Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Information Technology

Department of Software Engineering

Bachelor's thesis

VO-compatible post-processing proxy server of stellar spectra

Tomáš Peterka

Supervisor: Dr. P. Škoda PhD.

21st May 2012

Acknowledgements

I would like to thank Petr Škoda for his patience and enthusiasm. Special thanks belongs to Markus Demleitner from Astronomisches Rechen-Institut (ARI) of Heidelberg University for his kindness, immense help and valuable advices.

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.

In Prague on 21st May 2012 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tomáš Peterka. *VO-compatible post-processing proxy server of stellar spectra: Bachelor's thesis*. Czech Republic: Czech Technical University in Prague, Faculty of Information Technology, 2012.

Abstract

Amount of astronomic data increases exponentially with doubling rate every 6-9 months. The efficient handling and analysis of data spread around the world is a main goal of International Virtual Observatory Alliance (IVOA). Aim of this thesis is to extend service for publishing spectra with post-processing ability according to the latest IVOA standards.

Keywords IVOA, SSAP, server, spectral, post-processing, cutout, scale

Abstrakt

Množství atronomických dat exponenciálně roste. Každých 6 až 9 měsíců se objem dat zdvojnásobí. Efektivní správa a analýza dat rozprostřených po celém světě je hlavním cílem Sdružení pro mezinárodní vituální observatoř (IVOA). Cílem této práce je rozšířit službu pro poskytování spekter o schopnost zpracování spekter podle požadavku klienta. Služba musí odpovídat nejnovějších standardům vydaných IVOA.

Klíčová slova IVOA, SSAP, server, spektra, ořez, škálování

Contents

List of Figures

List of Tables

Introduction

Motivation and objectives

The phenomena of data avalanche is mostly visible in astronomy. As more telescopes are being built, the amount of data increases exponentially with doubling rate every 6–9 months.(5) In one of the biggest astronomical repository The Sloan Digital Sky Survey (SDSS) there are measurements for nearly 500 million stars and galaxies, and spectra of nearly two millions objects[1], all publicly available.

Spectral data are the most useful when studied together. Therefore astronomers need to understand and analyse huge sets of data from many repositories at once. International Virtual Observatory Alliance (IVOA) sets the standards for global interoperability of all astronomical data archives.(6)

The main purpose of this work is to create server tools for providing and post-processing spectra with usage of new Virtual Observatory standards. This work uses spectra and resources of the Astronomical Institute of the Academy of Sciences of the Czech Republic in Ondřejov.

This thesis is divided into three main parts: analysis and design, implementation, and conclusion. In the beginning of analysis chapter, there is a short and comprehensive overview of astronomical formats, history, technologies and present state. Second part of analysis is dedicated to feasibility study, requirements and sketch of our own solution. In implementation chapter, there is mentioned unfinished project of ingestor. Then is shown process of implementing DaCHS extension and its testing. Usage chapter provides guide how to use the extension and summarize improvements which it brings. In conclusion chapter you will find list of all achieved goals and future of this work.

[1] `http://www.sdss.org/`

Chapter 1

Analysis and design

Astronomers have been using one format called FITS for more than 30 years. It was really well designed and is still in use. Now there is a flood of data volumes produced by powerful ground and space-based instrumentation. The real advantage of this is that there are many observations of one object from different telescopes and in different times.

The old style of astronomy work consisted of obtaining and analysing individual observations from different archives. The data were downloaded from proprietary server via web form which was not sufficient for advanced filtering. Then they were manually converted to unique form (e.g. converting units and data formats) so the astronomer could use them for research. Big obstacle was to find these servers because there was not any list of available archives. Most of the servers were written by astronomers so each piece is unique. There is a possibility to unite proprietary servers with proxy service, or they can be replaced with more mature and standardized technology.

The ideal situation would be when all spectra are publicly available and there are mature tools for accessing them in unified manner. Then scientists can easily cooperate and analyse celestial objects in an innovative way not yet done before. Another big and neglected advantage of this mindset is that it is possible for almost everybody with sufficient knowledge to do top research without investments to technology. Ordinary people can download free tools and easily get real data. This could be new and interesting field of science and it can lead to new discoveries.

This idea could be realized through Virtual Observatory. Virtual Observatory aims to allow global access to world-wide spread astronomical archives. Virtual Observatory also aims to enable analysis of huge sets of data through standard set of advanced tools.

1.1 Notions and definitions

1.1.1 Virtual Observatory

The International Virtual Observatory Alliance (IVOA) was formed in June 2002 with a mission to "facilitate the international coordination and collaboration necessary for the development and deployment of the tools, systems and organizational structures necessary to enable the international utilization of astronomical archives as an integrated and interoperating virtual observatory." The IVOA focuses on the development of standards and encourages their implementation for the benefit of the worldwide astronomical community. (4)

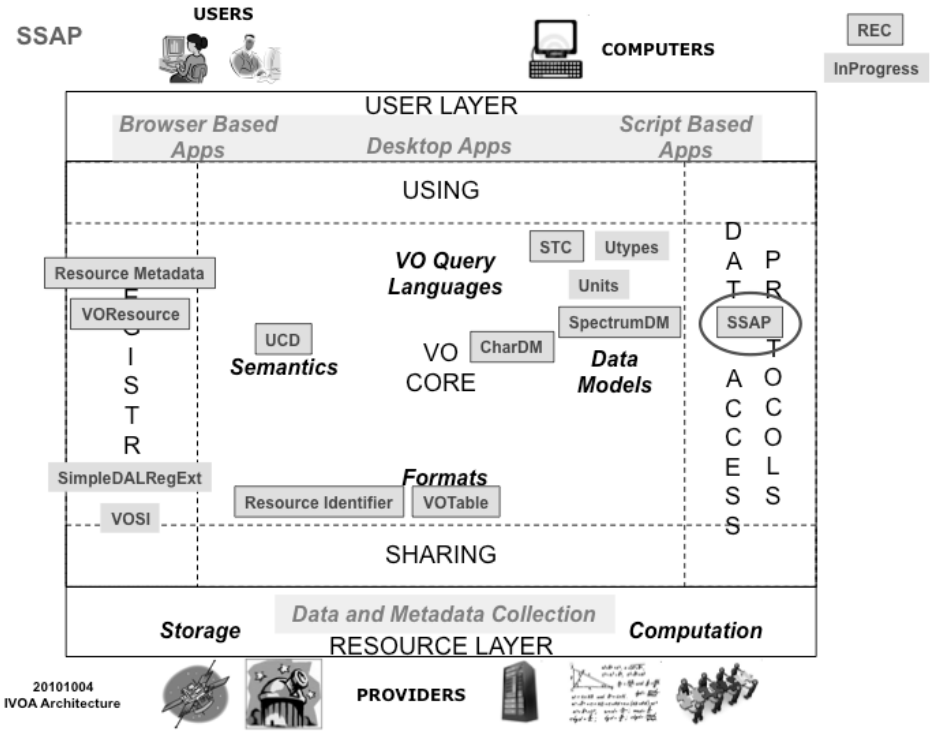


Figure 1.1: IVOA architecture

1.1.2 Simple Spectral Access Protocol

At this point it is worth to remark that I have worked only with spectra but astronomy deals with much more. Beside spectra there are e.g. data cubes, light curves, event list and data obtained in different way than via telescopes (e.g. neutrino detectors). Another worth-to-notice thing is that each astronomical branch has their own protocol. For example images are distributed

through Simple Image Access Protocol (SIAP). As the default protocol for spectra was chosen Simple Spectral Access Protocol. This protocol is currently version 1.1. This version contains only specification of `queryData` with sketch of `getData` It is obvious that the protocol is still under development and during this thesis we have provided many proposals and comments to the next version.

The protocol is HTTP based and specification defines parameters and their behaviour. Protocol has main guide-post — parameter called REQUEST (each argument must be upper-case). REQUEST can be `queryData` or `getData`. QueryData selects spectra from all available spectra on a queried server. GetData still has not been specified but it will be able to describe details about a downloaded spectra.

The list below contains only four parameters from many possible. They are valid only in combination with `REQUEST=queryData` First two parameters are mandatory for understanding how the searching works. The rest is optional but important for this thesis.

