Astro-WISE interfaces

Belikov, Andrey N.; Vriend, Willem-Jan; Sikkema, Gert

Published in:
Experimental Astronomy

DOI:
10.1007/s10686-012-9287-0

Exp Astron (2013) 35:301–318
DOI 10.1007/s10686-012-9287-0

ORIGINAL ARTICLE

Astro-WISE interfaces

Scientific information system brought to the user

Andrey N. Belikov · Willem-Jan Vriend · Gert Sikkema

Received: 28 July 2011 / Accepted: 17 January 2012 / Published online: 10 February 2012

Abstract From a simple text interface to a graphical user interfaces—Astro-WISE provides the user with a wide range of possibilities to interact with the information system according to the user's tasks and use cases. We describe a general approach to the interfacing of a scientific information system. We use this approach to create a number of services, which allows the user to browse the data stored in the system, to process the data and to exchange the newly created images and catalogs with the users within the system and wider astronomical community. Reusability of interfaces and services is another important feature of our approach. It reduces the time and resources spent to interface other information systems created from Astro-WISE.

Keywords Information system · Interfaces · Web services

1 Introduction

Any information system is as good as the ability of the user to exploit all features of this system, the same is true for a scientific information system. To build an extended, stable and easy-to-use system of interfaces to Astro-WISE and information systems derived from Astro-WISE we had to come a long way

A. N. Belikov · W.-J. Vriend · G. Sikkema (✉)
Kapteyn Astronomical Institute, University of Groningen, Landleven 12,
9747AB, Groningen, The Netherlands
e-mail: G.Sikkema@astro.rug.nl

A. N. Belikov
e-mail: belikov@astro.rug.nl

W.-J. Vriend
e-mail: vriend@astro.rug.nl

of analyzing users requests, trying to satisfy them and collecting feedback and proposals for improvement.

The types of interfaces from the point of view of the design of interfaces can vary from the simplest text interface to a dynamic user interface, and the appropriate choice for the type of interface is an important task for the developer. The idea of an integrated approach to interfacing a specific information system is discussed for quite a long period (see, for example, [1]). The approach which we selected for developing interfaces is closest to a model-based approach in interfacing [2].

In this paper we will describe not just interfaces to Astro-WISE but services which implement these interfaces with their non-trivial functionality. As a result, we will write more about services explaining which particular interfaces these services provide.

2 Interfaces and services: general overview

Astro-WISE as an information system has three main layers: metadata database, data storage grid and data processing grid. Figure 1 gives an overview of how the Astro-WISE services are mapped on the three main layers. For each layer we created a basic API. Such an interface exists as a collection of methods for each class in the object-oriented data model of Astro-WISE, for example, *store()* and *retrieve()* methods for the data items. But to use these

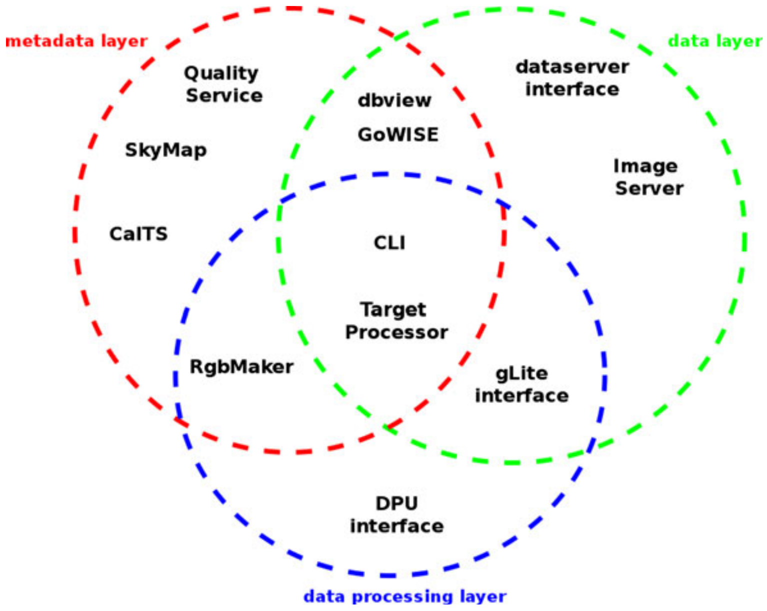


Fig. 1 The classification of Astro-WISE services by their access to layers of the system

methods directly the user has to learn the full data model of Astro-WISE, which includes hundred of classes with complicated relations between them. Even in the case of scientific information systems users are not ready for such a sacrifice.

The solution is to study the main requests and operations which the users will demand from the system and to combine them in a number of user-friendly services. Before analyzing requests we have to categorize users:

- **database administrator**—a user who has the highest privileges in the database;
- **project manager**—a user who validates the data and calibrates an instrument;
- **developer**—a user who can change the data model and pipelines;
- **researcher**—a most common type of users who must be allowed to browse the data and execute predefined pipelines and programs;
- **external users**—a group of users who have no account in the Astro-WISE system, but who are allowed to browse data published by Astro-WISE users.

All these groups of users will have a number of requests to the information in Astro-WISE and interact differently with the layers of the system. For example, external users should be allowed to perform data mining on metadata and data layers according to the access policy, which will prevent them from browsing private or unpublished data. They will also have no access to the data processing layer.

The way to deliver the services to the users was selected to be the same for almost all services—a HTTP(S) based web server written in Python and using a modular approach. The only Interface not using a HTTP server is the Command Line Interface. To access the metadata, two methods can be used: directly access the metadata, stored in the database, by SQL queries or use the corresponding Python classes, which will generate the SQL. In both cases it is verified that the user has the privileges to access the data.

In fact, all these services are a realization of an extended user interaction cycle [3, 4]. In the case of some services, this is a simplified cycle which involves database browsing only, but in most cases the service actually changes the state of the system.

