Astronomical Data Analysis Software and Systems XII
ASP Conference Series, Vol. 295, 2003
H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

Data Management for the VO

Patrick D. Dowler

Canadian Astronomy Data Centre, Herzberg Institute of Astrophysics
5071 W. Saanich Rd. Victoria, BC, Canada

Abstract. The Canadian Astronomy Data Centre has developed a general purpose scientific data model and an API for accessing a scientific data warehouse. The Catalog API defines a general mechanism for exploring and querying scientific content using a constraint-based design. The API provides access to separate but related catalogs and allows for entries in one catalog to be related to (usually derived from) entries in another catalog. The purpose of the API is to provide storage-neutral and content-neutral access methods for scientific data. The API defines a network-accessible Jini service.

We have developed the Canadian Virtual Observatory (CVO) as Jini services that implement the Catalog API. These catalogs store astronomical content: the pixel catalog provides uniform access to our many archival data holdings, the source catalog stores the results of image analysis, and the processing catalog stores metadata describing exactly how sources are extracted from pixel data. Entries in the source catalog are connected to entries in the processing and pixel catalogs from which they are derived.

1. Introduction

The primary design goal of the CVO is to enable science across the entire electromagnetic spectrum. We have achieved this by recognizing that the engineering principles at work in different wavebands are quite different and by developing a system which abstracts various types of VO content in order to remove these engineering details and complexity. The result is a data model that is sufficiently general to be used by all astronomers.

Reproducibility is the cornerstone of good science. As such, all content in the CVO includes a pedigree so that astronomers can examine exactly how any quantity was computed. This transparency allows for peer review by the user community and open, objective science.

The Catalog API developed at CADC provides a mechanism for programmatic remote access, discovery, and exploration of scientific content. The API models both the content and the data flow and allows full pedigree navigation via *derived from* links.

The CVO site holds a collection of catalogs: pixel data, processing instances, source, objects, and other products of processing. Each of these catalogs is an implementation of the Catalog API; they service as an excellent test-bed for VO content management and interoperability issues.

2. Abstraction

The reality of astronomical data services is that they are either data archives storing pixel data or they are static catalogs reproduced from referred publications. In the case of data archives, these services are filled with engineering concepts and terminology[1] which varies from one archive to the next, even from a single service provider. Furthermore, differences in the engineering challenges across the spectrum remain as artifacts in the archives, making multi-wavelength[2] science difficult if not impossible. Every archive is different and cross-archive research suffers as a result.

One can abstract all of these technical details into a relatively simple set of pixel data properties that describe the spatial, spectral, and temporal coverage and resolution of the observation. These *sampling properties* recognize the three-dimensional nature of astronomical observations and provides an analog description that is useful across the entire spectrum.

It is also useful to include some description of the actual content of the observation distinct from the intended content (usually derived from the observing program). These *content properties* would normally be measured from the pixel data and include quantities like the number of point sources, number of extended sources, density of sources, and detection limit for point sources.

For derived catalogs, the situation is not much better than for data archives. The catalog properties tend to carry their engineering heritage (baggage). In the optical regime, magnitudes are commonly used and each is named for the filter or filter system from which it is derived. With new filter systems being deployed for specific scientific goals - typically in large surveys - the list of different magnitude systems is growing and there is no straightforward way to aggregate the information. Astronomers working in other parts of the spectrum tend to use flux density rather than magnitude as a measure of brightness. Across the spectrum, there is no consistent use of wavelength, frequency, or energy - pick your favorite. In order to place all derived properties on the same *coordinate system* we have adopted a multi-dimensional approach. For all wavelength-dependent properties, we store and work with a function[3] $f(\lambda)$ and retain the analog nature of the spectral sampling. Clearly this is derived from the abstraction used for pixel data and avoids dragging the engineering heritage into derived catalog content.

---

[1]telescope, instrument, CCD, filter, grating, grism, etc.

[2]even wavelength is only commonly used in part of the spectrum

[3]the choice of λ is arbitrary ; we could use ν or E instead

3. Data Types

The abstractions described above for pixel data and derived catalog content require a set of data types not normally used for catalogs. In choosing the data types, we have to consider whether the types capture all of the information and whether they can be used in the context of a scientific data warehouse. That is, we must choose data types that can be indexed for fast access and searching within available database management systems

3.1. Pixel Properties

The required data types for describing the observational sampling are (1) a polygon on the surface of a unit sphere for spatial coverage, (2) an interval of floating point values for spectral coverage, and (3) an interval of date and time values for temporal coverage. In addition, one must also store the spatial, spectral, and temporal *span* (the size of the coverage) and the spatial, spectral, and temporal *resolution* (the size of the one resolution element), all of which are scalar (floating point or integer) values.

The content description properties are generally integer or floating point values; the dataset name and archive name are strings (or URLs) which enable retrieval of the data. All of these are primitive types easily stored and indexed by database systems.

3.2. Source Properties

The required data types for source properties fall into three categories: positional, wavelength-dependent, and wavelength-independent quantities.

For positional information, we require a point on the surface of a unit sphere and, for extended sources, an ellipse or other polygon (also on a sphere). The error value for points is typically an ellipse.

