Astronomical Data Analysis Software and Systems XV
ASP Conference Series, Vol. XXX, 2005
C. Gabriel, C. Arviset, D. Ponz and E. Solano, eds.

Accessing the EGSO Grid through a WSRF-Enabled API

André Csillaghy, Marco Soldati, Philipp Kunz

Department of Computer Science, University of Applied Sciences Northwestern Switzerland, CH-5210 Windisch, Switzerland

Robert D. Bentley

Mullard Space Sciences Laboratory, University College London

Isabelle Scholl

International Space University, Strasbourg Campus, France

Abstract. EGSO is part of the Virtual Solar Observatory. It provides a Grid for integrated access to distributed and heterogeneous solar data archives. Here we describe the motivation and approach we have chosen to implement the Grid access as a stateful web service. Using this kind of service allows us to consider the search and retrieval of data as an interaction between the client and the Grid. This way, the EGSO service can be merged into interactive data analysis systems such as Python or IDL.

1. EGSO in short

The European Grid of Solar Observations (EGSO, www.egso.org, Csillaghy et al, 2005) just finished the production of a data grid middleware integrating the access to solar data. EGSO provides the fabric of a virtual solar observatory. Using EGSO, everyone can straightforwardly access many available solar observations over the Internet. EGSO is an unprecedented service for the solar physics community. In addition, this project also provides a test bed for data grid research. Data grids allow exploiting the large potential of combining distributed archives. In this context, the EGSO experience links the specific needs of the solar physics community with fundamental Grid research.

EGSO allows querying in a unified manner distributed resources such as solar event and feature catalogs, observation catalogs and eventually retrieving the associated observations. EGSO can be accessed in two different ways. The first way is to use a web-based GUI. This allows a straightforward use of the system for inexperienced or occasional users. The second way is to use directly the Application Programming Interface (API) on which the GUI is built. This is the subject of this paper.

Using the API offers more flexibility than the GUI, and can provide complete access to the system from interactive environments such as IDL or python.

2. EGSO Architecture

EGSO is divided into three roles: the broker, the provider, and the consumer (Figure 1). In this sense the system follows the principles of Service-Oriented Architectures (SOA, Booth et al, 2003). These three roles can be replicated and distributed, hence allowing an appropriate load balancing.

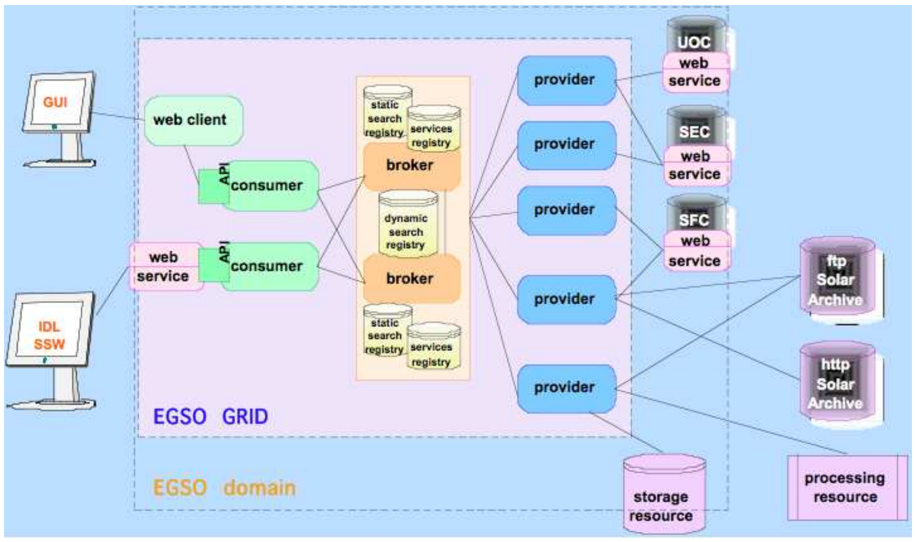

Figure 1. The EGSO architecture.

Clients get into the system either by connecting to the Web client or to the EGSO Web service that we will discuss later. Both are linked to the core consumer role by a unique API. Consumers can be duplicated as needed.

Consumers prepare queries and send them to a broker. The broker uses a service registry to locate rapidly which services are most adequate to resolve a query. The static and dynamic search registries allow some “easy” queries to be answered at once, without having to connect to the providers at all. Assuming that easy queries are more frequent, the broker can answer a significant number of queries on its own.

