Aided generation of search interfaces to astronomical archives

Zorba, Sonia, Bignamini, Andrea, Cepparo, Francesco, Knapic, Cristina, Molinaro, Marco, et al.

Sonia Zorba, Andrea Bignamini, Francesco Cepparo, Cristina Knapic, Marco Molinaro, Riccardo Smareglia, "Aided generation of search interfaces to astronomical archives," Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV, 991344 (26 July 2016); doi: 10.1117/12.2232594

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2016, Edinburgh, United Kingdom

Aided generation of search interfaces to astronomical archives

Sonia Zorba, Andrea Bignamini, Francesco Cepparo, Cristina Knapic, Marco Molinaro, and Riccardo Smareglia

INAF - Osservatorio Astronomico di Trieste, via G.B. Tiepolo 11, Trieste, Italy

ABSTRACT

Astrophysical data provider organizations that host web based interfaces to provide access to data resources have to cope with possible changes in data management that imply partial rewrites of web applications. To avoid doing this manually it was decided to develop a dynamically configurable Java EE web application that can set itself up reading needed information from configuration files. Specification of what information the astronomical archive database has to expose is managed using the TAP_SCHEMA schema from the IVOA TAP recommendation, that can be edited using a graphical interface. When configuration steps are done the tool will build a war file to allow easy deployment of the application.

Keywords: astronomical archives, web interfaces, Virtual Observatory

1. INTRODUCTION

The Italian astrophysical research infrastructure project IA2[*] hosts web based search interfaces to archives of various telescopes like LBT (Large Binocular Telescope), TNG (Telescopio Nazionale Galileo) and many others.

Changes in the discovery logic, supported capabilities or data collections, results data formats or data access policies affect interface maintenance. Each time one such change happens it could be necessary to rewrite search logic in the databases or modify the arrangement or elements of a web form, potentially on different web portals. This takes time and is also error prone, so a standardized and reliable way to manage this issue was sought for.

It was thus decided to develop a dynamically configurable Java EE web application that can set itself up reading needed information from XML configuration files and from a TAP_SCHEMA.[1]

A cooperation of various components is necessary both to build the web application and to run it.

In particular, a TAP_SCHEMA Manager application and a Portal Generator Wizard are used to facilitate the portal administrator in the generation of a custom search interface to the astronomical archive he or she is managing.

The generated portal uses external services to manage authentication and authorization of users and allows tar archives creation exploiting another external service based on the Universal Worker Service Pattern[2] (UWS) from the IVOA proposed recommendation, integrating OpenCADC UWS library provided by the Canadian Astronomy Data Centre (CADC[†]).

These set of applications takes also advantage from some web services provided by the Centre de Donnes astronomiques de Strasbourg (Strasbourg astronomical Data Center - CDS[‡]).

In the following sections we provide details about all the components: TAP_SCHEMA Manager (Sec. 2), Portal Generator Wizard (Sec. 3), UWS Tar Generation Service (Sec. 4) and Generated Portal (Sec. 5).

[*]IA2: www.ia2.inaf.it
[†]CADC: http://www.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/
[‡]CDS: http://cds.u-strasbg.fr/

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 991344 · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2232594