- *POS* (mandatory) means coordinates of target star. The coordinates are comma separated pair where first item is RA (right ascension) and the second one is DEC (declination). The coordinates must be written as decimal numbers. This is quite against conventions because astronomers are used to write coordinates in different formats. Right ascension is mostly written in HMS format (hours:minutes:seconds). The value tells us how far is the object we seek from vernal point on celestial equator. Complete circle with 360° has 24 hours in HMS format. That means every hour is 15° and the value is always positive.
  Declination is height above the celestial equator. Objects which are above equator have positive declination. It is mostly used in DMS format (degrees:minutes:seconds). For example Polaris has DEC about 89.3° what is $+89 : 1 : 12.0$ in DMS.

- *SIZE* (optional) defines diameter of the search region specified in decimal degrees.

- *BAND* (optional) restricts spectral axis (wavelength). It can have two meanings. If it is single number it means that queried spectra must contain this point. Second meaning is interval. Interval can define lower bound, upper bound or both. Units are meters. Astronomers are used to Ångströms, thus the most common format of band is $angstroms \times 10^{-10}$.

- *FLUXCALIB* (optional) tells how should be flux axis calibrated. Possible values are “absolute”, “relative”, “normalized” and “any” (the default)(2)

1.1.3 Data formats

Flexible Image Transport System (FITS)

FITS was released in 1981 and is commonly used in astronomy. The reason why it is still alive is a good design. Each file is composed from a header and data. Header is in human-readable ASCII form. Header is composed from cards. These card corresponds to the old punch cards. It means that each card is 80 characters wide(without trailing new-line character). Cards are grouped into *blocks* which have 36 cards each. With 1 byte per character the sum is 2880 bytes per *block*. Each card has form of `keyword = value / comment` where key can have up to 8 characters. In column 9 there is always `=` sign (columns are counted from 1). The value can be anything encoded to ASCII. Comment is optional. There are special keywords COMMENT and HISTORY but they are not significant. Size of data is written in header. There are many formats of data. It is main problem of this format. Common representation of data is so-called 1D image where there is only one row of data. Other values can be inferred from information in header.
More sane format is called binary table. Table in sense that it has columns and rows. Meta-data like column names and units are still written in header. But there is no need to calculate any data from information in header. This format is better to work with because of possibility to process it in stream. The main disadvantage of all FITS files is lack of semantics because there are only few keywords standardized.

VOTable

New format of data presented by IVOA, VOTable is designed as a flexible storage and exchange format for tabular data, with particular emphasis on astronomical tables. Interoperability is encouraged through the use of standards (XML). The XML fabric allows applications to easily validate an input document, as well as facilitating transformations through XSLT (eXtensible Style Language Transformation) engines. VOTable is an unordered set of rows, each of a uniform structure, as specified in the table description (the table meta-data). Each row in a table is a sequence of table cells, and each of these contains either a primitive data type, or an array of such primitives. VOTable is modelled on the FITS Table format; VOTable was designed to be close to the FITS Binary Table format.(3)
VOTable is composed from parameters and fields. Parameters are constants common for all spectra from a observatory. Fields describe columns in data table at the end of file. Data can be encoded in many formats but the most proper is binary data encoded using BASE64. VOTable exceed in semantic information about data and possibility to be processed in stream. Since it has strong semantic it can be used not only for description of one spectra, but it is returned by VO compatible spectra server as a list of found spectra.

1.1.4 Ondřejov 2m Telescope Archives

There are so far two main archives of stellar spectra observed with 2m telescope of Ondřejov observatory. The first is called CCD700 and contains so far almost 10 thousand of spectra and is still growing as the spectrograph creates spectra files in 1D FITS format every clear night. Second is a spectra archive called HEROS with about 2000 spectra acquired during the Ondřejov team part of observing time of Heidelberg Extended Range Optical Spectrograph (HEROS) connected to Ondřejov 2m telescope (years 2000-2003). As the HEROS project in Ondřejov was finished, the archive is frozen and all its modifications concern only the reprocessing of raw data or additional post-processing. Namely all spectra in archive are continuum normalized manually and put in separate folder.

1.2 Feasibility study

Feasibility study is divided into two parts. First part is a list of available server toolkits with proper implementation of SSAP. At the end there is a conclusion with discussion which option from the list is best for continuation of this thesis.

1.2.1 Existing SSA Server Toolkits

In VO there are currently more server toolkits already working with VOTables and SSA protocol. Most of them are older servers updated for SSAP. The criteria of selection are

- *implementation of latest VOTable and SSAP* – since our solution should be reference implementation it is crucial to use latest technologies.
- *extensibility of protocols* – the support of SSA protocol is obvious but if server toolkit will be modified to serve as a proxy than it will be needed to have more protocols to communicate with proprietary servers.
- *ingesting capability* – server toolkit is intended as reference service for Ondřejov data to propagate SSA and VO among stellar community. If propagation should be successful it has to offer more than the others. Ingesting capability is still uncommon. The reason for this is that each spectrograph and observatory has its own method of processing spectra thus different files. No one has designed general tool for indexing these files into database. Another reason for missing tool is that there was no standardized database schema. This will change with using VOTable.

It was found that no currently available server toolkit supports optional parameters of SSAP version 1.1. This is because SSAP specification is still

incomplete and under development thus it would be waste of time to implement things which could change. All listed server toolkits except DALserver were tested live at Ondřejov. DALserver was not because of the old version was not worth trying. New version is very promising but because of time schedule for this thesis it was impossible to consider it.

DALserver

- State: Old but robust server toolkit. New version will be soon.
- Author: Doug Tody, main creator of SSA protocol. The server toolkit is now maintained by VAO in USA.
- Technologies: Java (Apache Tomcat 5.5), MySQL 4.1
- Ingesting mechanism: yet none but it is planed

The last stable version of DALserver is 0.9. released in 2008. It probably does not support SSAP version 1.1. neither VOTable version 1.2. New version should come out very soon but there are no guaranties for that.
The server toolkit is well written and well documented. Therefore on first look it is easy to extend. A disadvantage is that DALserver itself cannot ingest FITS files hence it can not fill the database with data. You need a prefilled database with all mandatory keyword from SSAP. Only thing that DALserver can do is to perform basic computations and conversions of data from database. Mr. Tody has promised that in version 1.0. there will be database ingestion implemented. SSA Proxy has been made from this server to one of the biggest spectral database Sloan Digital Sky Survey [2]

Pleinplot

- State: Old and charged with features. New version unlikely.
- Author: Philippe Prugniel, Lyon
- Technologies: mainly written in C with benefits of FORTRAN and PERL
- Ingesting mechanism: feature-rich tool for importing data

Pleinpot is not just a server. It is complete set of tools for general analysis of catalogs and images. Installation archive has approximately 15GB. This is because it includes database of celestial objects. This server can create schema in the database, ingest data into database and perform many post-processing operations on spectra. The biggest disadvantage is that this server

[2] `https://webtest.aoc.nrao.edu/ivoa-dal/`

is fully written in C and other low level languages and it is very complicated (it contains about 1500 modules). It is almost impossible to modify the code. Development of this server is frozen in field of Virtual observatory because of changes of authors priorities.

GAVO DaCHS

- State: Under active development. VO friendly
- Author: Markus Demleitner from Astronomisches Rechen-Institut (ARI) of Heidelberg University.
- Technologies: Python and XML, PostgreSQL with PgSphere
- Ingesting mechanism: part of the toolkit. Almost brilliant design. It is possible to define new columns, compute columns, set constants and values from different source than FITS file.

Surprisingly great server. It is very well documented. The server has ability to ingest data into database from many types of sources. This is because some parts of data description can be Python code hence it is possible to connect to server during ingesting. Only the crucial parts of ingesting process are left on astronomer so ingesting routine definition is relatively simple. Basic configuration is via XML with optional nested python code. On the first look is the configuration difficult although there is great documentation. We recommend to contact directly Mr. Demleitner for help in initial phase.
The server can operate with more protocols at once. Server handles not even SSAP queries but also TAP and SIAP queries. Protocols and other definitions are done with XML too. These XML definitions are serialized into Python objects and then run. It is an advantage because it is easy and fast to extend the server when programmer has overview about all possible objects.

VODance

- State: Brand new product. Still under heavy development.
- Author: Marco Molinaro from Centro Italiano Archivi Astronomici
- Technologies: Java (Apache Tomcat), Django, MySQL.
- Ingesting mechanism: nothing. There is django admin for manual setting of meta data to columns in existing database. It is also possible to define type of columns via assigning utype and ucd. There are possible conversions from database to java types and some computations.