When the broker is unable to resolve a query or part of it, it gets help from one or several provider roles. These roles map the EGSO internal global data model to the schemata of the individual data providers. Special providers are used for several purposes: (1) To connect to the Unified Observing Catalog, which is summarized in the registry, and contains a list of all observations made by instruments registered with the system; (2) To connect to the the Solar Feature Catalog (SEC), which allows to search data by features such as filaments or active regions; (3) to connect to the Solar Event Catalog (SEC), which allows to search data by events such as solar flares and coronal mass ejections.

3. The EGSO API

3.1. Methods

The EGSO API features methods to discover, search and retrieve information in the system. The discovery tools includes the two following methods:

- `findAllParameters`: Allows to discover which parameters are registered in the system.
- `findAllResults`: Allows to discover which type of results will be returned by the system once a query is issued.

The search and retrieval tools includes the two following methods:

- `setValue`: Allows to pass a value to a given parameter of the internal data model.
- `getResult`: Allows to get the result associated with the parameter values set at the time of the call.

Additional methods are necessary to establish the relation between the parameters and the results. For this, a method `affects` and a method `isaffectedby` allow respectively to find out which parameters will affect a given result, or vice-versa.

EGSO supports parameter result independence: New parameters can be added to the data model without having to change the API. The new parameters get fully accessible through the API. Furthermore, the thin interface allows client interfaces to be built on only the two functions `setvalue` and `getresult` (of course at the cost of many configuration parameters).

All system messaging is done using VOTABLE (Ochsenbein, 2004).

3.2. How the API is used

The EGSO system works best in an interactive environment. Users will usually search for data first in a broad manner, identifying the information they need, and then will use increasingly accurate parameters. A typical session can be sketched work as follows:

1. The query resource is generated. A client (either a person or an application program) creates an instance of an EGSO object, with which he/she/it can use to access the EGSO domain.
2. The client explores the parameter space with the method `findAllParameters`. First, meta data information will be searched, from catalogs such as the UOC, SFC, or SEC. For instance, one can search information on *solar flares*.
3. The query parameters are set with the method `setValue`. Continuing with the example of solar flares, the values set here can restrict the time interval, the size and location of the solar flares to consider.
4. The query is sent to the system with the method `getResult`.
5. The query result is used as a starting point for the next selection operation. The method `setValue` is used again. The new selection criteria can involve other parameters, for instance restricting which observation instruments should be considered.
6. The method `getResult` is used again to obtain the list of observations.
7. The selection go on further by selecting the relevant observations with `setvalue`.

8. Finally, URLs associated with the data files are obtained with `getResult`. The associated files can be downloaded directly from the sites where they are located.

The API is built as a stateful web service. It follows the Web Service Resource Framework standard (WSRF, Foster et al, 2004). In particular, a WSRF-enabled WSDL file allows to generate stubs in virtually any language. These stubs can then be used in other applications. Furthermore, the WSRF standard allows the system to be integrated also into Web-service work flow orchestration tool.

4. Addressing the interactivity requirements

The work flow given above works best when used interactively. Interactivity is a requirement for many scientists. Opposite to batch processing, in many cases scientists want to be able to go back and forth between the different steps of this work flow, to be able to modify selection parameters and get new results. Furthermore, many scientists use scripting languages such as IDL or Python. This has two implications. First, *sessions* are needed, i.e. we cannot rely on stateless Web Services. Second, clients must be generated in several languages.

For these two reasons, we have followed the Web Service Resource Framework specification to create a stateful web service for the EGSO system. The service is directly attached to the consumer and can be integrated with other application, in principle written in any arbitrary language (with automatic "stubs" generation).

On the client side, we have built the prototype of an IDL client that binds directly with the EGSO Web Service. The IDL client uses the IDL-Java bridge. For example, an IDL EGSO object can be generated with
`o = egso()`.
The parameter space can be explored with
`o->which`.
Parameter values can be set with
`o->set, parameter = value`,
and results can be obtained with
`result = o->get( /parameter )`.

The VOTABLE-formatted results are transformed into IDL structures with a self-written IDL VOTABLE parser.

References

D. Booth et al., 2004, www.w3.org/ws-arch/

Csillaghy, A., Bentley, R.D., & Scholl, I., 2005, in Proc. VO Conf. Sofia.

Foster, I. and 12 co-authors, 2004, white paper, www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

Ochsenbein, F., Williams, R., Davenhall, C., Durand, D., Fernique, P., Hanisch, R, Giaretta, D., McGlynn, T., Szalay, A., and Wicenec, A., 2004, *Proc. Toward an International Virtual Observatory.*