A Basic Class Architecture for Data Acquisition in Astronomy: I - Requirements and Analysis†

F. Palagi[1] and G. Comoretto[2]

[1] IRA - INAF, Sezione di Firenze, l.go E. Fermi 5, 50125 Firenze (Italy)
[2] Osservatorio Astrofisico di Arcetri - INAF, l.go E. Fermi 5, 50125 Firenze (Italy)

IRA Technical Report n.° 385/06
Firenze 20/02/2006

Version of February 22, 2006

†source: /palagi/strumenti/basearch/doc/analysis/analysis.tex

Contents

Abstract

Astronomical observations are carried out using instrumentation (mainly detectors and telescopes) whose characteristics changes according to the spectral window involved. However there are still lot of common features, which are independent of the spectral window involved.

While developing data acquisition programs for some radioastronomical instruments built in Arcetri, these common aspects showed up clearly.

This report is the first of a series of reports that describe the Object Oriented analysis, design and implementation of a software architecture which will be the basis for the development of the control and acquisition programs of two instruments:

- *The ARCOS correlators installed at the Medicina and Noto radiotelescopes.*
- *A demonstration total power radiotelescope installed in Arcetri.*

The authors believe that this architecture can be used also to develop data acquisition programs in spectral ranges other than the radio band.

One possible application will be the definition of a standard interface through which the control programs of detectors installed on the SRT Radio Telescope can access to the ACS-Alma Common Software services.

The design process is as follows. The program requirements *are reviewed and modeled. The* analysis *model is then built: it is composed of different views and diagrams which illustrate the static, dynamic and functional behaviour of the model. The* design *model (the second report of this series) is then based on the analysis model which is modified and extended in a consistent way, following an iterative approach. The design model describes classes and objects in finer details so that they can be easily* implemented *in any object oriented programming language.*

The class architecture is implemented as a C++ class library.

Introduction

In this document we describe the instrumentation and the operations needed to carry out a astronomical observation. The approach is as general as possible, to design and implement a software architecture over which specialized applications can be more easily developed.

You can think of this structure as a software bus where you plug in the essential devices needed to carry on an astronomical observation, such as a detector, a telescope and an archiving system. The software bus provides the links to transfer commands and data from one device to another. Each *plug* in the bus defines the interface that the specialized device must conform to.

A command interpreter and a basic command list are also provided. The command list can be extended to meet the needs of the specific devices in each application.

The design process uses an Object Oriented approach and the Unified Modeling Language (UML) as the design language. The Object Oriented technology is one of the most advanced tool in software engineering that enhances code reusability and software maintenance. Nowadays it is widely use in the astronomical community and has been adopted for the development of AIPS++ and the software for the ALMA project (ACS).

Chapter 1

The requirements

In recent years the Radioastronomy Group of the Arcetri Astrophysical Observatory, in cooperation with the Institute of Radio Astronomy – Section of Florence, has developed some data acquisition programs for radioastronomical detectors. During this work some common aspects showed up clearly while describing the requirements for each program. This chapter describes these common aspects.

The following devices are the minimum set-up needed to carry out an astronomical observation.

- a telescope,
- a detector,
- an archiving system,
- a command interpreter,
- a display.

Also there is a minimum set of operations required to complete an observation. They are:

a) Input of the parameters used to set-up the equipments.

b) Set-up of the equipments.

c) Measurement execution, when the detector produces an *image* of the sky.

d) Readout of the *image* from the detector.

e) Display of the image for monitoring purposes.

f) Archiving the *image* in the archive system.

g) Log of the instrumentation activity.

Operations from *b*) to *f*) may be repeated more than once in a cyclical way.

In the following paragraph the role and functionalities of each device is described in greater details.

1.1 The Telescope

The telescope is a device that can be oriented in some specified direction of the sky, collects the incoming radiation and feeds it to one of its focuses, where a detector is placed.

In most cases the telescope is oriented to a source (of radiation) which has a few specific attributes. Among these we can list the following:

- The name.
- The coordinates that are expressed in a reference system which, generally, does not depend on the current time or the telescope location.
- The parallax, proper motion, radial velocity (usually referred to the LSR) and other physical properties.

As time is a crucial item in astronomical observations, a clock is needed to provide Universal Time (UT) and Sidereal Time (ST).

The telescope is moved using either a (Hour Angle, Declination) or topocentric (Az,El) mount. The telescope control system provides the correction for the telescope pointing errors.

The telescope has a communication port to provide a link to the outside world for remote control.

The telescope can provide its status, which includes:

- The current UT and ST.
- Its current position and its radial velocity component along the line of sight, referred to the LSR.
- Its status of motion: tracking, slewing, stopped ....
- Whether it is aligned (ON) or not aligned (OFF) with the source.
- Its location (name of the location, its geographical coordinates and height above sea level).
- The current meteo parameters.
- The source being observerd.
- The applied pointing corrections.
- The current and commanded pointing offsets.
- The current focus setting.
- The table of the available focuses.

1.2 The Detector

The *Detector* is the device that measures the power or intensity of the radiation collected by the telescope.

The determination of the power/intensity scale implies the use of calibration procedures which uses standard astronomical sources (calibrators) and (optionally) internal standard sources.

Any astronomical measurement produces an *image*, i.e. a set of counts, whose physical meaning is specified by a set of descriptive parameters. For instance they may include the sky position, the image extent and resolution both in space and frequency (or wavelenght), and its power/intensity scale.

The angular extent (field of view) and the sampling spacing of the image depend on the distribution of the detector elements (pixels) in the focal plane. The resolution of the image depends on the wavelength of observation and the linear dimension of the telescope primary mirror. The spectral extent and resolution depend only on the spectral response of the detector. If the detector is tunable in frequency the spectral response (instantaneous band) can be moved within a wider frequency interval (observable band).

1.3 The Archiving System

Any kind of measurement is stored on a permanent device for later use. Apart from the choosen archiving format, a file name and a storing device specification must be given.

We require that the archiving system conforms to one of the formats that are most common in the astronomical community, such as FITS and CLASS. TOOLBOX is also required because it has been used in Arcetri from the beginning of the radio spectroscopic activity.

We list the most relevant information to be included in the image description.

a) Source description (name, coordinates, epoch, V_{LSR} ...)

b) Telescope description (name, geographical coordinates, height above sea level....).

c) Detector description.

d) Name of the acquisition program.

e) Observer's name.

f) UT at the beginning and at the end of the measurement.

g) Data structure (Number of array dimensions, their lenght, number of subscans).

h) Comments.

1.4 The Log system

A log system monitors the instrumentation activity writing messages both on a permanent device as well as on the display.

1.5 The Command interpreter

Observer interaction can be done through different kind of interfaces: graphical and text interfaces.

A **text interface** reads command lines (ascii strings) which are interpreted by the *Command Interpreter*. Command lines can be edited in a script file which can be read by the *Command Interpreter* for batch activity. Instrumentation control is done using a set of predefined commands that constitute the instrumentation language.

The command interpreter reads the commands from the interface, checks for their validity and executes the corresponding operations. Commands can be used to:

a Define new parameter values.

b Carry out the observation.

c Request the current status of the system.

A **graphical user interface** (GUI) provides the same functionality as the text interface. It is composed of graphic controls through which the user interacts with the system. The GUI includes a text window where text commands can be entered.

1.6 The Display

The Display main task is to show the information coming out from the system, such as images and parameter values. It and must be configurable to represent the hardware structure of the devices composing the data acquisition system.

1.7 The *Business* model

The requirement description is often represented in a synthetic form by the so called *business* model. The *business use-case* diagram (fig. 1.1) and the business class diagram (fig. 1.2) represent what services the system should provide and the static structure of the system.

