	JMMC-TRE-2600-0001 Revision : 1.0 Date : 08/11/2004

JMMC

SEARCHCAL

SOFTWARE DESIGN DESCRIPTION

Author (sylvain.cetre@obs.ujf-grenoble.fr)
LAOG/JMMC

Author : Sylvain CETRE Institute : LAOG/JMMC	Signature : Date : 08/11/2005 
Approved by : Gérard Zins Institute : LAOG/JMMC	Signature : Date : 08/11/2005 
Released by : Gilles Duvert Institute : LAOG/JMMC	Signature : Date : 08/11/2005 

CHANGE RECORD

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
0.1	10/06/2005	S. Cètre	All
	Creation in Latex format		
0.2	26/09/2005	S. Cetre & S. Lafrasse	All
	Importation from Latex format, and corrections		
0.3	03/10/2005	S. Lafrasse	All
	Review and corrections		
0.4	13/10/2005	S. Cètre	All
	Update sclgui module description after MVC design pattern application		
0.5	14/10/2005	S. Lafrasse	All
	Last minute corrections		
0.6	08/11/2005	G. Zins	All
	Review		
1.0	08/11/2005	G. Zins	All
	Released		

TABLE OF CONTENTS

1 Introduction

1.1 Purpose

This document describes the design of the SearchCal software, developed by the JMMC Calibrators group.

1.2 Reference documents

[1] JRA4-PRO-2000-0001, Revision 1.0, JRA4 Programming Standards
[2] JMMC-MEM-2600-0004, Revision 2.0, SearchCal – Séquence d'interrogation des catalogues du CDS
[3] JMMC-MEM-2600-0005, Revision 3.0, SearchCal – Module de calcul de la zone de recherche pour les interrogations CDS
[4] JMMC-MEM-2600-0006, Revision 1.0, SearchCal - Calcul des magnitudes manquantes - Tables photométriques
[5] JMMC-MEM-2600-0007, Revision 2.0, SearchCal - Calcul des coordonnées galactiques
[6] JMMC-MEM-2600-0008, Revision 2.0, SearchCal - Calcul de la correction de l'absorption interstellaire
[7] JMMC-MEM-2600-0009, Revision 2.0, SearchCal - Calcul des diamètres photométriques
[8] JMMC-MEM-2600-0010, Revision 2.0, SearchCal - Calcul de la visibilité et de son erreur
[9] JRA4-MAN-2200-0001, In prep, Common Software – User Manual
[10] IVOA VOTable - http://www.ivoa.net/twiki/bin/view/IVOA/IvoaVOTable

1.3 Abbreviations and acronyms

ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
ASPRO	**A**stronomical **S**oftware to **P**repa**R**e **O**bservations
CDS	**C**entre de **D**onnées astronomiques de **S**trasbourg
CSV	**C**omma **S**eparated **V**alues
GUI	**G**raphical **U**ser **I**nterface
JMMC	**J**ean-**M**arie **M**ariotti **C**enter
MCS	**M**ariotti **C**ommon **S**oftware
MVC	**M**odel **V**iew **C**ontroller
OS	**O**perating **S**ystem
SDD	**S**oftware **D**esign **D**escription
STL	**S**tandard **T**emplate **L**ibrary
UCD	**U**nified **C**ontent **D**escriptor

UML	**U**nified **M**odeling **L**anguage
URL	**U**niform **R**esource **L**ocator
VO	**V**irtual **O**bservatory
XML	e**X**tensible **M**arkup **L**anguage

1.4 Document Conventions

The following typographic styles are used:

Bold: to highlight words in the text;

Italic: for parts of the text that have to be substituted with the real content before typing;

`Teletype: for names of programs, files and directories in the text;`

<...>: for parts of examples that have to be substituted with the real content before typing;

2 Overview

2.1 General SearchCal overview

The Figure 1 shows the environment in which SearchCal software is operating.

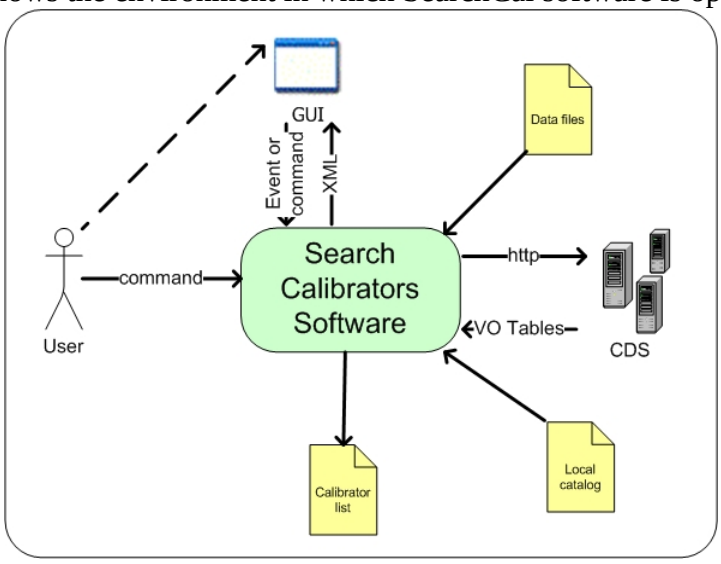

Figure 1 - SearchCal general software overview

The SearchCal Software is intended to be used for finding calibrator stars during preparation of interferometric observations. According to the observation constraints (science star position and magnitude, and observation band), the software provides to astronomers a list of potential calibrators by querying star catalogs. This list, ordered by increasing distance to the science object, is presented to the astronomer in a graphical interface, enabling him to visualize all morphologic and photometric properties of the potential calibrators. He can then filter this list according to position and/or properties before saving it in an ASCII file.

2.2 Software Environment

The SearchCal software is based on MCS. The common software should then be installed before the installation of SearchCal.

2.3 Module Description

The SearchCal software consists in the software modules listed in Table 1.

Module	Short description
vobs	**V**irtual **OBS**ervatory query & parsing library
alx	**A**stronomical **L**ibrary e**X**tension
sclsvr	**S**earch **C**a**L**ibrator **S**er**V**e**R**
sclgui	**S**earch **C**a**L**ibrator **G**raphic **U**ser **I**nterface

Table 1 - Inventory and short description of the SearchCal modules

2.4 Programming Standard

The standards prescribed by the “JRA4 - Programming Standards” (see [1]) are applied to the SearchCal software development.

3 Software Architecture

3.1 Architectural Description

As shown in Figure 2, the SearchCal Software is constituted by two processes:

- *SearchCal server* which is dedicated to the star catalog querying;
- *SearchCal GUI* which presents result to the astronomer and handles interaction with him.

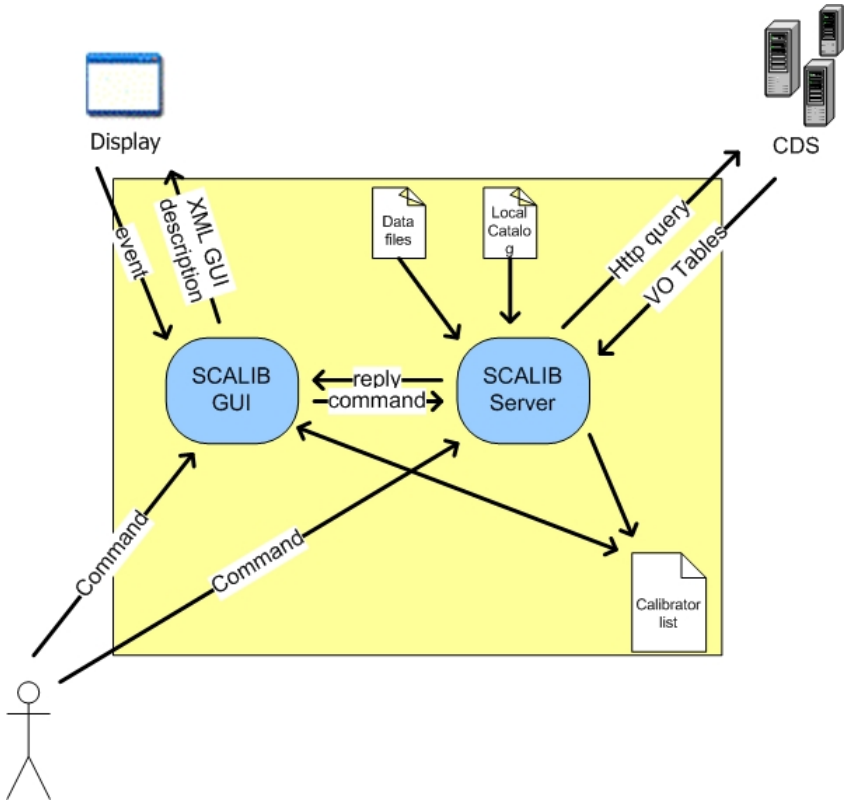

Figure 2 - Global architecture diagram of SearchCal

Note that it is possible to directly interact with SearchCal server using the MCS communication system, and then get result in text form or saved in text file in CSV format.

The list of commands accepter by the SearchCal server, register as `sclsvrServer`, is given in 5.2 section).

3.2 Program Structure, Class and Sequence Diagrams

3.2.1 SearchCal Server

The SearchCal server modules organisation is shown in Figure 3:

- `vobs`, provides methods to query star catalogs and parse result in order to build a list of stars.

- `alx` provides functions to compute specific astronomical parameters, such as angular diameter, in order to complete properties of the found stars.
- `sclsvr` contains the SearchCal server which is in charge to query star catalogs using `vobs`, complete star properties using `alx`, and then prepare the potential calibrators by selecting star having required photometric and morphologic properties.

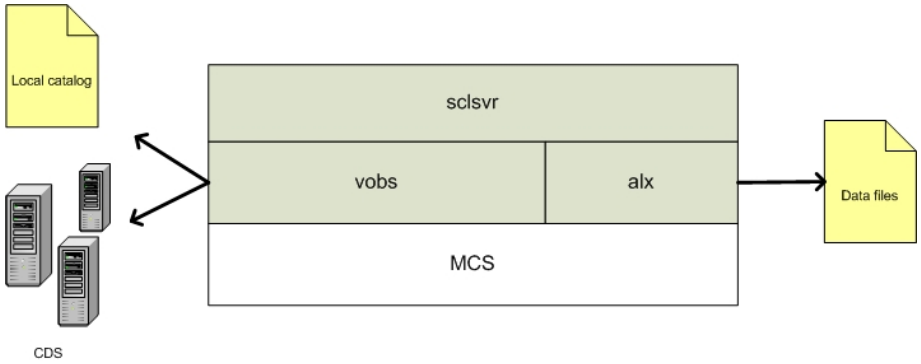

Figure 3 – SearchCal server module organization

3.2.1.1 vobs - Virtual Observatory Interface library

This module is used to query star catalogs (remote catalogs from CDS or local catalogs available as ASCII files) according to pre-defined scenario, parses resulting data to retrieve stars and their properties, and performs cross-identification to build star list.