Based on the user classification described above and typical requests to the system of these users, a distinction is made in the following functionalities:

* Html Interface—html interface for interaction;
* Client Interface—Python client interface for interaction;
* Query or Search—simple data mining abilities;
* Explore—the ability to provide more complicated data mining with subselection and modifying requests;
* Make—an ability to create new objects launching data processing;

Table 1 The different Astro-WISE services and their properties

Service	Html UI	Client UI	Query or search	Make	Qualify or update	Visualize	Explore
CalTS	X	–	X	(3)	X	–	–
Cutout	X	X	–	(4)	X	–	(2)
DbView	X	–	X	–	–	X	–
GoWISE	X	–	X	–	–	–	X
Image server	–	X	–	(4)	–	X	–
Quality	X	–	–	(3)	X	X	–
RGB	X	X	–	(4)	–	X	(2)
Skymap	X	–	–	–	–	X	X
Target processor	X	–	X	X	(1)	–	–
Virtual observatory	X	–	X	–	–	–	–

(1) New database objects are automatically qualified
(2) Has (primitive) interface showing data items created by other users
(3) Creation of comments
(4) Creation of non-database items (images, cutouts)

* Qualify or Update—an ability to verify objects and setting or changing the qualification parameters;
* Visualize—an ability to inspect the image visually.

Table 1 lists the functionalities per web service. Each web service realizes a group of use cases which users (actors) demand from the system. For example, the user (actor) researcher has to browse the metadata database, to select a data entity, according to specified criteria, and to inspect the image visually. All these can be done with the DbView Service. The use cases for Astro-WISE and, in turn, for Astro-WISE webservices are described in Astro-WISE architectural design,[1] these use cases are the basis for functionalities listed in Table 1. To implement functionalities of the services we used a combination of object-oriented and modular approach in programming (see Section 3), which is a specific feature of Astro-WISE. Each service is supplied with extended online help guiding the user through the service.

As we can see, there is a number of web services provided for Astro-WISE users. To make this paper and the introduction to Astro-WISE interfaces more user-friendly we divide all services on the groups according to their prior functionality, i.e. selecting a group of interfaces which provide data mining functionality, data processing, monitoring and control and, finally, visualization. Of course, this division is artificial in the sense that many of the services provide more than just one functionality. In following sections we will describe the result of interfacing in more details. Each service will be described according to the Abowd and Beale framework of interaction [4], i.e., describing tasks, input and outputs.

[1] http://www.astro-wise.org/Public/cdr.pdf

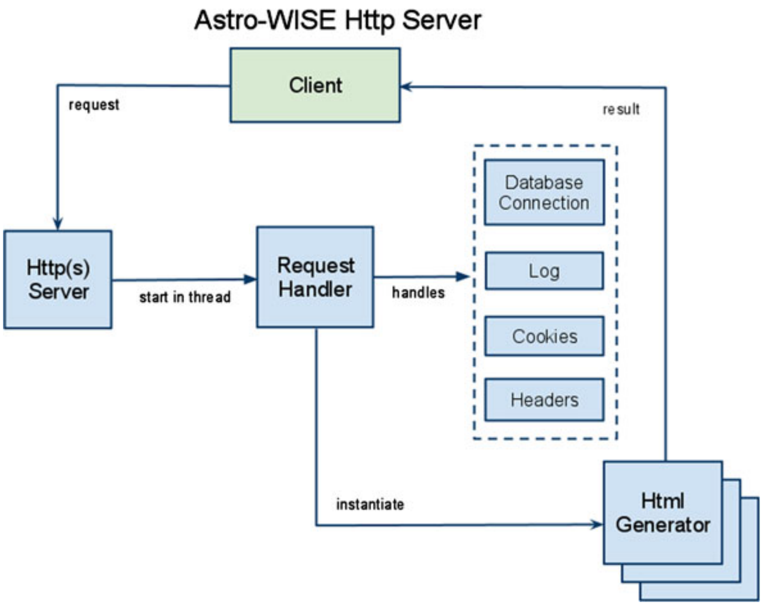


Fig. 2 The class diagram of a typical Astro-WISE interface service

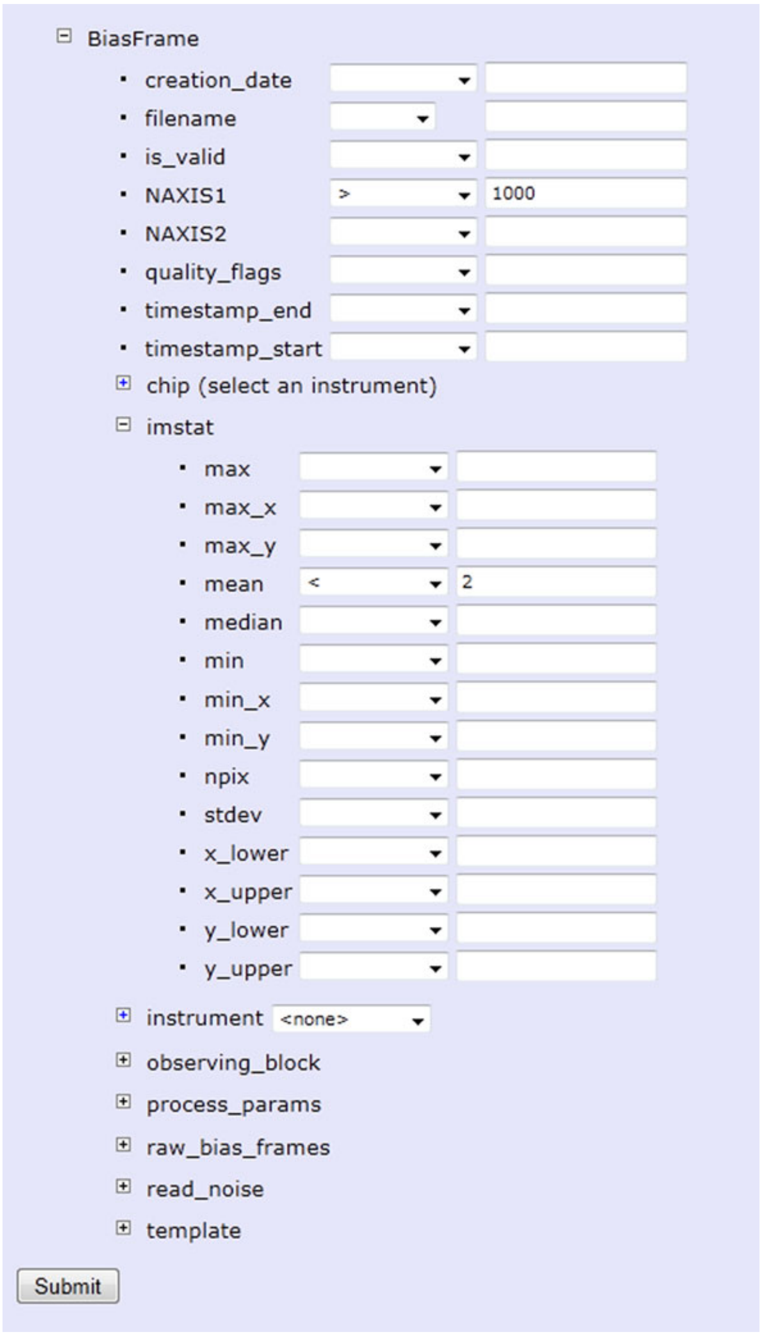
3 Modular approach to interface services