For wavelength-dependent properties (flux, size, shape, etc.) we are actually storing the floating point interval (spectral coverage) from the pixel data and a floating point value (scalar). This is a function evaluated (estimated, measured) over a fixed interval. In graphical terms, it is a horizontal line segment (λ_1, y) to (λ_2, y) where y is the *value* of the property in question. For our purposes, we can consider these wavelength-dependent source properties as functions and we can index them as *2D line segments in Cartesian coordinates* using one of several spatial indexing schemes (e.g., R-Trees, spatial decomposition); a general multi-dimensional indexing scheme would also be effective in accessing this functional information.

The wavelength-independent properties (redshift, spectral indices, object type, etc.) are scalar values and easily stored and indexed by database systems.

4. Reproducibility

The scientific method requires that all results be reproducible by the scientific community. This is necessary to verify correctness and, in the case of scientific data services, to build trust and confidence in the quality of the service. To this end, every piece of information in the VO must have a pedigree that users can follow in order to discover exactly where a value came from and exactly how it was produced.

5. Goals of the CVO

The primary goal of the CVO project is to create a system for scientists. Specifically, the system enables astronomers to do more than just find data; they must be able to produce scientific results and to produce results that cannot be (easily) obtained through other means. The CVO captures the information content and the data flow and enables a sophisticated level of exploration and discovery.

From a technology standpoint, the CVO project uses best practices and technologies to attain its scientific goals. This means the CVO is unencumbered by backwards compatibility and some content may never find its way into the system. To these ends, the CVO is implemented as a set of Jini services using the Java programming language (Edwards 1999, Sun Microsystems[4]).

Finally, from a practical point of view, we have implemented the system in order to find all the hidden details and complexity that must be dealt with in order for the VO to be a productive scientific instrument.

6. Catalog API

The CVO data model is defined by the Catalog API. The API can be divided into three components: content, discovery, and exploration.

6.1. Content

The content of the CVO is captured in three types of objects: `Entry`, `EntryProp`, and `EntryLink`. An `Entry` is a single *thing* in a catalog. An `EntryProp` is one property of an `Entry`. An `EntryLink` is a link between an `EntryProp` and another `EntryProp`, possibly in a different catalog. Thus, the `EntryProp` is the *unit of information* in a catalog; it is made up of:

```
link - an EntryLink denoting the origin of this EntryProp

prop_id - used to query the EntryPropMap

tuple_id - used to distinguish between multiple values

value - the value of the property

error - the error in the value
```

[4] `http://www.sun.com/jini`

```
rank - arbitrary way to denote multiple values as
       better or best
```

The `EntryLink` contains sufficient information to look up an `EntryProp` or an `Entry` in an arbitrary catalog. It contains:

```
entry_id - the unique ID of the linked Entry

prop_id - the prop_id of the linked EntryPropMap

tuple_id - the tuple_id of the linked EntryProp

catalog - the name of the catalog that contains the linked
          Entry
```

Finally, the `Entry` is simply a container for `EntryProp`s with a unique ID (the `entry_id`). Thus, the data model for the CVO is quite simple, keeping in mind the variety of value types permitted within an `EntryProp`.

6.2. Discovery

The CVO discovery model operates at two levels. The first level is service discovery, which is implemented using standard Jini *discovery and join* semantics. The core Jini platform provides the capability to find a *service registrar* on the network using either unicast discovery (basically a URL to the service registrar) or multicast discovery, where the client (1) requests that service registrar(s) contact it and (2) listens for service registrar announcements. In either case, once the client software contacts a service registrar, it can use the service registrar API to look up services by *type* and/or *attributes* (like name). To access the CVO, the client would look up services of type `ExplorableCatalog` (see below).

The second level of discovery is that of discovering the type of content available in an `ExplorableCatalog`. This is done by accessing the `EntryPropMap` for the catalog and looking at the `EntryPropDescriptor`s it contains. Each `EntryPropDescriptor` describes one property of `Entry`s in the catalog; it reveals information like the name, type, units, and a description of the property. All of the information needed to query an `ExplorableCatalog` and interpret the results is available in the `EntryPropMap`.

6.3. Exploration

The base features of the API are the methods of the `Catalog` interface:

```
public EntryPropMap getPropMap();

public Entry get(Long entry_id);
```

The `ExplorableCatalog` interface extends the `Catalog` interface, adding the following exploration methods:

```
public DataModel[] getCount(ConstraintSet cs);

public DataModel[] getRange(ConstraintSet cs,
                            Property prop);

public DataModel[] getHistogram(ConstraintSet cs,
                                Property prop,
                                Interval range,
                                int nbins);

public DataModel[] getTable(ConstraintSet cs);
```

The arguments and return types specified above require some explanation. The `ConstraintSet` argument is a set of simple `Constraints` on one or more properties in the catalog (see below). The `Property` arguments specify a single property of interest (by name). The `Interval` argument defines the range of values to be included in the histogram. The `DataModel` return type is the interface implemented by all content container types; the contract for each of the above methods is that they must return the specified `DataModel` (`Count`, `Range`, `Histogram`, or `Table`) but they can optionally return other `DataModels`. This allows implementations of the `ExplorableCatalog` interface to provide extra information if it is not costly to do so. For example, if the query to get the count is essentially the same as to get the range of values, the implementation is free to return both in order to avoid executing a very similar query in the near future. Each of the methods in the `ExplorableCatalog` interface is a request for a progressively more detailed summary or view of the set of `Entrys` specified by the `ConstraintSet`.