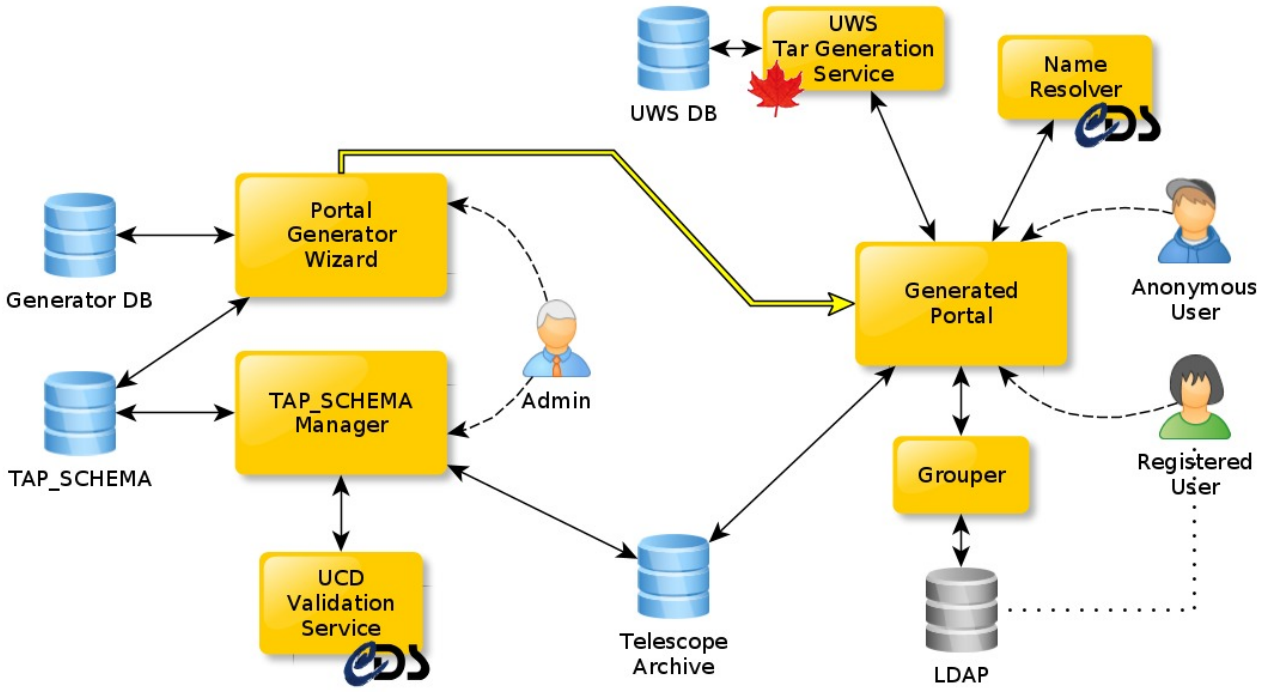

Figure 1. Portal generator system components

2. TAP_SCHEMA MANAGER

TAP_SCHEMA is a particular database schema defined in the Table Access Protocol (TAP) IVOA Standard. It is intended for containing metadata related to a TAP service, as schemas, tables and columns exposed by the service or datatype, indexing, description and other information about columns.

In the context of portals generation we exploit the TAP_SCHEMA in the configuration phase, to load information about the astronomical archive structure, and in the generation phase, to dynamically integrate some information (for example column descriptions) into the generated code.

The TAP_SCHEMA Manager is a Java EE application that provides a graphical user interface that can be used to create or edit a TAP_SCHEMA also by people that don't have specific SQL skills.

The application is composed by two modules:

- A data layer application based mainly on the Java Persistence API (JPA)
- A Java Server Faces (JSF) web application

The data layer could be used as a stand-alone Java library and is a dependency both of the TAP_SCHEMA Manager web application and the Generator Wizard.

Unlike the standard, our TAP_SCHEMA Manager allows the TAP_SCHEMA to have an arbitrary name and to be located both on a different database server and a different machine from the source schema, although TAP-1.1 goes this same direction. This implies also the support for multiple TAP_SCHEMA schemas on the same database server.

2.1 Why JPA?

JPA is a Java specification for mapping relational databases into Java objects. As most of the Java EE standards this specification has various implementations (currently the TAP_SCHEMA Manager uses the one known as EclipseLink).

It was chosen mainly for the following advantages:

- it makes the application more portable allowing to interact with a relational database in the same way regardless of the used relational database management system (RDBMS);
- has the ability to automatically manage dependencies between the database entities (for example if an entity is removed all the entities that refer to it by a foreign key are automatically removed and this works also for complex and multiple references);
- some implementations (as EclipseLink) support also schemas and tables creation, so TAP_SCHEMA structure generation is totally delegated to the JPA.

In the case of the TAP_SCHEMA Manager, since it has to execute some particular queries that load metadata, it has been necessary to maintain some specific SQL queries (for example to query the MySQL `INFORMATION_SCHEMA`). Anyway the use of JPA allows to avoid a lot of SQL code and makes future porting of the code to support other RDBMS easier.

2.2 UCD validation

Unified Content Descriptors (UCD) is an IVOA standard who defines a controlled vocabulary for describing astronomical data quantities.[3]

The `TAP_SCHEMA.columns` table, among the information relative to columns exposed by the TAP service, includes also UCDs.

UCDs are strings that must follow a defined format and allowing simple manual insertion could be error prone, so the TAP_SCHEMA Manager interface includes also a panel to perform search and validation of Unified Content Descriptors (UCDs) based on CDS dedicated services[§][¶].

Since this are online web services that could be temporarily unavailable the panel allows also manual UCD insertion, with a limited validation.

