Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of software engineering

Bachelor's thesis

Ondřejov Southern Sky CCD Photometry Survey: Catalog Server

Jiří Nádvorník

Supervisor: RNDr. Petr Škoda, CSc.

9th May 2013

Acknowledgements

I would like to thank my supervisor Dr. Škoda for great and colliding leadership of this project and my thesis. I am also very grateful to Dr. Demleitner for very good support in every aspect of this project. Without it would be my work much harder. Last but not least, I would like to thank my colleague Daria Mikhaliova for cooperation and her efforts to develop our work towards successful ending.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In V Praze on 9th May 2013 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Nádvorník, Jiří. *Ondřejov Southern Sky CCD Photometry Survey: Catalog Server*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2013.

Abstract

This bachelor's thesis takes as it's task to use and apply informatics as a solution to common astronomical problems. The main task is to preprocess, store and publish astronomical data from a southern sky survey made by DK-154 telescope in Chile within the part of observing time used by Czech astronomers in remote observing mode. These data are to be stored and processed at Ondřejov data center by scientists from Astronomical institute AVČR in Ondřejov. The practical output of it will be an online catalog of light curves extracted from these observations. Also, this output is to serve as an example of the implementation which may help further development of astronomical standards used in this thesis.

Keywords Virtual Observatory, server, light curves, SSAP, TAP, SIAP, CCD photometry, Sky survey

Abstrakt

Tato bakalářská práce si dává za úkol uplatnit informatiku jako řešení běžných astronomických problémů. Hlavním úkolem je zpracování, uložení a zveřejnění astronomických dat získaných z přehlídky jižní oblohy nafocené teleskopem DK-154 v Chile v rámci vzdáleného pozorování českými astronomy. Tyto data budou ukládána a zpracovávána v datovém centru v Ondřejově vědci z Astronomického ústavu AVČR v Ondřejove. Použitelným výstupem bude onlinový katalog světelných křivek získaných z těchto pozorování. Tento výsledek bude také sloužit jako příklad implementace, který by mohl pomoct při dalším vývoji astronomických standartů, které zde byly použity.

Klíčová slova Virtuální observatoř, server, světelné křivky, SSAP, TAP, SIAP, CCD fotometrie, přehlídka oblohy

Contents

List of Figures

Introduction

I choose this topic of my bachelor's thesis because of my attitude to astronomy and informatics both. It is a great connection of these sciences meant to create a more complex approach in solving astronomic issues like processing and storing large amounts of data. Another part is also to get the new information from this data, majority of which is usually not used in particular research. I am trying to make it easier for astronomers to extract this data, automize the processes as much as possible, and deliver the data to them in a clear and simple way.

We will be working on an OSPS[1] project, which has among other things taken as it's goal to extract light curves from it's observations.

This bachelors thesis can be summed up in one sentence. Try to extract, process, store and publish light curves from large number of images. It is to be both automated and flexible - doing automatically what the astronomers in Ondřejov need, but leaving all the parts of the process easily changeable and expandable. It is also to be simple in the means of astronomical output it will provide. This will be accomplished by using several astronomical standards, which are well known and used in the community.

We also try here to give a practical example as a reference implemntation for identifying the light curves. It should help any other people who will try to publish the light curves in the future. Such person can save its time and efforts by inspiring in our solution instead of creating a new one.

I must also mention the cooperation on this thesis with my colleague D. Mikhaliova. We are working on the same goal, mainly united by the purposes of the project as such. But, we have divided the work in two almost independent concepts. One part is taking care of ingesting and displaying the images, which is done by my colleague. The second part is

[1] Ondřejov Southern Photometry Survey

about extracting, ingesting and interpreting the information contained in these images. I will always mention only those parts of my colleague's work , which are affecting my work directly, in order to explain their impacts on my solution.

Chapter 1

Concepts involved

In this chapter I will provide an explanation of some terms and concepts I will later use in the following chapters. It shall only make clear the facts about these terms, their advantages and drawbacks will be discussed later, as well as their consequences on my work.

1.1 The Virtual Observatory

Virtual Observatory [7] (VO) is a network of data centres holding astronomical data. It uses the internet to form a scientific environment in which astronomical programs can be efficiently used.

The goals of the Virtual observatory are to:

- allow or improve direct access to all kinds of astronomical data (photometry, astrometry, time series, spectroscopy, ...) easily, in well defined protocols,
- provide the astronomers with an easy way to find, access and use any data relevant to them,
- ensure that all the data will be properly described, can be accessed and understood in the future,
- provide programs which can be used to interpret and use all this

By meeting all these goals, the VO will be capable of providing any kind of data in any format, taken on any telescope in any astronomical facility, in the same way. This will allow the scientists to make much more efficient

use of their data. They will be able to exploit the data more efficiently. In my task, I will use some of the protocols and constructs provided by the German Astronomical Virtual observatory (later GAVO), which I will describe later.

1.2 IVOA

The International Virtual Observatory Alliance [7] is a global scientific organization founded in 2002. It consists of many projects and data centres worldwide which are working towards the same purpouse - to ensure all astronomical datasets and other resources will be accessible in uniform standard. The list of IVOA current members can be seen on picture 1.1.

Figure 1.1: IVOA Members

1.3 Light curves

This is the critical term for my whole work, however, creating one is far more difficult, than understanding it. It is a simple graph of light intensity over a period of time. Mostly it is measured in logarithmic magnitude of the object. The problem is with the obvious time-series object it would need to represent it. There are no standarts for it at the time specified by IVOA, or any other organization.

1.4 Data formats

Concerning my problem, I will be mostly working with images, because I am making light curves of objects as they are identified on them. Therefore, I will describe the data formats which are used with images in astronomy, and provide a simple explanation of aspects important to me.

1.4.1 Metadata

The metadata can be translated as "data about data". Officially, there are two kinds of metadata - structural and descriptive metadata. The first one describes the structure of some dataset and the other one describes the data itself. Structural metadata are rather provided in astronomy by standards. The data itself only contains some MIME type[2], which is already specified in a standard, which defines how the specific MIME type should look like. The descriptive metadata however, is left open for anyone who creates the data. It's usage is described on some file formats below.

1.4.2 FITS

There is large number of image formats, but in astronomy there was developed an image format called Flexible Image Transport System [18], which was first standardized in 1981 already. The last version 3.0 is from year 2008. It has the capability of storing almost any kind of data thanks to it's abstract structure. They can contain almost any kind of astronomical data. It is often filled in already by the telescope which obviously knows a lot of information about the picture it is taking.

The primary header of the image can be read as ASCII text. It contains metadata, which describe what the image itself contains. It can contain 2D images or spectra, which are some sort of 1D image. But it can also carry far more complex structures like data cubes and even multi-dimensional spectra.

FITS files also can include more headers after the first data block (which is mostly the image). These headers can represent FITS extensions added to the original image later when reducing it[3]. They can also contain some data which cannot be easily seen on the image, but can be extracted from it.

[2]Multipurpose Internet Mail Extensions. Mime types are used to describe the type of content of a data file passed between two applications over the internet.

[3]Reducing is an astronomical term for post-processing the image in order to get rid of chip errors, normalize it, or fix coordinates accordingly to the ICRS system.

1.4.3 VOTable

Virtual Observatory Table [15] is another great file structure developed and used in order to standardize and represent all astronomical data in one standard, in one file format. It was designed as a file transfer format between astronomical applications. It is ensured by it's tabular structure, which is more transparent and flexible, than in the FITS images. This is actually an XML-structured file, which begins similarly to the FITS format by metadata, which are defining the rest of the file containment. In general, this file can transfer any kind of astronomical data. An example can be seen in appendix C, listing 4.

1.5 Protocols and related stuff

VOTable is the building stone for all protocols, which are actually used to transfer astronomical data. They are provided via Web-services running on a specific address of a server. Whereas file formats are unified for all the data as much as possible, the protocols are specialized for the astronomical task they are used for. There are protocols for accessing images, spectra, photometry and astrometry tables, simple cone search protocol, and others. All of them have to support GET parameters for the purpose of easy and independent querying. All of them are defined by IVOA alliance as the main authority providing worldwide standards for better usefulness of VO as such.

1.5.1 Registry

The astronomical registry [14] are services that provides data about all other astronomical services. When we want to find astronomical data, we first look in the registry, which services are available, that provide the data we need, and then we will choose the ones we want to query.

1.5.2 ADQL

All of the protocols can use some kind of a form as "API" for building the query, which is written in ADQL, the Astronomical Data Query Language [6]. It is based on SQL, but contains default functions and methods for geometrical queries like finding results in a region on sky. In my thesis, I use it for queries, that are not protocol dependent, or cannot be realized by other protocols due to their limitations.

1.5.3 TAP

The Table Access Protocol [12] is used to retrieve tabular data and therefore it can contain any sort of information. The characteristic query language we use here is the ADQL, but it can support other languages as well. The result of the query is a VOTable, which can be displayed very easily. The TAP endpoint on a server will provide metadata of its tables first, so we can easily find the tables we actually want to query by the ADQL language. Then we can enter synchronous, or asynchronous queries, which I will also use in my thesis.

1.5.4 SCS

The Simple Cone Search [9] is probably the simplest and most abstract protocol. Its mandatory input parameters are only ICRS coordinates and a radius, forming a conical region on the sky in which we want to find the objects. Other input criteria can be added too. The output on the other hand can be represented by any kind of VOTable - any kind of observation in the given region of sky. This is very useful for data types that are not sufficiently defined in other standards. In my work I use it for example to get a list of objects defined by their positions on the sky to verify whether they were identified correctly in other processes.

1.5.5 SIAP

The Simple Image Access Protocol [10] is used to publish images. The result has to contain some kind of link to the image file. How the link works is free to choose, but when somebody clicks it, it has to return the image in FITS or other graphics format (JPEG, PNG, etc.). This service is to be used for publishing the images I extract my data from, but it is not part of my thesis. It is the work of my colleague D. Mikhaliova.

1.5.6 SSAP

The Simple Spectral Access Protocol [11] works the same as SIAP, but it is used to publish spectra instead of images, thus it has to contain other sorts of data characteristic for the spectra. This one will play a big part in publishing our light curves in the end. A spectrum is very similar to the light curve, because where SSAP's primary information is a graph of flux (intensity) over a range of wavelengths, light curve needs intensity

(in astronomical convention usually in logarithmic magnitudes) over time range.

1.6 Clients

When we have data formats, in which to store the data, and the protocols to transfer them, we need some clients in which we could display and work with the data. These are mainly customized for the needs and requirements of astronomers and scientists. I will only try to test my solution to be compatible with some clients used for displaying my data in the end.

An important and great information is, that it really doesn't matter in which environment runs the server, as most of these clients are written in JAVA and therefore multi-platform. So long as the server provides services accordingly to the IVOA standard, the clients can access them from whatever platform they choose. A brief description of some clients we will use follows.

1.6.1 Aladin

Aladin [1], typical appearance of which can be seen on picture 1.2, is in my opinion the best and the most widespread solution for visualization of astronomical images and data concluded in them. It has some very useful features to verify wheter my algorithms are doing what they should be doing and compare the results from them. It can display all sorts of images in several sky-coordinate systems, but much more important is its capability to visualize tables of objects. It can render their positions in layers over the image providing very light and intuitive way to compare them.

1.6.2 TOPCAT

Whereas Aladin is specialized to display graphical data and images, TOPCAT [24] is specialized for working with tables. Through the registry system, it can find the most suitable servers for getting the information we need and then query them by for example ADQL. It also contains some examples of FAQ queries used to get the most common data on the server we are actually working with. TOPCAT has as well very powerful spread sheet capability.

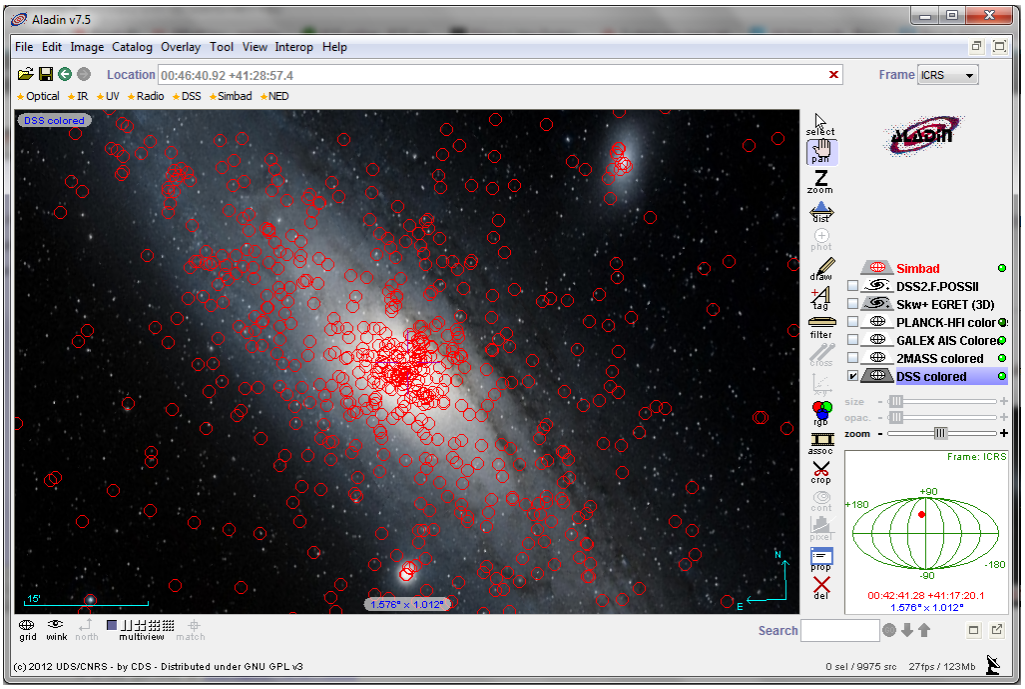


Figure 1.2: M31 Galaxy with Simbad objects over it

1.6.3 SPLAT-VO

SPLAT-VO [22] is specialized in viewing spectra. As it is relevant to our work, I mention it here too. It is able to find via VO Registry SSAP servers and query them. It can also quite nicely download and view the spectra, and render them in one graph over each other.

1.6.4 SAMP

Rarely will our work be so specific or a client so robust, that we could do all our work on it. Mostly, our work will be running on multiple clients because of our needs to collect different types of data to see the circumstances we are working in. This means we need our clients to communicate in some simple way. This is ensured by Simple Application Messaging Protocol [8]. The main structure is represented by a SAMP Hub, through which we can send point-to-point, or broadcasted messages to other running clients connected to the hub just with one click. In practice, we can, for example, find our information using TAP protocol in TOPCAT, filter the table and then send it via SAMP to Aladin to visualize it.

1.7 Servers

There also must be VO servers for running the Web-services, which will access our data through protocols. My requirements for that server as an end-user would be a light and clear access to the data, ergo to the services it provides. These services must work by the standards and my only concern is, whether they work fast and reliably.

For a person who ingests the data however, would be also important the absolute transparency, i.e. where are the data stored and how does they look like, what exactly the services are doing with it and how can I change it. With it also of course comes the possibility to change and bend the solutions of the server itself, to be able to apply it to a slightly different environment, or make it able to work with slightly different kinds of data. The simplicity on the outside provide lots of servers, however, the simplicity on the inside is quite another issue. We will discuss this in the analysis chapter.

Chapter 2

Analysis and design

2.1 Requirements

The official requirements for my thesis can be found in the thesis task, so I will rather mention here their consequences and purposes in my work.

2.1.1 Non-Functional Requirements

Platform

The environment will be Debian Linux running on a virtual server. The database used will be PostgreSQL [19] because of number of plugins with geometrical functions optimizing it's usage with astronomical data.

2.1.2 Functional Requirements

Database ingestion

The solution will have to prepare a structure in the database and store the data in it. This structure must be able to use all kinds of queries the protocols will use. It has to be reasonably fast too, ergo using efficient indexes to search in the data. The result of the query must be structured accordingly to the standard required by the protocol.

The ingestion has to be periodically repeatable and doesn't touch the data already ingested, only add the new data.

Identifying the light curve

The light curve will have to be extracted from the original data and present a unique ID, which will be the ID of the object itself. This means we will have one light curve per one object identified on the pictures.

Providing the data

There will be two services providing each one protocol (SCS for simple search in the data table, SSAP for publishing the light curve). Each of them will provide firstly a web-based access, and secondly a GET form returning VOTable and supporting parameters as defined in the IVOA specification.

