developer.
  E-mail: p.w.draper@durham.ac.uk
- **Mark Taylor** Current developer of TOPCAT (see 2.2.2).
  E-mail: m.b.taylor@bris.ac.uk

4.2.3 Current development process

Since the SPLAT-VO development is literally a world-wide activity, it does not fit exactly into any usual software process model such as those described in section 3.2.1. For the last several years, it is anyway close to agile methodics, since it has its *backlog* that contains (still updated) requirements coming mainly from:

- conferences;

- practical experience at Stellar Dept. of Astronomical Institute of Czech Academy of Sciences;
- IVOA Interoperability Meetings.

Based on current needs of astroinformatics community, the coordinators set the priorities to the requirements and in cooperation with developers perform an implementation analysis. Based on the estimated required time, the requirement is scheduled for implementation. It is usual to make a proof-of-concept of the requirement implementation first, so it may be presented on IVOA Interoperability Meetings and conferences to receive an expert feedback. This makes the scope of every version totally open, so it is mainly maintainer's choice when the final release will be made (usually after the preplanned features are tested enough). Every implemented requirement is continuously tested by our coordinators. Before the release, the maintainer updates the changelog and user documentation and prepares a final build with installer that is linked from SPLAT-VO homepage. In certain sense, SPLAT-VO development is therefore iterative - each requirement (and its implementaton) is re-evaluated for several times during its development until all sides are satisfied enough to close the scope of the version and make a release.

The bottleneck of this process is that the "backlog", communication and the entire interoperability are subject of e-mails. This makes tracking features in *backlog*, its implementation status, discussion and time estimates as well as scope of versions, very hard to control. In order to solve these issues, an issue tracking system is being introduced (see section 7.2).

Another problem is unability of coordinators to build a snapshot version (from current state of the repository) on their own and check the state of repository by means of continuous integration (see 3.2.2). In order to solve this problem, the "Dockerized" Jenkins CI system is being introduced and deployed in Stellar Department of the AI CAS (see sections 3.2.2, 3.2.6 and 7.3).

4.3 User Interface

SPLAT-VO is a multi-window application. This section covers the most important of them and their basic concepts. A complete description of SPLAT-VO user interface is a subject of official documentation [31].

4.3.1 Main Window

This is a starting and central window of SPLAT-VO. As shown on Fig. 15, it contains a *Global list of spectra* currently loaded to a running instance. Right side is a place of visualization settings for the spectrum or spectra selected in *Global list of spectra*. The top of the *Main*

[31] SPLAT-VO original homepage: `http://star-www.dur.ac.uk/~pdraper/splat/splat-vo/splat-vo.html`, GAVO SPLAT homepage: `http://www.g-vo.org/pmwiki/About/SPLAT`

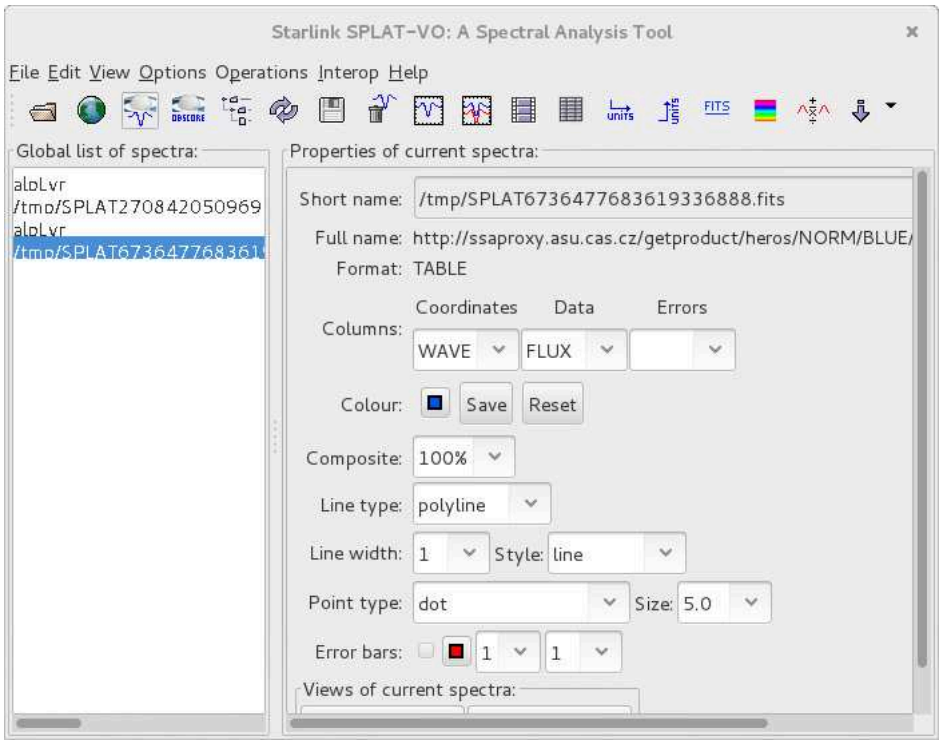


Figure 15: Main window.

Window contains the application menu - an entry point to where the spectra can be loaded and processed.

Some of the most used and most interesting windows achievable in SPLAT-VO from the *Main Window* are:

- *Query VO for spectra* (see section 4.3.2)
- *ObsCore Browser* (see section 4.3.3)
- *Plot Window* (see section 4.3.4)
- *View/modify a spectrum* (see section 4.3.5)

There are many other windows in SPLAT-VO, but those listed above should be sufficient for understanding of the aim of this thesis.

4.3.2 Query VO for spectra

This window (shown on Fig. 16) is an entry point to VO, where the user can search for spectra (in IVOA terms: perform a data discovery) and download them (in IVOA terms: perform a data access) through the SSAP protocol (see section 2.1.2). A complex interface consists of:

Service selection options on the left side where the user can select:

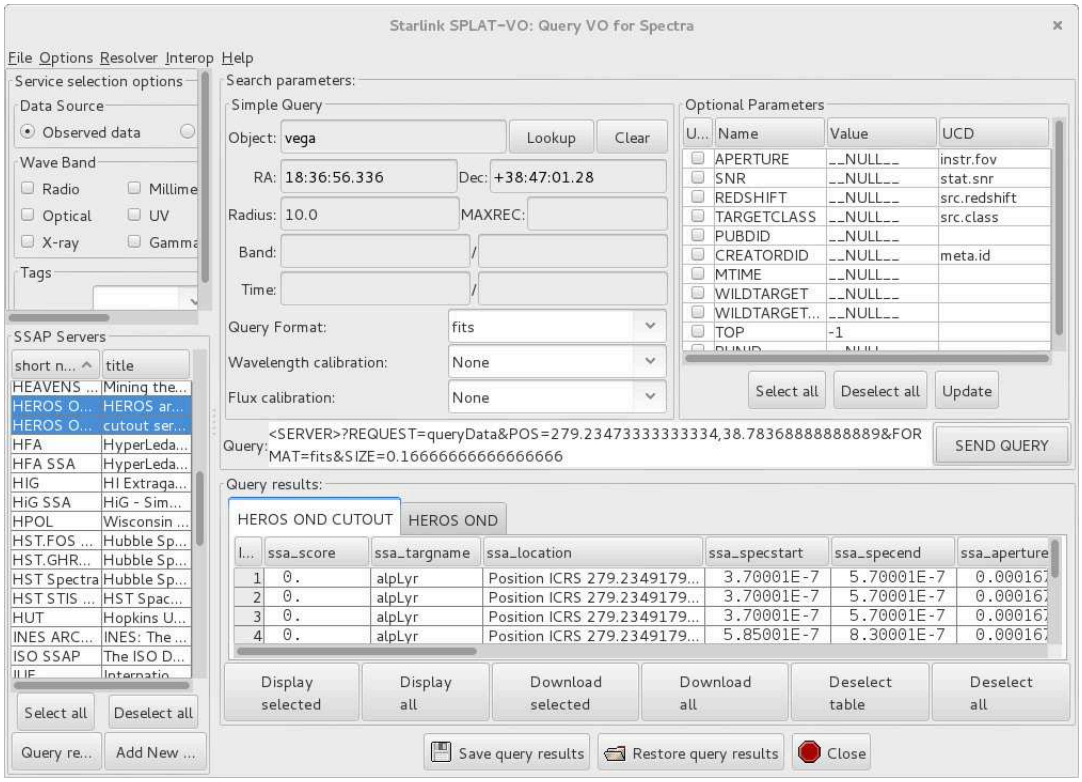


Figure 16: Query VO for spectra.

- *Data source* - observed or theoretical data
- *Wave band* - radio, optical, infrared, X-ray, etc.
- *SSAP servers* - concrete servers to be queried

Search and optional parameters in the middle and right, where the user can input parameters for data discovery (object name, right ascension, declination, radius, band, data format etc.)

Query results and data access options at the bottom of the window, where the user can find a tab for each SSAP server that found at least one spectrum

4.3.3 ObsCore Browser

ObsCore Browser is an experimental implementation of the ObsCore protocol (see section 2.1.2) for general observation data discovery and access (see Fig. 17). Similar to *Query VO for spectra* window, the user can select ObsCore services and perform a:

Cone search via *Cone search* protocol (see section 2.1.2) based on object name, right ascension, declination, radius, band etc.

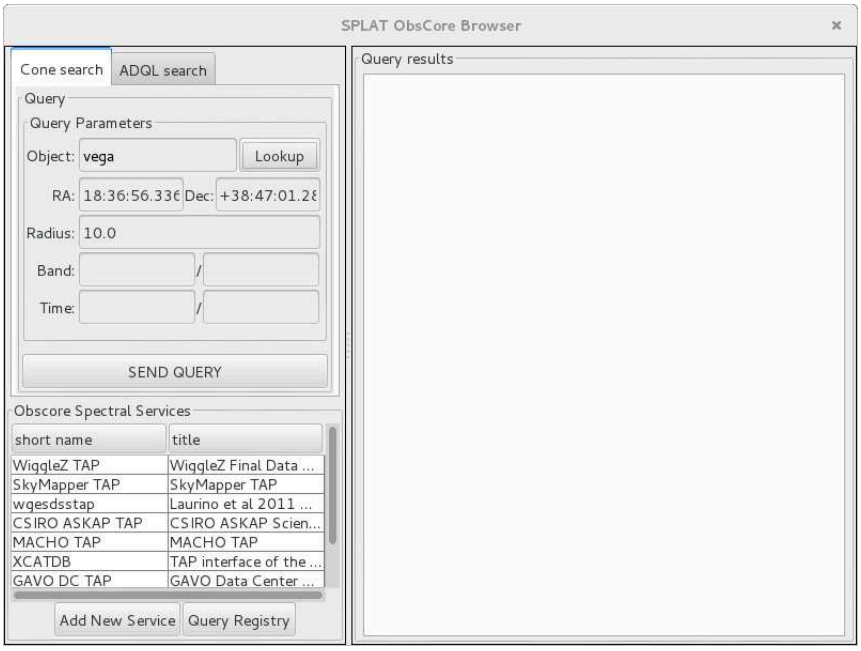


Figure 17: ObsCore browser. Source: [34].

ADQL search via *ADQL* protocol (see section 2.1.2) where more fine-grained parameters can be set.

Results of the query are then shown to the user in the *Query results* panel on the right side of the window.

4.3.4 Plot Window

Plot window is probably one of the most important SPLAT-VO windows at all. As the name implies, this window serves for plotting spectra (see Fig. 14 and 18) based on parameters set in *Main Window* (which can be overriden or precised within the *Plot window* itself).

The spectra plot itself covers most of the window, but at the top, the user can find an arsenal of spectral analysis tools. Just for example:

- Cutting regions from spectrum
- Fitting
- Comparing
- Statistics
- Multiple matching

And many more (a concrete description of the individual algorithms is beyond the scope of this thesis). Within these menus, the user can also highly customize the plot, invert axes, set them logarithmic and much more.

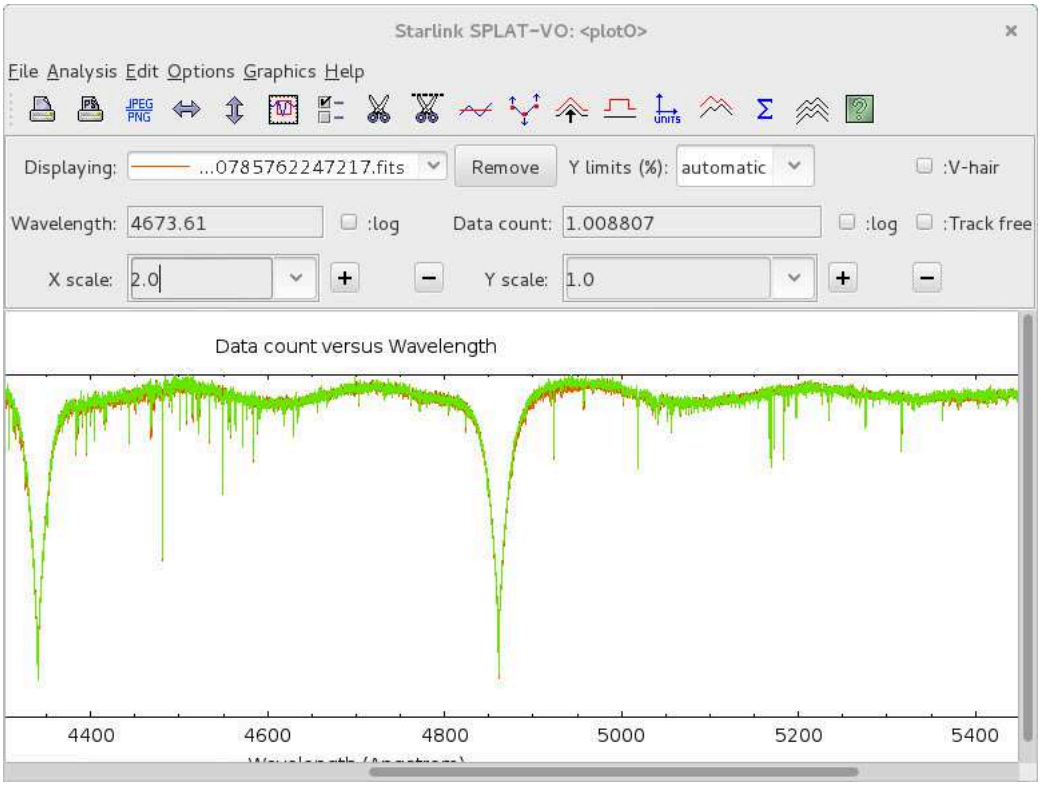


Figure 18: Plot window.

4.3.5 View/modify a spectrum

A simple window that allows the user to view spectral data itself in a form of a table (see Fig. 19). This window is (compared to others) very simple and almost featureless, yet it allows a manual manipulation with spectral data, both axes transformation, adding columns by computation, and newly data export to CSV format (see section 5.5).

4.4 Most Typical Use Cases

As the user interface implies, SPLAT-VO is capable of opening the spectra (and also light curves, if they are represented as a spectrum with time axis instead of wavelength axis) contained in multiple formats [34]. These data (files) can be loaded:

- locally from a file (many formats are supported) [34];
- via SSAP protocol[32] (Virtual Observatory) [34];
- via SAMP protocol[33] (Virtual Observatory) [34];
- as a in-memory result of some operations performed on other spectra [34].

[32] See 2.1.2
[33] See 2.1.2

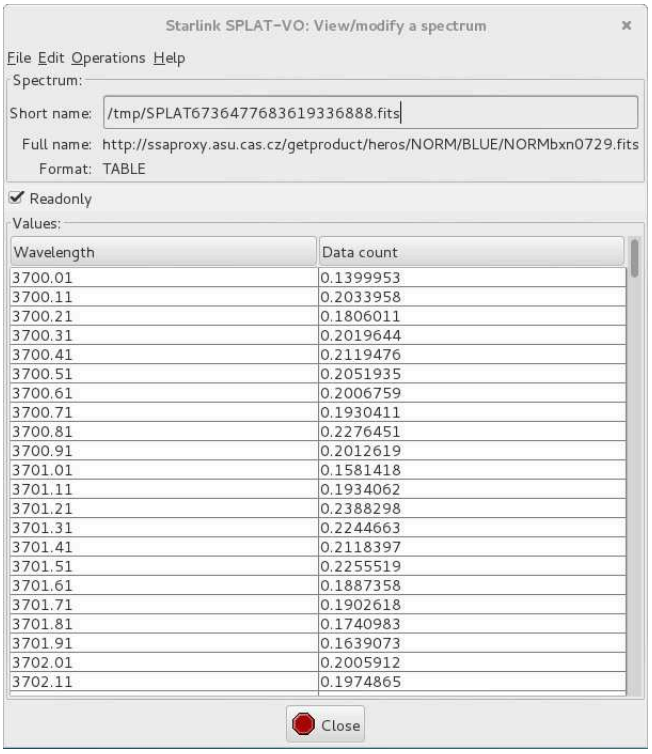


Figure 19: View/modify a spectrum.

User can also send the opened spectra via SAMP protocol to other tool and/or save the spectra in a different format and/or save the entire list of spectra to a local file [34]. He/she can perform many operations on the spectra. Just to get an idea:

- plotting one or more spectra to a plot window [34]
- performing actions on a plotted spectrum/spectra (e.g. cutting, fitting) [34]
- adjusting the visualization of plotted spectra (e.g. color or style of line etc.) [34]

4.5 Technical Description

SPLAT-VO is a desktop application written in Java SE[34] 1.6 (yet the original SPLAT was compatible with version 1.5). The GUI is built using Java Swing library, which makes it look the same way on all supported platforms.

Using the Java platform, SPLAT-VO is in general multi-platform and multi-arch application, yet the internal usage of native libraries from the original Starlink project, that are written in Fortran, C and C++, limits the platform support to following:

- GNU/Linux

[34] Java platform homepage: `https://java.com`

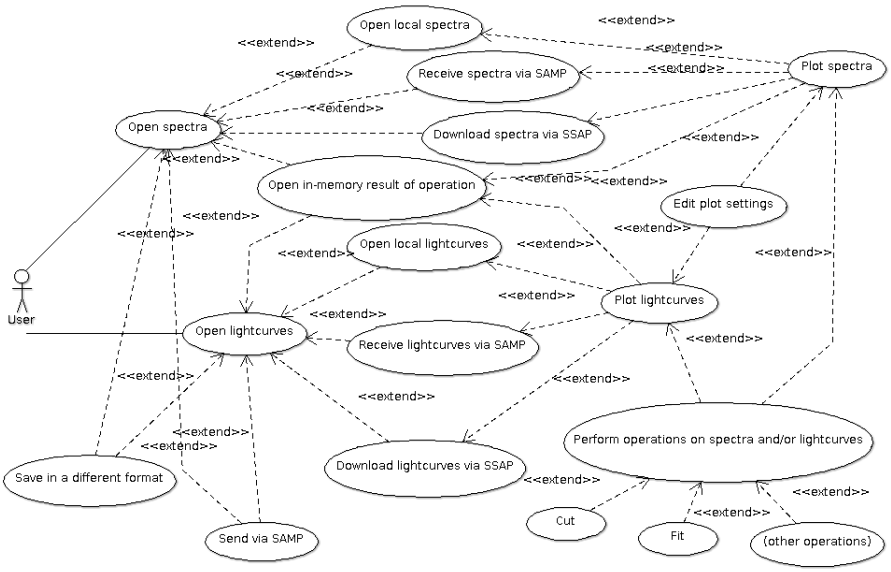


Figure 20: Use-case model of the most typical use cases. Source: [34].