3. PORTAL GENERATOR WIZARD

The Portal Generator Wizard is a JSF web application that helps the administrator to create and configure the portal web application.

After a portal administrator logged in into the Wizard application he or she can create or edit a portal configuration. To do this correctly the TAP_SCHEMA schema should be already edited with the TAP_SCHEMA Manager.

Each administrator can create multiple portal configurations that are stored in a database in XML format. A human-readable format was chosen in order to allow manual edit of the configuration in special cases. JAXB, the Java Architecture for XML Binding, is used to automatically map the XML to the classes used by the Generator.

3.1 Configuration steps

The Generator Wizard has various pages whose web forms and JavaScript components allow to insert configuration data. In the final step the Generator Wizard uses this configuration to generate the war package of a new portal.

[§]CDS interface with various UCD tools: http://cdsweb.u-strasbg.fr/UCD/tools.htx

[¶]CDS interface that suggests an UCD from a description in natural language: http://cdsweb.u-strasbg.fr/UCD/cgi-bin/descr2ucd

3.1.1 Tables configuration

The generated portal needs to know how to build the SQL statements for querying its astronomical archive. The main challenge is to instruct the portal about performing SQL JOIN clauses.

Analyzing the database structure of the existing archives of the "Telescopio Nazionale Galileo" (TNG) and the "Asiago Astronomical Observatory" (AAO), both managed by IA2, we have seen that both have a main table containing summary data (observation id, date and time, astronomical coordinates of the object, ...) and a collection of secondary tables, one for each telescope instrument.

So, in its first step, the wizard asks to the user to specify a main table and, optionally, a list of secondary tables.

If a foreign key constraint from a secondary table to the main table is found in the TAP_SCHEMA this will be used in the JOIN clause, otherwise the generator wizard asks to the user to specify also a foreign key constraint and update the TAP_SCHEMA adding a fictitious key.

Currently the tables can be joined only in this simple way but in the XML configuration specifying multiple join columns is allowed, so in the future the system could support more complex kinds of join.

3.1.2 Files/UWS configuration

In the files configuration step the user specify the URL of the UWS Tar Generation Service and which table fields store information as file names and files paths.

Currently the IA2 archives use one column to specify the file path and another for its name. Table could be both the main table or a secondary table.

3.1.3 Logging configuration

The archives save information about executed queries and downloaded files in a dedicated table. The Wizard interface allow to select fields to store:

- date and time;
- user name;
- IP address;
- the executed query string;
- the downloaded file name and path;
- a download code that can be customized to specify different kinds of download (for example to distinguish a single file download from a tar download);
- error code and error description if an exception happens during the query or the reading/writing of the file;

3.1.4 Users configuration

IA2 data access policy stipulates that all metadata is public and some images can be downloaded only by authorized users. A column in the archive database identifies the policy of the file (public or private) and another column specify the name of the group of users that can download a specific file. Both this fields are configurable.

A group named "ADMIN", who has full access to the data, is also expected.

User groups are loaded on user login calling a Grouper web service. The service returns groups with their full name (a sequence of strings colon separated that include parent groups) but usually the database column contains only the specific group name, so in this phase the user can also specify a groups prefix.