3.2.1.1.1 General UML class diagram

The class diagram of `vobs` module is shown in Figure 4.

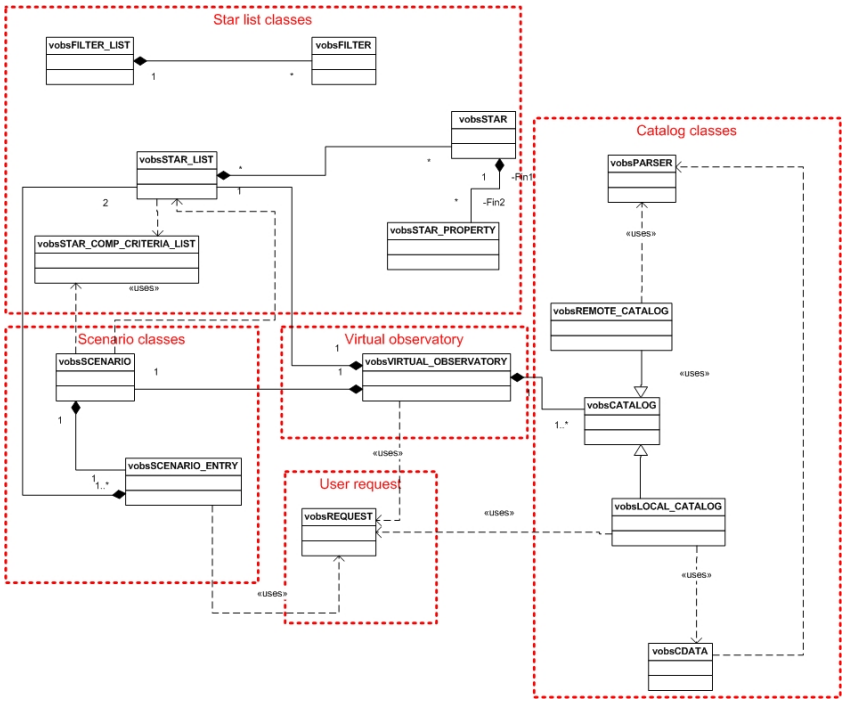

Figure 4 - vobs UML class diagram

The `vobs` classes can be grouped in 5 collections:

1. the ‘star list classes’ group which contains classes related to star and star list:
   - `vobsSTAR_PROPERTY` deals with star property defining its name, type and value.
   - `vobsSTAR` gathers all star properties.
   - `vobsSTAR_LIST` handles the list of stars.
   - `vobsSTAR_COMP_CRITERIA_LIST` specifies a list of star properties with the associated value range for star comparison.
   - `vobsFILTER` uses to filter star list.
   - `vobsFILTER_LIST` handles the list of filters.
2. the ‘catalog classes’ group which contains classes related to catalogs:
   - `vobsCATALOG` is base class to handle catalog.
   - `vobsLOCAL_CATALOG` is devoted to local catalog.
   - `vobsREMOTE_CATALOG` is devoted to remote catalog.
   - `vobsPARSER` parses VO-tables resulting from remote catalog query.
   - `vobsCDATA` parses CSV table coming from VO-tables or local catalog files.
3. the ‘user request class’ contains class related to the user request:
   - `vobsREQUEST` gives the list of user constraints.
4. the ‘scenario classes’ group which contains classes related to querying catalog scenario:
   - `vobsSCENARIO_ENTRY` is an entry of a scenario

- `vobsSCENARIO` is the querying scenario.

5. the ‘virtual observatory class’ contains the main class:
   - `vobsVIRTUAL_OBSERVATORY` is the main class orchestrating all other classes.

3.2.1.1.2 Star list classes

3.2.1.1.2.1 Star and star property

Seen by the software, a star only consists in a list of properties. The `vobsSTAR_PROPERTY` class is a simple class defining a star property, which is characterized by an identifier, a value, a type and the format used to print it out.

The `vobsSTAR` class provides methods to handle all the star properties. The star properties are stored in a STL map, where the key is the property identifier. A property identifier is unique and it is used to refer a property to define, set or get it. The definition of the star properties is done by the private `AddProperties()` method.

In addition to the classical methods to set or get a property, the `vobsSTAR` class provides the `IsSame()` method which is used to compare two stars and reports whether there are the same or not; according to the comparison criteria defined by the `vobsSTAR_COMP_CRITERIA` class. This method is very important because it is used to perform the cross-identification when looking for duplicated stars in a star list, or when merging two star lists. Indeed, a same star can be found in several catalogs with different coordinates.

3.2.1.1.2.2 Star list

The `vobsSTAR_LIST` class, based on STL list, implements classic linked list of `vobsSTAR` objects. In addition to the classical methods to manage the list, the class provides the `Merge()` method which allows merging of two lists with cross-identification using `vobsSTAR::IsSame()` method and `vobsSTAR_COMP_CRITERIA` class, as shown in Figure 5.

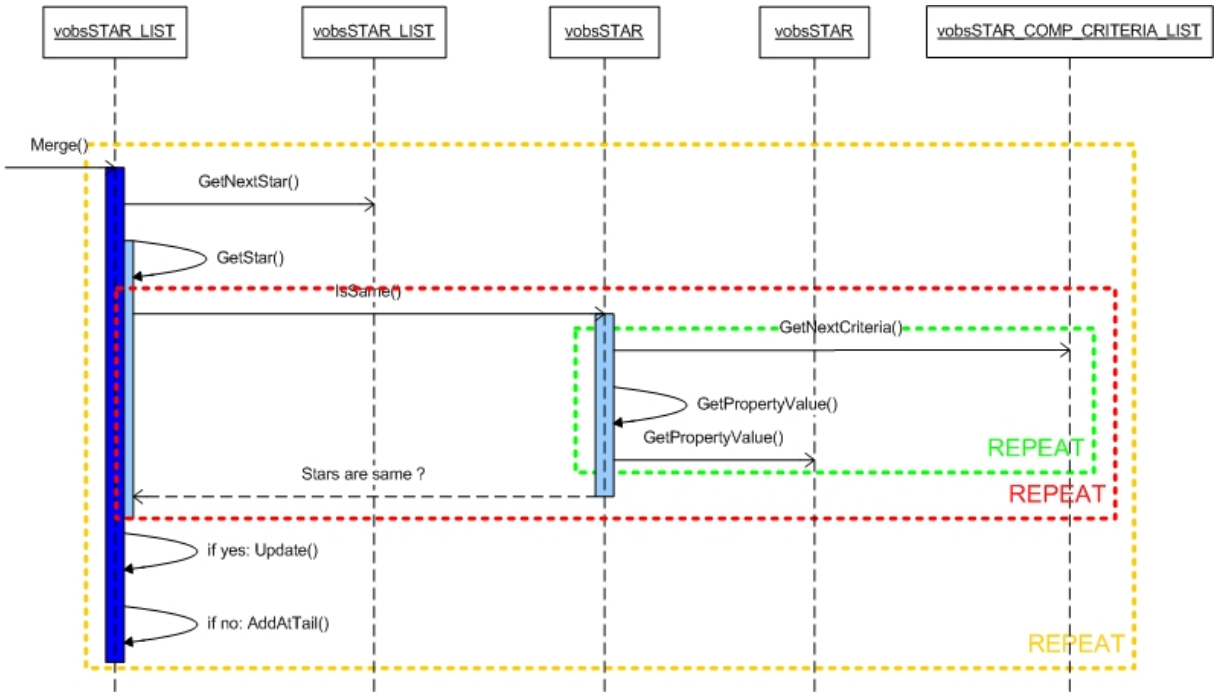

Figure 5 – List merging sequence

The `vobsSTAR_LIST` class also provides `Save()` and `Load()` methods which respectively save list in file and load list from file. These two methods use the `vobsCDATA` class (see 3.2.1.1.3.2) to build buffer in CSV format or to parse it.

3.2.1.1.2.3 Filter

A star list can be filtered using the `vobsXXX_FILTER` classes. All these classes inherit from the `vobsFILTER` class, as shown on Figure 6, which is an abstract class.

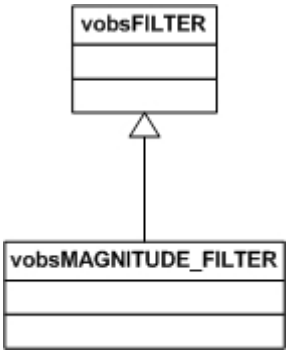

Figure 6 – Inheritance scheme of filter classes

The `vobsFILTER` class defines methods to enable/disable the filter, and the pure method `Apply()` which is used to apply filter on the star list; i.e. which removes all stars that do not match filtering criterions, as shown in Figure 7.

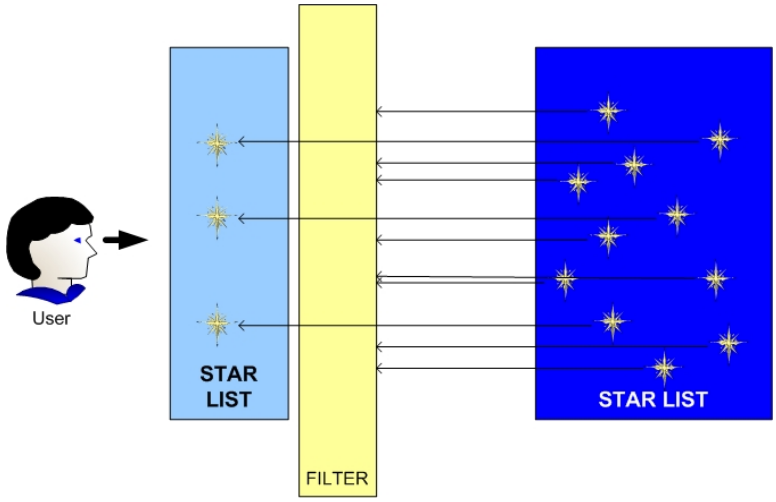

Figure 7 – Star list filtering

The `vobsFILTER_LIST` class provides method to manage a list of filter, and apply all these filters on a list of using the `Apply()` method, as shown in Figure 8.