Server uses Django only for administration. Main logic is done in Java. Server is still under development hence it will be hard to work on extensions and keep

up to date and not to spoil anything. It is hard to speak about disadvantages because the product is not complete. In the version we tested there was not the cone search. Query to database cut out square and then the square was cropped to cone shape. It is slow and memory consuming process.
This project is promising and it is worth to wait for stable version. Final release is planned to be distributed as a virtual machine file.

Our own solution

Our own implementation was one of the possible solutions as well. The advantage was that it would meet all our expectations. Of course the price would be a lot of time invested into solving problems which someone other has solved already.

1.2.2 Best option discussion

In all listed servers (except DaCHS) disadvantages overweighted advantages so we have chosen our own solution. This decision was made because DaCHS was discovered in later part of this thesis, and caused development of few subprojects which were stopped afterwards. First I started to work on script for indexing files into PostgreSQL called ingestor. I also prepared requirements and analysis of architecture of future server. During the work we have discovered that GAVO DaCHS was updated with SSAP capabilities. It impressed us with digesting mechanism which is similar to the one which was written early in the beginning of this thesis. Of course it has many other advantages described in the list above so it is the winner.

1.3 Requirements

Main request was to implement reference service with spectra post-processing located in Ondřejov.

1.3.1 Operating environment

Final product will run on latest Debian linux in virtual server. As a database was chosen PostgreSQL because of many plugins for cone search.

1.3.2 Functional requirements

1.3.2.1 Database preparation

Schema has to have ability to store various information about spectra eg. wavelength start and end point, target object name, target coordinates, time of observation etc. Database will be queried with usage of spatial queries and spherical trigonometry. Therefore coordinates must be in usable format.

Every data column must have meta attributes like name, ucd[3] , utype[4] and others required by VOTable.

1.3.2.2 Ingestor

Collection of tools for

- *schema creating* – Schema map can be partially generated from example FITS file. The application is expected to extract meta-data from given file and construct initial concept of data table. Table has to contain (relative) *source file path*, *embargo* for hiding private spectra, *owner* of the spectra and a unique *hash* for all local spectrographs. It should propose mapping from common FITS keywords to SSA names and types[5]. User will be able to edit these mappings and define conversions, computations, and constants in simple programming language (ideally Python). Final map should be serialized into text-based file for possible later manual modifications. Constants should be stored in the database because the ingesting script is planed to be independent on the server and vice versa. Every spectrograph will have its own schema map. Application will be able to generate SQL command for creating database schema from schema map.

- *data importing* – Simple console application intended to be periodically executed by CRON. It will import new spectra from filesystem into database using schema map. It has to be able to distinguish spectrograph by input arguments. During the import each file should be converted into SDM compliant FITS format[6].

- *schema clearing* — Script for cleaning all data related to a spectrograph. It has to dro all tables and delete all related files.

1.3.2.3 SSA server

Server will implement SSA protocol in version 1.1. List of crucial features

- *Scalability of protocol* – it has to be easy to modify protocol. Protocols should be stored as static definitions in a markup language. Main reason is that SSAP is still evolving and major changes are still expected.

- *VOTable generation* – server's response is always valid VOTable. Does not matter which protocol is used. Of course when serving static files it does not convert them.

[3]UCD Unified content descriptor `http://www.ivoa.net/Documents/latest/UCD.html`
[4]`http://www.ivoa.net/Documents/latest/UtypeListCharacterisationDM.html`
[5]This mapping will be provided in official spreadsheet by IVOA.
[6]Spectral Data Model `http://www.ivoa.net/Documents/latest/SpectrumDM.html`

- *Authentication and authorization* – it is possible to hide private spectra based on date or a key. No client software supports authentication but this will change in near future (e.g. in SPLAT-VO is the restricted access to spectra already being tested).

1.3.2.4 Proxy server

- *Pluggable protocols* – the server should communicate on one side with many proprietary archives and on the other side via latest SSAP. This requires protocol definitions separated from server logic. Optionally in a markup language.
- *Post-processing* – an ability to perform mathematical operations on spectra. Mainly two operations are required.
  - cutout – operation when wavelength (spectral axis) is cropped to concrete interval.
  - scale – operation on flux axis when all values are shifted near 1. This operation is not substitution for normalization. This operation is intended to enable quick comparison of many spectra (e.g. line profile changes).

1.3.3 Testing

Server has to communicate with the most used VO clients — *Splat-VO* and *VO-Spec*. All available formats (except the native one) have to be processed with these clients — which means that clients have to be able to visualise and analyse these spectra.
All spectra files have to correspond to SDM standards. This means that every FITS file has to be in binary table format and contain keywords which are specified by IVOA. The resulting FITS should contain standardized meta data and all the original meta data except invalid ones. The crucial attributes of each spectra are *utype* and *ucd* which describe data columns.

1.4 Analysis

There were two possibilities how to fulfil requests. It could be all-in-one application or divided into smaller pieces. First architecture in Figure 1.2

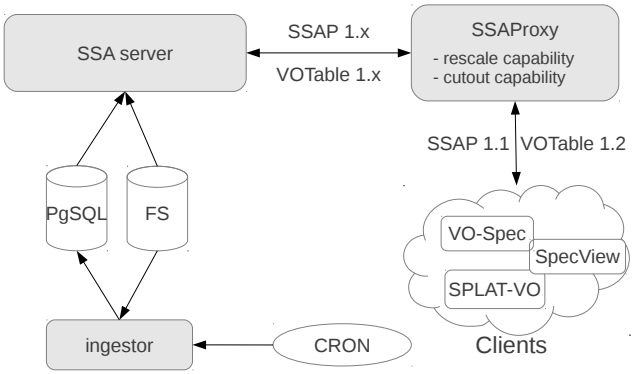


Figure 1.2: Proposed infrastructure 1

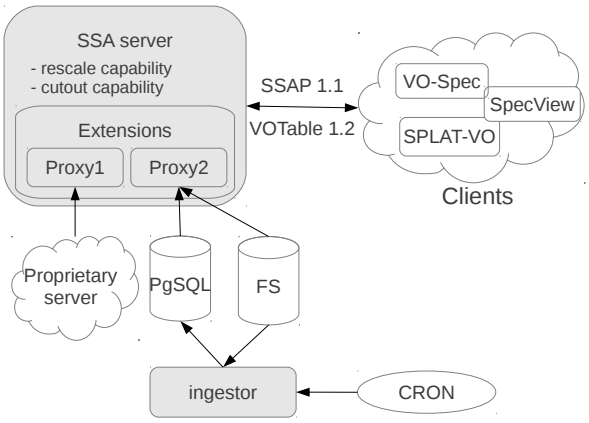


Figure 1.3: Proposed infrastructure 2

has proxy server independent of a base server which communicates with the database. This leads to easy deployment of the proxy server in front of any server. This solution is on first look very good but it has few disadvantages. First is that SSAP response requires meta data which could change with post-processing. We cannot modify meta data to correspond to post-processed spectra (e.g. correct spectral length, filesize) because of concrete files will be downloaded afterwards and there is not any other choice how to figure the meta data out. If we want to normalize data we probably would need to

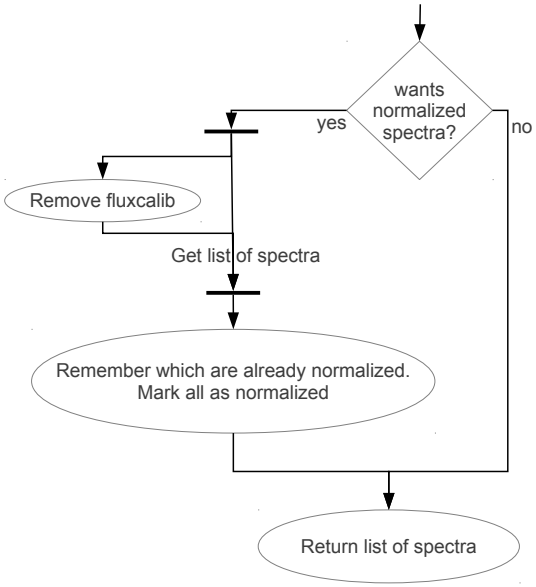


Figure 1.4: Normalization problem

get two list of spectra. List of all spectra (even unrectified) and normalized spectra to avoid normalization on already normalized spectra. We need to mark all spectra as normalized and remember which files should be processed. This algorithm may be naive and real implementation will be so much more complicated that it will be easier to replace the old server with VO server.