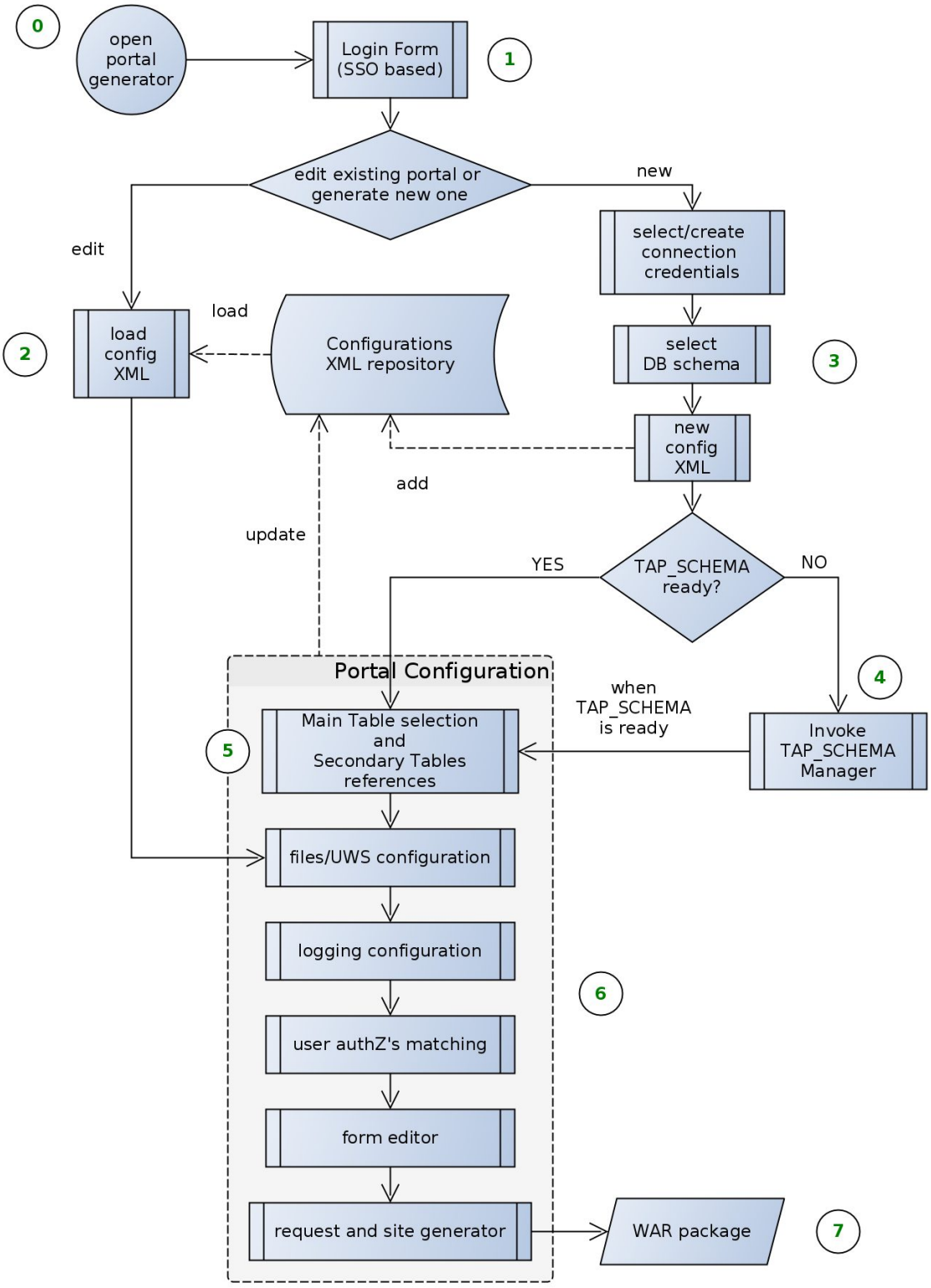

Figure 2. Portal Generator Wizard flow

3.1.5 Form editing

Form editing is the most interactive step of the Wizard and it heavily relies on JavaScript features, so in this Wizard step JSF interacts with the JavaScript framework AngularJS. It was chosen due to its ability to easily render elements of the DOM dynamically.

The user interface allows to create and select some entities called "grids" and shows the selected one as a 32-columns lattice with an increasable numbers of rows.

The first grid is named "main", is created by default and can't be renamed or deleted. It represents a content that will be always visible on the generated portal.

Secondary grids will be placed on a tabbed panel (visible if the "advanced search" mode checkbox is checked) and only the fields belonging to the visible tab will be involved in the generated query. The advanced search is mainly intended for searching on a single instrument of the telescope, but could be adapted also for other uses.

On each grid the user can add various components:

- **Text labels**
- **Inputs** Input fields with an optional label.

  They could be:

  - Simple inputs: single input element;
  - Range inputs: a pair of input elements used to specify a range of values;

  For each input the user has to specify the related database table and column and, for simple inputs, he or she can choose also the comparison type (equals or like patterns).

  It is possible to add validation constraints to the inputs, for example to force a value to represent a date in a specific format or a numeric value (in that case it is also possible to specify minimum and maximum values).

  Morever the user can choose a "dropdown" input type to show a list of selectable values on the input. A special case is the "ON/OFF dropdown" which corresponds to a boolean selector input.

- **Positional search** Special component used to select a region in the sky using RA and Dec coordinates and a radius. Theoretically a cone search should be implemented but this implies specific database requirements, like specific plugins for geometry data types in the RDBMS (for example pgSphere[‖] for PostgreSQL), or dedicated indexing through tessellation of the celestial sphere. So currently the search is performed on a box shaped region.