2.2 Testing

The services will return data in formats supported by visualization tools like Aladin, SPLAT-VO or TOPCAT accordingly to the IVOA specifications. If I want to make the output of this thesis usable, there is no better measure, than a satisfied user who in our case will typically be an astronomer using these applications. They will be therefore used for testing and confirmation, that we successfully met our requirements.

Also, I will try to use such examples, that will nicely illustrate and explain the processes used in our solution. It is always easier to understand explanation on pictures, than in text.

2.3 Options

There is always the option to start from scratch when we don't want to use any solutions already available. The reasons could be that they don't exist, they do not fit our problem exactly, we don't like the way they are written, or we simply want to do it by ourselves. But with all these reasons, we still can't provide that amount of functionality in one year's work as we could when using something already done. This problem is too complex to do it only by myself, and along with the possibilities that are currently available for these purposes, there is no need to try it.

2.3.1 Server toolkit criteria

In the VO there are currently several server toolkits used to publish data. The most important criteria for choosing the right one for my problem are

listed below.

- *Ingestion mechanism* - The ingesting mechanism has to be flexible to allow us to map our information correctly in the database. This can be complicated, because each telescope and observatory has a different way of creating the data, thus for example the FITS files bearing the same information can have different headers. Because of that, our mechanism should be able to modify the data before ingesting. Not only by altering strings, but also by mathematical calculation and data conversion from for example hour angle to degrees.
- *Support for the latest VO standards* - The toolkit will have to support all of the protocols I need for publishing my data and if it is to be usable worldwide, it has to be by latest standards. The supported protocols will be TAP for direct access to the tables, SCS for geometrical approach, SIAP for accessing the images[4] and SSAP for accessing the light curves.
- *Interoperability with other VO Tools* - It is also important to have a simple link to clients processing the data we get from our server. These could be for example Aladin or TOPCAT and the link will be mostly represented by connection to a SAMP Hub.

2.3.2 Existing VO server toolkits

Here I compare all the solutions listed on the official sites of IVOA [13] and discuss their advantages and drawbacks.

DALServer, DAL Toolkit, DSA

All of these solutions have the same major flaw, so I put them together. Their development is frozen and since the support for latest standards is crucial, we will no longer deal with them.

VODance

VODance is a light-weight solution that does not have an ingesting mechanism. It allows us to create a layer above our data, that will publish it using DAL[5] services without actually moving our data into the database. Since this solution has very little possibilities to transparently process and modify our data, it is not suitable for our problem.

[4] SIAP is critical for my colleague's work

[5] Data Access Layer

Saada [21]

- Author: Dr. Michel from Strasbourg Observatory.
- Status: Stable version, active support.
- Technologies: Java, Tomcat, Mysql, SaadaQL instead of ADQL.
- Ingestion mechanism: Automatical, easy to use, little possibility of custom mapping.

This solution is a very user-friendly environment, easy to set up and easy to use. It supports all of the protocols needed (including TAP, SCS, SIAP, SSAP, and web interfaces).

It's drawbacks however, are it's own querying laungage SaadaQL instead of standardized ADQL language. The mapping of data against attributes in the FITS header is rather click-based, and cannot be automatized very transparently. Since our FITS files will need a lot of custom mapping we indicated this as a major problem and focused on the rest of our possibilities.

GAVO DaCHS [3]

- Author: Dr. Demleitner from Heidelberg university.
- Status: Stable version, active support.
- Technologies: Python + XML, PostgreSQL + pgShpere and q3c.
- Ingestion mechanism: Automated, powerful custom mapping, preprocessing.

In this case we underwent the deepest testing on our problem, because we didn't come upon a shortage that would be in conflict with the primary criteria we defined. Here we encounter a very robust solution able to uniformly describe metadata and data to be ingested and publish it in multiple protocols (including TAP, SCS, SIAP, SSAP, and web interfaces).

All of this is very transparently defined in a structure called RD[6] Here is found the mapping for all of the metadata, and since it is XML tag based, it is very easy to modify. Using the functions and macros defined in this RD we can easily process and convert the data. The services used for publication are also defined in this file, so all is nice, clean and transparent.

[6]Resource Descriptor is an XML file. This is very convenient, as a normal user has usually no need to look into the Python code. Almost all is usable via XML tags in this file.

All of the XML tags are then processed as Python objects. That ensures very transparent approach and despite the fact that GAVO DaCHS server has at the last revision 70000 lines of code, it is quite easy to expand with new features. The Python language provides also a quite simple and agile method of using all of its possibilities in a command line.

Last but not least, the author of this software Dr. Demleitner is very cooperative, and along with a very good online documentation it was much easier to understand server's organization and structure.

The major advantage of this server however is its biggest drawback. Its complexity and robustness makes much harder to understand the code itself, if some sort of change or fix is needed. However, we need a lot of functionality and therefore we couldn't plan for a small and easy toolkit.

2.4 My solution

In the end, our requirements turned out to be very specific, so the choice was quite clear. GAVO DaCHS took advantages in almost every aspect we need.

The most important advantages are:

- *Complexity* - This server toolkit solution can systematically define data ingestion, storage and publishing services in a very sophisticated structure of an XML file.
- *Flexibility* - As already mentioned above, the XML resource descriptor ensures us great possibility to configure the server to our needs, and if that would not suffice, Python language provides quite easy way (relatively) to expand the server itself.

2.5 Analysis

The architecture of the whole process of ingesting, storing and publishing data is quite given by the choice of using a solution that is already optimized. The flow of the data is quite simple and is described on the picture 2.1.

2.5.1 Data ingesting

There will be two kinds of images[7]. Each will be FITS files, but they will have different content.

[7]In my thesis the images can be always understood as FITS images.

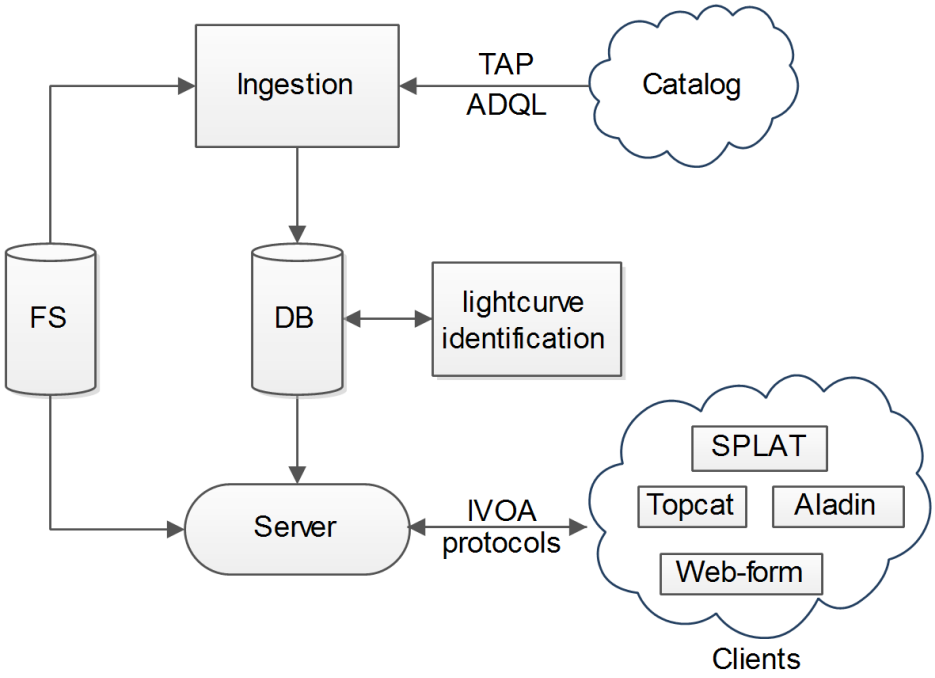


Figure 2.1: Analysis of the architecture

- *Raw data* - the first one will contain only raw data. As it was taken on the telescope (chip), it will be saved to the file, along with the metadata, that are known at the moment. These metadata may contain inaccurate numbers, like coordinates of the image, or noise in the image data.

- *Reduced data* -the second image type will be reduced. Data reduction is a term in astronomy for cleaning and improving the image. It contains of several steps, for example removing instrument signatures (e.g., bias, flatfield), or applying photometric and astrometric calibration. Here it is done by Munipack [17]. This is not only about improving the image, but also extracting the knowledge about the content of the image.

- *Binary table extension* - the second image type will be the most important to me, because it will contain a binary table extension. It is added by Munipack too, right after the original content of the FITS

image and has similar structure to it. It has its own ASCII header with metadata describing the actual content of the binary table below it. The binary table contains all objects identified on the image. An example can be found in the appendix C on listing 7. The binary table extension of course doesn't carry redundant information about the image itself. This information is contained in the primary header at the beginning of the file.

An illustration of how the preparation of the data looks like can be found on picture 2.2 and the structure of the headers can be found in appendix C. The raw data header on listing 5, the information added to the primary header by Munipack on listing 6 and the binary table extension header is found on listing 7.

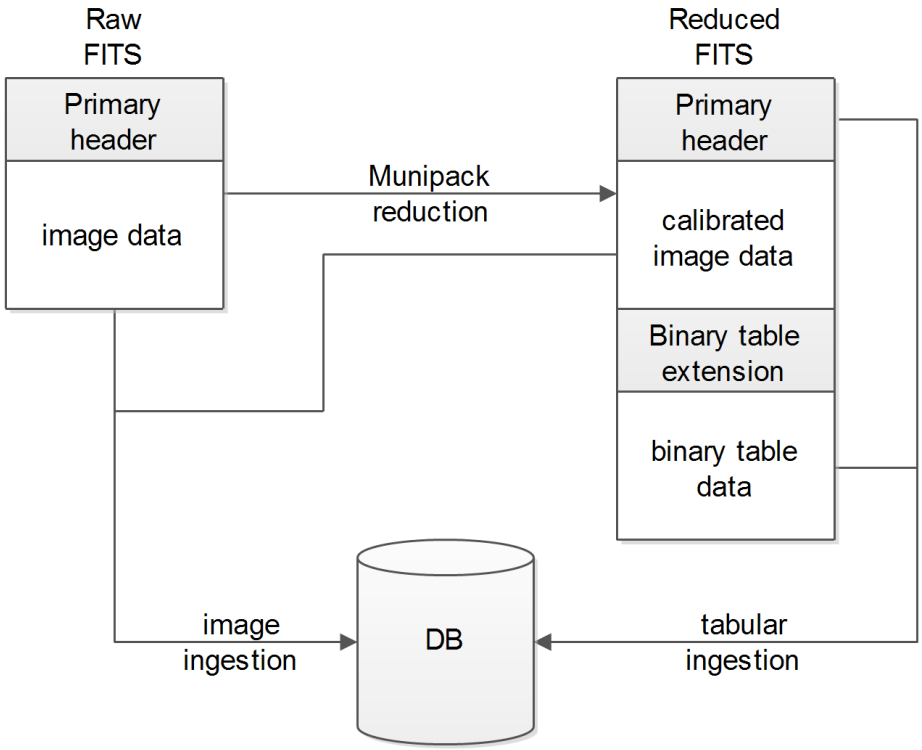


Figure 2.2: Analysis of the data

2.5.2 Building the light curve

The main problem is, that the observations of one object won't have even after calibration their positions equal. The radius of their occurrence can be statistically proclaimed as one arcsecond. We will assume that all observations taken in this radius represent one object and therefore form a single light curve.

There is also a big issue of how to identify the objects on our images. The best option came out as crossmatching[8] the objects on our images with an online catalog. However, this also can't be done by exact matching of the coordinates, because the coordinates vary around the object position in the catalog. Given by the accuracy of Munipack reduction, we can consider all observations within one arcsec radius from the catalog position as one object as well.

The visualization of what is exactly done can be found on image 2.3.

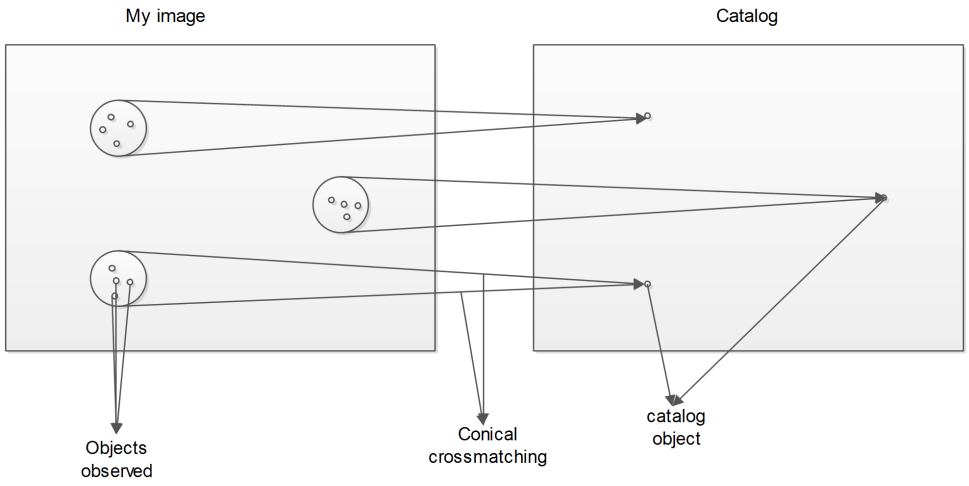


Figure 2.3: Catalog identification

[8]With crossmatching we mean here identifying an object we don't know by matching it's coordinates with an object we know.

2.5.3 Publishing light curve

2.5.3.1 SCS protocol

The intermediate product of our observations will hold complete information about them (the band, time, and their crossmatch id against catalog), and will be published by the SCS protocol. The SCS service is used to publish all our data before it is "divided" into the individual light curves. It is important, because when we have senseless values in the light curve, we can visualize the observation from which it was taken, and see what is wrong.

Since the SCS protocol by default provides only geometrical parameters, we will expand it by some other options that we need.

The input parameters supported by our SCS service will be :

- Position/Name - In a standard SCS service this is the mandatory RA and DEC parameters. A name resolver is attached to this column ergo standard object names like *ngc 330* will be resolved and converted to their coordinates, which will be passed to the query itself.

- Search radius - This one is mandatory for standard SCS too. It provides the radius within which the query will return results. Together with the Position of the target it forms the basic geometrical condition for the query.

- Bandpass - We add this column to the standard SCS definition. It provides a simple direct match condition of the band name entered. This means it will only return those results, which have the band name exactly same as provided in this field.

- Minimum Date + Maximum Date - If entered both, it will form a simple BETWEEN clause for the query. If entered only one of these, it will form an open interval and add it as a condition to the database query. Ergo with only minimum date entered, the query will return every observation since that date, if maximum date is entered, the observations taken before this date will be returned.

The SCS query is also specifically intended to be used by the Aladin client, because of simple visualisation of the results. We will ensure that we can access this service directly from Aladin too.

2.5.3.2 SSAP protocol

The final table which will be published via the SSAP will have as little information as possible to fulfill the standart requirements.

In the end, we are only trying to bypass them because we are trying to use the SSAP for something it wasn't built for. That is also the part for discussion, what can be done about it. In my solution, there will be the information identifying the light curve, like the date, band, or location. All of these are necessary because they will represent the parameters of the SSAP service and the light curve will be searched by them.

However, the most important column is the *accref*[9]. In SSAP it is usually a link to a physical FITS file carrying the spectrum. The CoRot approach also uses physical FITS files to store the light curve.

That in my opinion is unnecessary, and would only store redundant data on the hard drive. We can generate the light curve on fly if we have all the data in the database already. So we decided to use the *accref* as a URL which points to the TAP endpoint of our server and bears the ADQL query which extracts the light curve of given parameters from our server.

It will also be able to cut the spectra, when the minimum date, or maximum date of our observations will exceed the query date parameters. This will be actually quite easy compared to the light curve stored on hard drive, because here it will be only two more parameters of a SELECT query.

The input parameters supported by our SSAP service will be:

- Location - This parameter works the same as in SCS service. It is the center of a cone search done by the query. Name resolver is also attached.
- Search radius - The same as in SCS service. Completes the geometrical condition for the query.
- Bandpass - The same as in SCS service. Direct match condition against the name of the band will be added to the query.
- Minimum Date + Maximum Date - These will have a little more complex purpose, than in a simple SCS query. They will cut the light curve stored in the database, returning only observations taken after the minimum date and before the maximum date. Only one of these can be entered too, for open intervals.