Another disadvantage is that it will take much more time to implement because it will be necessary to implement SSAP client as well. With these obstacles in mind we have to say that the second architecture in Figure 1.3 is better. The development process will not be bound to concrete server which can change because it is not standardized. On top of it the second architecture provides something what Virtual Observatory still does not have – complete tool for spectra publishing.

In component diagram in figure 1.5 there is indicated data flow for better understanding. Data flow does not belong there but the diagram could be misunderstood without it.

Protocols component is entry point of server. It wraps one of the available protocols. Default protocol is of course SSAP 1.1. Component *protocols* accepts HTTP request and transforms protocol's parameters into inner representation. Because of query protocols are just a set of constraints the inner representation is set of *Condition Descriptors*.

Core controls flows of objects between components. Core holds thread pool and takes care about system resources. When core gets a new set of conditions it starts a thread from the pool. Data flow how the thread operates is visible

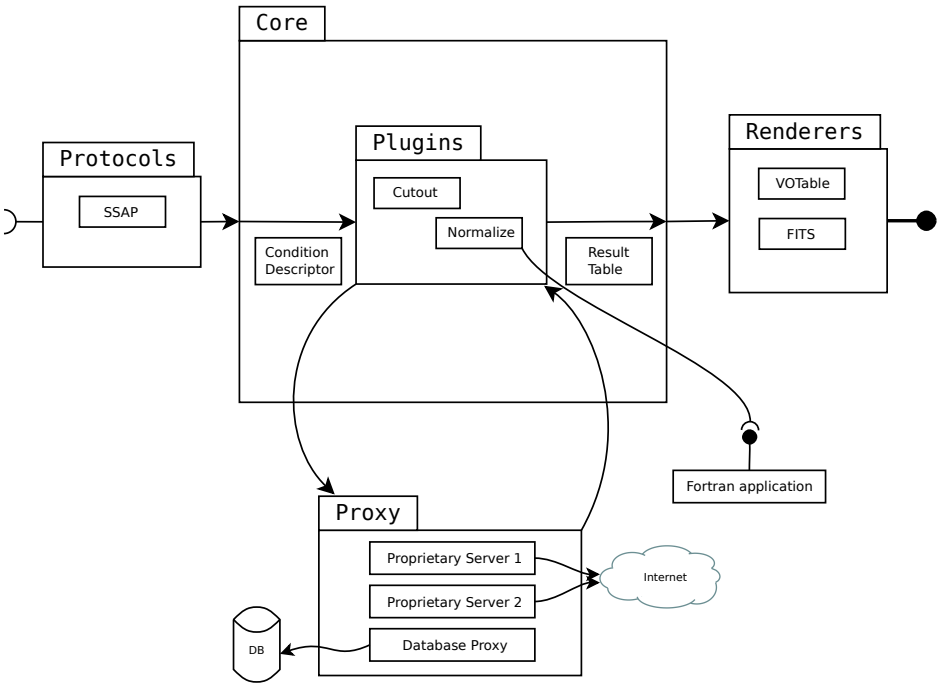


Figure 1.5: Component diagram

from Figure 1.5
Plugins component is closely bounded to the Core. Plugins can modify request and result. They can "sign" the request by modifying conditions and setting special attribute before the request is sent to proxies. The sign persists in response thus the response can be modified according to the sign as well.
Proxy transforms conditions into real data. Each proxy object can map conditions into query which can point to local database or another server at the internet. Proxy objects return *Result Table* or nothing if timeout expires. Wrapper around proxies then merges these tables into one and returns the table back to the *Core*.
Renderers component takes the *Result table* and renders results in requested format to the client. *Result table* will have similar format as binary table fits. It means that it will contain meta data and data separately.

Chapter 2

Implementation

2.1 Database and ingestor

Database

PostgreSQL was chosen as RDBMS because of cone search plugins. There is a list of available plugins.

- *PgSphere* handles spatial queries and spherical trigonometry. Known disadvantage is that it losses performance in larger databases. Its advantage is well written documentation, widespread usage and ability to perform advanced geometrical tasks.
- *q3c* has similar features as *PqSphere*. It is less widespread and documentation is not very good. But it excels in performance on large collections.

After careful selection was chosen *PgSphere* for its simple usage and good documentation.

Schema is easy since each table is dynamically generated from schema map. The only thing which remained to solve is how to store meta data. This schema is inspired from the one which *Pleinpot* uses. There are two tables for each spectrograph — data table and meta data table. Meta data will be joined to data table on column name. Each schema will be named after corresponding spectrograph.

Ingestor

Ingestor is a console script for indexing FITS files into database.
Requirements

- python 2.5+
- pyfits

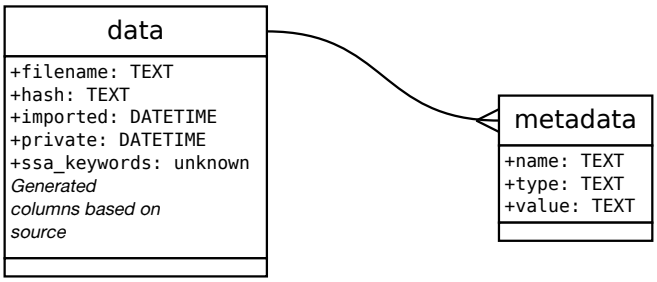


Figure 2.1: Simple database schema

Features

- It can create SQL script for creating tables. The reason why it does not directly create database table is the possibility of editing file before performing SQL CREATE command.

- It can periodically import new files into database. These files must be similar to reference file. The script has inner database of imported files. Script has maps which it uses for computing values or generating new columns which are not obtainable from file.

- It can clean everything with DROP command.

This software is build upon *fitsql* — script by Jan Fuchs from Astronomical Institute in Ondřejov. Although almost whole script was rewritten, the initial non-objective design was kept. DaCHS appeared precisely before major rewriting thus work on this script was stopped.
List of some features which were added to the script

- Configuration with .ini files – There is a global category [global] and local categories named after spectrograph for which it is currently used. Local configuration inherits from [global]. Each spectrograph has unique name, tablename, datasource (path to FITS files), and reference_fit (initial schema map will be build upon this file)

- Automatic table generation with SSA metadata – Dictionary with mapping from ordinary FITS keywords into SSA keywords was added.

- PgSphere integration – Values from FITS file can be mapped to PgSphere objects. This feature can be disabled in configuration.

- User defined computations – For each spectrograph it is possible to define mapping — computations and conversions on input values. Mapping is python dictionary with lambda functions.

Most interesting part is a user input into ingesting routine. Why is the user-specific configuration necessary is shown in the following example: Almost each FITS file has wrong format of key `dateObs` (date of observation). IVOA recommends the time format in ISO 8601. All FITS I worked with had date of observation as a date in format `YYYY-MM-DD` and time of observation in seconds from midnight. In addition the keyword was different for different spectrographs. It is obvious that this mapping cannot be automatically generated and user's intervention is necessary. The user is kept away from routines and writes only main logic. Python was chosen to be most suitable language because it comes up with nice structures and syntax that it is possible to let user write python code directly with almost no programming skills.

```
mapping = (
{
 'keys' : ("CRVAL1", "NAXIS1" ,"CDELT1"),
 'map'  : lambda crval1, naxis1, cdelt1: {
    "key"      : "META_SPEC_MIDDLE",
    "typename" : "datetime",
    "value"    : ((naxis1 - 1) / 2.0 * cdelt1) + crval1,
    "default"  : "NOT NULL"
    }
 },
 {
 'keys' : ("NAXIS1", "CDELT1"),
 'map'  : lambda naxis1, cdelt1: {
    "key"      : "META_SPEC_EXT",
    "typename" : "float",
    "value"    : float((naxis1 - 1) * cdelt1)
    }
 },
)
```

Figure 2.2: Example of mapping

Mapping always parses FITS file and returned values are directly mapped into database. Thus we need to specify input keys from FITS file. They are in item `"keys"`. Output must have at least three items. They are described in item `"map"`, where is a lambda expression which returns dictionary with name of new column, type of value and a value. Lambda expression is necessary

because of computations which are performed. It can not be static definition. Only this form does not constrain user from advanced functions.

2.2 DaCHS

DaCHS is close to be an ideal server. It almost meets architecture described in analysis in Figure 1.3. The major part is defined in markup language and is separated from underlying logic. Therefore it is easy to extend.

