Query driven visualization of large scale multi-dimensional astronomical catalogs

Buddelmeijer, Hugo

Chapter 5

Query Driven Visualization by Abstraction of Data Pulling

Abstract: Interactive visualization of astronomical catalogs requires novel techniques, because they are starting to contain billions of sources with hundreds of parameters. The creation as well as the extraction of the catalogs can be handled by data pulling mechanisms, which prevent unnecessary processing by having users request certain end products.

In this work we present query driven visualization as a logical continuation of data pulling. Scientists can request catalogs in a declarative way and set process parameters from within the visualization.

The coupling of processing and visualization is abstracted by designing new messages for the Simple Application Messaging Protocol. Support for these messages are implemented in the Astro-WISE information system and in a set of demonstrational applications.

5.1 Introduction

Large astronomical surveys require novel ways for handling the data they produce. For example, the Euclid mission will detect billions of galaxies (Laureijs, 2009) for which hundreds of parameters will be quantified, leading to terabytes of data to explore.

Data pulling can be used to achieve the scalability to create catalogs (Chapter 2): The processing steps necessary to create a catalog are determined by specifying the required target. The information system will determine how existing catalogs can be used to fulfill the request and will initiate the creation of new catalogs only when no suitable ones exist. This maximizes reusability of the catalogs and minimizes unnecessary calculations. This requires full *data lineage*, which means that catalogs are stored with all the information required to reprocess them.

Query driven visualization is a methodology to explore large data sets by limiting processing and visualization to the subsets of the data deemed "interesting" as defined by the user (Stockinger et al., 2006). Related work focuses on limiting the processing required for the visualization itself (Stockinger et al., 2006), the fast identification and retrieval of data (Stockinger et al., 2005; Bethel et al., 2006) or on the representation (Smith et al., 2006).

In this work, we see query driven visualization as the logical continuation of data pulling in an information system with full data lineage. The main contributions of our work follow from applying this novel viewpoint to source catalogs: (1) We limit the processing required to create the requested catalog itself, instead of the processing required for the visualization. (2) We permit requests in a more declarative form than direct database queries would allow. (3) We allow the user to inspect and influence the processing from within the visualization by exporting the data lineage. (4) We achieve a high level of abstraction that allows close interoperation between software.

We demonstrate our techniques with our Astro-WISE implementation and extension of the Simple Application Messaging Protocol.

5.1.1 Data Pulling and Declarative Querying

Data pulling is an excellent opportunity for query driven visualization. The autonomous discovery and creation of catalogs permits requests that are very declarative. A scientist can request parameters of sources without having knowledge of whether these parameters have already been calculated or not. An example program to pull catalogs is given with the 'Simple Puller' of section 5.3.3.

Compare this for example with an SQL-based system: To formulate an SQL query it is required to know which tables contain the required parameters, how to identify the relevant rows and columns, and often how to join tables. This becomes a non-trivial problem when catalogs are shared between multiple users and the number of catalogs and their sizes grow. At a certain stage it becomes too time consuming and error-prone to find required data by hand, especially when it is unknown whether it exists at all.

5.1.2 Full Data Lineage and Exploration

An information system with data pulling often has *persistent objects* with *full data lineage*: Each data set is represented as an object—in computer science terminology—that persists between sessions and users. In Astro-WISE these objects are called *process targets*. The process targets contain all the information required to create the data it represents from other process targets, this is called *backward chaining*.

The data lineage can be utilized in query driven visualization by having the visualization software request it. This allows the visualization software to show this information, either directly or processed in the visualization. An example of the former is given with the 'Tree Viewer' in section 5.3.3.

Furthermore, exporting the data lineage makes it possible for scientists to influence the processing by permitting the visualization software to change processing parameters. An example of this is given with the 'Object Viewer' in section 5.3.3.

5.1.3 Abstraction and Interoperation through SAMP

Data pulling mechanisms are well suited for abstraction on different levels: Firstly, pulling data does not require detailed knowledge of every processing step; secondly, these processing details themselves can be abstracted, because of the standardized data lineage.

Such an abstraction allows query driven visualization with interactive data processing to be performed between any visualization package and information system. The thoroughness of the interoperation will depend on the level of abstraction supported by both applications. We extended the Simple Application Messaging Protocol to facilitate such interoperation (section 5.2).

5.1.4 Astro-WISE

Query driven visualization requires an information system responsible for creation, storage and delivery of the data. We choose to use Astro-WISE for this, although any information system with data pulling and persistent objects would be suitable, because of the abstraction through SAMP. In Appendix 5.B we describe the details of our Astro-WISE SAMP implementation.

