Program pro interaktivní analýzu spektar v prostředí Virtuální observatoře

Programme for Interactive Spectra Analysis in Virtual Observatory Environment
VŠB - Technical University of Ostrava  
Faculty of Electrical Engineering and Computer Science  
Department of Computer Science

Bachelor Thesis Assignment

Student:  
David Andrešič

Study Programme:  
B2647 Information and Communication Technology

Study Branch:  
2612R025 Computer Science and Technology

Title:  
Program pro interaktivní analýzu spektrove v prostředí Virtuální observatoře  
Programme for Interactive Spectra Analysis in Virtual Observatory Environment

Description:  
Aim of this bachelor work is an extension of the current program SPLAT-VO by critical functions for standard analysis of stellar spectra in the Virtual observatory making any additions of new modules and implementation of subsequent changes to the GUI. 
Objectives:  
1st Introduction to SPLAT-VO program and its current capabilities.  
2nd Understanding the structure and relationships between modules - the study of the source code.  
3rd Basic introduction to the Virtual Observatory protocols - practice tests.  
4th Proposal for the implementation of new functionality required - by award consultant.  
5th Repairs identified errors.  
6th Testing, preparation of documentation changes and new features in the user manual.  
7th Delivery of complete documentation - even in the form of an addition to their own work.

References:  
http://www.aspbooks.org/a/volumes/table_of_contents/?book_id=420  
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: 16.11.2012

Date of submission: 07.05.2013

---

doc. Dr. Ing. Eduard Sojka  
**Head of Department**

prof. RNDr. Václav Snášel, CSc.  
**Dean of Faculty**
Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013

I declare that I worked out this bachelor thesis myself. I quoted all literary sources and publications that I used.

Ostrava, May 7, 2013
I would like to thank all people that helped me with this thesis, especially to Petr Škoda for his dedication, passion and patience and to Petr Šaloun for his attitude and having patience with me.

I would also like to thank people from SPLAT’s development team: Margarida Castro Neves, Markus Demleitner, Peter W. Draper and Mark Taylor for consulting and hints about SPLAT.

And finally, I would like to thank Ivan Zelinka for his assistance on a Nostradamus 2013 conference contribution that served as a base for this text.
Abstrakt

SPLAT je zkratkou pro Spectral Analysis Tool a je nástrojem pro zobrazování, modifikaci a analýzu astronomických spektrov, který byl vyvinut v roce 2003, původně jako součást projektu Starlink (a jeho podprojektu STARJAVA). Během svého vývoje byl rozšířen o podporu komunikace s prostředím Virtuální observatoře.

Dnes je stále jedním z nejlepších nástrojů pro spektrální analýzu, nicméně stejně jako každý jiný softwarový nástroj může být dále vylepšován a to jak v částech souvisejících s VO, tak v částech s VO nesouvisejících. Má rozšíření a opravy byly akceptovány komunitou rozvíjející jeho současné možnosti a budou součástí (pravděpodobně již) příští verze. Odborná veřejnost bude o mých výsledcích informována na konferenci Nostradamus, kam by můj článek přijat.

Klíčová slova: Virtuální observatoř, SPLAT-VO, spektrum, astronomie, astroinformatika, Java

Abstract

SPLAT is a shortcut for Spectral Analysis Tool and it is a tool for displaying, modifying and analysing astronomical spectra, originally developed in 2003 as a part of Starlink (and its STARJAVA package) project. During its development, the SPLAT was extended to include facilities that allowed an interoperability with the Virtual Observatory.

Today, it is still one of the greatest tools for spectra analysis, nevertheless, as every software tool, it can be still improved in both VO-related and non-VO-related ways. My adjustments and bugfixes were accepted by a community enhancing its current capabilities and will be a part of the (probably) next release. Expert community will be notified about my results at Nostradamus conference, where my contribution was accepted.

Keywords: Virtual observatory, SPLAT-VO, spectrum, astronomy, astroinformatics, Java
List of used shortcuts and symbols

ADQL  -  Astronomical Data Query Language
DNS   -  Domain Name Service
IVOA  -  International Virtual Observatory Alliance
REST  -  Representational state transfer
SAMP  -  Simple Application Messaging Protocol
SIAP  -  Simple Image Access Protocol
SPLAT -  Spectral Analysis Tool
SSAP  -  Simple Spectra Access Protocol
TAP   -  Table Access Protocol
UWS   -  Universal Worker Service Pattern
VO    -  Virtual observatory
# Contents

1 Introduction

2 Virtual observatory
   2.1 VO registry ............................................. 11
   2.2 Data Access Protocols .................................... 11
   2.3 Data Formats ............................................. 12
   2.4 Astronomical Data Query Language (ADQL) .............. 12
   2.5 Universal Worker Service Pattern (UWS) ............... 13
   2.6 VO-compatible applications ............................... 14

3 SPLAT-VO
   3.1 Historic background ...................................... 15
   3.2 User interface and features overview .................... 15
   3.3 SPLAT more technically .................................. 17

4 SPLAT’s new features and bug fixes
   4.1 Plot window .............................................. 19
   4.2 Global list of spectra .................................... 21
   4.3 Analysis menu ........................................... 23
   4.4 SAMP-compatible tools interoperability ................. 24
   4.5 Bug fixes ............................................... 26

5 Build features and HOWTO
   5.1 Java 1.7 compatibility ................................... 27
   5.2 Building using the buildscript .......................... 28

6 Summary ................................................................ 33

7 References ................................................................ 35

Appendices .................................................................. 36

A Selected Diffs ......................................................... 37
List of Tables