- MS Windows
- Mac OS

for two major architectures: x86 and x86-64.
For a build, SPLAT-VO uses a customized Apache Ant[35] and some of its modules supports scripting in BeanShell[36] and using Java EE.

4.5.1 Basic Entities and Architecture Overview

This section covers fundamentals of SPLAT-VO architecture and concepts.

SpecData, RemoteSpecData and SpecDataImpl SPLAT-VO is capable of loading spectra in many data formats (e.g. FITS, VOTable, etc.). This is achieved by internal conversion to an abstract entity called `SpecData` (see Fig. 21), that covers all common spectra properties (data and metadata). For spectra that comes from remote sources, SPLAT-VO uses the extension of `SpecData` called `RemoteSpecData` [34]. For a complete UML class diagram, please refer to appendix A.

This abstract representation of spectra is internally used by SPLAT-VO features. This abstract representation of spectra is internally used by SPLAT-VO features. `SpecData` also keeps a reference to `SpecDataImpl`, which is an interface for format-specific operations of the original spectrum file (e.g. `FITSSpecDataImpl`).

[35] Apache Ant homepage: `http://ant.apache.org`
[36] `http://www.beanshell.org/`

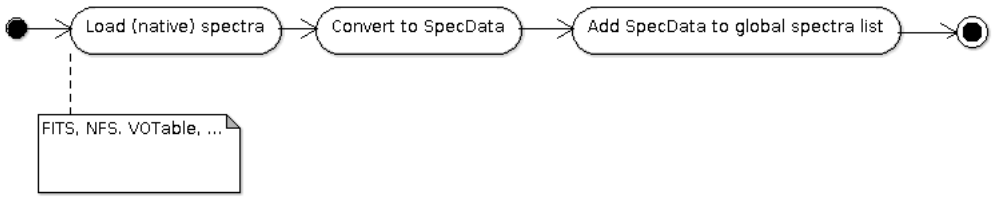


Figure 21: Workflow of loading spectra in native format to internal SpecData format. Source: [34].

SpecList is a singleton class that holds all spectra (instances of `SpecData`) loaded in SPLAT-VO. In this text, the term SpecList is therefore **interchangeable with *Global list of spectra*** (it is its implementation) [34]. For its complete class diagram, please refer to appendix B.

GlobalSpecPlotList is an aggregate singleton class that provides a direct access to the `SpecList` instance and list of all plots. It provides integrated control interfaces to both these objects and provides listeners for objects that want to be updated about changes in the lists of spectra or plots [34]. For a complete class diagram, please refer to appendix C.

SplatBrowser `GlobalSpecPlotList` is also referenced by `SplatBrowser` - the central SPLAT-VO window (see 4.3.1). So whenever this text notes that a spectrum is added to `SplatBrowser` or *Global list of spectra*, it means that it is actually added to `GlobalSpecPlotList` (and `SpecList` respectively) instance.

`SplatBrowser` class contains (among others) one crucial method called `tryAddSpectrum()`. This method (and its overrides) is responsible for loading spectra to `GlobalSpecPlotList` instance based on provided metadata (spectrum URL etc.).

SpectrumIO and Props As the name implies, `SpectrumIO` class is responsible for loading spectra to `SplatBrowser` (and *Global list of spectra* respectively), saving spectra (meaning individual `SpecData` instances) to various data formats (FITS, VOTable etc.) and serializing the entire *Global list of spectra* to a single file. As many other core classes in SPLAT-VO, `SpectrumIO` is a singleton class, so there is always a single existing instance.

Props is a public and static class embedded within `SpectrumIO`. It is a container class for describing the properties of a spectrum to be loaded (name, URL, pre-defined type, shortname, units etc.). Some metadata are required to be compatible with *Starlink AST*[37] library (please consult with JavaDoc for details). For a complete class diagram of `SpectrumIO` class, please refer to appendix D.

[37]Starlink AST homepage: `http://starlink.eao.hawaii.edu/starlink/AST`

SpecDataFactory creates and clones instances of `SpecData`. The type of the spectrum supplied is determined either by heuristics (based on the specification in `Props` instance) or by a given, known, type (`SpecDataFactory` also contains constants and enumerations of supported data formats and sources). For a complete class diagram, please refer to appendix E.

Process of Loading Spectra to SPLAT-VO A central point to add a spectrum or spectra to *Global list of spectra* is its owner, that is `SplatBrowser` instance (see section 4.5.1). This class contains several overriden methods called `addSpectrum()` and `tryAddSpectrum()`. The basic difference between them is only in the way they react on exception:

- `addSpectrum()` shows an error dialog (extension of `javax.swing.JDialog`)
- `tryAddSpectrum()` throws an exception (extension of `java.lang.Extension`)

They are called mainly from UI (`SplatBrowser` and its `displaySpectrum()` methods), `SpectrumIO`, `SpecList` and *Splat SOAP server*. Their arguments are also various: from spectrum URL and suggested type ID through `Props` instance containing spectrum metadata to entire `SpecData` instance.

The particular algorithm used for loading depends on provided parameters (a concrete overriden method), but the general idea is same in all cases as demonstrates the simplified UML sequence diagram (see Fig. 22) for method `tryAddSpectrum(SpectrumIO.Props props)`:

1. The user initiates the loading of spectra via UI etc.
2. `SpectrumIO` handles the loading in separate thread (instance of `SpectrumIO.Loader` class) by calling `tryAddSpectrum()` method of `SplatBrowser`.
3. Based on the provided `type`, `SplatBrowser` decides whether to expand spectra (via `SpecDataFactory.expandXMLSED()` method) or parse it. The spectrum is located on provided URL.
4. `SplatBrowser` then loops over all found spectra on the provided URL and checks whether it contains other embedded spectra. Then, for all found spectra, it calls `GlobalSpecPlotList` and its `.add()` method to add each spectrum to *Global list of spectra*.

The entire process of spectra loading is thread-safe.

4.6 Build Example

SPLAT-VO can of course be downloaded and installed from its homepage, but for some use cases, it might be useful to build the current snapshot. This HOWTO basically follows steps described in `README` file of Starjava package. The build is tested on GNU/Linux.

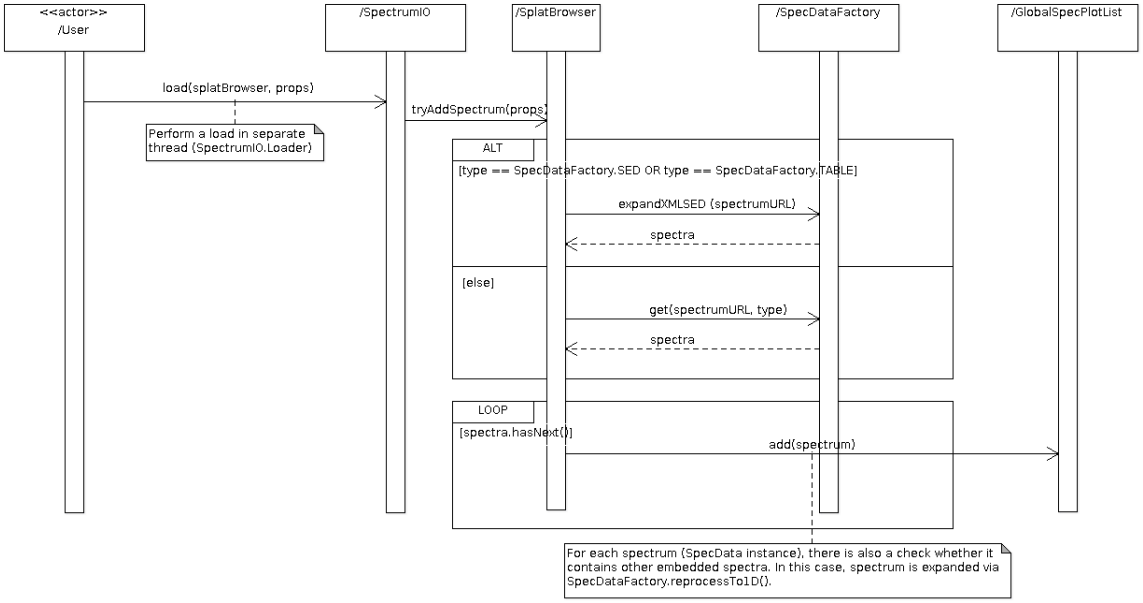


Figure 22: Simplified spectra loading sequence diagram.

4.6.1 Prerequisites

- Java Development Kit >=1.6 (we are officially supporting JDK 7, but for now, SPLAT-VO is still compiled to be compatible with JDK 6 using `target` argument)
- Java Advanced Imaging API[38]
- Set `STAR_JAVA` system variable to location of `java` binary. For example, if the JDK is installed in `/opt/java/jdk-1.6`, then the location should be `/opt/java/jdk-1.6/jre/bin/java`:

```
$ export STAR_JAVA=/opt/java/jdk-1.6/jre/bin/java
```

4.6.2 Build Steps

If all prerequisites are set, SPLAT-VO can be built by the following procedure:

1. Clone `Git` repository (`splat-gavo` branch)

```
$ git clone -b splat-gavo https://github.com/Starlink/starjava.git
```

2. Build the customized `ant`:

   (a) Enter the cloned directory `starjava` containing source files:

[38] JAI binaries and installation HOWTO: `http://download.java.net/media/jai/builds/release/1_1_3/INSTALL.html`

```
$ cd starjava
```

(b) Enter `ant` subdirectory with the customized `ant` source files

```
$ cd ant
```

(c) Adjust the `PATH` system variable to contain `ant` binaries required for `ant` build (overrides possibly existing `ant` installation)

```
$ export PATH=‘pwd‘/bin:$PATH
```

(d) Check that the customized `ant` is on the first place in the `PATH` system variable

```
$ whereis ant
```

(e) Back to `starjava` directory, set `ANT_BUILD` system variable to point to `ant` build directory and enter back the `ant` subdirectory:

```
$ cd .. # back to starjava directory
$ export ANT_BUILD=‘pwd‘/bin
$ cd ant # back to starjava/ant subdirectory
```

(f) Build `ant`:

```
$ ant -Dstar.dir=‘echo $ANT_BUILD‘ clean
$ rm -R -f ‘echo $ANT_BUILD‘/*
$ ant -Dstar.dir=‘echo $ANT_BUILD‘ install
$ ant -Dstar.dir=‘echo $ANT_BUILD‘ clean
```

(g) Ajdust `PATH` system variable to point to newly built `ant`:

```
$ export PATH=$ANT_BUILD/bin:$PATH
```

3. Build Starjava

(a) Get back to `starjava` directory

```
$ cd ..
```

(b) Clean ...

```
$ ant clean
```

(c) Treat a bug - in `splat/build.xml` under `target="build"` add `javac compilerarg` (SPLAT uses some internal `java.sun.*` classes and `javac` compiler complains about `ContentType` - ignore it):

```
<compilerarg value="-XDignore.symbol.file" />
```

(d) Treat some bugs in missing XSLT templates for docs by entering (still in `starjava` directory):

```
$ mkdir -p bin/etc/xdoc
$ mkdir -p topcat/src/bin/etc/xdoc
$ mkdir -p ttools/src/bin/etc/xdoc
$ cp -v -R xdoc/* bin/etc/xdoc/
$ cp -v -R xdoc/* topcat/src/bin/etc/xdoc/
$ cp -v -R xdoc/* ttools/src/bin/etc/xdoc/
```

(e) Build (with proper encoding)

```
$ ant -Dfile.encoding=iso-8859-1 build
```

(f) The build should be ready in `bin/bin/splat/` subdirectory - check by running:

```
$ ./bin/bin/splat/splat
```

4.7 Building Using Build Script

There is also a build script[39] that can be used to automate the build process. To use it, just download it to `starjava` directory and run:

```
$ ./_builder.sh
```

It checks the environment, detects JDK and can be used for full Starjava build or parameterized build with different JDK. Once when built, it also provides capabilities to skip `ant` build or to build only `splat` subpackage of Starjava, which makes the build much faster. To see the full list of capabilities, run:

```
$ ./_builder.sh --help
```

4.8 Creating Installation Package

SPLAT-VO can be tested and moved between systems without creating an installation package, but for production use, an installer is a necessity.

[39] Buildscript URL: `https://drive.google.com/file/d/0B_Kr8xwkCpBQamV6X05OZlZfVWM/view?usp=sharing`

4.8.1 Prerequisities

To create an installation package, one need to meet with the following prerequisites:

1. Have SPLAT-VO already built (see sections 4.6 and 4.7).
2. Have *IzPack*[40] tool for packaging applications and creating installers for Java application installed. Please note, that SPLAT-VO scripts are compatible with IzPack v4.x, version 5.x is not currently supported.
3. Have *C shell* installed (`csh` or `tcsh`[41])

4.8.2 Installer Creation Steps

- Add IzPack binaries to `PATH` system variable:

```
$ export PATH=${IZPACK_BIN_PATH}:$PATH # e.g. /opt/IzPack/bin
```

- Download and extract *extra files*[42] (containing icons, scripts etc.) to build directory (parent of `starjava` directory):

```
$ cp -Rv ${EXTRA_FILES_DIRECTORY}/* ${BUILD_DIRECTORY}/
```

- Prepare the environment - prefer customized Apache Ant binary and set `STAR_JAVA` system variable:

```
$ PATH=`pwd`/ant/bin:$PATH
$ export STAR_JAVA=$JDK_PATH
```

- Install binaries:

```
$ ./scripts/targetdeps splat install
```

- Remove unnecessary files:

```
$ cd ..
$ ./removed_files.lis
```

- Build installer JAR file

```
$ ./doit.csh
```

[40] IzPack homepage: `http://izpack.org/`
[41] tcsh homepage: `http://www.tcsh.org/Home`
[42] Extra files tarball URL: `https://drive.google.com/file/d/0B_Kr8xwkCpBQdW0yWm54djJ1T28/view?usp=sharing`

The final installer is a standard JAR file, that can be executed by:

```
$ java -jar splat-vo.jar
```

Since the entire build and installation package creation is fully deterministic and can be automatized, a Jenkins CI job was created (see sections 3.2.2 and 7.3).

5 Realized Improvements of SPLAT-VO

This section is focused on improvements so far realized to SPLAT-VO. It describes new features in SAMP communication, work with FITS files, spectrum CSV export, some visual tweaks and demonstrational support of time series, that is currently being standardized and implemented (see section 6.1).

5.1 More Efficient Work with SAMP Protocol

SAMP protocol (described in section 2.1.2) is a powerful mean of interactive cooperativity. SPLAT-VO now extends its current support for it to allow sending also original spectral data that has not been processed by SPLAT-VO. This is important because SPLAT-VO does not always keep all metadata and/or data. Sending is possible in VOTable (see section 3.1.4) and FITS (see section 3.1.3) format.

In order to send a SAMP message via SAMP hub to other connected SAMP client, the sender needs to obtain from hub a list of clients that supports the *message type* of the message. For VOTable and FITS, the corresponding message type is:

- **VOTable:** `table.load.votable`
- **FITS:** `table.load.fits`

Beside the *message type*, in order to send the message as table, the message needs to have also set the *send type* parameter to `table`.

Last important parameter in SAMP message metadata is the spectrum URL identified by `Access.Reference` keyword. This is a little bit tricky in case of in-memory spectra, that are results of some SPLAT-VO operation on opened spectra and has no URL. These files are therefore stored to system temporary directory and their URL in this storage is sent in SAMP message.

The implementation (shown on Fig. 23) uses `UniformCallActionManager` from the original Starjava project. This class has embedded classes `BroadcastAction` and `SendAction` that extends standard Java Swing `AbstractAction` class and are initiated by `UniformCallActionManager` on request from SPLAT-VO action managers initiated by `SampCommunicator` class and provides UI menus and event handlers for SAMP communication.

Unfortunately, `UniformCallActionManager` does not provide a required level of customization via its API (especially modifying the *message* and *send type*) and since it is not a direct part of SPLAT-VO, it is now *forked* and modified (including classes `BroadcastAction` and `SendAction`) in SPLAT-VO as `SplatUniformCallActionManager`. This action manager is extended by `SpectraAsTablesSendActionManager` (see appendix F.3) that also implements a new interface `EventEnabledTransmitter` extending the original SAMP `Transmitter` interface and standard Java AWT `MouseListener`, so it provides a standard SAMP functionality

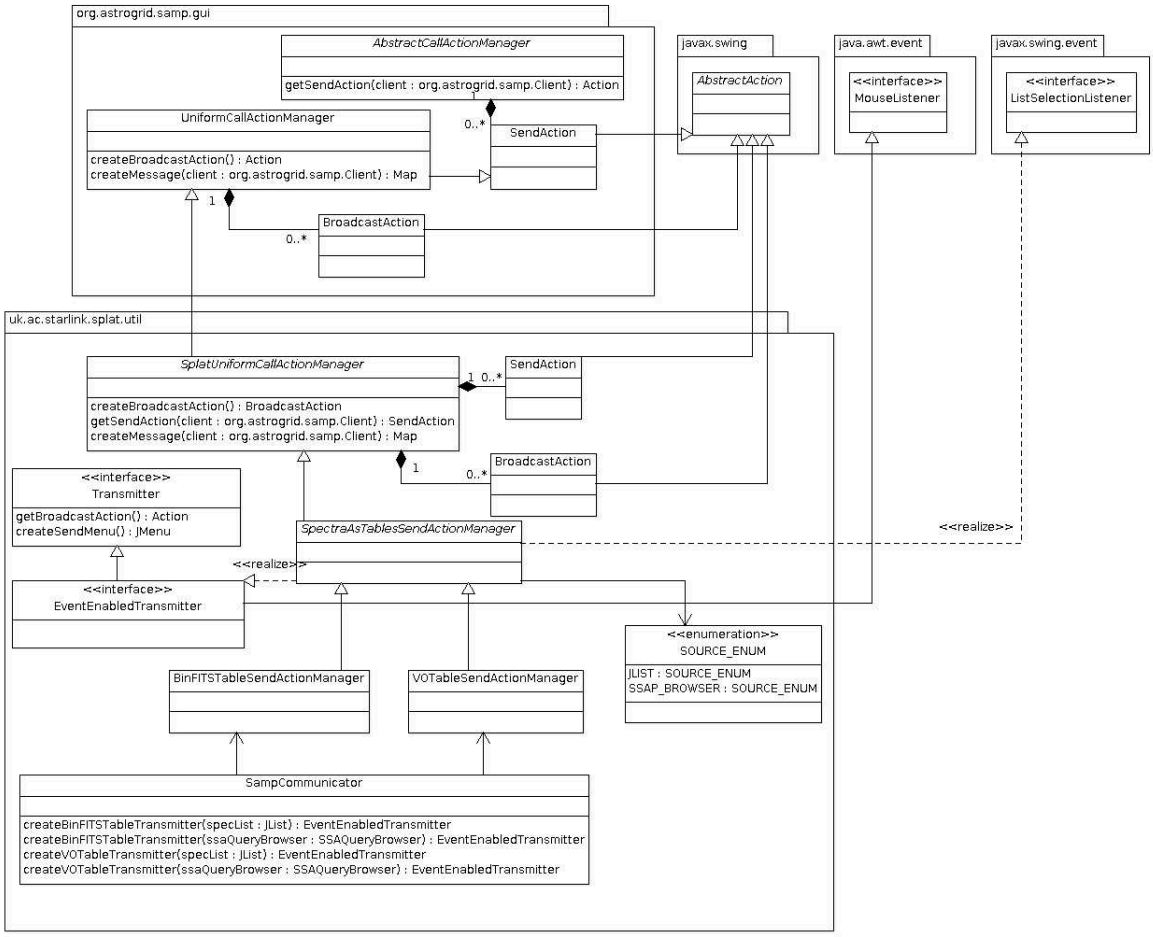


Figure 23: Simplified class diagram of *spectra as table SAMP sender*.