The SSAP query will be mostly accessed from the SPLAT-VO client. Therefore we will ensure that the client will be able to display our results.

[9] Access reference, used as link to download the file (mostly FITS)

Chapter 3

Implementation

The GAVO server uses a construct of the resource descriptor not as a configuration file, but as declarative programming language [4]. It consists of XML tags, which alone provide a structured access to scripts and functions from inside the GAVO DaCHS source code. The GAVO DaCHS package can be called as an "interpreter" for the programming language constructed within this document. The XML tags have variety of attributes, which also represent parts of code from the package. We can use predefined macros and functions inside the XML elements too.

Also, when the server package cannot provide the exact functionality we need, we can use elements named *script*, which allow us to add scripts in other languages, e.g. Python, SQL. These can be run at a specified point during table creation or teardown, which are exactly the moments where we can transform the data before ingesting it.

In the resource descriptor are defined all the information and processes needed for ingesting the data, or working with it. There is typically one RD file per one resource[10].

In this chapter, I will describe my RD program from the perspective of its functionality, and reference the lines of code in appendix B. If it is not enough, please seek out the GAVO DaCHS documentation, which is absolutely exhausting [5].

3.1 Database

PostgreSQL Database has a great option of creating database schema as a hierarchical structure above our tables. That is very useful to us, because

[10]By resource we mean here a collection of data of the same format

my colleague D. Mikhaliova will work in one schema used for storing the images and I in another one used for extraction the data from the images. This allows us to work almost independently and without the fear, that we would destroy each others data, but still we are using the same database ergo the same server for publication our data.

A simple database schema is described on figure 3.1. The sourceId is an identifier for one object in the catalog, obsname is the identifier for one image (i.e. file) and the starNo column is the identifier of an object inside one image (it is the number of the row at which it is stored in the binary table.

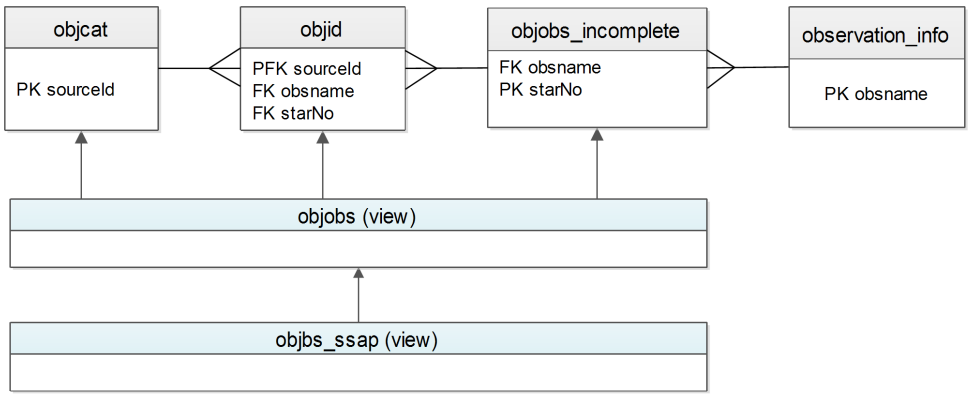


Figure 3.1: Database schema.

3.1.1 Defining target tables

The trickiest part for me in this is to store the data in the database in an organization which will provide fast access and searching for the light curves. The content of the concrete tables is defined below. The tables are defined in the RD in a very sophisticated way, an example can be found between lines 9 and 48 in the RD[11].

3.1.1.1 Tables

Since the reading from files is done by grammars which handle them by units which uniform structure (e.g. headers, binary tables), we cannot read

[11] Appendix B

from primary header, secondary header (binary table header) or the binary table at the same time. The best solution here is to create one table for each such section of the file.

Table observation_info
This table is used for the primary header of the FITS file. We will need metadata describing all the data on the image therefore they will be the same for all rows in the binary table extension. That gives us a reason to separate them in a single table too, because we don't want their redundant storage in each row of the binary table.

Table objobs_incomplete
We don't need any information from the binary header and all we want to ingest are the data in the binary table itself. That means one row per each object identified on the image stored in this particular FITS file. This can be easily done by a grammar too, and best solution is to create a second table, which will be joined with the first one on the *obsname* column[12].

Now we have all data, that will form the light curve, but we need yet something to identify the light curve itself.

Table objcat
After long analysis and discussion involving Dr. Demleitner, we choose to cross-match our observations against an online catalog. There were two possibilities useful for this task - UCAC4 catalog [25] and PPMXL catalog [20]. But where UCAC4 had hundreds of objects per one our image, the PPMXL had thousands. Also when trying to crossmatch, PPMXL left far less objects observed without match. In these circumstances we choose the PPMXL catalog.

Since the catalog contains literally astronomical amounts of data, cross-matching online would be very problematic. We have to download at least part of it and then ingest it to another table in the same database schema. When we have all the data in one database, the crossmatching will be very fast.

Table objid
The last table will have small number of columns. It represents a sort of m:n database relation, joining columns identifying the object within a catalog and columns identifying one observation of an object in our images. Therefor it contains foreign keys from these tables *objcat* and *objobs_incomplete*.

[12]Identifier of the file the data was taken from

It also contains the offset of how far was the observation from the catalog coordinates.

3.1.1.2 Views

There are two views in my database and they are both specialized for the protocol, which will access them.

View objobs
This view will contain all information about our observations we could retrieve and will used for publication via SCS protocol. One row will represent one observation of an object identified on our image. All of the rows which were successfully crossmatched against a physical object in the catalog contain an identifier of this object too, others are left N/A. This column will be used for the final separation of a light curve from this view. The script for creating the view can be found in the RD[13] between lines 235 and 247.

View objobs_ssap
This view will be based on the previous view *objobs*, and will be specifically used for publishing the light curve via SSAP protocol. It will contain accref column, which's content will be a URL with a database query. This query will extract one specific light curve from the *objobs* view, and return it as a VOTable compatible with the SSAP standards and convenient to be displayed in a spectral analysis clients like SPLAT-VO.

For searching in the lightc urves as such, no internal identifier will be used, but ICRS[14] coordinates. That will be quite user-friendly and correspond with the protocol requirements. Then it will also contain other simple information characterizing the light curve, like the band, time epoch of the light curve or publisher. The script for creating this view is very important and can be found in the RD[15] between lines 298 and 322.

3.2 Extracting from FITS images

When we have our tables defined, we need to fill them.

The first and easiest problem is to read data from FITS file. I'm not overlooking this, because this is not only sequential reading, but also combining the data from primary header with the Binary Table data. The

[13] Appendix B
[14] International Celestial Reference System
[15] Appendix B

second step is to transform the data to the form I want to be stored in the database. That includes formal naming corrections and datatype conversions. These ingestion scripts can be found in the RD[16] between lines 327 and 389.

The first step is done by grammars embedded in the GAVO DaCHS server. The grammar returns simple dictionaries of rows from the FITS file. That enables simple mapping from attribute name in the FITS file to the name of column in our database.

Filling observation_info table
For reading from headers of a FITS file, *FITSProdGrammar* can be used. This one we use to ingest data describing the image, e.g. coordinates of the image, the date of the observation, or the band. All of the computing done here is some conversion of a date format. Some strings are also parsed in tiny Python scripts (for example observation name which is parsed from the path to the file). That can be simplified by functions and macros usable within the ingestion procedure in the RD.

Filling objobs_incomplete table
For reading the Binary table from the FITS extension, we can use *BinaryGrammar*. This grammar works the same as all other grammars, returning dictionary with key as the name of the column and value as its content. There are lots columns in the binary table, but all of them can be useful when analyzing the light curve.

Updating tables
Both of these tables are filled within data elements in the RD. There are two ones for rewriting the whole table in the case of changing its structure, but the ones, that are automated, only add the new sources, ignore the ones already ingested. That is done by using the *ignoreSources* clause with the parameter as a database query, which will return accref of the files already ingested.

3.3 Downloading catalog

The PPMXL Catalog contains very large amounts of data. With one row per object identified in the catalog, the catalog contains about 900 million

[16] Appendix B

rows. It would be very ineffective to download the whole catalog and ingest it to our database, when all we need are a few fields from the catalog.

The only reasonable way that occurred to us was to download only part of the catalog as a result of a TAP query on the server within the areas. Geometrically, these areas are cone searches around our images, with a radius, that safely covers them. We can use the attributes from the original image header, where the coordinates of the center of the image can be found for it.

However, it would be very inefficient to build one area for each image. Not only that it would be much slower when because of much more constraints for the SELECT query, but also we must not forget, that we are limited with the length of the curl script we are using. If we didn't optimize our query, with large amounts of data could easily happen, that we exceed the operation system memory for commandline parameters. The only thing we could do then would be to re-compile the linux core, and that would make our solution very unportable.

What I do here is a little trick joining the table on itself using the coordinates on a geometrical condition. The condition is provided by a Q3C plugin function in the database, which takes x,y coordinates of two objects, and returns true, if the two objects are within the radius of each other.

The SQL script for joining the tables is very important and therefore extracted from the code and can be seen on listing 1. We select image coordinates (*orira* and *oridec*) from the *observation_info* table and get rid of duplicities which will occur in 1 arcmin radius.

Then the Python script for building ADQL Query condition is listed below on listing 2. Here we create from the orira and oridec columns the condition for the geometrical TAP query. It ensures, that all the PPMXL objects characterized as a POINT contained within the CIRCLE of 15 arcmin within center of the image will be returned as a result of the query.

The whole script can be found between the lines 391 and 430 in the RD[17].

[17]Appendix B

```
SELECT DISTINCT obs1.orira,obs1.oridec
  FROM bextract.observation_info as obs1
JOIN
  (SELECT observation_info.orira,observation_info.oridec
    FROM bextract.observation_info) as obs2
ON q3c_join(obs1.orira, obs1.oridec,
            obs2.orira, obs2.oridec,
            0.016666667);
```

Listing 1: SQL script for optimising the query.

```
"OR ".join(
   ["(1=
  CONTAINS(POINT('ICRS',ppmxl.main.raj2000,ppmxl.main.dej2000),
         CIRCLE('ICRS', %(orira).14f , %(oridec).14f, 0.25)))"
               %x for x in coordinates])
```

Listing 2: Python script for condition generation.

After building the query, we curl it on a link to a GAVO DaCHS online catalog TAP endpoint[18].

The VOTable returned by this service is stored into a temporary file and then ingested to the *objcat* table using a *VOTableGrammar*.

3.4 Crossmatching

The crossmatching is done in another script held in the dataelement *crossmatch*. In this script we fill the table objid, which holds foreign keys to *objcat*[19] and to *objobs_incomplete*[20].

These columns are joining the identifier of an object in a catalog with our concrete observation of this object. For one object in the catalog, there will be typically lots of observations. The crossmatching script is too long to be shown here, but the most important part on it is the condition on which

[18]The TAP endpoint is a URL, which points to a service supporting GET parameters, at which we can send the query using the parameters. The result is returned in a format we ask, it can be VOTable, FITS, etc. as defined in TAP specification by the IVOA organization.

[19]sourceId in the catalog table

[20]starNo and obsname - identifiers for one observation within the extracted binary table data

the objects[21] are crossmatched. It is shown on listing 3 - our observations are joined with the catalogs within 1 arcsec. The whole script can be found in the RD[22] between lines 432 and 454.

```
WHERE q3c_join(observation.raj2000, observation.dej2000,
        catalog.raj2000, catalog.dej2000,
        0.000277778)
```

Listing 3: Crossmatching ADQL script.

The crossmatching results will be joined into the *objobs* view. However, not all objects are matched against the catalog, and their sourceId is left as N/A. We could hold these when using a LEFT JOIN, instead of INNER JOIN when building the view, but that has a fatal impact on the speed of handling queries on the view.

Therefore we leave them unpublished at the time, until the algorithm of their identification will be implemented.

3.5 Publishing the data

In this section I explain how the services for SCS and SSAP publish the views *objobs* and *objobs_ssap*. The source code doing this logic can be found between lines 466 and 528 in the RD.

When defining the service, we can add some attributes like the name of it and some default arguments - the name will be used for access URL on the server. Also here is specified, in which form the service will be accessible. Typically we would want scs.XML for application access[23] and form access for web-based access.

The important query logic is done in a component called *dbCore*[24], which provides the access to the database as such.

3.5.1 SCS service

The first view[25] is used for "direct" access to the observations we provide and it is published[26] using SCS.

[21] the object is meant as one row in a table here
[22] Appendix B
[23] applications access via GET parameters
[24] The Database core
[25] objobs
[26] Publishing is done in an RD too, defined in data element service.

The dbCore for *objobs* view is called *scsCore* and is built from components called *condDesc*[27]. As this service is almost regular Cone search, we can use some predefined *condDesc* from the system RD's[28]

The *condDesc* for SCS can be inherited from a system RD for SCS, but it does not include the band and date condition we need. That means we have to define these condition descriptors for ourselves. When we want a direct match condition (equal) against one column in the table, we can use the *buildFrom* attribute of the *coneDesc*.

However, for the date, we would like to have quite different behavior. When the user fills in minimum date and maximum date of the observation, the building of the SQL query is quite simple. But we also would like to have an option to enter only the minimum date, or the maximum date. That would mean, we want a result taken later than the date, respectively sooner. This is not standard supported behavior according to the SCS specification by IVOA, so we have to define it for ourselves. That can be done in a structure called *phrasemaker*.

The *phrasemaker* structure is used for generating the ADQL query itself from the parameters translated by the protocol we are using. All it does is that it yields conditions for the WHERE clause in the final query. Typically, it is hidden in system RDs and we don't come in touch with it. But, if we want a behavior which is not standard, or not implemented in the RDs, we have to write it for ourselves. This case has occurred in *scsCore* with the *condDesc* for date. It can be found in the RD[29] between lines 484 and 504.

Altering Aladin for SCS

The SCS service also has to be usable directly from Aladin. That can be easily accomplished by storing in the registry, however, until it will be complete and working correctly, there is no reason to publish it. Mainly for the testing purposes, we solved this by telling the Aladin the information it would be normally provided to it by the registry.

That can be easily done by a glufile [23], which is an Aladin configuration construct able to define new server connections directly by providing address of the service, and parameters it supports. The final *glufile* can even be embedded into the Aladin jarfile so we don't need to load it manually.

[27] The Condition descriptor

[28] There are some system Resource Descriptors, which can be referenced in ours and easy our life with predefined procedures and macros.

[29] Appendix B

3.5.2 SSAP service

As the name foretells, this view is the one that in the end is seen published by SSAP table.

The location is built quite easily from coordinates within the *objobs* view, but the accref is a little bit complicated. Since we want it to contain a query, which will generate the light curve of the parameters we contain in this table, we have to generate the URL from the view. The URL also contains the TIME parameter, which is generated eighter from the mindate and maxdate of the light curve stored, or from mindate and maxdate given by the original query parameters, thus cutting our light curve on the time axis.

The SSAP service has to work under the SPLAT-VO client too, as it is the client we chose for visualizing the light curve. Problematic here is the format of the URL representing the light curve. SPLAT-VO has problems to understand it at the time.

3.5.3 TAP service

The table is published using the TAP endpoint automatically, if it is defined to support ADQL queries.

Chapter 4

Usage and testing

Since all the optimization we can do in our solution is to ensure, that all of the columns by which the tables will be searched have to be indexed, there is no sane reason for performance or other testing. Therefore all of the testing will be done as an end user testing, which was mainly provided by my supervisor Dr. Škoda.

By showing all the test cases on different endpoints of the services and using several applications to do so, I will prove, that my solution has fulfilled all of the original requirements and works as it should work.

Since all of the testing will be done by Aladin, TOPCAT and SPLAT-VO, this chapter is also illustrating their practical usage in our solution. I will also try to show a visual explanation of the processes used in our solution.

The solution is at the time *24/7* available at the address of `http://vos2.asu.cas.cz`, where our installation of GAVO DaCHS server is running and providing the services implemented in the scope of this thesis.

4.1 TAP service

The TAP service can be used directly on the server via its component called TAPshell, but for nicer access we can use the TOPCAT program.

4.1.1 TOPCAT

The topcat will access the TAP endpoint of the server which is located on a direct URL of our server `http://vos2.asu.cas.cz/__system__/tap/run/tap`.