Quite a big obstacle is complexity. In revision 2543 it had 138445 lines of code. The server is still signed as unstable what is immoderately careful. The server has more features than all other servers together and looks like most robust software which I have ever met. We suppose the that the test coverage is nearly perfect.

2.2.1 Setting up GAVO DaCHS

GAVO DaCHS is collection of libraries and binary files which can manipulate FITS files, VOTables and serve VO-compatible files via many protocols. It is written in python with usage of many uncommon libraries. The basic installation is well described in the documentation [7] If you are planning maintain the DaSCH server next to other servers it is highly recommended to use `virtualenv`. When setting up postgresql database do not forget to act as user *postgres*. After installation is default `GAVO_ROOT` set to `/var/gavo/` where is a directory `inputs`.

Input directory contains archives with *resource document* (abbreviation RD will be used). The document is generally named `q.rd` and placed under directory named after the archive. For example archive named ccd700 will have RD `/var/gavo/inputs/ccd700/q.rd` Resource document is a combination of XML and Python code. Every XML object will be serialized into python object and run. In RD are defined

- Services
- Definition for tables for metadata
- Importing routine
- Definition for tables for data
- Constraints definitions for metadata table

Service is the part which communicates with clients, performs queries and returns results. It is complex wrapper around a `Core`. Services in RD are represented by tag `service`. They have attributes `id`, to be referenced

[7] http://docs.g-vo.org/DaCHS/install.html

with inside RD, and `allowed` which specifies what type of front-end the service will use. Available front-ends are `ssap.xml` (communicates with SSAP 1.1), `form` (renders HTML form), `siap.xml` (uses SIAP) and many others. More front-ends are allowed to be run together. Name of front-end is the final part of URL. Now we have complete knowledge how the URL is built: `http://web_root/archive/RD/service_id/frontend` (RD has to be without extension.)

Tables are represented with tag `table`. Like all object in RD they have id attribute too. Metadata tables can be easily distinguished from data tables because they have `onDisk` attribute set to `"true"`. It means they are stored in the database. Table can contain more tags `column`. These tags can be directly written into `table` or `mixin` can be used for filling in predefined columns. Columns definition must be wrapped with `STREAM` to be accessible with `mixin`.

Importing routine is the most tricky and crucial part in RD. It is represented by tag `data`. This tag has to have a `grammar` and `make`. Grammar is used for yielding dictionary with parsed data from data source. Make is used for binding parsed values to database columns.(1)

2.2.2 Postprocessing extension to DaCHS

Our implementation of extensions had two stable versions. First version added `SSAPProcessCore` and `SpectralProduct` classes. In the second version there were these classes removed and code was integrated into existing standard classes. This could be done because of final (partially) understanding of server's architecture.

First implemenation was fully functional but naive and bounded to Ondřejov. It did not use inner mechanism of the server. First class added was `gavo.protocols.ssap.SSAPProcessCore`. It was direct child of `SSAPCore` which handles SSAP queries. Since `SSAPCore` was not designed as a superclass some changes had to be made in this class. The only purpose of the new class was to modify returned data - specifically access references of files[8]. The access reference was modified in a way which did not correspond with philosophy of server implementation. This mechanism is already replaced with better solution.
`gavo.protocols.products.SpectralProduct` did all the processing work. It cut spectra and scaled them. More parts of server were updated for new features to enable post-processing. Main changes were done in the section of manipulating FITS files. `SpectralProduct` had many issues. First it did not use parsed data from grammar in RD. It had own parsers. They tried to find source FITS file and guess its format. There were many converters which could succeed with few files but in production they will be useless. These parsers

[8] Access reference is URI where the spectra file is available.

could not replace data parsers from RD. Second it worked only with FITS file renderer. If a client wanted to cut spectra in VOTable it was not possible. Thus it did not fulfil assignment. In addition it did not take care if the spectra were already normalized and just scaled them. This implementation cut spectra, scaled spectra but only in Ondřejov in CCD700 archive.

Second implementation is far better. The modification of access reference is built directly in the standard `SSAPCore`. It do not create any artificial parameters but it just pass SSAP parameters into file access reference. The modification can be enabled and disabled with setting boolean `postprocess` parameter in SSAPCore in RD. `SpectralProduct` was removed and all computations are done in standard `SDMCore`[9]. This implementation comes with numerous advantages. Firstly redundant `SpectralProduct` was removed. Secondly it takes all advantages from RD because `SDMCore` is presented in RD and has to be set up by user. Finally it gained ability to render VOTable because it was the original purpose of the `SDMCore`. Extended `SDMCore` has support for rendering products according to mime type in the database. It can return SDM compliant binary table FITS and VOTable. When returning FITS it tries to find source file and extract header from it. If it finds the file it removes all data-related keywords, add or update SDM keywords and tries to validate the header. Validation is destructive - invalid keyword are removed.

Scaling now checks if spectra are not already normalized. It does not use mathematics for this but only checks if `ssa_fluxcalib` is set to `"normalized"`. Scaling is implemented as division of all flux[10] values by their modus. The computing of modus is interesting problem because it is surprisingly slow. The slowness is not caused by number of steps. It is the same as in mean computing. The main problem is hashing which is always used.

Algorithms were tested on three different files without threading. First algorithm used hash map to store number counts. It was naive implementation with default hash function for dictionaries in Python. Second algorithm used my own basic hash function and static array. This solution was very memory consuming and was not faster than the previous one. It indicates well designed `dict` type in Python.

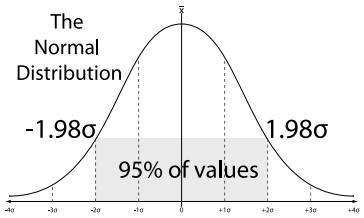


Figure 2.3: Normal distribution

Third algorithm was similar to the second but has improved memory usage with simple usage of statistics. Counting of mean and variance is fast thus they can be used for optimization. I suppose that values on flux

[9]SDMCore can render data in SDM format. It is binary table FITS and VOTable

[10]Spectrum is usually compound from two axis - spectral axis (wavelength) and flux axis with photons count or derived units

Table 2.1: Comparison of modus algorithms

algorithm type	time(relative)	memory usage
naive hash table	1	N/c
array and own hash function	1	N
2σ array	1.4	4σ

axis have normal distribution. With this knowledge and with definition of modus[11] is possible to restrict set of values to values not further than 2σ from the mean because there is 95 % of values. The remaining 5 % are so diverged that they cannot have significant impact on modus. They just widen interval thus consuming memory for nothing.

The c is a coefficient of how many numbers in array are equal. If there are many equal numbers then coefficient is large. If all numbers are the same then $c = N$, where N is the array size.

Cutout was optimized too. Data are stored in the class InMemoryTable which keeps data in linked list. Cutout sometimes needs to remove only few records. There are two possible approaches to this problem. Remove nodes which we do not want or build a new linked list with wanted nodes. If we compare the worst case when only the last one should be preserved then we get for the first case

$$1 + 2 + 3 + ... + n - 1 = \frac{n \times (n-1)}{2} = O(n^2) - O(n) = O(n^2)$$

In C/C++ it would be $O(n)$ but there are not pointers, only iterators. The only way to delete node from linked list is by index. This causes repeated passing from the beginning to wanted position. For the second case with appending wanted values at the end of linked list we get

$$\sum_{0}^{n-1} 1 = n - 1 = O(n)$$

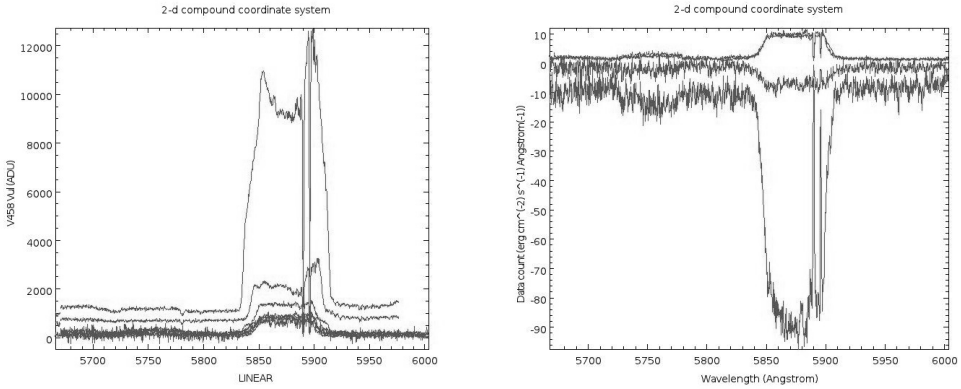
Second case is better and this is how it is implemented.