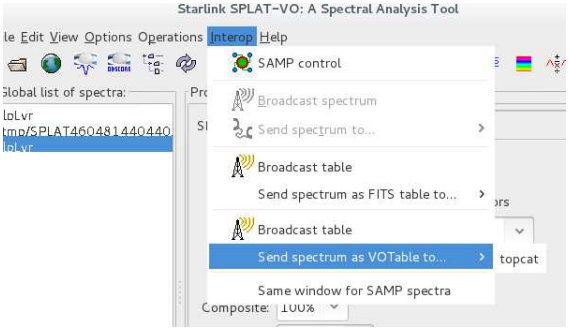


Figure 24: Sending spectrum as table via SAMP from *Main Window*.

and mouse event handler. It also imbeds a `SOURCE_ENUM` that identifies the source type that requests a communication via SAMP and affects the way how `SplatUniformCallActionManager` handles creating of the SAMP message.

`SpectraAsTablesSendActionManager` provides a common functionality for its descendants: `BinFITSTableSendActionManager` and `VOTableSendActionManager` (see appendix F.4). These classes are responsible for creating a SAMP message with approprite *message* and *send types*, MIME type and spectrum URL. They are instantiated by `SampCommunicator` that is called directly from *Main Window* or *Query VO for spectra*.

5.1.1 Sending Spectra Content from Main Window as Table

As described earlier, SPLAT-VO is now capable of sending the original spectra as table via SAMP. One of the way how to achieve this is *Main Window*, as shown on Fig. 24.

Technically, it as achieved by calling above noted `SampCommunicator` and its overriden methods `createBinFITSTableTransmitter(specList:JList)` for FITS SAMP interoperability and `createVOTableTransmitter(specList:JList)` for VOTable SAMP interoperability.

5.1.2 Sending Spectra Content from SSA Query Browser as Table

SSA query browser (also known as *Query VO for spectra*) is another part of SPLAT-VO UI from where the user can send a spectrum (or spectra) via SAMP as table (see Fig. 25). This place is for sending even more important than the *Main Window* since it allows to send directly selected results from SSAP query which highly improves the efficiency of work.

Technically, it as achieved by calling above noted `SampCommunicator` and its overriden methods `createBinFITSTableTransmitter(ssaQueryBrowser:SSAQueryBrowser)` for FITS SAMP interoperability and `createVOTableTransmitter(ssaQueryBrowser:SSAQueryBrowser)` for VOTable SAMP interoperability.

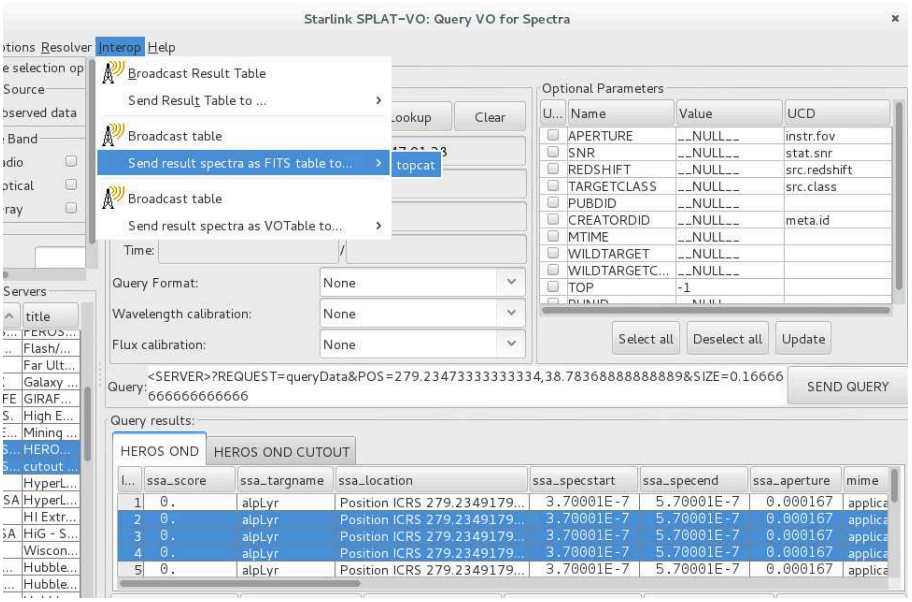


Figure 25: Sending spectrum as table via SAMP from *Query VO for spectra* window.

5.2 Access to All FITS Extensions

First version of this feature was already presented in [4] and is being continuously improved since then.

In the historic versions, SPLAT-VO was (simply put) reading only the first HDU found in the FITS file. But FITS can contain many HDUs in a form of an extension as described in 3.1.3. These extensions were basically inaccessible in SPLAT-VO [4].

Current versions of SPLAT-VO are able to load all HDUs located in the input FITS file. Each HDU produces a spectrum (a `SpecData` instance) in the *Global list of spectra* referencing the corresponding FITS header. This implicates that features like showing the FITS headers or ordering by its metadata (contained in FITS headers) are automatically working with this new feature, allowing SPLAT to become a general-purpose, feature-rich FITS viewer and editor as well and even more importantly, with ability to access all FITS headers (or more precisely its metadata) also improvement of its spectral analysis capabilities [4].

This is possible thanks to a gentle refactoring of SPLAT-VO architecture, that so far expected (generally speaking) that one source (file etc.) provides one spectrum. Newly, SPLAT-VO expects the multiplicity *1:M* so when there will be a need in the future to add a similar functionality (to load more spectra from one source file), SPLAT-VO will be ready for it [4].

Currently, there is one drawback that remains unresolved: in FITS world, it is common to have a primary HDU with no spectral data, yet with common observational parameters. Since this HDU is skipped, these metadata are inaccessible. After a long internal discussion, handling multi-HDU FITS files will be left to user with a dialog with some possible actions (including this one) and with option to perform the chosen action to all loaded spectra.

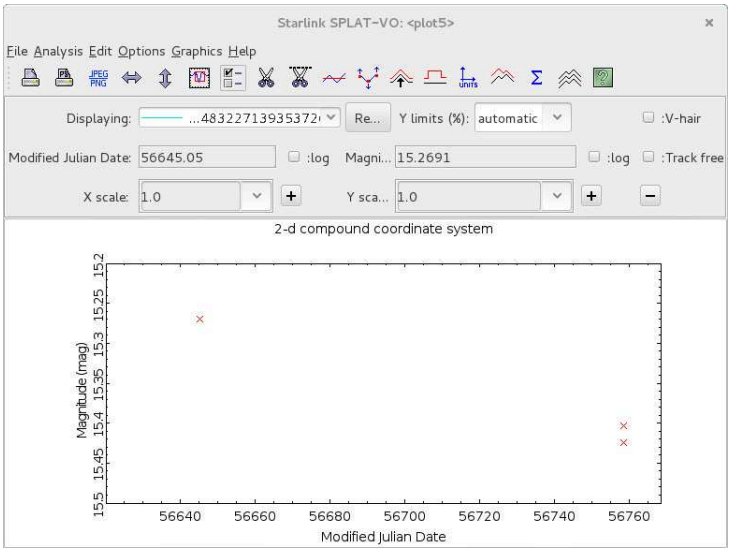


Figure 26: Time series demonstrational support example.

5.3 Time Series Demonstrational Support

This feature was also already presented at IVOA Interoperability Meeting in October 2015. In order to support the need for light curves (and time series in general) standard, that is crucial for handling data from several large scientific projects (including LSST[43], LIGO[44] and more), a demonstrational support of time series was implemented to SPLAT-VO. With support on server side in Astronomical Institute of the Czech Academy of Sciences, SPLAT-VO is capable of:

- Showing correct units on time axis (Julian Date and Modified Julian Date)
- Inverting y-axis in case of magnitudes (where the lower number indicates more brighter object)
- Customized visualization in *Plot window*, where the time series are plotted as larger crosses instead a polynom

See Fig. 26 for an example.

For data discovery, a standard SSAP protocol is used and the detection of time series is done by looking for Spectral Data Model (see section 2.1.2) parameter `ssa_producttype` (see appendix F.1 for code sample) in VOTable of:

- SSAP query results
- locally opened VOTable file

[43]The Large Synoptic Survey Telescope: `http://www.lsst.org/`
[44]Laser Interferometer Gravitational-Wave Observatory: `https://www.ligo.caltech.edu/`

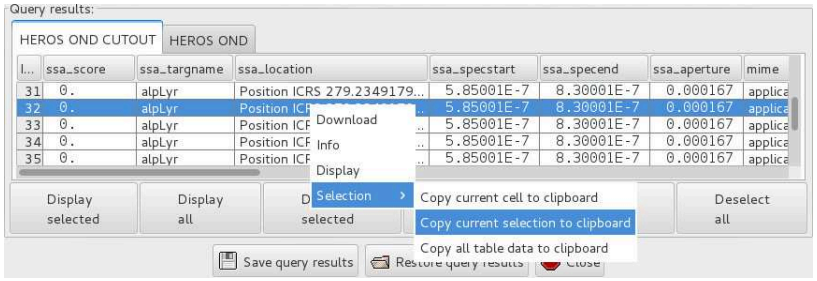


Figure 27: *SSA Query Results* window enhancements

Technical implementation consisted of new `ObjectTypeEnum` enumeration with `TIMESERIES` and `SPECTRUM` values. Creating this enumeration was the fastest way to achieve a suitable demo for time series, yet in future standardized implementation, this logic will move to inheritance-based architecture. `SpecData` and `TableSpecDataImpl` (VOTable format implementation of `SpecDataImpl`) classes were modified to contain support for `ObjectTypeEnum` and (since they are smart beans) to act accordingly based on these values. These classes are instantiated by `SpectrumIO` which was with `SpecDataFactory` also extended to accept this parameter and `SplatBrowser` now recognizes the `ObjectTypeEnum` in the core `tryAddSpectrum()` methods, which allows it to pre-set time series visual preferences and flip the Y-axis (see appendix F.2 for code sample) described above.

As the reader can see, this solution is only temporary, so it is not a part of standard SPLAT-VO distribution yet and there is a working progress on its standardization described in 6.1.

5.4 SSA Query Results Enhancements

One of drawbacks of SSAP query results in *Query VO for spectra* window was the inability to directly copy selected query results. When the user wanted (for example) copy a spectrum URL, he needed either to know a keyboard shortcut and copy the entire line (that can contain dozens of columns) or rewrite the URL manually. Therefore we added a new feature (shown on Fig. 27) that allows to:

- Copy current cell to clipboard
- Copy current selection to clipboard
- Copy all table data to clipboard

This feature has a form of context menu shown to user by clicking by right mouse button to query results table.

The implementation is centered around new `SSAQueryResultsTableSelectionMenu` class, that extends standard `javax.swing.JMenu` (see appendix F.6 for source code). This menu is added to a context menu in SSAP query results `JTable` and uses a new set of `JTable` utilities

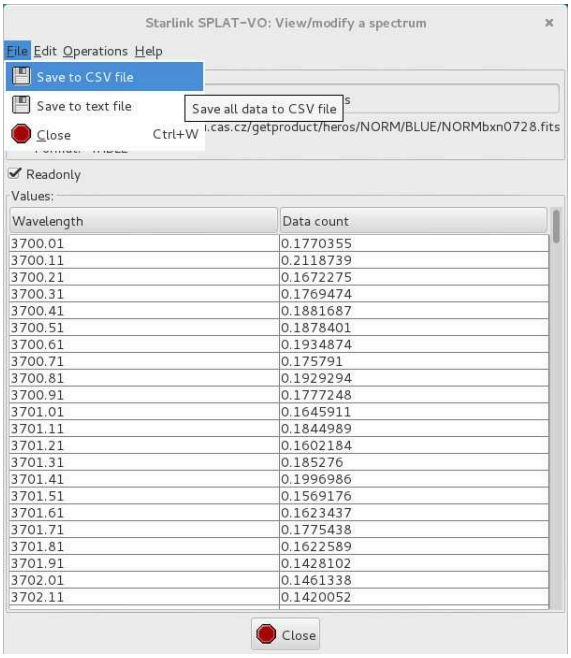


Figure 28: Export to CSV and text file feature.

in `JTableUtilities` class (see appendix F.5 for source code) to select the required content from SSAP query results `JTable`.

5.5 Spectral Data CSV Export

SPLAT-VO allows to view and edit spectral data in a form of a simple table. This feature is available from *Main Window* under *View - View/modify spectra values* menu. But it does not provide a direct functionality to export these values to an external file (exporting the result in-memory spectrum to FITS, VOTable etc. is not always a suitable option).

As shown on Fig. 28, SPLAT-VO can now export the spectral data to CSV (columns are delimited by a semicolon) and to a simple text file (columns are delimited by tabulator), that can both be used in other processing (including big data or manual). This feature is available under *File* menu.

As in case of copying results from *Query VO for spectra* window described in 5.4, implementaion uses a new `JTableUtilities` helper class (see appendix F.5 for source code) to select the required content from `JTable` containing the spectral data. This `JTable` is contained in `SpecViewerFrame` instance, that newly contains embedded classes initiating file chooser and handling the export itslef (extensions of standard `javax.swing.AbstractAction` called `SaveAsCSVAction` and `SaveAsTextAction` - see appendix F.7).

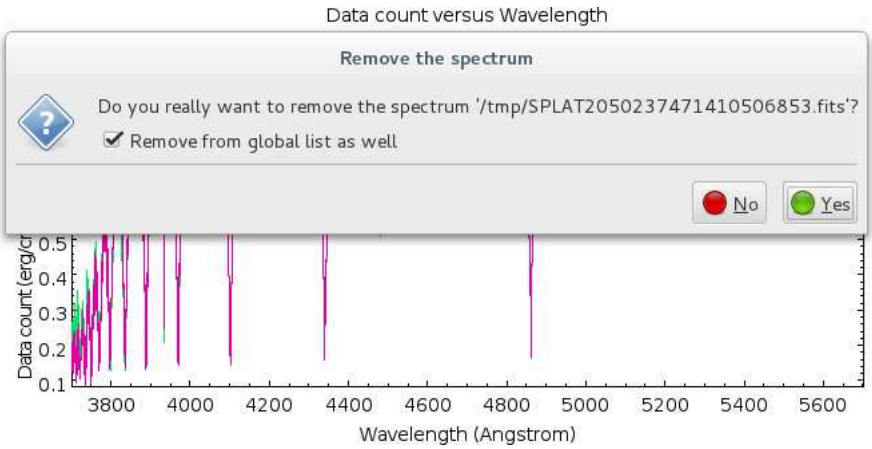


Figure 29: Visual delete of spectrum in action.

5.6 More Effective Spectra Deletion by Means of Visual Selection

This feature was also already previewed in [34]. When working with multiple spectra, it might be useful to select e.g. a noisy one and conveniently remove it from Working space [34].

The current version of SPLAT-VO allows a visual selection of a spectrum in *Plot window*. It highlights it and pre-selects in the local drop-down menu and *Global list of spectra* [34].

SPLAT-VO newly contains a delete feature, where after pressing a `Delete` key or clicking on an appropriate button, the user is asked to confirm the removal of the currently selected spectrum with the pre-checked option to remove it from the *Global list of spectra* as well (see Fig. 29).

The implementation consisted of creating a new `PlotControlKeyListener` (see appendix F.8) that detects a `Delete` key pressed and calls a new method `removeCurrentSpectrumFromPlot` (see appendix F.9) in `PlotControl` class that handles plotting the spectra. This method is also called from a new *Remove* button located at the right side of the drop-down menu with plotted spectra. Removing from *Main Windows* (`SplatBrowser` instance) is accompanied with visual pre-selection in *Global list of spectra*. This is ensured by a new interface `SpecListModelSelectionListener` that is implemented as an anonymous class in `SplatBrowser`. This listener listens for a new `SpecListModelSelectionEvent` fired by `SpecList` when new spectrum or spectra are set as selected.

6 Improvements Being Prepared for SPLAT-VO

Features described in this section are currently in a phase of development or pre-development (based on agreement with developers community and previous analysis acceptance). Part of these features were already previewed in [34]. Implementation of these features has been suspended because of change of priorities during the development (especially in favor of time series), but will continue as soon as more prioritized features will be implemented.

By implementing the following features, SPLAT-VO should allow the user to store all the data (including in-memory data) in a local storage and in an universal format, readable by other software tools. It will also read this stored data on startup, so it will allow the user to resume the previous work. It should also allow better organization of incoming data by grouping the data to user-defined groups. And it should improve the performance by lowering the memory consumption using the lazy loading of the spectra. These features are covered in sections 6.2, 6.3 and 6.4.

Currently most crucial feature in development is however a standardized support for time series, described in section 6.1 and presented in experimental phase on the upcoming IVOA Interoperability Meeting.

6.1 Time Series and Data Cubes Support via New Protocol

As mentioned in section 5.3, realized implementation of time series support was only demonstrational. It uses SSAP for data discovery, which is a bad practice since this protocol is intended for spectra only. Another bad practices can be found in implementation, that was only demonstrational.

This has changed lately thanks to Jiří Nádvorník's new data model using data cubes realized in VOTable. SPLAT-VO was given a support for this data model. It can open it and operate with it in the same way as described in section 5.3:

- Data discovery is performed via ObsCore protocol (see section 2.1.2).
- It looks for `<PARAM>` element in VOTable with `name="cube_dataset_producttype"` and `value="timeseries"` attributes. A corresponding (yet not final) UType is supposed to be `obscore:ObsDataset.dataProductType`.
- It will identify time axis (UType `spec:Cube.Data.TimeAxis.Value` - Modified Julian date in our example) and observable axis (UType `spec:Cube.Data.ObservableAxis.Value` - magnitude) and find corresponding data columns within VOTable (`<TABLEDATA>` element - see section 3.1.4).
- It will load and apply visualization properties for time series:
  - Render data by points (instead of a polynomial line as in case of spectra)

- Larger size for each point
- Render points as crosses
- Invert Y-axis

Visualization was refactored since demonstrational support. It is now driven by individual rendering properties for spectra and time series. When the object (spectrum or time series) is about to be rendered, SPLAT-VO will call a factory method (see Fig. 30) for creating the proper rendering properties. In case that object type could not be detected, the spectrum rendering properties are used as fallback.

