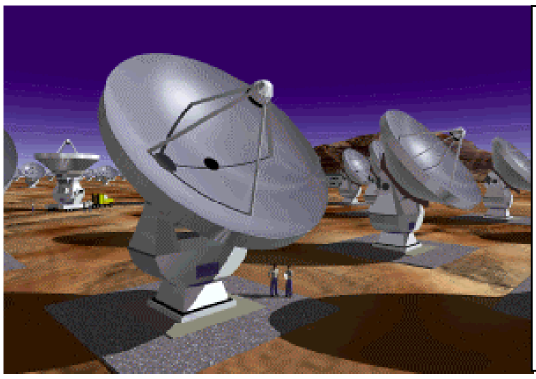


Atacama Large Millimeter Array	ALMA-SW-NNNN Revision: 0 2001-08-02

Initial Software Analysis

Software analysis

P. Grosbol, J. Schwarz (jschwarz@eso.org), R. Warmels
European Southern Observatory
G. Harris
National Radio Astronomy Observatory
D. Muders
Max-Planck-Institut für Radioastronomie
R. Lucas (SSR Consultant)
Institut de RadioAstronomie Millimétriaue

Keywords: ALMA, software

Author Signature:	Date:
Approved by: Institute:	Signature: Date:
Released by: Institute:	Signature: Date:

Change Record

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
0	2001-02-22	J. Schwarz et al.	All
	For comments at Grenoble SSR meeting, 1-2 March, 2001		
0.1	2001-04-12	PG & JS	Many
	Incorporate SSR comments		
0.2	2001-05-18	J. Schwarz	
	For circulation to ALMA s/w workers & SSR for comments		
0.3	2001-08-02	J. Schwarz	All; document reorganized
	Version for review		

Table of Contents

Table of Figures

1 Analysis: Purpose & Content

1.1 Purpose

"The Unified Software Development Process" (Jacobson, Booch & Rumbaugh, 1999) defines software analysis as:

> "A core workflow whose primary purpose is to analyze the requirements as described in requirements capture by refining and structuring them. The purpose of doing this is (1) to achieve a more precise understanding of the requirements, and (2) to achieve a description of the requirements that is easy to maintain and that helps us give structure to the system as a whole—including its architecture."

1.2 Analysis: Method & History

The process of software analysis is shown in terms of the requirements it starts with and the intermediate and final documents that it produces in the following figure.

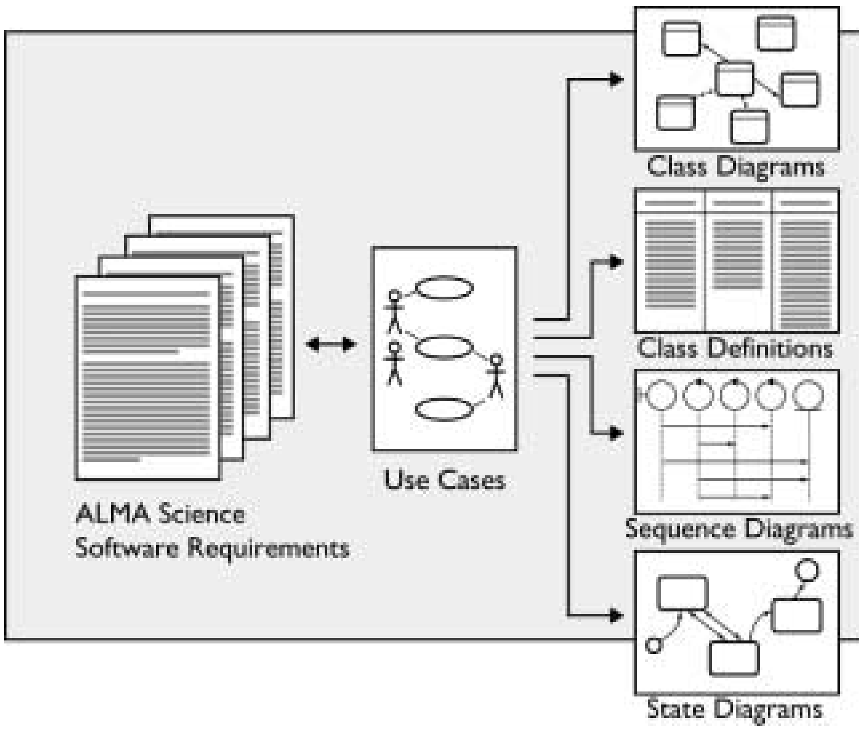


Figure 1-1: The Software Analysis Process

The first step in the analysis was to examine the requirements and Use Cases produced by the SSR and UC groups ("ALMA Software Science Requirements and Use Cases", Document No. ***ALMA-SW-0011***) to identify analysis classes and their responsibilities.

The derived initial list of classes was then grouped according to the main services (Proposal/Program Preparation, Scheduling, Executing, Imaging & Archiving; ***see Chapter 2, Analysis Classes and Packages***). Initial class hierarchies (mainly for the observing objects; similar to the hierarchy shown in the above report) were identified.

The next step involved translating the most important Use Cases into UML Sequence Diagrams. We decided to merge and rearrange some of the SSR Use Cases in order to represent the major functionalities of the ALMA software system. Since Sequence Diagrams show an explicit sequence of actions, we chose the "Observe Single Field" Use Case as an example of the execution of a typical Scheduling Block. We thus arrived at the following list of initial Sequence Diagrams (***see Chapter 2, Diagrams & Descriptions***):

- Create & Submit Observing Proposal
- Create & Submit Observing Program and Scheduling Blocks
- Schedule SB
- Dispatch SB (from a new Use Case, split off from the original Schedule SB)
- Execute SB
- Observe Single Field
- Process Data

The Sequence Diagrams are a representation of the steps in the basic course of the Use Cases along a time line. The actions specified in the basic course are translated into messages between instantiations of analysis classes. During the process of creating the Sequence Diagrams, additional necessary Analysis Classes were identified. It became clear that an overall Executive process was needed to manage the various services (Dispatcher, Scheduler, Sequencer, Subarray Allocator, Pipeline, Error Monitor) that are needed to operate the observatory, so an additional Use Case, "Operate ALMA System," and the corresponding sequence diagram were generated. Similarly, a sequence diagram for the SSR's "ObservePointingCalibration" Use Case was developed, mostly to show features that were obscured in "ObserveSingleField". We then identified additional (mainly administrative) software services that need to be present to operate the ALMA system. Use Cases and Sequence Diagrams for "Manage ALMA Facility" and "Administer Observing Programs" were developed (***see Section 3.3, Additional Sequence Diagrams***).

In addition, we found that significant implementation assumptions had made their way into the original Use Cases, and that these were making it difficult for us to outline the features of a system that would be adaptable to the requirements that would inevitably change (and that in fact are currently changing) as the ALMA project matured. Accordingly, we rewrote the Schedule SB, Execute SB and Dispatch SB Use Cases so that they would better reflect the spirit of the requirements. These revised Use Cases will be found in ***Section 9, Appendix***.

State / Activity diagrams for the major entity classes (Observing Project, Scheduling Block) were developed to show the state changes as the classes / objects are passed through the ALMA system during the proposal, approval, observation and data reduction phases (***see Section 2.2, State Diagrams***).

The Sequence Diagrams were then used to identify relations between classes. These relations are shown in Class Diagrams (***see Section 2.26, Class Diagrams/Hierarchies***), which depict the dependencies of classes and the class hierarchies. We invented a number of superclasses to simplify the system and group common classes.

To arrive at a modular system that allows distributing the future analysis, design and prototyping work, the classes were grouped into Class Packages. The splitting was done such that the number of interfaces between packages was minimized (***see Chapter 2, Analysis***

Classes and Packages). This work is not complete.

Next we considered possible error conditions and handling as well as security on the level of the current analysis software system. It turned out that there will be many layers of error conditions that need to be handled differently (*e.g.* how far errors or warnings will be passed up, logging and/or control activities). A matrix of possible high-level error conditions and subsequent actions was created (***see Chapter 4, Error Conditions & Handling Matrix***).

1.3 ALMA Use Case Roadmap

The ALMA Use Case Roadmap shows the highest-level Use Cases that we have considered in the software analysis. The first two, Manage ALMA Facility and Administer Observing Programs, represent the indispensable administrative tasks needed to keep the observatory running and the needs of the observers (namely, to have their programs reviewed and executed) satisfied.

The Operate ALMA System Use Case covers the day-to-day operation of the ALMA array and its associated computing infrastructure. Finally, the Observe With ALMA Use Case gives a proposer's/observer's/archive researcher's view of the system.

1.3.1 Actors

The stick figures shown on the borders of the Use Case diagrams are known as Actors. Although they are anthropomorphic, Actors represent any external entities with which the software system must interact, the most notable example of which is, of course, the ALMA hardware.

The UML definition of an Actor is:

> A coherent set of roles that users of use cases play when interacting with the use case.

One consequence of this is that the same person may appear as two or more different actors, when he/she plays two or more roles. The same person, for example, may act as proposer, observer, Staff Astronomer, Scientific (Archive) User and Reviewer.

The following table describes the Actors we have defined.

Proposer	A scientist applying for observing time on the ALMA facility. This person has a deep understanding of the physical problem to be investigated through the proposed observations but not necessarily of the details of aperture synthesis and the ALMA instrument. The main objective of this actor is to specify an observing proposal for ALMA
Referee	May be either a scientist who can judge the scientific importance of observing proposals or ALMA staff who can assess their technical feasibility. The actor reviews a set of proposals and provides an evaluation that can be used for the final ranking of all proposals.
Observer	The person responsible for the detailed specification of an accepted observing program, *i.e.*, the full specification of Scheduling Blocks, their dependencies and reduction requirements for all data obtained. The person in the Observer role may well be identical to the Proposer but is not required to be.
Scientific User	The end user of data from ALMA. The User obtains the data either as member of the team proposing the observations for which they were acquired, or as a researcher who has retrieved a useful data set from Science Archive.
Program Administrator	The role is generally responsible for all tasks related the administration and processing of observing programs. This includes general user support, management of the review process, overview of the status of observing programs, check of data quality, and preparation of data deliveries. As such, this role may later be detailed into several more specific actors.
ALMA Operator	Supervises the operations of ALMA. The Operator is responsible for the smooth execution of Scheduling Blocks and oversees the performance of all components.
ALMA Administrator	Responsible for administrational tasks related to the general ALMA facility. Such tasks may include making the long-term schedule for the array configuration, arranging availability of staff and visits of observers, and ensuring that maintenance action are planned and executed.
Technician	Responsible for maintenance of ALMA Hardware, using tools provided by the software system where necessary.
ALMA Hardware	Antenna, receiver, correlator and related communications and electronic hardware that is controlled by and from which data is acquired by the software system.

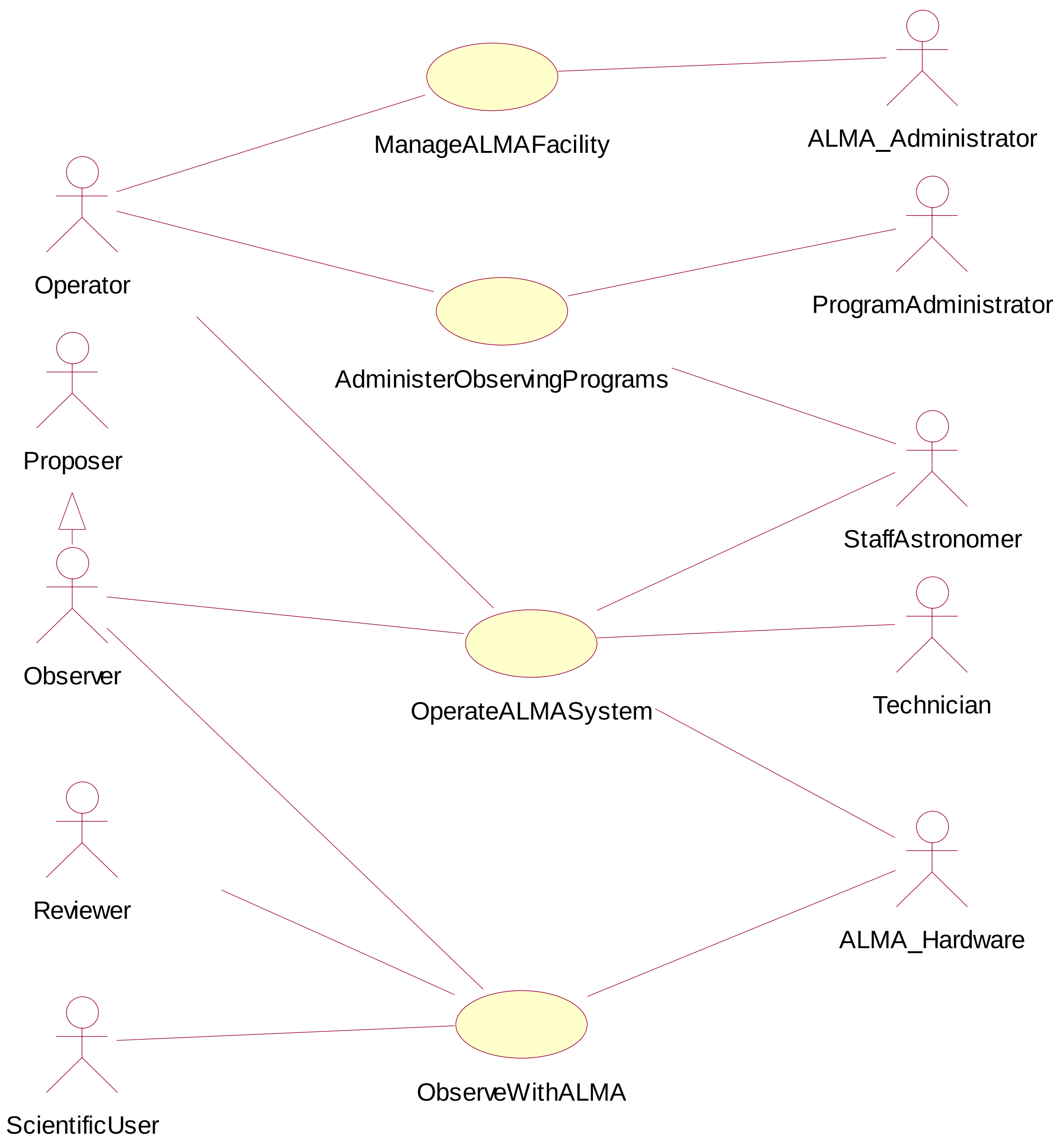


Figure 1-2: ALMA Use Case Roadmap

1.4 *Observe With ALMA Use Case*

Most of our attention has been on the details of the Observe With ALMA Use Case, whose lower-level Use Cases are shown in Figure 1-3. Each of these Use Cases appears either in the Science Software Requirements or in the Appendix to this document. Note that some “factoring” of the Use Cases has been done: ScheduleSB is used by DispatchSB (a situation somewhat analogous to the inclusion of one file in another), and Retrieve Archived Data *extends* Process Science Data in the sense that it represents an optional path through the latter Use Case. The Observe Single Field Use Case is a special case of ExecuteSB.

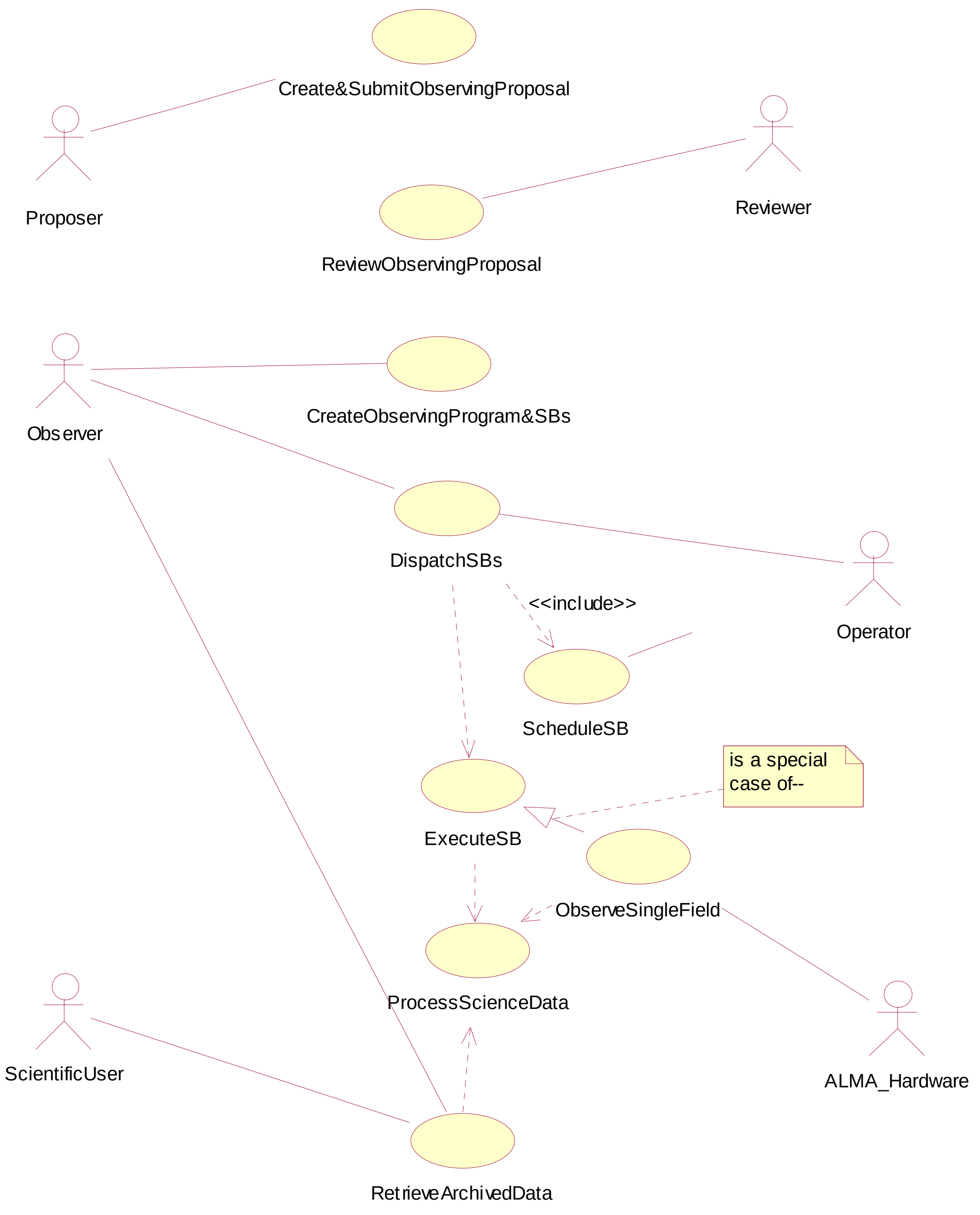


Figure 1-3: Observe With ALMA

2 Analysis Classes and Packages

This section lists and explains the purpose of the analysis classes used to satisfy the requirements from the SSR Use Cases and the additional classes identified during the analysis process. The list is organized according to the analysis class packages that we developed. The purpose of class packages is the grouping of classes such that the interfaces between packages are minimized. This modularizes the system and facilitates the distribution of subsequent detailed analysis, design and prototyping tasks among several software groups.

Package discussion needs to be rewritten.