1  UWS job’s objects ......................................................... 13
List of Figures

1. Main window of SPLAT. .................................................. 16
2. Plot window in SPLAT. .................................................. 17
3. SSAP window in SPLAT. .................................................. 18
4. Visual spectra selection. .................................................. 20
5. Structure of java.awt.Color. .......................................... 22
6. New universal table for spectra selection. .......................... 24
7. Enabling the new *All SAMP spectra to the same window* feature. 25
8. Diff for commit *Fixed jsamp sign algorithm for Java 1.7* ...... 38
9. Diff for commit *Visual selection of spectra and their highlighting after selection*
   - *Modified binary search algorithm* ................................ 39
10. Diff for commit *Visual selection of spectra and their highlighting after selection*
    - *highlighting algorithm* .......................................... 40
11. Diff for commit *Visual selection of spectra and their highlighting after selection*
    - *Closest spectrum selection* .................................... 41
12. Diff for commit *Visual selection of spectra and their highlighting after selection*
    - *Invert color* .................................................. 42
13. Diff for commit *Stack to FITS (image) and all SAMP spectra to same windows*
    - *spectra's sources* ............................................... 42
14. Diff for commit *SpecCutter - apply on multiple spectra - SplatSpectraSelectionTable* .......................................... 43
15. Diff for commit *SpecCutter - apply on multiple spectra - SpecDataCellRenderer* ................................................. 44
List of Source codes

1. ADQL example. Source: [16] .................................................. 12
2. Building using the new buildscript ......................................... 29
1 Introduction

Facing the data avalanche of the last years, scientists needed to find a way how to be able to process all the incoming data effectively. This caused a revolution in science that involved a development of new computer infrastructures across multiple fields of science. In case of astronomy, we talk about the astroinformatics. Using the potential of large data archives, power of supercomputer grids and abilities of newly developed protocols and software, the modern astronomer is able to easily find the resources required. Work on such complex tasks, like obtaining several spectra of multiple objects in multiple wavelengths, or even combining them with other data and performing an extensive analysis using the supercomputers, then sharing the results with new communities across the world has never been easier.

A great example of this approach is the Virtual observatory. An infrastructure based on a (among others) server-client basis that allows to access the data and computer resources using specialized software and communication protocols. It also allows access or even participation of amateur scientists, which already proved its usefulness in several occasions.

SPLAT-VO that this work aims to is an example of such VO-enabled tool. It brings a capabilities of one of the greatest tools for spectra analysis to basically everyone. Nevertheless, as every software tool, the SPLAT-VO can be still improved in both VO-related and non-VO-related ways. Aim of this work is then to improve capabilities that doesn’t immediately relate to the VO and to fix some of the bugs or deficiencies.

In the near future, after dealing with coordination and unification of SPLAT’s development, we intend to continue in adding new features and implementing new VO standards so the SPLAT can be a feature-rich, user-friendly and even more portable scientific tool and reference implementation of new VO standards.
2 Virtual observatory

Virtual observatory (VO) can be described as a collection of astronomical archives and software tools that utilize the internet to allow international collaboration and scientific research [4].

Little more in detail, VO can be divided between client and server parts, where servers are the individual data archives, webservices or even (grids of) supercomputers performing specific tasks (generally called resources) and clients are “simple” software tools run and operated by astronomers. The communication among all of these components is based on a DNS-like resolving mechanism called VO resource registry that keeps the list of available resources. Clients are then accessing to these resources through specialized VO protocols specific for the required kind of data (e.g. spectra or tables) or action (e.g. inter-client communication or procedure call), where most of the protocols has its own set of metadata describing its subject.

All of the specifications that the VO is based on are defined by International Virtual Observatory Alliance (IVOA) - a worldwide organisation that aims to defining the technical standards that are needed to make all the astronomical datasets and resources a seamless whole [4].

2.1 VO registry

As mentioned before, VO registry is a DNS-like mechanism to keep and distribute/discover a list of available VO resources. Every resource that is supposed to be available in VO then has to be registered in one of there registries, represented by a XML record and identified by an unique URI beginning with ivo://.

Full specification of registry record’s format and discovery mechanism can be found in the IVOA’s official specification2.

2.2 Data Access Protocols

The data available in VO are searched and accessed via several specialized protocols. This section briefly describes the most important ones of them.

Cone Search Serves for searching tabular data within a given radius around a position on the sky.

Simple Image Access Protocol (SIAP) Is designed for retrieving images or their parts within or overlapping a region of sky.

Simple Spectra Access Protocol (SSAP) Is designed for retrieving spectra with given properties.

---

1http://en.wikipedia.org/wiki/Domain_Name_System
2http://www.ivoa.net/documents/RegistryInterface/
**Table Access Protocol (TAP)** For accessing general table data. Can be queried by different query languages, including ADQL (see 2.4). Since querying from many distributed servers can take a lot of time, TAP has also support for both synchronous and asynchronous queries via UWS (see 2.5).

The IVOA designs and specifies several more protocols available in VO (e.g. SAMP, that is more closely described in 4.4), but for this thesis (and as an introduction to a world of VO), the above mentioned should be more than enough.

### 2.3 Data Formats

Among others, the IVOA specifies a new data format primary used in VO.

#### 2.3.1 VOTable

The VOTable is a XML representation of tabular data, that also allows associations between other VOTables.

The key role in VOTable specification plays the **metadata** (defined as a vocabulary of *Unified Content Descriptors* - UCD), that are sent at the beginning of the communication and are then followed by the actual (hyperlinks to) content.

The VOTable is a standard VO format that can one day replace other formats like FITS, that are currently embedded or hyperlinked within it. Full specification is available at IVOA’s webpage³.

### 2.4 Astronomical Data Query Language (ADQL)

ADQL is a query language based on standard SQL92, but extended for purposes of astronomy and Virtual observatory. One of the most important reasons for its existence is limited support for positional queries of standard SQL. Joining tables in standard SQL (and RDBMS generally) requires a precise keys (e.g. any form of id), but in astronomy, for cross-matching catalogues it is required to join more “approximately” (nearby in time or position).