  On this component the user can also include a text input that translate the name of an astronomical object in its RA/Dec coordinates calling the Sesame Name Resolver service of the CDS[**].

All this components could be placed on the grids moving them with the mouse and also their size could be adjusted dynamically.

3.1.6 WAR package generation

Portals war package generation starts copying a package skeleton which contains all classes and files used by all portals.

Then, both the configuration XML and TAP_SCHEMA metadata are read to build the xhtml search form page and a new XML configuration file (to distinguish the two XML files hereafter we will call the first "editable configuration" and the second "generated configuration").

[‖] pgSphere: http://pgsphere.projects.pgfoundry.org/
[**] Sesame Name Resolver: http://cds.u-strasbg.fr/cgi-bin/Sesame

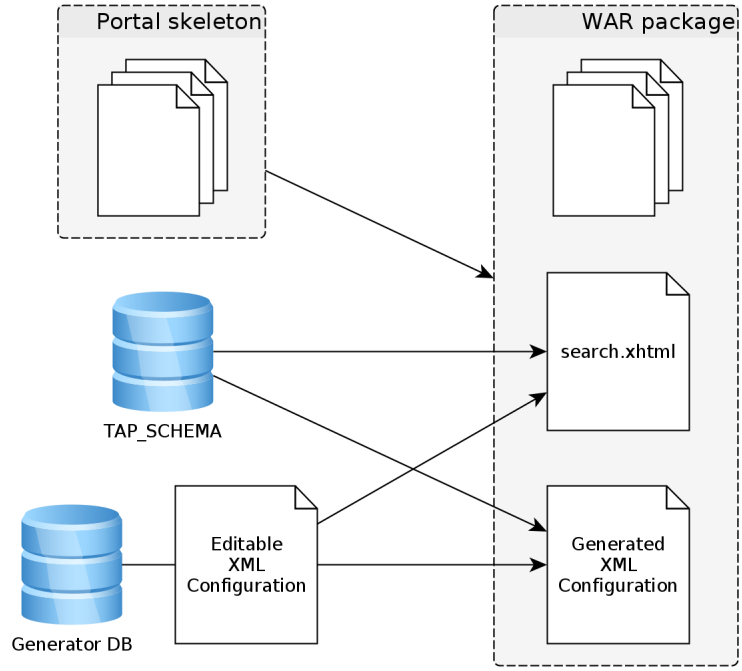

Figure 3. Generation process

Editable configuration information that is not regarding the search form position is copied into the generated configuration, while the remaining information is rearranged and split into the xhtml search form page and into the generated configuration (see Fig. 3). Indeed, during the generation process each text input added to the generated xhtml page is marked with an incremental ID and its associated information is put in the generated configuration. TAP_SCHEMA metadata are used in this phase too, so one could maintain the same editable configuration and produce a new portal (obviously this works if one edits existing metadata, as changing a column description, but problems could occur if big structural changes, as tables removal, are made).

Note that the generated configuration is designed only to load portal information when the web application starts and could be written in a pure machine-readable format, as a file containing serialized objects. It is maintained in XML only for debugging purposes and, if necessary, to allow quick modifications, as changing the database port number.

The portal skeleton is a Maven project, so Apache Maven Invoker API is used to build the project from the Wizard web application.

Currently the Generator Wizard can build a war package designed for GlassFish or Tomcat, but with small arrangements others application servers could be supported. Basically Tomcat support was achieved adding missing dependencies (Java EE components that the more complete GlassFish includes by default) and editing some configuration files.

4. UWS TAR CREATION SERVICE

Besides single file download, when a search is performed on the generated portal, search result rows with downloadable images can be selected and all images can be downloaded collectively as a tar archive. If the archive is bigger than 500 MB it will be split into multiple files.

Big tar creation may take some time and errors could happen, so this task is delegated to and external web service that uses the Universal Worker Service Pattern (UWS) from the IVOA proposed recommendation.

The UWS Pattern provides a standardized way to execute long asynchronous jobs and monitoring their state. When a job is initialized information related to it is stored in a database. The information is then updated each time the job change its state.

The idea is that a single Tar Creation Service could be used by multiple portals. The service needs two tables to store the data but we want to maintain different portals data separately, so the service was structured to have a pair of tables for each portal and an univocal prefix is used to differentiate the portals from each others.

Obviously when a new portal is deployed its two tables in the UWS database need to be created. Currently this is done using a script but implementing a small administrator panel on the UWS server could be useful.