Two activity diagrams illustrate how an observation (fig. 1.3) can be done and how commands from the consolle can be managed (fig. 1.4).

The complete business model can be found at the URL location:
`http://www.arcetri.astro.it/~palagi`

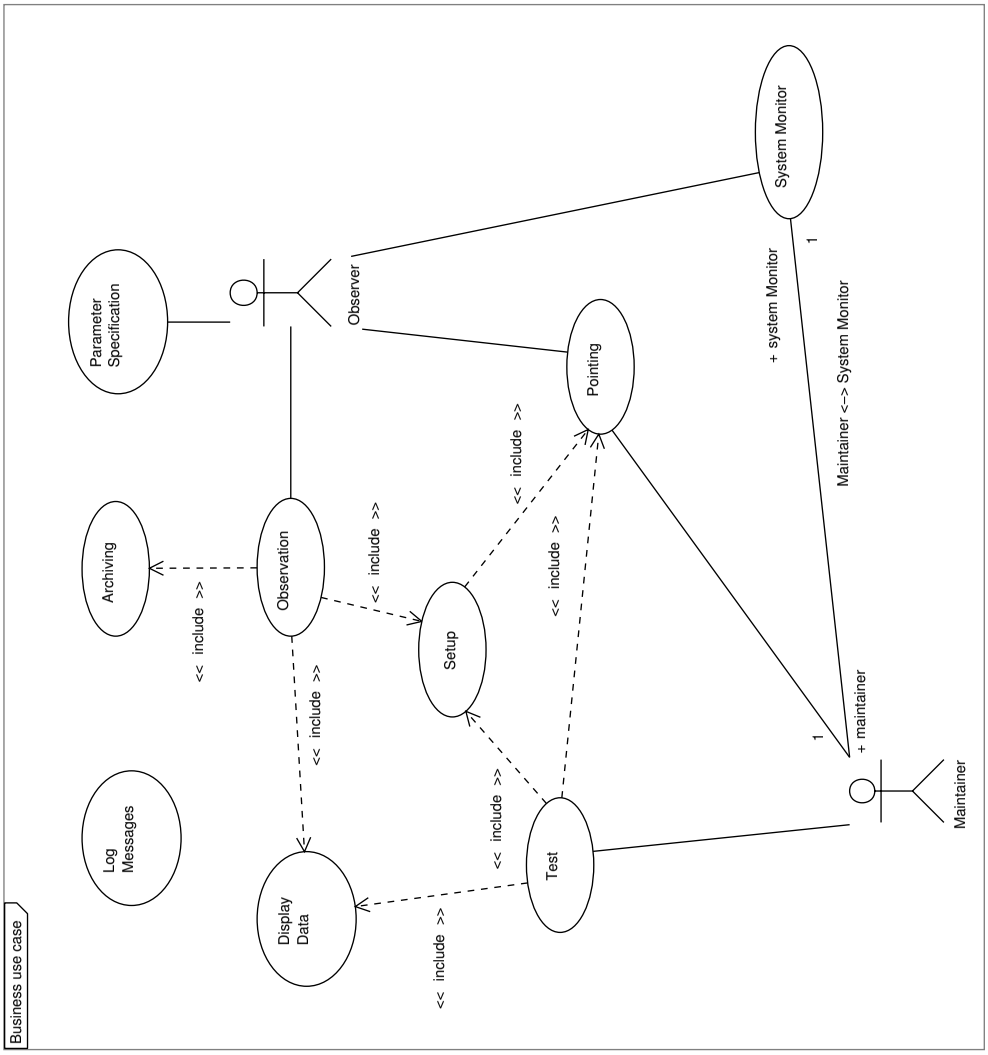


Figure 1.1: Use cases of the program.

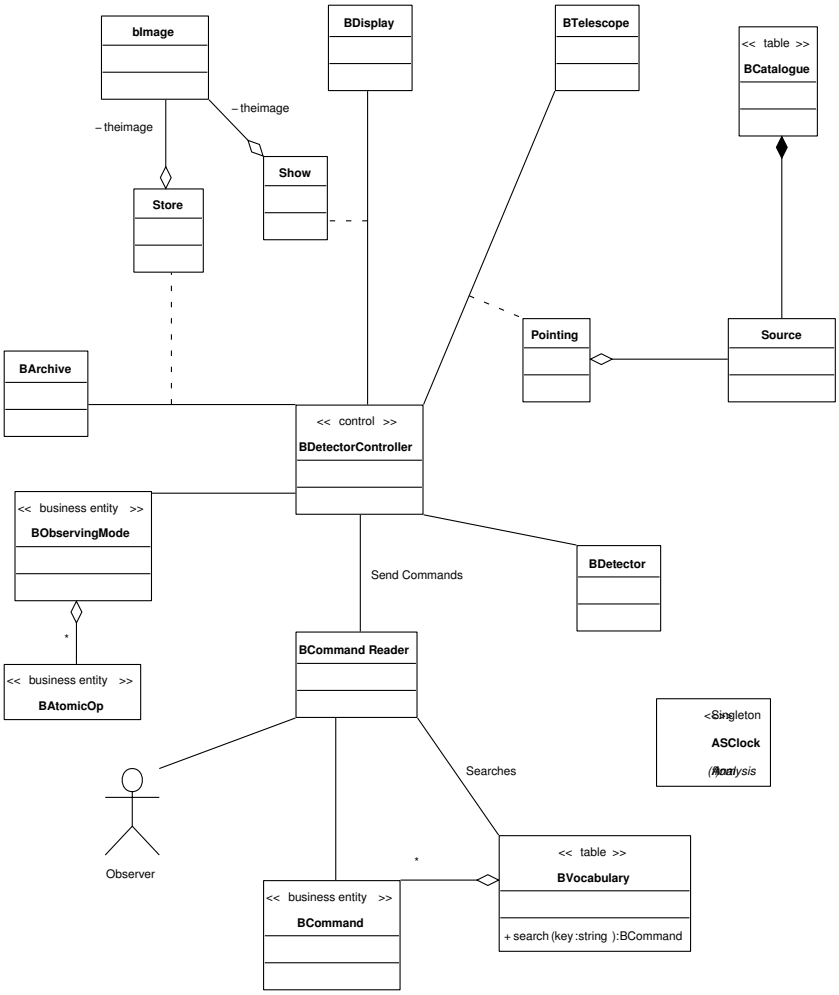


Figure 1.2: Requirements description: the so called *business class* diagram.

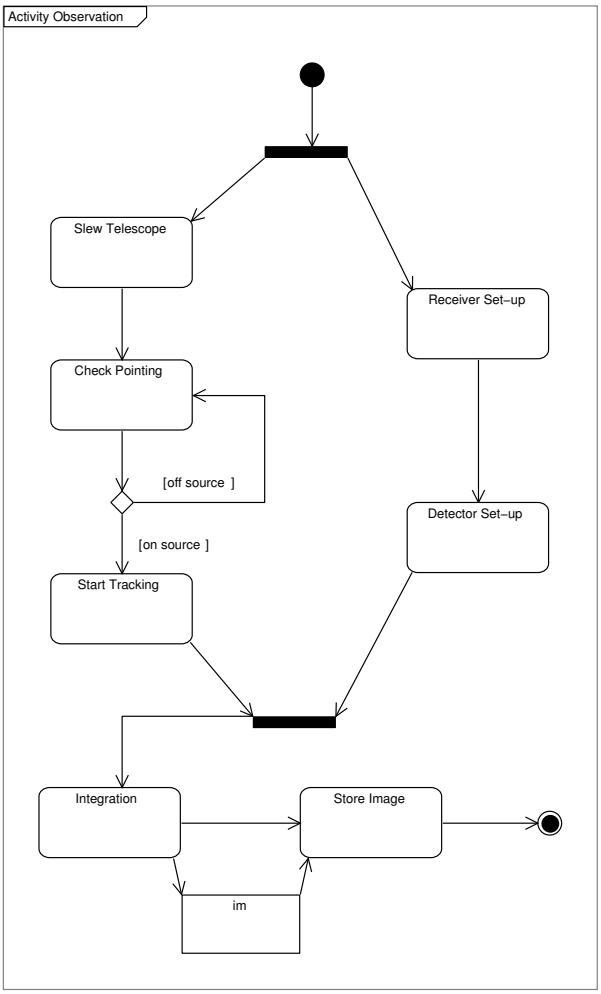


Figure 1.3: Requirements description: the observation activity diagram.