5.2 Interoperability through SAMP

The *Simple Application Messaging Protocol* (SAMP) is an International Virtual Observatory Alliance (IVOA) standard for interoperation between astronomical software. The idea behind it is akin to the UNIX-philosophy that tools should do one thing, should do that thing well and communicate with other programs for things they cannot do.

5.2.1 Simple Application Messaging Protocol

We give a short description of SAMP before discussing our additions. For details we refer to Appendix 5.A and to the official documentation[1]. The protocol uses a client-server model based on application defined messages. Clients register with the SAMP HUB and subscribe to certain types of messages. Clients can then send messages to individual clients or to any client that has registered for that kind of message. The receiving application should subsequently perform the action it has associated with the message. Lastly, the HUB will relay a response back to the sender if necessary.

The expected action that corresponds to a message is determined by the type of the message. Both default administrative messages and widely accepted application defined messages can be found on the SAMP wiki[2]. In the rest of this section we first describe relevant existing messages and subsequently introduce our proposed messages. We list the type of the messages and a description of the intended action of the receiver. Details, such as the parameters of all the messages, are given section 5.A.3.

5.2.2 Existing Catalog Related Messages

Several existing catalog related messages can be used in conjunction with our new messages:

- `table.load.votable`: Load a table in VOTable format.
- `table.load.fits`: Load a table in FITS format.
- `table.highlight.row`: Highlight a single row of an identified table.
- `table.select.rowList`: Select a list of rows of an identified table.

Exactly what 'highlighting' or 'selecting' means is left to the receiving application. Tables have three identifiers in SAMP: a table-id that is unique within the SAMP session, a URI where the catalog can be found and a human readable name. These identifiers are set with one of the `load` messages and used as a reference in the other messages. Rows are identified by their position in the table using zero-based indexing. Note that these messages can refer to any tabulated data set. In this chapter we limit ourselves to source catalogs only.

5.2.3 Data Pulling Messages

We designed new SAMP messages to create a system independent way to perform pulling of catalog data. The messages should be sent from visualization software to the information system handling the data:

[1] `http://www.ivoa.net/Documents/latest/SAMP.html`
[2] `http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/SampMTypes`

- `catalog.pull`: Pull a catalog and send it over SAMP using one of the `table.load.*` messages. The result could be an existing catalog or a new catalog created by the pulling mechanisms. This message requires the following parameters, detailed below: an identifier of a catalog to select the sources from, a selection criterion and a list of requested attributes of the sources.
- `catalog.derive`: Perform the same action as `catalog.pull`, but without sending the catalog data over SAMP.

Support for the `.pull` message is the minimum required to request catalog data from an information system. The `.derive` message is useful when it is necessary to inspect or modify the derivation of the catalog—using the messages in section 5.2.4—before visualization. These two messages require three parameters which we should elaborate on (see also section 5.A.4):

- `catalog-id`: An identifier of the base catalog to select the sources from. This can be a unique identifier of an existing catalog, but could also be a reference to a catalog that not yet exists, e.g. a photometric catalog for an observation that has not yet been reduced. It is also possible to designate identifiers for special catalogs, e.g. to denote the latest version of a catalog of an ongoing survey. It is left to the information system to inform scientists how to refer a specific catalog.
- `query`: A selection criterion to specify which sources of the original catalog are requested. This should be a logical expression referencing the `attributes` below. The exact specification of this expression is left to the information system. A logical choice would be the syntax of an `ADQL WHERE` clause (without the '`WHERE`' itself).
- `attributes`: A list of requested attributes (parameters) of the sources. It is not required that the catalog corresponding to the `catalog-id` contains these attributes. The data pulling mechanisms of the information system should try to find the requested attributes in related catalogs and should create new data sets if necessary. How an attribute should be specified, is left to the information system.

5.2.4 Object Messages

Several SAMP message types are defined for interaction with an information system with persistent objects. These messages allow the visualization software to gain information about the objects and inspect or even influence its processing. The persistent object related message are:

- `object.highlight`: Highlight an object.
- `object.info`: Return information about an object, see below.
- `object.change`: Change the value of a property of an object such as a process parameter or a dependency.

- `object.action`: Perform an action related to an object or property. Possible actions are retrieved using the `object.info` message.

The `object.highlight` message can be sent to any application, the others are supposed to be sent to the information system only.

SAMP `object.info` Data Structure

A specific SAMP map is defined as a return value for the `object.info` message, containing information about the object and its properties (see Appendix 5.A for details). For the object itself it includes information about what properties it has, its processing status and whether the object can be modified.