2.1 Observing Tool Package

2.1.1 Observing Program Specification Package

Class Name	**Description**	**Responsibilities**
ObservingTool	Enables the user to prepare an Observing Proposal and to transform it into an Observing Programme	know Observing Modes know Observing Programmes know SB repository order Observing Proposals/Programme
ObserverUI	Provides the main GUI to the Observing Tool to prepare and Observing Proposal/Programme or display Programme results	display results forward input to Observing Tool

2.1.2 Observing Template Package

Class Name	Description	Responsibilities
FieldSpecification	Specifies the reference position and the geometric pattern that the antennas should follow during one observation.	know reference position know geometric pattern compute next point on path
NutatorSpecification	Specifies the nutator configuration, necessary to prevent data taking while the nutator is moving.	know nutator mode status
ObservingModeParameter	Is an abstract class for all Observing Mode parameters that are required for an Observing Mode	know range of all valid parameter input
ObservingModeTemplate	Defines the framework in which a user can insert scientific and technical parameters in order to specify a valid Observing Mode Programme.	know available Observing Modes check valid parameter input create an observing script
SBTemplate	Defines the framework in which a user can insert scientific and technical parameters in order to create an SB for a specific Observing Mode	know Observing Mode create an observing script
SpectralSpecification	Contains the spectral setup for one Observation according to the pattern described in the Field Specification	know Observing Mode know spectral specs
TemplateScript	Is an abstract class, which can produce a standard observing script from a set of information provided such as observing mode, source specifications, and antenna setup.	create an observing script

2.1.3 Simulator Package

Class Name	Description	Responsibilities
Basic Simulator	Estimates flux, noise and beam for an observation given a detailed setup, observing conditions and a simple source model.	compute flux map compute noise estimate compute beam

2.1.4 Correlator Package

Class Name	Description	Responsibilities

CorrelatorTool	Provides a mapping of science specifications to Correlator configurations.	know correlator configurations find configuration

2.2 Observing Program Administration Package

Class Name	Description	Responsibilities
OperatorUI	Is a user interface class that forwards operator command to the ProgramTool and displays information from it.	display Program information forward commands to ProgramTool
ProgramTool	Provides the business model for the high level administration of ObservingPrograms.	change status of programs or/ scheduling blocks generate reports write data package

2.3 Observing Program Refereeing Package

Class Name	Description	Responsibilities
RefereeUI	Is a user interface for reviewers of observing proposal and serves as a front end for the RefereeTool	display Proposal information forward referee rating and comments to RefereeTool
RefereeTool	Defines the referees access to a proposal and provides options for appending ratings and comments.	view proposal add rating and comments submit review package

2.4 Observing Project Package

Class Name	Description	Responsibilities
Observing Project	Contains the Observing Proposal and the associated main Observing Program.	know Observing Proposal know main Observing Program know Project Control
ObservingProposal	Contains all mandatory scientific and technical information on the basis of which a proposal can be evaluated to granting observing time by the OPC.	know science objectives know performance goals know PI
ControlBlock	Contains status information such as the time allocated to the Observing Project and the Observing Programs. It may include Breakpoint Conditions	know status information
ProjectControl	Is the ControlBlock for the Observing Project	know Observing Project status
ObsUnitControl	Is the ControlBlock for the Observing Unit	know ObservingUnit status
ObservingUnit	Is an abstract prototype class to build an Observing Object hierarchy. Observing	knows ObsUnitControl knows ImageScript

	Programs (branch nodes) and Scheduling Blocks (leaf nodes) are derived from the ObservingUnit.	
ObservingUnitSequence	Contains a sequence of Observing Units.	know observing units know flow control script
FlowControlScript	Defines programmatically the sequence in which the Observing Units of an Observing Unit Sequence are executed.	know flow control commands

2.5 Program Package

Class Name	Description	Responsibilities
ArchivingSpecification	Defines which scientific data need to be archived.	know archiving specs
BreakpointConditions	Defines the conditions that will halt future execution of SBs of the Observing Programme.	know conditions for suspending Programme
ObservingProgram	Contains all technical and scientific information required to execute an Observing Program. It consists of an Observing Unit Sequence.	know science objectives know performance goals know scheduling blocks know breakpoints know data processing scripts know program status know archiving specs

2.6 Scheduling Block Package

Class Name	Description	Responsibilities
PerformanceGoal	Defines the observing goals (noise level, SNR, or image dynamic range) that are intended to be achieved by the Observing Programme.	know goal(s)
SchedulingBlock	Defines a sequence of observing scans and is the smallest part of an Observing Programme that can be individually scheduled and calibrated.	know calibration requirements know required observing conditions know required configuration know observing instructions

2.7 SB Script Package

Class Name	Description	Responsibilities
ImageScript	Defines how the quicklook or final image is to be produced.	know reduction commands
ObservingScript	Is an abstract class for Observing Scripts which are association classes created from Script Templates and Observing Mode Parameters	know observing script commands know observing mode parameters

2.8 Command Package

Class Name	Description	Responsibilities
Command	Is an abstract class for all commands (e.g. observing, pipeline)	execute command
ObservingDescriptor	Contains the parameters that describe the array configuration during the execution of the Observation.	know array configuration during scan execution
PipelineCommand	Defines a reduction step in the image pipeline script	know commands execute command
ScanCommand	Defines a set of observations with a common goal and is the smallest unit which can be executed by an observing script.	know command execute command

2.9 ALMA Executive Package

Class Name	Description	Responsibilities
MasterUI	Is a graphical user interface through which the ALMA facility is controlled and monitored.	display system status forward control commands to Executive
Executive	Starts and supervises all operations processes such as sub-array allocation, scheduling and error monitoring.	control subsystems change observing mode
ErrorMonitor	Checks periodically the state of all major subsystems and notifies the Executive if any problems are found.	know state of subsystems monitor subsystems
ExecWatchdog	Waits for a signal and raises an alarm if none arrives within a specific time.	know time since reset reset timer
ObservatoryPolicy	Contains all rules that govern the observatory policy.	know submission rules know review rules know observing modes know guaranteed data quality

2.10 Resource Management Package

Class Name	Description	Responsibilities
SubarrayAllocator	Allocates resources associated to sub-arrays such as antennas and parts of the correlator.	know antennas and their status know correlator and its status allocate subarray
ArrayConf	Is a logical set of antennas and associated resources.	know antennas know correlator
Resource	Is an abstract base class for all ALMA resources such as antennas, correlator and computer systems.	know status know allocation state know resource name and type

2.11 Submission Package

Class Name	Description	Responsibilities
Authorization	Performs all security checks concerning the access to data related to observing programs.	generate security keys verify authority
Validator	Checks the correctness of observing proposals and programs.	verify observing proposal verify observing program

2.12 Scheduling Package

Class Name	Description	Responsibilities
Dispatcher	Forwards either single scheduling blocks or, in dynamic mode, groups of scheduling blocks to be executed.	know operations mode dispatch scheduling block
Scheduler	Determines the optimal order of a set of scheduling blocks that are ready to be executed.	rank scheduling blocks

2.13 Script Execution Package

Class Name	Description	Responsibilities
Sequencer	Interprets and executes observing scripts to control antennas, receivers, correlator and data processing.	execute observing script
Subarray	Defines a set of ALMA antennas including all resources associated to them and required to perform a set of observations.	know allocated antennas know allocated part of correlator perform calibration observation execute scan command
CalPipeline	Performs the processing of calibration data and provides new calibration results in near real-time.	reduce calibration data

2.14 Online Calibration Package

Class Name	Description	Responsibilities
CalSolver	Coordinates the execution of all necessary calibration observation.	know calibration required perform calibration observations
Calibration	Is an abstract base class for all calibrations of a subarray and provides generic methods to access the status of a particular calibration.	know validity know validity period signal expiration estimate time to perform calibration know when must be done (might be deferrable) acquire calibration data
ArrayCal	Indicates that this is a calibration that is performed for the array as a whole (*e.g.*, baseline, delay, pointing session & beam shape)	As for Calibration class
ProjectCal	Indicates that this is a calibration that is performed for a single project or Observing Unit (although it could be shared among different ones)	As for Calibration class
CalStatusMonitor	Checks the calibration status of a subarray.	estimate time for calibration
CalPipeline	Performs the data reduction required to derive the calibrations.	reduce calibration data
Configurator	Sets up receivers, correlator as requested	Set hardware configuration
AntennaConfig	Contains information characterizing an antenna	Know antenna location Know antenna electronics configuration
ReceiverConfig	Contains information characterizing a receiver	Know receiver band Know receiver tuning Tune receiver

2.15 Data Processing Service Package

Class Name	Description	Responsibilities
ImagingPipeline	Generates all standard science data products.	know data processing server reduce data
QuickLookPipeline	Performs a fast data reduction of recent observations and provides the results in near real time.	know data processing server reduce data display results

2.16 Supervised Image Pipeline Package

Class Name	Description	Responsibilities
PipelineUI	Is a user interface for off-line usage of the Imaging Pipeline	display pipeline status and results forward commands to the PipelineTool
PipelineTool	Accepts high-level requests for off-line reduction of science data and forwards the explicit processing tasks to a server.	know data reduction server reduce data set

2.17 Science Archive System Package

Class Name	Description	Responsibilities
ArchiveUI	Is a user interface for access to the Science Archive	display results and status forward requests to ArchiveTool
ArchiveTool	Provides high level tools for doing research on the Science Archive.	know Science Archive server query archive retrieve data

Class Name	Description	Responsibilities
InfoService	Is an abstract class for archives, catalogs, repositories, data bases, etc.	know the archives know repositories know catalogs

2.18 Archive Package

Class Name	Description	Responsibilities
Archive	Is an abstract base class for all archives containing science related data	know available archive and their status
CalArchive	Contains all results produced by the Calibration Pipeline	know calibrator data search for calibrator data
LogArchive	Contains all logging information of the ALMA array	know all logging data search for logging data
RawDataArchive	Provides access to all raw science data acquired by ALMA.	know raw science data search for raw science data
ScienceArchive	Provides access to all calibrated science data acquired by ALMA	know calibrated science data search for calibrated science data
ManualArchive	Contains all documentation explaining the characteristics and potential usage of ALMA	know available manuals search for manuals

2.19 Catalog Package

Class Name	Description	Responsibilities
CatalogSet	Is a base class for catalogs containing astronomical objects or spectral lines	know catalog server know available catalogs and their status
CalibratorCatalog	Provides catalog of sources suitable for pointing or phase calibration of the array.	know calibrator data search for calibrator data
LineCatalog	Provides access to all physical data of molecular and/or atomic spectral lines relevant for observations.	know line data search for line data
SourceCatalog	Provides access to a database of astronomical objects	know source properties search for source properties

2.20 Repository Package

Class Name	Description	Responsibilities
RepositorySet	Is an abstract base class for collections of objects from the Observing Object hierarchy.	know repository server know available repositories and their status
ObservingProjectCatalog	Provides access to all ALMA Observing Projects	know Observing Project search for Observing Project
ConfigurationRepository	Contains technical data related to the antenna configuration, correlator, receiver, pointing models, and baseline solutions	know configurations
PersonRepository	Contains all information about persons such as Programme PIs and Co-Is, reviewers.	know person information search for person info
SB-Repository	Contains all valid SBs.	know SB information search for SB
ScheduleDB	Contains all information related to schedules for equipment or operational staff.	know schedules search for schedules
MaintenanceDB	Contains all information on maintenance actions.	know maintenance actions search for maintenance actions
Bodega	Maintains all information on spare parts and supplies	know spare part and supply critical limits signal low supplies search for parts

2.21 System Administration and Management Package

Class Name	Description	Responsibilities

AdminUI	Provides a user interface to the general administration tool.	forward requests to administration tool display reports
AdminTool	Controls all general administrational tasks for the ALMA facility such as generation of reports, scheduling of maintenance, and long term planning of array configurations and staff availability.	generate status and performance report define maintenance schedule define array configuration schedule define staff schedule

2.22 Utility Package

Class Name	Description	Responsibilities
Person	Contains all relevant information about a person.	know name know contact information

Class Name	Description	Responsibilities
EnvironmentData	Provides access to all environment data relevant to the ALMA facility.	know wind speed know water vapor know temperature know phase stability
ObsCondition	Contains all parameters describing the observing conditions and obtained directly from observations of calibrator sources.	know pointing know focus know phase calibration know bandpass calibration

2.23 Internet Package

Class Name	Description	Responsibilities
WebPage	Contains general information on Web pages made available to users.	know author, name and version know availability know page content post page
EmailService	Provides access to e-mail through Internet with a specified level of security.	know security level send e-mail verify receipt of e-mail check incoming e-mail

2.24 Class lookup table

For convenience we list all analysis classes alphabetically. The class descriptions and class packages can be found on the associated page numbers.

2.25 Package Diagrams

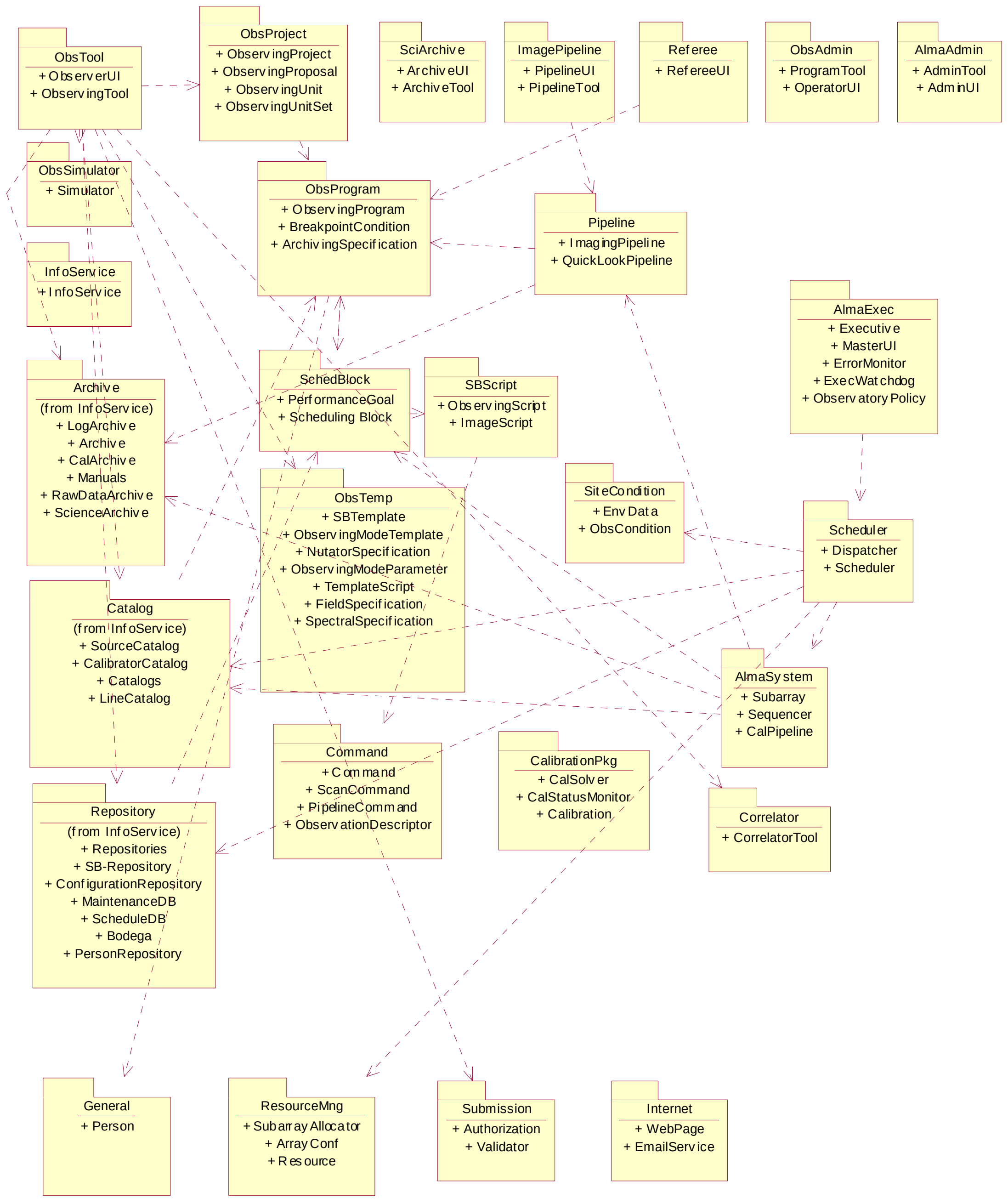


Figure 2-1: ALMA Logical Packages (Connections Illustrative Only)

2.26 Class Diagrams/Hierarchies

Most of the analysis classes presented are those that were necessary in order to express the Use Cases in terms of sequence diagrams. The methods for each class were similarly derived out of the need to serve other classes in order to complete the Sequence Diagrams. The class diagrams show each class, its methods and what other classes use these methods. They are more abstract than the sequence diagrams since they do not show when or how often each class makes use of a particular method in another class, but only that a connection is made.

It is clear that many classes have common functionalities, and these are best expressed by inheritance hierarchies, in which several classes inherit attributes and methods from a common parent. The class diagram for Calibrations is probably the best example in this document: Baseline, Gain, Beam shape and Bandpass calibrations are all variations on the theme of a general calibration class. It is to be expected that this commonality will lead to considerable savings in code development, since much of the software developed should be reusable in children of the same parent class.

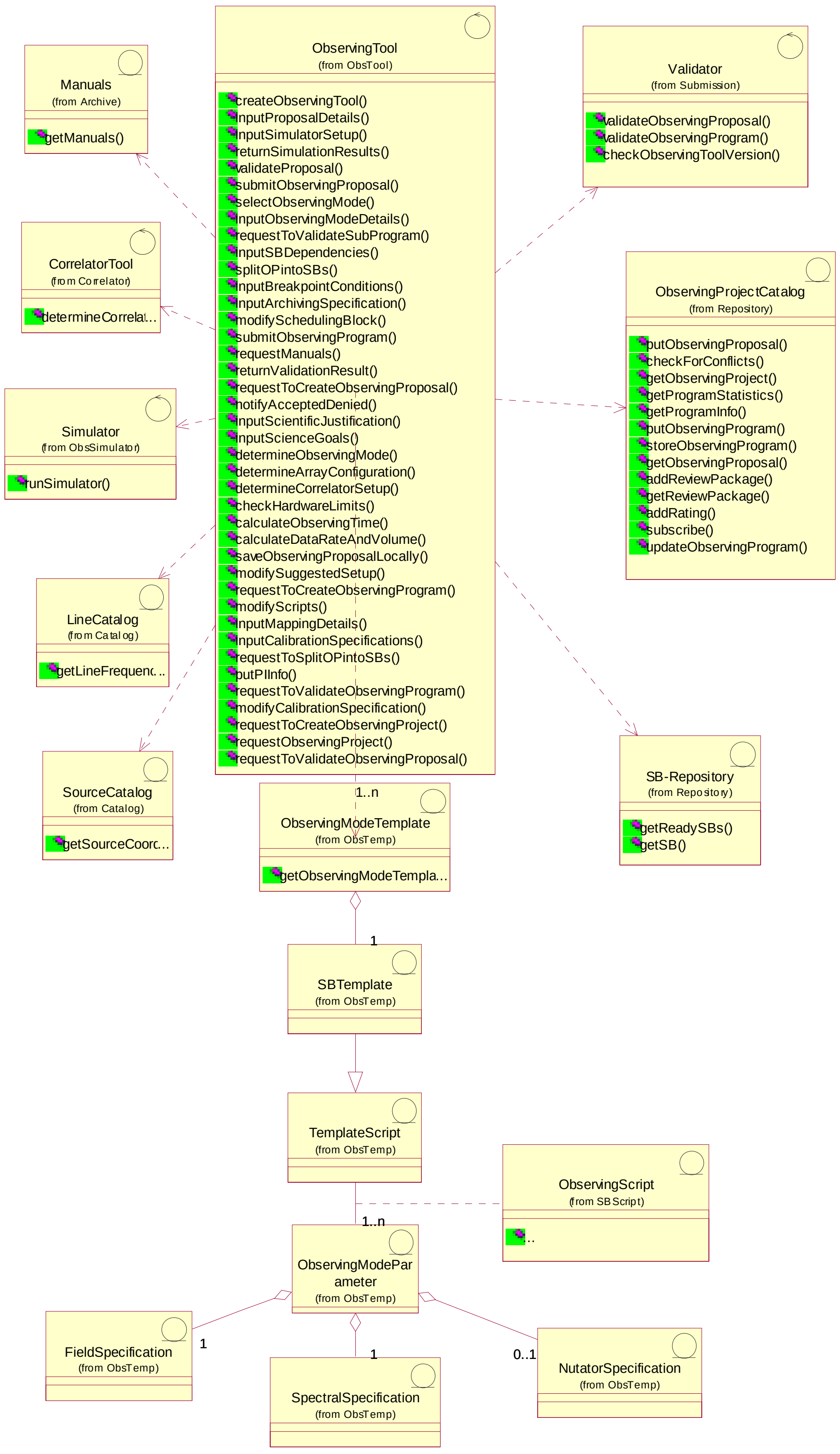


Figure 2-2: Observing Tool Class Diagram

The class diagram that shows the ObservingTool and the objects it references gives an early indication of the many methods, associated classes/objects, and InfoServices objects that will be needed to make this tool work. All these objects, in some form, will need to be exported to the Proposer at his/her home institution, or at least be reliably and efficiently accessible via a network connection to the

ALMA Science Operations Center or to one of the Regional Centers. The diagram also shows how ObservingModeTemplates, the bases for the generated SB scripts, are themselves built up out of lower level templates, which in turn are generated from a generic TemplateScript that takes uses an ObservingModeParameters object to "fill in the blanks". The ObservingModeParameters object itself is built from a FieldSpecification, an (optional) Nutator Specification, and a SpectralSpecification.