Scale had one issue and that was negative values which made fake peaks. These scaled data are used in for searching of emission line stars, which are distinguishable with high peaks in one wavelength. This is how the mistake was discovered.

2.2.3 Data preprocessing

Part of the testing was a development of new SSA-compatible archives of Ondřejov spectra. There are two archives. Newer and still growing CCD700 and older, closed, named HEROS.

[11] most common value



(a) (b)

Figure 2.4: Original and wrongly scaled spectra

CCD700 produce data in form of almost valid 1D FITS file. Data preparation consisted only from setting up right importing routine. 1D FITS means that the only row in the data table is filled with FLUX values. Spectral axis values are derivable from metadata (supposing equidistant interval of wavelength — using rebinning). They contain NAXIS1 which implies number of records, CRVAL1 is the wavelength in angstroms of the reference pixel CRPIX1 (in our case the first) at spectral axis and CDELT1 is difference between pixels at spectral axis.

```
SIMPLE  =                    T / Fits standard
BITPIX  =                  -32 / Bits per pixel
NAXIS   =                    1 / Number of axes
NAXIS1  =                 1997 / Axis length
DATE-OBS= '2001-12-09'         / UTC date start of observation
TM_START=                 8715 / 02:25:15
TM_END  =                 9347 / 02:35:47
UT      = '02:25:15'           / UTC of  start of observation
RA      = '5:21:57.4'          / 5.365944
DEC     = '41:47:53.9'         / 41.798306
CTYPE1  = 'LINEAR  '
CRVAL1  =     4269.51301773849
CRPIX1  =                   1.
CDELT1  =    0.127108484224366
```

Figure 2.5: Example of CCD700 FITS meta data

HEROS data were older and more dirty. Each FITS file should have `RA` and `DEC`. These information were hidden in comments in HISTORY block which is hard to parse.

Luckily these information were in an old database handled by Pleinpot. We decided to insert these information back to FITS files. The update script was complex because it merged RA and DEC from database and datetime of observation and observers' names from enclosed CSV into corresponding FITS.

HEROS has normalized and original spectra. It was needed to add sign of normalization into database. SSA protocol has key `FLUXCALIB` which can be "absolute", "relative", "normalized" and "any". It is optional argument therefore it was not implemented in DaCHS. This problem was initial stimulation for rewriting scale because it could scale these normalized spectra what is wrong.

2.3 Testing

DaCHS comes with its own testing framework based on unittest library. It extends the unittest library for some features related to accessing serialized resources like `getTestRD()` and `getTestTable()`. The main test class is `gavo.helpers.testhelpers.VerboseTest` which is subclass of `unittest.TestCase`. This class is used for integration tests as well. DaCHS itself has very high test coverage.

2.3.1 Unit testing

Doctests were the most used tests while implementing the extension because they are independent of any testing framework thus easy to use. Some constraints distinguish them from unittests therefore they can not substitue standard unittests. Doctests are special ability of python. They are written di-

```
def _parseBand(self, band):
    '''Return tuple with interval based on BAND
    >>> sdm = SDMCore(None)
    >>> sdm._parseBand("1000e-10")
    (None, None)
    >>> sdm._parseBand("1000e-10/")
    (1000.0, None)
    >>> sdm._parseBand("1000e-10/1400e-10")
    (1000.0, 1400.0)
```

Figure 2.6: Doctest example

rectly in documentation of the source code. Python can be run as follows: `python -m doctest -v ssap.py` performing all doctests in given file.

2.3.2 User testing

User testing corresponds to the specification in the section 1.3.3.

Two main clients were tested. VO-Spec, java client executable using Java Web Start, and Splat-VO which is standalone Java application. VO-Spec had problem with units of flux axis. The VO specifies that units has to be in *SI* format but data in Ondřejov have not flux units convertible to *SI*. The solution was to chose the nearest SI units possible. VO-Spec refused UCD as well. This error is because of mistake in SSA specification. It propose `em.wl` as the spectral UCD which is invalid according to the UCD specification. This mistake is known and in next version of UCD specification will be fixed.

Splat-VO is less pedantic therefore all formats were accepted.

The SDM compliance cannot be objectively verified because there is not any validator. The only possible test is to open rendered file and manually check if all mandatory fields are presented in the correct form.

Chapter 3

Usage

3.1 Using of postprocessing extensions

To enable the extension, set the `postprocess` parameter of `SSAPCore` to `"true"`.

```
<service id="pssa" allowed="ssap.xml">
    ...
    <ssapCore queriedTable="data" postprocess="true">
        <FEED source="coreDef"/>
    </ssapCore>
</service>
```

Then it is recommended to create more access references for each file. You have to tell the client which formats you provide. For each mime-type create extra row in database with unique accref and mime-type. This can be done in a grammar within `data` tag. Our solution is shown below.

```
<data id="import">
<sources recurse="true" patterns="data/*.fit"/>

<fitsProdGrammar>
 <rowfilter procDef="//products#define">
  <bind name="table">"\schema.data"</bind>
 </rowfilter>
 <rowfilter name="addSDM">
  <code>
  baseAccref = os.path.splitext(row["prodtblPath"])[0]
  row["prodtblAccref"] = baseAccref+".fit"
  row["prodtblMime"] = "image/plain"
  yield row
  row["prodtblAccref"] = baseAccref+".fits"
```

```
  row["prodtblPath"]="dcc://\rdIdDotted/mksdm?"+baseAccref+".fits"
  row["prodtblMime"] = "application/fits"
  yield row
  row["prodtblAccref"] = baseAccref+".vot"
  row["prodtblPath"]="dcc://\rdIdDotted/mksdm?"+baseAccref+".vot"
  row["prodtblMime"] = "application/x-votable+xml"
  yield row
  </code>
 </rowfilter>
</fitsProdGrammar>

 <make table="data">
    <rowmaker idmaps="*">
        ...
```

There are created three different access references for each file. First accref points to normal product. Others point to DCCProduct which is the main gate to SDMCore thus cut out and scale service. You can see that each accref differs only with extension. This is probably best way how to do this. Do not forget to use the right accref as a parameter to DCCProduct. Mime type is crucial for renderer and only `application/fits` and `application/x-votable+xml` are possible outputs of SDMCore. Be sure that you have set proper conversion from access reference to absolute path to file in embededGrammar in SDMCore. This is one of bad things in DaCHS that it does not have separated filename and access reference in db.

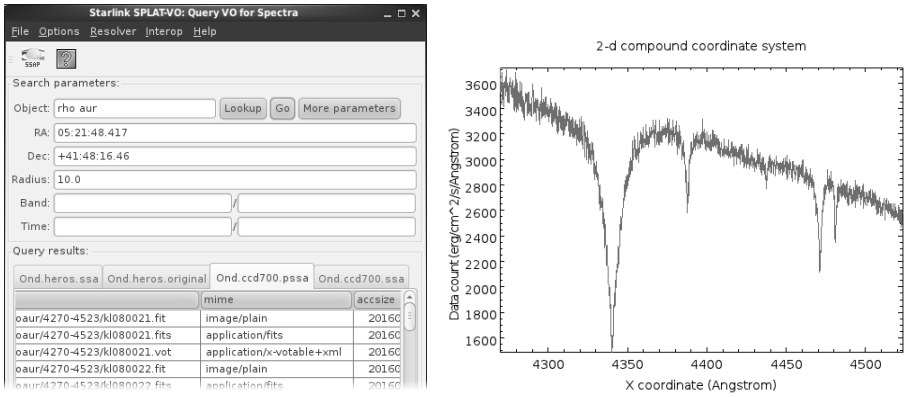
The reference services of our work are implemented in the Astronomical Institute of the Academy of Sciences of the Czech Republic in Ondřejov Access URLs are
`http://ssaproxy.asu.cas.cz/ccd700/q/pssa/ssap.xml`
`http://ssaproxy.asu.cas.cz/HEROS/q/pssa/ssap.xml`
CD700 archive does not have so far pre-normalized spectra, but the rescaling of flux happens on-the-fly. HEROS has all spectra manually normalized, but unrectified data are still presented. Normalized and unrectified spectra are distinguishable by accref because of normalized spectra starts with "NORM" and the others with "UNRECT". Normalized spectra has also NORM as a prefix in their name.