Let’s look at this example:

```sql
Select o.objId, o.ra, o.r, o.type, t.objId
from SDSS:PhotoPrimary o, TWOMASS:Photoprimary t
where xmatch(o,t) < 2.5
and Region('Circle;J2000.181.3,--0.76,6.5')
and o.type=3;
```

Listing 1: ADQL example. Source: [16]

³[http://www.ivoa.net/documents/VOTable/](http://www.ivoa.net/documents/VOTable/)
<table>
<thead>
<tr>
<th>Object</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>1</td>
</tr>
<tr>
<td>Owner</td>
<td>1</td>
</tr>
<tr>
<td>Completion time</td>
<td>1</td>
</tr>
<tr>
<td>Destruction time</td>
<td>1</td>
</tr>
<tr>
<td>Quote</td>
<td>0 - 1</td>
</tr>
<tr>
<td>Execution duration</td>
<td>0 - 1</td>
</tr>
<tr>
<td>Error</td>
<td>0 - 1</td>
</tr>
<tr>
<td>Results List</td>
<td>0 - 1</td>
</tr>
<tr>
<td>Parameters List</td>
<td>0 - 1</td>
</tr>
</tbody>
</table>

Table 1: UWS job's objects

In this example, we are cross-matching two catalogues (SDSS:PhotoPrimary and TWOMASS:Photoprima) using the `xmatch`. The `xmatch` takes two catalogues and calculates a probability that objects in these catalogues are the same. Then, it cross-matches on those, whose probability is (in this case) less than 2.5.

In the `where` clause, we are using the `Region` function that returns only those records that falls in this region of sky.

2.5 Universal Worker Service Pattern (UWS)

UWS is a REST⁴-based protocol and describes how to manage asynchronous execution of jobs on a service [11].

Every job consists of a set of objects that can be read or written in order to manage the job, as described in Tab. 1.

The job is treated as a state machine with these values of `Phase` object:

- QUEUED
- EXECUTING
- COMPLETED
- ERROR
- ABORTED
- SUSPENDED
- UNKNOWN
- PENDING

---

⁴https://en.wikipedia.org/wiki/Representational_state_transfer
Generally, UWS can be described as a subset of REST protocol. Its main purpose is to run a job on a service and periodically check its state (and change its behaviour if necessary) by calling a predefined URIs as defined by standard REST protocol. As a result, the long-time operation can run asynchronously on a service (the client doesn’t have to wait for it to end, just checks the state).

2.6 VO-compatible applications

Some of the below-listed applications were developed specifically for VO, some of them were self-contained applications or parts of larger projects. But there is something common for all of them: every application has its own scientific specialization and therefore implements only a subpart of (protocols of) VO and is generally written as multiplatform.

The list below is not a complete list of all existing VO applications, but gives to a reader a general overview of the most common desktop ones.

VOPlot Is a Java application for visualizing astronomical data. It has a support for VOTable, ASCII and FITS formats. It is also integrated with VizieR\(^5\) - a most complete database of published astronomical catalogues and data tables. [21].

TOPCAT A Java application for viewing and editing tabular data. It supports (among others) FITS and VOTable formats and can communicate with other tools using the SAMP protocol (see 4.4.1 for more details) [8].

Aladin A Java-based sky atlas for a visualization of digitized astronomical images and superimposing entries from astronomical catalogues or databases. It also integrates a support for Simbad\(^6\) and VizieR databases. [20]

VirGo A plugin for Stellarium\(^7\) planetarium that adds capabilities for browsing professional astronomical data. Unfortunately, it seems that it is no longer supported. [3]

SPLAT Spectral analysis tool. More detailed description is offered in 3.

VOSpec ESA's VO Spectral Analysis Tool written in Java. It can access the spectra, theoretical models and atomic and molecular line databases registered in the VO and offers many analysis features that can be used by a scientist on the data. [2]

Specview An another spectral visualization and analysis software written in Java, supporting many input and output formats. [17]

\(^5\)http://vizier.u-strasbg.fr/viz-bin/VizieR
\(^6\)http://simbad.u-strasbg.fr/simbad/
\(^7\)http://www.stellarium.org/
3 SPLAT-VO

SPLAT is a shortcut for Spectral Analysis Tool[12] and it is a tool for displaying, modifying and analysing astronomical spectra.

3.1 Historic background

SPLAT was developed in 2003 as a part of Starlink (and its STARJAVA package) project[12]. In 2005, the Starlink was closed down and in 2006 took over by Joint Astronomy Centre® that relicensed some of its parts under GNU/GPL licence[6]®.

During its development, the SPLAT was extended to include facilities that allowed an interoperability with the Virtual Observatory[12]. SPLAT itself uses the SSAP[10] (Simple Spectra Access Protocol) protocol for obtaining the spectra from SSAP servers and SAMP[11] (Simple Application Messaging Protocol) protocol to interoperate with other SAMP-compatible tools like TOPCAT®.

Since 2012, SPLAT-VO is being developed by the GAVO® (German Astrophysical Virtual Observatory) in cooperation with the Astronomical Institute of the Academy of Sciences of the Czech Republic[12]®. Its development is currently focused on implementing new VO standards. The need to extend the scientific features that doesn’t immediately relate to the VO and fix some of the bugs or deficiencies resulted in this bachelor thesis at VŠB-TUO® supervised by Petr Škoda of Astronomical Institute of the Academy of Sciences of the Czech Republic that scoped some (so far for time reasons just minor) changes to SPLAT-VO as described below.

In the near future, after dealing with coordination and unification of SPLAT’s development, we intend to continue in adding new features and implementing new VO standards so the SPLAT can be a feature-rich, user-friendly and even more portable scientific tool and reference implementation of new VO standards.