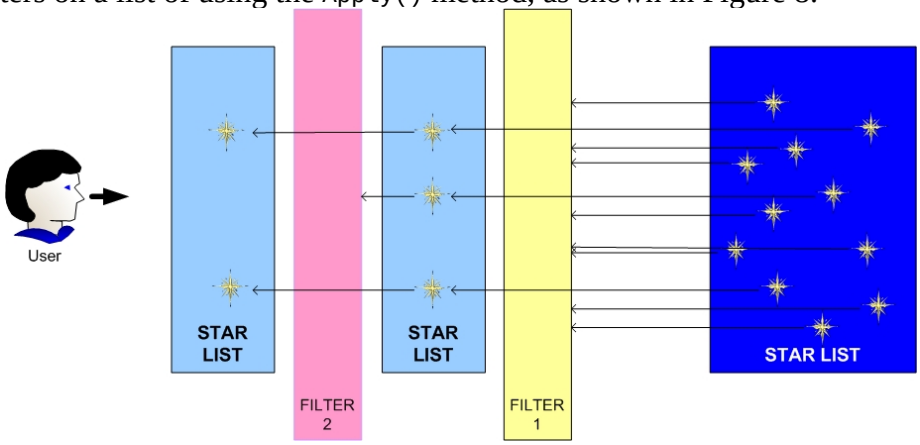

Figure 8 – Applying list of filters on star list

3.2.1.1.2.4 Comparison criteria

All merging, filtering and comparison operations on stars or star lists use the `vobsCOMP_CRITERIA_LIST` class. This class handles a list of criterions, where a criterion is a pair constituted by property Id and the associated range value. The criterions are stored in a STL map. The `vobsCOMP_CRITERIA_LIST` class provides method to add, delete and get a criterion.

3.2.1.1.3 Catalog classes

3.2.1.1.3.1 Local and remote catalogs

The information to be retrieved comes from remote catalogs, located on CDS server and accessible using http protocol, or from local catalogs located on local disk.

The `vobsCATALOG` class is an abstract class which defines all methods to interact with a

catalog (local or remote), while `vobsREMOTE_CATALOG` is dedicated to remote catalogs and `vobsLOCAL_CATALOG` to local ones, as shown in Figure 9.

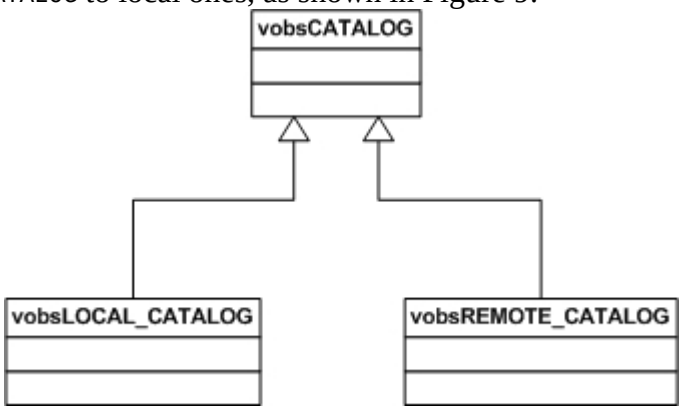

Figure 9 - Catalog class diagram

The `vobsCATALOG` class defines the `Search()` pure method, which is the method used to find stars in catalog. The `Search()` provides two methods to find stars in a catalog. The first one, called ***primary request***, looks for stars having magnitude in a given range and which are located in a given sky area. And the second one, called ***secondary request***, looks for stars belonging a given list. The primary request is used to build a list of stars, and the secondary request is used to complete the properties of the stars of this list.

In the `vobsREMOTE_CATALOG` class, the `Search()` method consists in:

- preparing first the URL giving the location of the catalog to query and the search criteria,
- getting VO-table from this URL and parsing it to retrieve stars matching to the request.

To prepare URL, the `vobsREMOTE_CATALOG` class provides two `PrepareQuery()` methods; one for primary request type and another one for secondary request. They use the `Write<parts>()` methods which are in charge to prepare a part of the URL. These methods are `WriteQueryURIPart()`, `WriteQueryConstantPart()`, `WriteQuerySpecificPart()`, `WriteReferenceStarPosition()` and `WriteQueryStarListPart()`.

As shown in Figure 10 for 2mass catalog, a dedicated class is created for each remote catalog used by software. This class inherits from the `vobsREMOTE_CATALOG` class, and should overload `WriteQuerySpecificPart()` method which prepares the catalog specific part of the URL. There are two `WriteQuerySpecificPart()` methods, one related to primary request, and the other one for the secondary request (see 3.2.1.4.1 section for more details).

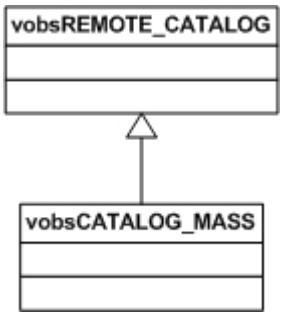

Figure 10 - 2mass catalog

In the `vobsLOCAL_CATALOG` class, the `Search()` method consists in scanning the stars of the catalog, and finding the ones matching the user request; i.e. the stars located in the given sky area and having magnitude in the specified range. The format of the local catalog file has been defined to be easily parsed using the `vobsCDATA` class (see 3.2.1.1.3.2 section) to create the star list.

Note that the local `vobsLOCAL_CATALOG` class is only foreseen to deal with primary request.

3.2.1.1.3.2 Parser

The stars to be retrieved either from VO-Table obtained by querying remote catalogs or from local catalog file, are stored in a table in CSV format; i.e. where each line corresponds to one star and where all properties of this star are separated by tabs. The UCD and name of each column (i.e. star property) is given in the VO-Table header in XML format, as shown in the example below:

```
<FIELD name="HD" ucd="ID_ALTERNATIVE" datatype="I" width="6">
    <DESCRIPTION>[1/358431]? HD/HDE number.</DESCRIPTION>
</FIELD>
```

and in the local catalog file in the first two lines of the file in CSV format as shown below:

```
<UCD list>
<Property name list>
<Property list of the star #1>
...
< Property list of the star #N>
```

The `vobsCDATA` class is used to parse the CSV table stored in an internal dynamic buffer of this class. This buffer is populated either with the `AppendLines()` method when data is extract from VO-Table, or with the `Load()` method when data is read from local catalog file. The property description (UCD and name corresponding to property), used during parsing, is set by `SetParamsDesc()` method. When buffer is populated and property description set, the `Extract()` method can be invoked to

extract star properties, instantiate star and add the new star to the star list. The property Id (see 3.2.1.1.2.1 section) is deduced by `GetPropertyId()` method, from the name and UCD associated to the property.
The `Extract()` method is a template method in order to be usable with classes deriving from `vobsSTAR` class. The `vobsCDATA` class also provides the `Store()` method, implemented in the same way than `Extract()` method, to save a list of stars in file.

The dynamic buffer is built from the VO-Table, retrieved from remote catalog, by the `Parse()` method of the `vobsPARSER` class, while this buffer directly loaded from local catalog files.

The `vobsPARSER` class uses the `libgdome` library to parse the VO-Table (XML document) for preparing dynamic buffer and pass it to `vobsCDATA` class.

The Figure 11 summarizes the sequence to retrieve star list from a remote star catalog.

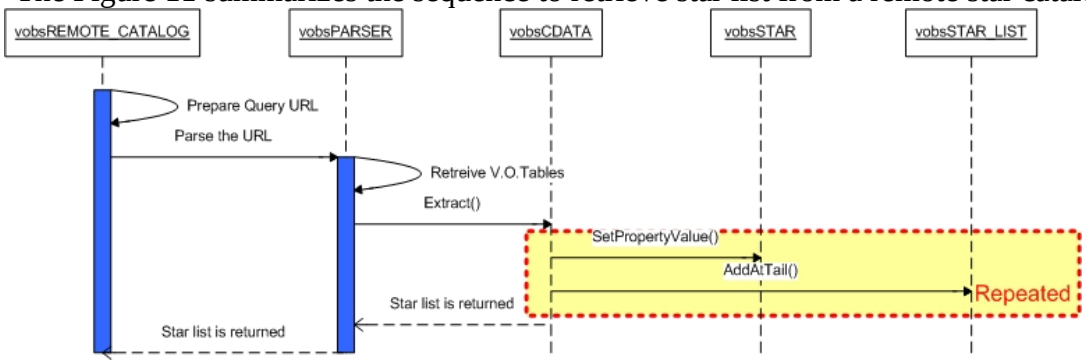

Figure 11 – Remote catalog querying sequence

3.2.1.1.4 User request

The `vobsREQUEST` class deals with parameters defining constraints, such magnitude range or sky area, for finding stars in catalogs.

3.2.1.1.5 Scenario classes

To build the star list, the software queries a list of catalogs, and merges the resulting star lists together. The querying catalog sequence is defined using `vobsSCENARIO` and `vobsSCENARIO_ENTRY` classes.

3.2.1.1.5.1 Scenario entry

The `vobsSCENARIO_ENTRY` class defines a step of the querying catalog sequence. A step is defined by:

- the catalog to query,
- the input star list,

- the output star list,
- the copying method,
- the cross-identification criterions.

The input star list is used to specify the request type: primary or secondary (see 3.2.1.1.3.1). When input star list is empty or is a NULL pointer, a primary type request is performed; otherwise it is a secondary type request.
After the star catalog has been queried, the resulting star list is copied into the output list. The copying method could be:

- `vobsCOPY` : the output list is cleared first, and the resulting star list is copied into.
- `vobsMERGE` : the resulting list is merged with the output list; the cross-identification is done using the criterions given by `vobsSTAR_COM_CRITERIA_LIST` instance. Stars which are not yet in the output list are added.
- `vobsUPDATE_ONLY` : ditto `vobsMERGE`, excepted that the stars which are not yet in the output list are ignored.

3.2.1.1.5.2 Sequential scenario

The `vobsSCENARIO` class consists in an ordered list of `vobsSCENARIO_ENTRY` instances which corresponds to the catalog querying sequence, as shown in Figure 12.

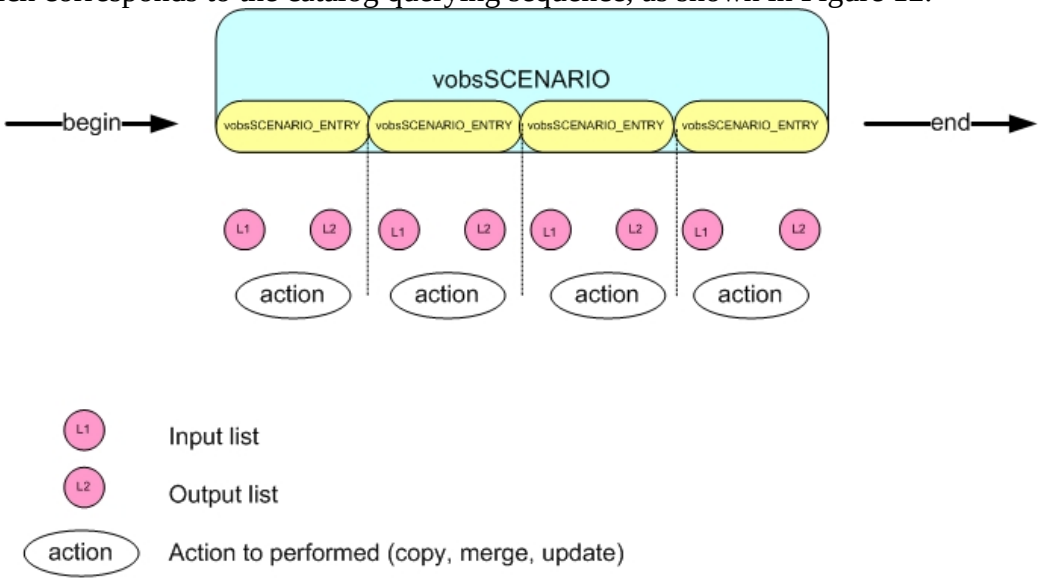

Figure 12 – Scenario

A step is added using the the `AddEntry()` method, and the `Execute()` method executes step by step the scenario. The output list of the last scenario entry is the resulting star list.