When we access the endpoint, we can see list of tables the server actually provides. Here we can choose which of them we want to use and write our own ADQL query. After selecting the service by its URL, the ADQL query can be sent as provides the example 4.1. We select here a light curve by its internal sourceId. The "mag<99" condition is meant to get rid of manually excluded objects. These are identified by a magnitude 99.99, which is a fixed value for a broken object (bad result of calibration).

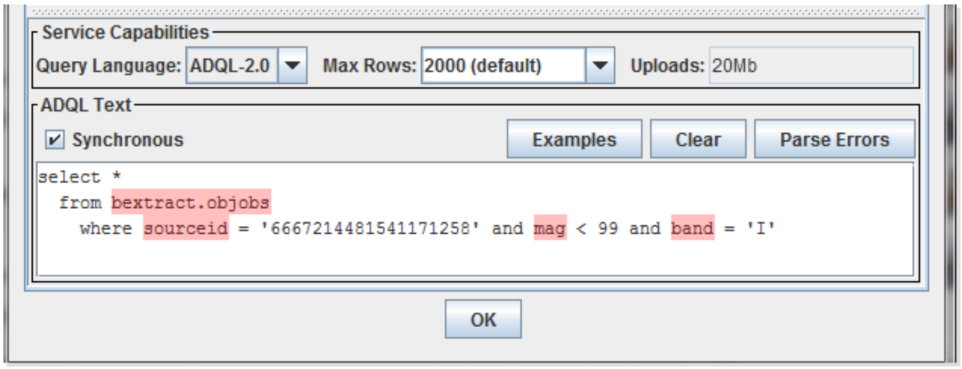


Figure 4.1: SCS query example in TOPCAT.

4.2 SCS service

The SCS service can be accessed eighter by a web-based form through the GAVO DaCHS server or by the application XML service endpoint.

4.2.1 Web-access

The web form can be accessed by direct URL of the service, which in our case is `http://vos2.asu.cas.cz/extract/q/scs/form` . On this link we can test our web-based access to the SCS service accessing our observations.

An example of a query is shown below. There is an example of how to fill in the form at the picture 4.2.

An image can be found below.

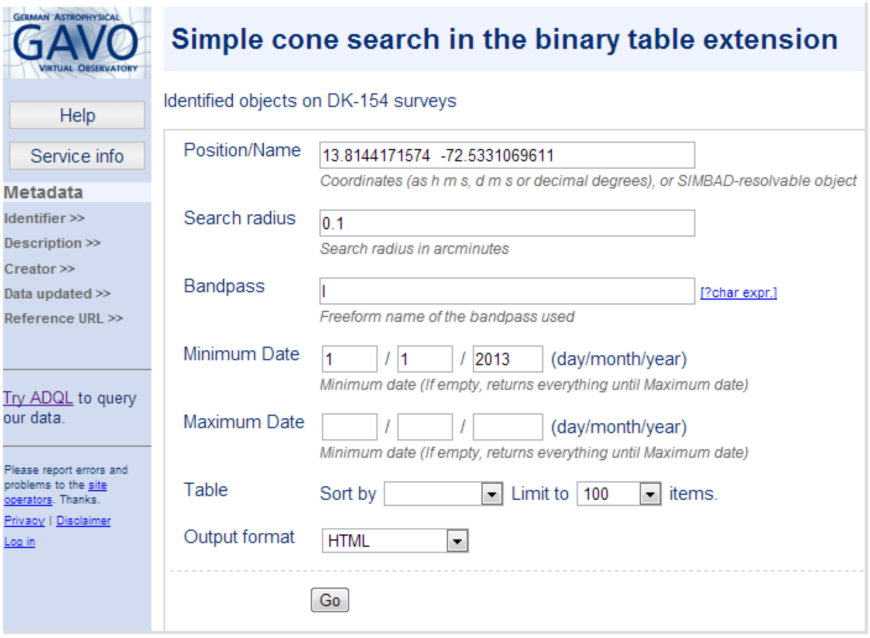


Figure 4.2: A simple example of the SCS form.

4.2.2 TOPCAT

For the application access the URL is different `http://vos2.asu.cas.cz/extract/q/scs/scs.XML?`. It can be used by any application supporting the SCS default parameters - for example TOPCAT. When selecting an SCS query in TOPCAT, we can query our server as described on picture 4.3.

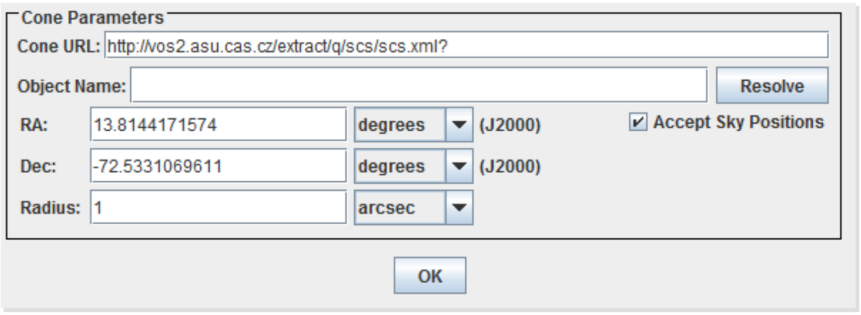


Figure 4.3: TOPCAT SCS query

4.2.3 Aladin

The SCS service however, can be accessed directly from Aladin too. We can query the SCS service as described in the guide[30] F.2. The result follows on picture 4.4 and when we zoom in on one of the cluster of points, we can see the observations, which represent one object (i.e. one light curve), on the figure 4.5. The line on the picture is measuring the distance between the most distant observations - it is less than 1 arcsec - therefore all of the points will be identified as one light curve.

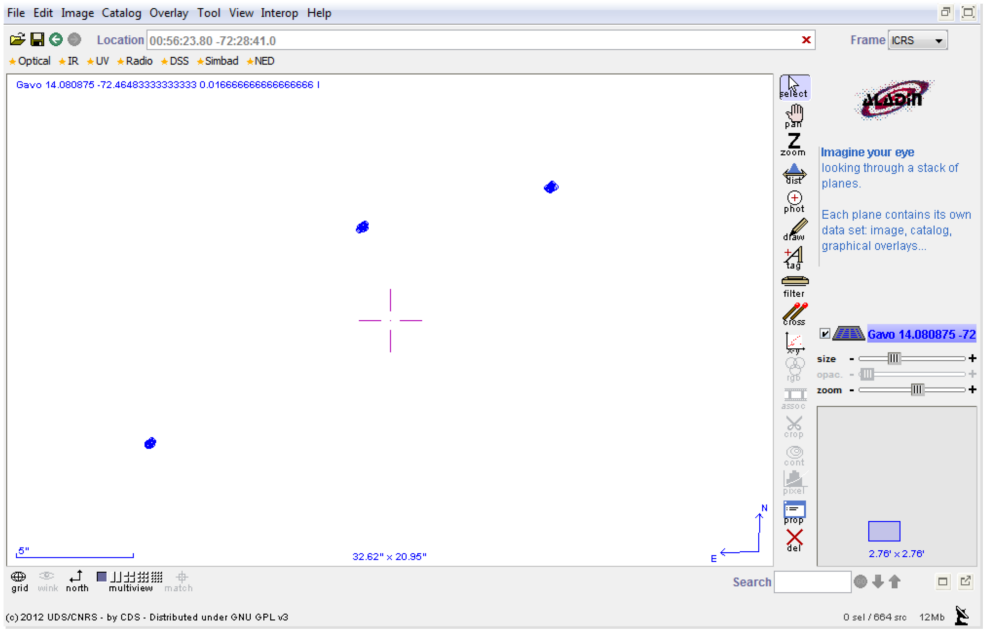
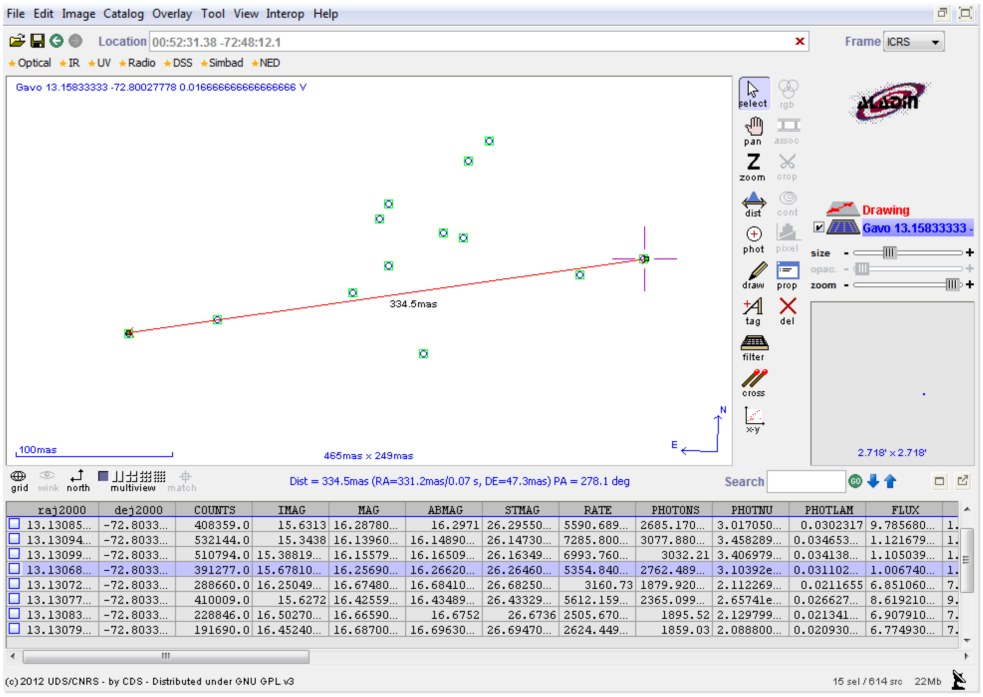


Figure 4.4: Clusters of observations.

[30] Appendix F



raj2000	dej2000	COUNTS	IMAG	MAG	ABMAG	STMAG	RATE	PHOTONS	PHOTNU	PHOTLAM	FLUX
13.13085...	-72.8033...	408359.0	15.6313	16.28780...	16.2971	26.29550...	5590.689...	2685.170...	3.017050...	0.0302317	9.785680...
13.13094...	-72.8033...	532144.0	15.3438	16.13960...	16.14890...	26.14730...	7285.800...	3077.880...	3.458289...	0.034653...	1.121679...
13.13099...	-72.8033...	510794.0	15.38819...	16.15579...	16.16509...	26.16349...	6993.760...	3032.21	3.406979...	0.034138...	1.105039...
13.13068...	-72.8033...	391277.0	15.67810...	16.25690...	16.26620...	26.26460...	5354.840...	2762.489...	3.10392e...	0.031102...	1.006740...
13.13072...	-72.8033...	288660.0	16.25049...	16.67480...	16.68410...	26.68250...	3160.73	1879.920...	2.112269...	0.0211655	6.851060...
13.13077...	-72.8033...	410009.0	15.6272	16.42559...	16.43489...	26.43329...	5612.159...	2365.099...	2.65741e...	0.026627...	8.619210...
13.13083...	-72.8033...	228846.0	16.50270...	16.66590...	16.6752	26.6736	2505.670...	1895.52	2.129799...	0.021341...	6.907910...
13.13079...	-72.8033...	191690.0	16.45240...	16.68700...	16.69630...	26.69470...	2624.449...	1859.03	2.088800...	0.020930...	6.774930...

Figure 4.5: Zoomed observations representing the light curve.

4.3 SSAP service

The SSAP service will be mostly used directly from SPLAT-VO client. The result of these queries can be displayed as our wanted light curve. Being able to display it in such programs as SPLAT-VO is the goal of this thesis.

4.3.1 SPLAT-VO

An example of a light curve can be retrieved as on picture F.6 in the guide[31] and the result - the light curve - can be seen on picture 4.6. Sadly, the nature of our observations will not provide us some nice periodical steady lighcurves. The observations can't be taken periodically, because we can't for example observe very well, when it's cloudy. There are always several observations of the object in a short period of time, and then a long pause until next image of the object is taken.

[31] Appendix F

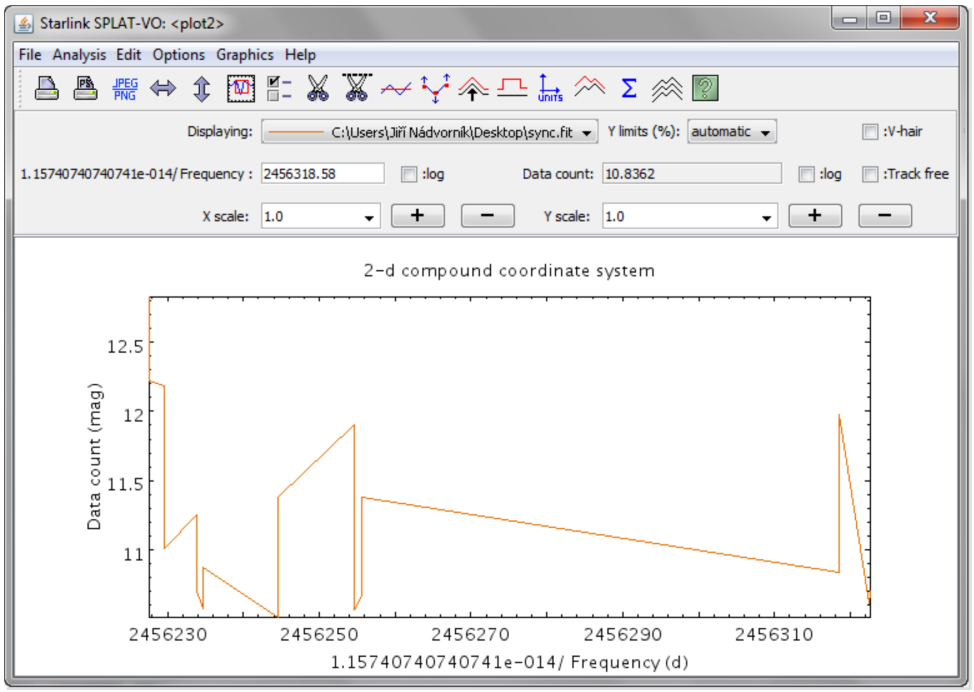


Figure 4.6: The final light curve.

4.4 SAMP usage

All of the applications already mentioned have the possibility to send results of their work via SAMP Hub to other applications. On one click, we can transfer any data output from our application to other applications connected to the SAMP Hub. An illustration is shown in the following process.

First, we select all observations which appear to be of one object in Aladin and then send them to TOPCAT using the interop button as shown on the picture 4.7. Then we switch to the TOPCAT window, open the table and then we can dispose of results we don't want to be shown. These can be for example the 100 values in magnitude columns, which mean their observations are somehow broken. Then we make a row subset from the others as shown on the picture 4.8.

This subset represents a light curve in a simple table and we can eighter display it in TOPCAT as on picture 4.9, or send to SPLAT-VO for further work. When we send it from TOPCAT by SAMP and switch to the

SPLAT-VO window then we can choose the heliocentric date for x axe and magnitude for y axe. The light curve displayed then is shown on picture 4.10. We can see, that the light curve looks the same in TOPCAT and SPLAT-VO.

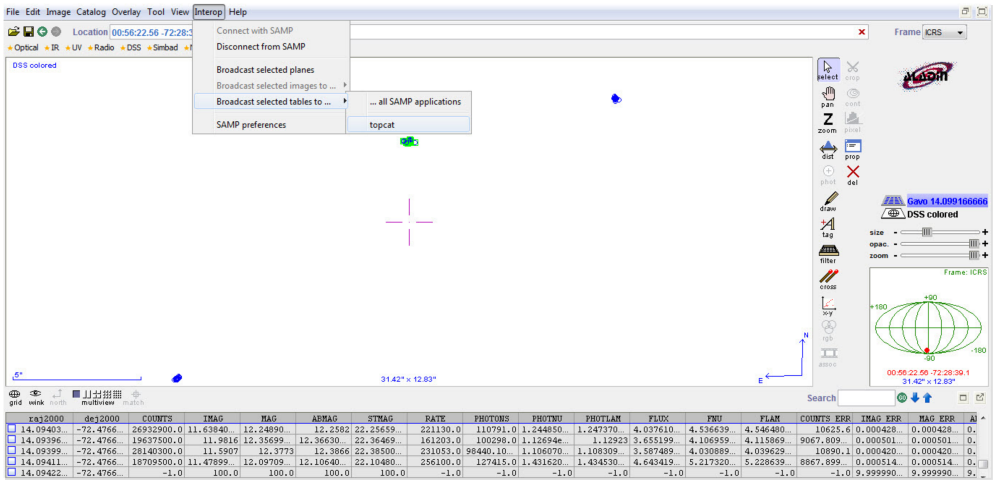


Figure 4.7: Send to topcat button in the Aladin application.