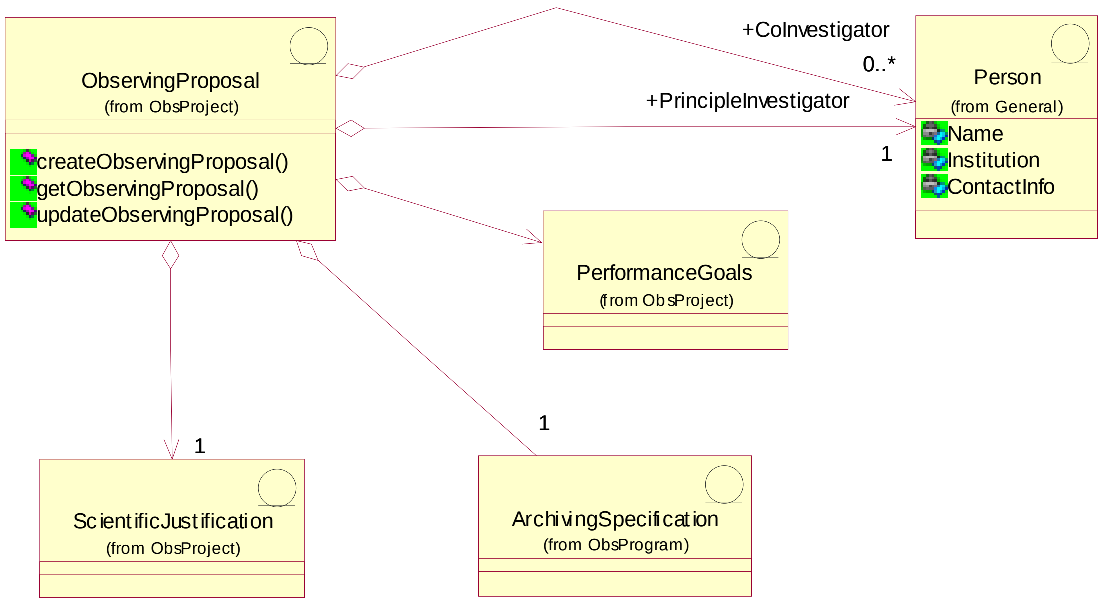


Figure 2-3: Observing Proposal Class Diagram

The class diagram for the ObservingProposal displays its various supporting entity classes: scientific justification, performance goals, archiving specification, and so on.

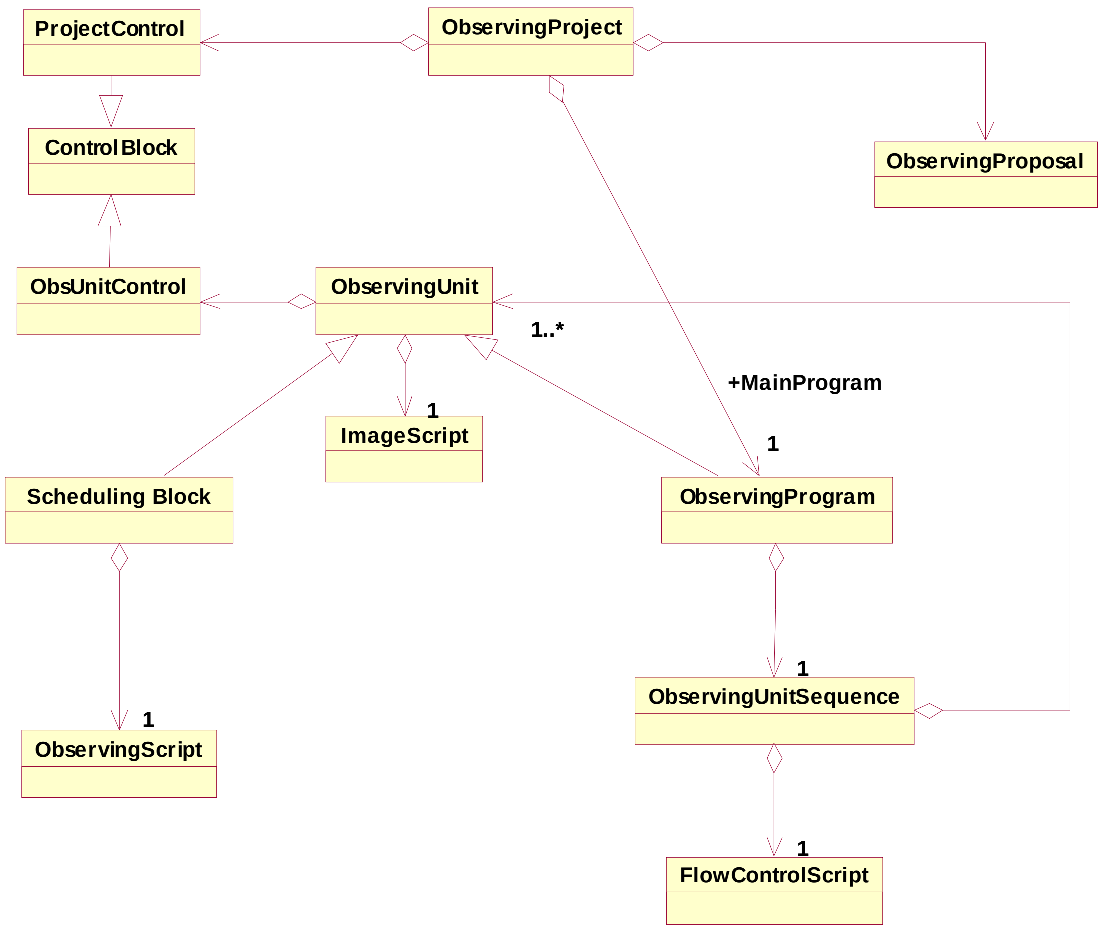


Figure 2-4: Observing Project Class Diagram

When an Observing Proposal is approved, its associated Observing Project must be filled in with the appropriate Phase II information. The class diagram for the Observing Project shows the way in which this information can be hierarchically organized into Observing Units, each of which contains an Observing Unit Sequence. An Observing Unit Sequence may recursively contain one or more Observing Units, but eventually this hierarchy must terminate in Scheduling Blocks, which are by definition non-recursive. An approved and ready-to-schedule Observing Project must contain at least one Scheduling Block.

Each Observing Unit Sequence has its own Flow Control Script, which establishes the order (if any) in which the component Observing Units are to be executed. The Observing Unit's Control Block includes status information (*e.g.*, how much observing time has been allocated to and used by this Observing Unit) and may include Breakpoint Conditions (indicating whether execution of this Observing Unit must be delayed pending a decision by the Project's PI). Associating such conditions with each level in the hierarchy allows the PI to temporarily halt a project at, for example, the start of an ACA or single-dish observation (each of which would naturally be characterized by a single Observing Unit), or at the more finely-grained level of a Scheduling Block.

The presence of a Control Block at all levels of the hierarchy also permits prioritization (presumably by the ALMA Time Allocation Committee, but if desired, also by the PI and by the ALMA Operations Staff) at any level of detail desired. If ALMA Operations Policy forbids articulated prioritization by PIs, reviewers or ALMA staff, then this capability will either not be implemented or can easily be disabled; the point to note is that the infrastructure to support it will exist.

Because the abstract nature of the above diagram can be somewhat hard to follow, we present an *object* diagram to illustrate how a particular kind of project might be handled. We hypothesize a mosaic of a galaxy that combines OTF, ACA and single-dish observations. The project hierarchy for this particular case might be as shown schematically in the following figure.

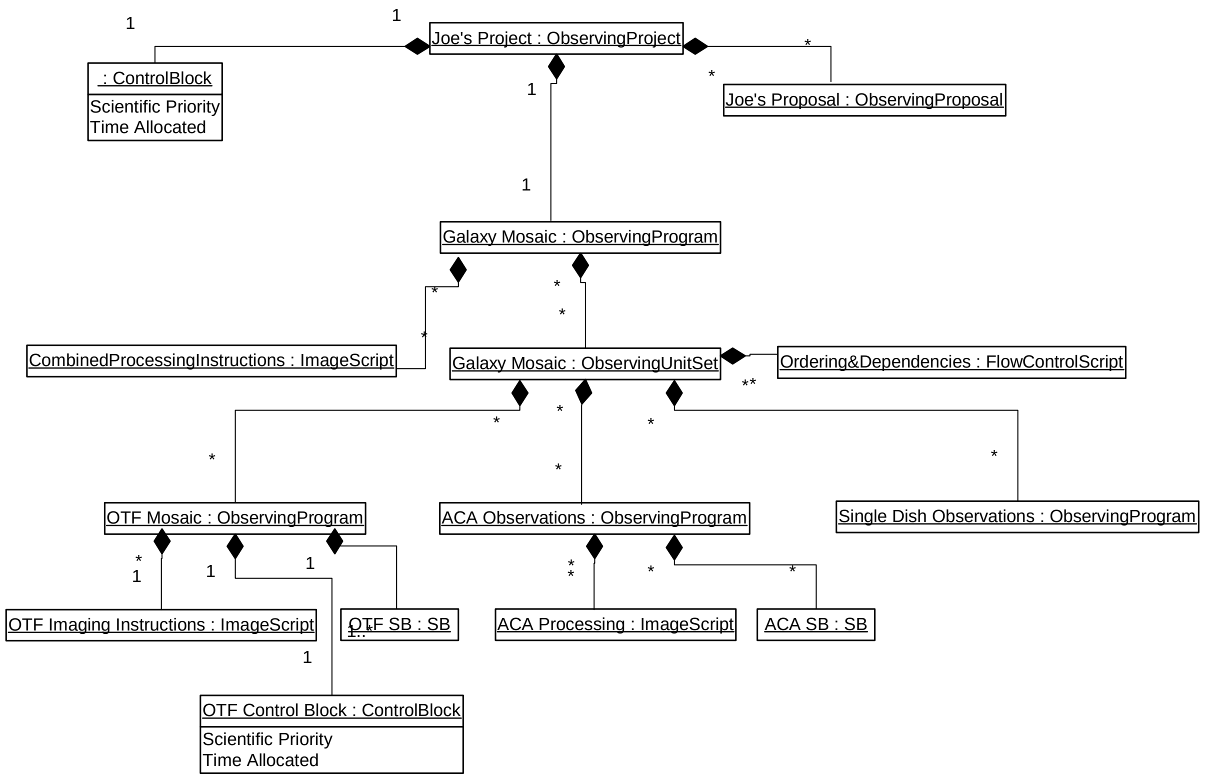

Joe's Project : ObservingProject
: ControlBlock
Scientific Priority
Time Allocated
Joe's Proposal : ObservingProposal
Galaxy Mosaic : ObservingProgram
CombinedProcessingInstructions : ImageScript
Galaxy Mosaic : ObservingUnitSet
Ordering&Dependencies : FlowControlScript
OTF Mosaic : ObservingProgram
ACA Observations : ObservingProgram
Single Dish Observations : ObservingProgram
OTF Imaging Instructions : ImageScript
OTF SB : SB
ACA Processing : ImageScript
ACA SB : SB
OTF Control Block : ControlBlock
Scientific Priority
Time Allocated

Figure 2-5: Object Diagram for Galaxy Mosaicing Project

Note that control blocks appear at the highest (project) level and also at the lower levels (shown here only for the OTF portion of the project). ImageScripts are shown attached to the OTF and ACA portions of the project, as well as to the higher-level “Galaxy Mosaic” Observing Program. A Flow Control Script takes care of any orderings and/or dependencies among the OTF, ACA and Single Dish Observing Programs. In order not to clutter the diagram with unnecessary details, we have left off various instances of the Image Scripts, Flow Control Scripts (which could, for example, be attached to a set of SBs) and Control Blocks.

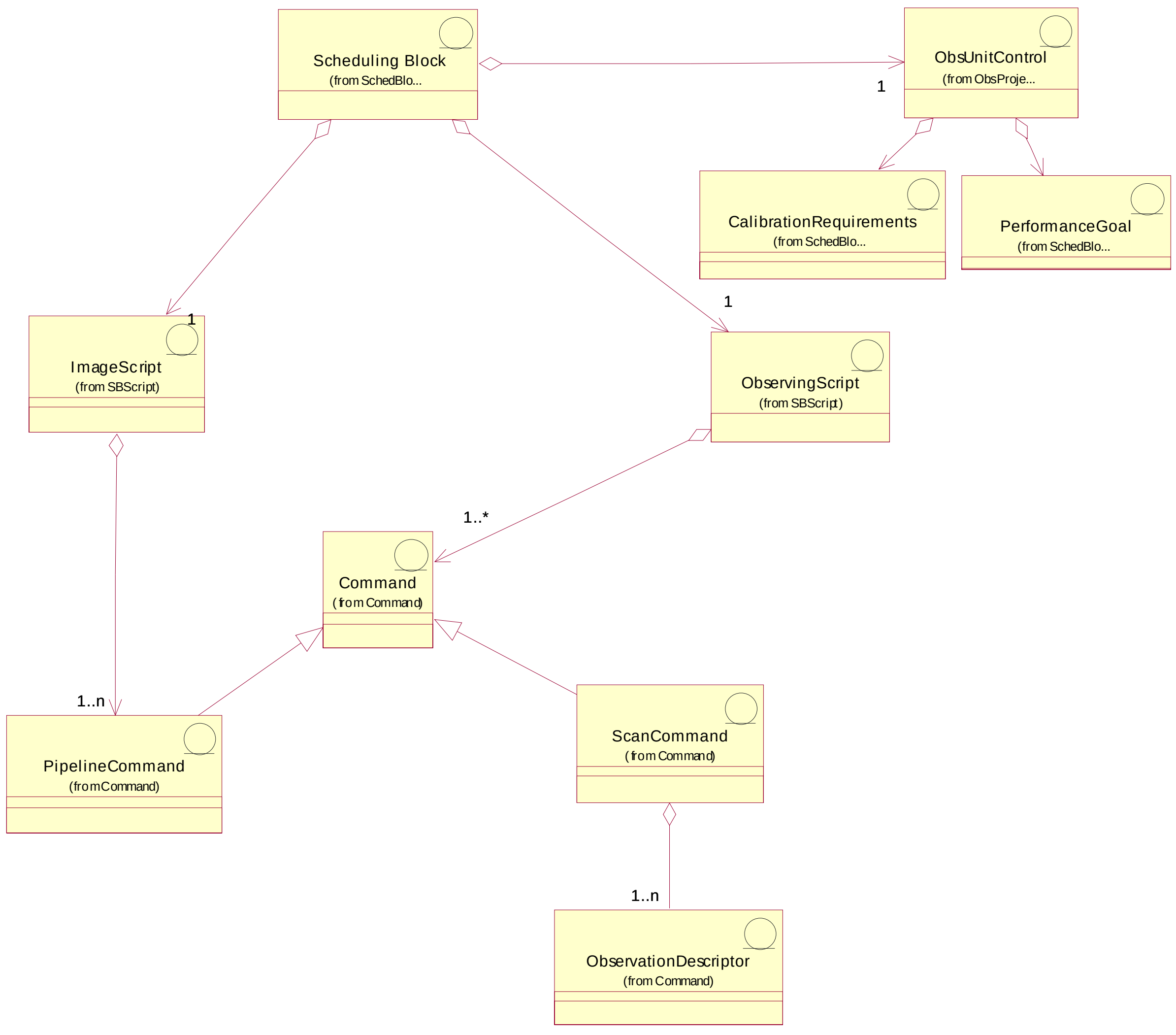


Figure 2-6: Scheduling Block Class Diagram

The Scheduling Block Class Diagram shows its component scripts (for observation and for imaging), as well as control parameters such as initial and final calibration requirements and performance goals.

The remaining class diagrams mostly show objects in a star configuration, with the principal object at the center, and all the objects it references and/or manages arranged around it. Thus the Sequencer, the Scheduler and the Imaging Pipeline are all control classes that make use primarily of other entity objects (although the Sequencer and Scheduler, for example, do have simple interactions with each other).