3.2.1.1.6 Virtual Observatory

The `vobsVIRTUAL_OBSERVATORY` class is the main class of `vobs` module. It provides the `Search()` method which is the entry point. This method is called with a `vobsREQUEST` instance. From the observing band, a querying scenario is loaded by the `LoadScenario()` method, before to be executed. The Figure 13 shows this sequence and interactions with other classes.

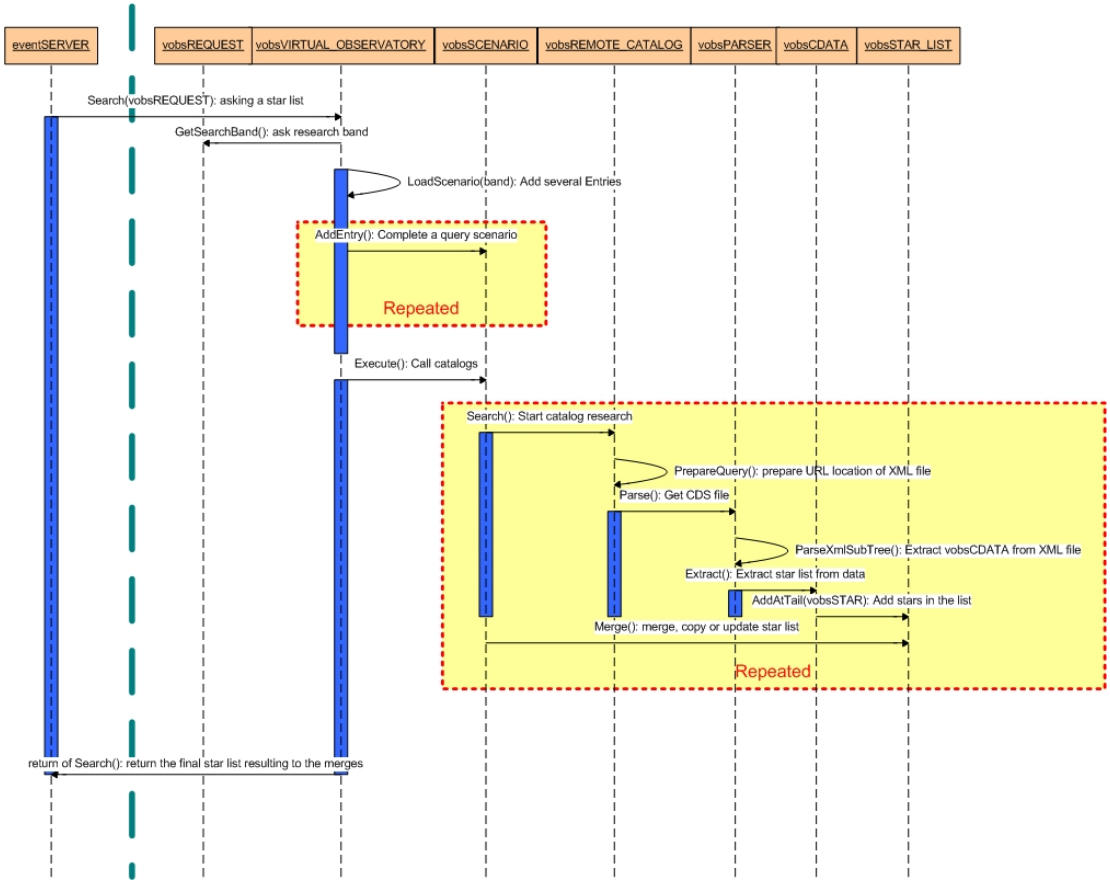

Figure 13 – Request sequence diagram

3.2.1.2 alx - Mathematical library

The `alx` module is a mathematical library that provides astronomical-specific functions. SearchCal uses `alx` to deal with:

- Galactic coordinates (see [5]);
- Correction of the interstellar extinction (see [6]);
- Missing magnitude computation (see [4]);
- Angular diameter (see [7]);
- Visibility (see [8]);
- Research area (see [3]).

3.2.1.3 sclsvr - SearchCal Server

3.2.1.3.1 General UML classes diagram

The class diagram of `sclsvr` module provides is shown in Figure 14.

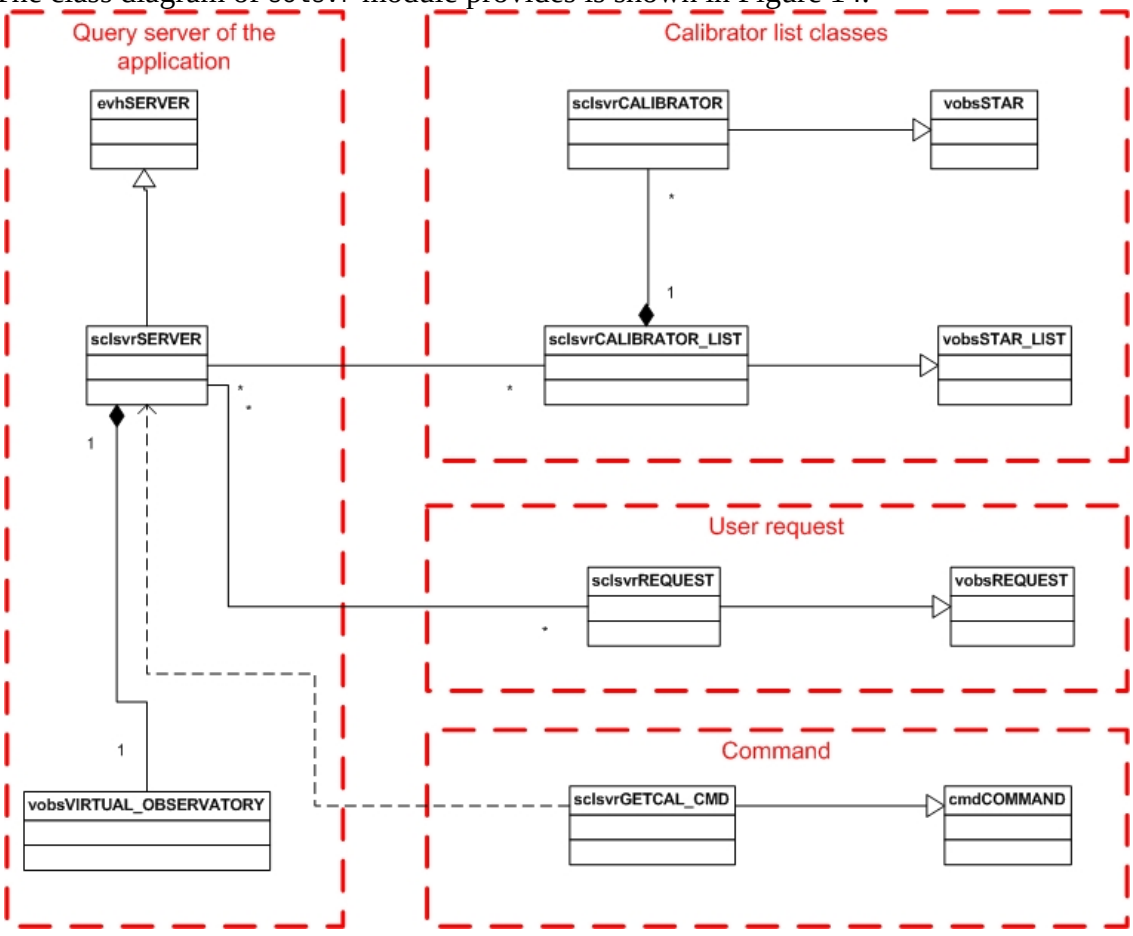

Figure 14 – sclsvr UML class diagram

The sclsvr module can be grouped in 4 collections:

1. the ‘calibrator classes’ group which contains classes related to calibrator and calibrator list:
   - `sclsvrCALIBRATOR` adds specific calibrator properties.
   - `sclsvrCALIBRATOR_LIST` handles the list of calibrators.
2. the ‘user request class’ contains class related to the user request:
   - `sclsvrREQUEST` completes the list of user constraints.
3. the ‘command classes’ contains classes related to the commands accepted by application:
   - `sclsvrGETCAL_CMD` deals with GETCAL command
   - `sclsvrGETSTAR_CMD` deals with GETSTAR command
4. the ‘server class’ contains the main application class:
   - `sclsvrSERVER` is the main application class.

3.2.1.3.2 Calibrator classes

3.2.1.3.2.1 Calibrator

The `sclsvrCALIBRATOR` class inherits from `vobsSTAR` class (see Figure 15) in order to add calibrator's properties, such as diameter or visibility. This is done by `AddProperties()` method.

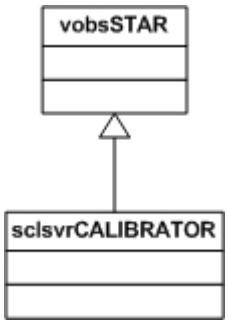

Figure 15 - Calibrator class

The calibrator's properties are computed using the `alx` library (see 3.2.1.2) by the `Complete()` method.

3.2.1.3.2.2 Calibrator list

The `sclsvrCALIBRATOR_LIST` class inherits from `vobsSTAR_LIST` class (see Figure 16) in order to manage a list of calibrators.

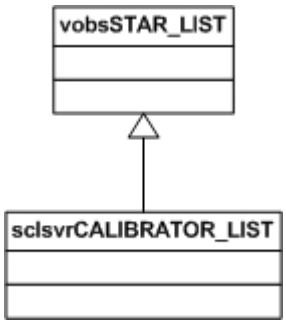

Figure 16 - Calibrator list class

The creation of a calibrator list from the star list is done by the `Copy()` method which uses the copy constructor of `sclsvrCALIBRATOR` class to create `sclsvrCALIBRATOR` object from a `vobsSTAR` instance.