UWS standard specifies that a job representation can contain an element called "jobInfo" that can be used by implementations to include any extra information within the job description. This element is used by the portal to send the list of selected files to the UWS tar service. The portal retrieve file paths directly from the SQL query it executed. Tar Creation Service loads the specified files from its filesystem, so, if the physical files are stored in another location, solutions like the Network File System (NFS) have to be set up.

UWS job details contain also some parameters that can be added both when the job starts or when it ends. Result parameters are used to store the name of the generated tar archive. A combination of the date when the archive was generated and a pseudo-random string has been chosen for the file name.

Generated portals allow tar creation also to anonymous users. In this case the user ID is a pseudo-random generated string which will then stored in a cookie.

Tar archives of each user are stored in a directory whose name is the ID of the user that owns that files. A better approach to this folders structure could be to implement the VOSpace IVOA standard.

Users can delete generated files (on the UWS service database the job is marked as "Deleted By User" and then the physical file is deleted from the disk), but our portal policy stipulates that in any case tar archives older than a month will be removed. This is done by a script that runs periodically.

The web service is based on the UWS library provided by the CADC[††], whose servlets have been transformed into REST services that follow the JAX-RS standard.

The service has URIs to:

- start a new job (tar creation);
- delete a job (tar archive removal);
- get current jobs list (tar archives list returned in JSON format);

When the portal asks to start a new tar generation it also begin to request periodically the jobs list (polling). When there are no more jobs in progress the polling stops and the user is notified about the results.

4.1 Security considerations

It should be noted that, since the UWS standard is intended for jobs management, the UWS service is used only to generate tar files and get information about them and not to download the files themselves (when a portal user asks to download a file is the portal itself that reads the file, possibly by NFS). In practice the service is accessed always from the portal itself and never from other clients. Also in the polling case the AJAX call is sent to the portal and then is the portal that calls the UWS service and send a modified response to the browser.

Anyway, since the UWS service could be hosted on a public machine along with other services, security considerations are necessary.

If the UWS service was accessible to external clients, one would have read the jobs list of a specific user and would have use it to delete his or her jobs. This would be annoying for that user but is not really dangerous, since jobs don't contain sensitive data and tar archives could be created again.

[††] https://github.com/opencadc/uws/

A really more dangerous security threat that must be avoided is the possibility to start a new job using a list of arbitrary files. In fact, as mentioned above, the portal passes the list of desired files paths to the UWS service and this behavior could be exploited by a malicious user to retrieve any file on the server.

A first security measure could be to allow the UWS service to access only to a set of folders that contains data related to the portal. It could be nevertheless possible to access to data destined to some specific registered users, so basically we want to allow that only portals web applications can access the UWS service. An IP-based filter could be implemented but it could not be safe enough since IP spoofing is possible, so it is necessary that the UWS service authenticates the portals that use it. Best way to do this could be establishing HTTPS connections and use a password or a mutual authentication certificate-based.

The disadvantage of this approach is that it complicates portals configuration and makes the generated war package not more a ready-to-deploy product. In fact HTTPS should be configured directly on the server that hosts the portal and this involves the use of certificates, preferably signed by third-party authorities. If the certificate is self-signed portals themselves may have to import the UWS certificate into their truststore (the file that contains set of certificates of trusted certification authorities) and in the case of mutual authentication the UWS service has to import portal certificates too. Moreover the HTTPS and authentication configurations could vary a lot between different application servers.

If, due to this issues, portals administrators don't want to use HTTPS, portals and the UWS service should at least have a way to encrypt the files list and, possibly, the jobs IDs. This could still allow reply attacks but they don't represent a severe threat (in the worst case a user could see duplicated tar files).

Currently the Portal Generator Wizard assumes HTTPS is used and, during the war package generation, it tries to retrieve the UWS server certificate from the URL specified in the configuration. If the certificate can be obtained the Wizard imports it into the portal truststore, otherwise the war package will be produced anyway and a warning message will be shown.

At the moment the portals share a secret key with the UWS service. The key value, that is sent with each message, can be configured in the Wizard and has to be put manually on the UWS server. In the future the Wizard could support other ways to authenticate the portals on the UWS service, allowing to choose one of them during the configuration process.