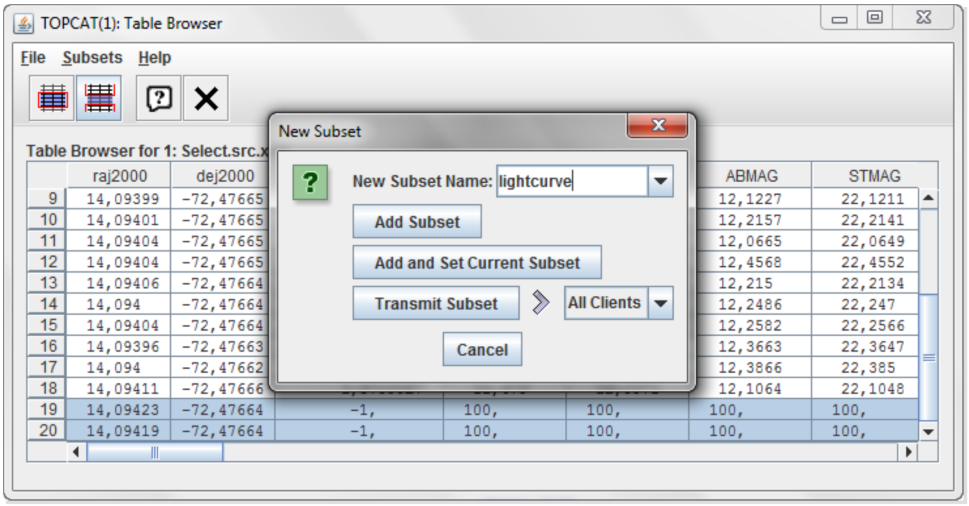


Figure 4.8: Cleaning up the result in TOPCAT and creating a subset from it.

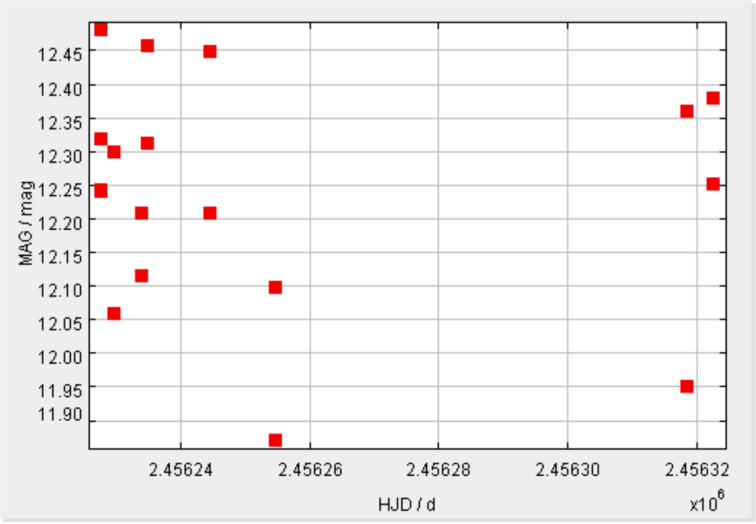


Figure 4.9: Displaying the light curve in TOPCAT graph.

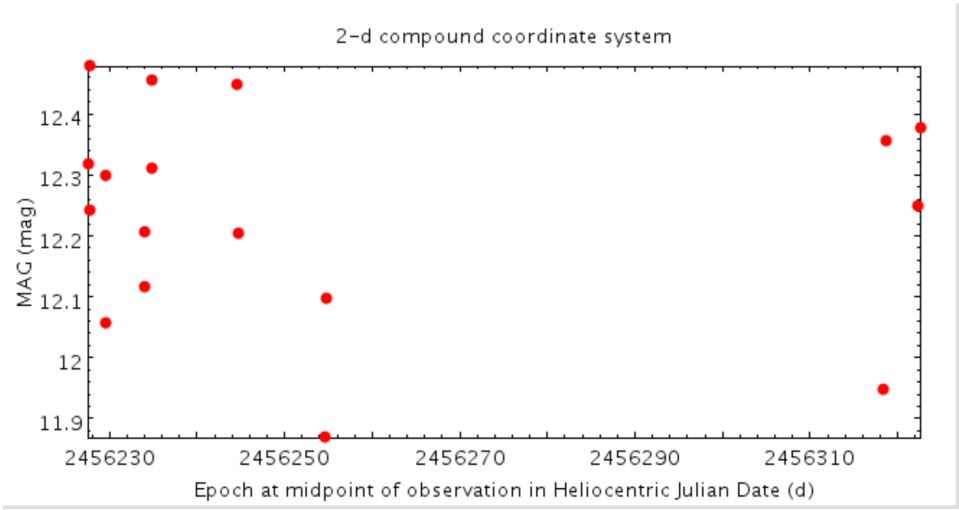


Figure 4.10: Displaying the light curve in SPLAT.

4.5 Usage with SIAP

This thesis is very closely linked with my colleague's Daria Mikhaliova's work. It is very useful to combine our results, so I will represent this in the following process.

Also, by this example I will show the circumstances in which I define our light curve. On the first picture we can see the current main target of the project observations - the SMC[32].

On the other images, I have zoomed to one image and sliced the Aladin panels away for better detail. On the first image 4.11 we can see the image obtained by SIAP, ergo my colleague D. Mikhaliova's thesis. The second image 4.12 is the same one, but with the objects identified on the image[33] as yellow pluses. When we put our crossmatched objects over the image 4.13, we can see that almost all of them are identified inside the blue circles.

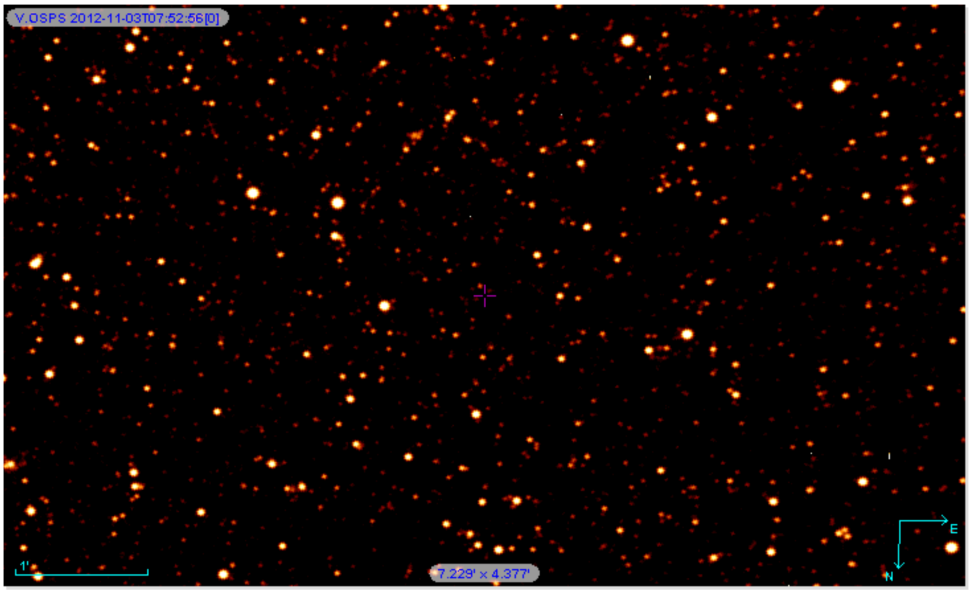


Figure 4.11: Image taken by the DK-154 telescope in Chile in Aladin.

[32]Small Magellanic Cloud
[33]These are the rows from the binary table

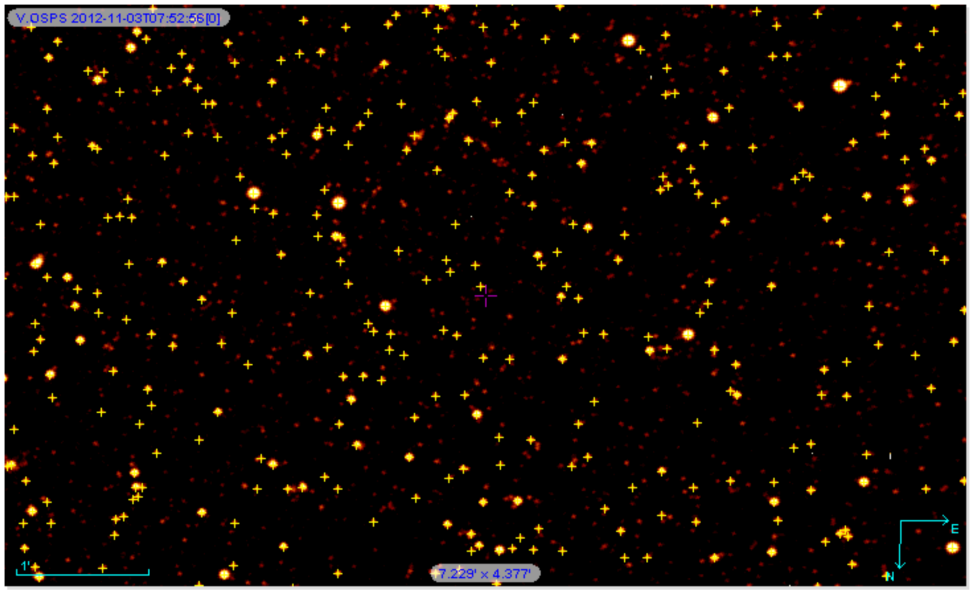


Figure 4.12: Original image with yellow pluses as identified objects.

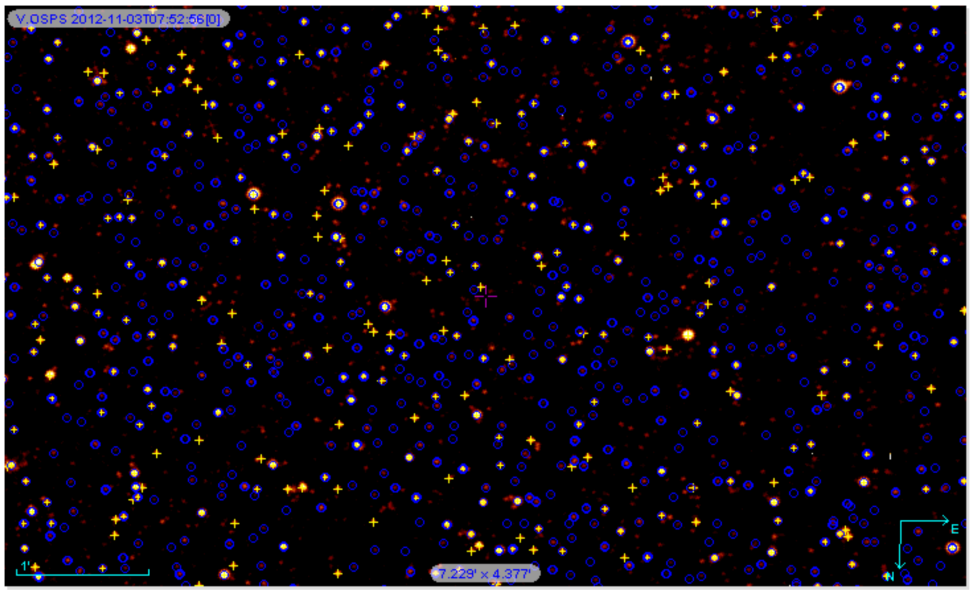


Figure 4.13: Previous image with crossmatched objects as blue circles.

Chapter 5

Future work

There is still much to be done in this project. The identification against the catalog has its advantages, but does not identify all our light curves because there are lots of objects not listed in the catalog. These will have to be identified in other ways and will add more valuable data to our output.

Another thing is, that the other parts of the project are changing in order to improve the output and I will have to adapt my solution too. There will be also changes to the SSAP protocol too, maybe even new SSAP version will take place. That will also add more ways to improve our solution and output of the OSPS project.

Conclusion

The goal of this thesis has been met. All of the light curves identified within pictures taken by OSPS project have been published and are available on our server for inspection. The publication is done with usage of official standards and is able to support the original idea of Virtual Observatory — to publish any sort of astronomical data under unified access standards.

The ingestion mechanism is able to workin two ways. Eighter rewrites the tables, for example when their structure is changed, or corrupted, or only updates them, which is normal behavior of the script. The update script completely ignores all files already ingested, and it is therefore as efficient, as it should be.

The publication of light curves had to be done with SSAP protocol, which was never meant for it, but was quite similar to the principal of interpreting a light curve. Still, we have achieved admirable results and by showing a practical example, we have given a reason to expand the time-series SSAP aspects, which I believe will be a good solution for this topic in the future.

We also presented our results at the IVOA Interoperability meeting in Heidelberg [16], where we shared our experience with identification and interpretation light curves and tried to help with the development of the SSAP standard towards the time-series direction.

Bibliography

[1] Aladin. [cit. 2013-05-08]. Available at WWW: <http://aladin.u-strasbg.fr/>

[2] *GAVO DaCHS installation documentation*. [cit. 2013-05-08]. Available at WWW: <http://docs.g-vo.org/DaCHS/install.htm/>

[3] GAVO Data Center Helper Suite server. [cit. 2013-05-08]. Available at WWW: <http://dc.zah.uni-heidelberg.de/>

[4] Declarative programming. [cit. 2013-05-08]. Available at WWW: <http://en.wikipedia.org/wiki/Declarative_programming>

[5] German Astrophysical Virtual Observatory: *GAVO DaCHS Documentation*. [cit. 2013-05-08]. Available at WWW: <http://vo.ari.uni-heidelberg.de/docs/DaCHS/>

[6] International Virtual Observatory Alliance: *ADQL*. [cit. 2013-05-08]. Available at WWW: <http://www.ivoa.net/documents/cover/ADQL-20081030.html>

[7] International Virtual Observatory Alliance: *IVOA*. [cit. 2013-05-08]. Available at WWW: <http://www.ivoa.net/>

[8] International Virtual Observatory Alliance: *SAMP*. [cit. 2013-05-08]. Available at WWW: <http://ivoa.net/documents/SAMP/index.html>

[9] International Virtual Observatory Alliance: *SCS*. [cit. 2013-05-08]. Available at WWW: <http://www.ivoa.net/documents/latest/ConeSearch.html>

[10] International Virtual Observatory Alliance: *SIAP*. [cit. 2013-05-08]. Available at WWW: <http://www.ivoa.net/documents/SIA/1>

[11] International Virtual Observatory Alliance: *SSAP*. [cit. 2013-05-08]. Available at WWW: <http://ivoa.net/documents/SSA/>

[12] International Virtual Observatory Alliance: *TAP*. [cit. 2013-05-08]. Available at WWW: <http://www.ivoa.net/documents/TAP/>

[13] International Virtual Observatory Alliance: *Virtual server toolkits*. [cit. 2013-05-08]. Available at WWW: <http://wiki.ivoa.net/twiki/bin/view/IVOA/PublishingInTheVONew>

[14] International Virtual Observatory Alliance: *VO Registry*. [cit. 2013-05-08]. Available at WWW: <http://www.ivoa.net/documents/RegistryInterface/>

[15] International Virtual Observatory Alliance: *VOTable*. [cit. 2013-05-08]. Available at WWW: <http://www.ivoa.net/documents/VOTable/>

[16] IVOA Interoperability meeting. International Virtual Observatory Alliance, [cit. 2013-05-08]. Available at WWW: <http://www.g-vo.org/pmwiki/Interop/Interop>

[17] Munipack. [cit. 2013-05-08]. Available at WWW: <http://munipack.physics.muni.cz/>

[18] National Aeronautics and Space Administration: *FITS*. [cit. 2013-05-08]. Available at WWW: <http://fits.gsfc.nasa.gov/fits_documentation.html>

[19] PostgreSQL. [cit. 2013-05-08]. Available at WWW: <http://www.postgresql.org/>

[20] PPMXL catalog. [cit. 2013-05-08]. Available at WWW: <http://irsa.ipac.caltech.edu/Missions/ppmxl.html>

[21] Saada server. [cit. 2013-05-08]. Available at WWW: <http://saada.unistra.fr/saada/>