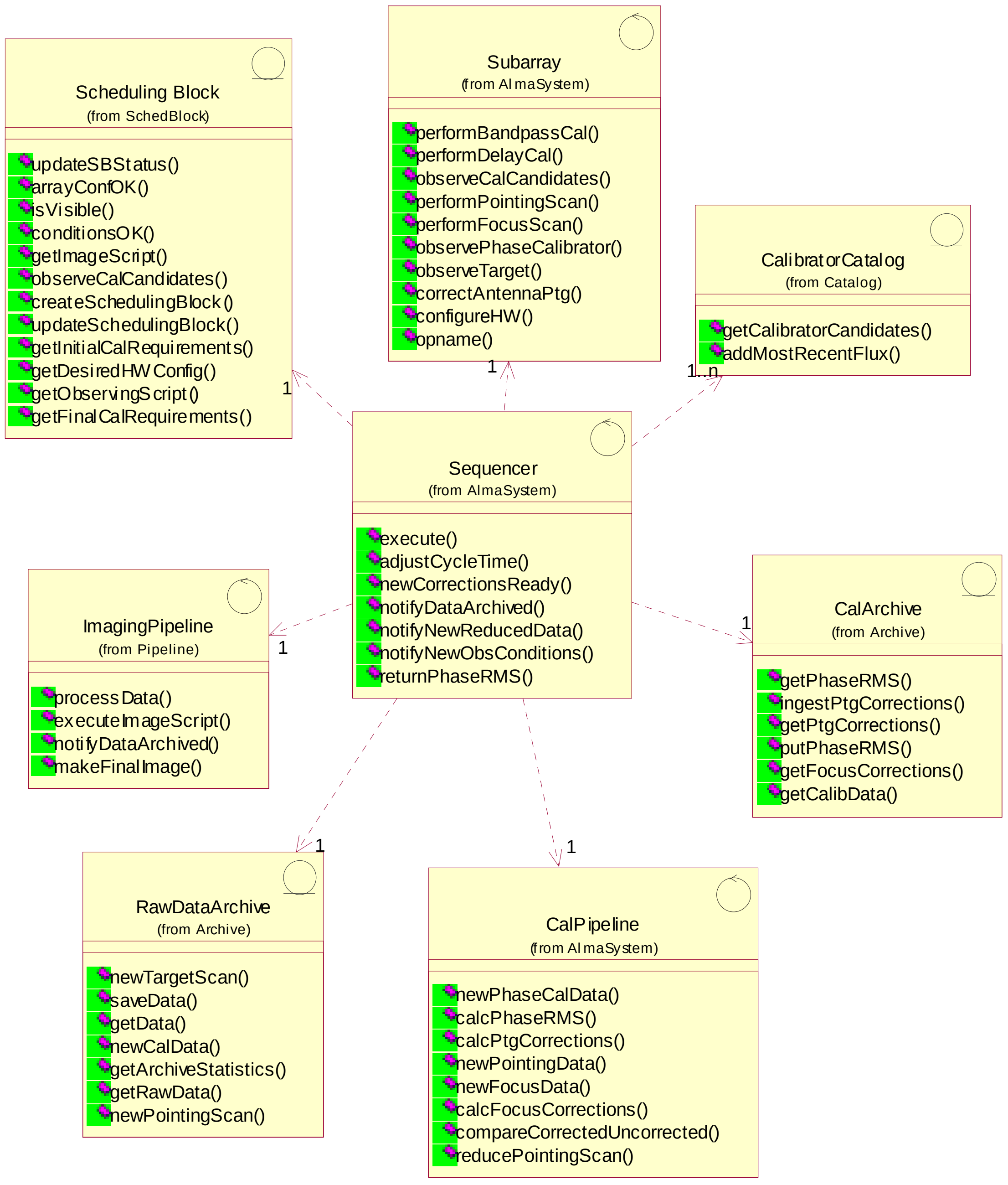


Figure 2-7: Sequencer Class Diagram

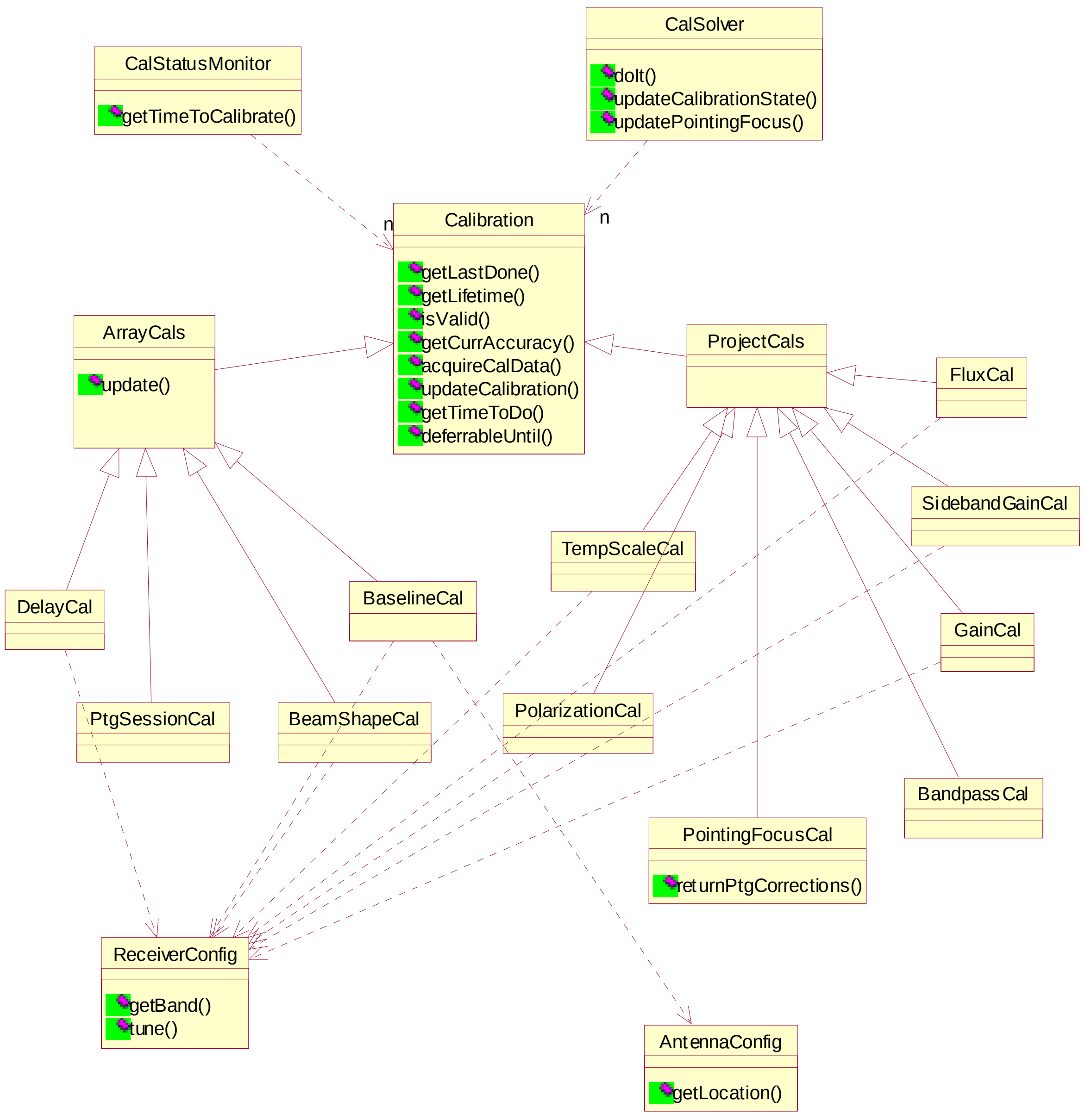


Figure 2-8: Calibration Class Diagram

Calibration objects have the responsibility to know their states and the dependencies of those states (on elapsed time and on the relevant hardware configuration), to be able to direct the hardware to perform the observations necessary to provide data for an update of the calibration, and to be able to initiate and control the near-real-time reduction of this data as necessary. When requested, each object will also be able to return the time required to perform such an update (useful for scheduling purposes). These capabilities are intended to be common to (almost) all calibration objects, whose classes are therefore shown as an inheritance hierarchy. Where appropriate, mechanisms will be provided for notification by the calibration objects when they become invalid; direct programmatic interrogation will always be possible. An additional control class, CalSolver, is provided to bring a set of resources (antennas, receivers, correlator) to a desired calibration state (or to do nothing if that state has already been reached).

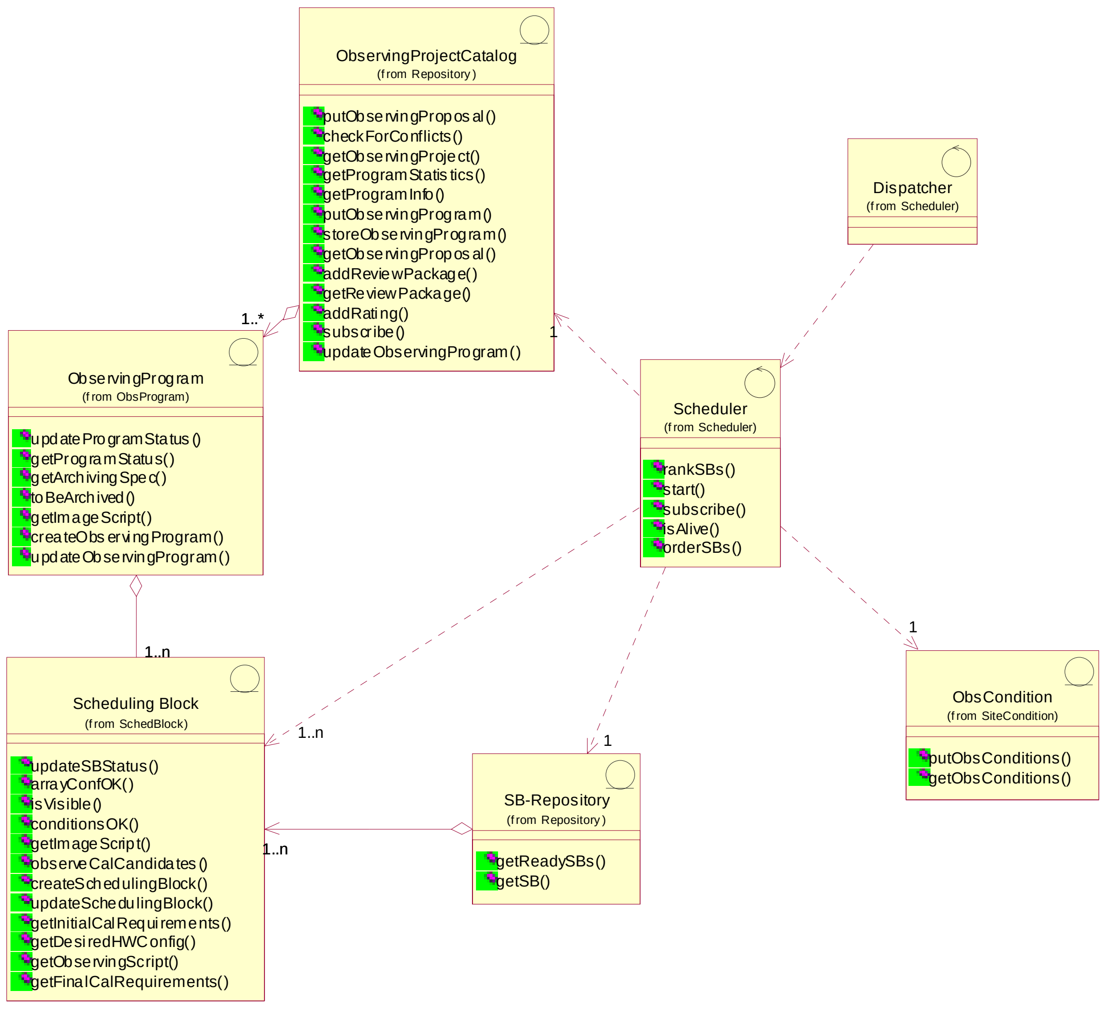


Figure 2-9: Scheduler Class Diagram

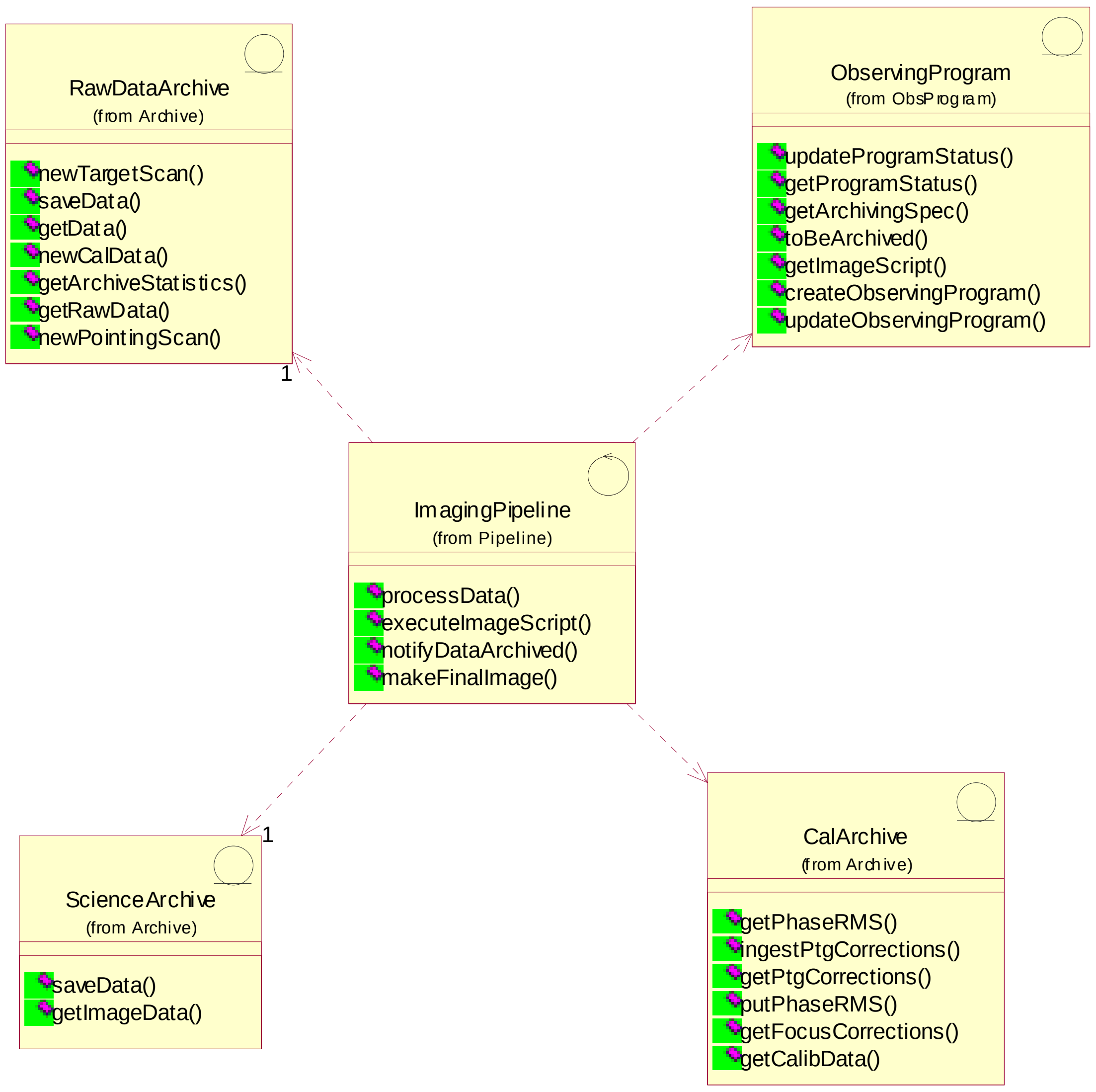


Figure 2-10: Pipeline Class Diagram

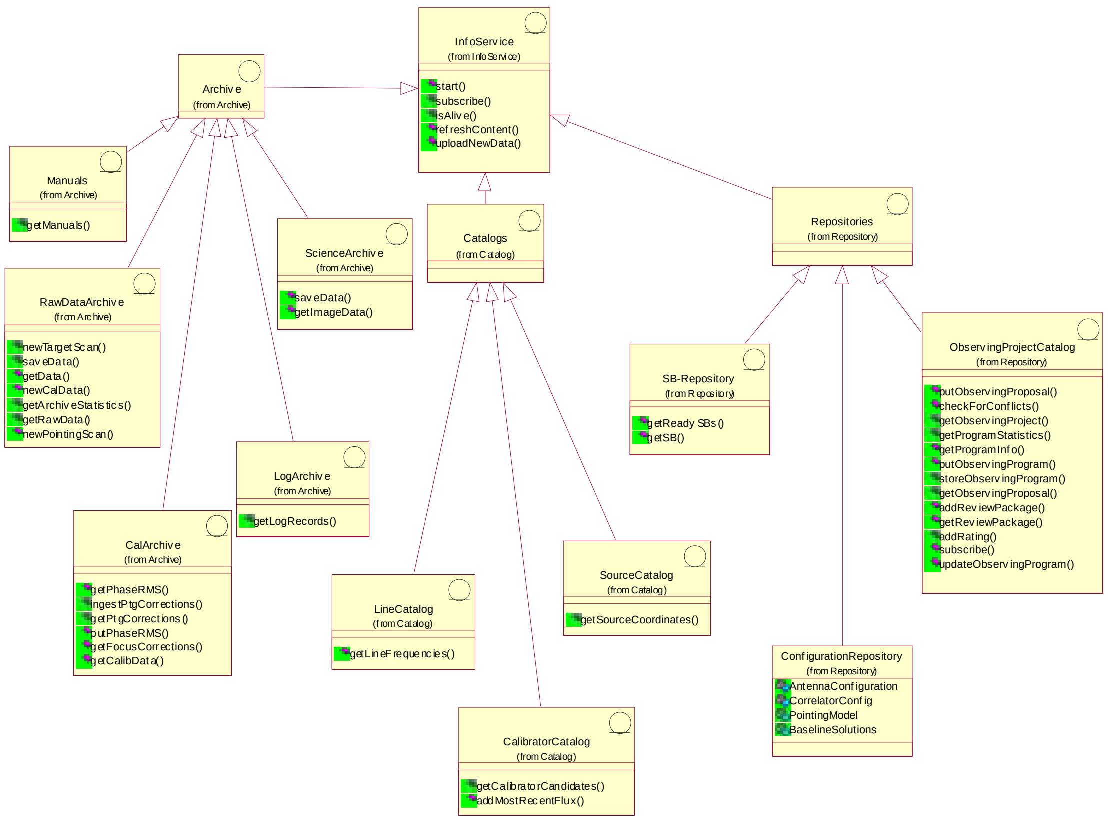


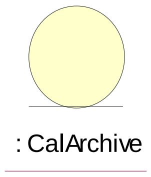
Figure 2-11: InfoServices Class Diagram

3 Use Case Realizations

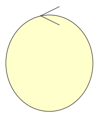
3.1 Initial Sequence Diagrams & Description

Each of the following sequence diagrams is derived (not necessarily line-by-line) from the corresponding Use Case. Each follows a timeline that proceeds from top to bottom. Each class used for analysis is one of three kinds and has a corresponding icon below which is displayed its name. (Informal definitions of the classes used can be found in the preceding section, along with class diagrams, where the relationships among the classes warrant display.)

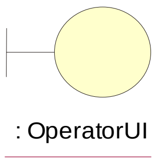
Entity class: responsible primarily for accepting, storing and retrieving persistent data. Archives are typical examples. Icon is a circle resting on a horizontal line.



Control class: responsible for control, coordination and/or processing. The Dispatcher and the Observing Tool are examples. Icon is a circle with an arrow pointing counterclockwise



Boundary class: represents an interface, either to a human or to another hardware or software system (or subsystem, depending on the context). Icon is a vertical line connected to a circle by a horizontal one.



Messages are shown as directed horizontal lines between objects; we say that the originating object is invoking a *method* in the receiving object.

It is important to understand that a sequence diagram represents a single scenario, that is, only *one* of many possible paths through a Use Case. Some conditional branching may be shown on a sequence diagram, but this is the exception rather than the rule. Ultimately, one hopes to explore all the “interesting” scenarios for each Use Case.

3.1.1 Create & Submit Observing Proposal

The proposer creates, validates and submits a Phase I Proposal to the ALMA Observatory for review, using an observatory-supplied Observing Tool. (Note that creation of an Observing Proposal implies the creation—by the system—of an enclosing Observing Project; see the previous chapter for details of this relationship.) He/she can either create a new or retrieve and edit a locally existing Phase I Proposal and normally works off-line. Network access is required for consulting the on-line ALMA Observer manuals, catalogues, etc., and to verify that the version of the Observing Tool in use is current.

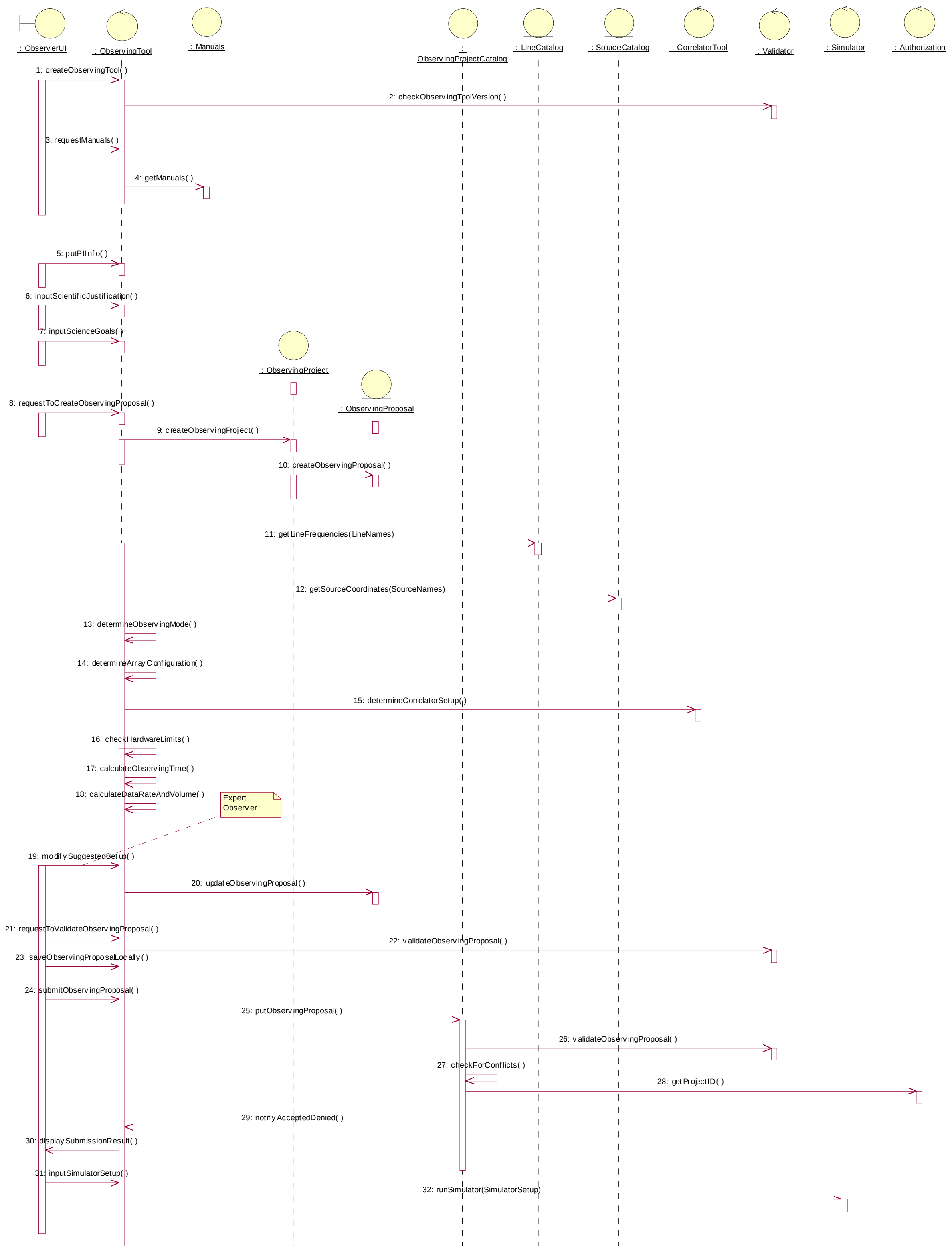

: ObserverUI
: ObservingTool
: Manuals
: ObservingProjectCatalog
: LineCatalog
: SourceCatalog
: CorrelatorTool
: Validator
: Simulator
: Authorization
1: createObservingTool( )
2: checkObservingToolVersion( )
3: requestManuals( )
4: getManuals( )
5: putPIInfo( )
6: inputScientificJustification( )
7: inputScienceGoals( )
: ObservingProject
: ObservingProposal
8: requestToCreateObservingProposal( )
9: createObservingProject( )
10: createObservingProposal( )
11: getLineFrequencies(LineNames)
12: getSourceCoordinates(SourceNames)
13: determineObservingMode( )
14: determineArrayConfiguration( )
15: determineCorrelatorSetup( )
16: checkHardwareLimits( )
17: calculateObservingTime( )
18: calculateDataRateAndVolume( )
Expert Observer
19: modifySuggestedSetup( )
20: updateObservingProposal( )
21: requestToValidateObservingProposal( )
22: validateObservingProposal( )
23: saveObservingProposalLocally( )
24: submitObservingProposal( )
25: putObservingProposal( )
26: validateObservingProposal( )
27: checkForConflicts( )
28: getProjectID( )
29: notifyAcceptedDenied( )
30: displaySubmissionResult( )
31: inputSimulatorSetup( )
32: runSimulator(SimulatorSetup)

Figure 3-1: Create & Submit Observing Proposal

1. The Proposer starts the Observing Tool (OT).
2. The OT asks the Validator (which is located within the ALMA system) to check if this is the most recent OT version.