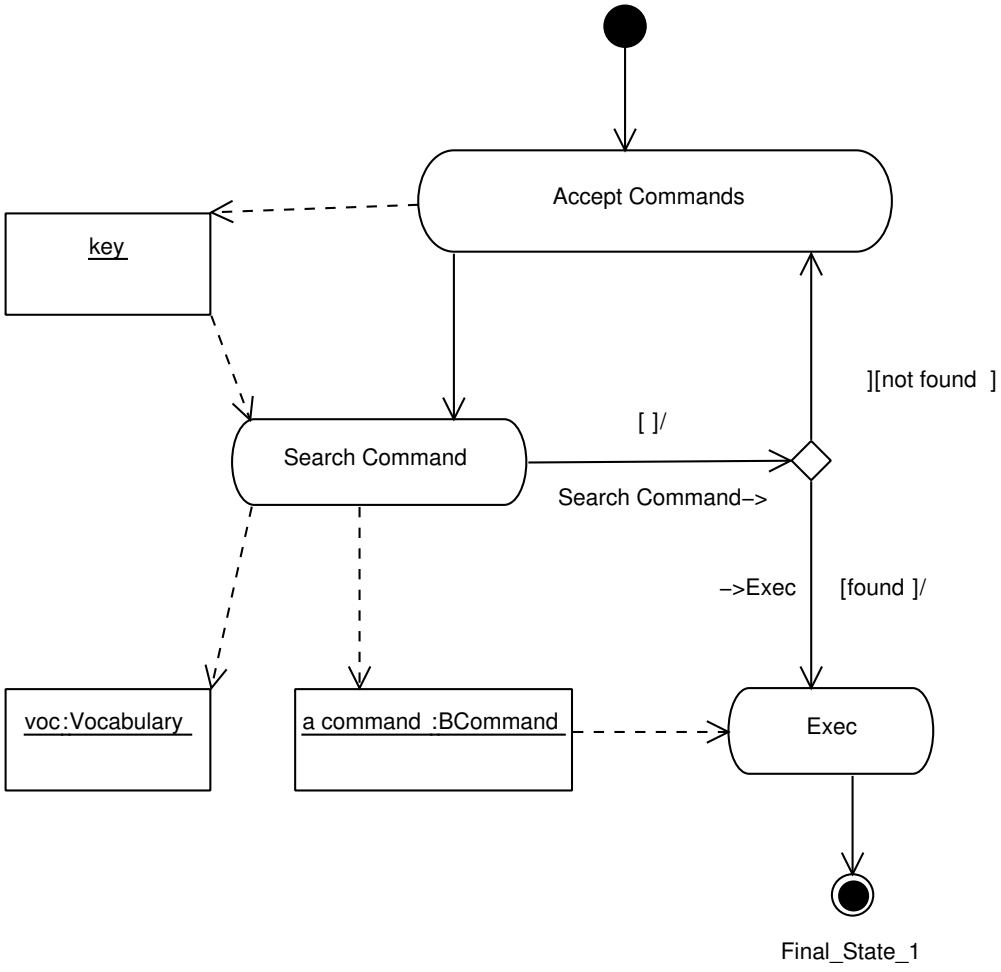


Figure 1.4: Requirements description: the command input activity diagram.

Chapter 2

Requirement analysis

The analysis of the program requirements consists in defining the static view of the system and in the use case specification. The final model is the result of several iterations between this two steps. Each use case is realized by a collaboration that show how classes cooperate in the use case realization.

2.1 The Object Model

The class model describes the classes in the system together with their relations, from a static point of view. Classes are identified starting from the *business* class diagram of fig. 1.2 and reviewing the requirement description. Each class is described in separate paragraphs with emphasis on functional and information requirements. Functions to access class attributes are not listed explicitly.

A detailed list of attribute and operations can be found in the web document.

The role of some class in the system can be better described through a `<<stereotype>>` label. For instance, classes that interact with the outside world are stereotyped as `<<boundary>>` while classes that represent items typical of the *business* being handled are sterotyped as `<<business entity>>`. Finally `<<control>>` classes coordinate the activity of other classes.

2.1.1 Detector

General description The detector is the core class of this architecture. This statement is based on the following very simple considerations. If one has even a very sofisticated and large telescope, but no detector in its focus, no measurement can be done. On the other hand if you have a detector, but you do not have any telescope, you can do a measurement though of a very bad quality.

As the detector is the *core* of the design, the `Detector` class has the focus of control while doing an observation, therefore it knows the sequence of steps needed to get an image of the sky. The `Detector` is stereotyped as a `<<control>>` class. It sends to the telescope the request to point the sky object we want to observe, takes a good quality image of the observed source and finally asks the archive to store the image in some suitable format.