3.2.1.3.3 User request class

The `sclsvrREQUEST` class inherits from the `vobsREQUEST` class (see Figure 17) in order to add interferometric parameters: maximum baseline and observing wavelength.

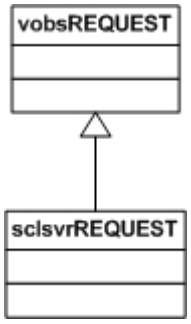

Figure 17 – User request

These parameters are used to compute the calibrator's properties (see 3.2.1.3.2.1).

3.2.1.3.4 Command classes

The `sclsvrGETCAL_CMD` and `sclsvrGETSTAR_CMD` classes, handling respectively the `GETCAL` and `GETSTAR` commands, are generated from `sclsvrGETCAL.cdf` and `sclsvrGETSTAR.cdf` command definition files (see [9]).
The description of these commands is given in 5.2 section.

3.2.1.3.5 Application server class

The `sclsvrSERVER` class is the main application class which inherits from `evhSERVER` class and has an instance of `vobsVIRTUAL_OBSERVATORY` as class member, as shown in Figure 18.

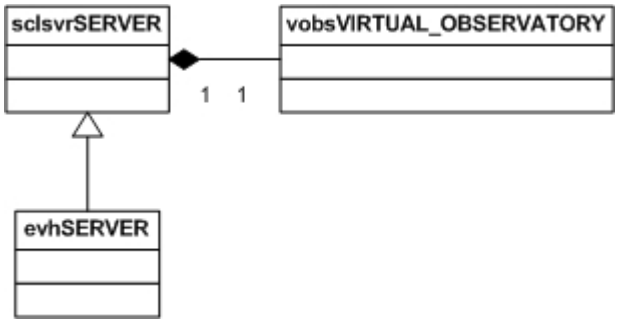

Figure 18 - Server class

The installation of the `GetCalCB()` and `GetStarCB()` callbacks to respectively handle the `GETCAL` and `GETSTAR` commands is done in the `AppInit()` method.

The Figure 19 shows the sequence corresponding to the `GETCAL` command handling.

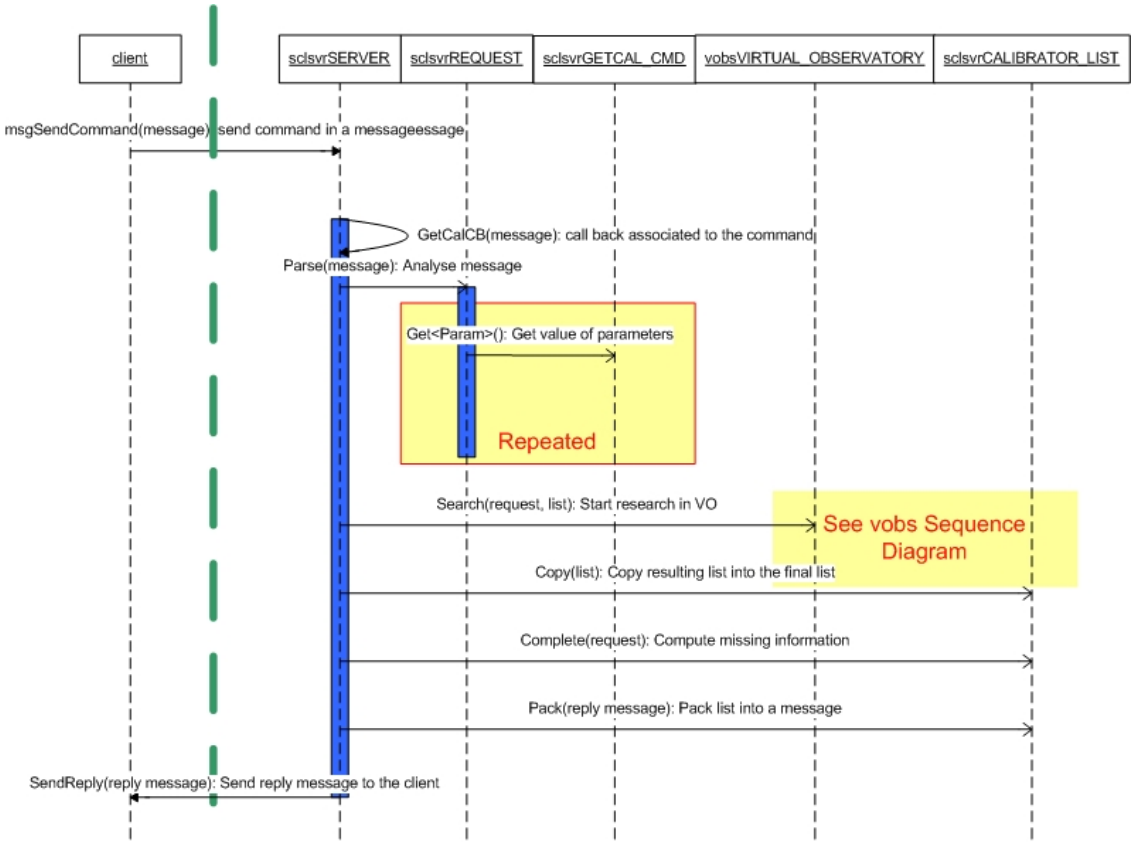

Figure 19- sclsvr sequence diagram

The `sclsvrSERVER` class is instantiated in the `main()` procedure defined in `sclsvrServer.c` file. The name of program is `sclsvrServer`.

3.2.1.4 How to

3.2.1.4.1 Add a new remote catalog

To add a new remote catalog, you have to create a new class inheriting from `vobsREMOTE_CATALOG`, called `vobsCATALOG_XXX` where `XXX` is the common name used to refer this catalog. This new class must overload the `WriteQuerySpecificPart()` method(s) to give the specific options used to query catalog; in order to retrieve expected star information in this catalog.

Use existing classes as example.

3.2.1.4.2 Modify a scenario

The scenarios are defined in the `vobsVIRTUAL_OBSERVATORY::LoadScenario()` method. You can:

- remove a catalog from a scenario by removing the corresponding `AddEntry()`,
- add a new catalog in a scenario by adding it using `AddEntry()`,
- change sequence querying order by moving `AddEntry()` calls.

3.2.1.4.3 Add a property to retrieve

If the property does not yet exist, you have to define the corresponding property Id in `vobsSTAR.h` file:

```
#define vobsSTAR_XXX  “XXXX”
```

where `XXX` is the property UCD.

And add this new property into the star property list in the `vobsSTAR::AddProperties()` method:

```
AddProperty(vobsSTAR_XXX, "Xxx", vobsFLOAT_PROPERTY, "%.3f");
```

or

```
AddProperty(vobsSTAR_XXX, "Xxx", vobsSTRING_PROPERTY);
```

If the property Id does not match a unique UCD or if there is some ambiguity with the UCD significance, you have to add the pair UCD/name corresponding to this property in `vobsCDATA::GetPropertyId()` method.

If this property is not yet retrieved from the catalog, you have to update the `WriteQuerySpecificPart()` method(s) to request it.

3.2.1.4.4 Modify a command

To add/remove a parameter to `GETCAL` or `GETSTAR` commands, you have to update the corresponding Command Definition File (`sclsvrGETCAL.cdf` or `sclsvrGETSTAR.cdf`), and to update either the `vobsREQUEST` class if this new parameter is used when querying catalog, or `sclsvrREQUEST` if it is only used when dealing with calibrators.

The class(es) exploiting this new parameter must be updated too.

3.2.2 SearchCal GUI

The SearchCal GUI consists in `sclgui` module, which contains the GUI of the application. The GUI is in charge to :

- forward user request to the SearchCal server, and retrieve the corresponding calibrator list,
- show the calibrator list in GUI,
- handle user interactions (filtering, saving …).

3.2.2.1 General UML class diagram

The GUI application is designed using the MVC paradigm. Therefore it is divided into three concerns:

- Model - encapsulates application data,
- View - obtains data from the model and presents it to the user,
- Controller - receives and translates input to requests on the model or the view.

As shown in Figure 20, the change notification mechanism is based on the MVC active model; i.e. the controller handles interactions with the views and translates them into actions to be performed on the model. The model applies changes on data and notifies attached views. Then the view queries the model to obtain data from the model and displays the information.

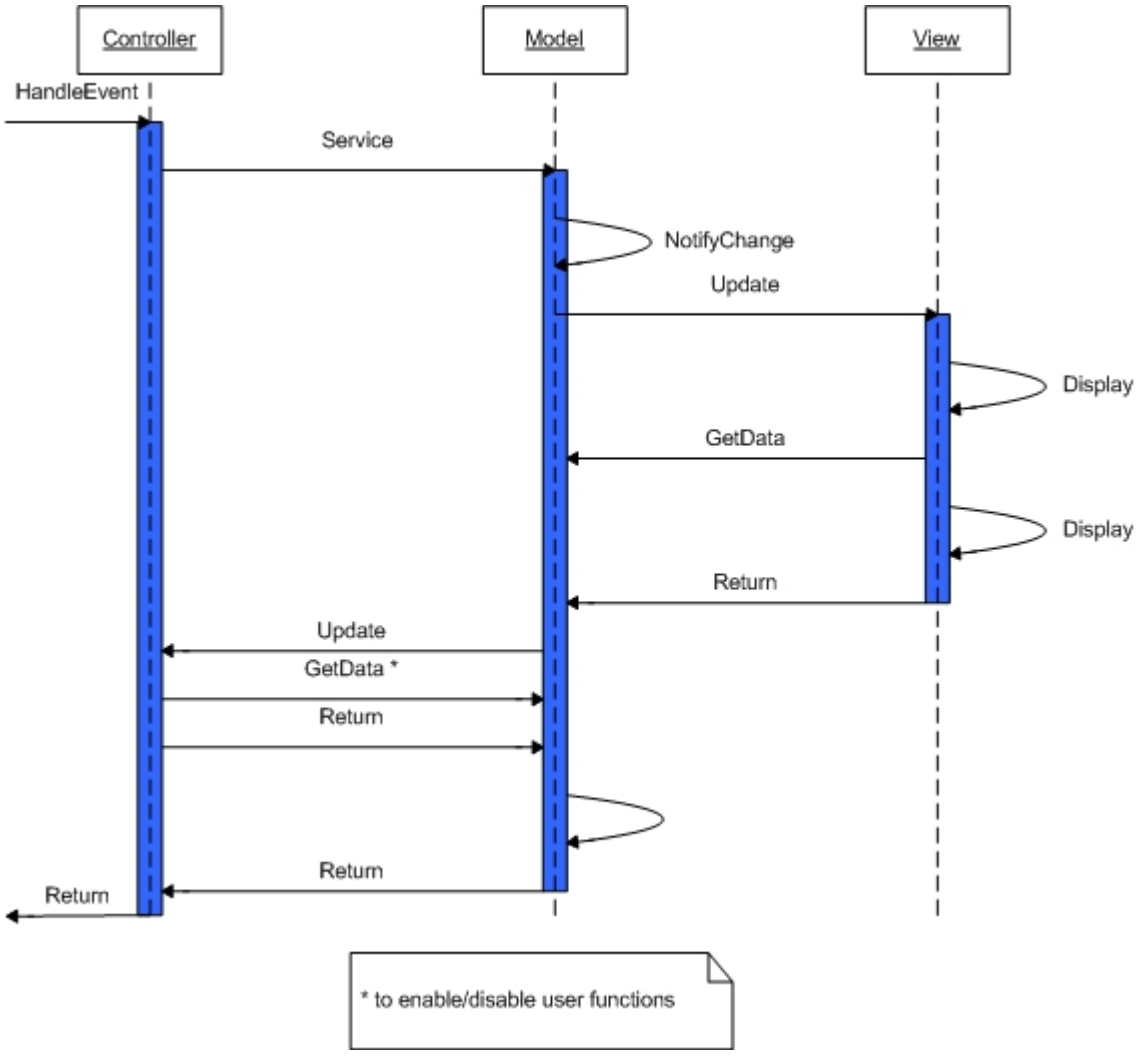

Figure 20- MVC design : active model

The Figure 21 shows all the `sclgui` classes. These classes are logically organized according to MVC design:

1. the 'model classes':
   - `sclguiCALIBRATOR_LIST_MODEL` encapsulates calibrator list,
   - `sclguiREQUEST_MODEL` encapsulates user request,
   - `sclguiFILTER_LIST_MODEL` encapsulates all filters to be applied on calibrator list.
2. the 'view classes' inheriting from `fndMVC_VIEW` class, excepted the 2 last static sub-panel classes:
   - `sclguiCALIBRATOR_LIST_VIEW` displays calibrator list,
   - `sclguiREQUEST_VIEW` displays user request parameters and science star information,
   - `sclguiACTIONS_VIEW` shows the default file names according the user request,

- `sclguiFILTER_VIEW` base class of the filter view classes,
- `sclguiDISTANCE_FILTER_VIEW` displays distance filter,
- `sclguiMAGNITUDE_FILTER_VIEW` displays magnitude filter,
- `sclguiSPECTRAL_TYPE_FILTER_VIEW` displays spectral type filter,
- `sclguiLUMINOSITY_FILTER_VIEW` displays luminosity filter,
- `sclguiVARIABILITY_FILTER_VIEW` displays variability filter,
- `sclguiMULTIPLICITY_FILTER_VIEW` displays multiplicity filter,
- `sclguiVISIBILITY_FILTER_VIEW` displays visibility filter.
- `sclguiBUTTONS_SUBPANEL` displays control buttons in main window,
- `sclguiCONFIRM_SUBPANEL` displays widgets related in the confirm window used to confirm file overwriting,

3. the ‘controller class’:
   - `sclguiCONTROLLER` handles user interactions and commands, and modifies data model accordingly.

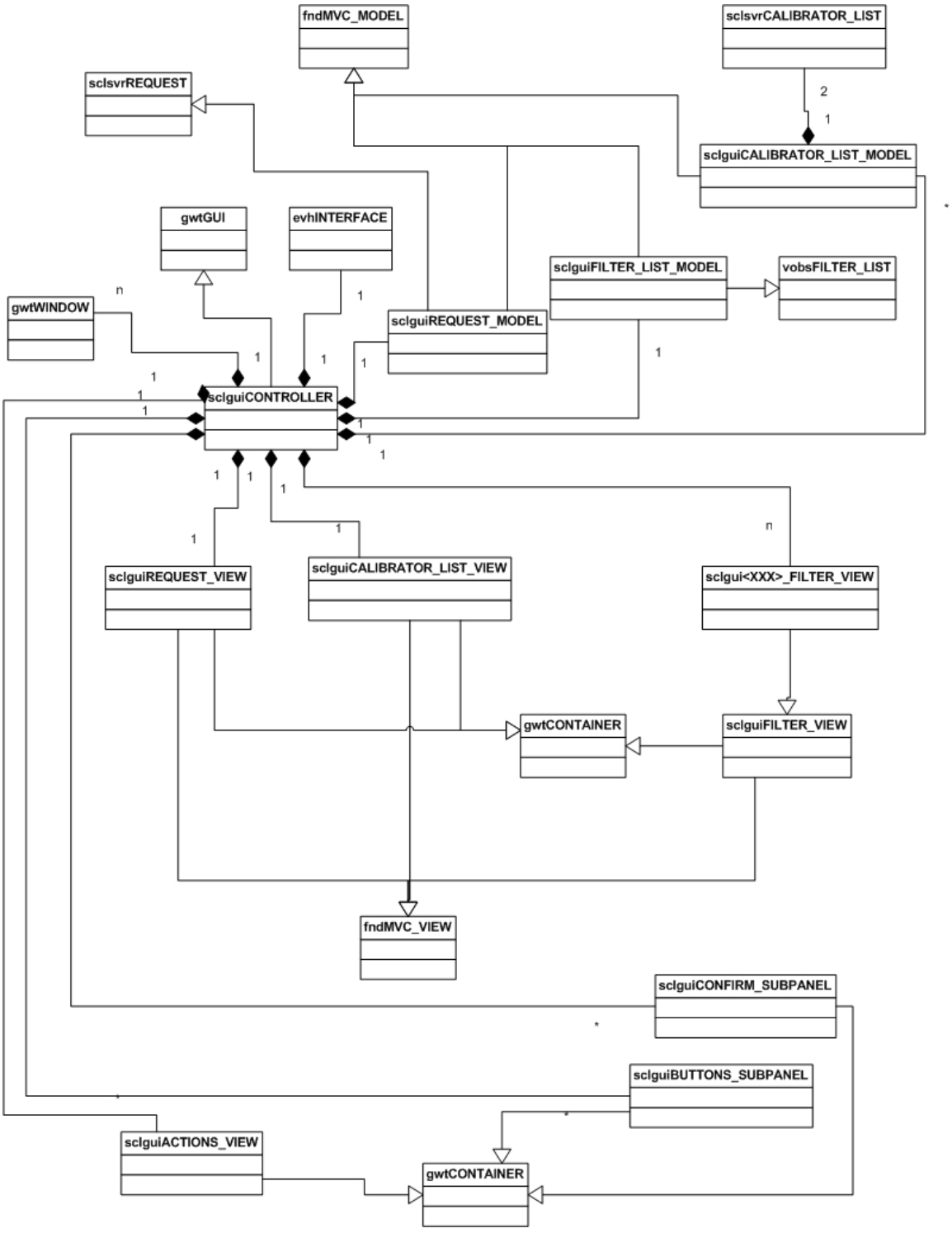

Figure 21 - sclgui UML classes diagram

3.2.2.1.1 Model classes

3.2.2.1.1.1 Calibrator list model class

The `sclguiCALIBRATOR_LIST` class, inheriting from the `fndMVC_MODEL`, handles the list of calibrators selected by user. To do this, it contains 3 `sclsvrCALIBRATOR_LIST` instances:

- the first one contains the list of potential calibrators returned by SearchCal server,
- the second one contains the list of calibrators deleted by user,
- the last one contains the selected calibrators; i.e. potential calibrators, minus filtered calibrators, minus deleted calibrators.

The list of potential calibrators can be set either by `SetList()` or `LoadFromFile()` method. The list of filters to be applied on this list is set by `SetFilterList()` method. And the calibrators to be removed from the final are given by `DeleteCalibrator()` method. Every time list of selected calibrators is changed, the *`sclguiCALIBRATOR_LIST_VIEW`* attached view is notified by calling the `NotifyViews()` method.

3.2.2.1.1.2 User request model class

The `sclguiREQUEST_MODEL` class, inheriting from the `fndMVC_MODEL`, encapsulates the `sclsvrREQUEST` class, in order to notify *`sclguiREQUEST_VIEW`* and *`sclguiACTION_VIEW`* attached views.

3.2.2.1.1.3 Filter list model class

The `sclguiFILTER_LIST_MODEL` class, inheriting from the `fndMVC_MODEL`, encapsulates the `vobsFILTER_LIST`. it holds all the following filters:

- `vobsVARIABILITY_FILTER`
- `vobsMULTIPLICITY_FILTER`
- `vobsMAGNITUDE_FILTER`
- `vobsDISTANCE_FILTER`
- `vobsSPECTRAL_FILTER`
- `vobsLUMINOSITY_FILTER`
- `sclsvrVISIBILITY_FILTER`

This class provides, for each filter, methods to set parameters, and enable/disable it. When filter parameter or state is changed, all the attached *`sclguiXXXX_FILTER_VIEWS`* views are notified.
The inline `Apply()`method is used by `sclguiCALIBRATOR_LIST` class to apply all filters on the potential calibrator list..

3.2.2.1.2 View classes

The Figure 22 shows the main window of the application. This interface is composed of sub-panels which are either static widget groups or MVC views.

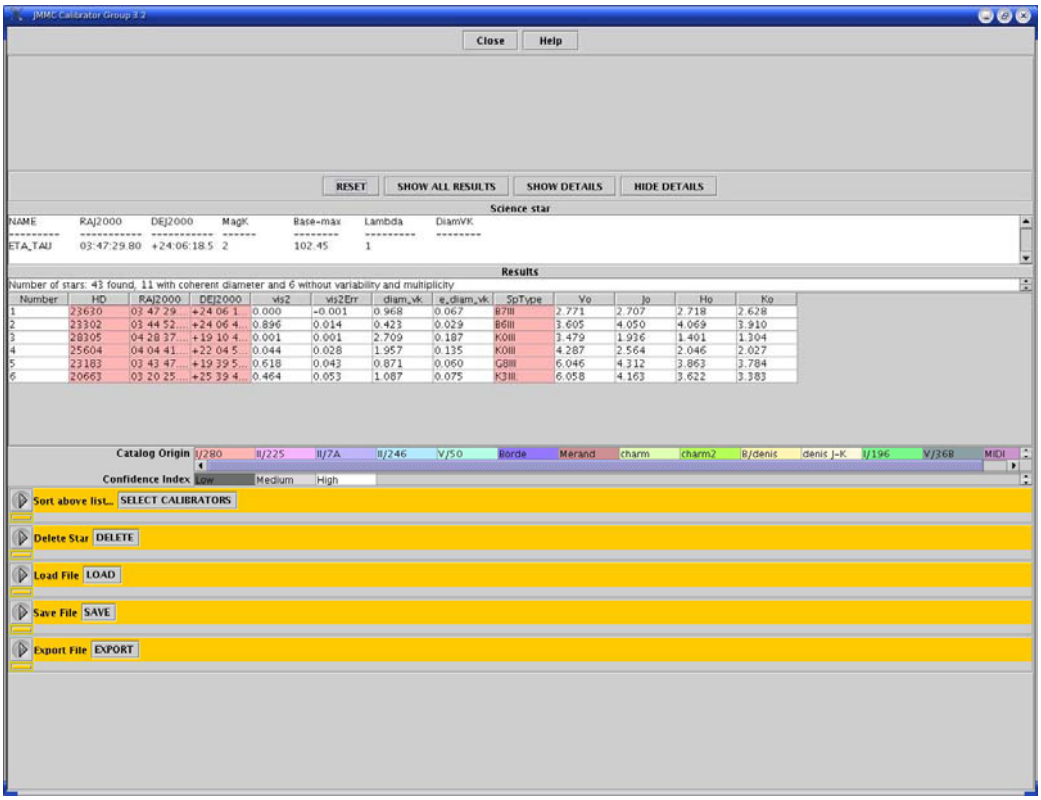

Figure 22 - Main window of the GUI.