The properties of an object include process parameters and references to the progenitors of the object. The returned information about a property include its name, current value and optionally other values it can be set to. Furthermore the information system can define actions that can be performed on the object or its properties.

5.3 SAMP HUB and Clients

The new SAMP messages are implemented in the Astro-WISE information system and demonstrated by a set of proof-of-concept applications. We first describe relevant existing SAMP applications, subsequently the Astro-WISE SAMP connectivity and end with the applications to demonstrate the new messages. Figure 5.1 shows a diagram of the interoperability between Astro-WISE and several SAMP applications.

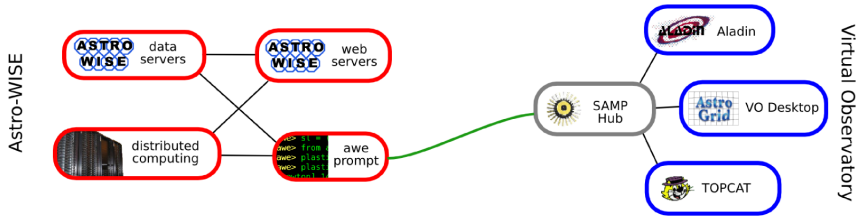

Figure 5.1: The connectivity between Astro-WISE and SAMP. On the left the Astro-WISE system in red and on the right SAMP enabled applications in blue. The SAMP HUB in the center is gray.

5.3.1 Existing SAMP applications

We list existing SAMP applications that are relevant to catalog data.

SAMP Application: SAMP HUB

The *HUB* is the center of SAMP to which the other applications connect. The HUB can be a standalone program or can be integrated in one of the clients, e.g. Aladin and Topcat include one.

SAMP Application: Topcat

Topcat[3] is a table viewer/manipulator written in Java. The visualization power of Topcat lies in its interactivity. Selections performed in one window propagate to other windows and by the use of SAMP messages to other applications.

SAMP Application: Aladin

Aladin is an FITS image viewer that can load images up to 50K by 50K pixels and overlay information from source catalogs. It can view local files and or connect directly to several repositories, by the Virtual Observatory and otherwise.

5.3.2 Astro-WISE and SAMP

Astro-WISE has SAMP connectivity in the interactive Python prompt and on the webservices.

SAMP Application: `awe`-prompt

We developed a Python module for SAMP connectivity in the Astro-WISE `awe`-prompt and other Python applications. This module is described in detail in Appendix 5.B. All messages from section 5.2 are supported.

SAMP Application: Astro-WISE DBViewer

With the Astro-WISE *DBViewer* one can view all content of the database and can send query results over SAMP. The DBViewer is beyond the scope of this thesis.

5.3.3 Query Driven Visualization Prototype

A set of proof-of-concept applications has been developed to demonstrate different ways in which SAMP clients can use the query driven visualization messages. They interact with the Astro-WISE `awe`-prompt through SAMP only and have little knowledge about Astro-WISE, if at all.

SAMP Application: Simple Puller

The *Simple Puller* (figure 5.3) represents the most basic way an application can pull catalog data. Its sole capability is to send a `catalog.pull` message, it cannot receive messages. It requires a minimum amount of input:

- An identifier of the base catalog from which the sources are selected.
- A list of required attributes.
- A query to select the sources.

[3] `http://www.starlink.ac.uk/topcat/`

The only knowledge the user needs to have about the information system is how these parameters should be specified. This service could be built into existing visualization tools quickly. The demo application uses a web-based interface with the server running locally and relies on other SAMP applications for the actual visualization, e.g. Topcat (figure 5.5).

SAMP Application: Tree Viewer

The *Tree Viewer* (figure 5.2) shows how a SAMP application can use the `object.info` message to give the user more information about the data lineage and derivation of a particular dataset.

This demo application recognizes several of the classes used in Astro-WISE and is able to interpret some of their properties. The application allows exploration of the dependency graph of a pulled catalog by presenting it as a dot[4] graph. Clicking on a node sends the `object.highlight` message, allowing interaction with the `awe`-prompt.

SAMP Application: Object Viewer

The *Object Viewer* demonstrates how an application can use the object related messages (`object.info`, `object.change` and `object.action`) to influence the properties of process targets and other objects. It has knowledge about the Astro-WISE Source Collection classes and allows many of the actions that can be performed in the `awe`-prompt to be done through the web-based GUI.

5.A Appendix: SAMP Protocol and Messages

We give the details about SAMP that are necessary to describe our proposed messages and to present our Astro-WISE implementation (Appendix 5.B).