Behaviour of these archives differs. Cutting service works on both as expected. It is possible to define open intervals as well. Scaling is activated when client adds `FLUXCALIB=normalized` into SSAP query. HEROS returns only normalized spectra as is written in SSA recommendation. CCD700 has not any normalized spectra thus it returns all spectra matching other conditions and scale them when they are downloaded.

This behaviour does not conform to SSA recommendation since it defines FLUXCALIB parameter as following: "FLUXCALIB specifies the minimum

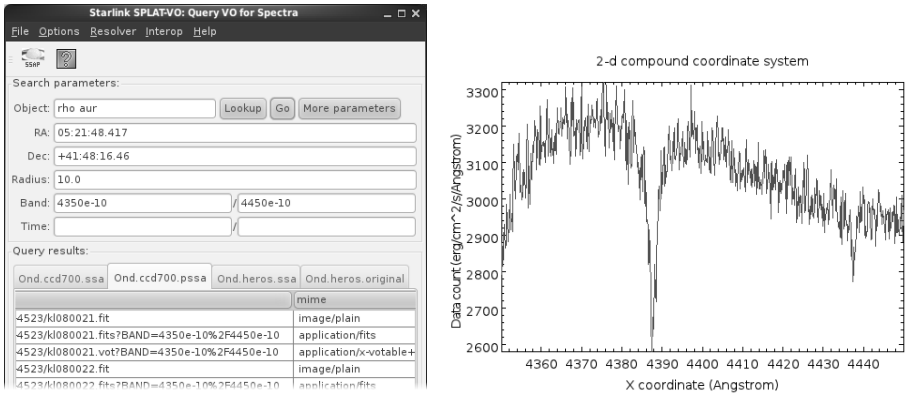
level of flux calibration for acceptable data. Possible values are “absolute”, “relative”, “normalized”, and “any” (the default). If “relative” is specified, spectra which have an absolute flux calibration will be found as well. “Normalized” refers to spectra which have been normalized by dividing by a reference spectrum (including continuum normalization).”(2) As you can see the extension does not do what is specified as normalization. It only shifts values near to y=1 to be comparable with other spectra. Robust fitting is complicated mathematical operation which requires external application.

Postprocessing is easy to notice. Access reference will have arguments like SSA query. URI with all post-processing abilities activated looks like `http://ssaproxy.asu.cas.cz/getproduct/ccd700/data/rrlyr/4895-5147/ng190014.vot?BAND=4900e-10%2F5000e-10&FLUXCALIB=normalized`



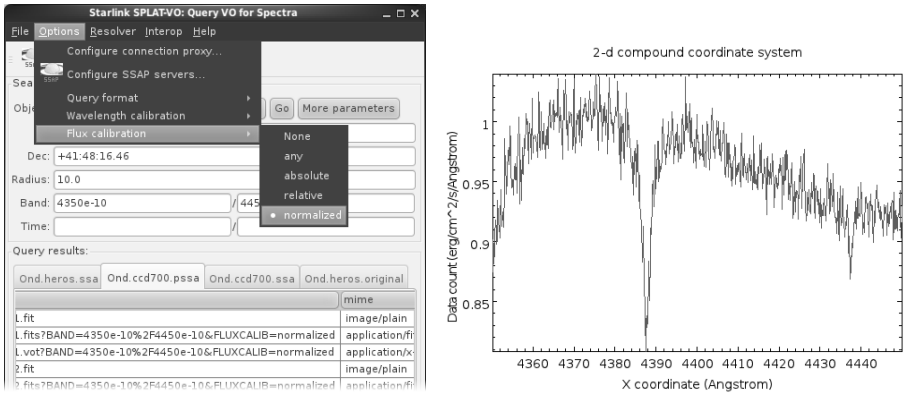
(a) (b)

Figure 3.1: Query and result without post-processing



(a) (b)

Figure 3.2: Query and result with cutout



(a) (b)

Figure 3.3: Query and result with cutout and scale

3.2 Examples

This extension brings few significant improvements into spectral analysis. First it spends less time with downloading spectra. Some spectra has spectral axis longer than 10000 pixels (e.g. the HEROS is about 20000 points and the merged echelle spectra from VLT UVES are about 200000 points). But

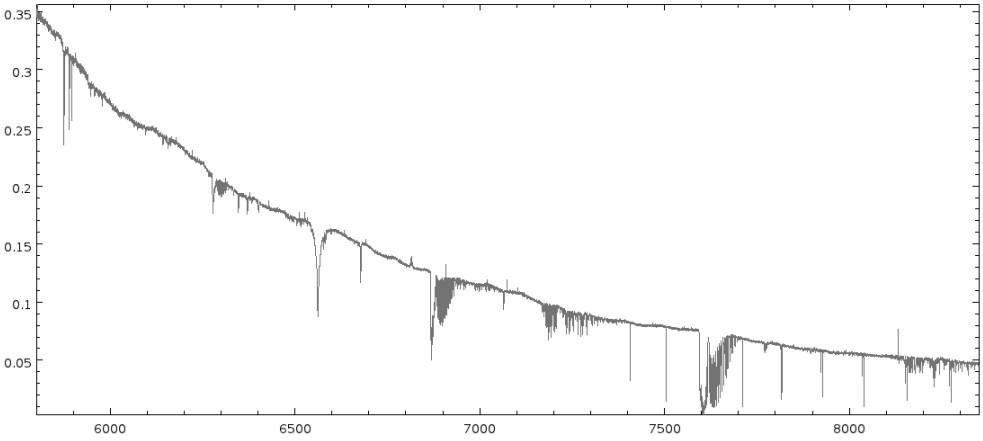


Figure 3.4: Example of long spectra

astronomers mostly needs only small fragment of this data at once (e.g. profile of one spectral line). This can sound insignificantly but astronomers work with many spectra at once. Figure 3.5 shows cutout of only 12 spectra but in real application the spectra are downloaded by tens and thousands. Then the speed-up is really significant.