3. The Proposer asks for the ALMA Manuals to learn about the instrument. This step could happen at any time.
4. The OT fetches the Manuals.
5. The Proposer inputs his/her name, institution, address, etc., along with similar information concerning Co-Investigators, if any. We assume for simplicity that the Proposal is being prepared either by the PI or his/her proxy.
6. The Proposer enters the Scientific Justification for the Observing Proposal (OProp).
7. The Proposer enters the Scientific Goals of the OProp. The Science Goals comprise:
    - source ID or coordinates.
    - desired angular resolution and largest structure.
    - source flux and S/N or rms.
    - line identification or frequencies.
    - desired velocity resolution and width.
    - desired dynamic range.
    - archiving specification.
8. The Proposer requests the OProp to be created and the calculations necessary for Phase I to be done.
9. The OT creates a new Observing Project, the top level container holding the Observing Proposal and later the hierarchy of Observing Programs (see Create & Submit Observing Program and SBs).
10. The OT creates the OProp.
11. The OT gets line frequencies from the Line Catalog.
12. The OT gets source coordinates and properties from the Source Catalog.
13. The OT determines the Observing Mode necessary to perform the proposed project based on the largest structure.
14. The OT determines the necessary Array Configuration based on the angular resolution.
15. The OT determines the Correlator Setup based on the line frequencies and the desired velocity resolution.
16. The OT checks whether the proposed setup is within the hardware limits.
17. The OT calculates the necessary observing time under average observing conditions.
18. The OT calculates the expected Data Rate and Data Volume.
19. The (expert) Proposer optionally modifies the suggested setup.
20. The OT updates the OProp.
21. The Proposer optionally asks to validate the OProp.
22. The OT asks the Validator to validate the OProp.
23. The Proposer optionally saves the OProp locally.

24. The Proposer submits the finished OProp.
25. The OT sends the OProp to the Observing Project Catalog.
26. The Observing Project Catalog asks the Validator to validate the OProp.
27. The Observing Project Catalog checks for conflicts with other Observing Proposals/Observing Programs
28. The Observing Project Catalog gets a unique ID for the OProp from Authorization
29. The Observing Project Catalog notifies the OT whether the OProp has been accepted or denied.
30. The OT informs the Proposer about the submission result. In case of acceptance it returns the ProjectID as unique identifier.
31. At any time the Proposer optionally runs the Simulator to find out if a given project is feasible with ALMA.
32. The OT asks the Simulator to simulate the given setup.

3.1.2 Create & Submit Observing Program & SBs

Starting from an approved Observing Proposal, the proposer/observer uses the Observing Tool to deliver the complete technical specifications for the Observing Programme to the Phase II Scheduling Block Repository. The Observing Programme adds one or more subprograms and Scheduling Blocks and relations/dependencies among them to the approved Proposal

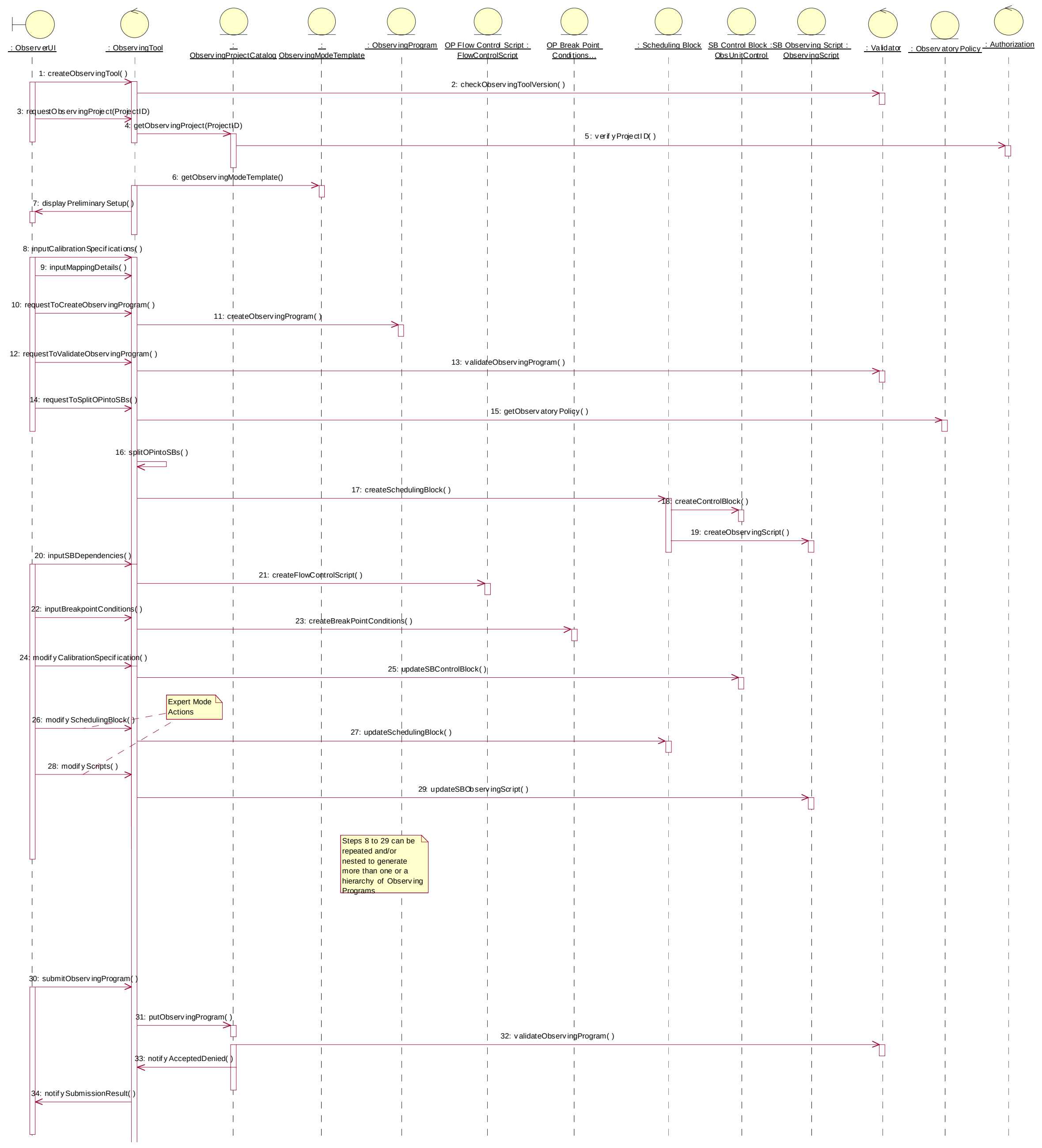

: ObserverUI
: ObservingTool
: ObservingProjectCatalog
: ObservingModeTemplate
: ObservingProgram
OP Flow Control Script : FlowControlScript
OP Break Point Conditions...
: Scheduling Block
SB Control Block : ObsUnitControl
SB Observing Script : ObservingScript
: Validator
: Observatory Policy
: Authorization
1: createObservingTool( )
2: checkObservingToolVersion( )
3: requestObservingProject(ProjectID)
4: getObservingProject(ProjectID)
5: verifyProjectID( )
6: getObservingModeTemplate()
7: display Preliminary Setup( )
8: inputCalibrationSpecifications( )
9: inputMappingDetails( )
10: requestToCreateObservingProgram( )
11: createObservingProgram( )
12: requestToValidateObservingProgram( )
13: validateObservingProgram( )
14: requestToSplitOPintoSBs( )
15: getObservatoryPolicy( )
16: splitOPintoSBs( )
17: createSchedulingBlock( )
18: createControlBlock( )
19: createObservingScript( )
20: inputSBDependencies( )
21: createFlowControlScript( )
22: inputBreakpointConditions( )
23: createBreakPointConditions( )
24: modifyCalibrationSpecification( )
25: updateSBControlBlock( )
Expert Mode Actions
26: modifySchedulingBlock( )
27: updateSchedulingBlock( )
28: modifyScripts( )
29: updateSBObservingScript( )
Steps 8 to 29 can be repeated and/or nested to generate more than one or a hierarchy of Observing Programs
30: submitObservingProgram( )
31: putObservingProgram( )
32: validateObservingProgram( )
33: notifyAcceptedDenied( )
34: notifySubmissionResult( )

Figure 3-2: Create & Submit Observing Program & SBs

1. The Observer starts the Observing Tool (OT).
2. The OT asks the Validator (which is located within the ALMA system) to check if this is the most recent OT version.
3. The Observer requests his/her Observing Project using the previously assigned ProjectID.
4. The OT asks the Observing Project Catalog to fetch the given Observing Project
5. The Observing Project Catalog asks Authorization to verify the ProjectID.
6. OT fetches the Observing Mode Templates according to the Observing Modes listed in the Observing Proposal.

7. The OT displays the preliminary setup.
8. The Observer enters the desired calibration specifications.
9. The Observer enters the desired mapping details.
10. The Observer asks the OT to create the Observing Program (OP).
11. The OT creates the OP.
12. The Observer optionally asks the OT to validate the OP.
13. The OT asks the Validator to validate the OP.
14. The Observer requests the OP to be split into Scheduling Blocks (SBs).
15. The OT fetches the Observatory Policy to determine policies such as maximum time per SB.
16. The OT splits the OP into SBs according to the Observatory Policy.
17. The OT creates the SBs.
18. The SB creates the necessary SB Control Block.
19. The SB creates the necessary SB Observing Script.
20. The Observer optionally enters SB dependencies.
21. The OT creates a Flow Control Script which encodes the SB dependencies.
22. The Observer optionally enters Breakpoint Conditions to be able to check preliminary observing results.
23. The OT creates the Breakpoint Conditions.
24. The Observer optionally modifies the Calibration Specifications.
25. The OT updates the Calibration Specifications.
26. The (expert) Observer optionally modifies the SB(s).
27. The OT updates the SB(s).
28. The (expert) Observer optionally modifies the SB Observing and/or Image Script(s).
29. The OT updates the SB Observing and/or Image Script(s).

Note: Steps 8 to 29 can be repeated and/or nested to generate more than one or a hierarchy of Observing Programs.

30. The Observer submits the finished OP.
31. The OT sends the OP to the Observing Project Catalog.

32. The Observing Project Catalog asks the Validator to validate the OP.
33. The Observing Project Catalog notifies the OT whether the OP has been accepted or denied.
34. The OT informs the Observer about the submission result.

3.1.2.1 Main Findings:

1. The Validator and the Simulator need to be exportable in order to be able to use the Observing Tool locally. The Observing Tool and the Validator need to be synchronized with the internal ALMA version.
2. Identified the need for an authorization / security mechanism to allow access to proposals/programs only for the PI/CoI's, the reviewers and the ALMA staff.
3. Need details on how to determine the ideal Observing Mode, the necessary Array Configuration, the Observing Time and the Data Rate and Volume from given Science Goals for further analysis.
4. The requirements say that the general user does not have to modify anything in the setup that was suggested when the proposal was created. All modification steps are thus optional or for expert observers.

3.1.3 Dispatch SBs

The Dispatcher is responsible for marshalling commands and scripts for execution. It can receive these commands and scripts in any of four modes: dynamically scheduled, interactive, manual and technical. The first two of these modes require the passing of complete Scheduling Blocks; only the dynamic and interactive modes are displayed here. The SB's script itself is executed by the Sequencer, which might be a simple script interpreter, or might be required to know something about an SB’s structure as well (in particular, its configuration and calibration requirements, to the extent that these are not embedded in a script); execution of the instrument-specific commands is performed by the Subarray object, described in the "Observe Single Field" Sequence Diagram.

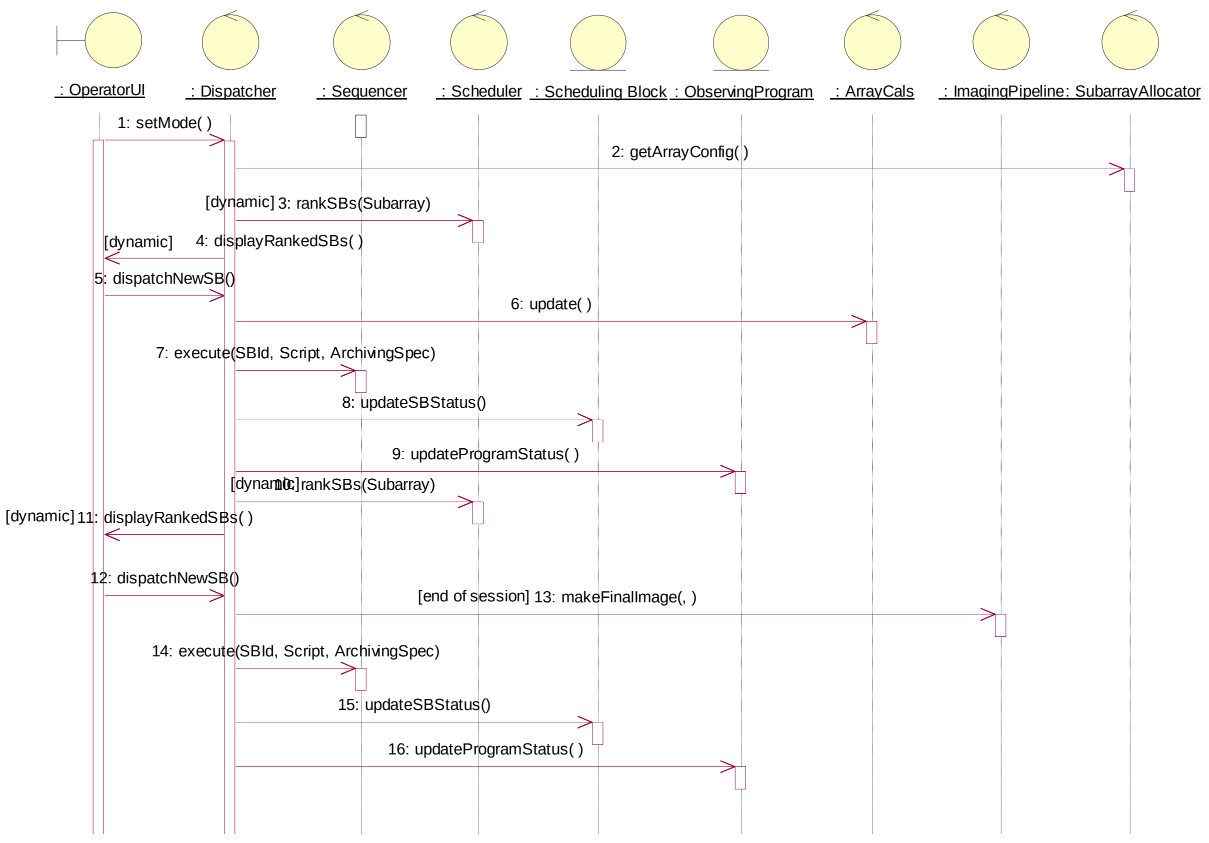

: OperatorUI
: Dispatcher
: Sequencer
: Scheduler
: Scheduling Block
: ObservingProgram
: ArrayCals
: ImagingPipeline
: SubarrayAllocator
1: setMode( )
2: getArrayConfig( )
[dynamic] 3: rankSBs(Subarray)
[dynamic] 4: displayRankedSBs( )
5: dispatchNewSB()
6: update( )
7: execute(SBId, Script, ArchivingSpec)
8: updateSBStatus()
9: updateProgramStatus( )
[dynamic] 10: rankSBs(Subarray)
[dynamic] 11: displayRankedSBs( )
12: dispatchNewSB()
[end of session] 13: makeFinalImage(,)
14: execute(SBId, Script, ArchivingSpec)
15: updateSBStatus()
16: updateProgramStatus( )

Figure 3-3: Dispatch SBs

1. The operator instructs the Dispatcher to use either dynamic or interactive scheduling: setMode().
2. Dispatcher asks the SubarrayAllocator for the antennas and correlator resources that it may use: getArrayConfig().
3. If dynamic mode is in use, the Dispatcher asks the Scheduler for an ordered list of Scheduling Blocks that can run with currently allocated antennas and correlator resources: rankSBs(Subarray)
4. If dynamic mode is in use, the Dispatcher requests the Operator UI to display the ranked SBs to the operator: displayRankedSBs().
5. Operator may request the Dispatcher to choose a different SB for execution; a reason for the change is given at the same time: dispatchNewSB(Reason). In interactive mode, this step is mandatory
6. Dispatcher asks the ArrayCals to update any array calibrations that have become invalid and that cannot or should not be deferred: update().
7. Dispatcher asks the Sequencer to execute the highest-ranked (dynamic mode) or manually selected (interactive mode) SB: executeSB(). Although the Dispatcher waits for the Sequencer to return (along with a status) after executing the SB, it may timeout and request abnormal SB termination if the SB's time limit is exceeded: During the

wait period, Dispatcher may request execution of 1 or more SBs that can use any antennas not required by the currently executing highest-ranked SB, *i.e.*, it may loop on steps 2-8.

8. Upon completion (or abnormal termination) of the SB, the Sequencer updates the SB's status: updateSBStatus().
9. Sequencer updates the status of the ObservingProgram corresponding to the just-terminated SB: updateProgramStatus().
10. In dynamic mode, Dispatcher requests another ordered list of SBs from the Scheduler: rankSBs().
11. See Step #3.
12. See Step #4.
13. If the new SB is from a different Programme (or Subprogramme) than the previously executed one, then the session is ending, and the Dispatcher instructs the ImagingPipeline to create a definitive image with all data taken so far: makeFinalImage().
14. execute(SBId,SBScript,ArchivingSpec).
15. See Step #8.
16. See Step #9.

3.1.4 Schedule SB

Important to note in the following Sequence Diagram is that the Scheduler relies on other objects to let it know which SBs are actually ready to run; having assembled the necessary information, the Scheduler uses the TBD algorithm referred to in requirements 4.0-R3 to prioritize these SBs.