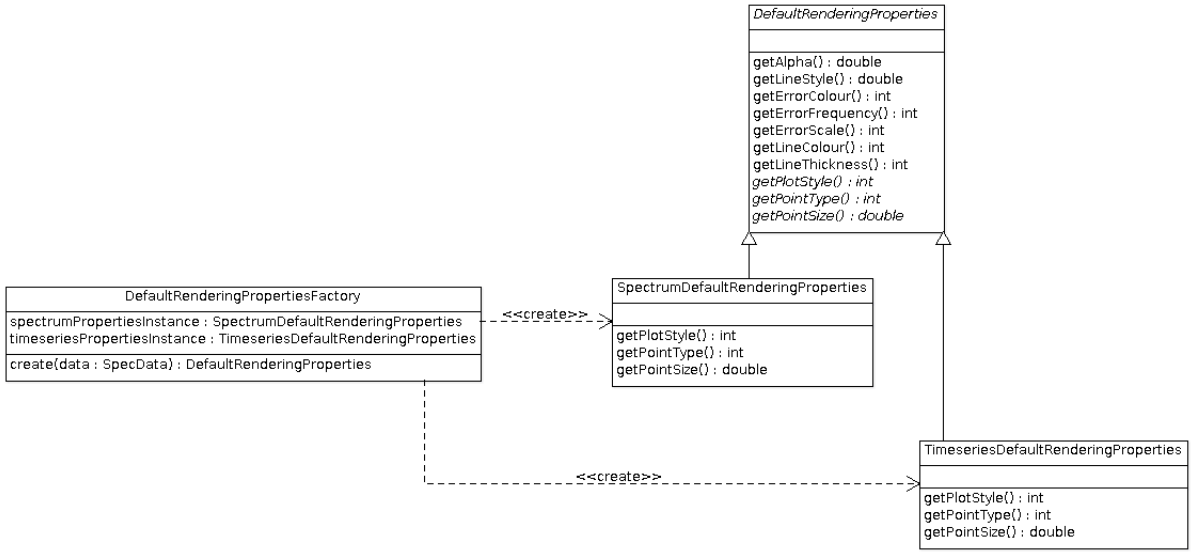


Figure 30: Default rendering properties factory.

This implementation of time series and data cubes support will be presented by Petr Škoda on the upcoming IVOA Interoperability Meeting in May 2016 in Cape Town. Based on the results, other refactoring is prepared:

- Inheritance-based architecture of `SpecData`. Currently, the object type (spectrum or time serie) is identified by `ObjectTypeEnum` enumeration tied to `SpecData` entity. This is sufficient for demonstrational purposes, but for a real use, it is unmaintainable. Therefore, a new abstract class called `ObservedData` with all common properties will be created. The current class `SpecData` will extend it, as well as a new class `TimeSeriesData`. The reason of waiting with the refactoring for results of IVOA Interoperability Meeting presentation is the fact that almost all current usages of `SpecData` class would have to be replaced by the new `ObservedData`, which is difficult to manage and priorities and implementation details may change after that.

- Larger support for data cubes: allow basic operations described in section 3.2.7. Basically, this will require switching between current and several more drop down boxes in visualization settings in *Main Window*, based on the object type. But the refactored rendering properties are now ready for this.

Aim of these iterative modifications is to continue on standardization of time series within IVOA (protocol and data model using data cubes), which is currently a pressing issue, as described in 5.3.

6.2 Working Space

Simply put: this feature will save all spectra loaded in SPLAT-VO at the moment to a user-defined folder [34].

In current implementation, SPLAT-VO downloads all remote spectra to a system temporary directory, spectra from local files and spectra that are results of some operations are not saved anywhere. This limits the user-experience because the user has no direct access to all loaded spectra in order to use it with other tools (manual saving of the spectra or sending it via SAMP is not sufficiently general and comfortable) [34]. By implementing this feature:

- The user will be able to restore the previous work in SPLAT-VO just by starting it (SPLAT-VO will ask the user to automatically load the content of the *Working space* if any).
- The user will be able to immediately work with all loaded spectra in other tools (the *Working space* directory will be editable and accessible, so the user can use other tools and access the stored spectra - **the access for other tools is read-only**, only SPLAT-VO will be able to write to a *Working space*)

There are also several issues that needs to be considered and solved during the implementation:

- Working space content must be synchronized with the global list of spectra (`SpecList`). This will be achieved by:
  - creating an interface `WorkingSpaceTransactionalProcedure` with `proceed()` method
  - creating a singleton class `SpecWorkingSpace` with `getInstance()` and `update(SpecList, WorkingSpaceTransactionalProcedure)` methods
  - adding Working space calls to `add` and `remove` methods of `SpecList` and:
  - making `SpecList`'s `add` and `remove` methods transactional by rewriting the original code to a form of anonymous `WorkingSpaceTransactionalProcedure` class implementation (the original code will be the body of its `proceed()` operation). The transactions themselves can be implemented in two ways:

    * using 3rd party library (XADisk[45] or Apache Commons Transaction[46]) *(preferred)*
    * writing the proprietary transactional system - probably using a lock file mechanism - at the beginning of a transaction, a lock file with metadata about transaction progress would be created and at the end of a transaction, this lock file would be deleted (means a *commit* of the transaction).
  - by attaching a new `SpecDataWorkingSpaceListener` (implementing already existing `SpecListener` - see Fig. 32) to a `GlobalSpecPlotList` and implement transactional event handlers for spectrum `MODIFIED` and `CHANGED` events (`ADDED` and `REMOVED` events are already covered and needs to be covered outside these event handlers because of the need to have add/remove methods completely transactional).
  - locking each file in the *Working space* while SPLAT-VO runs (using `java.nio.channel`) - this will ensure that *Working space* content will not be modified by any user or any other tool (in other words: no external action will compromise the integrity of the *Working space*)

- At least in the first release, the user should have an option to turn off this feature - this will be achieved by adding a new option under *Options* menu located at the top of the main window. This option will be called *Working space settings* and will open a simple window containing:
  - checkbox *Enable Working space* (checked by default) . **Disabling it will cause the deletion of the *working space* content (integrity precaution)**. An event will be fired when the state of the *working space* was changed.
  - directory selection tool to point to the *Working space* directory (label: *Working space directory*) - this field will be disabled if the *Working space* is not enabled
  - these settings will be stored in the system using the standard Java Preferences API (`java.util.prefs.Preferences`) as in case of other SPLAT-VO settings

- SPLAT-VO will have to be able to load all stored data during startup (if the *Working space* is enabled) to global list. This will be ensured by new method `loadFromWorkingSpace()` in `SpecList` and `loadWorkingSpaceContent()` in `SpecWorkingSpace` (throws `WorkingSpaceException` if anything goes wrong). Since the *Working space* content will be editable by external tools when the SPLAT-VO is not running (when running, the *Working space* content is locked and read-only), the loading will have to use general mechanism for opening a local file (same as in case of the *File -> Open* feature in the main window). In case of any error during loading the files, an `WorkingSpaceException` will be thrown,

[45] `https://xadisk.java.net/`
[46] `http://commons.apache.org/proper/commons-transaction/`

reported to the user via alert box and the *Working space* loading process will be aborted as one.

- SPLAT-VO deals with many sources and formats of spectra, but the `SpecList` contains only the `SpecData` instances. The *Working space* mechanism will therefore store these `SpecData` instances in a form of FITS with BINTABLE exetnsion (required by *Spectra groups* feature described in 6.3 - VOTable representing the spectra group can reference only a FITS BINTABLE, this also means that the original incoming spectra must be held in the system temporary directory as until now) with as many metadata as possible.

6.2.1 Use-case view

A list of *Working space* use-cases is shown at UML use-case diagram at Fig. 31.

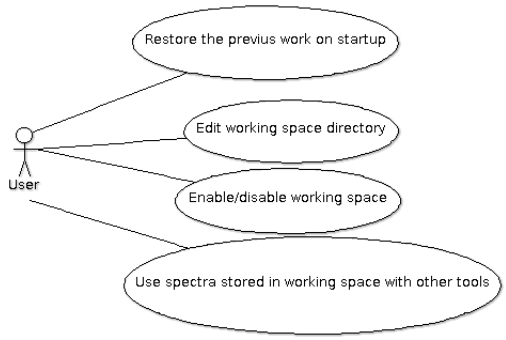


Figure 31: Working space - use case diagram. Source: [34].

6.2.2 Logical view

Logical view of *Working space* is represented by UML class diagram at Fig. 32. A complete description of individual classes is available at the beginning of this section.

6.2.3 Implementation view

Implementation view of *Working space* is represented by UML component diagram at Fig. 33. It shows a current SPLAT-VO using the standard *Global list of spectra*, that will be directly using the new *Working space* component.

6.2.4 Process view

There are two crucial processes that for better understanding requires a visualization using UML activity diagram:

- **Instantiation process** shown on Fig. 34 describes all neccassities required for successful instantiation of *Working space*.

- **Event handling process** shown on Fig. 35 describes all major events fired and handled by *Working space* and current SPLAT-VO components.

6.2.5 User interface impacts

Wireframes that visualizes user impacts of *Working space* described at the beginning of this section are shown on Fig. 36 (integration to *Main Window* menu) and Fig. 37 (a simple dialog with *Working space* settings).

6.3 Spectra Groups

This feature will allow the user to organize incoming spectra to groups and therefore to work with only a subset of all loaded spectra relevant to current work. When the lazy loading feature (described more closely in 6.4) is implemented, this feature will also allow to significantly reduce memory consumption of SPLAT-VO because only the spectra of the currently selected group will be fully initialized [34].

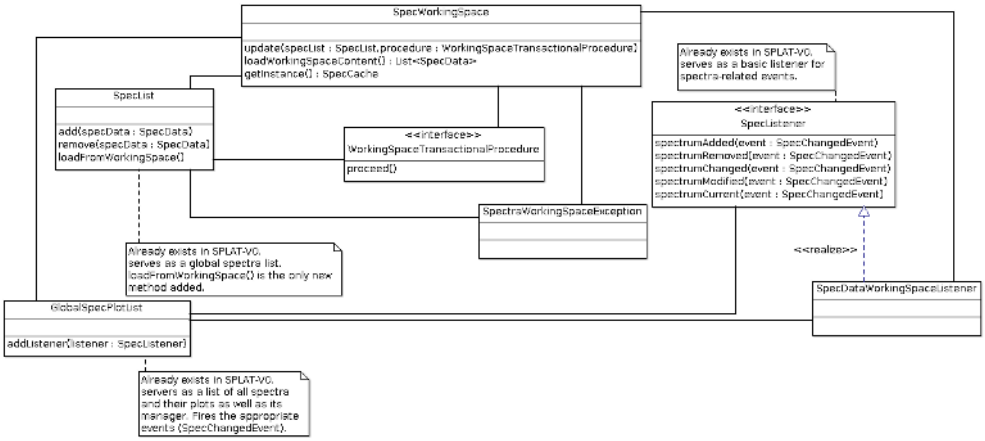


Figure 32: Working space - class diagram.

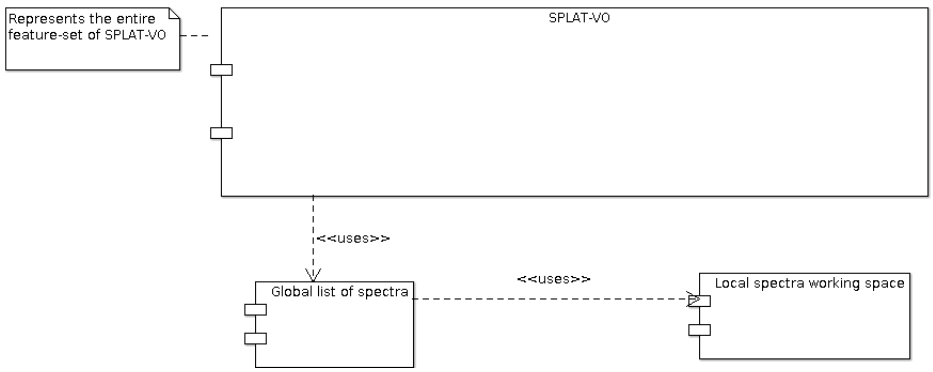


Figure 33: Working space - component diagram.

Implementation of this feature will require implementation of *Working space* as well (see 6.2), because it requires all the spectra to be locally available (references in group VOTable) and in the required format (FITS with WAVE/FLUX columns) [34].

In points:

- Each spectra group will be represented by VOTable stored in *Working space*. Each spectrum in the VOTable will be represented by an individual reference under corresponding `TABLE` element. All spectra will be grouped under a single `RESOURCE` element. Please refer to appendix G to an example of such a VOTable
- Since the current VOTable generators and writers works with a `StarTable` instance, a new (simplified) VOTable generator called `SpecGroupVOTableWriter` will need to be written. This new generator will have a `write()` method taking a `SpecData` instance as an argument, finding its reference in working space and generating the appropriate VOTable.
- The name of the VOTable in working space will be in this format:
  `splatvo_{unique UUID}.sg.vot`
- The user-editable name and description of the spectra group will be stored in VOTable `NAME` and `DESCRIPTION` elements.

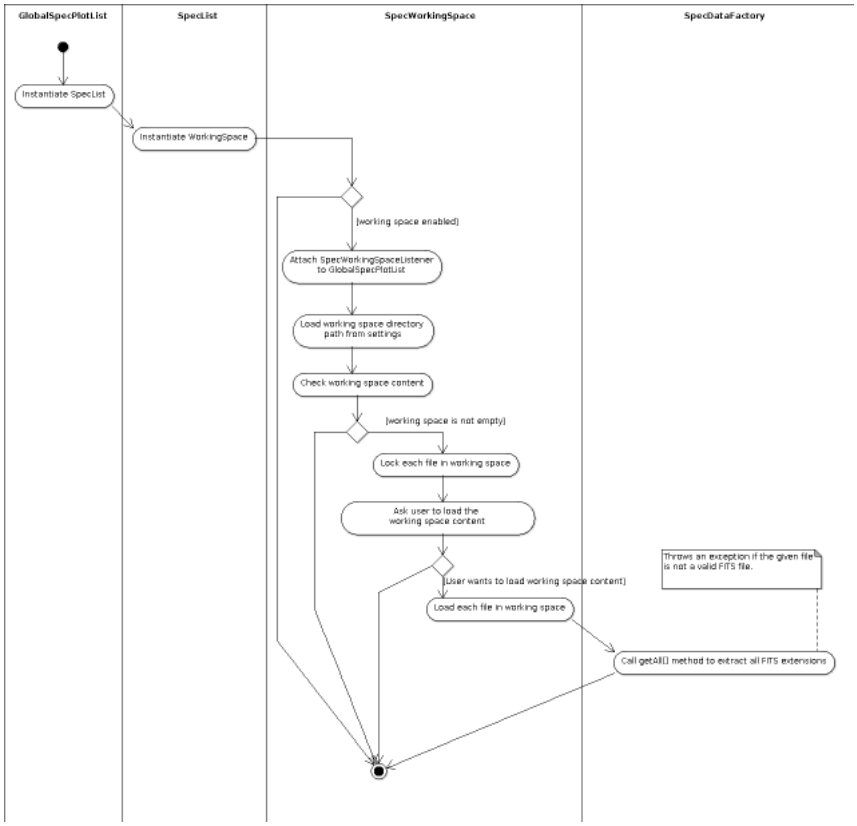


Figure 34: Working space - instantiation.

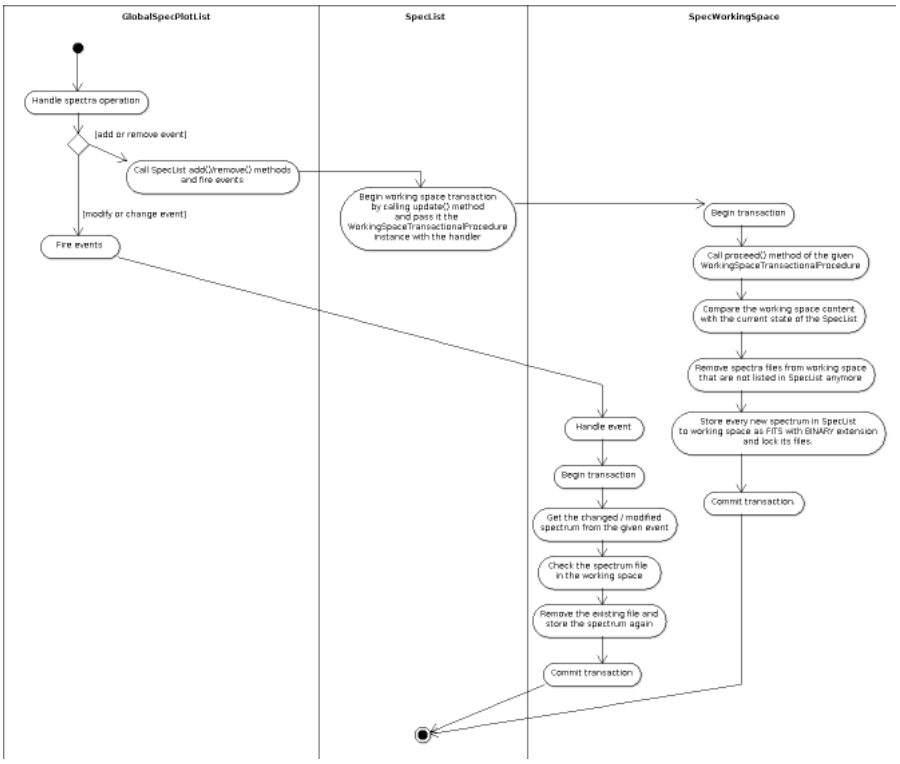


Figure 35: Working space - handling events.

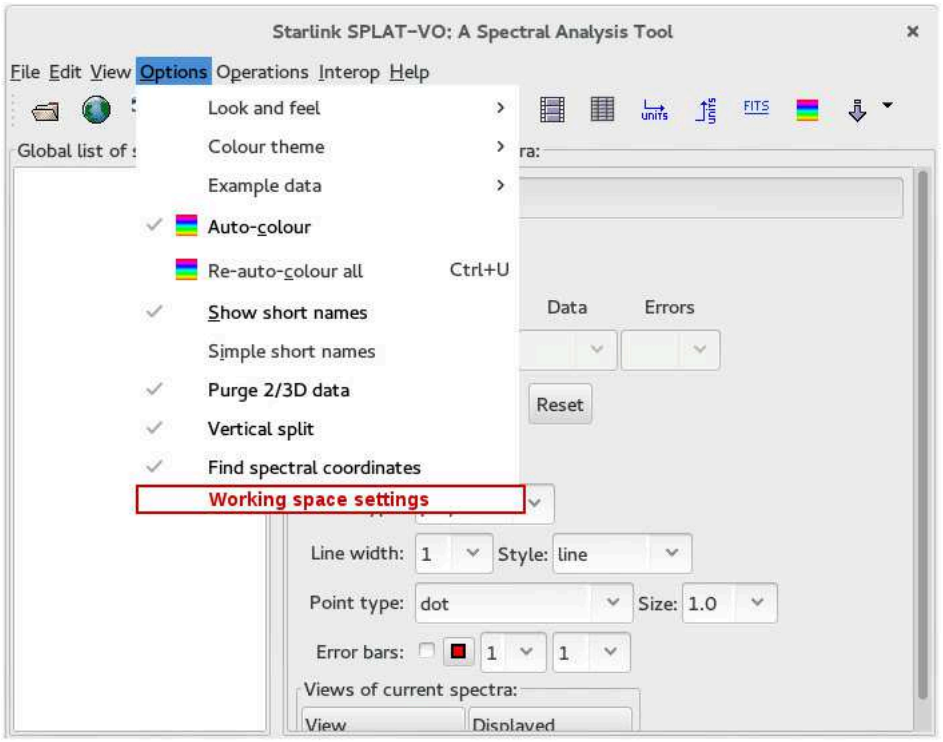


Figure 36: Working space options menu - wireframe.