A last concern to be discussed is the possibility of DoS (Denial of Service) attacks. Pointing out that no web service could be totally immune to this kind of threat, should be noted that a user (possibly also a simple anonymous user that accesses the portal in the legitimate way) could potentially asks to create a tar with 500 files (the default maximum number of rows displayed in the portals). Each one of this files is a gzipped FIT image, averagely bigger than 10 MB. It is therefore clear that a such action, repeated multiple times, could rapidly consume all space on the UWS server disk. Hence, the service should have a way to limit the number of files added into tar archives in given a period of time, at least for anonymous users.

5. GENERATED PORTAL

The generated portal is also a JSF web application. Portal main page is a search form to query the database of the astronomical archive and retrieve a list of astronomical images and their associated metadata.

5.1 Private data management

Access to part of the data is expected to be restricted to some specific users, so the portal has a login form and after an user is authenticated using a Single Sign On system the portal connects to another web service to retrieve the list of groups to which they belong.

Currently the portal can be configured to manage authorization using Grouper, an enterprise access management system provided by the Internet2 Community.

Grouper was chosen because of its ability to delegate children groups creation to non-administrator users. In this way a principal investigator (PI) can share some programs with his/her collaborators without be forced to share the entire account permissions.

At the moment the authentication is based on multiple LDAP servers (on the form login there is a dropdown that allows the user to select his or her organization), but in the near future probably Shibboleth, a federated identity solution provided by Internet2 too, will be used.

5.2 VOTable

VOTable is an IVOA standard that specifies a XML format for representing tabular data.[4]

Generated portals can convert the query result list into a VOTable using the Starlink Tables Infrastructure Library[5] (STIL), developed by Mark Taylor at the University of Bristol.

STIL supports loading data directly using an SQL query. In the portal however the query has already been executed when the user asks for the VOTable, so the table is built using objects loaded in memory.

Generated VOTable can be both downloaded as an XML file and sent using the SAMP protocol (see Sec. 5.3).

5.3 SAMP

SAMP (Simple Application Messaging Protocol) is an IVOA standard that improves astronomical software interoperability.[6]

Communication in SAMP is realized using a star-like topology, in which a "Hub" acts as an intermediary between the various applications that support the protocol and were registered on the Hub.

SAMP has various ways to map the operations in its abstract interfaces to specific I/O operations and these ways are called "Profiles". In particular there are the Standard Profile, suitable for desktop applications, and the Web Profile, suitable for desktop applications.

Generated portals support the SAMP Web Profile using the samp.js JavaScript library developed Mark Taylor.[7] This library exploits cross-origin AJAX requests to call a specific port of the localhost, where the Hub is listening.

Each SAMP message contains a "MType": a string that define what kind of message it is.

Portals can send messages with the following MType:

- `table.load.votable`: send a VOTable;
- `image.load.fits`: send a FITS image;

A portal calls periodically localhost on the SAMP Web Profile port to check if a Hub is active. In that case an icon becomes green and it is possible to register on the Hub. When the portal asks for registration the Hub shows an alert message and asks the user to confirm that he or she wants to accept the portal request. This is a security measure used to avoid that undesirable applications register to the Hub.

After the portal is registered a list of active applications is loaded in a menu and the user can select one of them as a target for the messages. Is no application is selected messages will be sent anyway, broadcasting them to all recipients.

In practice, both the MTypes used by portals don't put directly the requested data into the message, but expect to have an argument named "url", whose value is the location of the document to be loaded. Since VOTable are generated on-the-fly and images could be private this URLs can't be fixed. For this reason data relative to each SAMP request is stored in an object contained in the user session and a temporary URL is created to allow access on it. This implies that the URL will not be avaliable anymore after user logout, but this isn't a problem because these URLs should be accessed a few moments later the SAMP message is sent.

6. FURTHER NOTES ABOUT INVOLVED TECHNOLOGIES

6.1 Java Server Faces

Most of IA2 services are running on GlassFish servers, so using Java EE technologies has been a natural choice.

In this context JSF has been chosen as the framework to build the user interfaces of all described applications. Compared to JSP it offers more abstractions (as an automatic way to update small parts of a page using AJAX or prebuilt validation tags) so it could result into cleaner code and faster development.

This advantages however are possible only assuming to have a good knowledge of the framework, specially if it has to be used in "borderline" situations.

For example AngularJS integration wasn't painless and implied writing a small extension to the JSF component that manages the partial rendering of the page.

Another detail that has been necessary to consider in order to prevent undesirable behaviors is the fact that JSF AJAX requests are queued so that they look like as if they are synchronous. This affected the development of polling features, that we had to manage separately using the JAX-RS (REST services) standard. In fact, although JSF provides JavaScript API to manually send AJAX calls, these are intended only for performing partial page updates and not to be run periodically in background.