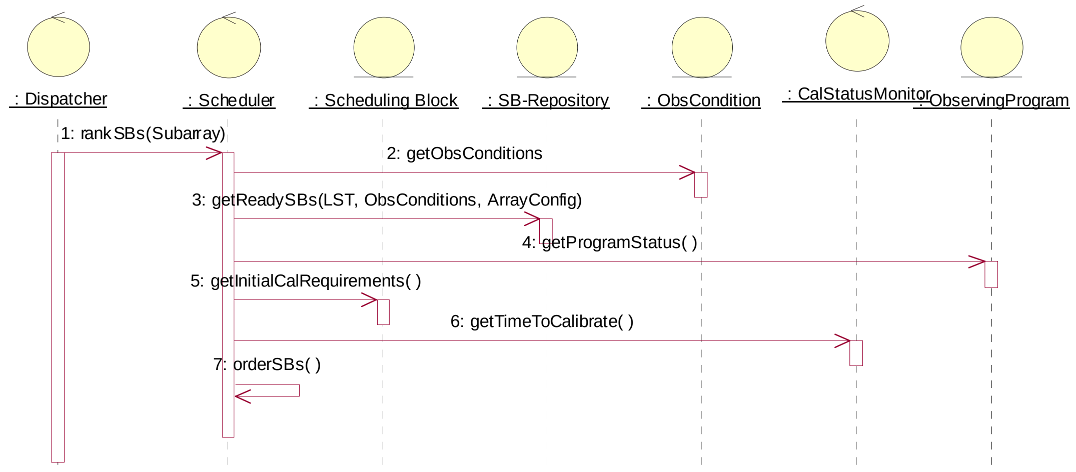


Figure 3-4: Schedule SB

1. Dispatcher tells Scheduler to return an ordered list of Scheduling Blocks: rankSBs(ArrayConfig).
2. Scheduler requests observing conditions from ObsCondition: getObsConditions().
3. Scheduler requests from SB Repository all SBs that can run with the current array configuration and observing conditions, are visible in the current LST range, and are not currently being edited or waiting on breakpoints and that do not have dependencies on other SBs that have not yet been run: getReadySBs(LST,ObsConditions,ArrayConfig).
4. Scheduler queries the ObservingProgram of each remaining SB to determine its status: getProgramStatus().
5. Scheduler gets each SB's startup calibration requirements: getInitialCalRequirements().
6. For each SB, Scheduler asks the CalStatusMonitor to determine the time necessary to bring SB to its required calibration state: getTimeToCalibrate().
7. Scheduler uses all its information concerning program status, scientific priority, time to execute (including time to calibrate) and conditions to rank the SBs: orderSBs().

3.1.5 Execute SB

The Dispatcher passes the highest-priority (or manually-selected) Scheduling Block to the Sequencer for execution. The Sequencer acquires configuration, calibration and observing instructions from the Scheduling Block and carries them out with the help of objects that know how to configure, calibrate and observe via the hardware and processing engines (not shown here). The calibration objects can (optionally) skip a requested calibration update if the desired calibration state has already been reached.

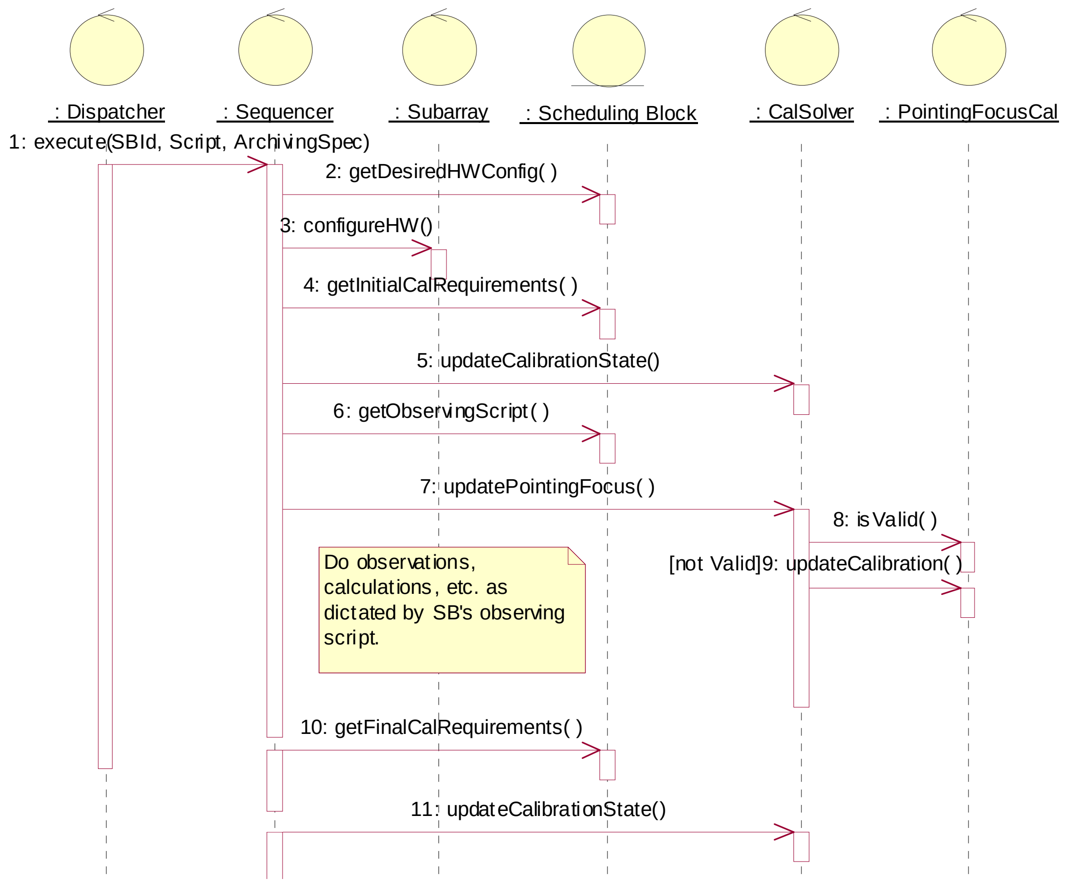

: Dispatcher
: Sequencer
: Subarray
: Scheduling Block
: CalSolver
: PointingFocusCal
1: execute(SBId, Script, ArchivingSpec)
2: getDesiredHWConfig( )
3: configureHW()
4: getInitialCalRequirements( )
5: updateCalibrationState()
6: getObservingScript( )
7: updatePointingFocus( )
8: isValid( )
[not Valid]9: updateCalibration( )
Do observations, calculations, etc. as dictated by SB's observing script.
10: getFinalCalRequirements( )
11: updateCalibrationState()

Figure 3-5: Execute SB

1. Dispatcher instructs the Sequencer to execute a Scheduling Block
2. Sequencer gets the SB's hardware configuration requirements (correlator setup, receiver tuning, etc.)
3. Sequencer instructs the Configurator to configure the hardware.
4. Sequencer gets the SB's initial calibration requirements.
5. Sequencer instructs the CalSolver to perform any necessary calibrations to bring the calibration state to that required by the SB.
6. Sequencer gets the SB's observing script and executes its commands.
7. Periodically, the Sequencer (as instructed by the script) will instruct the CalSolver to update the Pointing and Focus calibrations. (It is also possible to use an asynchronous notification mechanism, in which the PointingFocusCalibration object itself could notify the Sequencer when its validity expires).
8. CalSolver asks the PointingFocusCalibration object whether it is still valid.
9. If the calibration is no longer valid (either because a determined time has elapsed

since the last calibration update or because the antenna has slewed beyond a certain amount) CalSolver requests the PointingFocusCalibration object to update itself. (These two steps could be combined so that the PointingFocusCalibration object updates itself if necessary. The choice made her leaves more explicit control in the hands of the CalSolver.) See Section 3.1.7, ObservePointingCalibration, for details.

10. Once the observing script has finished, the Sequencer gets the SB's final calibration requirements.

11. Sequencer requests the CalSolver to ensure that the final calibration state has been reached or, if desired, that particular calibrations are redone.

3.1.6 Observe Single Field

This Sequence Diagram shows the execution of the script of an SB designed to execute a (part of a) Single Field interferometric observation. A basic mechanism used here is that of notifying objects when new data (either raw or processed) is available, but requiring those objects to request the data when they are ready for it.

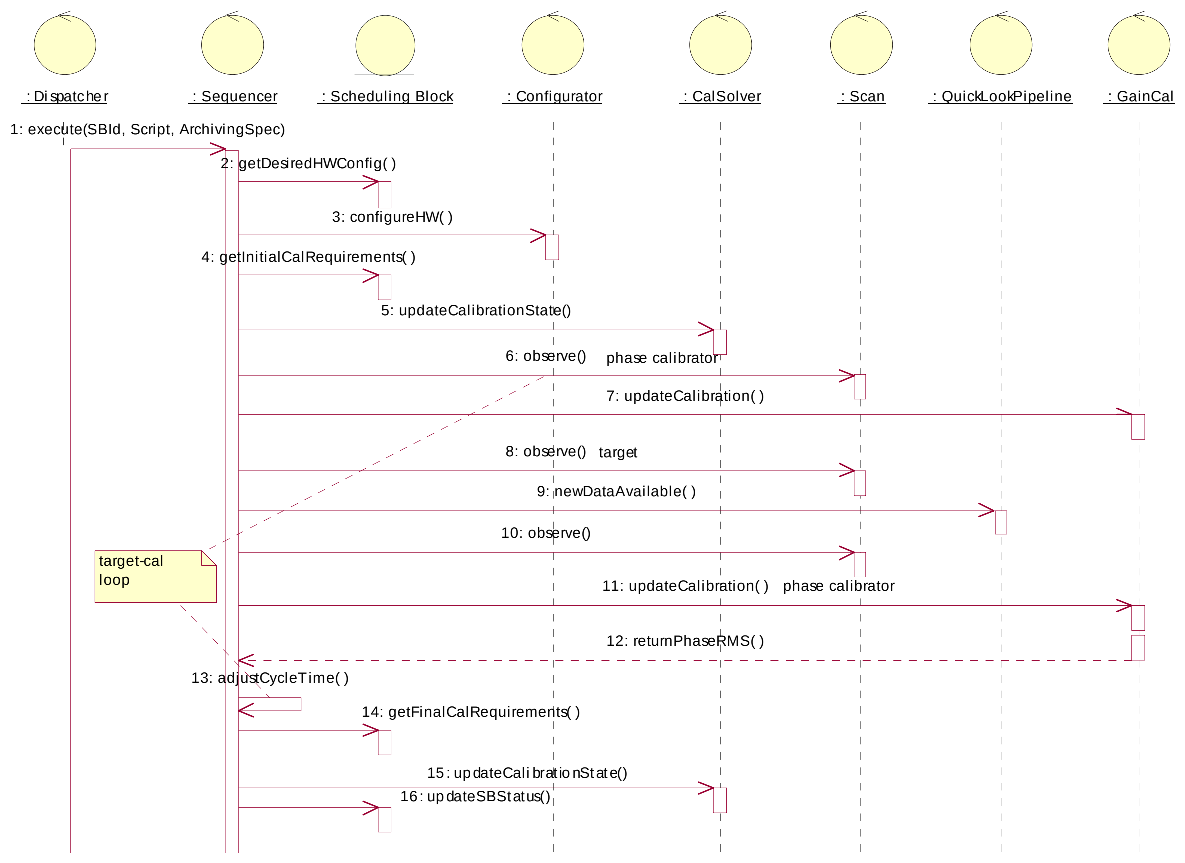


Figure 3-6: Observe Single Field

1. Sequencer receives instruction from Dispatcher to execute a Scheduling Block: execute(SBId,PreambleScript,ArchivingSpec).
2. Sequencer gets the SB’s hardware configuration requirements (correlator setup, receiver tuning, etc.)
3. Sequencer instructs the Configurator to configure the hardware.
4. Sequencer gets the SB’s initial calibration requirements.
5. Sequencer instructs the CalSolver to perform any necessary calibrations to bring the calibration state to that required by the SB. If this is the first SB in a session, these will include gain, bandpass, focus and pointing calibrations. If suitable calibration

sources have not been defined, CalSolver will coordinate the actions necessary to find them.

6. Sequencer instructs the Observation object to observe the phase calibration source: observe().

7. Sequencer instructs the GainCal object to process (asynchronously) the calibration data just acquired: updateCalibration()

8. Sequencer instructs the Observation object to observe the target source: observe().

9. Sequencer notifies the QuickLookPipeline that there is new target data to process; the pipeline will proceed asynchronously: newDataAvailable().

10. Sequencer instructs the Observation object to observe the phase calibration source: observe().

11. Sequencer instructs the GainCal object to process (asynchronously) the calibration data just acquired: updateCalibration()

12. CalPipeline asynchronously returns a new value of the phase rms when it has accumulated and processed enough phase calibrator data: returnPhaseRMS(). (CalPipeline adds this new result to the Observing Conditions; this interaction is not shown here.)

13. If the rms is too high, the Sequencer adjusts the calibrator cycle time to compensate: adjustCycleTime().

    Sequencer loops through Steps 7 to 13 until the desired performance goal is reached or the SB's time (minus some tolerance for final calibrations) is exhausted.

14. Sequencer gets SB's final calibration requirements.

15. Sequencer instructs CalSolver to update all calibration states (performing whatever calibrations are necessary) accordingly.

16. Dispatcher updates the status of the just-completed SB: updataSBStatus(newStatus).

3.1.6.1 Questions & Issues

1. How should the asynchronous delivery of the CalPipeline's results be handled for the various types of calibration? Is there some point at which lack of calibration results becomes fatal to execution of the SB?

2. So many SBs from the same program or sub-program might be executed contiguously that some parts of the calibration lose enough accuracy to become invalid. As a framework for a mechanism to handle this, the CalStatusMonitor (see class diagrams) has been defined; this object can be asked whether the current calibration status is satisfactory for any specified SB, as well as how long it should take to bring any needed calibrations up-to-date. Each calibration object would include a "maximum time interval between runs" to make sure that it gets executed as often as necessary; for calibrations for which this concept makes no sense, *e.g.*, gain and temperature scale calibrations, this capability would be suppressed, and the calibrations would be performed under direct programmatic control.

3.1.7 ObservePointingCalibration

Pointing and Focus calibrations require concurrent observing and processing activities. A PointingFocusCal object has the responsibility for managing these activities and ensuring that a satisfactory result is reached unless a preset time expires or a manual interrupt occurs.

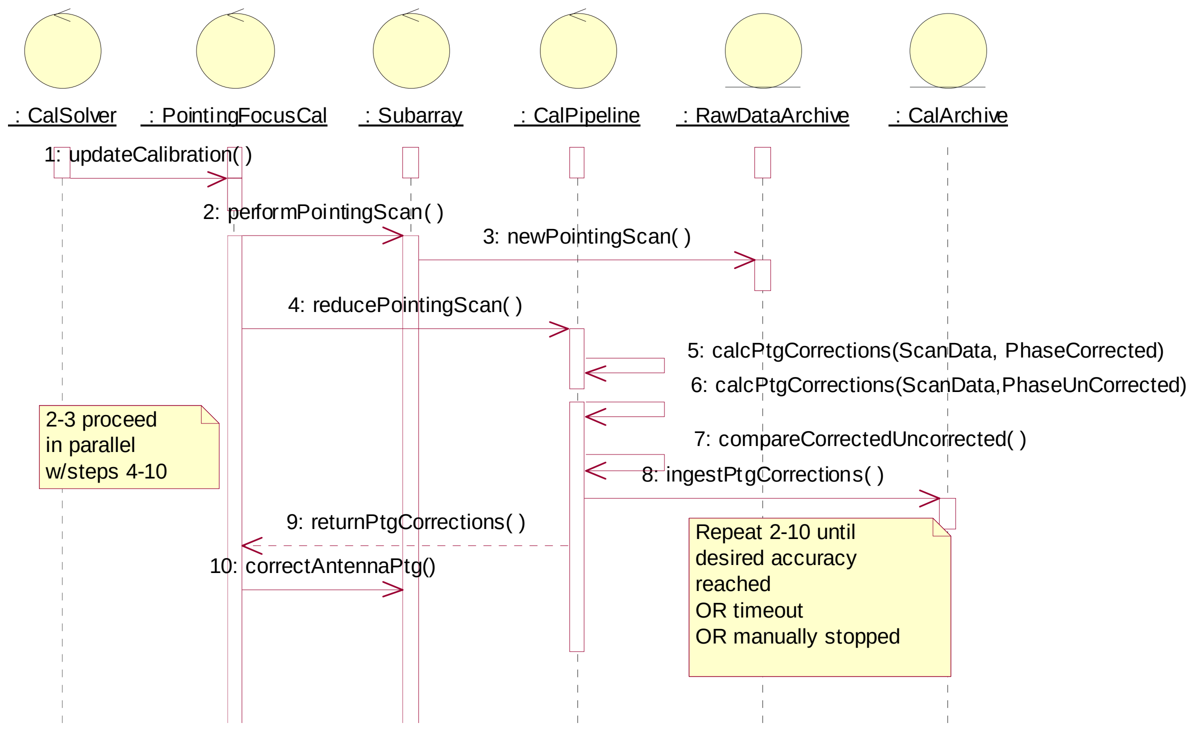

: CalSolver
: PointingFocusCal
: Subarray
: CalPipeline
: RawDataArchive
: CalArchive
1: updateCalibration( )
2: performPointingScan( )
3: newPointingScan( )
4: reducePointingScan( )
5: calcPtgCorrections(ScanData, PhaseCorrected)
6: calcPtgCorrections(ScanData,PhaseUnCorrected)
7: compareCorrectedUncorrected( )
8: ingestPtgCorrections( )
9: returnPtgCorrections( )
10: correctAntennaPtg()
2-3 proceed in parallel w/steps 4-10
Repeat 2-10 until desired accuracy reached OR timeout OR manually stopped

Figure 3-7: Observe Pointing Calibration

1. CalSolver instructs PointingFocusCal to update its state.
2. PointingFocusCal instructs the Subarray (in this case, all the antennas at its disposal) to perform a pointing scan: performPointingScan().
3. When scan is completed, Subarray stores scan data in RawDataArchive: newPointingScan().
4. PointingFocusCal tells CalPipeline to reduce the pointing scan: reducePointingScan(). These steps are repeated while the CalPipeline performs steps 4-8 below.
5. CalPipeline calculates the pointing offset corrections using phase-corrected data.
6. CalPipeline calculates the pointing offset corrections using phase-uncorrected data.

CalPipeline compares the errors on the offset corrections calculated in Steps 5 and 6, choosing the set with the smaller errors: compareCorrectedUncorrected(). (S. Scott: “errors estimated by taking a series of offset estimates and computing their scatter. These estimates may have a weight, for example, from the fitting process, or by measuring the scatter of the visibilities on shorter timescales.”)

7. CalPipeline stores the new pointing offset corrections and errors in the CalArchive: ingestPtgCorrections().

8. CalPipeline returns the new corrections to PointingFocusCal
9. If necessary, results so far are used to correct the pointing of one or more antennas: correctAntennaPtg(Antenna_#)
10. Steps 2-10 are repeated until the desired pointing accuracy is reached, the preset time limit has been reached, or the calibration is terminated manually.

3.1.8 ProcessData

This Sequence Diagram displays the interactions necessary to process raw data into calibrated images and to archive the results. Sequence diagrams for Quicklook and standard Image reduction are virtually identical. The major difference is the particular Image Script that is executed (deconvolution is included only for the standard reduction). Thus the Process Data diagram covers both cases.