3.2 User interface and features overview

In the next few paragraphs, we will get familiar with the most basic SPLAT’s components, primarily in the sense of later-described modifications.

3.2.1 Most basic UI components

SPLAT’s user interface consists of several key components. Each one has its own special window. The most important one is the main window (Fig. 1). Basically every I/O

---

1. www.jach.hawaii.edu
2. www.gnu.org/licenses/gpl-3.0.txt
3. www.ivoa.net/Documents/SSA/
4. www.ivoa.net/Documents/latest/SAMP.html
5. www.star.bristol.ac.uk/~mbt/topcat/
6. www.g-vo.org
7. www.asu.cas.cz
8. VŠB - Technical University of Ostrava - www.vsb.cz
operation concerning spectra depends on this window, because it is an entry point for opening spectra of any source (local, SSAP, through SAMP settings etc.) and working with the global list of spectra. The global list of spectra is an application-global container for every spectrum that is worked with in SPLAT. It’s located on a main window’s left side and each spectrum that is contained and selected in it can be individualized for more plotting (the main window’s right side with spectrum-specific plot settings). Under a File menu, there is also an action for saving the whole global list of spectra to a file (more closely described in 4.2.1)

![Image of SPLAT window]

Figure 1: Main window of SPLAT.

Second most important component is the plot window. As the name says, its purpose is to plot the spectrum. Beside this, it is an entry point to SPLAT’s analysis features arsenal, which is available through a context menu at the top of the window. Middle section contains several components that affects the plot and/or its behaviour (it’s purpose is quite clear from a Fig. 2, interested readers can continue to [12] for a full documentation). The plot (or plots) itself is contained in the largest, bottom section.

The last component important in the sense of this thesis is the SSAP Query Window (Fig. 3). This window requires a certain minimal knowledge of a work of SSAP protocol and infrastructure of the Virtual observatory. It’s an entry point for querying the selected SSAP servers for a spectra of specified (or looked up) objects that can be then downloaded to a global list of spectra.

3.2.2 Other components

Beside these, SPLAT has many more components (for example, note that none of the analysis features is mentioned here), but these are the most basic and its knowledge is
essential to understanding the section 4. We will meet some of them in the rest of this thesis, the rest is described in detail in the official documentation, that can be found at [12].

3.3 SPLAT more technically

As a part of STARJAVA, SPLAT is primarily a desktop Java\(^6\) (originally v1.5) application that uses the Swing library for visual components. As a part of Starlink, its analysis features are based on a native, platform-dependent libraries originally used in the Starlink and written in Fortran, C, C++ etc.

The usage of platform-dependent libraries is treated by keeping its binaries for different platforms and architectures a part of standard distribution (the correct version is determined during the runtime). Keeping the native libraries limits SPLAT’s usage in some way (yet it runs on 32 and 64bit versions of Windows, Linux/*nix and Mac), but it also ensures the stability and correctness of the output since they are developed and used for so many years. Besides this, using native code can prove to be quite faster in some cases than in high-level Java.

Some modules also allows scripting in BeanShell\(^7\) and using Java EE. Apache Ant\(^8\) is used as a build system (building is more closely described in 5).

\(^6\)http://java.com/
\(^7\)http://www.beanshell.org/
\(^8\)http://ant.apache.org
Figure 3: SSAP window in SPLAT.
4 SPLAT's new features and bug fixes

As implied earlier, beside the strictly Virtual Observatory related changes, which will allow the user even more detailed searching for spectra, the upcoming version of SPLAT-VO focuses on the user experience, interoperability with other tools and work with the spectra itself. On the next few pages, features added as a part of this thesis will be described in detail.

4.1 Plot window

The purpose of plot window is quite simple: plotting every spectrum selected in the global list of spectra and allowing the user to perform analysis on it. Every plot window can contain one or more spectra and when working with SPLAT, the user can have opened many plot windows with many spectra plotted in it.

4.1.1 Added visual spectrum selection capability

So far, when the user worked with multiple spectra in one common plot window, he has had a limited possibility of selection just one of them for further actions. Selection of eg. noisy spectrum and its deletion from plot window could be done only by trial-and-error procedure and therefore quite problematic and time-wasting.

In the upcoming version, the user can click by mouse inside the plot window and SPLAT will automatically select the closest spectrum. The spectrum gets selected in the local spectra drop-down menu above the plot itself, and in the global list of spectra in the main window. From there, it can be easily deleted or modified for plotting and analysis.

The actual selection is done by getting the \([x, y]\) coordinates of mouse click (Fig. 4: 1), that are then transformed to coordinates with the plot's \([0, 0]\) as beginning. Then the algorithm searches for all spectra with \(x\) coordinates containing the \(x\) coordinate of the mouse click (Fig. 4: 2 - spectrum \(a\) is out). For this, the native (platform dependent) Ast library from the original Starlink project is used. Each spectrum is represented by series of \([x, y]\) coordinates, each coordinate represents one value. The Ast library transforms this discrete list of values to a continuous line that we see in the plot window and it's this set of transformed coordinates, that the selection algorithm uses. So when any spectrum detected on the \(x\) coordinate, the algorithm will again use the modified binary search algorithm\(^{19}\) to select the closest spectrum on the \(y\) axis (Fig. 4: 3 - spectrum \(c\) is out, \(b\) is closest). This is quite similar to the procedure on \(x\) axis, the Ast-transformed coordinates are used to locate the closest match of each \(x\)-colliding spectrum's \([x, y]\) coordinate to the mouse click coordinates.

\(^{19}\)The usual binary search algorithm searches for the exact match of searched item in the sorted list by getting closer to it by dividing the current interval (where the first interval is \(0...n\)) by two. Then, by comparing the searched value to the subinterval's borders, it divides the first or second subinterval in the next iteration. The modified algorithm doesn't look for the exact match, but gets as close as possible to the searched item and then compares the first item on left side with first item on the right side and selects the closest to the searched one.
Figure 4: Visual spectra selection.
The selection of the spectrum is quite reliable and for ensuring this, the selected spectrum gets immediately highlighted by few-seconds blinking in inverted color (as more closely described in 4.2.2).