5.A.1 SAMP Protocol

SAMP is in principle language-agnostic and is based on abstract interfaces. That is, it specifies which functions the HUB and the clients must have in order to send and receive messages, but not the exact protocol that the applications use to call those functions. The rules which describe how SAMP functions are mapped to the internally used protocol is described in a SAMP *Profile*. One standard profile based on XML-RPC is described in the official documentation, and this is what is used in Astro-WISE and in the prototype applications.

5.A.2 SAMP Data Types

Only three data types are supported in SAMP, because it is language- and even communication-protocol-agnostic:

[4] `http://www.graphviz.org/`

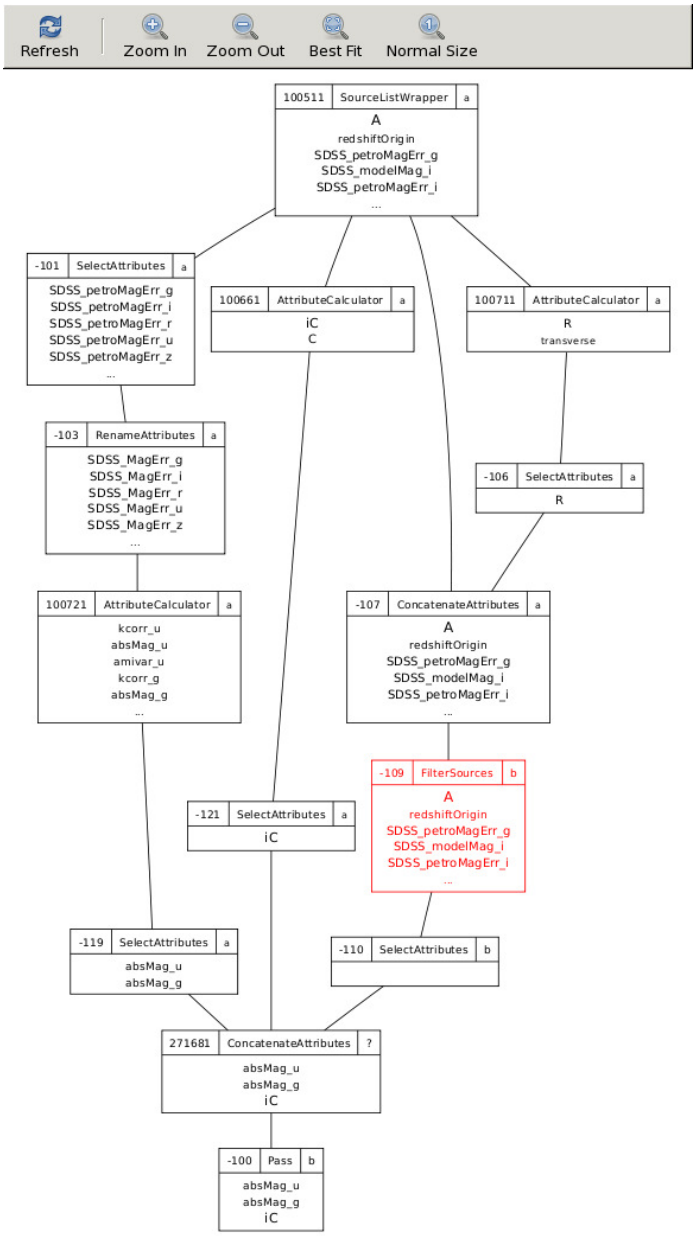

Figure 5.2: SAMP application for exploring dependency graphs of catalog objects in AstroWISE. Every node shows the catalog identifier on the top left, the class of the catalog in the top center and an identifier for the set of sources on the top right. The attributes of the sources are shown in the rest of the box.

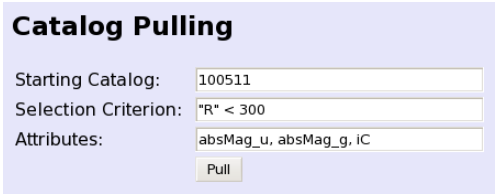

Figure 5.3: The Simple Puller application for pulling catalogs over SAMP. It can pull data from any information system that accepts the `catalog.pull` message.