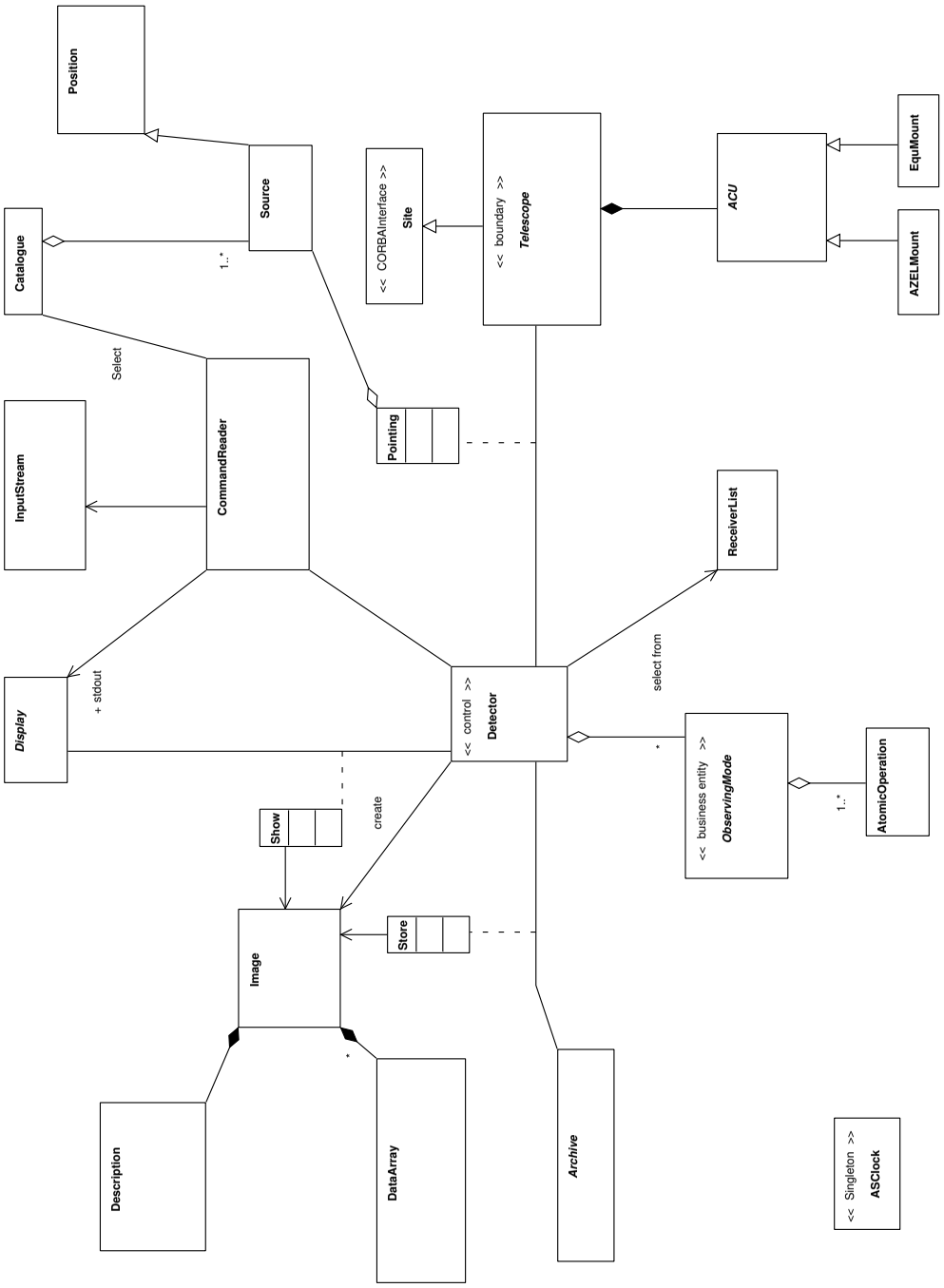


Figure 2.1: Requirements specification: the class diagram.

Functional requirements

- The `Detector` gets commands from `CommandInterpreter`.
- The `Detector` asks `Telescope` to point the source and to collect its radiation.
- The `Detector` gets an `Image` of the source.
- The `Detector` sends the `Image` to `Display`.
- The `Detector` asks `Archive` to store the `Image`.

Informative requirements The detector provides a set of specific `ObservingMode` objects.

Constraints None

2.1.2 Observation type

General description Each kind of detector has its own operation to get an image of the source. The `ObservingMode` class capture this aspect of the system. In turn an `ObservingMode` consists of a sequence of *atomic* operations.

It is contained in `Detector`.

Informative requirements

- Code.
- Name.
- Number of repetitions.

Functional requirements

- Execution of the operation sequence.

Constraints

2.1.3 Atomic Operation

General description Calls the function that realizes each elementary step in the observation sequence, i.e. pointing, calibration, measurement etc..

It is contained in `ObservingMode`.

Informative requirements

- Code
- Function to be executed.

Functional requirements

- Run the associated function.

Constraints None

2.1.4 Image

General description Contains the output of an observation in the form of an array of values and the physical description of the image. The image is a typical product of any astronomical system so this class is stereotyped as *business entity*.

Is created by `Detector`.

Informative requirements

- The image data array.
- The image description

Functional requirements

Constraints None

2.1.5 Image Data

General description It is the array that contains the observed data. It may have up to four dimensions.

Informative requirements

- The data array.
- The number of points in the array.
- The number of axes of the array
- The dimension of each axes of the array.

Functional requirements

- Data insertion.

Constraints

- It is part of `Image`.
- Up to four dimensions.

2.1.6 Image Description

General description Contains the physical description of the image. It is structured into sections.

Informative requirements

- source section.
- detector section.
- telescope section.

Functional requirements

- Data editing.

Constraints

- It is part of `Image`.

2.1.7 Telescope

General description The telescope is located in a site on the earth (by the moment we do not consider a telescope mounted on a satellite), collects the radiation from the observed sky object. To do this it uses a HourAngle-Declination or topocentric (Az,El) mount. This class is stereotyped as `<<boundary>>`.

It is derived from `Site`.

Functional requirements

- Receives pointing messages from the detector.
- Converts the source coordinates to coordinate system of the mount.
- Moves to the requested direction.
- Moves to a predefined stow position.
- Return its status, upon request.

Informative requirements

- Description of the telescope location: site name, geographical coordinates.
- Description of the telescope mirror: dimension, focal lenght.
- Stow position (coordinates).
- Current status of motion: tracking, slewing, stopped.

Constraints None

2.1.8 Archive

General description It Receives an `Image` from the detector, formats and stores it in the output file.

Informative requirements

- The output file.

Functional requirements

- Gets the image from the detector.
- Formats the image data.
- Formats the image description
- Writes to the output file.

Constraints None

2.1.9 LogSystem

Stores and display messages from the other classes in the system. Stereotyped as `<<Utility>>`.

Informative requirements

- The log file
- The log display

Functional requirements

- Writes in the log file
- Send message to the display

Constraints None.

2.1.10 CommandInterpreter

General description This class manages the input from the Observer or the batch input stream. It identifies the input command, validates it and executes the associated operation.

Informative requirements

- List of available commands.

Functional requirements

- Creates the newly accepted command.
- Accepts commands from the input stream.
- Executes the associated operation.

Constraints

2.1.11 Command

General description This class describes the command functions. It parses the command line into the keyword and the parameter list. It converts the parameters in the input line to the internal values.

Informative requirements

- Keyword.
- List of parameters.
- Keyword separator.
- Parameter separator.

Functional requirements

- Parses the input line into the keyword and the list of parameters.
- Converts the input parameters.

Constraints None.

2.1.12 Vocabulary

General description This class holds the functions that execute commands. It searches the command in its list and run it.

Informative requirements

- Keyword.
- Function to be run.

Functional requirements

- Maintains the list of commands.
- Searches the command code in the list of commands.
- Runs the function.

Constraints None.

2.1.13 Source

This class describes the source or sky object to be observed. A source is characterized by two set of parameters: astrometric and physical parameters. Astrometric parameters include name, position in a specific reference system and eventually proper motions.

Physical parameters include all characteristics that are usefull to perform an observation, such as magnitude, radial velocity and the like. By the moment we identify:

distance Distance from the sun.

vlsr Radial velocity with respect to the Local Standard of Rest [km/s].

Flux Density A value [Jy] at a specific frequency [GHz].

It is derived from Position

Informative requirements Name. Physical parameters, such as its spectral distribution.

Functional requirements

Constraints None.

2.1.14 Position

This class describes the position of a point in a coordinate system.

Informative requirements

- X value.
- Y value.

Functional requirements

Constraints None.

2.1.15 Clock

General description A class `Clock` is added to provide time informations to all the classes in the model. As it is used by many of these classes, no explicit link is shown in the diagram.

It is used by many objects in the system.

Informative requirements System time.

Functional requirements Sidereal Time, Universal Time, Local time.

Constraints There is only one object of this class in the system.

2.2 Behaviour specifications

The *use-cases* specification describes the various functionalities of the program in more details.

Each Use-case may be decomposed in more detailed use-cases connected by `<<include>>` and `<<extends>>` relations. The first one is used to gather functionalities that are (or might be) common to more than one use-case, the second indicates some extension (e.g. exception conditions or options) to the standard behaviour.

Fig. 2.2 shows the *use-case* diagram of the requirement specifications. In the following sections each use case is described in text form and through sequence and collaboration diagrams. This activity aims to identify operations of the classes that collaborate in the realization of each use-case.

A scenario is an instance of a use-case, showing the interactions between involved objects. One or more scenarios may be used to describe the use-case realization.

The names of the objects that take part the scenarios are in sans serif fonts. Usually the name of the object is tha same as that of the instantiated class.

2.2.1 Actors

Two external actors, the observer and the manager, interact with the system through two different use-cases, which means that they use the system in two different ways and configurations.

An actor operates on the system through one or more interfaces.