The interesting feature of Astro-WISE interfaces is their modularity, i.e., all interfaces are built from the standards blocks written in Python. In the core of each service (except CLI) is a web server written in Python and invoking a number of services as Python modules, see Fig. 2. Each interface or functionality is realized in a separate Python module, which can be reused in a number of services. Modules are glued together by the parent Python class and a set of configuration parameters, which are provided separately as a detached file during the initialization of the service. Each service has an assigned TCP/IP port and a fully qualified domain name.

Reusability of modules allows easily to switch functionality implemented in one interface to another one, and create a new web service which implements a different set of tasks. This feature is useful not only for Astro-WISE but also for developing of new information systems derived from Astro-WISE, for example, for LOFAR Long-Term Archive data mining service.

4 Data mining services

Data mining functionality is implemented in a number of Astro-WISE services, three of these are dedicated completely to data mining: DbView, the main Astro-WISE service to browse the metadata database; GoWise, the service for the quick search in the metadata database and the Virtual Observatory service.



⊟ BiasFrame

creation_date		
filename		
is_valid		
NAXIS1	>	1000
NAXIS2		
quality_flags		
timestamp_end		
timestamp_start		

⊞ chip (select an instrument)

⊟ imstat

max		
max_x		
max_y		
mean	<	2
median		
min		
min_x		
min_y		
npix		
stdev		
x_lower		
x_upper		
y_lower		
y_upper		

⊞ instrument <none>

⊞ observing_block

⊞ process_params

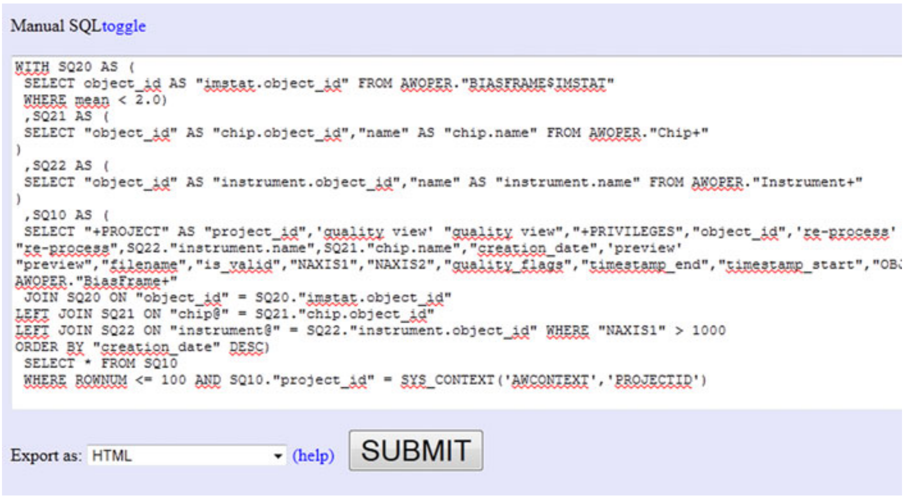
⊞ raw_bias_frames

⊞ read_noise

⊞ template

Submit

Fig. 3 The set of attributes for a particular class


Manual SQLtoggle

```
WITH SQ20 AS (
 SELECT object_id AS "imstat.object_id" FROM AWOPER."BIASFRAME$IMSTAT"
 WHERE mean < 2.0)
 ,SQ21 AS (
 SELECT "object_id" AS "chip.object_id","name" AS "chip.name" FROM AWOPER."Chip+"
)
 ,SQ22 AS (
 SELECT "object_id" AS "instrument.object_id","name" AS "instrument.name" FROM AWOPER."Instrument+"
)
 ,SQ10 AS (
 SELECT "+PROJECT" AS "project_id",'quality view' "quality view","+PRIVILEGES","object_id",'re-process'
"re-process",SQ22."instrument.name",SQ21."chip.name","creation_date",'preview'
"preview","filename","is_valid","NAXIS1","NAXIS2","quality_flags","timestamp_end","timestamp_start","OB
AWOPER."BiasFrame+"
 JOIN SQ20 ON "object_id" = SQ20."imstat.object_id"
LEFT JOIN SQ21 ON "chip@" = SQ21."chip.object_id"
LEFT JOIN SQ22 ON "instrument@" = SQ22."instrument.object_id" WHERE "NAXIS1" > 1000
ORDER BY "creation_date" DESC)
 SELECT * FROM SQ10
 WHERE ROWNUM <= 100 AND SQ10."project_id" = SYS_CONTEXT('AWCONTEXT','PROJECTID')
```

Export as: HTML (help)

Fig. 4 A SQL window which represents a user selection

4.1 DbView

The core of the data mining services is the DbView service.[2] It provides access to the metadata database and allows the user to browse according to the user's privileges. DbView provides an html interface for SQL queries, this is an html-based interface which allows to retrieve not only the data item itself but build a tree of dependencies for this item.