The Catalog API also includes a `MutableCatalog` interface that defines the methods used to add, update, and remove content. This interface is described elsewhere.

6.4. Constraints

The `ConstraintSet` argument used in exploratory queries holds a collection of `Constraints`. These `Constraints` are simple query predicate components that can be put together to build arbitrarily complex queries. For all data types, one can use the `Known` and `Unknown` constraints to require the existence or non-existence of the property. For scalar data types (int, float, date, string) one can use the `Eq`, `Leq`, `Geq`, and `Between` constraints to select certain value(s). For interval types (date and float intervals are currently supported) one can use the `Intersect` constraint to specify that the interval property value contains a scalar value of the same base type (i.e., `Intersect(Interval.Date, Date)` and `Intersect(Interval.Float, Float)`). Finally, for geometric types (points, lines, circles, ellipses, and polygons) one can use the `Intersect` constraint to test for intersection of a geometric type with another geometric type. In addition, one can construct arbitrary algebraic expressions with numeric scalar data types and use them with any constraints that work with scalar data types. Thus, the constraint system is a toolbox of simple `Constraint` types which work with `Property`, `Constant`, and `Operator` arguments to specify the query predicate.

7. The CVO Experiment

The CVO project is an experiment in developing and deploying VO functionality. Our base content that has motivated the design is the CNOC1 catalog (Yee et al. 1996) and the WFPC2 Association project[5], but we have also kept in mind other survey catalogs (2MASS, SDSS, etc.) and future telescopes and instruments (CFHT Megacam, IFUs, etc.) as they all add scale and complexity to the VO content.

The CVO is an experiment in VO content management. The scale and complexity underlines the need for dynamic discovery and exploration, automation of processing, and bi-directional linkage between data and results: data flow in one direction and pedigree navigation in the other.

The CVO is an experiment in interoperability. By building the tools and infrastructure for various interdependent catalogs, we have learned many lessons about what information must be stored, what things can remain implementation details (internal), and what things must be agreed upon by all participants. The abstractions of observational data to spatial, spectral, and temporal sampling are the primary result of this interoperability experiment. In addition, we have developed an architecture which allows for remote catalog services and arbitrary linkages between the content within different catalogs.

The CVO is an experiment in integrating VO with operational systems. We treat archives as external (legacy) systems that *publish* their content to the VO. Users can initiate data retrieval (from an archive) after exploring the pixel catalog. Although we use our processing catalog to store processing details and track execution status, the data processing itself occurs in the CADC Distributed Processing System (also a Jini service).

8. Summary

The primary design goal of the CVO is to use abstraction to separate engineering and science - hide the engineering, in fact - in order to deliver uniform access across the entire spectrum. The most important aspect of the design is that pixel data should be described by the spatial, spectral, and temporal sampling of the observation. It is vital to characterize both the coverage and resolution of the sampling of an observation. In addition, data analysis can provide some useful and interesting summary information about the content of pixel data. For example, one could measure the number of point and extended sources, source density, detection limits, to name a few. All of these properties of the pixel data help astronomers to find data that is useful for their specific scientific goals in a telescope and instrument-agnostic fashion.

Reproducibility is the cornerstone of good science. As such, the VO must include access to the pedigree of every piece of information so that astronomers

[5] `http://cadcwww.dao.nrc.ca/wfpc2/`

can examine exactly how any quantity was computed. The astronomer must be able to retrieve the input data and reproduce the result. This detailed pedigree requirement is necessary so that VO content may be peer-reviewed by the user community and so that the VO can gain the confidence and trust of the users.

The Catalog API developed at CADC provides a mechanism for programmatic remote access, discovery, and exploration of scientific content. The current version is very general and not astronomy-specific; it could be used in many fields of science. The API models both the content and the data flow and allows full pedigree navigation via *derived from* links.

CVO is a collection of 3-5 catalogs: pixel data, processing instances, sources, objects, and other products of processing. Archives publish abstracted pixel data to the pixel catalog. Software agents find new pixel data and create processing catalog entries for them. The processing catalog entries are eventually executed to produce processing products and source catalog entries. Finally, source catalog entries are cross-identified to produce object catalog entries. Pixel content from a variety of CADC archives and external catalogs will be published to the CVO pixel catalog; this and the subsequent processing will make the CVO a rich playground for astronomers.

Acknowledgments. The entire CADC group has provided invaluable assistance in developing the concepts and ideas that have resulted in the CVO.

References

Edwards, K. 1999, Core Jini, Sun Microsystems Press

Yee, H. K. C., Ellingson, E., & Carlberg, R. G. 1996, ApJS, 102, 269