Scale operation brought better optical comparison of spectra. Sometimes it

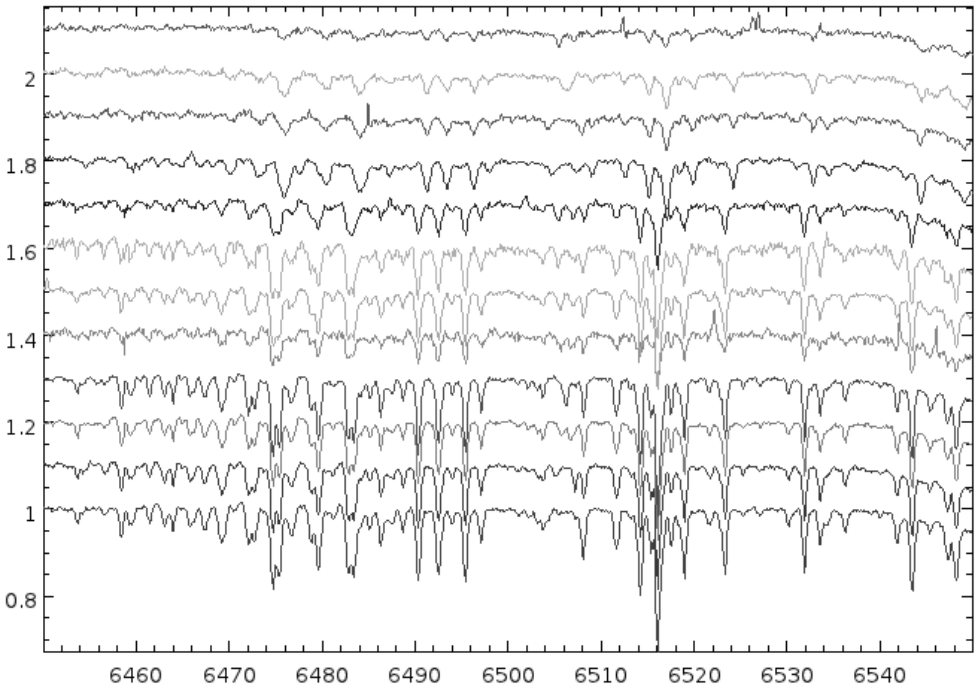
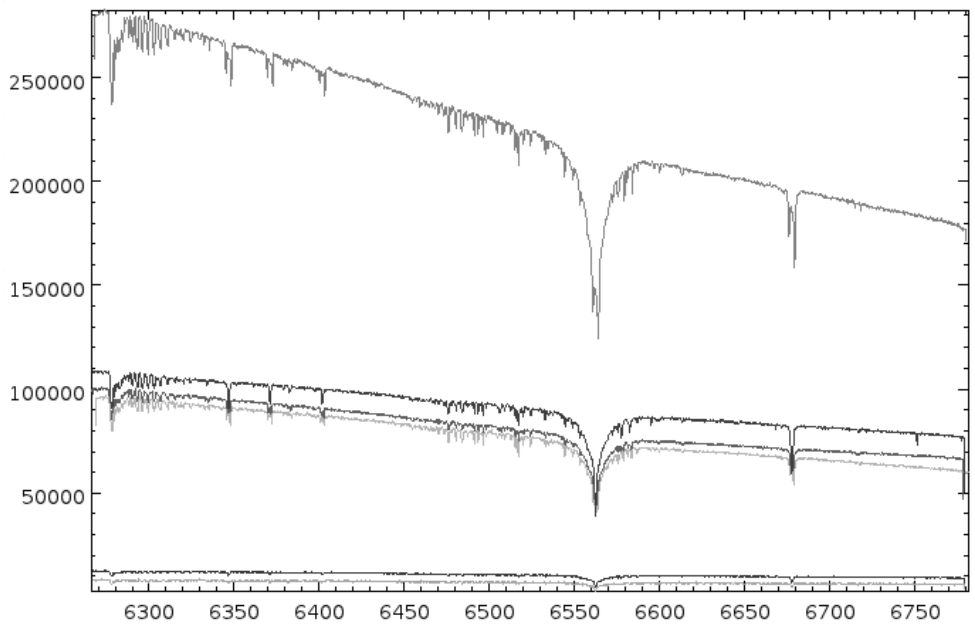
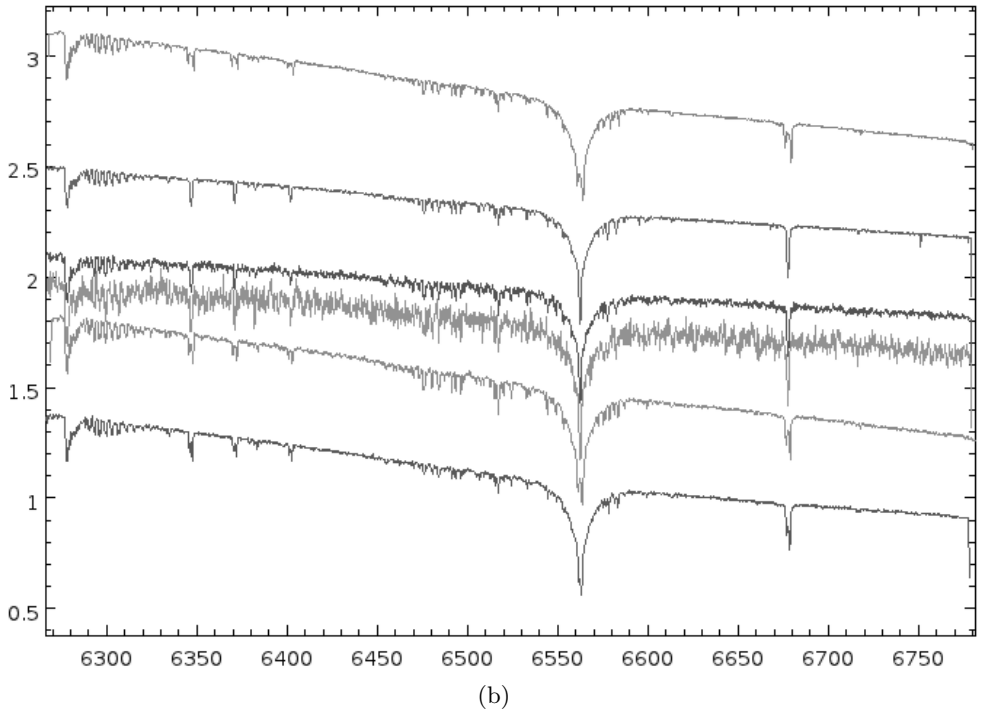


Figure 3.5: Example of cutouted details

is better to look at the spectra than analyse them with mathematical models. Of course that mature tools for spectra analysis provide calibration of flux axis. But astronomer has to set calibration for each spectra separately and it is very time-consuming activity. It is better when it is done automatically.



(a)



(b)

Figure 3.6: Unrectified and scaled spectra

Conclusion

The main goal — adding spectra post-processing capability to SSAP server was achieved. It is done through implementation of SSAP which does not correspond with current SSAP recommendation. But it was unavoidable because there is not (yet) specification for obtaining data. This will come in next release of SSAP with modification called `getData`. Current specification describes only selection queries and it is called `queryData`.

This thesis can be used as a list of available SSAP servers thanks to the feasibility study.

Development of database ingestor was cancelled because of similar product in DaCHS, but may be used as a stand-alone tool for ingesting 1D FITS spectra from other archives in a database - e.g. as a part of tasks connected with usage of other SSA server tookits. DaCHS server met our expectations about ingesting, architecture, protocol extensibility and SSAP 1.1 support. That is why the final decision resulting from feasibility study was to use and modify this server toolkit.

Beside this work some proposals for modification were sent [12] to IVOA for consideration. These changes are related to `getData` modification to enable postprocessing and extension of `queryData`. Proposed extensions are `WILDTARGET`, `WILDTARGETCASE` and `getTargetNames`. All requested changes of SSAP were built into DaCHS.

Thanks to Mr. Demleitner's goodwill DaCHS obtained mechanism for embargoing private spectra. Mechanism will be built in SPLAT-VO client soon. This feature will help to spread Virtual Observatory tools to people who disagreed with full publication of data.

Ondřejov's archives HEROS and CCD700 were moved to new infrastructure. HEROS data were cleaned and updated for important metadata. These archives have been used already as reference service for presentations and propagation of Virtual Observatory.

In future this work is to be merged with `getData` extension. This temporary modification of SSAP shows possibilities and desired features.

[12] `http://docs.g-vo.org/ssaevolution.html`

Bibliography

(1) Demleitner, M.: *GAVO DaCHS documentation*. Available at WWW: <http://docs.g-vo.org/DaCHS/>

(2) Doug Tody, e. a.: Simple Spectral Access Protocol Version 1.1 [online]. 2012. Available at WWW: <http://www.ivoa.net/Documents/SSA>

(3) Francois Ochsenbein, R. W. e. a.: VOTable Format Definition Version 1.2[online]. 2009. Available at WWW: <http://www.ivoa.net/Documents/VOTable/>

(4) IVOA: About IVOA. Available at WWW: <http://www.ivoa.net/pub/info/>

(5) Peter Quinn, A. L.; Hanisch, B.: The Management, Storage and Utilization of Astronomical Data in the 21st Century. 2004. Available at WWW: <http://www.ivoa.net/pub/info/OECD-QLH-Final.pdf>

(6) Škoda, P.: Baltic Astronomy. In *OPTICAL SPECTROSCOPY WITH THE TECHNOLOGY OF VIRTUAL OBSERVATORY*, 2011, p. 19, arXiv:1112.2779v1.

Appendix A

Acronyms

CSV Comma-Separated Values

DAL Data Access Layer

DEC Declination

FITS Flexible Image Transport System

IVOA International Virtual Observatory Alliance

VO Virtual Observatory

SSAP Simple Spectral Access Protocol

UCD Unified Content Descriptors

XML Extensible markup language

RD Resource Descriptor

RA Right ascension

SI International System of Units

accref Access reference

Appendix B

Contents of enclosed CD

- `readme.txt` ..... brief description of content
- `src/`
  - `impl/` ..... sources of implementation
    - `fitsql/` ..... unfinished database ingestor
    - `dachs/` . extended astronomical server with post-processing abilities
  - `thesis/` ..... sources of this document in LaTeX
- `res/`
  - `readme.txt` ..... deploy guide for DaCHS
  - `ccd700/` ..... spectral archive with Resource Descriptor
  - `gavo.rc` ..... GAVO DaCHS configuration file
- `text/` ..... text of the thesis
  - `BP_Peterka_Tomas_2012.pdf` ..... thesis in PDF format