The usage of modified binary search algorithm is crucial when considering the user’s computer performance so it won’t cause any slow-down of SPLAT’s interface.

A fragment of selection’s source code can be found in Appendix A, Fig. 11.

A fragment of modified binary search’s source code can be found in Appendix A, Fig. 9.

4.2 Global list of spectra

The global list of spectra is a part of the SPLAT’s main window. It is a container (JList) for every spectrum opened no matter from which source. Since it is a source of each spectrum in SPLAT (works sort of as a proxy pattern, no matter what the source is, or in what format the spectrum is, each one can be used transparently and in the same way since it is transformed to an internal SpecData object), every major component of SPLAT is tightened to it.

4.2.1 Saving in FITS format

Previous versions of SPLAT were able to save its global list of spectra only to their native binary format (more technically, it’s just a serialized object representing the JList with individual spectra’s instances). But sometimes, there may be a need to save that list to a more universal, standardized format. For example, SPLAT allows to perform many operations on spectra, which results in a new spectrum object that is added to a global list of spectra (eg. cutting a part of spectrum). Saving the list of such spectra to a standardized format would then allow its opening in another tool and continuing to work with it in a way the SPLAT wasn’t designed to.

Therefore, the upcoming version of SPLAT-VO will allow the user to save the global list of spectra to a universal FITS (Flexible Image Transport System) format. The FITS format is multipurpose format for storing the scientific data, primarily designed and used for astronomy[7]. The FITS allows storing multiple data of multiple type in it, for example, it can contain the image of a star and its spectrum (or spectra) with corresponding metadata. This is done by a concept of extensions. For a programmer, the extension can be described as an instance of a class, where class itself is a type of extension. We recognize these basic types of extensions[1]:

**IMAGE** Provides a means of storing a multidimensional array similar to that of the FITS primary header and data unit.

**TABLE** ASCII table extension type contains rows and columns of data entries expressed as ASCII characters.

**BINTABLE** Provides a more flexible and efficient means of storing data structures than is provided by the TABLE extension type.
Every FITS file consists of one primary Header Data Unit (HDU) and of an unlimited number of extension HDUs. Every HDU contains (beside its actual data) the ASCII header with corresponding metadata as key/value pairs, where the most important keys are standardized and understood by FITS readers (e.g. EXTENSION key identifying the type of the current extension).

In the upcoming version of SPLAT, every spectrum will be represented by its own IMAGE extension. So when having, let's say five spectra in the global list, it will result in a FITS output with five IMAGE extensions (one for each spectrum). The only thing that the user will have to do for this, is selecting the newly added FITS/IMAGE format in a dialog window for saving the global list of spectra, instead of the previous proprietary format.

It should be said, that in the current state the SPLAT cannot open the global list of spectra saved in FITS/IMAGE format, but since the cruciality of saving to more universal formats, it will be added very soon. We are also planning to add the support for another types of FITS extensions (e.g. BINTABLE), as well as other formats, like VO-Table.

4.2.2 Highlighting the spectrum after selection

When user clicked on a spectrum in global spectra list and this spectrum was plotted to a window(s) with multiple spectra, he had no way of knowing, which one of them it is.

In the upcoming version, when user clicks on a spectrum in global spectra list, the spectrum gets immediately highlighted in all plot windows that it is opened in. The highlighting has a form of a few-seconds blinking in an inverted color and is performed sequentially (highlighting in one plot window comes after the previous one is finished).

The algorithm used to do this is the same as in the case of visual spectra selection as briefly described in 4.1.1 and designed to not cause any slow-down of SPLAT's interface. This is done by listening the mouse-click event and performing all the iterations of inverting the spectrum's plot color, repainting the components and sleeping itself in special threads, so SPLAT's user interface doesn't get freezeed during the sleep phase and is not touched otherwise. The highlighting threads are also synchronized, so one starts highlighting after the previous is finished and the user gets a perfect sequential overview of the spectrum's plots.

A fragment of highlighting's source code can be found in Appendix A, Fig. 10.

As for the inversion, the spectrum's color is treated as java.awt.Color in its integer expression. Basic colors - red (R), green (G) and blue (B) - are encoded in a way described in Figure 5.
For inverting the color, we need to decode each basic color (this can be done by class's getters), invert it by subtracting this decoded value from 255 (range for each color) and putting it back to its decade by multiplying it by the corresponding offset (1 for blue, 256 for green and 65536 for red). The final inverted color is then a sum of these inverted components.

A fragment of inversion's source code can be found in Appendix A, Fig. 12.

### 4.3 Analysis menu

The plot window's Analysis menu is a doorway to SPLAT's arsenal of analytic features like cutting regions from spectra, fitting, interpolation, filtering and many more. Many of these functions are still based on native, platform dependent libraries of the original Starlink project.

#### 4.3.1 Cut window: saving the ranges

In the 'Cut regions from spectrum' window, user could simply read ranges of regions from a local file (feature available under 'File' menu). But what if he made a visual selection of ranges from a currently plotted spectrum and wanted to save it to a local file?

Since the upcoming version, SPLAT can save the currently defined ranges to a local file (feature is available under 'File' menu) in the same format as the reading feature expects, that is one range (with a whitespace between its beginning and ending value) per one line:

```plaintext
# Generated by SPLAT-VO

# Range 1
8601.177 8621.531

# Range 2
8729.026 8748.744
```

Adding this feature was the simplest of all, since SPLAT already possessed this feature, just was using it to a different purpose. Thanks to this, it consisted only from adding the menu item and binding the existing function to it.