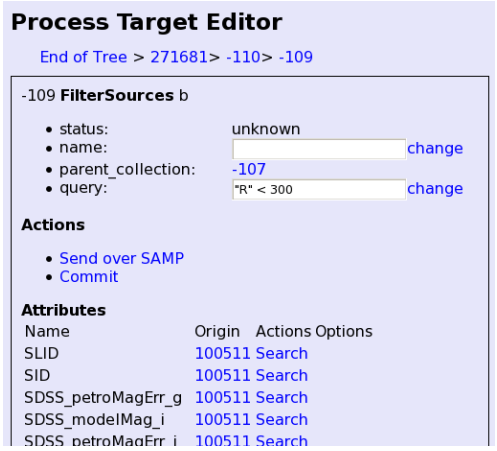

Figure 5.4: SAMP application to view and modify details of individual catalogs or other objects. The highlighted catalog from figure 5.2 is shown.

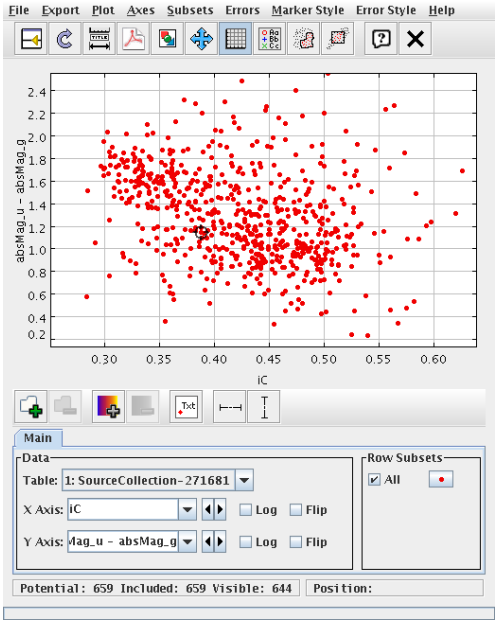

Figure 5.5: A Topcat scatter plot showing a color-concentration diagram of the catalog pulled figure 5.3. A bimodality between red, concentrated, galaxies and blue, extended galaxies can be seen.

- `string`: A scalar value consisting of a sequence of ASCII-characters.
- `list`: An ordered array of data items.
- `map`: An unordered associative array with a string as key.

Other scalar types have to be mapped to strings, and there is a specification to represent integers, floats and booleans as strings. These data types can be nested to any level: e.g., it is possible to have a map with lists as values.

5.A.3 SAMP Messages

SAMP applications communicate through messages of specific types. Message types that start with `samp.` are administrative messages defined by the protocol, the others are defined by application authors. Clients are supposed to give a general reply with success or failure of a requested operation, even if no response is required. We give a description of the message types, the required arguments, the expected action and the response, if required.

SAMP Application Messages

The application defined messages relevant for catalog data or otherwise implemented in Astro-WISE are:

- `table.load.votable`: Load a table in VOTable format. Arguments:
  - `url` (string): URL of the file to load.
  - `table-id` (string): Identifier, used to refer to the loaded table in subsequent messages.
  - `name` (string): Name, used to label the loaded table in the application GUI.
- `table.load.fits`: Load a table in FITS format. The parameters are identical to `table.load.votable`.
- `table.highlight.row`: Highlight a single row of an identified table by row index. The table to operate on is identified by one or both of the *table-id* or *url* arguments. At least one of these must be supplied; if both are given they should refer to the same thing. Exactly what highlighting means, is left to the receiving application. Arguments:
  - `table-id` (string): Identifier associated with the table by a previous message.
  - `url` (string): URL of the table.
  - `row` (int): Row index (zero-based) of the row to highlight.
- `table.select.rowList`: Select a list of rows of an identified table by row index. Exactly what selection means, is left to the receiving application. Arguments:

  - `table-id` (string): Identifier associated with the table by a previous message.
  - `url` (string): URL of the table.
  - `row-list` (list of int): list of row indices (zero-based) defining which table rows are to form the selection.

- `image.load.fits`: Load a 2-dimensional FITS image. Arguments:
  - `url` (string): URL of the FITS image to load.
  - `image-id` (string): Identifier which may be used to refer to the loaded image in subsequent messages.
  - `name` (string): Name which may be used to label the loaded image in the application GUI.

- `coord.pointAt.sky`: Direct attention (e.g. by moving a cursor or shifting the field of view) to a given point on the celestial sphere. Arguments:
  - `ra` (float): Right ascension in degrees.
  - `dec` (float): Declination in degrees.

5.A.4 Query Driven Visualization

We designed new SAMP messages and data structures to enable query driven visualization through data pulling mechanisms. The `object.*` messages assume that the information system uses an object oriented model for science products such as catalogs (section 5.1).

QDV SAMP: Message Types

The proposed messages are:

- `catalog.derive`: Create a catalog through data pulling. Arguments:
  - `catalog-id` (string): Identifier of the catalog to select the sources from.
  - `query` (string): Selection criterion for the sources.
  - `attributes` (list of strings): Names of the attributes.