6.2 Context and Dependencies Injection

Along with JSF also the CDI Java EE standard is used. CDI (which reference implementation is called Weld) allows to inject JavaBeans into others or recalling them by name inside JSF pages using the Expression Language (EL).

CDI also allows to define a "scope" in which a JavaBean exists. CDI `@SessionScoped` annotation is very useful to automatically manage the data associated to an user session.

Also the `@ConversationScoped` annotation is largely used in this projects. It defines a scope related not only to the user session but also to the browser tab. In this way in the Portal Generator Wizard it is possible editing multiple portals simultaneously in the same user session and parallel searches are permitted on the generated portals.

6.3 Bootstrap

Bootstrap, an open source front-end framework originally developed by Twitter, was used in the Portal Generator applications suite.

It was chosen because it is one of the most popular front-end framework and it is easily customizable.

It has a good grid system that was exploited in the Portal Generator Wizard search form editor to create the editable grids. In that case the default 12-columns grid was replaced with a 32-columns grid to allow better precision in component positioning.

7. FURTHER DEVELOPMENT

Some possible improvements are already been mentioned:

- using VOSpace to store portal users tar files;
- realize a real Cone Search instead searching on a box shaped region (or even enable generic geometry searches);
- using Shibboleth as Single-Sing-On service;
- support additional way to authenticate portals on UWS service;
- implement UWS service admin panel;

- improve offline UCD validation;

Regarding the generated portal it could be useful to improve logging (for example adding the user-agent).

In near future will be also necessary to support database structures different from the simple model having a main table joined with secondary tables. In fact for Medicina[‡‡] and Noto[*] radiotelescopes, whose archives are managed by IA2 too, the queries are performed using some User Defined Functions (UDF).

To support this particular architecture a new branch has to be added to the Portal Generator Wizard flow showed in Fig. 2, after the database selection, specifying if the portal will use a model based on UDFs or TAP_SCHEMA.

Furthermore, an architectural improvement that we have considered is setting up the generated portals with a reverse proxy solution, as HAProxy, to have a better service availability and a balanced load.

REFERENCES

[1] Dowler, P., Rixon, G., and Tody, D., "IVOA Working Draft - Table Access Protocol." IVOA Documents, 28 April 2016 http://www.ivoa.net/documents/TAP/. (Accessed: 24 May 2016).

[2] Harrison, P. and Rixon, G., "IVOA Proposed Recommendation - Universal Worker Service Pattern." IVOA Documents, 05 May 2016 http://www.ivoa.net/documents/UWS/. (Accessed: 24 May 2016).

[3] Derriere, S., Gray, N., Mann, R., Martinez, A. P., McDowell, J., Glynn, T. M., Ochsenbein, F., Osuna, P., Rixon, G., and Williams, R., "IVOA Recommendation - An IVOA Standard for Unified Content Descriptors." IVOA Documents, 19 August 2005 http://www.ivoa.net/documents/latest/UCD.html. (Accessed: 24 May 2016).

[4] Ochsenbein, F., Williams, R., Davenhall, C., Demleitner, M., Durand, D., Fernique, P., Giaretta, D., Hanisch, R., McGlynn, T., Szalay, A., Taylor, M., and Wicenec, A., "IVOA Recommendation - VOTable Format Definition." IVOA Documents, 20 September 2013 http://www.ivoa.net/documents/VOTable/. (Accessed: 24 May 2016).

[5] Taylor, M. B., "TOPCAT & STIL: Starlink Table/VOTable Processing Software," *Astronomical Data Analysis Software and Systems XIV ASP Conference Series* **347** (2005).

[6] Taylor, M., Boch, T., Fitzpatrick, M., Allan, A., Fay, J., Paioro, L., Taylor, J., and Tody, D., "IVOA Recommendation - Simple Application Messaging Protocol." IVOA Documents, 11 April 2012 http://www.ivoa.net/documents/SAMP/. (Accessed: 24 May 2016).

[7] Taylor, M. B., Boch, T., Fay, J., Fitzpatrick, M., and Paioro, L., "SAMP: Application Messaging for Desktop and Web Applications," *Astronomical Data Analysis Software and Systems XXI ASP Conference Series* **461** (2012).

[‡‡]Medicina Radio Astronomical Station: http://www.med.ira.inaf.it/
[*]Noto VLBI antenna: http://www.noto.ira.inaf.it/