[22] SPLAT-VO. [cit. 2013-05-08]. Available at WWW: <http://star-www.dur.ac.uk/~pdraper/splat/splat-vo/splat-vo.html>

[23] Strasbourg astronomical Data Center: *Aladin glufile docummentation*. [cit. 2013-05-08]. Available at WWW: <http://aladin.u-strasbg.fr/glu/>

[24] Topcat. [cit. 2013-05-08]. Available at WWW: <http://www.starlink.ac.uk/topcat/>

[25] UCAC4 catalog. [cit. 2013-05-08]. Available at WWW: <http://www.usno.navy.mil/USNO/astrometry/optical-IR-prod/ucac>

Appendix A

List of used acronyms

ADQL Astronomical Data Query Language

API Application Programming Interface

DaCHS Data Center Helper Suite

DAL Data Access Layer

DB Database

DEC Declination

DSA Deep Space Antenna

FITS Flexible Image Transport System

FS File System

GAVO German Astrophysical Virtual Observatory

ICRS International Celestial Reference System

IVOA International Virtual Observatory Alliance

MIME Multipurpose Internet Mail Extensions

N/A Not Applicable, Not Available or No Answer

NASA National Aeronautics and Space Administration

OSPS Ondřejov Southern Photometry Survey

RA Right Ascension

RD Resource Descriptor

SAMP Simple Application Messaging Protocol

SCS Simple Cone Search

SIAP Simple Image Access Protocol

SQL Structured Query Language

SSAP Simple Spectral Access Protocol

TAP Table Access Protocol

URL Uniform Resource Locator

VO Virtual Observatory

VOTable Virtual Observatory Table

XML Extensible markup language

Appendix B

Resource descriptor

The source code of the file q.rd.

```
<?XML version="1.0" encoding="iso-8859-1"?>

<resource schema="bextract" resdir="extract">
 <meta name="title">DK-154 objects</meta>
 <meta name="description">Identified objects on DK-154 surveys</meta>

 <!-- table holding informations about the image, e.g. observation date, band,
  meteorological information, ... -->
 <table id="observation_info" onDisk="True" adql="true" primary="obsname" >

  <meta name="description">Common information (band, dateObs,...) on
                           observations. 1 row per file</meta>

  <column name="accref" type="text"
   description="internal file reference"/>

  <column name="dateObs"  ucd="VOX:Image_MJDateObs"
   type="double precision" unit="d" tablehead="Obs. date"
   verbLevel="0" description="Epoch at midpoint of observation"
   displayHint="type=humanDate"/>
  <column name="HJD"  ucd="time.start;obs"
   type="double precision" unit="d" tablehead="HJD"
   verbLevel="0"
   description="Epoch at midpoint of observation in Heliocentric Julian Date"
   displayHint="type=humanDate"/>
  <column name="band" ucd="VOX:BandPass_ID"
   tablehead="Bandpass" description="Freeform name of the bandpass used"
    type="text" verbLevel="10"/>
  <column name="obsname" type="text"
   description="Identifier for the source file"
   ucd="meta.id;meta.main"/>
  <column name="TEL_ALT" type="double precision"
   description="Horizontal telescope coordinates altitude"/>
  <column name="TEL_AZ" type="double precision"
   description="Horizontal telescope coordinates azimuth"/>
  <column name="AIRMASS" type="double precision"
   description="Airmass of target location"/>
  <column name="PRESSURE" type="double precision"
   description="Atmospheric pressure [mbar]"/>
```

```
40     <column name="TEMPERAT" type="double precision"
41      description="Outside temperature of enviroment [Celsius]"/>
42     <column name="AGON" type="double precision"
43      description="?"/>
44     <column name="ORIRA" type="double precision"
45      description="Original position of the center of the image (RA)"/>
46     <column name="ORIDEC" type="double precision"
47      description="Original position of the center of the image (DEC)"/>
48    </table>
49
50    <!-- table of our observations, one row per one object identified on the image.-->
51    <table id="objobs_incomplete" onDisk="True" primary="obsname,starNo"
52      mixin="//scs#q3cindex" adql="true">
53     <meta name="description">Observations of objects without common information
54      (band, dateObs,...) </meta>
55     <column name="accref" type="text"
56      description="internal file reference"/>
57
58     <column name="obsname" type="text"
59      description="Identifier for the source file"
60      ucd="meta.id;meta.main"/>
61     <column name="starNo" type="integer" required="True"
62      description="Running number of the extracted object"
63      ucd="meta.id"/>
64
65     <column name="raj2000" unit="deg" type="double precision"
66      ucd="pos.eq.ra;meta.main"
67      description="Observed RA of the object"/>
68     <column name="dej2000" unit="deg" type="double precision"
69      ucd="pos.eq.dec;meta.main"
70      description="Main value of declination"/>
71     <column name="FLUX" unit="W/m2"
72      ucd="phot.flux"
73      description="Photon flux"/>
74          <column name="COUNTS" unit="cts"/>
75          <column name="RATE" unit="cts/s/m2"/>
76          <column name="IMAG" unit="mag"/>
77          <column name="PHOTONS" unit="ph/s/m2"/>
78          <column name="PHOTNU" unit="ph/s/m2/Hz"/>
79          <column name="PHOTLAM" unit="ph/s/m2/nm"/>
80          <column name="FNU" unit="W/m2/Hz"/>
81          <column name="FLAM" unit="W/m2/nm"/>
82          <column name="MAG" unit="mag"/>
83          <column name="ABMAG" unit="abmag"/>
84          <column name="STMAG" unit="stmag"/>
85          <column name="COUNTS_ERR" unit="cts"/>
86          <column name="RATE_ERR" unit="cts/s/m2"/>
87          <column name="IMAG_ERR" unit="mag"/>
88          <column name="PHOTONS_ERR" unit="ph/s/m2"/>
89          <column name="PHOTNU_ERR" unit="ph/s/m2/Hz"/>
90          <column name="PHOTLAM_ERR" unit="ph/s/m2/nm"/>
91          <column name="FLUX_ERR" unit="W/m2"/>
92          <column name="FNU_ERR" unit="W/m2/Hz"/>
93          <column name="FLAM_ERR" unit="W/m2/nm"/>
94          <column name="MAG_ERR" unit="mag"/>
95          <column name="ABMAG_ERR" unit="abmag"/>
96          <column name="STMAG_ERR" unit="stmag"/>
97
98    <!-- we decided not to use these columns from the bintable -->
99      <!--  <column name="SKY_COUNTS" unit="cts/arcsec2"/>
100         <column name="SKY_RATE" unit="cts/s/m2/arcsec2"/>
101         <column name="SKY_IMAG" unit="mag/arcsec2"/>
```

```
102         <column name="SKY_PHOTONS" unit="ph/s/m2/arcsec2"/>
103         <column name="SKY_PHOTNU" unit="ph/s/m2/Hz/arcsec2"/>
104         <column name="SKY_PHOTLAM" unit="ph/s/m2/nm/arcsec2"/>
105         <column name="SKY_FNU" unit="W/m2/arcsec2"/>
106         <column name="SKY_FLAM" unit="W/m2/nm/arcsec2"/>
107         <column name="SKY_MAG" unit="mag/arcsec2"/>
108         <column name="SKY_ABMAG" unit="abmag/arcsec2"/>
109         <column name="SKY_STMAG" unit="stmag/arcsec2"/>
110         <column name="SKY_COUNTS_ERR" unit="cts/arcsec2"/>
111         <column name="SKY_RATE_ERR" unit="cts/s/m2/arcsec2"/>
112         <column name="SKY_IMAG_ERR" unit="mag/arcsec2"/>
113         <column name="SKY_PHOTONS_ERR" unit="ph/s/m2/arcsec2"/>
114         <column name="SKY_PHOTNU_ERR" unit="ph/s/m2/Hz/arcsec2"/>
115         <column name="SKY_PHOTLAM_ERR" unit="ph/s/m2/nm/arcsec2"/>
116         <column name="SKY_FLUX_ERR" unit="W/m2/arcsec2"/>
117         <column name="SKY_FNU_ERR" unit="W/m2/Hz/arcsec2"/>
118         <column name="SKY_FLAM_ERR" unit="W/m2/nm/arcsec2"/>
119         <column name="SKY_MAG_ERR" unit="mag/arcsec2"/>
120         <column name="SKY_ABMAG_ERR" unit="abmag/arcsec2"/>
121         <column name="SKY_STMAG_ERR" unit="stmag/arcsec2"/>-->
122 </table>
123
124 <!-- table holding the catalog objects needed for crossmatching-->
125 <table id="objcat" onDisk="True" primary="sourceId"
126   mixin="//scs#q3cindex" adql="True">
127  <meta name="description">Objects imported from PPMXL catalog
128   (used to crossmatch my observations)</meta>
129
130  <column name="sourceId" type="text"
131   ucd="meta.id;meta.main"
132   description="Identifier (Q3C ipix of the USNO-B 1.0 object)"/>
133  <column name="raj2000" type="double precision"
134   ucd="pos.eq.ra;meta.main"
135   description="Catalog RA of the object"/>
136  <column name="dej2000" type="double precision"
137   ucd="pos.eq.dec;meta.main"
138   description="Main value of declination"/>
139  <column name="e_raepRA" type="double precision"
140   ucd="stat.error;pos.eq.ra;meta.main"
141   description="Mean error in RA*cos(delta) at mean epoch"/>
142  <column name="e_deepDE" type="double precision"
143   ucd="stat.error;pos.eq.dec;meta.main"
144   description="Mean error in Dec at mean epoch"/>
145  <column name="pmRA" type="double precision"
146   ucd="pos.pm;pos.eq.ra"
147   description="Proper Motion in RA*cos(delta)"/>
148  <column name="pmDE" type="double precision"
149   ucd="pos.pm;pos.eq.dec"
150   description="Proper Motion in Dec"/>
151  <column name="e_pmRA" type="double precision"
152   ucd="stat.error;pos.pm;pos.eq.ra"
153   description="Mean error in pmRA*cos(delta)"/>
154  <column name="e_pmDE" type="double precision"
155   ucd="stat.error;pos.pm;pos.eq.dec"
156   description="Mean error in pmDE"/>
157 </table>
158
159     <!--table of my object IDs when identified against the catalog-->
160 <table id="objid" onDisk="True" primary="sourceId,obsname,starNo"  adql="true">
161 <meta name="description">Table of IDs of my objects
162  (not particular observations)</meta>
163
```

```
164
165     <column original="objcat.sourceId"/>
166     <column original="objobs_incomplete.obsname"/>
167     <column original="objobs_incomplete.starNo"/>
168
169     <column name="pos_off" unit="deg" type="double precision"
170      description="Position offset (catalog vs. our observation) in degrees"/>
171
172     <foreignKey source="sourceId" inTable="objcat"/>
173     <foreignKey source="starNo,obsname" inTable="objobs_incomplete"/>
174   </table>
175
176   <!--FINAL SCS VIEW on the tables observation_info, objobs_incomplete and objid-->
177   <table id="objobs" onDisk="True" adql="True">
178    <meta name="description">Object observations with additional meta</meta>
179
180    <LOOP listItems=" raj2000
181         dej2000
182         COUNTS
183         IMAG
184         MAG
185         ABMAG
186         STMAG
187         RATE
188         PHOTONS
189         PHOTNU
190         PHOTLAM
191         FLUX
192         FNU
193         FLAM
194         COUNTS_ERR
195         IMAG_ERR
196         MAG_ERR
197         ABMAG_ERR
198         STMAG_ERR
199         RATE_ERR
200         PHOTONS_ERR
201         PHOTNU_ERR
202         PHOTLAM_ERR
203         FLUX_ERR
204         FNU_ERR
205         FLAM_ERR
206         ">
207     <events>
208      <column original="objobs_incomplete.\item"/>
209     </events>
210    </LOOP>
211    <column name="obsname" type="text"
212     description="Identifier for the source file"/>
213    <column name="starNo" type="integer" required="True"
214     description="Running number of the extracted object"/>
215
216    <column original="objid.sourceId"/>
217    <column original="objid.pos_off"/>
218
219    <LOOP listItems=" dateObs
220         HJD
221         band
222         TEL_ALT
223         TEL_AZ
224         AIRMASS
225         PRESSURE
```

```
226         TEMPERAT
227         AGON
228         ORIRA
229         ORIDEC">
230      <events>
231       <column original="observation_info.\item"/>
232      </events>
233    </LOOP>
234
235    <!-- creating the view objobs-->
236          <viewStatement>
237           CREATE OR REPLACE VIEW \curtable AS(
238            SELECT \colNames FROM
239            (\schema.observation_info
240            JOIN
241            \schema.objobs_incomplete
242            USING (obsname)
243            JOIN
244            \schema.objid
245            USING (starNo,obsname)))
246
247          </viewStatement>
248       </table>
249
250       <!-- view for representing the lightcurve via SSAP Protocol-->
251       <table id="objobs_ssap" onDisk="true" adql="true">
252
253    <column name="accref" type="text" tablehead="Product key"
254     description="Access key for the data"
255     verbLevel="1" displayHint="type=url"
256     utype="Access.Reference"/>
257    <column name="mime" type="text" verbLevel="20"
258     tablehead="Type"
259     description="MIME type of the file served"
260     utype="Access.Format"/>
261    <column name="band" ucd="VOX:BandPass_ID"
262     tablehead="Bandpass" description="Freeform name of the bandpass used"
263      type="text" verbLevel="10"/>
264
265    <column name="ssa_dstitle" type="text" required="True"
266     utype="ssa:DataID.Title" ucd="meta.title;meta.dataset"
267     tablehead="Title" verbLevel="15"
268     description="Title or the dataset (usually, spectrum)"/>
269    <column name="ssa_pubDID" type="text"
270     utype="ssa:Curation.PublisherDID"
271     tablehead="P. DID" verbLevel="15"
272     description="Dataset identifier assigned by the publisher"/>
273    <column name="ssa_location" type="spoint"
274     utype="ssa:Char.SpatialAxis.Coverage.Location.Value"
275     ucd="pos.eq"
276     verbLevel="5" tablehead="Location"
277     description="ICRS location of aperture center" unit="deg,deg"/>
278    <column name="ssa_publisher" type="text" required="True"
279     utype="ssa:Curation.Publisher"
280     tablehead="Publisher" verbLevel="25"
281     description="Publisher of the datasets included here."/>
282
283
284    <column name="min_date" type="double precision"
285     utype="ssa:Char.TimeAxis.Coverage.Location.Value" ucd="time.epoch"
286     unit="d"
287     verbLevel="5" tablehead="Start Date Obs."
```

```
288      description="The first date of observation"
289      displayHint="type=humanDate"/>
290     <column name="max_date" type="double precision"
291      utype="ssa:Char.TimeAxis.Coverage.Location.Value" ucd="time.epoch"
292      unit="d"
293      verbLevel="5" tablehead="End Date Obs."
294      description="The last date of observation"
295      displayHint="type=humanDate"/>
296
297
298     <!-- creating the view objobs_ssap from view objobs-->
299      <viewStatement>
300      CREATE OR REPLACE VIEW \curtable AS (
301       SELECT (
302          select ('http://vos2.asu.cas.cz/tap/sync?LANG=ADQL&amp;REQUEST=doQuery&amp;
303          FORMAT=fits&amp;QUERY='|| 'select * from bextract.objobs where sourceid = '''
304          || sourceid || ''' and band = ''' || band || ''''')) as accref,
305       'application/fits' as mime,
306       band as band,
307       'ASU CAS lightcurve' as ssa_dstitle,
308       sourceid as ssa_pubDID,
309       (SELECT  spoint (obs1.raj2000 *pi()/180.0 , obs1.dej2000  * pi()/180.0  ))
310       as ssa_location,
311       'ASU CAS' as ssa_publisher,
312       (SELECT min(hjd) from
313        (select distinct hjd from
314           \schema.objobs as obs2 where obs2.sourceid=obs1.sourceid) as subq)
315        as min_date,
316       (SELECT max(hjd) from
317        (select distinct hjd from
318           \schema.objobs as obs2 where obs2.sourceid=obs1.sourceid) as subq)
319         as max_date
320
321      FROM \schema.objobs as obs1)
322        </viewStatement>
323       </table>
324
325
326
327    <!-- data element for importing the observation_info table.
328     Drops the table before ingesting. -->
329    <data id="import_observation_info" dependents="merge_observation_data" auto="False">
330     <sources recurse="True" pattern="data/*.fits"/>
331
332     <fitsProdGrammar>
333      <rowfilter procDef="//products#define">
334       <bind key="embargo">parseTimestamp(@DATE_OBS.split(".")[0])
335                           +datetime.timedelta(days=365)</bind>
336       <bind key="owner">"beusers"</bind>
337       <bind key="table">"\schema.observation_info"</bind>
338        </rowfilter>
339     </fitsProdGrammar>
340      <make table="observation_info">
341       <rowmaker id="make_observation_info" idmaps="*">
342        <map dest="obsname">str.join("/", \inputRelativePath.split('/')[2:])</map>
343        <map dest="dateObs">dateTimeToMJD(parseTimestamp(@DATE_OBS.split(".")[0]))</map>
344        <map dest="HJD">@JD_HELIO</map>
345        <map dest="band">vars["FILTB"]</map>
346        <map dest="accref">\inputRelativePath</map>
347        <map dest="ORIRA">hmsToDeg(@ORIRA , sepChar=":")</map>
348        <map dest="ORIDEC">parseAngle(@ORIDEC , "dms", sepChar=":")</map>
349
```

```
350       <apply name="Halpha">
351          <code>
352         if vars["FILTB"]=="H-alpha_narrow":
353           vars["FILTB"] = "Ha-r"
354          </code>
355       </apply>
356      </rowmaker>
357    </make>
358  </data>
359
360  <!-- data element for importing the table objobs_incomplete.
361    Drops the table before ingesting. -->
362  <data id="import_observations" dependents="merge_observation_data" auto="false">
363   <sources recurse="True" pattern="data/*.fits"/>
364   <fitsTableGrammar/>
365    <make table="objobs_incomplete">
366     <rowmaker id="make_objobs_incomplete" idmaps="*">
367      <map dest="obsname">str.join("/", \inputRelativePath.split('/')[2:])</map>
368      <map dest="starNo">\rowsProcessed</map>
369      <map dest="accref">\inputRelativePath</map>
370      <simplemaps>raj2000: RA, dej2000:DEC</simplemaps>
371     </rowmaker>
372    </make>
373  </data>
374
375  <!-- data element for updating the table observation_info -->
376  <data id="update_observation_info" original="import_observation_info"
377        updating="True" auto="true" >
378   <sources recurse="True" pattern="data/*.fits">
379    <ignoreSources fromdb="select accref from bextract.observation_info" />
380   </sources>
381  </data>
382
383  <!-- data element for updating the table objobs_incomplete -->
384  <data id="update_observations" original="import_observations"
385        updating="True" auto="true">
386   <sources recurse="True" pattern="data/*.fits">
387    <ignoreSources fromdb="select accref from bextract.objobs_incomplete"/>
388   </sources>
389  </data>
390
391  <!-- data element for downloading the data from catalog as votable. -->
392  <data id="get_catalog_data"  auto="false">
393  <make table="objcat">
394
395   <script type="preImport" lang="python">
396    import subprocess
397    with base.getTableConn() as conn:
398     coordinates= list( conn.queryToDicts(
399      "SELECT DISTINCT obs1.orira,obs1.oridec FROM bextract.observation_info as obs1 " +
400      "JOIN" +
401      "(SELECT observation_info.orira,observation_info.oridec " +
402      "FROM bextract.observation_info) as obs2
403       ON q3c_join(obs1.orira, obs1.oridec, obs2.orira, obs2.oridec, 0.016666667);"))
404     print coordinates
405
406     Query = ("SELECT TOP 10000000 ipix, raj2000, dej2000, e_raepRA, e_deepDE,
407          pmRA, pmDE, e_pmRA, e_pmDE " +
408               "FROM ppmxl.main WHERE ")
409     Query += ("OR ".join(["(1=
410       CONTAINS(POINT('ICRS',ppmxl.main.raj2000,ppmxl.main.dej2000)," +
411         "CIRCLE('ICRS', %(orira).14f , %(oridec).14f, 0.25)))"%x for x in coordinates]))
```

```
412
413       subprocess.call("curl -FLANG=ADQL -FREQUEST=doQuery
414                       -FQUERY=\"%s\" -FFORMAT=\"votable/td\" \
415           http://dc.g-vo.org/tap/sync -o \"data/ppmxl_needed.XML\""% (Query),shell=True)
416     </script>
417   </make>
418
419   </data>
420
421   <!-- data element for ingesting the catalog from votable file. -->
422   <data id="import_catalog"  auto="false" >
423    <sources pattern="data/ppmxl_needed.XML"/>
424    <voTableGrammar/>
425    <make table="objcat">
426     <rowmaker id="make_objcat" idmaps="*">
427      <map key="sourceId" source="ipix"/>
428     </rowmaker>
429    </make>
430   </data>
431
432   <!-- data element used for crossmatching the objects with catalog-->
433   <data id="cross_match"  auto="false" dependents="merge_observation_data">
434    <make table="objid">
435     <script type="preIndex" lang="SQL" name="Crossmatch objcat with objobs">
436      INSERT INTO \schema.objid (
437       SELECT sourceId,obsname,starNo,pos_off FROM (
438        SELECT *,
439        <!--this is distance between our observation and catalog's.
440         Counted by simple Pythagoras-->
441        (|/ ((catalog.raj2000-observation.raj2000)^2 +
442        (catalog.dej2000-observation.dej2000)^2)) AS pos_off
443         FROM
444         (SELECT cat.sourceId,cat.raj2000,cat.dej2000 FROM \schema.objcat AS cat) AS catalog,
445         (SELECT obs.starNo,obs.obsname,obs.raj2000,obs.dej2000
446            FROM \schema.objobs_incomplete AS obs) AS observation
447         WHERE q3c_join(observation.raj2000, observation.dej2000,
448          catalog.raj2000, catalog.dej2000,
449          0.000277778)
450       ) as q
451      );
452     </script>
453    </make>
454   </data>
455
456   <!-- data element for merging the data into objobs view -->
457   <data id="merge_observation_data">
458    <make table="objobs"/>
459   </data>
460
461   <!-- data element for creating the SSAP view from objobs view -->
462   <data id="make_ssap">
463    <make table="objobs_ssap"/>
464   </data>
465
466   <!-- database core for the SCS service using objobs view -->
467   <dbCore id="scsCore" queriedTable="objobs">
468      <condDesc original="//scs#humanInput"/>
469     <condDesc original="//scs#protoInput"/>
470     <FEED source="//scs#coreDescs"/>
471     <condDesc buildFrom="band"/>
472
473    <condDesc combining="True">
```

```
474     <inputKey name="date_min" type="date"  ucd="pos.eq.date"
475      description="Minimum date (If empty, returns everything until Maximum date)"
476      tablehead="Minimum Date">
477     </inputKey>
478
479     <inputKey name="date_max" type="date" ucd="pos.eq.date"
480      description="Minimum date (If empty, returns everything until Maximum date)"
481      tablehead="Maximum Date">
482     </inputKey>
483
484     <phraseMaker id="DatePhrase" name="dateSQL">
485      <code>
486       if (inPars["date_min"]):
487        minTS = dateTimeToMJD(datetime.datetime.combine(
488         inPars["date_min"],
489         datetime.datetime.strptime("0:0:0", "%H:%M:%S").time()))
490       else:
491        minTS = 0.0
492
493       if (inPars["date_max"]):
494        maxTS = dateTimeToMJD(datetime.datetime.combine(
495         inPars["date_max"],
496         datetime.datetime.strptime("23:59:59", "%H:%M:%S").time()))
497       else:
498        maxTS= dateTimeToMJD(datetime.datetime.now())
499
500       yield "dateObs BETWEEN %%(%s)s AND %%(%s)s"%(
501        base.getSQLKey("date_min", minTS, outPars),
502        base.getSQLKey("date_max", maxTS, outPars))
503      </code>
504     </phraseMaker>
505    </condDesc>
506    </dbCore>
507
508    <!-- database core for SSAP service using the objobs_ssap view -->
509    <dbCore id="ssaCore" queriedTable="objobs_ssap">
510     <condDesc buildFrom="ssa_location"/>
511    <condDesc buildFrom="band"/>
512
513    </dbCore>
514
515   <!-- service for SCS -->
516   <service id="scs" core="scsCore" allowed="scs.XML, form">
517    <meta name="title">Simple cone search in the binary table extension</meta>
518    <meta name="shortName">Bintable SCS</meta>
519    <meta name="testQuery.ra">149.416138018881</meta>
520    <meta name="testQuery.dec">0.415292172381177</meta>
521    <meta name="testQuery.sr">0.01</meta>
522   </service>
523
524   <!-- service for SSAP -->
525   <service id="ssa" core="ssaCore" allowed="ssap.XML,form">
526    <meta name="shortName">DK-154 light curves</meta>
527    <meta name="description">SSAP interpreted light curves</meta>
528   </service>
529  </resource>
```