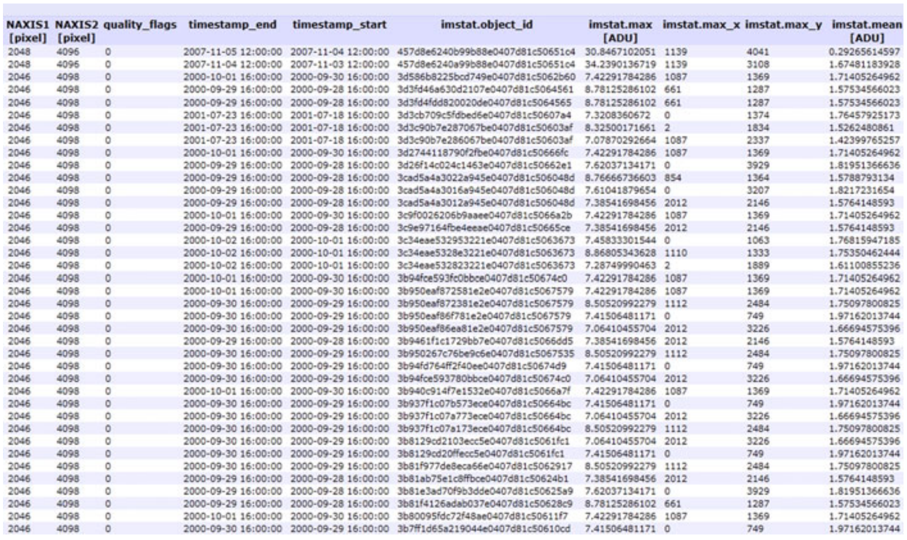
DbView presents the Astro-WISE data model to the user. Practically all classes of Astro-WISE can be browsed in DbView (for the full list see dbview Tables page[3]). The user can browse the Astro-WISE classes, make a request by providing a range of values for each attribute of the class (Fig. 3) or by a direct SQL statement (the user can modify it, see Fig. 4), which will result in a list of data items (Fig. 5).

The Astro-WISE data model has following types of data:

* `desc`; atomic types as integer, string, float, datetime
* `link`; pointer to objects
* `link self`; pointer to objects of the same type
* `link inline`; same as link, but the linked object is not stored as a separate object.

Types described above can occur as single instances and lists. A persistent class is an aggregation of these types. The data model is defined and implemented in the programming language Python. Database administrators can

[2]http://dbview.astro-wise.org
[3]http://dbview.astro-wise.org/DbView?mode=table_sel&table_sel=all



NAXIS1 [pixel]	NAXIS2 [pixel]	quality_flags	timestamp_end	timestamp_start	imstat.object_id	imstat.max [ADU]	imstat.max_x	imstat.max_y	imstat.mean [ADU]
2048	4096	0	2007-11-05 12:00:00	2007-11-04 12:00:00	457d8e6240b99b88e0407d81c50651c4	30.8467102051	1139	4041	0.29265614597
2048	4096	0	2007-11-04 12:00:00	2007-11-03 12:00:00	457d8e6240a99b88e0407d81c50651c4	34.2390136719	1139	3108	1.67481183928
2046	4098	0	2000-10-01 16:00:00	2000-09-30 16:00:00	3d586b8225bcd749e0407d81c5062b60	7.42291784286	1087	1369	1.71405264962
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3d3fd46a630d2107e0407d81c5064561	8.78125286102	661	1287	1.57534566023
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3d3fd4fdd820020de0407d81c5064565	8.78125286102	661	1287	1.57534566023
2046	4098	0	2001-07-23 16:00:00	2001-07-18 16:00:00	3d3cb709c5fdbed6e0407d81c50607a4	7.3208360672	0	1374	1.76457925173
2046	4098	0	2001-07-23 16:00:00	2001-07-18 16:00:00	3d3c90b7e287067be0407d81c50603af	8.32500171661	2	1834	1.5262480861
2046	4098	0	2001-07-23 16:00:00	2001-07-18 16:00:00	3d3c90b7e286067be0407d81c50603af	7.07870292664	1087	2337	1.42399765257
2046	4098	0	2000-10-01 16:00:00	2000-09-30 16:00:00	3d2744118790f2fbe0407d81c50666fc	7.42291784286	1087	1369	1.71405264962
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3d26f14c024c1463e0407d81c50662e1	7.62037134171	0	3929	1.81951366636
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3cad5a4a3022a945e0407d81c506048d	8.76666736603	854	1364	1.5788793134
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3cad5a4a3016a945e0407d81c506048d	7.61041879654	0	3207	1.8217231654
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3cad5a4a3012a945e0407d81c506048d	7.38541698456	2012	2146	1.5764148593
2046	4098	0	2000-10-01 16:00:00	2000-09-30 16:00:00	3c9f0026206b9aaee0407d81c5066a2b	7.42291784286	1087	1369	1.71405264962
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3c9e97164fbe4eeae0407d81c50665ce	7.38541698456	2012	2146	1.5764148593
2046	4098	0	2000-10-02 16:00:00	2000-10-01 16:00:00	3c34eae532953221e0407d81c5063673	7.45833301544	0	1063	1.76815947185
2046	4098	0	2000-10-02 16:00:00	2000-10-01 16:00:00	3c34eae5328e3221e0407d81c5063673	8.86805343628	1110	1333	1.75350462444
2046	4098	0	2000-10-02 16:00:00	2000-10-01 16:00:00	3c34eae532823221e0407d81c5063673	7.28749990463	2	1889	1.61100855236
2046	4098	0	2000-10-01 16:00:00	2000-09-30 16:00:00	3b94fce593fc0bbce0407d81c50674c0	7.42291784286	1087	1369	1.71405264962
2046	4098	0	2000-10-01 16:00:00	2000-09-30 16:00:00	3b950eaf872581e2e0407d81c5067579	7.42291784286	1087	1369	1.71405264962
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b950eaf872381e2e0407d81c5067579	8.50520992279	1112	2484	1.75097800825
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b950eaf86f781e2e0407d81c5067579	7.41506481171	0	749	1.97162013744
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b950eaf86ea81e2e0407d81c5067579	7.06410455704	2012	3226	1.66694575396
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3b9461f1c1729bb7e0407d81c5066dd5	7.38541698456	2012	2146	1.5764148593
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b950267c76be9c6e0407d81c5067535	8.50520992279	1112	2484	1.75097800825
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b94fd764ff2f40ee0407d81c50674d9	7.41506481171	0	749	1.97162013744
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b94fce593780bbce0407d81c50674c0	7.06410455704	2012	3226	1.66694575396
2046	4098	0	2000-10-01 16:00:00	2000-09-30 16:00:00	3b940c914f7e1532e0407d81c5066a7f	7.42291784286	1087	1369	1.71405264962
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b937f1c07b573ece0407d81c50664bc	7.41506481171	0	749	1.97162013744
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b937f1c07a773ece0407d81c50664bc	7.06410455704	2012	3226	1.66694575396
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b937f1c07a173ece0407d81c50664bc	8.50520992279	1112	2484	1.75097800825
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b8129cd2103ecc5e0407d81c5061fc1	7.06410455704	2012	3226	1.66694575396
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b8129cd20ffecc5e0407d81c5061fc1	7.41506481171	0	749	1.97162013744
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b81f977de8eca66e0407d81c5062917	8.50520992279	1112	2484	1.75097800825
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3b81ab75e1c8ffbce0407d81c50624b1	7.38541698456	2012	2146	1.5764148593
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3b81e3ad70f9b3dde0407d81c50625a9	7.62037134171	0	3929	1.81951366636
2046	4098	0	2000-09-29 16:00:00	2000-09-28 16:00:00	3b81f4126adab037e0407d81c50628c9	8.78125286102	661	1287	1.57534566023
2046	4098	0	2000-10-01 16:00:00	2000-09-30 16:00:00	3b80095fdc72f48ae0407d81c50611f7	7.42291784286	1087	1369	1.71405264962
2046	4098	0	2000-09-30 16:00:00	2000-09-29 16:00:00	3b7ff1d65a219044e0407d81c50610cd	7.41506481171	0	749	1.97162013744