An interface is a collection of operation that defines the role of the actor. The interface is *realized* by a class that implements such operations. Operations are activated by an event produced by the actor. One of such events may be the activation of a control in a GUI or the input of a text command from the console.

2.2.2 Command Input

observer interacts with the *Input Command* use-case. Use-cases that represents command executions are modeled as `<<exstension>>` of this use-case. Two examples are *Select a source* and *Observation*.

This section describes the interactions occuring when any command is input from the current input stream of the **text interface**.

Commands are character strings that have the following format:

```
command = param1[,] param2[,] ...[,] paramN
```

The first word is the command identifier (Keyword), which is separated from the command parameters by an = sign. Parameters are separated by commas or blanks. Each keyword is associated with an operation and stored in the vocabulary of the system. Keywords are added to the vocabulary by the classes that realize part or the whole of an interface.

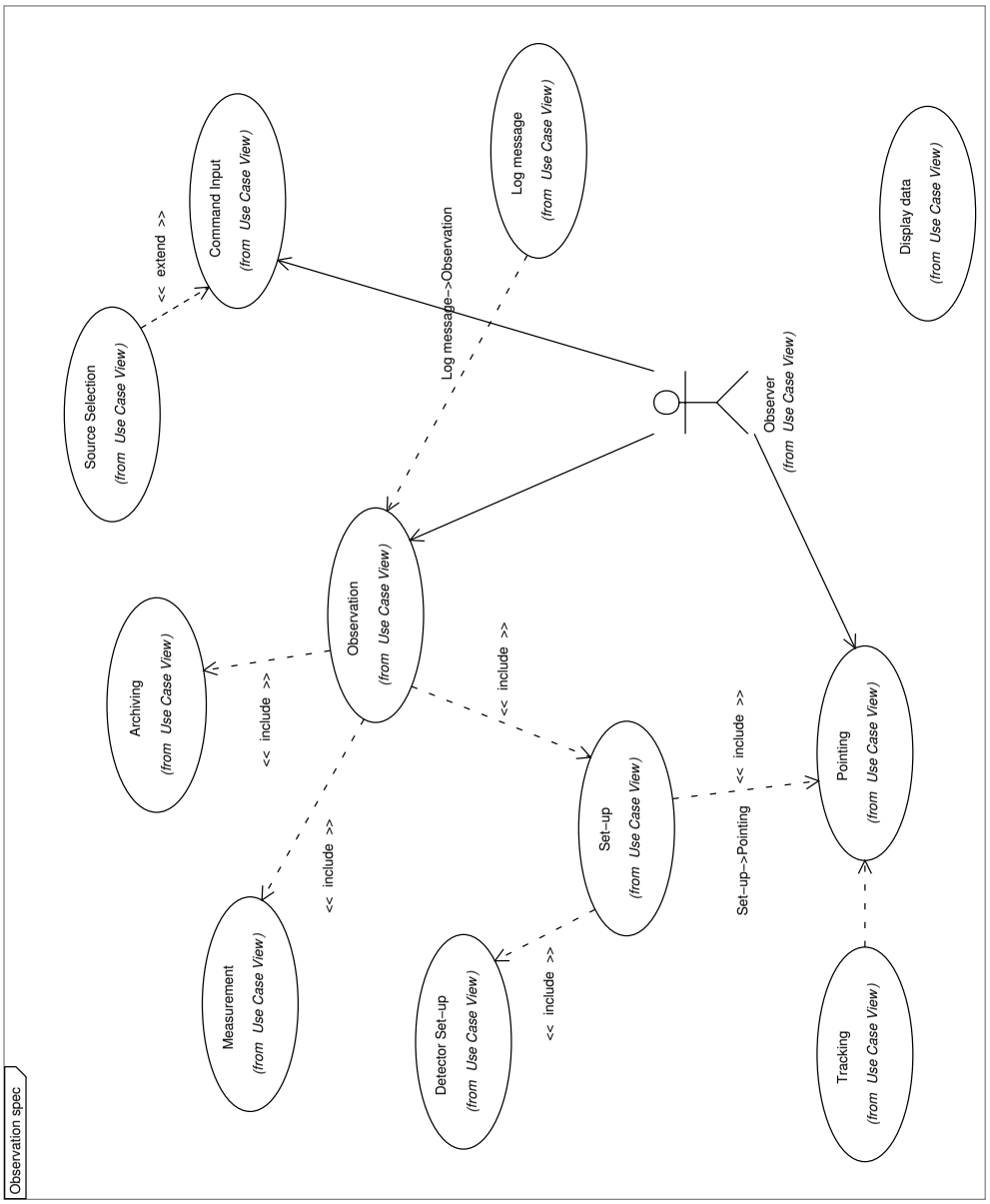


Figure 2.2: Specification of the Observation use cases.

The command interpretation is done in two steps. First the keyword is separated from the parameter string by the `CommandInterpreter`. The keyword is then used to search for the command in the `Vocabulary`. The parameter string is passed to tha called function which is aware of their format and meaning.

Pre-condition None

Main Event Flow

- observer issues a command on the keyboard
- ci reads the command
- ci parses the command
- ci searches the command in voc.
- voc sends the run message to the command object. The parameter string is passes as argument of the message.

ExceptionalEvent Flow

- ci searches the command in voc.
- voc does not find the keyword.
- voc issue a warning message and returns an error.

Constraints None.

2.2.3 Use-case: Observation

The *Observation* use-case `<<include>>` three smaller use-cases, *Set-up*, *Measurement* and *Archiving*.

The *Observation* is the core use-case of the system. The following objects collaborates to its realization:

- det of class `Detector`.
- tel of class `Telescope`.
- im of class `Image`.
- ar of class `Archive`.
- ci of class `CommandInterpreter`.
- w3oh of class `Source`.

Pre-condition The source to be observed is selected.

The use-case starts when observer selects the *on-source* observing mode.

det performs its set-up operation, which includes the tel pointing to the observed source position. det starts the integration and saves the result in im. det sends im to archive.

The use-case ends when im is written into the output file.

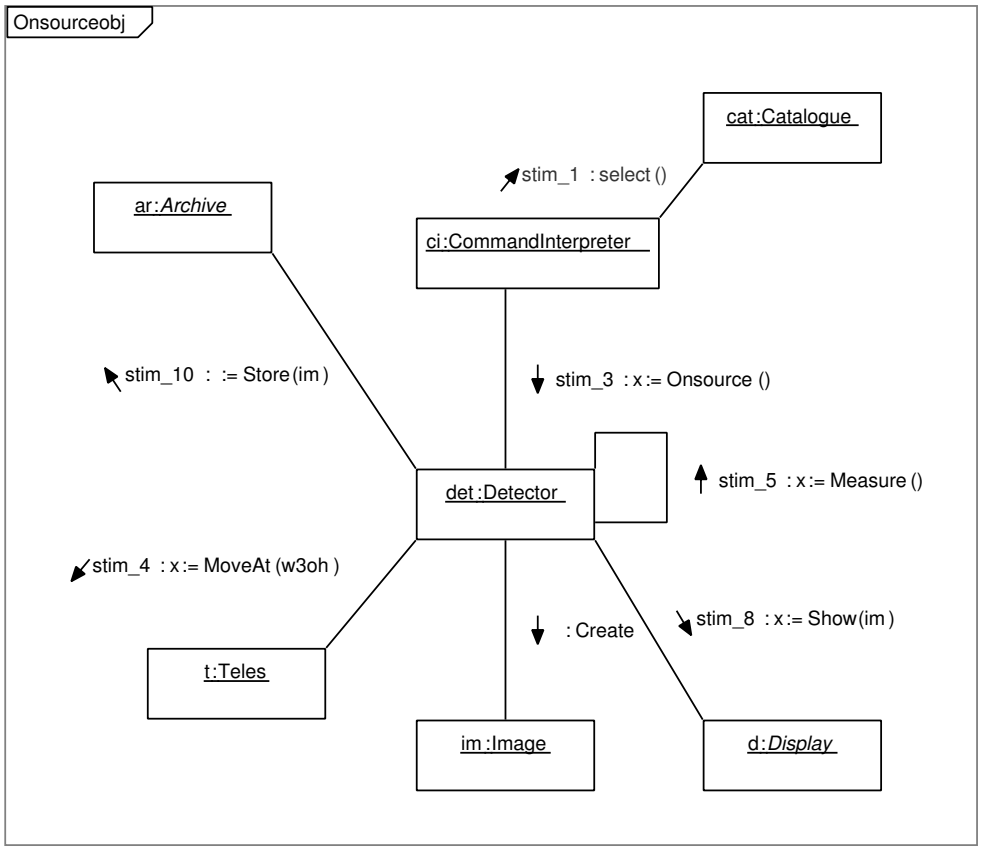


Figure 2.3: Analysis: object diagram for the observe scenario of the *observation* use case.