- `catalog.pull`: Perform the same action as `catalog.derive` and send the data over SAMP. Arguments:
  - `catalog-id` (string): Identifier of the catalog to select the sources from.
  - `query` (string): Selection criterion for the sources.
  - `attributes` (list of strings): Names of the attributes.

- `object.highlight`: Highlight an object. Arguments:

  - `class` (string): Class of the object.
  - `object-id` (string): Identifier of the object.

- `object.info`: Returns a SAMP map with information about an object as described below. Arguments:
  - `class` (string): Class of the object.
  - `object-id` (string): Identifier of the object.

- `object.change`: Change a property of an object. Arguments:
  - `class` (string): Class of the object.
  - `object-id` (string): Identifier of the object.
  - `property-id` (string): Identifier of a property of the object.
  - `value` (string): New value of the property.

- `object.action`: Perform an action related to a an object. Arguments:
  - `class` (string): Class of the object.
  - `object-id` (string): Identifier of the object.
  - `property-id` (string): Identifier of a property of the object.
  - `action-id` (string): Identifier of the action.

QDV SAMP: Data Format

SAMP data structures are defined to send information about objects between applications. The structures are designed to be generic enough that they could be used for any information system.

Information about an object itself, e.g. the response to an `object.info` message, is communicated through a map with the following keys:

- `class` (string): The class of the object. A client that has knowledge about the used classes could handle known classes in a special way.
- `id` (string): Identifier this object, unique in combination with the class.
- `status` (string): Indication the processing status of this object (see below).
- `properties` (list of maps): Properties of this object (see below).
- `actions` (list of maps): Actions that can be performed on this object (see below).
- `readonly` (boolean): Flag to indicate that the object cannot be modified.

Properties of an object, for example process parameters, are described with a map with the following keys:

- `name` (string): Name of the property, as used by the object.
- `class` (string): The class that the value of the property should have, or a primitive such as 'int'.
- `description` (string): A human readable description of the property.
- `value` (string): The used value for the property. This is the `id` of the object if the property represents another object.
- `options` (list of maps): Possible values for the property, if applicable (see below).
- `actions` (list of maps): Actions that can be performed on the property (see below).
- `readonly` (boolean): Flag to indicate that the property cannot be modified.

An action that can be performed on an object or property is defined by a map with the following keys:

- `id` (string): A unique identifier for this action.
- `name` (string): A human presentable name for this action.

QDV Samp: Object Status

The status value of an object refers to the processing status of the object. It can have the following values:

- `ok`: The object has been processed, or can be processed while retrieving the result.
- `automatic`: The object has to be processed before the can be retrieved. This can be done without user interaction.
- `new`: This is a non-persistent object, which can be processed without user interaction.
- `depends`: This is a new object, which can be processed only after human intervention. For example to set a process parameter that has no proper default.
- `not`: As it is, this object cannot be processed, e.g. because a dependency cannot be fulfilled. The scientist might be able to solve the problem, but whether this is the case is not clear to the information system.
- `unknown`: The status is unknown.

QDV Actions

The actions value of the dictionaries refer to actions that can be performed through the `object.action` message. Examples of possible actions are given in section 5.B.8.

5.B Appendix: SAMP in the `awe`-prompt

We have developed a Python module for Astro-WISE to connect to SAMP from the `awe`-prompt. This allows an astronomer to combine the large scale data handling from Astro-WISE with the visualizations from other SAMP applications.

This appendix is most interesting for readers already familiar with Astro-WISE. All relevant terms are introduced briefly for readers new to Astro-WISE. The functionality that is not query driven visualization specific, is included as well for completeness.

5.B.1 Data in Astro-WISE

We give an overview of how catalogs of sources are handled within Astro-WISE before discussing the SAMP connectivity. In particular we introduce the classes that we refer to later in the section.

Astro-WISE: Catalog Data

Source catalogs that can be used for data pulling are called *Source Collections*. There are different Source Collection classes, depending on the operation used to create the catalog. For example, an *Attribute Calculator* Source Collection is used to calculate new attributes (parameters) of sources.

Other Astro-WISE classes that relate to catalogs are: the *SourceList* for photometric parameters derived from images, the non-persistent *TableConverter* to manipulate tabular data in Python and the *PhotSrcCatalog* used for photometric calibration.

A SourceList has a `SLID` as identifier, and sources in a SourceList are labeled with a `SID`. The `SLID-SID` combination uniquely identifies a source. A Source Collection has a `SCID` as identifier and can contain sources from multiple SourceLists.