Fig. 5 Figure showing the result of a query

invoke a Python script which generates on the fly data schemes for the Oracle database from the persistent Python class definitions. It is also possible to generate the Python code from, for example, xml. This approach is used by the Multi Unit Spectroscopic Explorer (MUSE[4]) data processing system, which will be based on Astro-WISE.

The Python data model building blocks are :

* `DBObject`; base class for persistent classes
* `DataObject`; persistent class with an attached file
* `persistent`; method for making class attributes persistent.

The persistent Python classes are stored in the database. The relation between Oracle tables and Python classes are stored in Oracle Comments. Oracle Comments are a specific feature of Oracle which allows to store additional information on the table or column in the database. The information is a free-format text which can be retrieved to understand what kind of information is stored in the table or column. These Comments provide a way of storing metadata about Oracle objects, generally we use them to couple Python classes and attributes to Oracle tables and columns. DbView web service uses these Comments to generate the html query forms. The query form lets the user query on the desc attributes of the class, and the desc attributes of child classes. After submitting the form the DbView web service will generate SQL statement, which is used to query the database. The result can span multiple classes and is rendered as an html table.

[4]http://www.eso.org/sci/facilities/develop/instruments/muse/

DbView is a core service for the data mining with Astro-WISE, as it combines links to many other services, including Quality Service, Cut-out Service, Image Server and Target Processor. The resulting tables are enriched with links to: files, cutouts, object views and other web services. The DbView web services is not bound to a specific data model implementation. It uses the meta-data of the data model, in the Oracle Comments, to generate the html forms. Specific implementations can extend the functionality of the generic DbView web service.

4.2 GoWise

The GoWise web service[5] is a simplified DbView version. It only allows the user to give an object as input. The object can be specified by name or coordinates. In case of an object name the service will resolve the name to coordinates, using Simbad web service.[6] Given the coordinates the service queries a set of tables (data item classes) in the database, for one or all projects. The result of these queries is rendered in html tables. Only a subset of the attributes of the resulting data item classes is shown. In case there are more results then a defined maximum not all results will be shown but the service gives the option to zoom in on a specific project or data item class.

This service is aimed at the novice user. It gives a rough overview of the data of an object or coordinate range in the system, and provides links to additional services for the discovered data items.

4.3 Virtual observatory interfaces

Virtual Observatory (VO) interfaces are realized as a separate VO service,[7] which enables browsing the metadata database and retrieving the data from dataservers. Astro-WISE provides two VO interfaces: Simple Image Access Protocol for images and ConeSearch for sources. Both interfaces and registry are built upon standard Python classes for web services described earlier. Each data entity in Astro-WISE has a persistent attribute which shows the scope of visibility of this entity. For example, if for some image the attribute `privileges=1`, this mean that the image can be accessed by the owner of the image only. Rising this attribute to 5 will allow to VO user to browse the metadata of this image and to retrieve the image itself.

All data items in Astro-WISE are assigned to some *project* which is a group of users who share data and can work on this data collection together. Usually a project is connected to a specific scientific use-case. Most of the data items (except catalogs merged from other catalogs and catalogs external to Astro-WISE like 2MASS PSC, for example) are associated with some instrument. A

[5]http://gowise.astro-wise.org
[6]http://simbad.u-strasbg.fr/simbad/
[7]http://www.astro-wise.org/portal/aw_vo.shtml

VO user has two options: select data items by a project or by an instrument. In the first case the user will retrieve data related to some use-case, in the second case the user browses an archive of the instrument (for example, WFI@2.2m). If the user wants to retrieve the data for a specific object or range of coordinates through a set of projects he has to issue multiple requests, one request per project/instrument, and combine the data in the external application. This task can be easily peformed with the use of the DbView service which can return the data in a VOTable format and has a SAMP implementation.

5 Data interfaces

The DataServers form the backbone of the data file distribution of the Astro-WISE system. These services use the HTTP protocol to serve files. The files are linked in the database by filename, as defined by the `DataObject` class. The interface is straightforward, a user requests a file by filename, the dataserver returns the file content. Multiple data servers can form a logical unit.

6 Command line interface

Command Line Interface (CLI, or Astro-WISE Environment—AWE) is the simplest and most powerful way the user can interact with the Astro-WISE system. CLI is developed on the base of Python and in fact it is an environment put on the standard Python CLI. Astro-WISE CLI gives access to all Astro-WISE classes and libraries and allows to build a Python program.