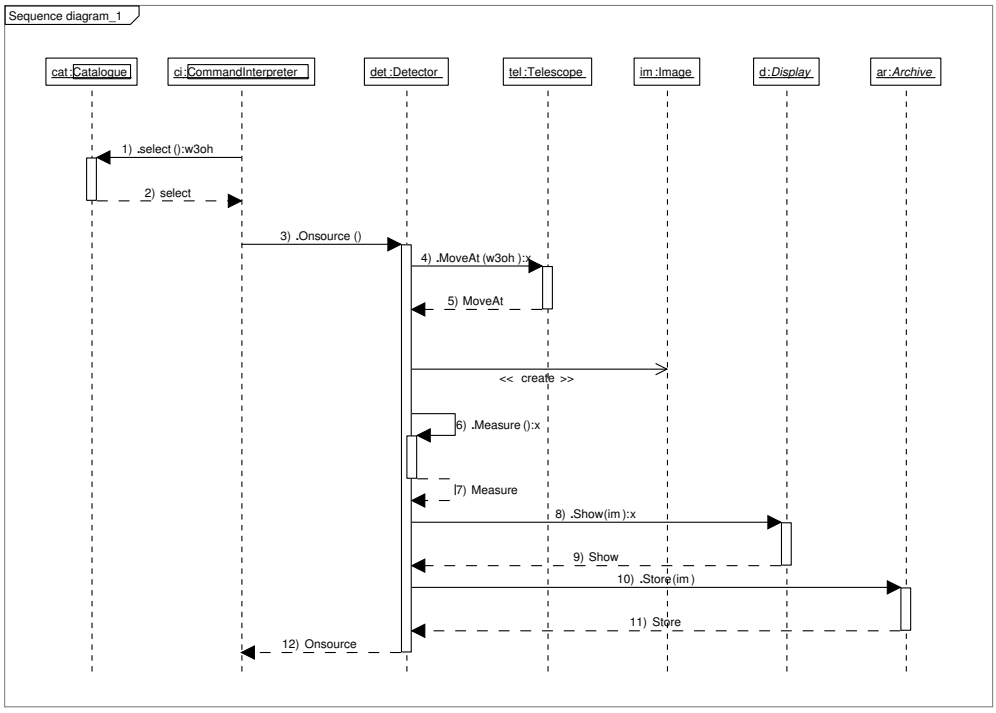


Figure 2.4: Analysis: sequence diagram for the observe scenario of the *observation* use case.

Scenario: Source Observation

Fig.s 2.3 and 2.4 show the scenario for the observation of a generic source in the *on-source* mode.

a observer issues the `On-source` command.

b ci identifies the command and send the `Onsource` message to det.

c `det` creates im.

d det starts the *set-up* use-case (`<<include>> set-up`)

e det starts the integration (`<<include>> measurement`).

f At the end of the integration time det stores the image data and its description in im.

g det shows im on the `display`.

h det asks archive to store im.

i archive stores im in the output file (`<<include>> out`).

Fig. 1.3 shows the activity diagram for the observation of a generic source in the *on-source* mode.

2.2.4 Use-case: Select a source

A source can be selected from a list or catalogue. The catalogue must be read before the source can be selected. All catalogues in the system are searched and a list of catalogues is compiled.

The *Select a source* use-case represents the management of some source list or catalogue by the Observer. The `<<extend>>` link toward *Input Command* indicates that the source to be observed may be selected from a catalogue.

Source selection

Fig. 2.5 shows the scenario for the selection of the source to be observed from the source catalogue.

Precondition A list of catalogues has been compiled.

Main Event Flow

a Observer selects the source catalogue cat.

b ci shows the list of sources in the catalogue.

c Observer selects w3oh.

d ci creates the object commanded of class `Source`.

The use-case terminate when the commanded object is created.

2.2.5 Use-case: Set-up

The system set-up consists in (fig. 2.6):

- Configuring det
- Selecting the receiver rec from the receiver list and configuring rec
- Asking tel to track the current source.

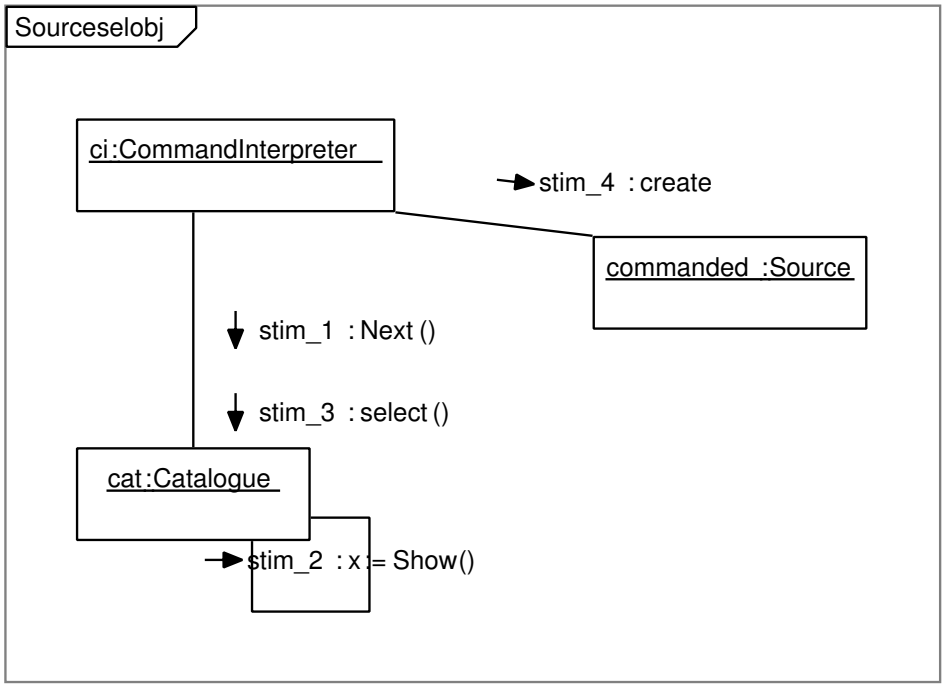
Pre-condition The configuration parameters are defined for source, receiver and detector.

Main Event Flow

- det checks the paramenters for consistency.
- Based on the selected frequency, det selects and set-up the rec receiver.
- det asks tel for source tracking (`<<include>> tracking`).

Exceptional Event Flow

- det finds inconsistent paramenters.
- det send an error message to the logbook, indicating the fault parameter.