Appendix C

Data examples

In this chapter are enlisted some data file examples.

Listing 4: VOTable structure.

```
SIMPLE  =                    T / conform to FITS standard
BITPIX  =                   32 / unsigned short data
NAXIS   =                    2 / number of axes
NAXIS1  =                 2148 / length of data axis
NAXIS2  =                 2048 / length of data axis
EXTEND  =                    T / this is FITS with extensions
HISTORY Created with RTS2 version 0.9.4 build on Oct  8 2012 11:05:08.
CTIME   =           1357700226 / exposure start (seconds since 1.1.1970)
USEC    =               388661 / exposure start micro seconds
JD      =     2456301.62298611 / exposure JD
DATE-OBS= '2013-01-09T02:57:06.388' / start of exposure
OBJECT  = '99942   '           / object name
EXPOSURE=                   30. / exposure length in seconds
EXPTIME =                   30. / exposure length in seconds
INSTRUME= 'DFOSC_FASU'         / name of the data acqusition instrument
TELESCOP= 'DK-1.54 '           / name of the data acqusition telescope
ORIGIN  = 'ASU CAS - NBI'      / organization responsible for data
FOC_NAME= 'FASC    '           / name of focuser
UTSTART = '2013-01-09T02:57:06.388'
EQUINOX = '2000.   '
CCD_TYPE= 'E2V44-82'           / camera type
```

Listing 5: FITS Raw data

```
COMMENT === Astrometric Solution by Munipack ===
COMMENT Type: absolute
COMMENT Reference catalogue: UCAC4 Catalogue (Zacharias+, 2012)
COMMENT Projection: GNOMONIC
COMMENT Number of objects used = 412
COMMENT RMS =    348.1E-03 [arcsec]
COMMENT Scale =     0.3962148723     +-   2.1E-02 [arcsec/pix]
COMMENT cos(pa) =   -0.9999504532 +-  -8.2E-03
COMMENT sin(pa) =   -0.0099544538 +-  -8.2E-01
COMMENT Position Angle (pa)  =  180.5703576118 +-   4.7E+01 [deg]
COMMENT Alpha center projection (CRVAL1) =   17.0770261431 +-   5.1E-03 [deg]
COMMENT Delta center projection (CRVAL2) =  -72.5417937886 +-   1.9E-03 [deg]
COMMENT Horizontal center (CRPIX1) =  1074.000 [pix]
COMMENT Vertical   center (CRPIX2) =  1024.000 [pix]
COMMENT Catalogue RA,DEC [deg]        Data X,Y [pix]     Residuals [arcsec]
COMMENT   16.78505620 -72.51274280  266.021  760.823     3.5E+00     3.5E+00
COMMENT   17.26649030 -72.53821920 1591.985 1000.217  -689.4E-03    -5.2E+00
COMMENT   17.07079000 -72.51833140 1054.126  811.279   257.3E-03    39.8E-03
COMMENT   16.87286120 -72.51271730  513.720  766.331   104.3E-03    11.7E-03
COMMENT   17.19225030 -72.62536120 1394.129 1780.816    32.2E-03  -132.4E-03
COMMENT   17.16164500 -72.52949340 1304.178  909.805  -199.6E-03   291.9E-03
COMMENT   16.76438450 -72.46051870  211.742  297.170  -428.6E-03  -408.3E-03
COMMENT   16.80444300 -72.47590870  322.344  434.923    11.1E-03  -134.8E-03
COMMENT   17.06155300 -72.62464840 1039.246 1777.780    97.9E-03  -215.8E-03
COMMENT   16.75966590 -72.59286340  216.084 1499.182   -11.4E-03   -30.4E-03
COMMENT   16.81920770 -72.44286120  359.186  134.733  -293.9E-03  -417.9E-03
COMMENT   17.30662740 -72.44465310 1693.814  135.889   264.5E-03  -245.2E-03
COMMENT   16.95130060 -72.56559370  733.254 1244.004   273.0E-03   -82.6E-03
COMMENT   17.04826650 -72.52176170  993.163  842.835   214.9E-03    -2.7E-03
COMMENT   16.74724090 -72.56489370  179.251 1245.687  -280.8E-03  -206.5E-03
COMMENT   16.98505390 -72.54621250  824.276 1067.030  -300.5E-03  -202.4E-03
COMMENT   16.84688650 -72.44256840  434.882  130.798  -208.7E-03  -458.7E-03
```