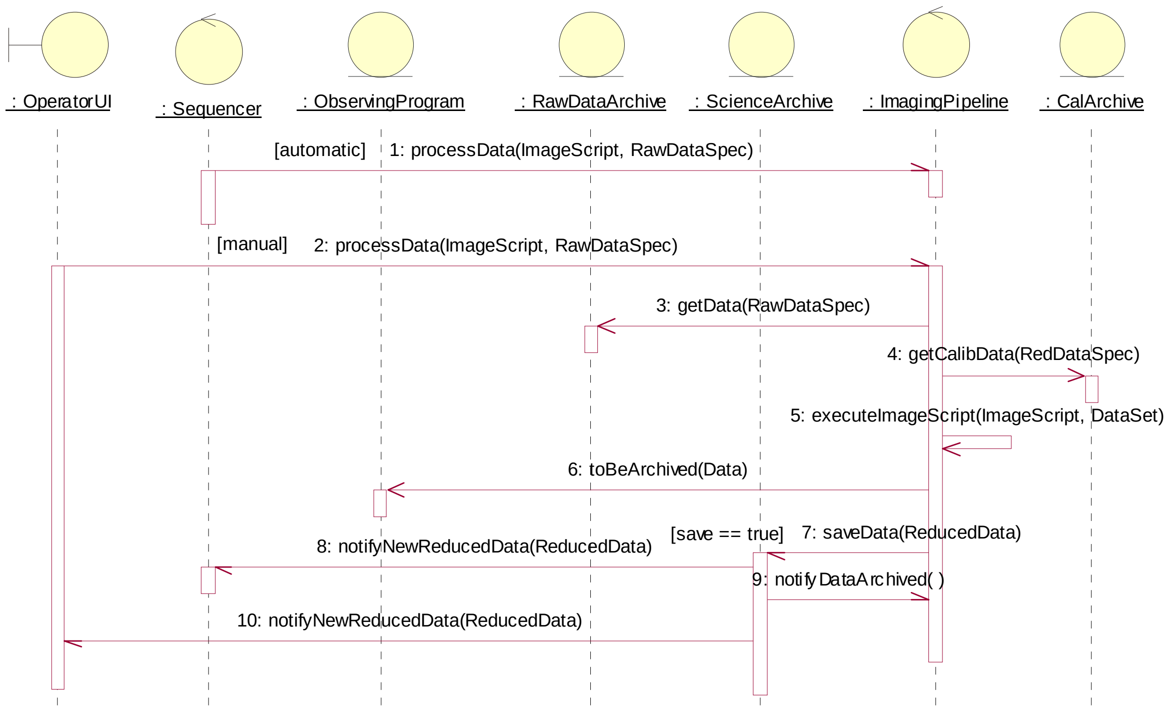


Figure 3-8: Process Science Data

1. In automatic mode, the Sequencer has been notified that new data are available and decides (perhaps at the end of a session) that the ImagingPipeline should process them. This is done asynchronously to enable the observatory to proceed with observing while the data are being reduced.
2. In manual mode, the User has been notified that new data are available and may decide that the ImagingPipeline should process them. This is done asynchronously to enable the User to perform other tasks while the data are being reduced.
3. The ImagingPipeline retrieves the data to be reduced from the RawDataArchive by 'getData'. This migrates the data to a place convenient for the ImagingPipeline.
4. The Image Production Script specifies what types of calibration data are required to

reduce the raw data in hand. These calibration data are obtained from the Calibration Archive using the method 'getCalibData(RedDataSet)'.

5. Once the IPScript and the data are available, the script is executed by 'executeImageScript(ImageScript, DataSet)'.
6. The Pipeline checks with the Observing Program if the reduced data actually should be stored using 'toBeArchived(Data)'.
7. In case they should be stored in the Science Archive, they are saved by the method 'saveData(ReducedData)'.
8. When the Reduced Data have been saved in the Science Archive, the Imaging Pipeline notifies Sequencer of its availability.
9. The Science Archive notifies the Pipeline that the data have been saved successfully so that temporary copies can be removed.
10. Also the User is notified when a final, deconvolved image is available.

3.2 Proposal/Project Preparation Activity Lifecycle

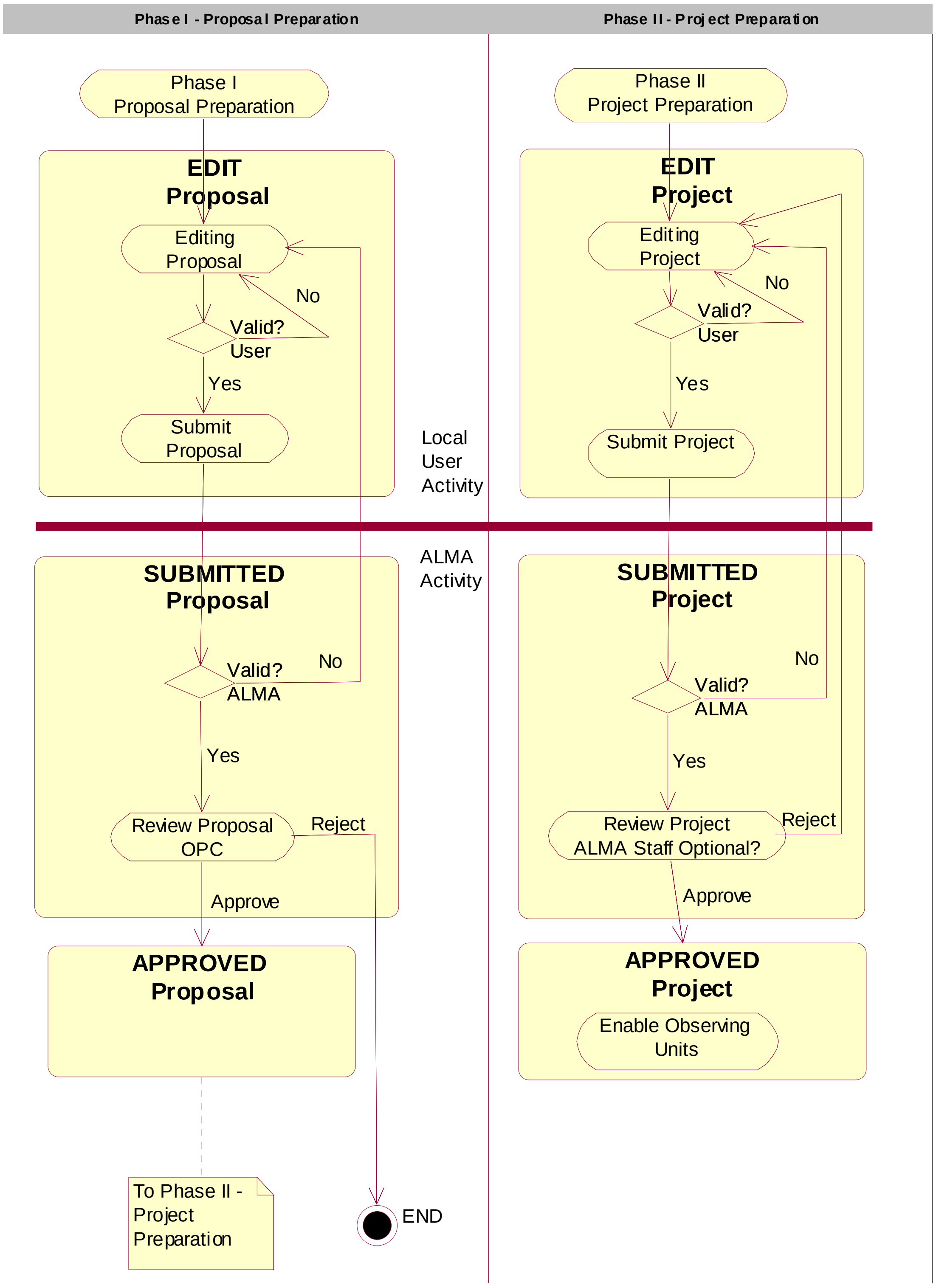


Figure 3-9: Proposal/Project Preparation Activity Diagram

3.2.1 Phase I Proposal Preparation State Diagram

Description

In Phase I Proposal Preparation the Observer creates a Proposal and submits it to the Observatory. The Proposal is reviewed and either accepted or rejected.

States

1. EDIT Proposal

The Observer enters all mandatory Phase I information. Optionally, Phase II information can be provided.

The editing process ends with the submission of a locally validated proposal to the Observatory.

The OT provides a number of additional services to help the Observer with the creating of the Observing Proposal, *e.g.*, array configuration, correlator setup, observing time required, anticipated data rates and data volumes.

2. SUBMITTED Proposal

The submitted proposal is validated by the Observatory and is reviewed by the Observing Programme Committee (OPC) on its scientific and technical merits

The Observer is informed in case:

- the proposal is accepted;
- the proposal is rejected;
- the validation failed (proposal needs to be corrected and re-submitted).

3. APPROVED Proposal

The proposal is granted observing time and the Observer is required to submit Scheduling Blocks.

3.2.2 Phase II Project Preparation

Description

In Phase II Proposal Preparation the Observer creates one or more Observing Units (detailed down to the Scheduling Blocks [SB] level) for the Observing Project and submits these to the Observatory. The Observing Units are stored in the Observing Project Catalog, and the validated SBs (which are the only form of the Observing Units that can be scheduled) are also stored in (or pointed to be) the Observatory SB repository.

States

1. EDIT Program

The Observer enters all mandatory Phase II information and creates Observing Units, down to the level of Scheduling Blocks (Observing Scripts and initial and final calibration requirements). An Observing Unit may be either 1) an Observing Program (which itself includes an Observing Unit Set), including specifications for the order (if any) in which the Set is to be observed, and how the data from the Set is to be reduced, or 2) a Scheduling Block. Templates are provided all standard observing modes.

The Observing Unit creation process ends with the submission of a locally validated project to the Observatory.

Again, the OT provides a number of additional services to help the Observer with the creating of Observing Units, *e.g.*, array configuration, correlator setup, observing time required, anticipated data rates and data volumes.

2. SUBMITTED Program

The submitted Observing Units are validated by the Observatory and optionally reviewed by the ALMA Operations Staff.

The Observer is informed in case:

- the Project is accepted and stored;
- the Project is not accepted and needs further work.

3. APPROVED Program

The approved SBs are accessible from the SB repository, and waiting for scheduling and execution.

3.3 Additional Sequence Diagrams

During the process of creating the Sequence Diagrams, it became clear that an overall Executive process was needed to manage the various services (Dispatcher, Scheduler, Sequencer, Subarray Allocator, Pipeline, Error Monitor) that are needed to operate the observatory, so an additional Use Case, "Operate ALMA System," and the corresponding sequence diagram were generated. We then identified additional (mainly administrative) software services that need to be present to operate the ALMA system. Use Cases and Sequence Diagrams for "Manage ALMA Facility" and "Administer Observing Programs" were developed. The additional Use Cases are given in the Appendix, while their realization through sequence diagrams are presented in this section.

3.3.1 Sequence Diagram: Operate ALMA System

This is the top-level executive program in the ALMA Observing System [AOS]. It initializes, operates and supervises the interfaces, services, observations and hardware.

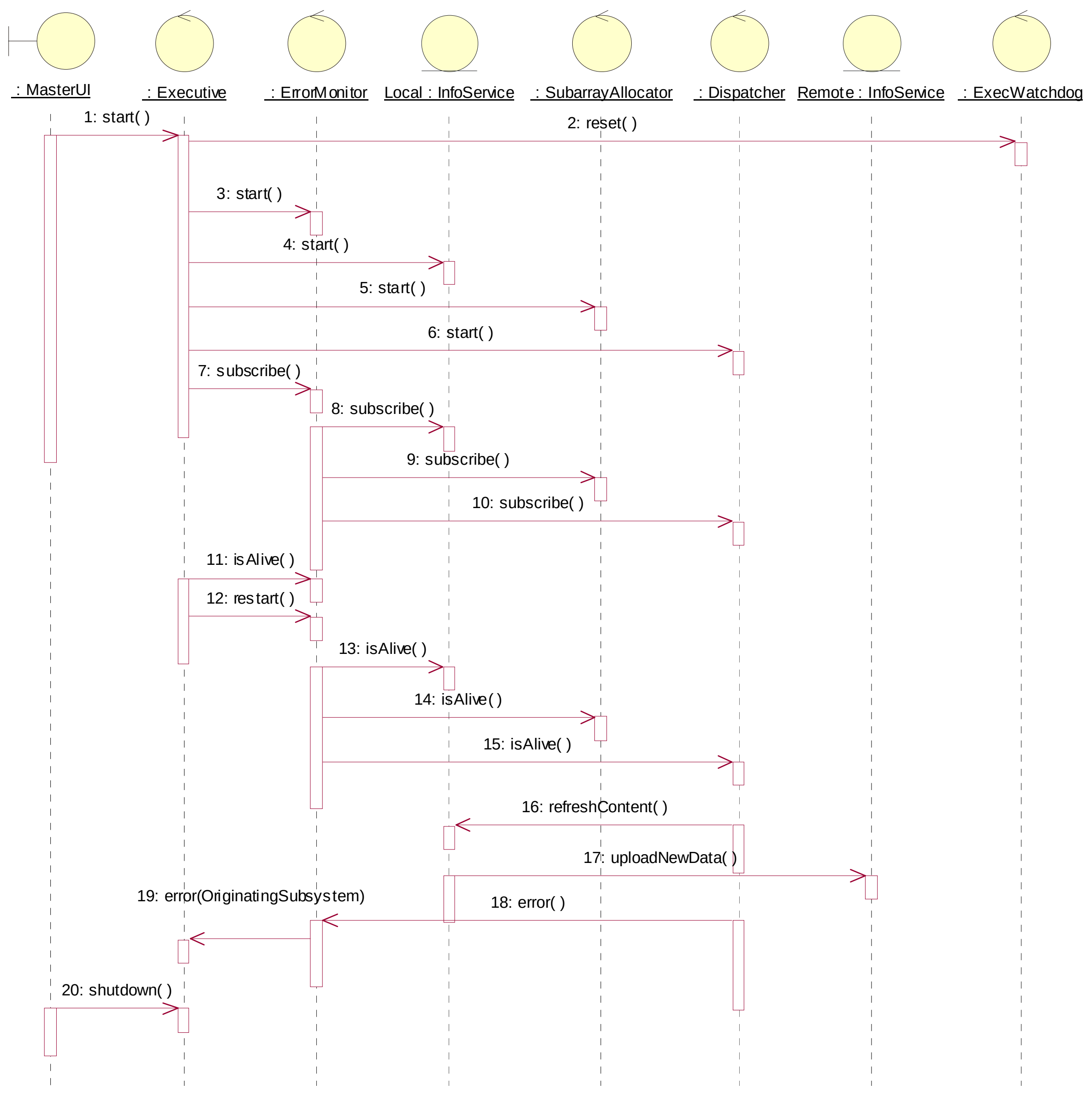


Figure 3-10: Operate ALMA System

1. The operator at the MasterUI instructs the Executive to begin operations: start().
2. Executive resets the ExecWatchdog.
3. Executive starts the Error Monitor.
4. Executive starts the local set of InfoServices.
5. Executive starts the SubarrayAllocator.
6. Executive starts the Dispatcher.
7. Executive subscribes to the notifications of the ErrorMonitor.
8. ErrorMonitor subscribes to the error notifications of the local InfoServices.
9. ErrorMonitor subscribes to the error notifications of the SubarrayAllocator.

10. ErrorMonitor subscribes to the error notifications of the Dispatcher.
11. Executive periodically polls the ErrorMonitor to check that it is still operating: isAlive().
12. If the value returned by isAlive() is false, the ErrorMonitor has died, and the Executive attempts to restart it: restart().
13. Executive periodically polls the local InfoServices to check that it is still operating: isAlive().
14. Executive periodically polls the SubarrayAllocator to check that it is still operating: isAlive().
15. Executive periodically polls the Dispatcher to check that it is still operating: isAlive().
16. When the Dispatcher finds the supply of SB's insufficient, it asks the local InfoServices to download more: refreshContent().
17. Local InfoServices instructs a remote version of InfoServices to upload new SB's: uploadNewData().
18. In case of an error encountered by the Dispatcher, it notifies the ErrorMonitor (via the subscription mechanism): error().
19. ErrorMonitor distributes the error notifications to Executive (again via the subscription mechanism): error().
20. After fifty or sixty years of successful operation the operator shuts down the Executive and brings the ALMA project to a conclusion: shutdown().

3.3.2 Sequence Diagram: Manage ALMA Facility

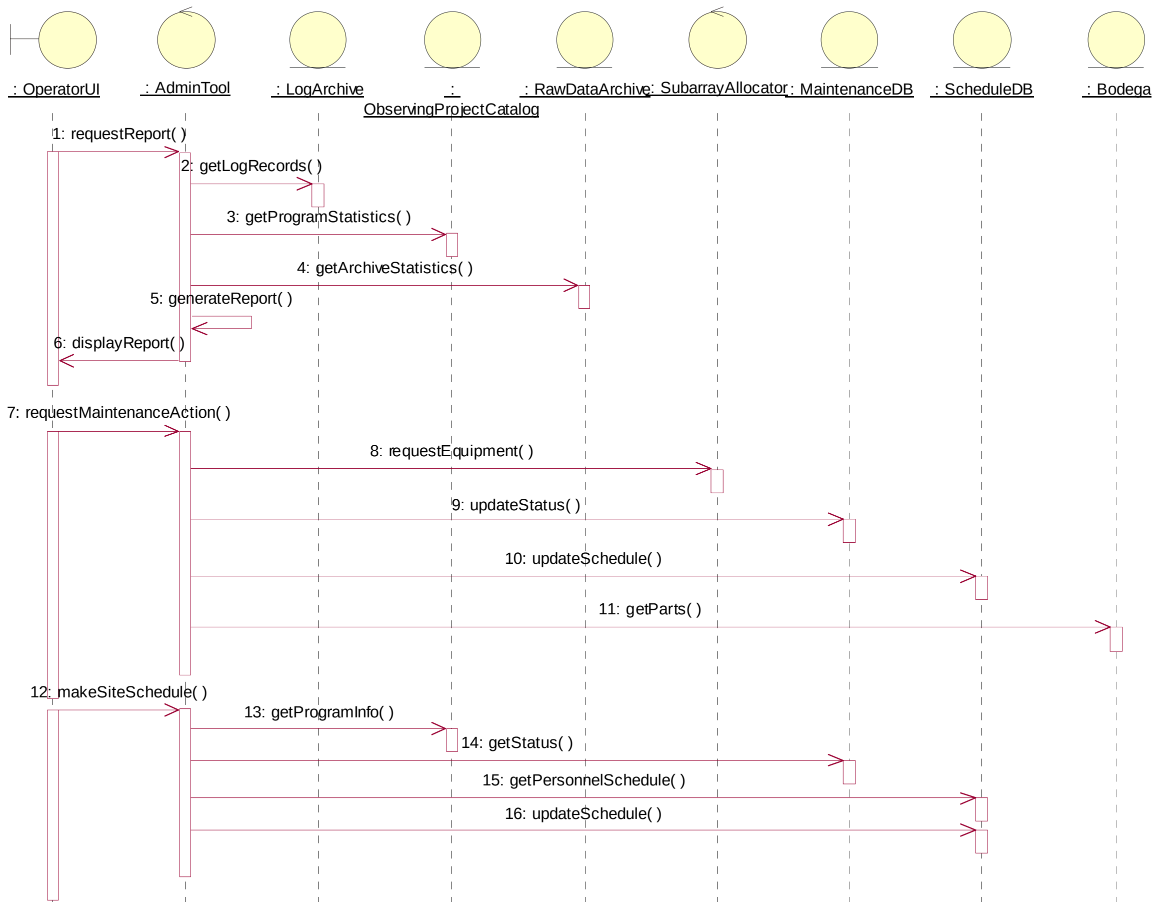


Figure 3-11: Manage ALMA Facility

1. The User requests a report containing information on the general performance of ALMA such as efficiency, number of SB's and Programs executed, reliability, etc. This may be done for a given period and for a specific set of subsystems.
2. All log records relating to the period and equipment in question are retrieved. They are then scanned for relevant information.
3. Statistics on all active Programs are obtained.
4. General information on data volume and access rates is retrieved from the Archive.
5. The information is collected into a report.
6. This report is displayed.
7. Such a report may suggest that certain maintenance actions be performed. The User judges if this should be done and submits a request.
8. The equipment to be placed under maintenance is requested to be reserved so that maintenance can start.
9. When the equipment is allocated to maintenance this and related information are recorded

in the Maintenance Database.

10. Changes in schedules due to the maintenance are made in the Schedule Database.
11. Spare parts are obtained from the Warehouse, which also is responsible for replenishing supplies.
12. The User requests a general long-term schedule to be made for the site. This may include array configurations, availability of staff etc.
13. The specifications of the Observing Programs to be scheduled are obtained.
14. The maintenance status of equipment is retrieved.
15. General information of availability of staff is obtained to ensure that personnel can perform the schedule.
16. The schedule is updated.