- The spectra group will serve as a source of spectra loaded to the current *Global list of spectra* (in other words: the current *Global list of spectra* will contain only the spectra of the currently selected spectra group)
- Each spectrum added to *Global list of spectra* will therefore be added to the currently selected group (it can be later regrouped to one or n groups).
- Each spectrum will need to be assigned to at least one group
- There will be default group called `(default)`.
- A new panel with a selectbox, Create, Edit and Delete button will be added to a *Main Window*. It will be positioned left from global spectra list. A multiselectbox with a list of all assignable spectra groups will be added to the spectra details panel located right from *Global list of spectra*.
- This feature must take in count, that *Working space* may be disabled during the runtime. The *Working space* will fire an event about this and spectra groups will handle it by showing an alert window to the user. After closing it, all corresponding panels in the *Main Window* will be hidden and all models will be reset (that means also cleared, since the *Working space* content is deleted when the feature is turned off).
- In case of implementing the lazy loading feature, only the spectra of the currently selected group can be fully initialized - for more details, please refer to 6.4.
- Creating, editing and deleting of groups, as well as assigning the spectra to them, will need to be fully transactional - for possible ways of ensuring this, please refer to 6.2.

6.4 Spectral Data Lazy Loading

As shown on Fig. 38 (where figures *1* and *2* shows minimal and maximal memory usage based on JVM garbage collection status when no spectra are loaded and figures *3* and *4* shows minimal

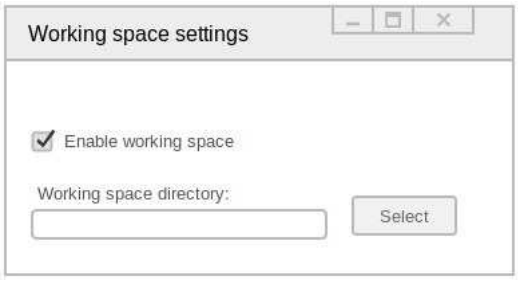


Figure 37: Working space settings - wireframe.

and maximal memory usage after loading 60 spectra) and Fig. 39, spectral data consumes a considerable amount of memory. Opening a large data set can cause `java.lang.OutOfMemoryError: Java heap space` error or significantly limit the usability of the user's system [34].

In order to avoid this, spectral data will be loaded "lazily". This means that only spectra of one spectra group (see section 6.3) or only e.g. actually plotted spectra (depends on further specification) will be "fully loaded" (including data itself, otherwise only metadata would be "fully loaded/initialized") [34].

Lazy initializaton (loading of all spectral data) will be done on demand (calling the getter) by loading the spectral data from *Working space* (see 6.2) [34]. The implementation of *Working space* is mandatory here because of in-memory and remote spectra that could not be covered by lazy initialization (will have to be fully initialized from the beginning or would have to implement much more complex mechanism).

The uninitialization will be performed on (last remaining) *Plot window* close event. Other features working with spectral data outside of *Plot window* will not cause spectrum uninitialization and the spectrum will remain initialized (these features are not used so commonly as *Plot window*, so implementing the uninitialization will be an overhead at this moment - anyway, the mechanism will exist in case of the need). Also, in case of implementing the spectra groups feature (see section 6.3), the uninitialization will be performed during the switch of the spectra groups (all spectra located in the old group would be uninitialized).

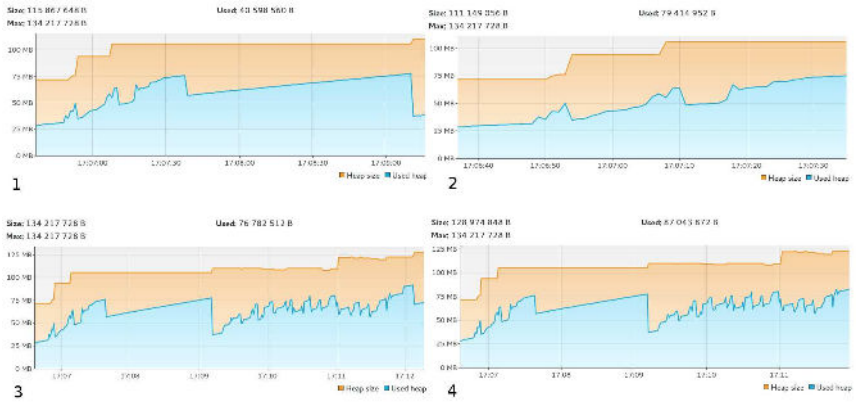


Figure 38: Memory usage of SPLAT-VO. Source: [34].

More technically

The `SpecData` will implement new `SplatLazyInitialization` interface with declared methods `initializeLazyFields(level:FieldInitializationLevel)` and its opposite called similarly `uninitializeLazyFields(level:FieldInitializationLevel)` , where `FieldInitializationLevel` is an enumeration with concrete levels of initialization (currently considered values: `ALL` and `FIRST`, which will load all data with exception of collections and data arrays).

The concrete initialization of the spectra will be performed using the new singleton class `SpecLazyInitializer` with `initialize(spectrum : SpecData, level:FieldInitializationLevel)` and `uninitialize(spectrum : SpecData, level:FieldInitializationLevel)` methods. The getter in `SpecData` will then call the `SplatLazyInitialization.initializeLazyFields(FieldInitializationLevel.ALL)` that will call the `SpecLazyInitializer.initialize(this, level)`.

A new method called `getStoredSpecData(spectrum:SpecData):SpecData` will be added to `SpecWorkingSpace` class. This method will return last stored variant of the given spectrum, so the `SpecLazyInitializer` can use it to initialize all required elements of the spectrum.

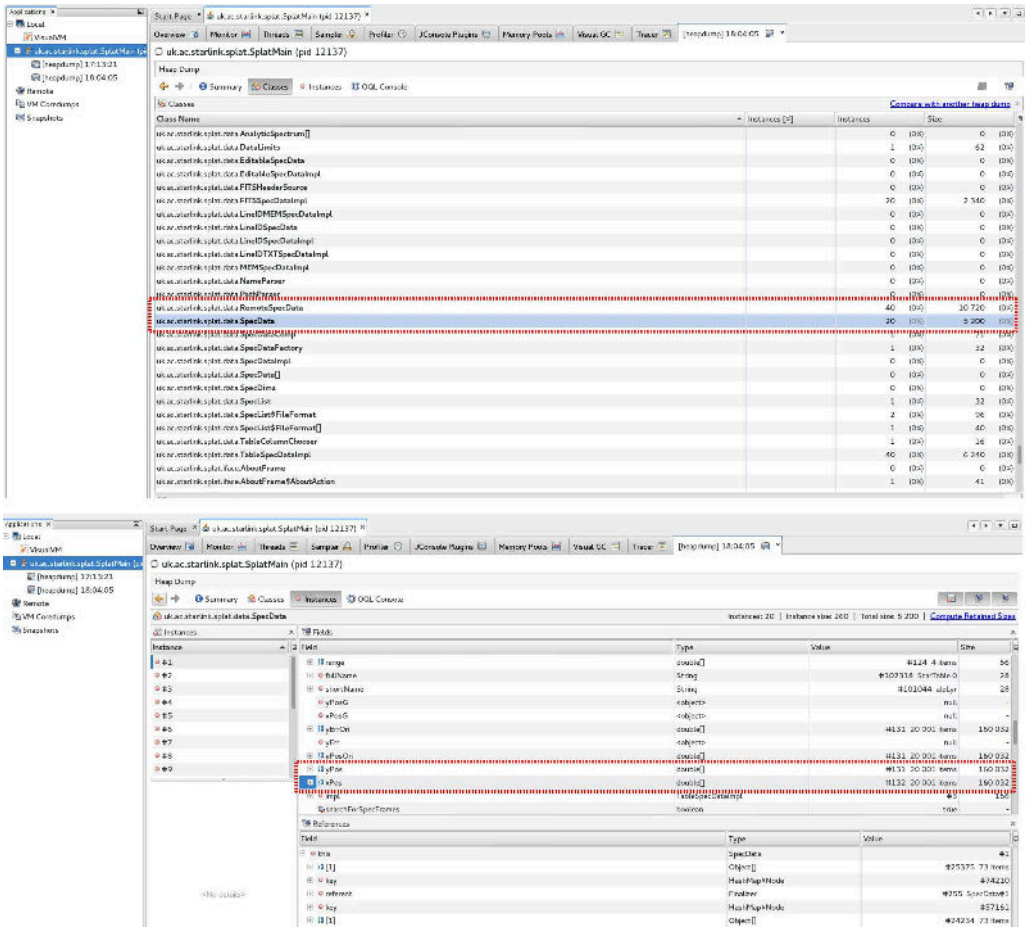


Figure 39: Memory usage of spectra. Please note the highlighted sections that shows how much memory spectral data consumes. It is clear that the spectral data are the largest objects of spectra instances.

7 SPLAT-VO Development Process Improvements

Among SPLAT-VO software modifications, there was also a need to improve and unify team communication, project documentation and ensure that the repository is always compilable in a sense of CI (see section 3.2.2).

7.1 Wiki Documentation

To make a central project documentation, a wiki page at Stellar Department of the AI CAS wiki was created[47] as shown on Fig. 40. This page contains basic information about SPLAT-VO, its history, development team and contacts, code repository and build information, list of published articles, issue tracking, continuous integration etc.

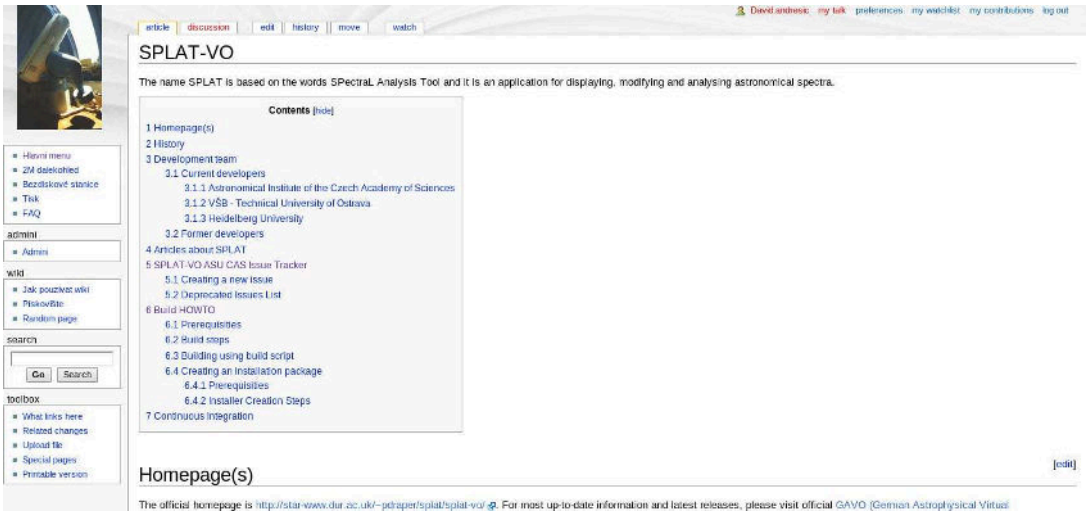


Figure 40: SPLAT-VO page at Stellar Department of the AI CAS wiki.

7.2 Issue Tracking

As described in section 4.2.3, current development process heavily relies on e-mail communications with all its disadvantages:

- Not all relevant participants are added to e-mail copies
- Many ideas are simply lost in tons of other ideas
- Many ideas are mixed with other in one e-mail conversation
- After several copies, it is hard to track even an authorship of individual parts of the conversation

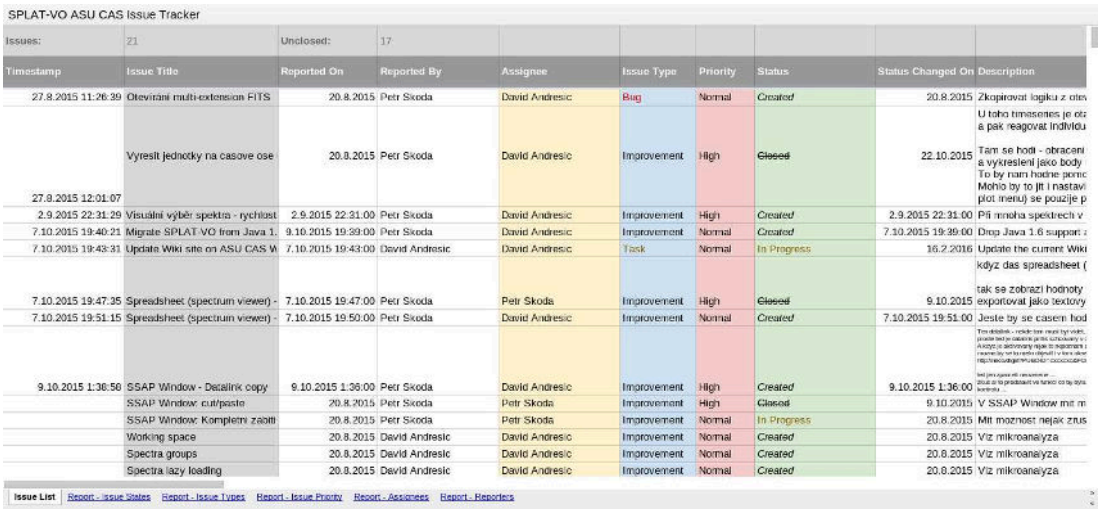
[47] Stellar Department of the Astronomical Institute of the Czech Academy of Sciences wiki: `https://stelweb.asu.cas.cz/wiki/index.php/SPLAT-VO`

- It is very hard to manage and track the scope of each version
- It is very hard to track the status of individual issues and requests

In order to solve this, a temporary and simple issue tracker was created. After some time, it was unified with a new official issue tracker.

7.2.1 Simple Issue Tracker Using Google Docs

At first, a growing number of new issues emerged and required a centralized tracking place. Therefore a simple issue tracker was created[48] using *Google Docs*[49]. It was using a simple *Google Spreadheet* (see Fig. 41) filled by pre-prepared *Google Forms* form (see Fig. 42) that created the individual issues.



SPLAT-VO ASU CAS Issue Tracker

Issues: 21 Unclosed: 17

Timestamp	Issue Title	Reported On	Reported By	Assignee	Issue Type	Priority	Status	Status Changed On	Description
27.8.2015 11:26:39	Otevírání multi-extension FITS	20.8.2015	Petr Skoda	David Andresic	Bug	Normal	Created	20.8.2015	Zkopírovat logiku z ote
27.8.2015 12:01:07	Vyresit jednotky na casove ose	20.8.2015	Petr Skoda	David Andresic	Improvement	High	~~Closed~~	22.10.2015	U toho timeseries je ot a pak reagovat individu Tam se hodi - obraceni a vykreslení jako body To by nam hodne pomo Mohlo by to jit i nastavi plot menu) se pouzije p
2.9.2015 22:31:29	Vizuální výběr spektra - rychlost	2.9.2015 22:31:00	Petr Skoda	David Andresic	Improvement	High	Created	2.9.2015 22:31:00	Při mnoha spektrech v
7.10.2015 19:40:21	Migrate SPLAT-VO from Java 1.	9.10.2015 19:39:00	Petr Skoda	David Andresic	Improvement	Normal	Created	7.10.2015 19:39:00	Drop Java 1.6 support
7.10.2015 19:43:31	Update Wiki site on ASU CAS W	7.10.2015 19:43:00	David Andresic	David Andresic	Task	Normal	In Progress	16.2.2016	Update the current Wiki
7.10.2015 19:47:35	Spreadsheet (spectrum viewer) -	7.10.2015 19:47:00	Petr Skoda	Petr Skoda	Improvement	High	~~Closed~~	9.10.2015	kdyz das spreadsheet (tak se zobrazi hodnoty exportovat jako textovy
7.10.2015 19:51:15	Spreadsheet (spectrum viewer) -	7.10.2015 19:50:00	Petr Skoda	David Andresic	Improvement	Normal	Created	7.10.2015 19:51:00	Jeste by se casem hod
9.10.2015 1:36:56	SSAP Window - Datalink copy	9.10.2015 1:36:00	Petr Skoda	David Andresic	Improvement	High	Created	9.10.2015 1:36:00	
	SSAP Window: cut/paste	20.8.2015	Petr Skoda	Petr Skoda	Improvement	High	~~Closed~~	9.10.2015	V SSAP Window mit m
	SSAP Window: Kompletni zabiti	20.8.2015	Petr Skoda	Petr Skoda	Improvement	Normal	In Progress	20.8.2015	Mit moznost nejak zrus
	Working space	20.8.2015	David Andresic	David Andresic	Improvement	Normal	Created	20.8.2015	Viz mikroanalyza
	Spectra groups	20.8.2015	David Andresic	David Andresic	Improvement	Normal	Created	20.8.2015	Viz mikroanalyza
	Spectra lazy loading	20.8.2015	David Andresic	David Andresic	Improvement	Normal	Created	20.8.2015	Viz mikroanalyza

Issue List | Report - Issue States | Report - Issue Types | Report - Issue Priority | Report - Assignees | Report - Reporters

Figure 41: Deprecated SPLAT-VO issue tracker created in Google Spreadheet.

This simple spreadsheet allowed every basic issue tracking feature that was required at the moment:

- Creating issues with name and description
- Adding comments to them
- Track changes (via comments and *Google Docs* versioning)
- Assign issues to concrete people
- Time tracking (creation date, last change date)

[48] Simple issue tracker in Google Docs: `https://docs.google.com/spreadsheets/d/1NQkhiBf3t6KkOar1n3Q2hYM6XOdQ8g-qt3ZpDB94fyo/pubhtml?widget=true&headers=false`

[49] Google Docs URL: `https://docs.google.com/`

- Status tracking (several states were supported)

This simple issue tracker was used only by Czech group of development team and is currently deprecated and replaced by the official issue tracker (see section 7.2.2).

7.2.2 Unified Issue Tracker Using GitHub Issues

During last months, we initiated and unified the official central issue tracking system. From several different possibilities described in section 3.2.4, we chose *GitHub Issues*. The main reasons were the following:

- Our repository was already hosted on GitHub
- It fully supports our basic ideas (each request or idea, its status, conversation, assignee, history and target version is kept within an issue)
- Supports basic workflow (create - close - reopen)
- With support of multiple labels (basic and custom), we can set a fine grained workflow and organize the work.
- With customizable milestones support we can set a target version for each issue and track the scope of each release
- We can assign issues to concrete people
- Supports basic filtering (by assignee, author, label, milestone and state)
- Has simple WYSIWYG editor

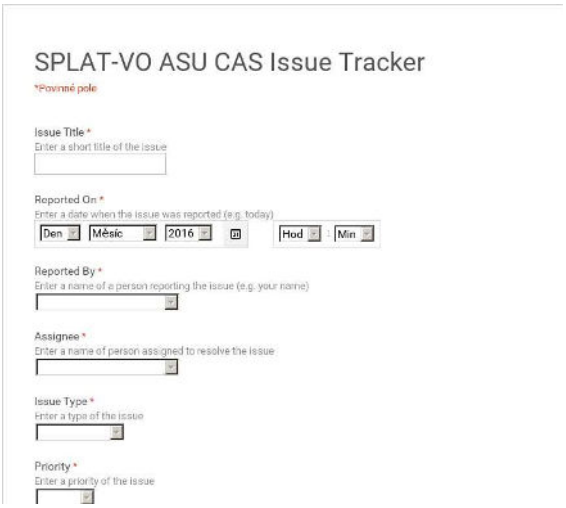


Figure 42: Deprecated SPLAT-VO issue tracker created in Google Spreadsheet - filling form.