Listing 6: FITS Reduced data

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                  208 / width of table in bytes
NAXIS2  =                 1227 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                   26 / number of fields in each row
TTYPE1  = 'RA      '           / label for field   1
TFORM1  = '1D      '           / data format of field: 8-byte DOUBLE
TUNIT1  = 'deg     '           / physical unit of field
TTYPE2  = 'DEC     '           / label for field   2
TFORM2  = '1D      '           / data format of field: 8-byte DOUBLE
TUNIT2  = 'deg     '           / physical unit of field
TTYPE3  = 'COUNTS  '           / label for field   3
TFORM3  = '1D      '           / data format of field: 8-byte DOUBLE
TUNIT3  = 'cts     '           / physical unit of field
TTYPE4  = 'IMAG    '           / label for field   4
TFORM4  = '1D      '           / data format of field: 8-byte DOUBLE
TUNIT4  = 'mag     '           / physical unit of field
TTYPE5  = 'MAG     '           / label for field   5
TFORM5  = '1D      '           / data format of field: 8-byte DOUBLE
TUNIT5  = 'mag     '           / physical unit of field
TTYPE6  = 'ABMAG   '           / label for field   6
TFORM6  = '1D      '           / data format of field: 8-byte DOUBLE
```

Listing 7: FITS Binary table

Appendix D

Installation of GAVO DaCHS

The GAVO DaCHS installation proccess is very well described in the official documentation [2] and all the needed information is there. What I will describe here is an concrete example of our file structure at the virtual server running at `http://vos2.asu.cas.cz`.

We have installed it from sources, but it can be easily installed from the package too, as described in the install docs [2].

D.1 Sources

After the installation, sources can be found in the */usr/share/pyshared/gavo* folder and can be debugged in there, if a problem occurs.

D.2 Database

The database is installed in the default folder at */usr/share* and it's configuration files are at */etc/postgresql/8.4/pgdata*. The database itself is stored in the home folder of the user which administrates the server at */home/voadmin/gavodb*.

The user with the privileges to run the database and data ingestion is *voadmin* and plays the role of the administrator of the whole server.

All the options we had to change was the localhost ip address, because on a virtual server it obviously isn't the standart *127.0.0.0*. It had to be changed in files */var/gavo/etc/dsn* file and in */etc/postgresql/8.4/pgdata/pg_hba.conf* file.

D.3 Filesystem

The gavo system files are stored in */var/gavo* folder. For us are important the config files in *etc* folder, and the logs in *log* folder. The RD files are stored in *inputs* folder. In the *extract* folder is my RD for extracting the light curves, in *dk154_rawdata* and *dk154_reduced* are my colleague's RDs for publishing the images.

The actual data is mounted in the location */emc/archdata*, and the folders which we need are linked to the folders of our descriptors for easy referencing.

Appendix E

Ingesting manual

The ingestion will be done typically by one "administrator", who will look after the server. However, the usage of the ingestion mechanism is quite simple and user friendly. I will describe the list of commands the ingestion consists of.

Gavo server supports variety of commands, but for the ingestion itself we need only one. The command is *gavo imp* and the arguments, that follow are all from my solution. I will describe here all the commands I support and need, concerning the ingestion of data to the server.

gavo imp - Basic server command. Displays help and options we can use, but for our solution only few of them are reasonable.

gavo imp -c - This option we will use the most. When it encounters an erroneous input file, it does not stop, but continues until it checks all of them.

gavo imp -m - Updates only metadata of our table. Usable, when we need to change some description of a column and don't want to touch the data.

gavo imp "filepath" - The *filepath* argument is the name of the RD which represents the biggest part of my solution. It is typically named *q.rd*, so the command is *gavo imp q*. This command alone does automatically ingestion of new files found in the data folders. It only adds new data, does not touch the data already ingested. However, it only writes the data, and does not the crossmatching. When we want to rewrite the tables, or do the crossmatching, we have to add other parameters. These are the names of the data elements within the

RD and are used to define ingesting mechanisms - typically one data element per a table.

gavo imp q import_observation_info - This script recreates the first table - the one that holds information about the observations, one row per file. The ingestion is quite fast, typically minutes.

gavo imp q import_observations - Imports the actual data of the observations by extracting it from the binary table extension. Here we typically work with thousands of rows per one file the ingestion time depends of how many objects were identified on these images. Ingestion can be slow, typically hours.

gavo imp q get_catalog_data - This script downloads the catalog data needed for the crossmatching into a file. It has also the import_catalog element "chained up", so that one does not need to be called in order to ingest the data from the file to the database. This script typically takes tens of minutes.

gavo imp q import_catalog - This command ingests the catalog from the file to the database. This script typically takes minutes.

gavo imp q crossmatch - Script for the actual crossmatching. Fills the table objid and identifies objects on objobs view against a catalog. This script is very fast - typically seconds.

gavo imp q merge_observation_data - This script creates the views. It is typically called within the "gavo imp q" command. But we need to call it after the crossmatching, in order to update the views. As this script does not change any data, it is instant.

Appendix F

Userguide for services

F.1 Web-based

The web-based access is a simple form. There is one special URL per a service running on the server, their list is below. The SIAPform is not part of my work, but I mention it too for complete information.

- SCS - `http://vos2.asu.cas.cz/extract/q/scs/form`
- SSAP - `http://vos2.asu.cas.cz/extract/q/ssa/form`
- SIAP raw - `http://vos2.asu.cas.cz/dk154_rawdata/q/sia/form`
- SIAP reduced - `http://vos2.asu.cas.cz/dk154_reduced/q/sia/form`

Their usage of the web-forms is quite clear, so I will describe only one of them. On the first image F.1 there is a simple example of how to fill in the web form. After sending the query, we can display the result and use the options available for the result table (e.g. send via SAMP, display as a simple plot, ...).

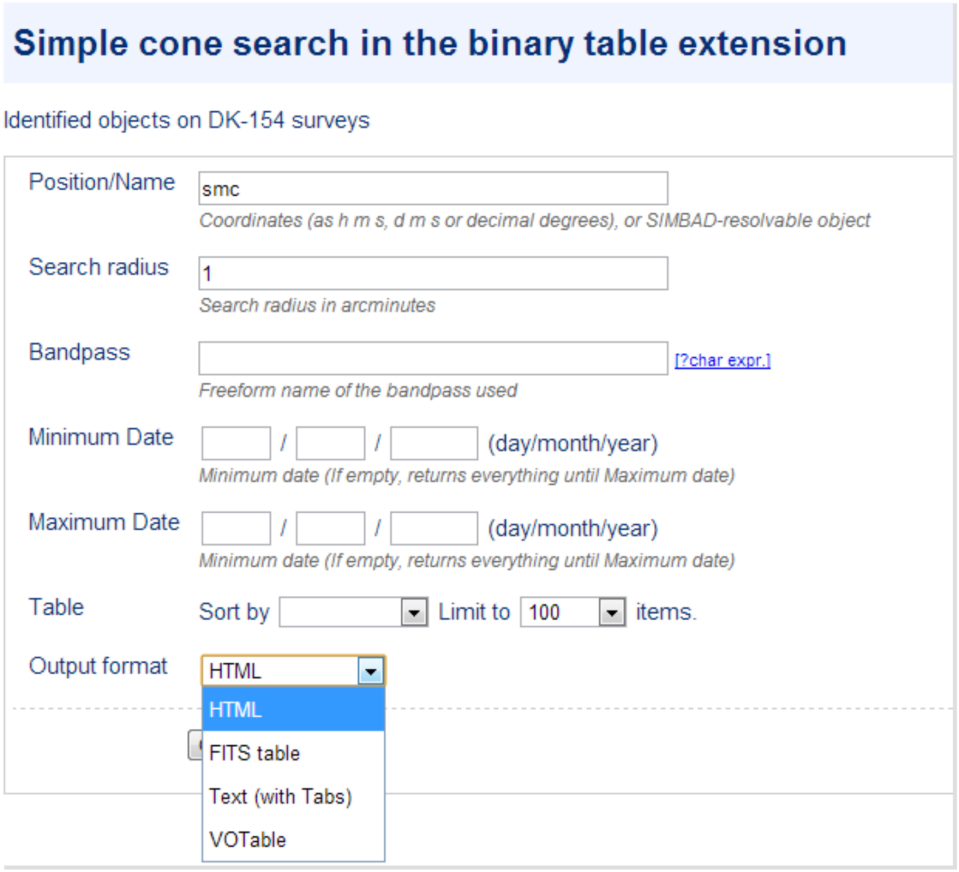


Figure F.1: Web-form example.

F.2 Aladin

Here I will describe a short way to display our SCS data via Aladin.

1. We start by clicking the open button in the upper right corner.

2. Next we can select the *DK 154 folder* of services, and select the *DK154 SCS* service.

3. On the image F.2 we can see the form which Aladin uses for adding the input parameters to the query. We can fill in the data accordingly to its example.

4. On the next image F.3 we can see the output of our query visualised on the sky by Aladin.

5. When we want to display the image as a background to our objects, we can send a query to the SIAP endpoint instead of SCS endpoint as described on step 2.

6. Then, we have to move the image downwards on the planes list to the right on the Aladin window as on the image F.4

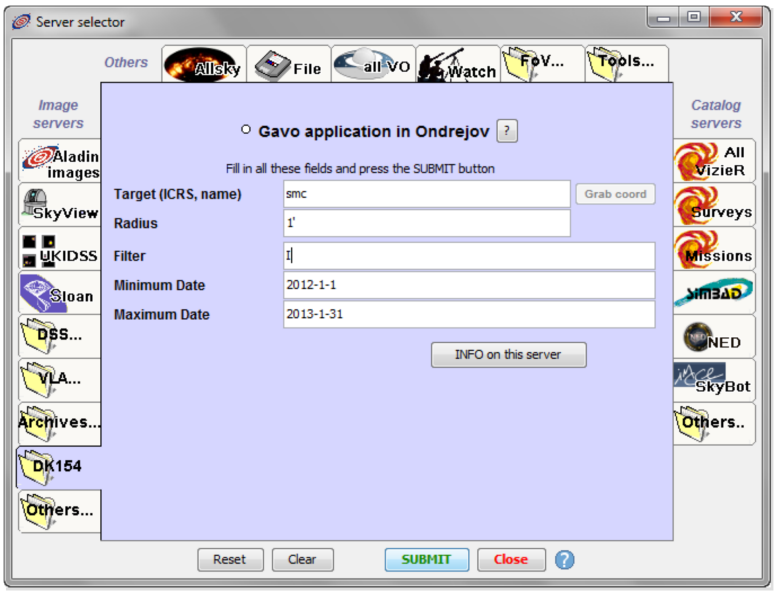


Figure F.2: Enter the query parameters.

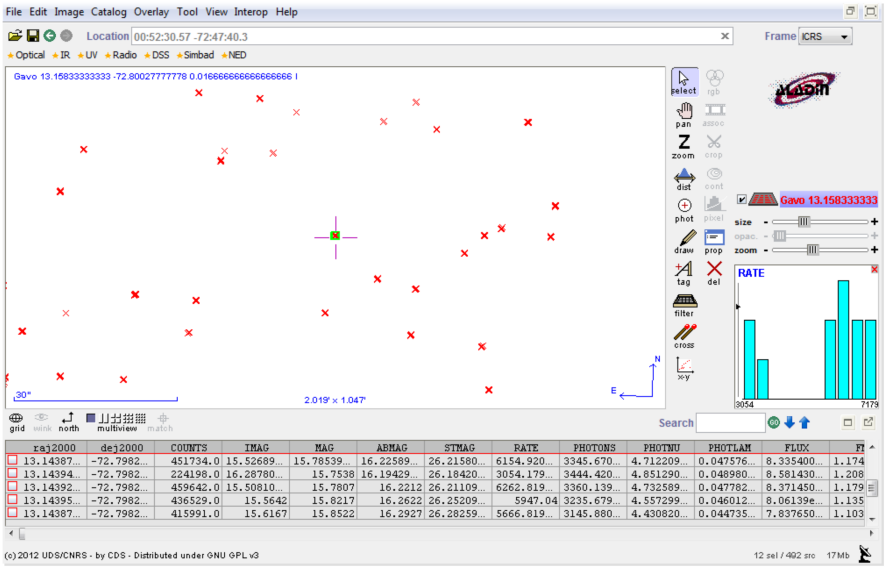


Figure F.3: Checkout the result, we can select objects on the image.

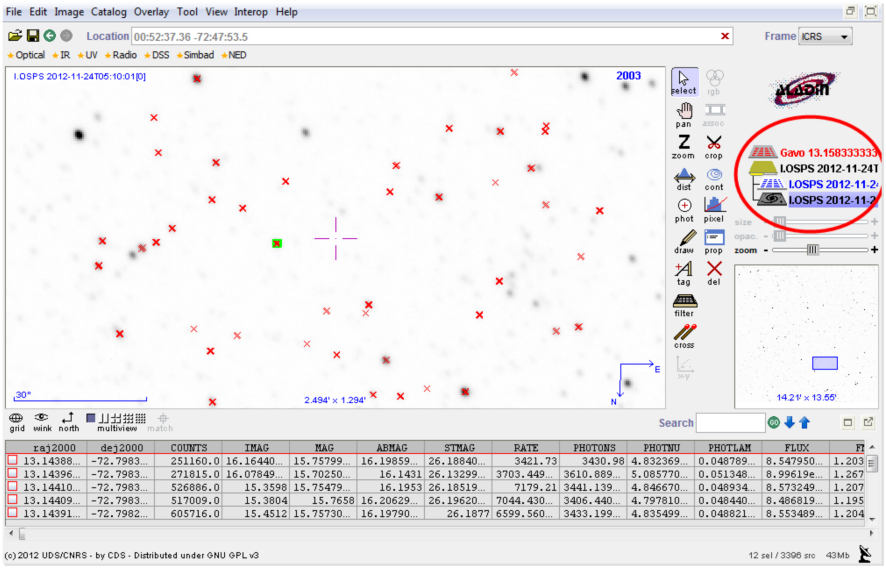


Figure F.4: Move the image downwards in the planes list.

F.3 SPLAT-VO

In SPLAT-vo, we don't have we have to add the server manually. Otherwise is the extraction standard, the walkthrough is described below.

1. First, we select the SSAP icon on the top panel in the SPLAT-VO window. This will open a window with VO servers providing any sorts of SSAP protocol.

2. Then we have to add our own server by clicking the option Add New Server button. Here we fill in the information as on the picture F.5.

3. On the other screen we can then enter the query by typing for example "SMC" to the *Object* box, then clicking Lookup. That fills the coordinates for us. The *Radius* is clear, but because of the SSAP definition, the *Band* is defined by its limits in meters. When entering the symbolic names we can bypass it, but we have to enter it as an interval too. I/I means exactly I bandpass. Since we don't have much observations yet and we want as many as we can get, we leave the *Time* parameter empty.

4. When we send the query, we have to set up the x and y parameters to hjd (heliocentic date) and mag (magnitude) as on picture F.7. Then we can display the light curve by clicking Display spectra as highlighted on the picture.

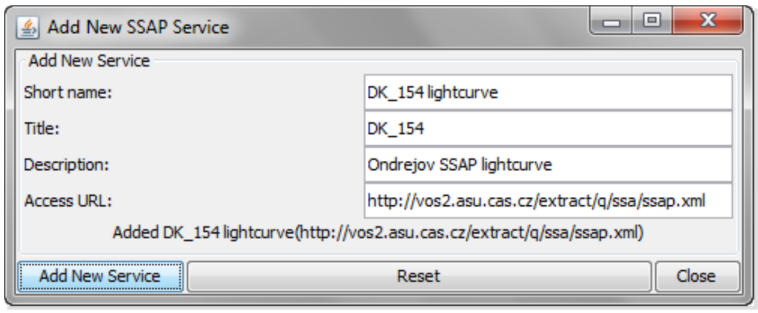


Figure F.5: Add our server to the list.

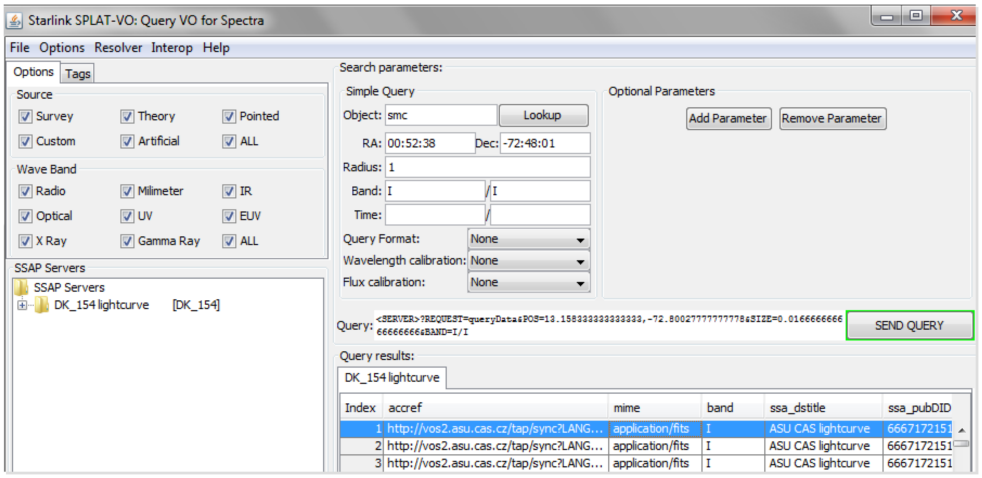


Figure F.6: Fill in the query parameters.

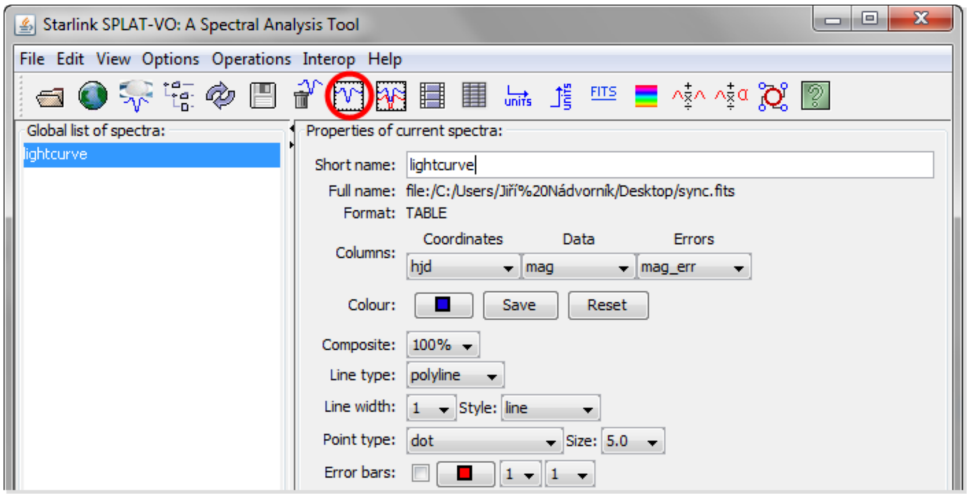


Figure F.7: Display the light curve.

Appendix G

CD contents

- `readme.txt` ..... brief description of the CD content
- `clients` ..... Folder with runnable clients
  - `Aladin.jar` ..... Aladin with the glufile included
  - `topcat-full.jar` ..... Newes version of Topcat.
  - `splat.jnlp` ..... SPLAT-VO web-start
- `src`
  - `impl` ..... source codes of implementation
    - `q.rd` ..... Resource Descriptor file
    - `glufile` ..... glufile embedded in th Aladin.jar
  - `thesis` ..... source form in format LaTeX
- `text` ..... text of the thesis
  - `thesis.pdf` ..... text u PDF