The actual data pulling is performed with the non-persistent *Source Collection Tree* class. Instances of this class are used to handle the query driven visualization SAMP messages.

Astro-WISE: Image Data

Image data in Astro-WISE is handled by various *Frame* classes. They are only relevant for this chapter because they can be broadcasted over SAMP.

5.B.2 SAMP Classes

The Astro-WISE SAMP client consists of three classes.

SAMP Class: ThreadXMLRPC

The *ThreadXMLRPC* class is a multithreaded XML-RPC server which the HUB connects to in order to send messages. This class does not contain any SAMP or Astro-WISE specific code.

SAMP Class: SampProxy

An instance of the *SampProxy* class is a basic SAMP client. This class contains all SAMP code that is not Astro-WISE specific, and can therefore be used by other Python applications as well.

SAMP Class: Samp

The *Samp* class is derived from SampProxy and contains all Astro-WISE specific code. The metadata that the class declares to the HUB —as stored in its `metadata` property— is:

```
author.email              buddel@astro.rug.nl
author.name               Hugo Buddelmeijer
home.page                 http://www.astro-wise.org
author.affiliation        Kapteyn Astronomical Institute, Groningen
samp.name                 Astro-WISE
samp.description.html     <p>Astro-WISE</p>
samp.documentation.url    http://www.astro-wise.org
samp.icon.url
    http://www.astro-wise.org/pics/logo-samp-astrowise.png
samp.description.text     Astro-WISE.
```

5.B.3 Communication with HUB

Connecting and registering with the HUB is done automatically when a Samp object is instantiated. This can be suppressed by setting the `register` parameter of the initializer to False.

Functions

- `getSettingsHub()`: Returns the settings of the HUB, as read from the file `~/.samp`.
- `connect()`: Connects to the XML-RPC server provided by the HUB. Returns the server connection.
- `register()`: Registers the client with the HUB and declares message subscriptions.
- `getClients()`: Returns the metadata of the other registered clients.

5.B.4 Sending Data

Sending data is most easily achieved with one of the `broadcast*` functions. The `broadcast*` functions send data to all clients (that can handle that data type).

Functions

- `broadcast()`: This function accepts as parameter an object that is either a Source Collection, a SourceList, a Frame, a PhotSrcCatalog or a TableConverter and dispatches it to the relevant function below.
- `broadcastSourceCollection()`: Sends the contents of a Source Collection.
- `broadcastSourceList()`: Sends the contents of a SourceList.
- `broadcastCatalog()`: Sends the contents of a PhotSrcCatalog
- `broadcastTableConverter()`: Sends the contents of a TableConverter.
- `broadcastVOTable()`: Broadcasts a VOTable using `table.load.votable`.
- `broadcastFrame()`: Sends a Frame as a FITS file.
- `broadcastImage()`: Broadcasts a FITS image using `image.load.fits`.

These function require as argument an object of the corresponding class. Optional arguments for the table related messages are `filename`, `tableid` and `name`, which are respectively the filename where the (intermediate) VOTable or FITS file is saved, an identifier for the table and a name to display in the other programs. All table and image related functions respectively use `broadcastVOTable()` and `broadcastImage()`.

5.B.5 Sending Interaction Messages

Interactivity between different SAMP programs for tabular data can be achieved with the following functions. Their first argument is a 'table key' (see section 5.B.10), the second the respective source identifiers.

Functions

- `highlight()`: Highlights a row by sending a `table.highlight.row` message, identified by a SAMP row number.
- `select()`: Select a group of rows by sending a `table.select.rowList` message, identified by a list of SAMP row numbers.
- `highlightSource()`: Highlights a row by sending a `table.highlight.row` message, identified by a `SLID` and `SID`.
- `selectSources()`: Select a group of rows by sending a `table.select.rowList` message, identified by a list of `SLID`s and `SID`s.

5.B.6 Sending General Messages

The SAMP protocol describes four methods to send messages. These are mapped to similar function names which can be used to send messages manually. This is useful to send messages that are not supported by the class directly. Since the message types are agreed upon between individual applications, any application developer can create his or her own messages. All the above functions use the `callAll()` method.

Functions

- `notify()`: Sends the message to one specific receiver, it is not possible to reply.
- `call()`: Sends the message to one specific receiver. The receiver is expected to reply, but the program continues.
- `callAll()`: Sends the message to all clients that have registered for this message type. The receivers are expected to reply, but the program continues.
- `callAndWait()`: Sends the message to one specific receiver. This function requires a `timeout` parameter. The program halts and waits at most `timeout` seconds on the recipient to reply.