CLI is set by a configuration file, which specifies the user's login, the initial project, dataserver, Distributed Processing Unit (DPU) and numerous other configuration options. CLI is also used as a programming environment which allows submitting Python programs to Astro-WISE, the user can write his own program involving Astro-WISE classes and libraries and execute it through CLI.

There is a web-based version of the CLI,[8] although not given the full performance and functionality of the local CLI, it can be used when the user has no local CLI installed.

7 Data processing interfaces

Data processing interfaces provide to the user an access to the data processing layer of Astro-WISE. The task of data processing interfaces is to implement an uniform way for the user to access different data processing facilities

[8]http://awe.astro-wise.org

employed by Astro-WISE, including “native” Astro-WISE processing elements (High Performance Computing cluster in Groningen, for example) and “external” processing facilities (for example, BiGGrid processing element in Amsterdam).

7.1 Target processor

The Target Processor web service[9] gives the user an html interface to the Target Processing Interface. Target processing is a certain way of processing. A user requests a certain object and the system will then determine what has to be made and how [5]. The Target Processor uses the data model to determine what has to be made to fulfill the request of the user. In the most extreme case only the raw objects exist and everything above (or below) is processed by the Target Processor, until the user requested object is made. The Target Processor traverses the data model, using the persistent dependencies of the classes. For each dependency the Target Processor checks if an instance of the dependency already exists in the database, and if so, whether the dependency is uptodate. The decision model looks like:

```
object exist ?
    no -> make object
    yes -> object uptodate ?
        no -> remake object
        yes -> use object
```

The *exist* method queries the database, using a set of parameters. The set of parameters depends on the class. If the query results in multiple objects the most recent is returned. The *uptodate* check is more complicated. This first checks if the object is flagged, if so the object is out-of-date. Then for all the dependencies of the object it is checked if they are the newest version. Essentially calling the *exist* method on every dependency checks if this returns the current dependency. If not all dependencies represent the newest version, the object is out-of-date. The last check is to call the *uptodate* method of all the dependencies. If not all dependencies are up-to-date, the object is out-of-date. This recursive calling of the *uptodate* method can be done to a certain depth, or until a raw class is hit. Raw objects can be flagged, but have no dependencies to check.

7.2 DPU

The DPU provides a HTTP interface for submitting jobs on compute clusters. The jobs are Python pickles and contain the parameters to run each job. The user instantiates the jobs, defines the hierarchical structure of the jobs and which compute cluster to use. The DPU then takes care of submitting the jobs

[9]http://process.astro-wise.org

to the specified compute cluster, monitors the progress of the jobs and returns the logs. The jobs are responsible for storing the resulting files on a data server and populate the database by committing an instance of the datamodel.

7.3 gLite interface

Making use of Grid storage and processing facilities requires confirming to standards. We use the following packages to implement these standards:

* jLite;[10] proxy certificate handling
* dcache-srmclient;[11] srm interaction (ls, cp, rm, mkdir)
* globus;[12] myproxy handling.

These packages are Java only, making the installation of them easy and architecture independent. We made Python wrappers for these packages, so they can be easily used throughout the system.

7.4 RGB maker

The RGB Maker web service[13] generates a RGB image from 3 input frames. The service aids the user in selecting the possible input frames, presenting an overview of all frames with the same pixel size and target area on the sky. The three input frames are cut (using the Image Server) to the same dimensions and the utility stiff[14] is used to combine the three frames to a RGB image.

The RGB images are stored on the web server and are visible to other users. The metadata of the RGB images is not stored in the database.

8 Monitoring and control services

One of the core task in the producing a survey is to verify it's quality. There are a number of services in Astro-WISE which allow doing this—CalTS is tracing all calibrations used for data processing, Quality service allow to visualize information about the quality of a particular image, it's astrometric and photometric solution. Together these services allow to trace the quality of all data processing steps.

[10]http://code.google.com/p/jlite/
[11]http://www.dcache.org/downloads/1.9/
[12]http://www.globus.org/toolkit/
[13]http://rgb.astro-wise.org
[14]http://www.astromatic.net/software/stiff

8.1 CalTS

The CalTS web service[15] provides an html interface for updating or qualifying the calibration objects in the database. Calibration objects in the "astro" implementation have five attributes which determine whether and when the object will be used for data processing in the Astro-WISE system [6]:

* `timestamp_start` (date)
* `timestamp_end` (date)
* `creation_date` (date)
* `quality_flags` (int)
* `is_valid` (int)

The timestamp attributes define the date range for which this calibration object should be used. The date range can be in the order of a day to years. In case there is overlap in timestamp range between multiple calibration objects, the object with the newest (latest) `creation_date` will be used. There are two flags indicating the quality of the object; the `quality_flags` will be set by the system automatically, the `is_valid` flag is (un)set by an authorized user. When the `quality_flags` is set (i.e., non-zero) or the `is_valid` is unset these attributes indicate that the object should not be used. Using the CalTS web service a user can specify a date range and a calibration object type. The CalTS web service will then render a graphical overview of when the calibration objects of this type are relevant. The user can then adjust the timestamp ranges of the objects and (un)set the `is_valid` flag.