3.2.2.1.2.1 Buttons sub-panel class

The `sclguiBUTTONS_SUBPANEL` class displays, as shown in Figure 23, the 4 main buttons of the GUI.

Figure 23- Buttons sub-panel

The callbacks associated to these buttons, defined in `sclguiCONTROLLER` class, are `ResetButtonCB()`, `ShowAllResultsButtonCB()`, `ShowDetailsButtonsCB()` and `HideDetailsButtonCB()`.

3.2.2.1.2.2 User request view

The `sclguiREQUEST_VIEW` class is the MVC view associated to the user request data model. It displays information on science star as well as request parameters, as shown in Figure 24.

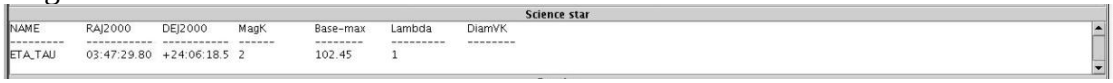

Figure 24- User request view

3.2.2.1.2.3 Calibrator list view

The `sclguiCALIBRATOR_LIST_VIEW` class is the MVC view associated to the calibrator

list data model. It displays the list of selected calibrators in a table with summary, and the legend giving the color code used to show catalog origin and confidence index, as shown in Figure 25.

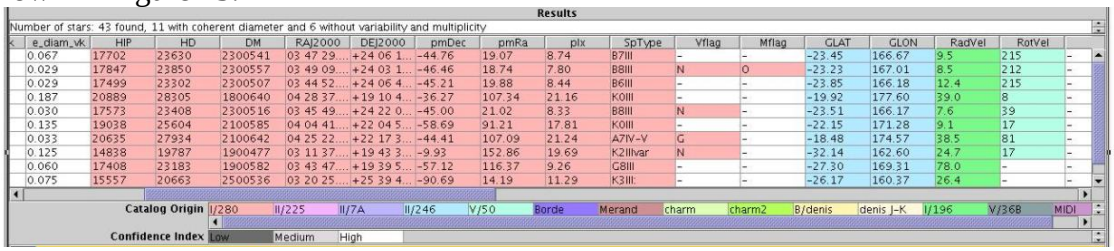

Figure 25 - Calibrator list view

3.2.2.1.2.4 Actions view

The `sclguiACTION_VIEW` class is the MVC view containing buttons for I/O file actions, and deletion/filtering actions, as shown in Figure 26. It is associated to the user request data model in order to set the default file name for saving or exporting actions. The callbacks associated to the action buttons, defined in `sclguiCONTROLLER` class, are `SelectButtonCB()`, `DeleteButtonCB()`, `LoadButtonsCB()`, `SaveButtonsCB()` and `ExportButtonCB()`.

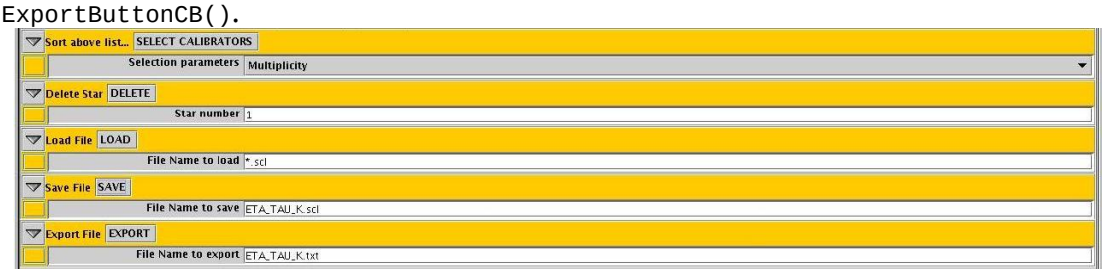

Figure 26 – Actions view

3.2.2.1.2.5 Filter views

The `sclguiXXX_FILTER_VIEW` classes are the MVC views associated to the filter data models. These views are displayed in separated windows, as shown in Figure 27 to Figure 33. Each window has an OK button which is associated to a `XxxxButtonCB()` callback, where `Xxxx` is the filter name. This callback sets the associated filter, and updates the calibrator data model.

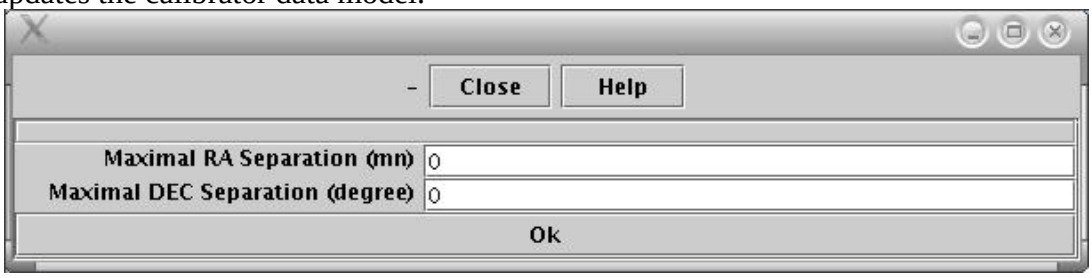

Figure 27 – Distance filter view

Figure 28 – Magnitude filter view

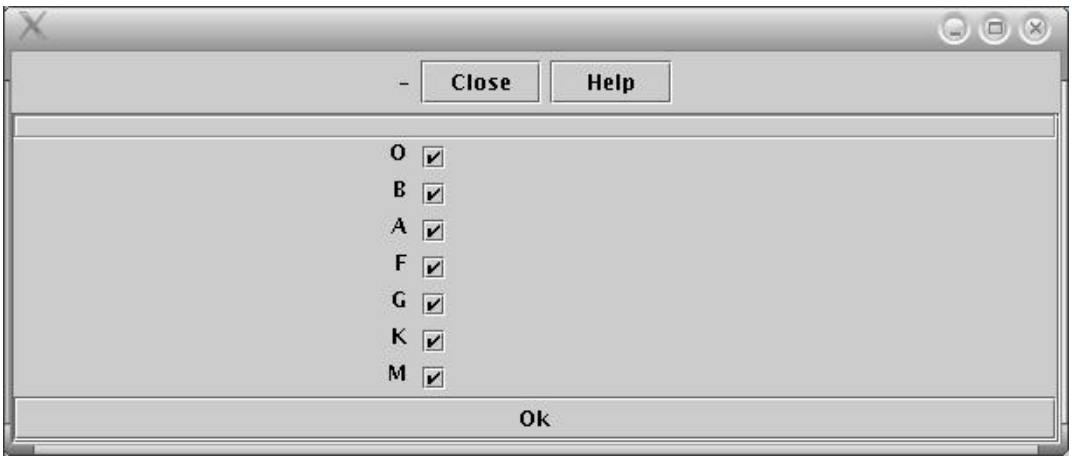

Figure 29 – Spectral type filter view

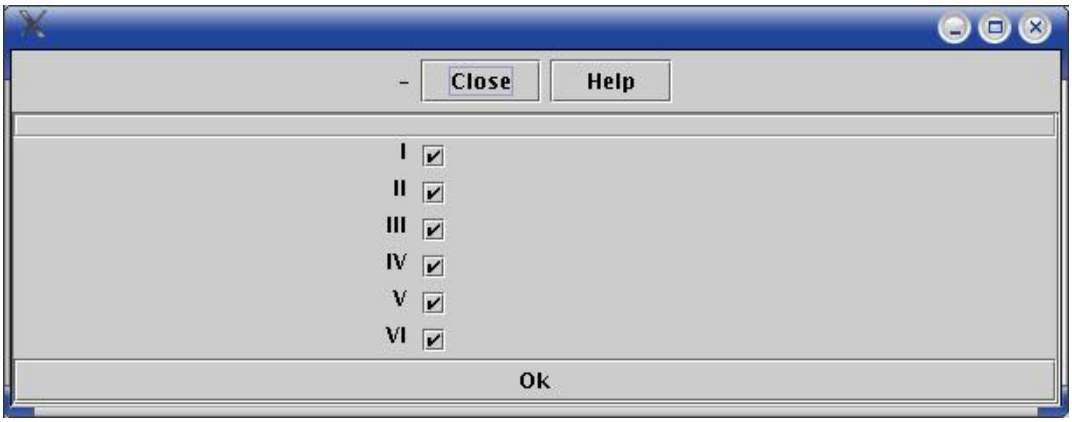

Figure 30 – Luminosity class filter view

Figure 31 – Multiplicity filter view

Figure 32 – Variability filter view

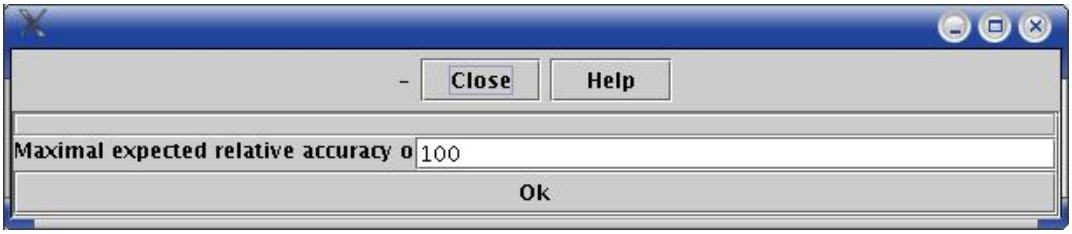

Figure 33 – Visibility filter view

3.2.2.1.2.6 Confirm sub-panel

The `sclguiCONFIRM_SUBPANEL` class is used to ask user the confirmation of file overwriting. When the name, given by user, corresponds to an existing file, this sub-panel is shown in a window allowing user to cancel operation or to overwrite file.