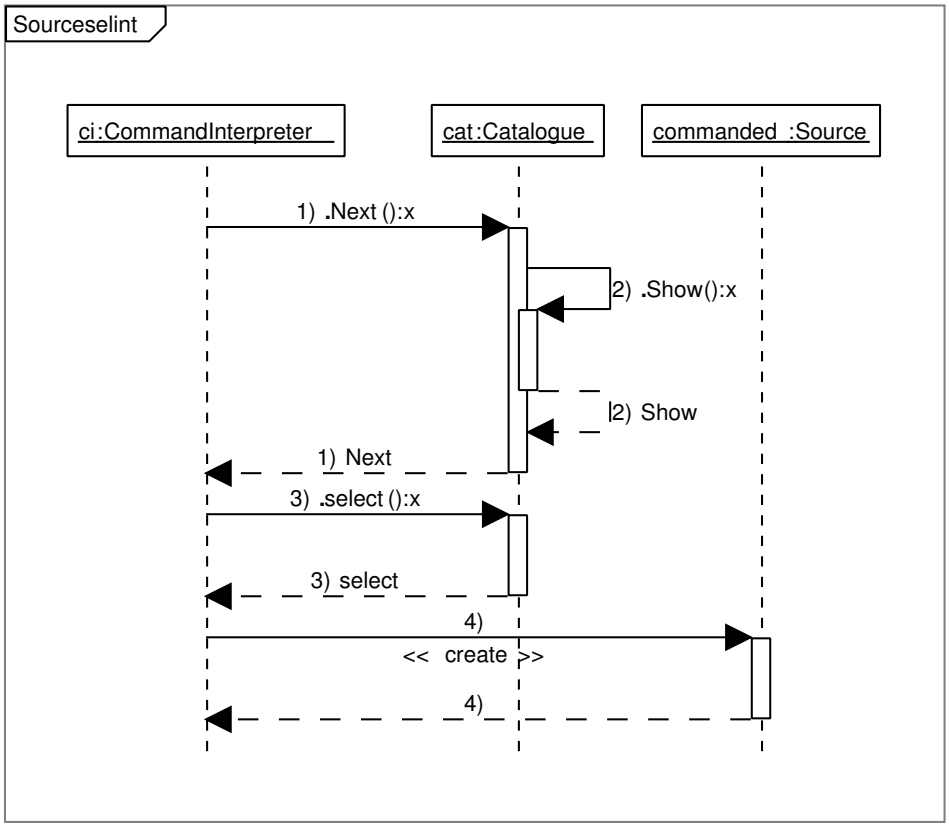


Figure 2.5: Analysis: object (top) and sequence (bottom) diagrams for the *select a source* use case.

Exceptional Event Flow

- tel finds that the source is below the horizon.
- tel reports the *unobservable* condition to det
- det stops the observation and send an error message to the logbook, indicating the fault parameter.

Postcondition The source is tracked.
The use-case terminate when tel reaches the requested position.

2.2.6 Use-case: Pointing

The Observer uses the system to point the telescope in some specified direction. The pointing model corrections are applied. Optionally *Tracking* can be started and stopped, meaning that the position is countinuosly updated in time.

tel moves the antenna to the commanded source position. Optionally the tracking mode can be started.

Pre-condition The commanded source is specified. Equatorial coordinates are assumed. A position threshold for *onsource* condition is set (depends on the resolving power).

Main Event Flow

- tel moves to the commanded position.
- tel set the move status to *slewing*.
- tel waits until the commanded position is reached.
- tel set the status to *onsource*.

Exceptional Event Flow

- If tracking is requested the tracking activity is started.
- the move status is set to *tracking*

Exceptional Event Flow: position not observable

- The position is below the horizon. acu does not move and the *not observable* status is returned.
- the move status is set to *idle*

2.2.7 Use-case: Tracking

Tracking is activated to follow a source moving in the telescope reference frame.

Pre-condition This is an cycle. The cycle is executed while the *tracking* status is true.

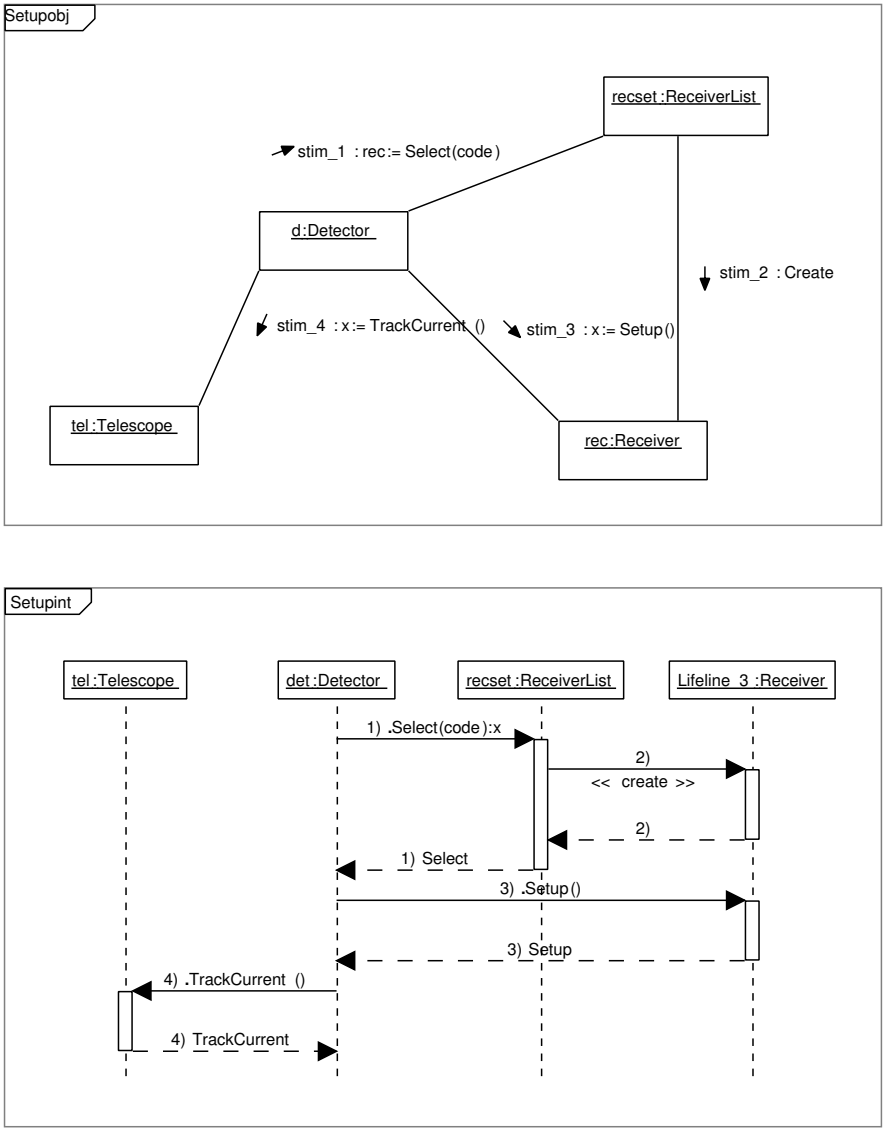


Figure 2.6: Analysis: object (top) and sequence (bottom) diagrams for the setup use case.

Main Event Flow

- tel compute the new commanded position.
- tel moves the antenna to the commanded position.

2.2.8 Use-case: Measurement

While the telescope is *onsource*, det goes on measuring, until the integration time is exausted. At the end of the measurement the image data are read from the detector hardware. If Observer issues a `break` command the observation is aborted and the integrated data are lost.

The activity connected to this use-case involves the class `Detector` so it is best represented by a state diagram (Fig. 2.7) for this class.

Pre-condition tel is *onsource*.

Main Event Flow

- det set the progress time to the integration time.
- det starts the measurement.
- Every second of time: While tel is *onsource* and the progress time is greater than 0, the progress time is decreased by one
- det reads the measured data from the detector unit.