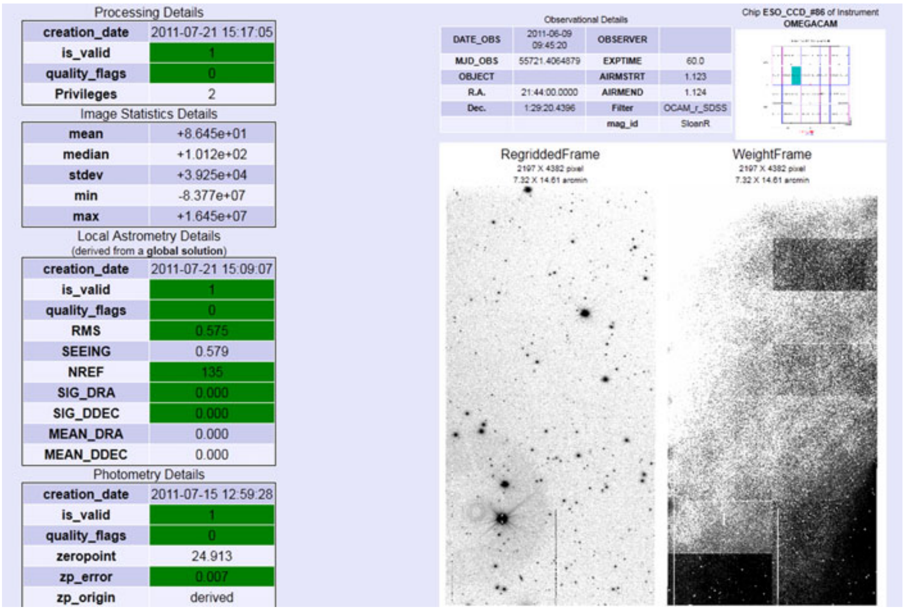
8.2 Quality service

The Quality Service gives an overview of the quality of a data item. An html page (Fig. 6) is rendered with tabular and graphical data. This data is extracted and calculated from the metadata in the database and file(s) on the dataserver. The user can inspect the various quality plots and decide to invalidate the data item. The quality of the Astro-WISE system and the quality service is fully covered in [7].

9 Visualization services

Astro-WISE develops visualization of the data stored in the system by creating "native" Astro-WISE services (Image Server, visualization in a number of other services like Quality Server and DbView) and exchanging the data with the external visualising applications which is in most cases implemented by using Simple Application Messaging protocol (see Section 9.3).

[15]http://calts.astro-wise.org



Processing Details

creation_date	2011-07-21 15:17:05
is_valid	1
quality_flags	0
Privileges	2

Image Statistics Details

mean	+8.645e+01
median	+1.012e+02
stdev	+3.925e+04
min	-8.377e+07
max	+1.645e+07

Local Astrometry Details
(derived from a global solution)

creation_date	2011-07-21 15:09:07
is_valid	1
quality_flags	0
RMS	0.575
SEEING	0.579
NREF	135
SIG_DRA	0.000
SIG_DDEC	0.000
MEAN_DRA	0.000
MEAN_DDEC	0.000

Photometry Details

creation_date	2011-07-15 12:59:28
is_valid	1
quality_flags	0
zeropoint	24.913
zp_error	0.007
zp_origin	derived

Observational Details

DATE_OBS	2011-06-09 09:45:20	OBSERVER	
MJD_OBS	55721.4064879	EXPTIME	60.0
OBJECT		AIRMSTRT	1.123
R.A.	21:44:00.0000	AIRMEND	1.124
Dec.	1:29:20.4396	Filter	OCAM_r_SDSS
		mag_id	SloanR

Fig. 6 Quality information of a reduced science frame. On the *left* are numerical quality indicators, on the *right* is the image and the weight frame. Not all information provided by the service is shown in this plot

9.1 SkyMap

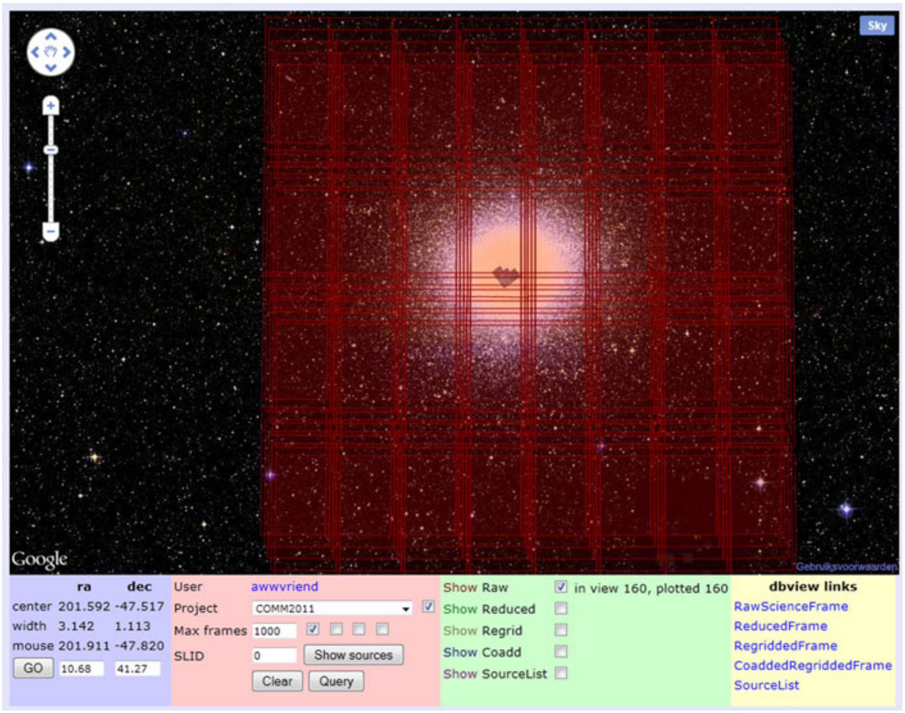
SkyMap is a web service[16] that is build on top of Google Sky.[17] Google Sky interfaces the visible sky with the same interface as Google Maps. The user can pan and zoom to explore the sky, from fully zoomed out to almost arc-second.

The SkyMap web service overplots Google Sky with objects in the Astro-WISE database (Fig. 7). These are frames (from raw to reduced), SourceLists and individual sources. In zoomed-out mode only aggregative information per square degree of all objects is shown. When zooming in the actual objects are plotted on the sky. The user can select a filter, observer or object name to refine the shown objects.