#### 4.3.2 Cut window: performing on multiple spectra

And cutting regions from spectrum again: until now, when user defined all the wanted regions and performed the cut action, SPLAT cutted the regions only from the currently active spectrum.

In the upcoming version, this window will possess a table of all currently plotted spectra in the corresponding plot window. User then can select multiple of it (buttons for selecting and deselecting all the spectra are matter of course) and perform a cut (or any other action in this window) on all the selected spectra.
Adding this feature was quite straightforward. It consisted from creating a new class extending the standard JTable, its appropriate cell-render class and passing it a JList with all currently plotted spectra. The JList’s items then get rendered to this table. One of the new methods returns a list of all selected spectra, which is only a step from modifying the action functions (cut etc.) to iterate over it.

This new spectra selection table is written very generally so adding a similar feature anytime in the future will then be quite easy.

Fragments of table and its renderer can be found in Appendix A, Fig. 15 and 14.

4.4 SAMP-compatible tools interoperability

Simple Application Messaging Protocol (SAMP) is an universal, XML-based, event-driven messaging protocol for exchanging control and data information and thus allows the (not just astronomical) software to interoperat[9].

Usage of SAMP allows building an event-driven publish/subscribe messaging system with both synchronous and asynchronous communication. The whole communication is centralized, all elements communicates with a SAMP hub that takes care of routing the messages[9]. In the typical scenario, an application interested in using the SAMP looks for a SAMP hub using the appropriate discovery mechanism (the hub itself - using the same mechanism - guards that there is only one SAMP hub running), registers with it and sends to it metadata containing its name, type of messages that it’s interested in and asks the hub for information about other clients. Then, when some hub client (publisher) has a message to publish, it will notify the hub that will decide (based on type of the mes-
sage) which registered clients (subscribers) will be notified as well. When a subscriber is notified, it can ask the hub for the message, the hub will then require the message from the publisher and sends it to a subscriber.

For having a better idea about SAMP’s purpose and capabilities, imagine an (e-)conference (or any other form of real-time cooperation) where one user performs an action in his client (e.g. clicking on a spectrum in TOPCAT) and other users in their completely different clients see the same information but in a form native for their clients (e.g. spectrum sent by TOPCAT plotted in SPLAT) in the same time.

4.4.1 All SAMP spectra to the same plot window

In the previous versions of SPLAT, all spectra received via the SAMP protocol got opened in their own plot windows. Very often, it would be useful or even necessary for analysis to have them plotted in the same window, but for this, the user had to do it manually.

As stays in the title, the upcoming version will be able to do this for the user automatically. We added new checkbox under the ‘Interoperability’ menu that can be used to switch between the current state (each spectrum to its own plot window) and plotting all the spectra received from SAMP to their common plot window.

Technically it is done by adding the new ‘Source type’ attribute into the spectrum’s I/O process and mapping every plot window to its initial spectrum’s source type (in case of SAMP, this reference is removed from every possibly opened ‘SAMP-containing’ plot window when user disables this feature, so it allows to plot one bulk of spectra received from SAMP to one window and another bulk to another window). Then, when the new spectrum is received (in this case from SAMP, but this may vary in the future, read more) and the feature is switched on, SPLAT will first look for an existing plot window that is mapped to SAMP source type and if found, it will plot the spectrum into it (and properly resizes it as usual).

As said, this feature was added using the new attribute for spectrum’s I/O process defining the spectrum’s source. So far this attribute is bind just to SAMP protocol, but it allows simple modifications of I/O behaviour based on the spectrum’s source in the future by simple writing the extra logic for non-default source type.
4.5 Bug fixes

Since the SPLAT is based on Starlink’s libraries, which has been in use for many years, discovery of new bugs is limited to less critical parts of SPLAT. Yet, for some users, they make the SPLAT unusable. We are now in the process of removing those we know about, but we of course welcome new bug reports.

4.5.1 SSAP Query window

In some cases, when user tried to open the SSAP Query window to search for spectra via SSAP protocol, the only thing, that he saw was a window with a list of queried SSAP servers. He couldn’t enter any values or parameters.

This was caused by a NullPointerException in SSAP Query window initialization. After hotfixing it and consulting with Margarida Castro Neves of Heidelberg University, the problem was located in a PropertyChangeSupport property initialization. Since the JPanel already has it, it was fixed by removing it and refactoring the rest of the code to use the JPanel’s one.
5 Build features and HOWTO

It will still take some time before the new version will be released, but if anyone would like to try it now or participate on testing, here is the essential build20 'HOWTO' and short description of its innovations.

5.1 Java 1.7 compatibility

By Java 1.7 compatibility is meant its building compatibility (runtime compatibility is generally guaranteed by Java itself). Until now, Java 1.6 has been used to build SPLAT. Building with 1.7 was causing build failures for several reasons described below.

Still, building with Java 1.7 keeps SPLAT compatible with Java 1.6 runtime environment.

5.1.1 JSAMP's jar sign algorithm

When built with 1.7, SPLAT was throwing java.lang.SecurityException: invalid SHA1 signature file digest exception during its start. The reason is the changed sign algorithm for jars in Java 1.7. This problem can be fixed by adding the digestalg="SHA1" parameter to jarsigner, which will force using the previous sign algorithm.

Since STARJAVA/SPLAT is being built by the Ant utility, the digestalg parameter should come to the JSAMP's build.xml file. It also requires modification of STARJAVA/SPLAT's customized Ant (added some STARJAVA-special Ant targets) in the sense of adding the support for these new parameters to its source, since they are available from higher versions of Ant that STARJAVA contains.

A fragment of this modification can be found in Appendix A, Fig. 8.

5.1.2 Java 1.6 backward compatibility

By default, SPLAT's binaries are runnable under a version of Java that has been used to compile it. Keeping the backward compatibility therefore required adding the target parameter for javac in build.xml file of SPLAT and all of its required tools.