3.2.2.1.3 Controller class

The `sclguiCONTROLLER` class is the event handling server class. It inherits from `gwtGUI` class. It handles all callbacks associated to the GUI widgets, and the one associated to the `GETCAL` command. The Figure 34 shows the sequence corresponding to the `GETCAL` command callback, and the execution sequence of the callback associated to a filter.

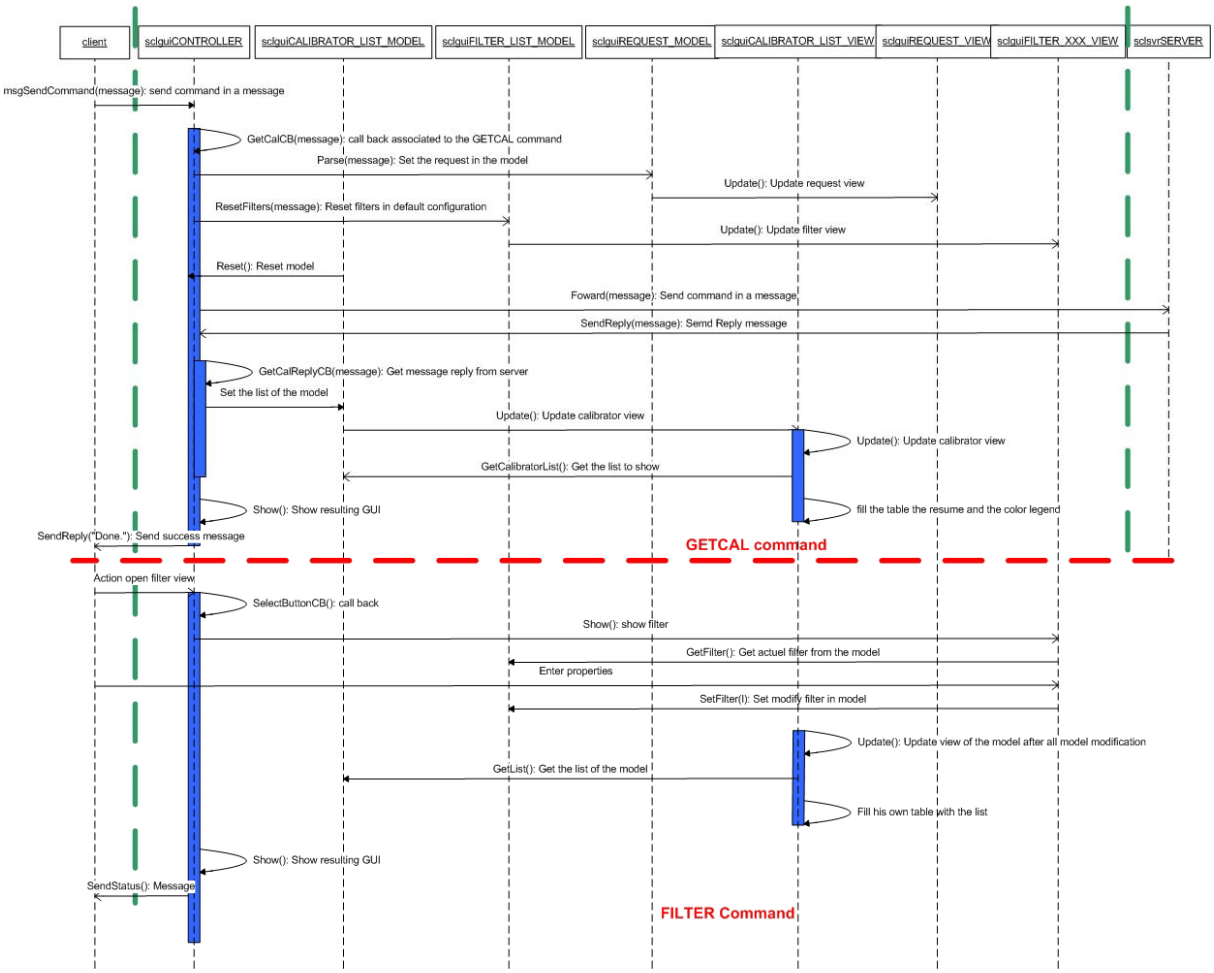

Figure 34 - Callback sequence diagram

3.2.2.2 How To

3.2.2.2.1 Add a new view

To add a new view, you have to create a new class derivating from the `fndMVC_VIEW` class, and probably `gwtCONTAINER` class to manage widgets used to represent information of the attached data model. Then you have to implement the `Update()` method which will be called every time the attached data model changes.

You have to declare this class as member of the `sclguiCONTROLLER` class, and attach it to the data model in its `AppInit()` method, using the `AddView()` method of the model.

4 Installation

4.1 Pre-requisite

The common software should be installed on the workstation, and the user account correctly configured. Refer to [9] for further details.
The XML-based GUI server must be installed and started.

4.2 Access to CVS repository

You should have an account on the `mariotti.fr` server, and set the `CVSROOT` and `CVS_RSH` environment variables, as shown below for bash.

```
export CVSROOT=:ext:[<user>@]mariotti.fr:/cvs/sw
export CVS_RSH=ssh
```

where *<user>* is your login on mariotti server.

4.3 Start installation

You should first retrieve the installation module, named sclins:

```
% cvs co sclins
<user>@mariotti.fr's password:
```

And then intall it:

```
% cd sclins/src
% make all man install
```

You should now start the installation:

```
% cd

% sclinsInstall
-> All the SCALIB modules will be installed (or just updated) from
   '<user home>' directory

Press enter to continue or ^C to abort

<user>@mariotti.fr's password:
....
```

You can obtain help on the installation script options using `-h` option.

4.4 Quick start

To start SearchCal software, just type:

```
% sclinsStart
```

Hereafter, is a simple command send to server to get resulting calibrator list saved in file.

```
% msgSendCommand sclsvrServer GETCAL “-objectName ETA_TAU –mag 2 –maxReturn 50 –diffRa 1200 –diffDec 600 –band K –minMagRange 0 –maxMagRange 4 –ra 03:47:29.7999 –dec 24:06:18.5 –baseMax 102.45 –wlen 1 –file SaveFile.txt”
```

And here, is a command to get the result displayed in the GUI, offering filtering/saving/exporting capabilities.

```
% sclinsQuery -objectName ETA_TAU –mag 2 –maxReturn 50 –diffRa 1200 –diffDec 600 –band K –minMagRange 0 –maxMagRange 4 –ra 03:47:29.7999 –dec 24:06:18.5 –baseMax 102.45 –wlen 1
```

It is also possible to send a command to GUI server (`sclsvrServer` and `sclguiControl` should be running):

```
% msgSendCommand sclguiControl GETCAL “-objectName ETA_TAU –mag 2 –maxReturn 50 –diffRa 1200 –diffDec 600 –band K –minMagRange 0 –maxMagRange 4 –ra 03:47:29.7999 –dec 24:06:18.5 –baseMax 102.45 –wlen 1”
```

5 Data Design

5.1 Files

The configuration files are listed in the following table.

File	*Module*
Local Catalog `vobsMidiCatalog.cfg`	vobs

Astronomical Table
`alxAbsIntPolynomial.cfg`
`alxAngDiamPolynomialForBrightStar.cfg`
`alxColorTableForBrightDwarfStar.cfg`
`alxColorTableForBrightGiantStar.cfg`
`alxColorTableForBrightSuperGiantStar.cfg`
`alxExtinctionRatioTable.cfg`
`alxStarPopulationInKBand.cfg`
	alx

Command
`sclsvrGETCAL.cdf`
`sclsvrGETSTAR.cdf`
	sclsvr

Table 2 - Inventory and short description of the SearchCal software configuration files

5.2 Commands

Here is the complete list of commands handled by `sclsvrSERVER`:

```
GETCAL - Get a list of interferometric calibrators for a given science
object.
GETSTAR - Get informations about the given star.
VERSION - Returns the version of the software.
DEBUG - Changes logging levels on-line.
HELP - Get help on supported commands by the application.
STATE - Returns the state and the sub-state of the server.
EXIT - Halt the software.
```

Here is the complete description of the GETCAL command:

```
NAME
      GETCAL - Get a list of interferometric calibrators for a given
      science object.

SYNOPSIS
```

```
GETCAL -objectName <string> -mag <double> -maxReturn <integer> -
diffRa <integer> -diffDec <integer> -band <string> [-minMagRange
<double>] [-maxMagRange <double>] -ra <string> -dec <string> -
baseMax <double> -wlen <double> [-file <string>]

DESCRIPTION
      Get a list of interferometric calibrators for a given science
      object. It returns any stars found in CDS catalogs around the
      science object, which can be used as calibrator during an
      observation. It computes diameter and expected visibility for
      each found calibrator.

PARAMETERS
      -objectName <string>
            science object name

      -mag <double> (range from ’-5.0’ to ’20’)
            science object magnitude

      -maxReturn <integer> (default = ’50’)
            maximum number of calibrators to be returned

       -diffRa <integer> (default = ’1800’) (unit = ’arcmin’) (range
      from ’0’ to ’3600’)
            right ascension value of the search box size

      -diffDec <integer> (default = ’600’) (unit = ’arcmin’) (range
      from ’0’ to ’1800’)
declinaison value of the search box size

      -band <string> (default = ’K’)
            observation band

      -minMagRange <double> (range from ’-5.0’ to ’20’)
            minimum value of the range magnitude

      -maxMagRange <double> (range from ’-5.0’ to ’20’)
            maximum value of the range magnitude
```

```
-ra <string> (unit = 'HH:MM:SS.TT')
       right ascension coordinate of the science object

-dec <string> (unit = 'DD:MM:SS.TT')
       declinaison coordinate of the science object

-baseMax <double> (default = '100') (unit = 'm') (minimum value
of '0.1')
       maximum baseline length

-wlen <double> (unit = 'um') (range from '0.5' to '20')
       observing wavelength

-file <string>
       name of the file in which results should be saved
```

5.3 Catalog classes

Here is the complete list of catalog classes with the corresponding name used in the software.

Catalog class	*Catalog name*
vobsCATALOG_ASCC	I/280
vobsCATALOG_BSC	V/50/catalog
vobsCATALOG_SBSC	V/36B
vobsCATALOG_CHARM	J/A+A/386/492/charm
vobsCATALOG_CHARM2	J/A+A/431/773
vobsCATALOG_CIO	II/225/catalog
vobsCATALOG_DENIS	B/denis
vobsCATALOG_DENIS_JK	J/A+A/413/1037
vobsCATALOG_HIC	I/196/main
vobsCATALOG_LBSI	J/A+A/393/183/catalog
vobsCATALOG_MASS	II/246/out
vobsCATALOG_MERAND	J/A+A/433/1155
vobsCATALOG_MIDI	MIDI
vobsCATALOG_PHOTO	II/7A

Table 3 - Inventory of catalog classes