- Privacy: on public repositories, all issues are visible to a regular, unregistered visitors, but the conversation can be locked and only team members can add comments.

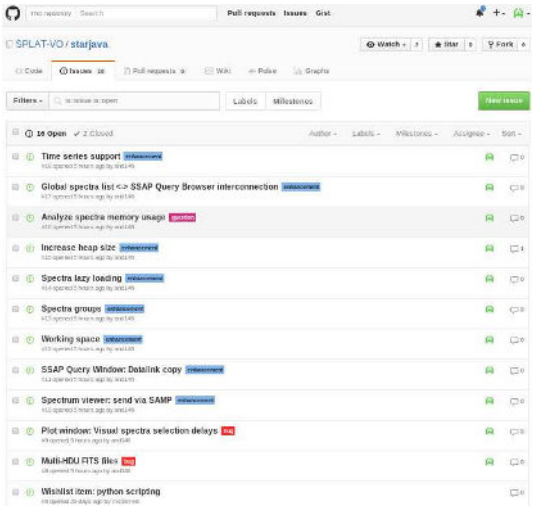


Figure 43: Official SPLAT-VO issue tracker in GitHub Issues.

For *GitHub Issues* also spoke the fact that anything more complex can be achieved by a combination of basic features and none of the other issue tracking systems is so easy to use (and deploy). Currently, we have all opened issues migrated to this new central issue tracker[50] (see Fig. 43)

7.3 Automatized Build with Jenkins CI inside Docker

In order to have a possibility to build SPLAT-VO installation package from the current repository and to ensure that the source code in repository is buildable at any time, we deployed Jenkins CI in a Docker container on server of Stellar Dept. AI CAS[51]. The container runs from custom Docker image derived from standard Jenkins CI Docker image[52].

As shown on Fig. 44, this instance of Jenkins CI contains at this moment one job (called `SPLAT-VO_Build-installer`), that is parametrized by a single parameter `Branch` identifying a Git branch from which the SPLAT-VO will be built (see Fig. 45).

Jenkins CI will pass these arguments to shell script that will build SPLAT-VO and create its installer by the same process as described in section 4.6 and 4.8. The resulting installers are versioned and accessible from the same Jenkins CI environment (see Fig. 46).

[50]SPLAT-VO GitHub Issues: `https://github.com/SPLAT-VO/starjava/issues`
[51]Jenkins CI instance for SPLAT-VO URL: `http://antares.stel:50080/`
[52]Standard Jenkins CI Docker image: `https://hub.docker.com/_/jenkins/`

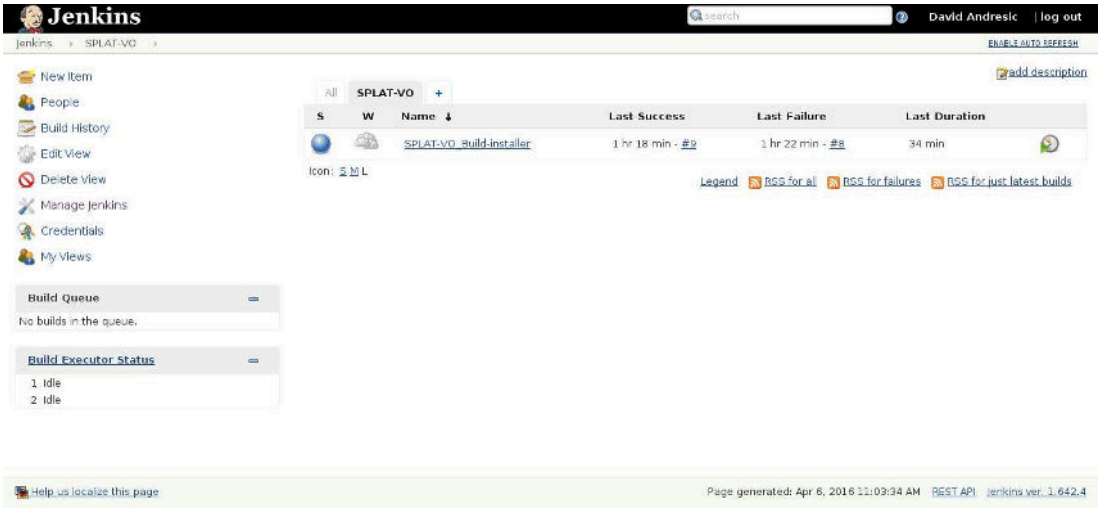


Figure 44: Jenkins CI for SPLAT-VO - main job.

Figure 45: Jenkins CI for SPLAT-VO - main job detail with parameters and result.

Figure 46: Jenkins CI for SPLAT-VO - build artifacts history.

8 Suggestions for Further Refactoring

Some parts of SPLAT-VO contain certain bad practices that works well at this moment, but makes future enhancements very difficult and tricky. Among those most obvious are:

Logic on presentation layer A typical example of this bad practice is *Query VO for spectra*, but basically every UI class is affected by this. The presentation layer is responsible for drawing the UI and it should not contain any other logic. SPLAT-VO heavily uses a non-UI code in UI classes which causes too many copy/pastes and makes further enhancements very difficult and unmaintainable.

Listeners embedded on presentation layers Listeners are the most used parts of SPLAT-VO affected by this bad practice, but in general: SPLAT-VO contains too many large classes that should be separated to several smaller, more readable and maintainable classes that makes a logical structure. A listener is a nice example of this, since it is tied to an event and not a particular window or panel, where it is placed at this moment.

Too many copy/pastes This bad practice relates to the above ones. The same or very similar code is copied and pasted elsewhere. A nice example of this is logic for loading spectra, which is contained in `SplatBrowser` in almost ten methods and again in *Query VO for spectra* and other parts of SPLAT-VO. This makes the code almost unmaintainable and change or enhancement of spectra loading algorithm is very time-wasting and buggy.

Strong dependence on native Starlink libraries This limits some enhancements because these native libraries are very little maintained in these days and very difficult and time-wasting to change. The fact that they are written in languages such as Fortran or C/C++ limits the portability of SPLAT-VO, which is crucial especially in modern days.

Using throw/catch in a sense of conditions A nice example of this bad practice is *Main Window* class `SplatBrowser` and its `addSpectrum()` method. Instead of determining the conditions, the code simply attempts to treat the input in one way and switches to another when exception is thrown. With not properly logging exceptions, this is very dangerous and makes any diagnostics and maintenance very difficult.

Insufficient use of unit tests SPLAT-VO contains only a few unit tests and new code is not covered at all. This produces very buggy releases since SPLAT-VO relies only on user testing of limited functionalities performed by one or two persons. This testing also does not cover regression testing, that ensures that the implementation of new features did not break the old code.

Use of deprecated tools and libraries From the most obvious libraries and tools, we can name *Apache Ant* (that has been replaced by *Apache Maven*[53] and *Gradle*[54]), jUnit (used in very old version) and Java Runtime Environment, that has been recently upgraded to 1.6, but even this is ten years old technology with deprecated memory management and missing modern features.

[53] Apache Maven homepage: `https://maven.apache.org/`
[54] Gradle homepage: `http://gradle.org/`

9 Conclusion

Deep and documented analysis of SPLAT-VO architecture allowed to implement the support for time series and data cubes, which is current pressing issue in astroinformatics. SPLAT-VO is also ready for more exotic forms of FITS data format and enhanced collaboration via SAMP protocol. With new data exports, SPLAT-VO significantly enhances collaboration with other tools in general. SPLAT-VO is now also improved in a way of user experience and performing a scientific research is now therefore more painless.

Development of SPLAT-VO is now more professional by using central issue tracking system, continuous integration and project Wiki documentation.

For further enhancements, there is a set of analyzed and prepared features that highly improves its work with large data sets and user experience that is designed for working on many different large projects. There is also a set of refactoring suggestions that should move SPLAT-VO more to modern ages as well.

During the work on this thesis, 17 improvements, 1 bug, and 3 general tasks have been created. From this number of issues, 6 were tested and closed, 2 are currently being tested, 5 are in progress (at least with deep implementation analysis) and 8 minor-severity issues or tasks are created and waiting for higher priorities to be resolved. These statistics (gathered in less than 10 minutes) also shows described benefits of the new issue tracking system that I have suggested and picked-up. For higher efficiency, I have suggested testing process enhancements that also includes higher usage of unit testing (the Jenkins CI instance, that I have created, is ready for this task). In dozens of new or modified classes (most of them core ones), several thousands of new lines of code has been added. During merges with other branches and analysis phase for large refactoring and improvements suggestions, considerably large part of code has been reviewed, which resulted in suggestions described at the end of this thesis. The number of issues that currently remains in progress and number of reviewed lines of code also shows how frequently and rapidly priorities change in SPLAT-VO development. This remains to be one of the greatest bottlenecks of SPLAT-VO development, yet benefits of the implementation of my suggestions are currently showing up. A new technical and non-technical documentation, that I have created helps to centralize the knowledge about SPLAT-VO and pass the know-how to new members of the development team.

My adjustments were accepted by SPLAT-VO development community. Results were accepted for Nostradamus 2015 conference, as well as presented on IVOA Interoperability Meetings in October 2015 and May 2016. I will submit current results to the SIMS2016 conference, where the deadline is after finishing this thesis.

Beside the need of major refactoring and other (not only) technical adjustments, that have been described in this thesis, SPLAT-VO is still a remarkable software tool of its time capable of serving to its purpose in these days with benefits of being a reference implementation to newly emerging VO standards.

References

[1] PennState (2016) *Astroinformatics in a Nutshell.* Available at: https://asaip.psu.edu/Articles/astroinformatics-in-a-nutshell. Cited on Mar 07, 2016

[2] Petr Škoda. *Optical Spectroscopy with the Technology of Virtual Observatory.* Baltic Astronomy. 2011, Vol. 20, p. 531-539. Cited on Mar 07, 2016. Available at: http://adsabs.harvard.edu/abs/2011BaltA..20..531S [online].

[3] International Virtual Observatory Alliance (2016) *International Virtual Observatory Alliance.* Available at: http://ivoa.net/. Cited on Mar 07, 2016

[4] Petr Šaloun, David Andrešič, Petr Škoda and Ivan Zelinka. *Visualization of Large Amount of Spectra in Virtual Observatory Environment.* International Journal of Automation and Computing. 2014, 11(6), 613-620 [cit. 2016-03-08]. DOI: 10.1007/s11633-014-0845-y. ISSN 1476-8186. Available at: http://link.springer.com/10.1007/s11633-014-0845-y [online]

[5] International Virtual Observatory Alliance (2010) *IVOA Architecture.* Available at: http://ivoa.net/documents/Notes/IVOAArchitecture/20101123/IVOAArchitecture-1.0-20101123.pdf. Cited on Mar 08, 2016

[6] International Virtual Observatory Alliance (2012) *SAMP - Simple Application Messaging Protocol.* Available at: http://www.ivoa.net/Documents/latest/SAMP.html. Cited on Mar 09, 2016

[7] International Virtual Observatory Alliance (2012) *Simple Spectral Access Protocol* Available at: http://www.ivoa.net/Documents/latest/SSA.html. Cited on Mar 09, 2016

[8] Mireille Louys, Doug Tody, Patrick Dowler, Daniel Durand, Laurent Michel, Francois Bonnarel, Alberto Micol and the IVOA DataModel working group (2016) *Observation Data Model Core Components and its Implementation in the Table Access Protocol Version 1.1* Available at: http://www.ivoa.net/documents/ObsCore/. Cited on Mar 10, 2016

[9] Doug Tody, Ray Plante (2009) *Simple Image Access Specification Version 1.0* Available at: http://www.ivoa.net/documents/latest/SIA.html. Cited on Mar 11, 2016

[10] Patrick Dowler, Guy Rixon, Doug Tody (2010) *Table Access Protocol Version 1.0* Available at: http://www.ivoa.net/documents/TAP/. Cited on Mar 11, 2016

[11] Roy Williams, Robert Hanisch, Alex Szalay, Raymond Plante (2008) *Simple Cone Search Version 1.03* Available at: http://www.ivoa.net/documents/latest/ConeSearch.html. Cited on Mar 11, 2016

[12] Mark Taylor (2015) *TOPCAT - Tool for OPerations on Catalogues And Tables Version 4.3-2* Available at: http://www.star.bris.ac.uk/ mbt/topcat/sun253/sun253.html. Cited on Mar 12, 2016

[13] Glenn Elert (2015) *The Nature of Light* Available at: http://physics.info/light/. Cited on Mar 12, 2016

[14] Australia Telescope National Facility (2016) *Types of Astronomical Spectra* Available at: http://www.atnf.csiro.au/outreach/education/senior/astrophysics/spectra_astro_types.html. Cited on Mar 12, 2016

[15] National Aeronautics and Space Administration (2013) *Light Curves and What They Can Tell Us* Available at: http://imagine.gsfc.nasa.gov/science/toolbox/timing1.html. Cited on Mar 12, 2016

[16] Thomas A. McGlynn (2014) *The FITS Support Office* Available at: http://fits.gsfc.nasa.gov/. Cited on Mar 16, 2016

[17] M. Donahue, T. Kimball (1997) *FITS File Format. In: HST Data Handbook.* Available at: http://www.stsci.edu/documents/dhb/web/c02_datafiles.fm2.html. Cited on Mar 16, 2016

[18] William D. Pence (2012) *FITS Extension Names.* Available at: http://fits.gsfc.nasa.gov/xtension.html. Cited on Mar 16, 2016

[19] Francois Ochsenbein, Roy Williams, Clive Davenhall, Markus Demleitner, Daniel Durand, Pierre Fernique, David Giaretta, Robert Hanisch, Tom McGlynn, Alex Szalay, Mark Taylor, Andreas Wicenec (2013) *VOTable Format Definition Version 1.3.* Available at: http://www.ivoa.net/documents/VOTable/. Cited on Mar 17, 2016

[20] Ivo Vondrák (2002) *Úvod do softwarového inženýrství.* [in Czech] Available at: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf. Cited on Mar 20, 2016

[21] Douglas Hughey. (2009) *The Traditional Waterfall Approach* Available at: http://www.umsl.edu/ hugheyd/is6840/waterfall.html. Cited on Mar 20, 2016

[22] Think Agile. (2016) *Software Development Methodologies* Available at: http://agilerules.blogspot.cz/2014/07/software-development-methodologies.html. Cited on Mar 20, 2016

[23] Martin Fowler. (2006) *Continuous Integration* Available at: http://www.martinfowler.com/articles/continuousIntegration.html. Cited on Mar 20, 2016

[24] Glen Stansberry. (2008) *7 Version Control Systems Reviewed* Available at: https://www.smashingmagazine.com/2008/09/the-top-7-open-source-version-control-systems/. Cited on Mar 20, 2016

[25] Center for Knowledge Management, Faculty of Electrical Engineering, Czech Technical University in Prague. (2011) *Základní informace – o co se jedná a k čemu to slouží* Available at: http://czm.fel.cvut.cz/vyuka/A4M33CPM/Download/DatoveKostky.pdf. Cited on Mar 25, 2016

[26] Howard Hamilton. (2012) *Data Cubes* Available at: http://www2.cs.uregina.ca/ dbd/cs831/notes/dcubes/dcubes.html. Cited on Mar 25, 2016

[27] tutorialspoint.com. (2016) *Data Warehousing - OLAP* Available at: http://www.tutorialspoint.com/dwh/dwh_olap.htm. Cited on Mar 25, 2016

[28] Oracle Corporation. (2002) *Data Warehousing Concepts* Available at: https://docs.oracle.com/cd/B10500_01/server.920/a96520/concept.htm. Cited on Mar 26, 2016

[29] tutorialspoint.com. (2016) *Data Warehousing - Schemas* Available at: http://www.tutorialspoint.com/dwh/dwh_schemas.htm. Cited on Mar 26, 2016

[30] Petr Škoda, Peter W. Draper, Margarida Castro Neves, David Andrešič and Tim Jenness. *Spectroscopic analysis in the virtual observatory environment with SPLAT-VO.* Astronomy and Computing. 2014, 7-8, 108-120 [cit. 2016-03-29]. DOI: 10.1016/j.ascom.2014.06.001. ISSN 22131337. Available at: http://linkinghub.elsevier.com/retrieve/pii/S2213133714000250 [online]

[31] Peter W. Draper. (2015) *Starlink SPLAT-VO.* Available at: http://star-www.dur.ac.uk/ pdraper/splat/splat-vo/. Cited 29 May 2016.

[32] Petr Šaloun, David Andrešič, Petr Škoda a Ivan Zelinka. *Upcoming Features of SPLAT-VO in Astroinformatics*. s. 475 [cit. 2016-03-30]. DOI: 10.1007/978-3-319-00542-3_47. Available at: http://link.springer.com/10.1007/978-3-319-00542-3_47 [online]

[33] Starlink. (2015) *Starlink.* Available at: http://starlink.eao.hawaii.edu/starlink. Cited 29 May 2016.

[34] Petr Šaloun, David Andrešič, Petr Škoda and Ivan Zelinka. *Better Spectra Manipulation in SPLAT-VO*. p. 373 [cit. 2016-04-01]. DOI: 10.1007/978-3-319-29504-6_36. Available at: http://link.springer.com/10.1007/978-3-319-29504-6_36 [online]

A SpecData Class Diagram

Figure 47: Complete SpecData and RemoteSpecData class diagram. Source: [34].

B SpecList Class Diagram

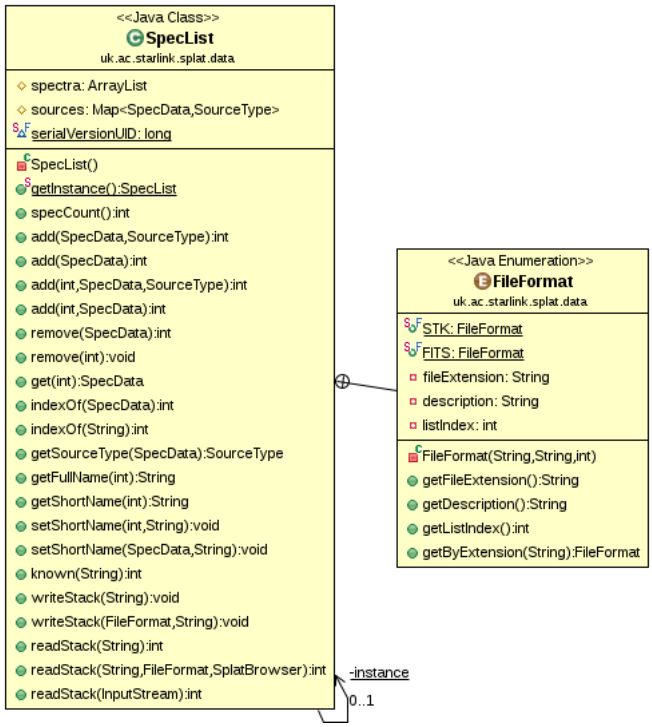


Figure 48: Complete SpecList class diagram. Source: [34].

C GlobalSpecPlotList Class Diagram

Figure 49: Complete GlobalSpecPlotList class diagram. Source: [34].