5.1.3 Other modifications

This section closely relates to 5.2 and usage of a build script. By default, this script builds the entire STARJAVA package so when javac 1.7 is used, it throws errors in other tools as well (yet some of them are not required to successfully build the SPLAT). This is a short list of modifications required to successfully use the javac 1.7, yet none of them is so far a part of its original distribution.

20As said before, our current primary goal is to unify the development since the SPLAT's source codes are located in several repositories. Version described in this article can be found at Github: https://github.com/and146/SPLAT-ARI/
As first, Frog was using a Sun’s proprietary (not Java-standard) library for exporting the image (JPEGCodec and JPEGImageEncoder), which was removed in Java 1.7. This needed to be rewritten to use its standard equivalent (ImageIO).

Then, TOPCAT is using in SyntheticColumnQueryWindow class getters and setters for a parameter called type. Unfortunately, this class extends the Java’s java.awt.Window which since 1.7 contains an internal enum called Type. This was causing the incompatible return data type of

```java
uk.ac.starlink.topcat.SyntheticColumnQueryWindow.getType()
```

during the build with javac 1.7.

And finally, on some systems the build could fail on invalid encoding of TAMFITS’s source. This was fixed by explicit adding the encoding="iso-8859-1" parameter to its build.xml file.

### 5.2 Building using the buildscript

Basically, until now the only way how to build SPLAT was (at least in some way) following the instructions in the original STARJAVA’s README file. To automatize this (and to add some more ‘shortcuts’), the build script was created.

The build script (_builder.sh) can be used on every Linux system and beside the automation of STARJAVA’s build process with some system-checks included it also allows quite user-friendly way to build only a subpart of STARJAVA or to use a specific version of Java (for full list of its capabilities, run it with the --help parameter).

#### 5.2.1 Source

As said before, our current primary goal is to unify the development since the SPLAT’s source codes are located in several repositories. Version described in this article can be found at Github: https://github.com/and146/SPLAT-ARI/.

#### 5.2.2 Prerequisites

Checks for most of them are included in the build script, so just shortly here:

- **Java Advanced Imaging (JAI)**
  (http://download.java.net/media/jai/builds/release/1_1_3/INSTALL.html)

- **Java Development Kit (JDK) >= 1.6**
  (http://www.oracle.com/technetwork/java/javase/downloads/index.html)

- **Git**
  (http://git-scm.com/)
5.2.3 Build

The actual build is quite straightforward, so again, very shortly here (demonstrated on GNU/Linux):

```bash
# first, download the source
(starjava.parent) $ git clone git://github.com/and146/SPLAT-ARI/

# enter the newly created directory and run the build script
(starjava.parent) $ cd SPLAT-ARI
(SPLAT-ARI) $ ./builder.sh

# the binaries should be located in (starjava.parent)/bin:
(SPLAT-ARI) $ ./bin/bin/splat
```

Listing 2: Building using the new buildscript.

5.2.4 Creating an installation package

This section describes how to create an installation package of SPLAT-VO. It's based on Peter Draper's HOWTO as well as Margarida Castro Neves's modifications of it.

Logo, some scripts and some other useful files are contained in a file called `extra_files.tar'. This is a file created by Peter W. Draper and is generally not public. Yet, these files shouldn't be necessary unless you want to create an uniform, 1:1 installer package.

This HOWTO describes creating a package on a GNU/Linux OS.

1. Prerequisites

   (a) **Starjava's source codes**

   Since some utilities from the original Starjava distribution are required, the first step is getting the source code of the original Starjava, containing the SPLAT-VO. We can get it and build it using the build script as described previously in 5.2 (if we intend to create a package with modified version) or we can download and build the currently official source codes from [http://starlink.jach.hawaii.edu/starlink/](http://starlink.jach.hawaii.edu/starlink/) and proceed as described in its README file.

   It should be noted, that this will change very soon (probably right after finishing this thesis), because of merging and unification of the source code and its relocation to a Git repository. The only difference then will be a different URL with the source codes and the way how to build it.

   (b) you should have installed the **C-shell interpreter** (`csh` or `tcsh`\(^2\))

2. **IzPack**

   For actual creating the package, the IzPack\(^2\) is used. IzPack is a famous tool for creating a platform-independent installer packages on Java platform.


\(^2\)[http://izpack.org/](http://izpack.org/)
(a) Run IzPack’s installer

```
$ java -jar izPack-install-4.3.5.jar
```

(b) After the installation, you should add the IzPack’s bin directory to the $PATH, or make the necessary symbolic links:

```
# ln -s /opt/IzPack/bin/compile /usr/bin/compile
```

3. extra_files.tar

   Extract the extra_files.tar scripts to the parent of (starjava) directory

4. Prepare the environment

   (a) Prepare the source codes in (starjava) directory (clean, checkout from svn etc.)

   (b) Run this script from extra files tarball:

   ```
   (starjava, parent) $ source setup
   ```

   This script prepares the environment for building SPLAT. Please note, that all the paths in the script should be adapted to your system.