The functions with a specific receiver require a `receiverid` parameter with the SAMP id or name of the intended receiver. These can be discovered with the `getClients` function. For Topcat and Aladin it is also possible to use the shortcuts `topcat` and `aladin` as `receiverid`.

5.B.7 Receiving Table Related Messages

The SAMP client accepts the `table.load.votable`, `table.highlight.row` and `table.select.rowList` messages. Received tables are cached (section 5.B.10) and table interaction information can be retrieved with the following functions.

Functions

- `highlighted()`: Requires a table identifier as argument and returns the latest highlighted row from that table as an integer row number.
- `selected()`: Requires a table identifier as argument and returns the latest selected rows from that table as a list of integer row numbers.
- `highlightedSource()`: Identical to `highlighted()`, but converts the returned SAMP row identifier to a `SLID-SID` combination. This requires the table to have columns with these names.
- `selectedSources()`: Identical to `selected()`, but converts the returned SAMP row identifiers to `SLID-SID` combinations. This requires the table to have columns with these names.

5.B.8 Query Driven Visualization Data Structures

The SAMP data structure for the `object.info` message is created with the `get_export()` function of the referenced object. Only Source Collection (section 3.3) instances can currently be exported over SAMP. The properties that are exported by `get_export()` are:

- All persistent properties that do not relate to data caching. References to other Astro-WISE objects are exported as the unique identifier of the object.
- Process parameters of Attribute Calculators (section 3.6) are exported as if they are regular properties.
- The names of the attributes of the sources are exported as the `attribute|%i` properties, where the `%i` are consecutive integers. The `SCID`s of the Source Collections that the attributes originate from are exported as the `origin|%i` properties.

The actions that can be performed on Source Collections are:

- `commit`: Commits a transient Source Collection.
- `copy`: Creates a copy of a Source Collection.
- `make`: Process the Source Collection. The exact composition of sources and values of the attributes are determined.
- `send`: Broadcasts the catalog data corresponding to a Source Collection over SAMP.

Only the `attribute|%i` properties have an action:

- `search`: Search for Source Collections that could be used as a progenitor to provide the attribute. These will be listed in the `options` of the property.

5.B.9 Receiving Query Driven Visualization Messages

The query driven visualization messages from section 5.A.3 are supported, but only with respect to Source Collections. The parameters of the messages are interpreted as follows:

- `catalog-id`: The `SCID` of a Source Collection.
- `query`: An Oracle SQL `WHERE` clause, with attributes in double quotes.
- `attributes`: A list of attribute names.
- `class`: The name of an Astro-WISE class. Only Source Collection classes are supported at the moment.
- `object-id`: The `SCID` of a Source Collection.

- `property-id`: The name of a property, as defined by the `get_export()` function.
- `action-id`: The identifier of an action that can be performed on an object or property, as defined by the `get_export()` function.

Two properties are added to the SAMP class for the new messages: The `source-collectiontree` property of the SAMP class is a Source Collection Tree that is used for the catalog pulling messages and the `highlighted_sourcecollection` property is the latest highlighted Source Collection. The query driven visualization messages are handled as follows:

- `catalog.derive`: The `derive()` of the Source Collection Tree is called to derive a new Source Collection from the specified one.
- `catalog.pull`: Performs the same action as `catalog.derive` after which the resulting Source Collection is broadcasted.
- `object.highlight`: Sets the `highlighted_sourcecollection` member of the Samp instance.
- `object.info`: Returns information about a Source Collection by calling its `get_export()` function.
- `object.change`: Change a property of a Source Collection by calling the `change()` method of the object.
- `object.action`: Perform an action related to a Source Collection, either directly in the SAMP class or through the `action()` function of the object.

In the future the `object.` messages might be supported for other Astro-WISE objects as well.

5.B.10 Storing Information

The SAMP client stores information about every table that is sent or received through SAMP in its `tables` member. This is a dictionary with as primary key the SAMP `table-id`. For Python objects (Source Collection, SourceList, PhotSrc-Catalog, TableConverter), the object itself can be used as key as well. For Source Collections and SourceLists, the resp. `SCID` and `SLID` are keys as well. Values of the `tables` member are dictionaries with the following keys:

- `name`: Human readable name of the table.
- `url`: The url where the file is located.
- `tableid`: The `table-id` that is used within SAMP.
- `type`: The format of the table file (always 'votable').
- `selected`: A list of latest row numbers selected with a `table.select.rowList` message.

- `highlighted`: The latest row highlighted with a `table.highlight.row` message.
- `data`: A TableConverter instance holding the contents of the catalog.