9.2 Image server

In the astronomical world pixel data is stored in FITS format, as it is also done in Astro-WISE. The FITS format is not supported by web browsers. To be able to link to FITS files and view them in browsers we made the Image Server. This

[16]http://skymap.astro-wise.org
[17]http://www.google.com/sky/



	ra	dec
center	201.592	-47.517
width	3.142	1.113
mouse	201.911	-47.820
GO	10.68	41.27

User awwvriend
Project COMM2011 ✓
Max frames 1000 ✓ ☐ ☐ ☐
SLID 0 Show sources
Clear Query

Show Raw ✓ in view 160, plotted 160
Show Reduced ☐
Show Regrid ☐
Show Coadd ☐
Show SourceList ☐

dbview links
RawScienceFrame
ReducedFrame
RegriddedFrame
CoaddedRegriddedFrame
SourceList

Fig. 7 SkyMap web service, showing 5 OmegaCam observations in the SDSS z' filter. The dither pattern of the 8 by 4 ccd's is clearly visible, coverage of the ccd's is about 1 by 1 degree

HTTP service converts the requested FITS file to PNG and returns this image to the user. The user does not have to use dedicated viewing software to inspect FITS files.

Next to converting FITS files, the Image Server is also used to make cutouts of FITS files. When only a small part of a large image is needed for inspection or data analysis the Image Server can make a cutout and return the cutout in PNG or FITS format. The header of the FITS file is changed according to the cutout coordinates.

9.3 SAMP

The Simple Application Messaging Protocol (SAMP[18]) is a messaging protocol that enables (astronomy) software tools to interoperate and communicate. We adopted the SAMP protocol in the DbView web service and the Client

[18]http://www.ivoa.net/Documents/SAMP

Interface. Data from DbView of CLI can be send to the analysis software, for example, TopCat[19] using the SAMP protocol.

Communication through SAMP is based on application defined messages. We designed new messages [8] to interact with the client interface. This allows third party software to request both existing and new data and even influence the processing through SAMP. The messages are designed to allow other clients to interact with Astro-WISE on various levels, depending on the knowledge they have of the inner workings of Astro-WISE. This results in a wide range of possible applications that could function as an auxiliary interface to Astro-WISE.

10 Authentication and authorization service—Login server

The Login server is used to hand out user credential proxies. These user credential proxies can be encoded in browser cookies. The users only need to login once, the cookie will be used for multiple services. The proxies can also be forwarded to other services, using SAMP, which then uses the proxy to authenticate the user. Proxies can be limited in time, for example for one month, or for a number of usages.

11 Conclusion and future work

The set of interfaces described above was implemented not only for Astro-WISE but can be inherited by any information system developed on the base of Astro-WISE. LOFAR Long-Term Archive is using a DbView-based interface to access and browse the data. The astronomy specific web services can be used as a basis for other implementations, because the way of modeling the underlying data model will be the same.

The developing of interfaces for any information system created from Astro-WISE follows the same pattern due to the same infrastructure inherited from Astro-WISE, i.e., metadata, data files and data processing layer. Each of these layers have a simple general API which is kept practically untouched, and the specific web services are build based on the tasks performed by users of the newly created system. This approach was already used for developing interfaces for the LOFAR Long-Term Archive[20] and Monk.[21]

The system of interfaces and services of Astro-WISE creates a balance between the high level GUI (SkyMap, Quality Service etc.) and the low-level

[19] http://www.star.bris.ac.uk/~mbt/topcat/
[20] http://lofar.astro-wise.org
[21] http://application22.target.rug.nl:8802

interfaces (CLI). The user can select an appropriate set of services to perform his task or to write his own program from scratch and use CLI to execute it in Astro-WISE.

A number of services are linked, allowing to follow the object selected with one of the services to another service. For example, an image selected with the use of DbView can be followed in the Quality Service by the link from DbView, an image selected with the Target Processor is linked to DbView, Quality Service etc. This is due to the fact that all data entities in Astro-WISE have an unique identifier. To exchange the data between Astro-WISE services and non-Astro-WISE applications (topcat, Aladin) SAMP is used. This makes it possible to return data from the external application to Astro-WISE with the use of SAMP as well, for example, visualizing a catalog from Astro-WISE in Aladin, selecting a subset from the catalog and sending it back from Aladin to Astro-WISE.

The future development is targeted towards further automatization of the process of interfacing of a newly created system for the user. For example, a number of web services (data mining services, first of all, DbView) are quite common and should be realized in any information system for the reason of browsing the metadata. In the future DbView-like service can be created using the basic interface components written in Python and can generate the system-depending part from the data model of a newly created system. Such a services were already implemented for LOFAR Long-term Archive.[22] The simplicity of the realization of data mining services are due to the fact that they deal with the data model only and have the only source of information—the database itself, which stores, in the case of Astro-WISE, all relationships between the data objects.

In the case of data processing services and monitoring and control services the task is more complicated due to the fact that a new pipeline for the data processing should be integrated. In this case an external pipeline can be wrapped into Python modules (if the pipeline obeys to modular principles of programming) and each Python module is treated as an independent recipe which can be interfaced to the web service like Target Processor. This work can be automatized to some limit which is defined by specific user requirements to the services which can not be deduced from the data model or pipeline (for example, outline of monitoring and quality control services).

Acknowledgements Astro-WISE is an on-going project which started from a FP5 RTD program funded by the EC Action "Enhancing Access to Research Infrastructures".

[22]http://lofar.astro-wise.org

References

1. Frank A.U., Mark D.M.: Language Issues for Geographical Information Systems, In: Maguire, D. J., Goodchild, M. F., Rhind, D. W. (eds.) Geographical Information Systems: Principles and Applications, vol. 1, pp. 147 (1991)
2. Trætteberg, H.: Model-based User Interface Design (2002). ISBN 82-471-5459-5
3. Norman, D.A.: The Psychology of Everyday Things. Basic Books, New York (1988)
4. Abowd, G.D., Beale, R.: Users, systems and interfaces: a unifying framework for interaction. In: Proceedings of the HCI'1991 conference on people and computers VI, 73 (1991)
5. McFarland, J., Sikkema, G.: Optical Image Pipeline: Development and Implementation. Experimental Astronomy, Astro-WISE issue (2011)
6. Begeman K., Belikov A.N., Boxhoorn D., Valentijn E.A.: The Astro-WISE datacentric information system. Exp. Astron. (2012, submitted)
7. McFarland, J., Neeser, M., Heraudeau, P.: Quality control for astronomical data. Exp. Astron. (2012, submitted)
8. Buddelmeijer, H., Valentijn, E.A.: Query driven visualization of astronomical catalogs. Exp. Astron. (2012, accepted)