   (c) As an alternative, this should be enough at least for already built binaries (see 1):

   ```
   java_binaries=$(echo 'which is java' | tr ""n"")
   for b in $java_binaries
   do
     STAR_JAVA="$b"
     break
   done
   echo $STAR_JAVA
   ```

5. Installing only the SPLAT relevant stuff

   (a) If you so far didn’t build the Ant\textsuperscript{23} and the rest of the sources, now would be a good time

   (b) Run this:

   ```
   (starjava) $ ./scripts/targetdeps splat install
   ```

   (c) Let’s test it:

   ```
   (starjava) $ cd ..
   (starjava, parent) $ ./bin/splat/splat
   ```

\textsuperscript{23}http://ant.apache.org/
6. **Update documentation etc.**
   
   (a) generate the `sun243.ps` file
   
   (b) update the `splat/src/docs/splat.news`
   
   (c) update help files if necessary

7. **Remove all unnecessary files**
   
   (starjava_parent) $ ./removed_files.lis

8. **Finalization before creating the installer**
   
   (a) Update release version in `install.xml` (which should be copied to the release directory)
   
   (b) make sure all scripts have execute permission

9. **Build installer JAR file**
   
   By running this command:
   
   (starjava_parent) $ ./doit.csh

   For this IzPack had to be installed, and it’s `bin` directory needs to be in the `$PATH` (to execute the `compile command`). Some files are also needed:

   - `Unix_shortcutSpec.xml`
   - `shortcutSpec.xml`
   - `Starlink.gif` (should also be copied to the release directory)

   They are all part of `extra_files.tar`.

   And that’s it! You have successfully created an installer package of SPLAT-VO. Testing the release and finalization should come next, as well as some other publishing procedures if you are a part of a developer team.
6 Summary

Being more focused on user experience should allow to current users to work with SPLAT more effectively while for more reserved astronomers that so far had a reason to avoid SPLAT, we hope we can provide a reason or reasons to give it a try.

My results were accepted for Nostradamus 2013 conference and will be published in conference proceedings book, see [13].

In the near future, we are planning to work on (beside strictly VO-related parts, that are currently being developed at Heidelberg University) user-defined line lists, working with spectrum’s header information in plot window or supporting more file formats and at least some of these changes should be a part of the next release as well.

The nearest goal anyway will be the unification and migration of the sources from SVN to a Git (Github) repository.

David Andrešič
7 References


A Selected Diffs

This section contains diffs of selected modifications that were done as a subject of this thesis.

Full versions can be found at: https://github.com/and146/SPLAT-ARI/.

Please note that these diffs are for visual reasons in a form of an image. The readability is therefore primarily ensured for a computer viewer, rather than for a printed hardcopy.
protected String tsacert;  
/* signature algorithm */  
private String sigAlg;  
/* */  
private String digestAlg;  
/* */  
/* error string for unit test verification: */  
private static final String ERROR_TOOL_AND_SIGNEDJAR =  
public void setSigAlg(String sigAlg) {  
this.sigAlg = sigAlg;  
}  
/* */  
public String getSigAlg() {  
return sigAlg;  
}  
/* */  
/* Digest Algorithm: optional */  
public void setDigestAlg(String digestAlg) {  
this.digestAlg = digestAlg;  
}  
/* */  
private void signJar(File jarSource, File jarTarget) {  
addValue(cmd, "-signjar=" + jarTarget.getAbsolutePath());  
addValue(cmd, "-signature-algorithm=" + sigAlg);  
addValue(cmd, "-digest-algorithm=" + digestAlg);  
addTimestampAuthorityCommands(cmd);  
}
```java
/**
 * Finds the closest coordinate in the array to the passed one
 * @param coordinates Array of (SORTED) candidates
 * @param coordinate Coordinate to locate
 * @return Index of the nearest coordinate (or -1 if coordinates are null)
 */
protected static int binarySearchForClosestCoordinate(double[] coordinates, double coordinate) {
    /*
    * Algorithm is based on binary search
    */
    int index = -1;
    if (coordinates != null) {
        // set the initial borders
        int left = 0;
        int right = coordinates.length - 1;
        // loop until the closest value candidate is found
        while (left <= right) {
            int i = (left + right) / 2;
            index = i;
            if (coordinate < coordinates[i])
                right = i - 1;
            else
                left = i + 1;
        }
        /*
        * the detected value is not always the closest
        * one, it may be +/- one position off
        */
        if (index >= 0) {
            double diff = Math.abs(coordinate - coordinates[index]);
            if (index > 0) {
                double lowerDiff = Math.abs(coordinate - coordinates[index - 1]);
                if (lowerDiff < diff) {
                    index = index - 1;
                    diff = lowerDiff;
                }
            }
            if (index < coordinates.length - 1) {
                double higherDiff = Math.abs(coordinate - coordinates[index + 1]);
                if (higherDiff < diff) {
                    index = index + 1;
                    diff = higherDiff;
                }
            }
            return index;
        }
    }
    return index;
}
```

Figure 9: Diff for commit Visual selection of spectra and their highlighting after selection - Modified binary search algorithm
Figure 10: Diff for commit Visual selection of spectra and their highlighting after selection - highlighting algorithm
Figure 11: Diff for commit Visual selection of spectra and their highlighting after selection - Closest spectrum selection
**Figure 12:** Diff for commit **Visual selection of spectra and their highlighting after selection - Invert color**

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

public static int invertColor(int color) {
    int origColor = new Color(color);
    * color's integer has format:
    * red: bits 16-23
    * green: bits 8-15
    * blue: bits 0-7
    *
    * so we need to invert the individual components (R,G,B)
    * and put them to their decade
    *
    int invertedRed = ((255 - origColor.getRed()) * 65536); 
    int invertedGreen = ((255 - origColor.getGreen()) * 256); 
    int invertedBlue = ((255 - origColor.getBlue())); 
    
    return invertedRed + invertedGreen + invertedBlue;
}
```

**Figure 13:** Diff for commit **Stack to FITS (image) and all SAMP spectra to same windows - spectra’s sources**

```java
// require a lot of work catching the problems with interrupting
// incomplete objects.

/**
 * Source types, from which the spectra may come from
 */

public static enum SourceType {
    UNDEFINED,
    SAMP,
    SSAP,
    LOCAL,
    ...
}
```
Figure 14: Diff for commit SpecCutter - apply on multiple spectra - SplatSpectraSelectionTable
Figure 15: Diff for commit SpecCutter - apply on multiple spectra - SpecDataCellRenderer