D SpectrumIO Class Diagram

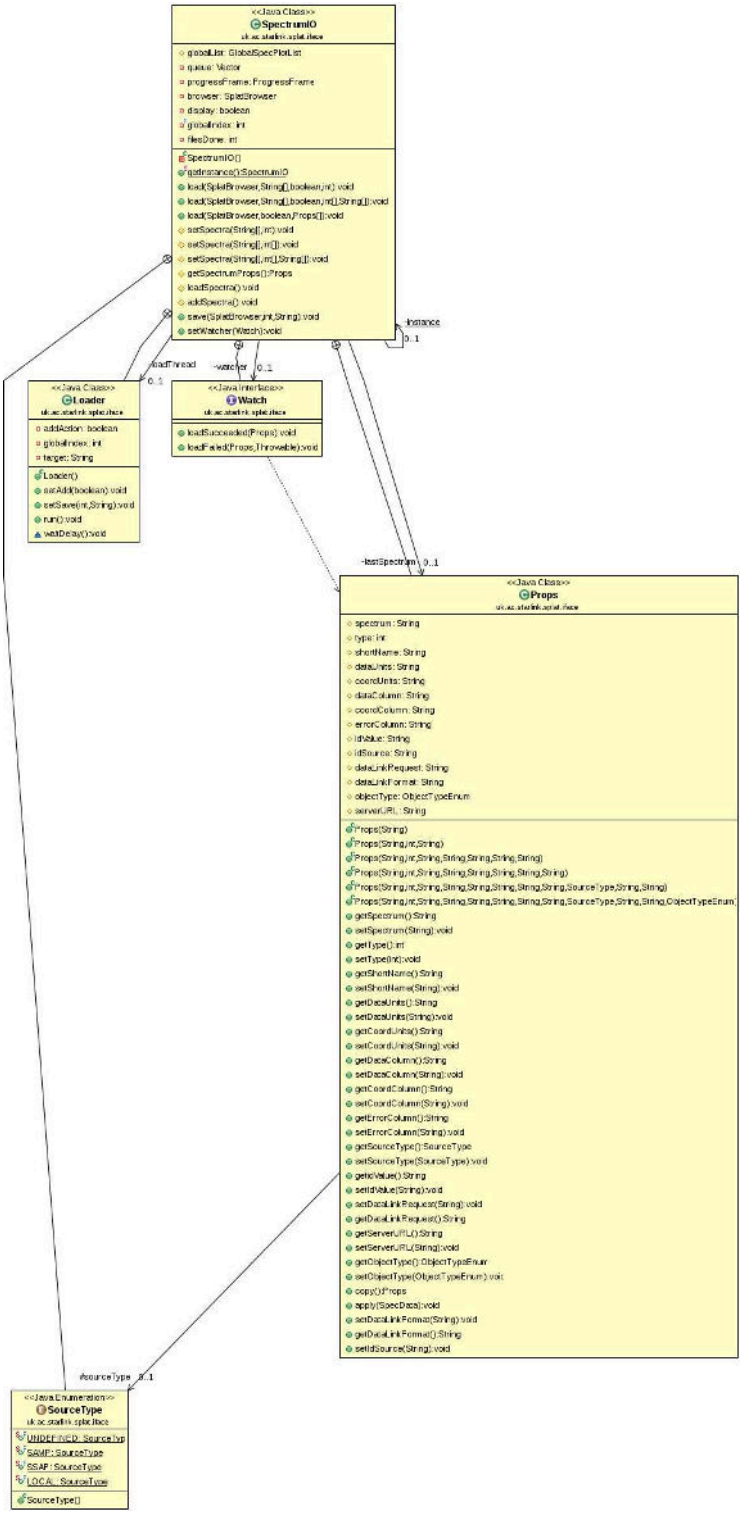


Figure 50: SpectrumIO and its dependencies class diagram.

E SpecDataFactory Class Diagram

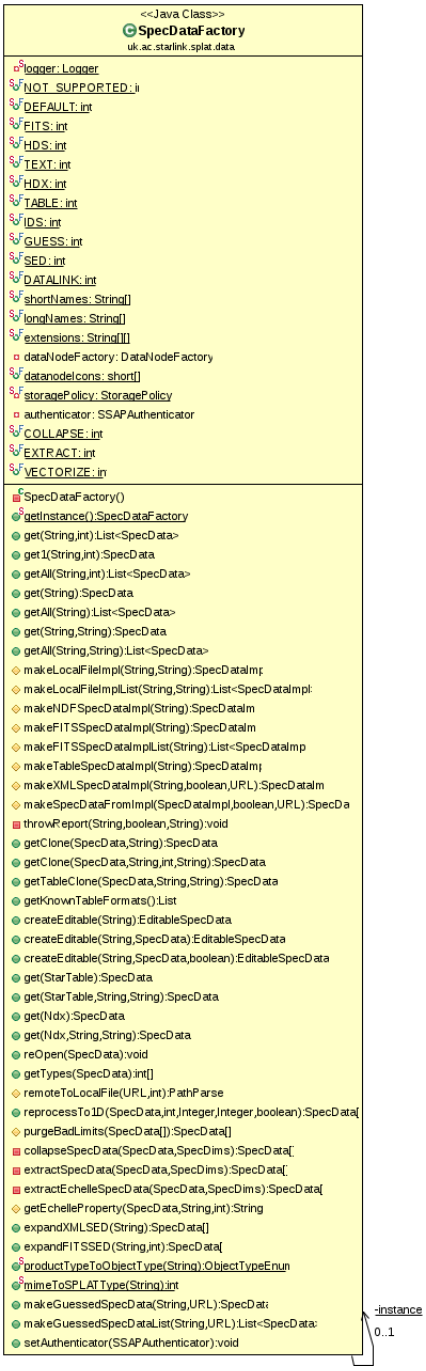


Figure 51: SpecDataFactory and its dependencies class diagram.

F Selected Diffs and Source Codes

This appendix section contains a few selected diffs with some interesting or crucial parts of SPLAT-VO improvements.

F.1 SSAP: Time Series Product Type Detection

```
 import nom.tam.fits.Header;
 import uk.ac.starlink.ast.FrameSet;
 import uk.ac.starlink.splat.util.SEDSplatException;
 import uk.ac.starlink.splat.util.SplatException;
 import uk.ac.starlink.splat.util.UnitUtilities;
@@ -558,15 +561,28 @@ public class TableSpecDataImpl
      * a series of vector cells describing the spectrum.
      */
     protected void readTable( long row )
         throws SplatException
     {
-        //  Access table columns and look for which to assign to the various
+        // Detect timeseries
+        for (Object oParam : starTable.getParameters()) {
+                if (oParam instanceof DescribedValue) {
+                        DescribedValue param = (DescribedValue) oParam;
+                        if (param.getInfo().getName().equals("ssa_producttype")) {
+                                if (param.getValue() != null && param.getValue().equals("timeseries")) {
+                                        setObjectType(ObjectTypeEnum.TIMESERIES);
+                                }
+                        }
+                }
+        }
+
```

Figure 52: Time series product type detection in VOTable.

F.2 Plot Window: Y-axis Flipping for Time Series

```
 import uk.ac.starlink.diva.FigureStore;
 import uk.ac.starlink.splat.ast.ASTJ;
 import uk.ac.starlink.splat.data.DataLimits;
+import uk.ac.starlink.splat.data.ObjectTypeEnum;
 import uk.ac.starlink.splat.data.SpecData;
 import uk.ac.starlink.splat.data.SpecDataComp;
 import uk.ac.starlink.splat.data.SpecDataFactory;
 import uk.ac.starlink.splat.util.SplatException;

@@ -734,10 +735,22 @@ public class DivaPlot
             }
             if ( ! dataLimits.isYAutoscaled() ) {
                 yMin = dataLimits.getYLower();
                 yMax = dataLimits.getYUpper();
             }
+
+            // inverse Y axis for timeseries
+            if (getSpecDataComp() != null) {
+               if (getSpecDataComp().get() != null) {
+                       for (SpecData specData : getSpecDataComp().get()) {
+                               if (ObjectTypeEnum.TIMESERIES.equals(specData.getObjectType())) {
+                                       dataLimits.setYFlipped(true);
+                                       break;
+                               }
+                       }
+               }
+            }
         }
     }
```

Figure 53: Y-axis flipping for time series in *Plot window*.

F.3 SAMP: Spectra as Tables Action Manager

```
/*
 * Copyright (C) 2009 Science and Technology Facilities Council
 *
 *  History :
 *     06-MAR-2009 (Mark Taylor):
 *        Original version.
 *     14-JUL-2009 (Peter Draper):
 *        Give up on 1D FITS and always transmit FITS tables.
 *     16-OCT-2009 (Peter Draper):
 *        Send SSA meta-data as required by HIPE (paul.balm@sciops.esi.int)
 *        More SSA 1.0 compatible.
 *     16-FEB-2016 (David Andresic):
 *        Send spectrum as table.
 */
package uk.ac.starlink.splat.util;

import java.awt.event.MouseEvent;
import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import javax.swing.JList;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import org.astrogrid.samp.Message;
import org.astrogrid.samp.gui.GuiHubConnector;
```

```
import uk.ac.starlink.splat.data.SpecData;
import uk.ac.starlink.splat.data.SpecDataFactory;
import uk.ac.starlink.splat.iface.GlobalSpecPlotList;
import uk.ac.starlink.splat.iface.SpectrumIO.Props;
import uk.ac.starlink.splat.vo.DataLinkParams;
import uk.ac.starlink.splat.vo.SSAQueryBrowser;
import uk.ac.starlink.util.URLUtils;

/**
 * Provides GUI actions for sending spectra by SAMP.
 *
 * @author Mark Taylor
 * @author David Andresic
 * @version $Id$
 */
public abstract class SpectraAsTablesSendActionManager
    extends SplatUniformCallActionManager
    implements EventEnabledTransmitter, ListSelectionListener
{
   protected static enum SOURCE_ENUM {
      JLIST,
      SSAP_BROWSER
   }

   private SOURCE_ENUM spectraSource;

   /**
    * Message type
    */
   private String mType;

   /**
    * Send type
    */
   private String sendType;

   /**
     * Global list of spectra.
     */
    private JList specList;

    /**
     * SSA Query Browser instance
     */
    private SSAQueryBrowser ssaBrowser;

    /**
     * Currently selected index in the global list of spectra.
     */
    private int selectedIndex = -1;

    /**
     * Map holding URL of each spectrum
     */
    private Map<SpecData, String> spectraUrls = new HashMap<SpecData, String>();

    /**
     * Constructor.
     *
     * @param specList global list of spectra
     * @param hubConnector controls connection with SAMP hub
     */
    public SpectraAsTablesSendActionManager( SSAQueryBrowser ssaBrowser,
                                             GuiHubConnector hubConnector, String mtype, String sendtype )
    {
        super( ssaBrowser, hubConnector, mtype,
               sendtype );
        this.ssaBrowser = ssaBrowser;
        this.spectraSource = SOURCE_ENUM.SSAP_BROWSER;
        updateSpecState();
        this.mType = mtype;
        this.sendType = sendtype;
    }
```

```
/**
 * Constructor .
 *
 * @param specList    global list of spectra
 * @param hubConnector    controls connection with SAMP hub
 */
public SpectraAsTablesSendActionManager( JList specList,
                                         GuiHubConnector hubConnector, String mtype, String sendtype )
{
    super( specList, hubConnector, mtype,
        sendtype );
    this.specList = specList;
    this.spectraSource = SOURCE_ENUM.JLIST;
    specList.addListSelectionListener( this );
    updateSpecState();
    this.mType = mtype;
    this.sendType = sendtype;
}

/**
 * Implement ListSelectionListener interface to ensure that this object
 * keeps track of the current selection state in the global spectrum list .
 */
public void valueChanged( ListSelectionEvent e ) {
    updateSpecState();
}

@Override
public void mouseClicked(MouseEvent arg0) {updateSpecState();}
@Override
public void mouseEntered(MouseEvent arg0) {}
@Override
public void mouseExited(MouseEvent arg0) {}
@Override
public void mousePressed(MouseEvent arg0) {}
@Override
public void mouseReleased(MouseEvent arg0) {}

/**
 * Invoked when the selection state of the global spectrum list
 * may have changed.
 */
protected void updateSpecState() {

  switch (getSpectraSource()) {
     case JLIST:
        int [] indices = specList.getSelectedIndices();
          selectedIndex = ( indices == null || indices.length != 1 )
                          ? -1
                          : indices[ 0 ];
          setEnabled( selectedIndex >= 0 );
        break;
     case SSAP_BROWSER:
        List<Props> props = ssaBrowser.getSpectraAsList(true);
        setEnabled(props != null && props.size() > 0);
        break;
     default:
        throw new IllegalStateException("Unsupported source type.");
  }


}

/**
 * Returns the currently-selected spectrum, if any.
 */
protected List<SpecData> getSpecData()
{
    if (getSpectraSource() == null) {
     throw new IllegalStateException("There is no spectra source defined for SAMP send action manager.");
    }

    switch (getSpectraSource()) {
     case JLIST:
        return Arrays.asList(GlobalSpecPlotList.getInstance().getSpectrum( selectedIndex ));
```

```
            case SSAP_BROWSER:
                return getSpectraFromSSAQueryBrowser();
            default:
                throw new IllegalStateException("Unsupported spectra source.");
            }

    }

    /**
     * Constructs and returns a message for transmitting load of the
     * currently selected spectrum.
     */
    protected abstract List<Message> createMessages()
        throws IOException, SplatException;

    /**
     * Returns a URL corresponding to an existing resource given by a
     * location string, if possible. If <code>loc</code> is an
     * <em>existing</em> file, a file-type URL is returned.
     * Otherwise, if <code>loc</code> can be parsed as a URL,
     * that is returned. Otherwise, <code>null</code> is returned.
     *
     * @param loc string pointing to resource (URL or filename)
     * @return URL describing <code>loc</code>, or null
     */
    protected static URL getUrl( String loc )
    {
        if ( loc == null ) {
            return null;
        }
        File locFile = new File( loc );
        if ( locFile.exists() ) {
            return URLUtils.makeFileURL( locFile );
        }
        else {
            try {
                return new URL( loc );
            }
            catch ( MalformedURLException e ) {
                return null;
            }
        }
    }

    public SOURCE_ENUM getSpectraSource() {
      return spectraSource;
    }

    /**
     * Extracts the currently selected spectra from SSA Query Browser
     *
     * @return
     */
    private List<SpecData> getSpectraFromSSAQueryBrowser() {
      List<SpecData> spectra = new LinkedList<SpecData>();
      List<Props> props = ssaBrowser.getSpectraAsList(true);
      spectraUrls.clear();

      // Inspired by SplatBrowser.tryAddSpectrum() and simplified

      if (props != null) {
        for (Props p : props) {
          System.out.println("and146: props url: " + p.getDataLinkRequest() + " / " + p.getidValue() + " / " + p.getShortName() + " /
                  " + p.getSpectrum());
          if ( p.getType() == SpecDataFactory.SED || p.getType() == SpecDataFactory.TABLE ) {
            try {
              List<SpecData> sp = Arrays.asList(SpecDataFactory.getInstance().expandXMLSED( p.getSpectrum() ));
              for (SpecData s : sp) {
                spectra.add(s);
                spectraUrls.put(s, p.getSpectrum());
              }
//                spectra.addAll(Arrays.asList(SpecDataFactory.getInstance().expandXMLSED( p.getSpectrum() )));
            } catch (SplatException e) {
              throw new RuntimeException("Unable to extract spectra from SSA Query Browser.", e);
            }
```

```
            } else {
                try {
                    if (p.getType() == SpecDataFactory.DATALINK) {
                        DataLinkParams dlparams = new DataLinkParams(p.getSpectrum());
                            p.setSpectrum(dlparams.getQueryAccessURL(0)); // get the accessURL for the first service read
                            String stype = null;
                            if (p.getDataLinkFormat() != null ) { // see if user has changed the output format
                                stype = p.getDataLinkFormat();
                                p.setType(SpecDataFactory.mimeToSPLATType(stype));
                                //props.setObjectType(SpecDataFactory.mimeToObjectType(stype));
                            }
                            else if ( dlparams.getQueryContentType(0) == null || dlparams.getQueryContentType(0).isEmpty()) //if not,
                                use contenttype
                                p.setType(SpecDataFactory.GUESS);
                            else {
                                stype = dlparams.getQueryContentType(0);
                                p.setType(SpecDataFactory.mimeToSPLATType(stype));
                                //props.setObjectType(SpecDataFactory.mimeToObjectType(stype));
                            }
                    }
                    List<SpecData> sp = SpecDataFactory.getInstance().get( p.getSpectrum(), p.getType() );
                    for (SpecData s : sp) {
                        spectra.add(s);
                        spectraUrls.put(s, p.getSpectrum());
                    }
//                  spectra.addAll(SpecDataFactory.getInstance().get( p.getSpectrum(), p.getType() )); ///!!! IF it's a list ???
                } catch (Exception e) {
                    throw new RuntimeException("Unable to extract spectra from SSA Query Browser.", e);
                }
            }
        }
    }

    return spectra;
  }

  /**
   * Returns URL of the given spectrum, if exists
   *
   * @param spec
   * @return
   */
  protected String getURLOfSpec(SpecData spec) {
    return spectraUrls.get(spec);
  }
}
```