Exceptional Event Flow Telescope off source.

- tel is *off*source. The integration is suspendend and the progress time is not decreased.
- a time-out counter is started.
- If tel goes back to *onsource* the time-out counter is reset and the integration is resumed.
- if the time-out counter reaches the time-out limit the integration is stopped and an errore message is sent to the log system. No data is saved.

Exceptional Event Flow Observer break.

- Observer issues the `break` command.
- det stops the integration.
- im is cleared.

2.2.9 Use-case: Output

The observed image is stored in ar in the specific format. The image is sent to a graphical display for monitoring purposes. The collaboration and sequence diagram for this use-case are shown in fig. 2.8.

Pre-condition The observation parameters are already defined.

Main Event Flow

- det copies the data set read by the physical detector (hardware) in im.
- det completes the description section of im.
- det sends im to the graphical display.
- det sends im to ar.
- ar converts im to its format and writes it into the output file.

Exceptional Event Flow

2.3 Maintenance

The *Maintenance* use-case is drawn simply to show that there are two possible way of use of the system, but it is beyond the scopes of this paper.

2.3.1 Use-case: System Monitoring

The system monitoring can be initiated by any unit in the sistem (log activity) or by the Observer to have some status information.

Pre-condition The observation parameters are already defined.

Main Event Flow Status request.
Observer initiates the use-case.

- Observer requests the telescope status.
- ci asks tel for its status of motion.
- tel sends the current status of motion to the log system. (log message)

Main Event Flow Log message.
Any object can initiate the use-case.

- The logging object sends a message to the log system.
- the log system add the system time information.
- The log system send the message to the display and write to the log file.

Exceptional Event Flow

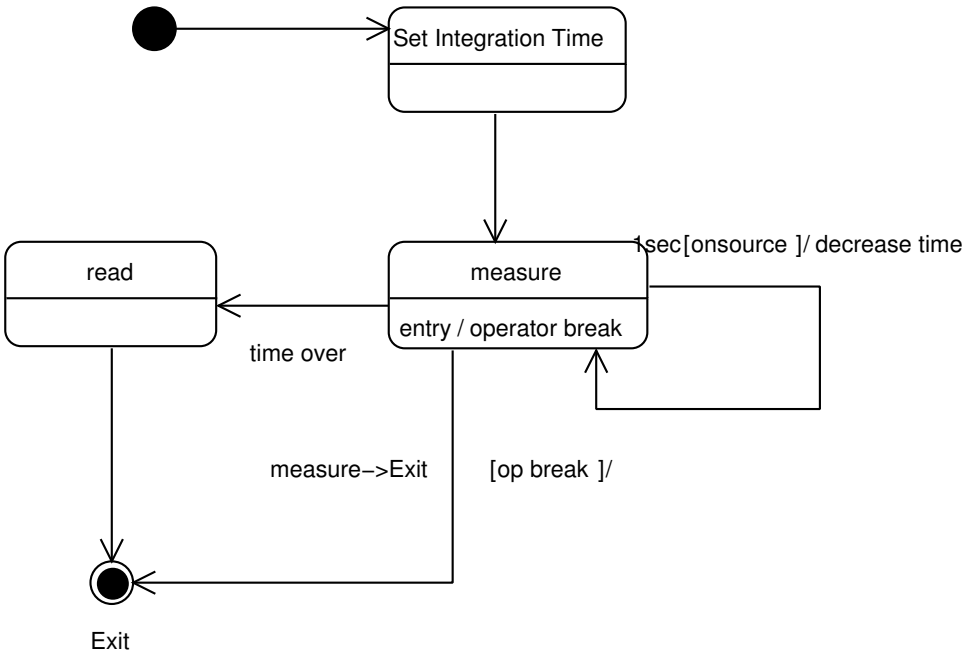


Figure 2.7: Analysis: state diagram of the Detector class describing the measurement use case.

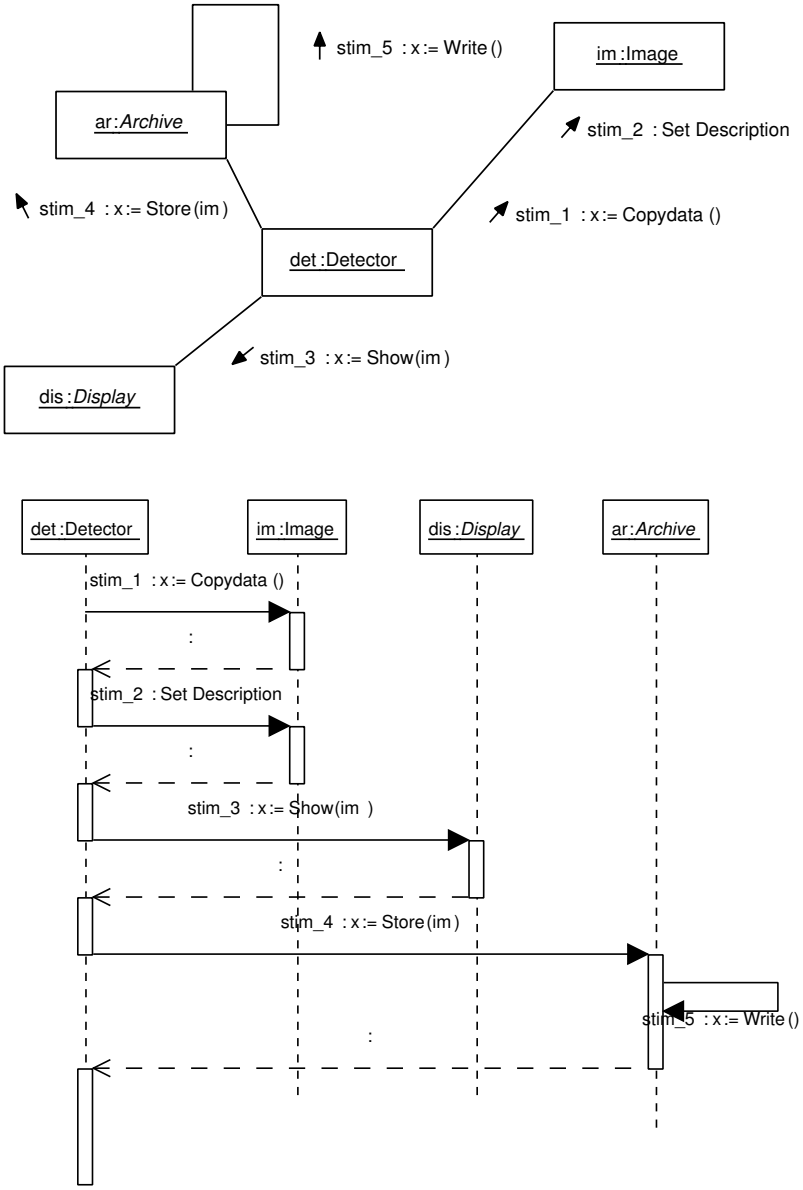


Figure 2.8: Analysis: Collaboration (top) and sequence (bottom) diagrams for the output use case.

Bibliography

[1] G. Booch, J. Rumbaugh and I. Jacobson, 1999, *The Unified Modeling Language User Guide*, Addison Wesley Longman, Inc., Reading, MA, USA

[2] R. Pooley and P. Stevens, 1999, *Using UML, Software Engineering with Objects and Components*, Addison Wesley Longman Ltd., Harlow, Essex, UK

[3] Rumbaugh J. and Blama M., 1995, *Modelli a Progetti Object–Oriented* Prentice Hall International–Jackson Libri eds., Milano, Italia

[4] Booch G., 1994, *Object-Oriented Analysis and Design with Applications*, Benjamin/Cumming Publishing Co., Redwood City, Ca, USA