F.4 SAMP: VOTable Send Action Manager

```
/*
 * Copyright (C) 2009 Science and Technology Facilities Council
 *
 *  History:
 *     06-MAR-2009 (Mark Taylor):
 *        Original version.
 *     14-JUL-2009 (Peter Draper):
 *        Give up on 1D FITS and always transmit FITS tables.
 *     16-OCT-2009 (Peter Draper):
 *        Send SSA meta-data as required by HIPE (paul.balm@sciops.esi.int)
 *        More SSA 1.0 compatible.
 *     16-FEB-2016 (David Andresic):
 *        Send spectrum as table.
 */
package uk.ac.starlink.splat.util;

import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
```

```
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.event.ListSelectionListener;

import org.astrogrid.samp.Message;
import org.astrogrid.samp.gui.GuiHubConnector;

import uk.ac.starlink.splat.data.SpecData;
import uk.ac.starlink.splat.data.SpecDataFactory;
import uk.ac.starlink.splat.vo.SSAQueryBrowser;
import uk.ac.starlink.util.URLUtils;

/**
 * Provides GUI actions for sending spectra contained inside VOTable by SAMP.
 *
 * @author Mark Taylor
 * @author David Andresic
 * @version $Id$
 */
public class VOTableSendActionManager
    extends SpectraAsTablesSendActionManager
{
    private static final String MTYPE = "table.load.votable";
    private static final String SENDTYPE = "table";


    /**
     * Constructor.
     *
     * @param ssaQueryBrowser SSA Query Browser instance
     * @param hubConnector controls connection with SAMP hub
     */
    public VOTableSendActionManager( SSAQueryBrowser ssaQueryBrowser,
                                     GuiHubConnector hubConnector )
    {
        super( ssaQueryBrowser, hubConnector, MTYPE,
               SENDTYPE );
        updateSpecState();
    }

    /**
     * Constructor.
     *
     * @param specList global list of spectra
     * @param hubConnector controls connection with SAMP hub
     */
    public VOTableSendActionManager( JList specList,
                                     GuiHubConnector hubConnector )
    {
        super( specList, hubConnector, MTYPE,
               SENDTYPE );
        specList.addListSelectionListener( this );
        updateSpecState();
    }

    /**
     * Constructs and returns a message for transmitting load of the
     * currently selected spectrum.
     */
    protected List<Message> createMessages()
        throws IOException, SplatException
    {
      List<Message> messages = new LinkedList<Message>();

      for (SpecData spec : getSpecData()) {
         String fmt = spec.getDataFormat();
            String mime = null;
            URL locUrl = null;
            File tmpFile = null;
            System.out.println("and146: specdata url: ");

            // See if we already have a VOTable spectrum ready to use.
            if ( "VOTable".equals( fmt ) || "TABLE".equals( fmt )) {
               if ( new File( spec.getFullName() ).exists() ) {
```

```
                mime = "application/x-votable+xml";
                locUrl = getURLOfSpec(spec) == null ? getUrl( spec.getFullName() ) : new URL(getURLOfSpec(spec));
            }

            else {
                tmpFile = File.createTempFile( "spec", ".vot");
                tmpFile.deleteOnExit();
//              locUrl = URLUtils.makeFileURL( tmpFile );
                locUrl = getURLOfSpec(spec) == null ? URLUtils.makeFileURL( tmpFile ) : new URL(getURLOfSpec(spec));
                mime = "application/x-votable+xml";
                spec = SpecDataFactory.getInstance()
                            .getTableClone( spec, tmpFile.toString(),
                                            "votable" );
                spec.save();
                  assert tmpFile.exists() : tmpFile;
            }
          } else {
            throw new SplatException("Invalid data format of the spectrum.");
          }

          assert mime != null;
          assert locUrl != null;

          // Prepare a metadata map describing the spectrum.
          // There should probably be more items in here.
          Map meta = new HashMap();
          meta.put( "Access.Reference", locUrl.toString() );
          meta.put( "Access.Format", mime );
          String shortName = spec.getShortName();
          if ( shortName != null && shortName.trim().length() > 0 ) {
              meta.put( "vox:image_title", shortName );
              meta.put( "Target.Name", shortName );
          }

          // Units.
          String dataUnits = spec.getDataUnits();
          String coordUnits = spec.getFrameSet().getUnit( 1 );
          if ( dataUnits != null && coordUnits != null ) {
              if ( ! coordUnits.equals( "" ) ) {
                  meta.put( "vox:spectrum_units",
                            coordUnits + " " + dataUnits );
                  meta.put( "Spectrum.Char.SpectralAxis.unit", coordUnits );
                  meta.put( "Spectrum.Char.FluxAxis.unit", dataUnits );
              }
          }

          // Columns.
          String xColName = spec.getXDataColumnName();
          String yColName = spec.getYDataColumnName();
          if ( xColName != null && yColName != null ) {
              meta.put( "vox:spectrum_axes", xColName + " " + yColName );
              meta.put( "Spectrum.Char.SpectralAxis.Name", xColName );
              meta.put( "Spectrum.Char.FluxAxis.Name", yColName );
          }

          // Prepare and return the actual message.
          Message msg = new Message( MTYPE );
          msg.addParam( "url", locUrl.toString() );
          msg.addParam( "meta", meta );
          if ( shortName != null && shortName.trim().length() > 0 ) {
              msg.addParam( "name", shortName );
          }
          System.out.println("and146: check #3");
          System.out.println("and146: URL: " + locUrl.toString());

          messages.add(msg);
      }

      return messages;
    }

    @Override
    public JMenu createSendMenu() {
      switch (getSpectraSource()) {
      case JLIST:
```

```
            return super.createSendMenu("Send spectrum as VOTable to...");
        case SSAP_BROWSER:
            return super.createSendMenu("Send result spectra as VOTable to...");
        default:
            throw new IllegalStateException("Unsupported source.");
        }
    }
}
```

F.5 JTable Utilities

```
package uk.ac.starlink.splat.util;

import java.util.logging.Logger;

import javax.swing.JOptionPane;
import javax.swing.JTable;

/**
 * Class of static members that provide utility functions for JTable.
 *
 * @author Andresic
 * @version $Id$
 *
 */
public class JTableUtilities {

    // Logger.
    private static Logger logger = Logger.getLogger( "uk.ac.starlink.splat.util.JTableUtilities" );

    /**
     * Class of static methods, so no construction.
     */
    private JTableUtilities()
    {
        // Do nothing.
    }

    /**
     * Gets the content of currently selected JTable cell as String
     * or returns null, if the selection is invalid.
     *
     * @param table
     * @return
     */
    public static String getCurrentCellContent(JTable table) {
        Utilities.checkObject(table, "Table must be set.");

        int row = table.getSelectedRow();
        int col = table.getSelectedColumn();

        boolean validSelection = row > -1 && col > -1;

        if (validSelection) {
            Object value = table.getValueAt(row, col);
            String strValue = value == null ? "" : value.toString();

            return strValue;
        } else {
            logger.warning("Invalid selection.");
            return null;
        }
    }

    /**
     * Gets all the content of the given JTable String
     * or returns null, if the selection is invalid.
     *
     * @param table
     * @param lineBreak
     * @param cellBreak
     * @return
```

```
 */
public static String getAllContent(JTable table, String lineBreak, String cellBreak) {
   Utilities.checkObject(table, "Table must be set.");
   Utilities.checkObject(lineBreak, "Line break must be set.");
   Utilities.checkObject(cellBreak, "Cell break must be set.");

  int numCols=table.getColumnCount();
    int numRows=table.getRowCount();
    int [] rowsSelected=Utilities.range(0,numRows);
    int [] colsSelected=Utilities.range(0,numCols);

    return getContent(table, lineBreak, cellBreak, numCols, numRows,
        rowsSelected, colsSelected);
}

/**
 * Gets all the content of the given JTable (including column names) as String
 * or returns null, if the selection is invalid.
 *
 * @param table
 * @param lineBreak
 * @param cellBreak
 * @return
 */
public static String getAllContentWithHeaders(JTable table, String lineBreak, String cellBreak) {
   Utilities.checkObject(table, "Table must be set.");
   Utilities.checkObject(lineBreak, "Line break must be set.");
   Utilities.checkObject(cellBreak, "Cell break must be set.");

  // get column names
  String headerColumns = getColumnNames(table, lineBreak, cellBreak);

  // get table data
  String content = getAllContent(table, lineBreak, cellBreak);

  return headerColumns + content;
}

/**
 * Gets the content of current JTable selection as String
 * or returns null, if the selection is invalid.
 *
 * @param table
 * @param lineBreak
 * @param cellBreak
 * @return
 */
public static String getCurrentSelectionContent(JTable table, String lineBreak, String cellBreak) {
  int numCols=table.getSelectedColumnCount();
    int numRows=table.getSelectedRowCount();
    int [] rowsSelected=table.getSelectedRows();
    int [] colsSelected=table.getSelectedColumns();

    return getContent(table, lineBreak, cellBreak, numCols, numRows,
        rowsSelected, colsSelected);
}

//

private static String getContent(JTable table, String lineBreak, String cellBreak,
     int columnCount, int rowCount, int[] selectedRowsCount, int[] selectedColumsCount) {

  if (columnCount > 0 && rowCount > 0) {
        StringBuffer value = new StringBuffer();

        for (int i=0; i<rowCount; i++) {
                for (int j=0; j<columnCount; j++) {
                 value.append(escapeContentBreaks(table.getValueAt(selectedRowsCount[i], selectedColumsCount[j]), lineBreak,
                        cellBreak));
                   if (j<columnCount-1) {
                     value.append(cellBreak);
                   }
                }
                value.append(lineBreak);
        }
```

```
            return value.toString();
          } else {
            logger.warning("Invalid selection.");
            return null;
          }
    }

    private static String getColumnNames(JTable table, String lineBreak, String cellBreak) {
      StringBuilder headerColumnsSB = new StringBuilder();
      for (int i = 0; i < table.getTableHeader().getColumnModel().getColumnCount(); i++) {
        Object headerColumn = table.getColumnName(i);
        headerColumnsSB.append(headerColumn == null ? "" : headerColumn.toString());

        if (i != table.getTableHeader().getColumnModel().getColumnCount() - 1) {
          headerColumnsSB.append(cellBreak);
        }
      }
      String headerColumns = headerColumnsSB.toString() + lineBreak;

      return headerColumns;
    }

    private static String escapeContentBreaks(Object cell, String lineBreak, String cellBreak) {
        return cell == null ? "" : cell.toString().replace(lineBreak, " ").replace(cellBreak, " ");
    }
}
```

F.6 SSA Query Results Selection Menu

```
package uk.ac.starlink.splat.vo;

import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;

import uk.ac.starlink.splat.util.JTableUtilities;
import uk.ac.starlink.splat.util.Utilities;
import uk.ac.starlink.table.gui.StarJTable;

/**
 * Popup menu for StarJTable wit SSA Query results.
 *
 * @author Andresic
 *
 */
public class SSAQueryResultsTableSelectionMenu extends JMenu {

    private static final long serialVersionUID = 1L;

    private static final String CELL_BREAK = "\t";
    private static final String LINE_BREAK = System.getProperty("line.separator");

    private static final String TITLE = "Selection";

//  private StarJTable starJTable;

    public SSAQueryResultsTableSelectionMenu() {
        super(TITLE);

        addMenuItems();
    }

    private void addMenuItems() {
        add(createCopyCurrentCellItem());
        add(createCopyCurrentSelectionItem());
        add(createCopyAllTableDataItem());
    }

    /**
     * Menu item for copying the currently selected cell content to clipboard.
     *
     * @return
     */
    private JMenuItem createCopyCurrentCellItem() {
        JMenuItem menuItem = new JMenuItem("Copy current cell to clipboard");

        menuItem.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                String content = JTableUtilities.getCurrentCellContent(getStarJTable(arg0));

                if (content != null) {
                    Utilities.addStringToClipboard(content);
                } else {
                    JOptionPane.showMessageDialog(getStarJTable(arg0), "Invalid selection. Please select some cell.");
                }
            }

        });

        return menuItem;
    }

    /**
     * Menu item for copying the all current selection content to clipboard.
```

```
     *
     * @return
     */
    private JMenuItem createCopyCurrentSelectionItem() {
        JMenuItem menuItem = new JMenuItem("Copy current selection to clipboard");

        menuItem.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                String content = JTableUtilities.getCurrentSelectionContent(getStarJTable(arg0), LINE_BREAK, CELL_BREAK);

                if (content != null) {
                    Utilities.addStringToClipboard(content);
                } else {
                    JOptionPane.showMessageDialog(getStarJTable(arg0), "Invalid selection. Please select some area.");
                }
            }

        });

        return menuItem;
    }

    /**
     * Menu item for copying the all current selection content to clipboard.
     *
     * @return
     */
    private JMenuItem createCopyAllTableDataItem() {
        JMenuItem menuItem = new JMenuItem("Copy all table data to clipboard");

        menuItem.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                String content = JTableUtilities.getAllContent(getStarJTable(arg0), LINE_BREAK, CELL_BREAK);

                if (content != null) {
                    Utilities.addStringToClipboard(content);
                }
            }

        });

        return menuItem;
    }

    protected StarJTable getStarJTable(ActionEvent e) {
        JMenuItem jmi = (JMenuItem) e.getSource();
        JPopupMenu jpm = (JPopupMenu) jmi.getParent();
        Component component = jpm.getInvoker();

        return traverseToStarJTable(component);
    }

    private StarJTable traverseToStarJTable(Component component) {
        System.out.println("and146: " + component);
        if (component == null) {
            return null;
        }

        if (component instanceof StarJTable) {
            return (StarJTable) component;

        } else {
            if (component instanceof JPopupMenu) {
                return traverseToStarJTable(((JPopupMenu)component).getInvoker());
            } else {
                return traverseToStarJTable(component.getParent());
            }
        }
    }
}
```

F.7 Spectrum Export to CSV and Text File

```
822 +
823 +   /**
824 +    * Inner class defining Action for saving the table data as CSV file.
825 +    */
826 +   protected class SaveAsCSVAction extends AbstractAction
827 +   {
828 +            private static final long serialVersionUID = 1L;
829 +
830 +            public SaveAsCSVAction( String name, Icon icon, String shortHelp )
831 +       {
832 +           super( name, icon  );
833 +           putValue( SHORT_DESCRIPTION, shortHelp );
834 +       }
835 +
836 +       /**
837 +        * Respond to actions from the buttons.
838 +        */
839 +       public void actionPerformed( ActionEvent ae )
840 +       {
841 +            writeAllTableDataToCsvFile();
842 +       }
843 +   }
844 +
845 +   /**
846 +    * Inner class defining Action for saving the table data as text file.
847 +    */
848 +   protected class SaveAsTextAction extends AbstractAction
849 +   {
850 +            private static final long serialVersionUID = 1L;
851 +
852 +            public SaveAsTextAction( String name, Icon icon, String shortHelp )
853 +       {
854 +           super( name, icon  );
855 +           putValue( SHORT_DESCRIPTION, shortHelp );
856 +       }
857 +
858 +       /**
859 +        * Respond to actions from the buttons.
860 +        */
861 +       public void actionPerformed( ActionEvent ae )
862 +       {
863 +            writeAllTableDataToTxtFile();
864 +       }
865 +   }
```

Figure 54: Spectrum export to CSV and text file - actions.

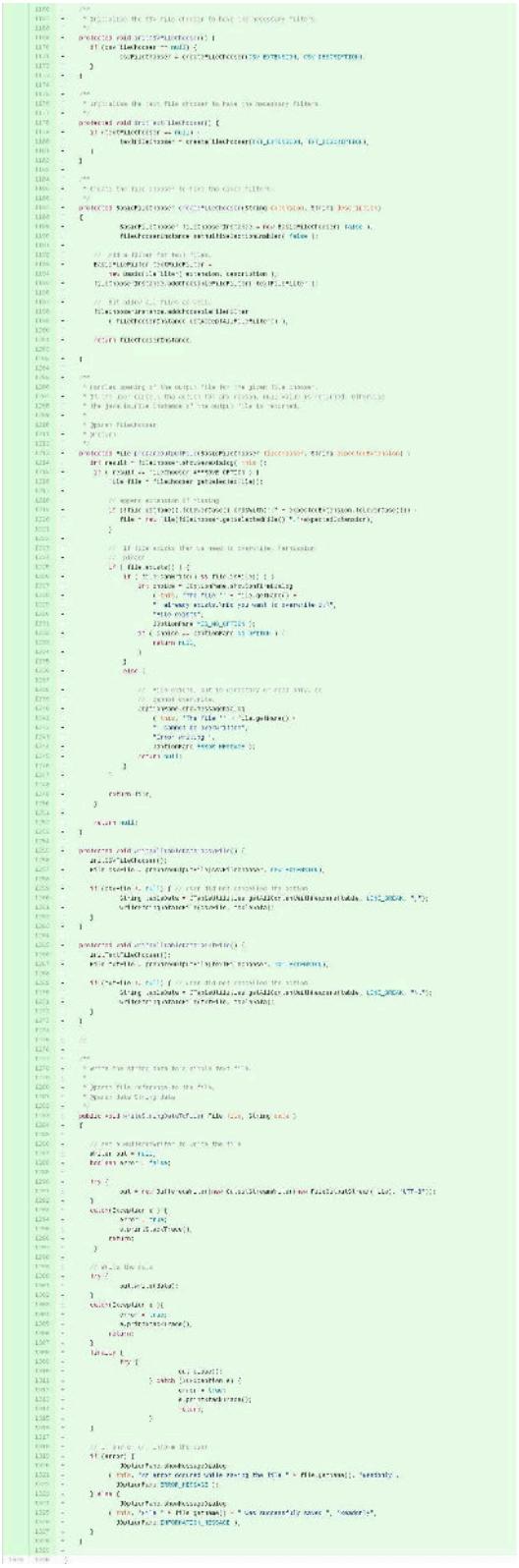


Figure 55: Spectrum export to CSV and text file - file choosers and writing methods.

F.8 Plot Control Key Listener

```
/**
 *
 */
package uk.ac.starlink.splat.plot;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

/**
 * PlotControlKeyListener listens for PlotControl's KeyEvents.
 *
 * @author David Andresic
 * @version $Id$
 */
public class PlotControlKeyListener implements KeyListener {

    private PlotControl plotControl;

    public PlotControlKeyListener(PlotControl plotControl) {
        this.plotControl = plotControl;
    }

    @Override
    public void keyPressed(KeyEvent e) {
        if (e != null) {
            switch (e.getKeyCode()) {
                case KeyEvent.VK_DELETE:
                    deleteSpectrum();
                    break;
                default:
                    // noop
                    break;
            }
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {
        // so far no operation
    }

    @Override
    public void keyTyped(KeyEvent e) {
        // so far no operation
    }

    /* Actual handlers */

    private void deleteSpectrum() {
        plotControl.removeCurrentSpectrumFromPlot();
    }
}
```

F.9 PlotControl: Remove Current Spectrum From Plot

```
2247 +    /**
2248 +     * Removes the currently selected spectrum from the plot
2249 +     */
2250 +    public void removeCurrentSpectrumFromPlot() {
2251 +              SpecData currentlySelectedSpectrum = (SpecData)nameList.getModel().getSelectedItem();
2252 +
2253 +              // message + global list checkbox
2254 +              String message = String.format("Do you really want to remove the spectrum '%s'?",
2255 +                      currentlySelectedSpectrum.getShortName()
2256 +                      );
2257 +
2258 +              JCheckBox checkBox = new JCheckBox("Remove from global list as well", true);
2259 +
2260 +              Object[] params = {message, checkBox};
2261 +
2262 +              // ask the user
2263 +              int n = JOptionPane.showConfirmDialog( this,
2264 +               params,
2265 +                       "Remove the spectrum",
2266 +                       JOptionPane.YES_NO_OPTION );
2267 +
2268 +              // return without taking action on 'No'
2269 +              if ( n == JOptionPane.NO_OPTION ) {
2270 +              return;
2271 +       }
2272 +
2273 +       // remove the spectrum from plot
2274 +       SpecDataComp specDataComp = getSpecDataComp();
2275 +       specDataComp.remove(currentlySelectedSpectrum);
2276 +       repaint();
2277 +
2278 +       // remove the spectrum from global list, if required
2279 +       if (checkBox.isSelected()) {
2280 +              globalList.removeSpectrum(currentlySelectedSpectrum);
2281 +       }
2282 +      }
```

Figure 56: PlotControl: Algorithm fot removing current spectrum from plot.

G Spectra Group VOTable example

The following VOTable contains a reference to 2 FITS files containing the actual spectra.

```
<?xml version="1.0"?>
<VOTABLE version="1.3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.ivoa.net/xml/VOTable/v1.3"
 xmlns:stc="http://www.ivoa.net/xml/STC/v1.30" >
  <RESOURCE name="splatVoSpectraGroup">
    <DESCRIPTION>SPLAT-VO (ver.) Spectra Group VO-Table</DESCRIPTION>
    <TABLE name="NORMbxn0727.fits">
      <FIELD name="WAVE" datatype="double" ucd="em.wl" unit="angstrom"/>
      <FIELD name="FLUX" datatype="double" ucd="phot.flux.density" unit="erg/cm**2/s/angstrom"/>
      <DATA>
        <FITS>
          <STREAM href="file:///path/to/spectrum/NORMbxn0727.fits"/>
        </FITS>
      </DATA>
    </TABLE>
    <TABLE name="NORMbxn0728.fits">
      <FIELD name="WAVE" datatype="double" ucd="em.wl" unit="angstrom"/>
      <FIELD name="FLUX" datatype="double" ucd="phot.flux.density" unit="erg/cm**2/s/angstrom"/>
      <DATA>
        <FITS>
          <STREAM href="file:///path/to/spectrum/NORMbxn0728.fits"/>
        </FITS>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```