3.3.3 Sequence Diagram: Administer Observing Program

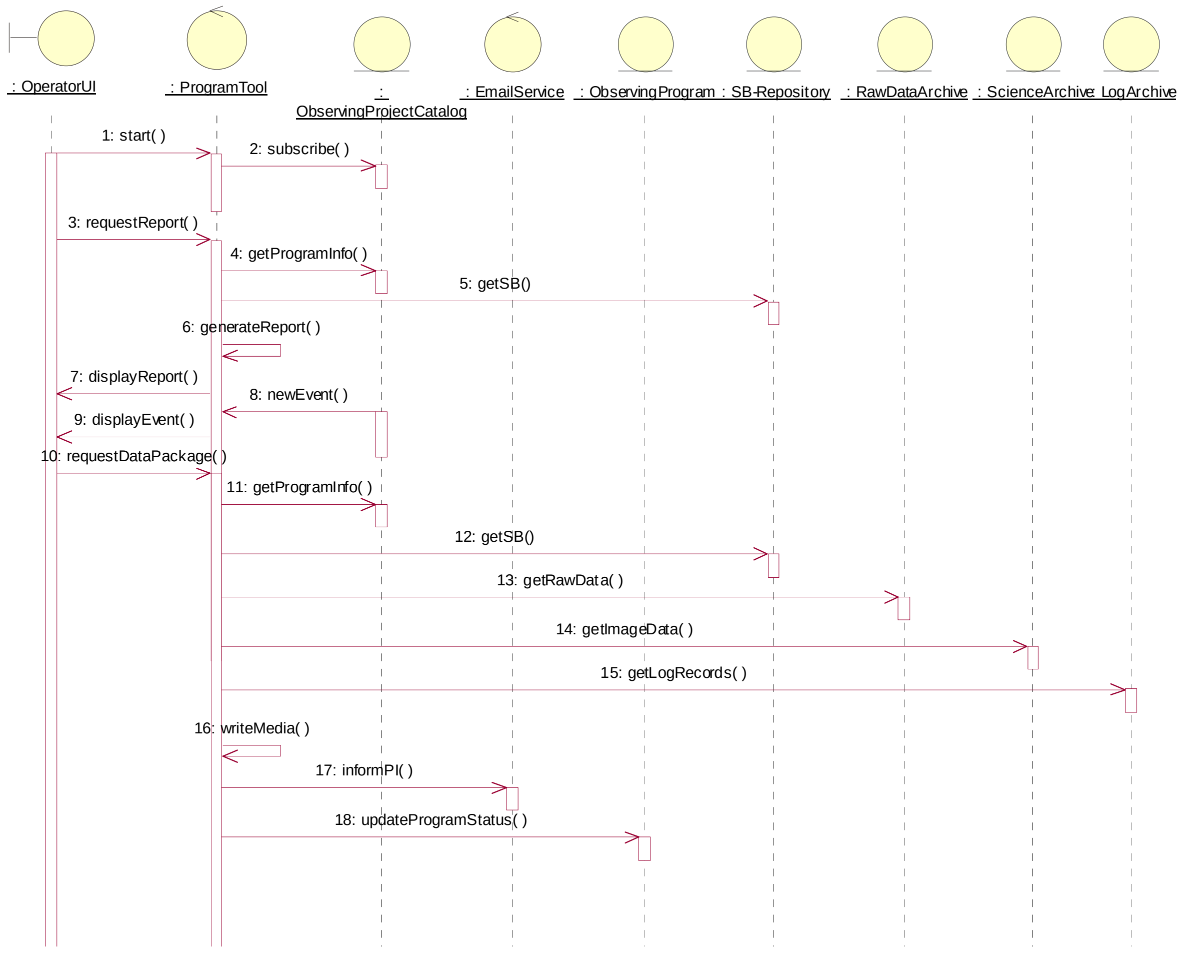


Figure 3-12: Administer Observing Program

1. The User starts the Program Administration Tool.
2. The Program Tool subscribes to the Observing Program Catalog in order to be informed a

Program changes its state e.g. reaches a breakpoint or become complete.

3. The User requests a general report of the current state of all active Observing Programs.
4. The Tool the status of each active Program.
5. From the Program information associated SB's are identified and they are obtained from the Repository.
6. With this information the Tool generates a complete report as specified by the User.
7. The report is displayed to the User.
8. A Observing Program has changed its state to one which require the PI to be informed and obtain data. As the Tool has subscribed to such event, it is informed.
9. The Tool communicates the event to the Operator.
10. If the Operator acknowledges a data package is generated and the PI informed.
11. The full information for the Program is retrieved from the Catalog.
12. All relevant SB's are obtained.
13. From Program and SB information, all associated data produced for the Program are identified. The raw data are retrieved from the Archive.
14. All processed image data are then retrieved.
15. All important log messages related to the Program are obtained from the Log Archive.
16. The information (*e.g.*, data, logs) is collected into a standard data package format. This package is then transferred to media that the PI can access. This could be media like DVDs, which can be shipped to the PI but may also be an ftp server that s/he has access to.
17. The PI is informed (*e.g.*, by e-mail) that the Program state has changed and related data are available. The system verifies (*e.g.*, by requiring a return receipt for the e-mail) that the notification has been received.
18. The Program status is updated to record that the PI has obtained access to the data.

4 Error Conditions & Handling Matrix

Error conditions of all types will of course be a fact of life for ALMA. In the accompanying matrix, we have begun to make an incomplete matrix of examples of the different kinds of things that can go wrong, who should be notified, what effect these faults are likely to have on ALMA as a whole, on the faulty subsystem, and on the further performing of Observing Programs. Guesses at the time necessary to get the system up and running again, as well as that necessary to provide a definitive fix for the fault are given. For example, a system may crash or hang because of a memory leak due to faulty software. While rebooting the affected computer may allow the system to continue to run for a certain period of time, finding the source of the memory leak and fixing it will take much longer.

Subsystems wishing to be informed of errors in another subsystem can subscribe to the other subsystem's error publishing mechanism. Only some errors can be signaled by the error handling system, however. Crashes or hangs of major subsystems will only be recognized if those subsystems are polled regularly, and such a procedure is foreseen in the "Operate ALMA" Use Case and Sequence Diagram presented later in this document.

		Notify														
Error Type	Severity	Op?	PI?	Staff astronomer?	Disable ALMA?	Disable subsystem?	SPR?	Log?`	Reboot?	Hold OB/SB?	Go to next pgm?	Blank?	Flag?	Time to resume ops	Time to fix	Subsystems to notify
Fire		Y	N		Y	Y	N	Y	N	Y	N					
Hardware																
Antenna alarm		Y	N		N	Y	N	Y	N	Maybe	Maybe	Y		~5 min	~3 days	
Receiver		Y	N		N	Y	N	Y	N	Maybe	Maybe	Y		~5min		
Correlator		Y	N		N	N	N	Y	N	Maybe	Maybe					
Computer																
Archive Computer		Y	N		Y		N	Y	Y	N	N			~5min		
Pipeline Computer		Y	N		N		N	Y	Y	N	N					
Scheduling Computer		Y	N		N		N	Y	Y	N	N					
Comm facilities		Y	N		Maybe	Maybe	N	Y	Maybe	N	N					
Active Components														~10min		
Passive Components																
Software																
Crash or hang		Y	N		N	N	Y	Y	Y	Maybe	Maybe			~5min		
SB times out		Y	Y		N	N	N	Y	N	Maybe	Maybe			~1min	days	
SB fails		Y	Y		N	N	Y	Y	N	Y	Y			~1min	days	
Pointing failure		Y	N		N	Maybe	Y	Y	N	Y	N					
Focus failure		Y	N		N	Maybe	Y	Y	N	Y	N					
Tsys out of bounds		Y	N		N	Maybe	N	Y	N							
Raw Data Archive		Y	N		Y	N	Y	Y	Maybe	N	N			~5 min	days	
Cal Pipeline																
Anomalous data		Y	N		N	N										
One-time (e.g., crash or memory leak)							Y	Y	Y	N	N	N	N	mins	days	
Design/implement'n		Y	N		N	Y	Y	Y	Maybe	Maybe	Maybe	N	N	?	weeks	
QL Pipeline		Y	?	Y	N	N	Y	Y	Maybe	Maybe	Maybe	N	N	mins	days	
Bad images																
System-detected																
QC-detected																
Proposal Preparation																
OT General																
Installation fails							Y								1-3 d	
Data can't be saved															hours	
Data can't be retrieved locally															hours	
Data corrupted															hours	
Wrong version															mins	
OT Subsystems																
Editor crash/hang							Y								hours	
Validator crash/hang							Y								hours	
Validator wrong res							Y								hours	
Data can't be sent															hours	
Correlator tool																
Simulator																
ALMA Proposal Ingest																
Mail system down															mins	
PI/PII data can't be stored															< 1 hr	
Receipt notify fails			Y				Y								< 1 hr	
Validator crash/hang							Y								1-2 hrs	
Valid. Report not sent			Y				Y								1-2 hrs	
Phase I/II Repository																
Full															mins	

5 Security considerations

In order to protect ALMA Operations from outside interference, whether well intentioned or malicious, we envision a set of basic access restrictions on interactions with the outside world. In particular, an Operations process must initiate any transfers into or out of databases that are in use by Operations. For example, when the Scheduler or Exec notices that its supply of SB's is low, it can request an update from, say, the ALMA Science Operations Center (exact name and location TBD, but we imagine it as being, say, in Santiago). Similarly, ALMA data is staged to mirror archives *outside* the charmed circle of Operations before it can be freely accessed; this policy applies to the use of proprietary as well as public domain data. Even the PI is not to access the online ALMA archive, which will presumably be located at the OSF.

In many cases information such as observing proposals, SB’s and project status will be exchanged between the ALMA observatory and external institutes and scientists via the Internet. To ensure that this information arrives without being corrupted or compromised, these exchanges must be protected and verifiable through the use of digital signatures and checksums; some may need to be encrypted. Since e-mail delivery is not 100% reliable, it must be possible to verify receipt when e-mail is used for information critical to either the Observatory or to the PI.

6 Architecture

Most of the major subsystems that emerge from the analysis are clients of some subsystems and servers for others. Sequencer objects, for example, could serve not only the Dispatcher, but also any simple engineering interface that can pass ASCII text (scripts or script commands). We can foresee that, particularly as the entire ALMA observatory is being implemented, each of these subsystems is likely to be implemented and tested, if not actually used, in a more-or-less standalone mode before the entire system is integrated. We have tried to minimize interference by one subsystem with the work of another: we have not yet seen the need to introduce interrupt mechanisms for the high-level software. One subsystem may subscribe to the service offered by another system, but these are usually subscriptions to a notification service; more substantial tasks, such as large data transfers, are always initiated by the subsystem needing the service.

We have tentatively defined where (*e.g.*, on what piece of computer hardware) each piece of the system will be located. The definition of minimal interfaces allows us to distribute the pieces across different platforms, and to use a communications scheme such as that provided by CORBA implementations. (This choice has already been made for the ALMA Common Software and the software for the ALMA Test Interferometer.)

6.1 Overall system flow

The following activity ("swimlane") diagram gives a schematic view of the general order of events in the life of an ALMA program, as well as the parts of the system responsible.

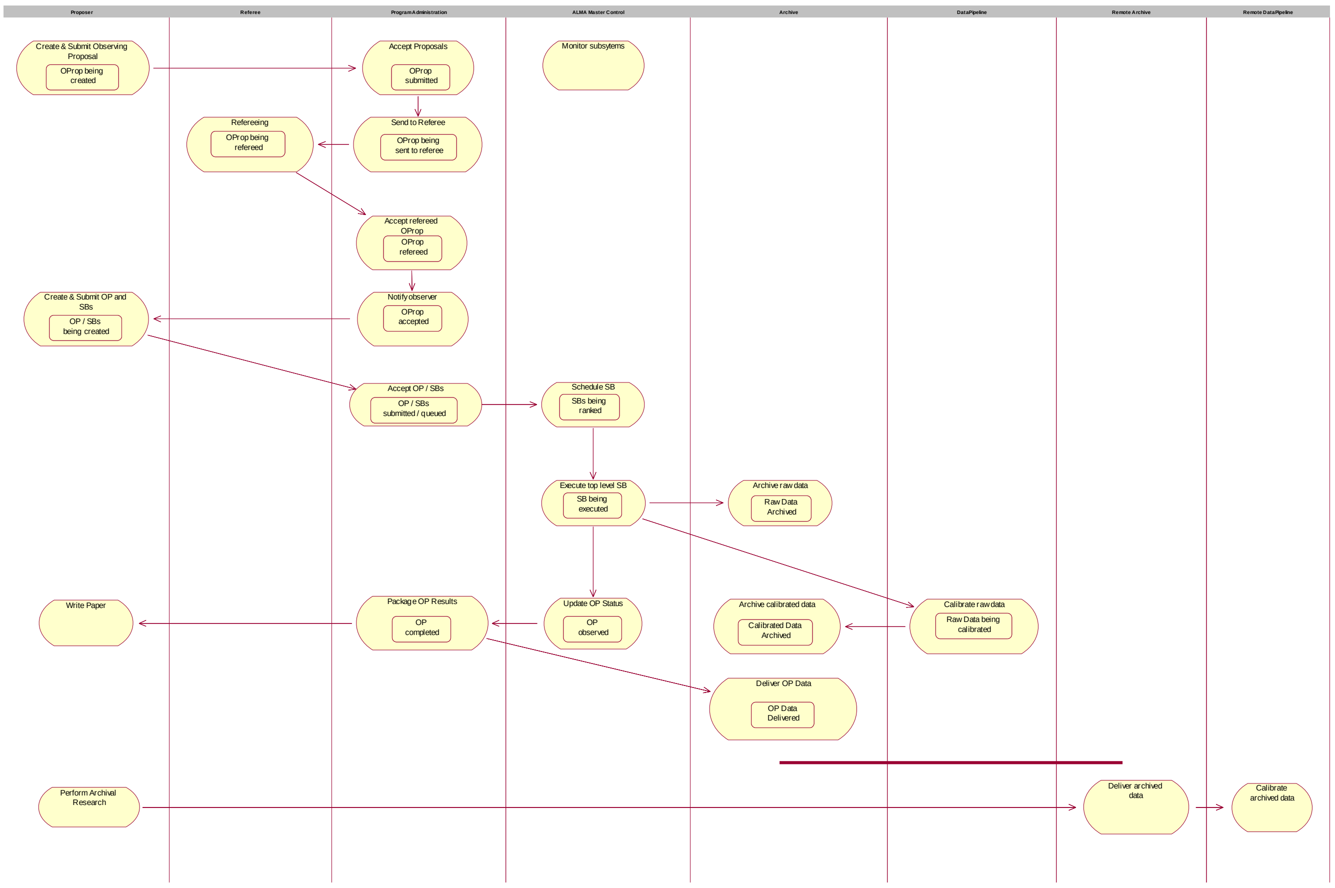


Figure 6-1: ALMA Swimlane Diagram

6.2 *Major services*

The principal services that have emerged from this analysis are:

- **Executive**: the basic supervisor process
- **Dispatcher & Scheduler**: the process responsible for determining which Observing Programme to perform next
- **Sequencer**: a script language processor
- **Subarray**: our abstraction for the ALMA hardware
- **Subarray Allocator**: a general ALMA resource allocator
- **Pipeline**: responsible for real-time (Calibration), not-so-real-time (Quicklook), and Science Data (production of final images and datacubes) Reduction
- **Error Monitor**: receiver and notifier for fault conditions

6.3 Deployment

We have made a first cut at allocating ALMA software to individual processors or nodes, and have defined ten general types of nodes or processors to understand deployment and communication issues. They are:

N1) **Proposer**: Used by the Proposer to prepare observing proposals for ALMA. It will often be external to the secure ALMA network (*e.g.*, because it is on the Proposer's laptop) and exist in several instances.

N2) **Referee**: Used by referees during their evaluation of proposals. It may be external to the secure ALMA network and exist in several instances.

N3) **ALMA Master Control**: Controls the ALMA facility and can therefore only exist in one instance. It is naturally in the secure ALMA network and accessed by the ALMA operator.

N4) **ALMA Administration**: Used by ALMA staff for administration of the general ALMA facility. The node can exist in several instances, but all must be on the secure ALMA network.

N5) **Program Administration**: Used by staff for performing tasks related to the administration of observing programs.

N6) **Data Processing**: Performs general pipeline processing of data. It can both be internal or external to ALMA.

N7) **Archive**: This type of node provides access to the InfoServices. It may exist in several instances but normally on the secure ALMA network.

N8) **Observer**: Used by observers (A3) to detail observation (i.e. to specify Scheduling Blocks) which then can be executed immediately after. This node is on the secure ALMA network and normally only in one instance.

N9) **Scientific User**: Used by scientific users to perform their work when interacting with ALMA data. It is normally external to ALMA and can exist in multiple instances.

N10) **ALMA System**: This is a generic node of the ALMA real-time system, *e.g.*, antenna, correlator, real-time computer.

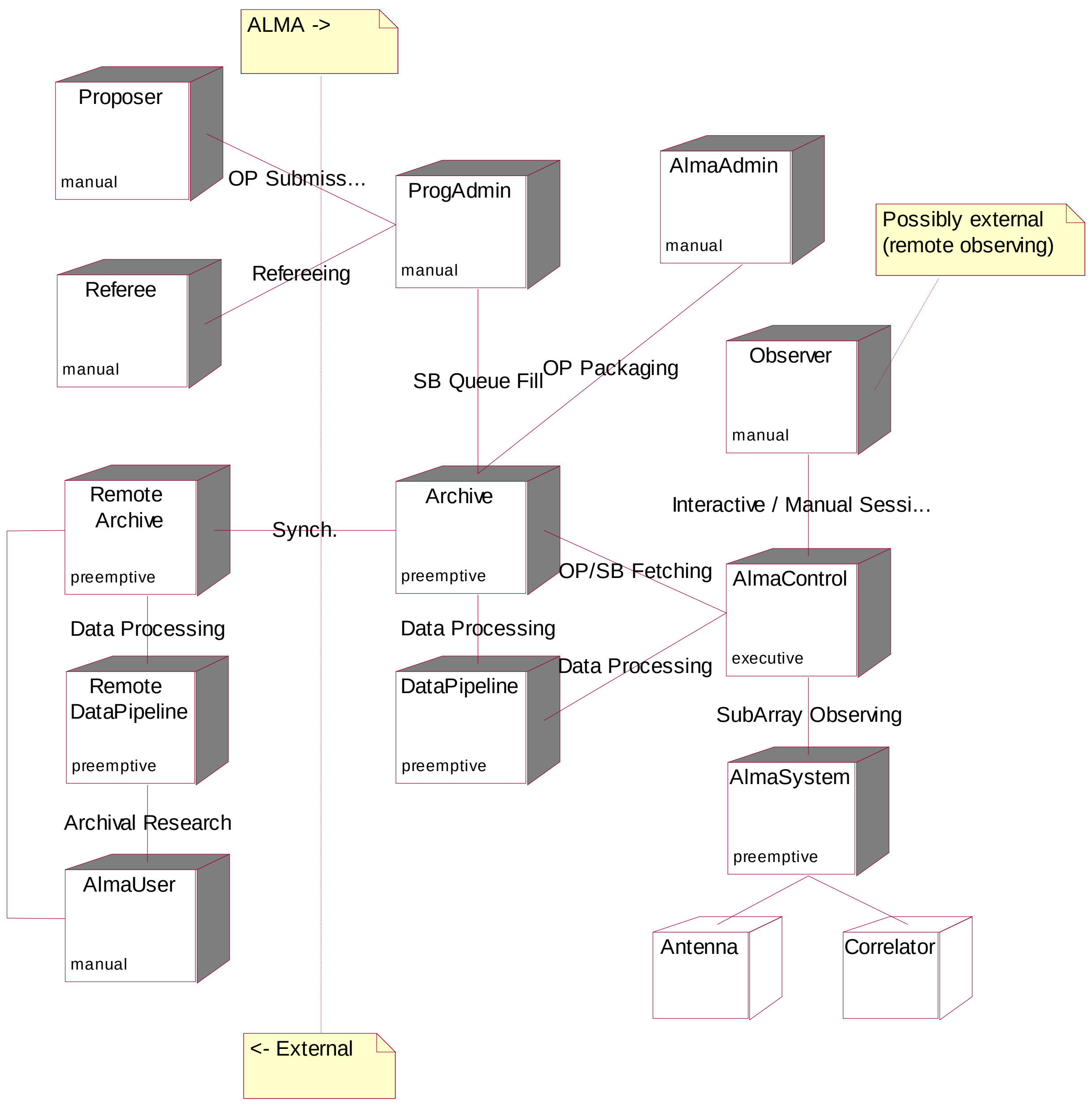


Figure 6-2: ALMA Deployment Nodes

7 What now ?

The work of analysis, the "refining and structuring" of the requirements, is far from complete. Obviously missing are diagrams and class definitions for archival research and the details of pipeline processing. The requirements themselves are continuing to evolve, and a mixture of further analysis, preliminary design, and prototyping of the major subsystems will be needed to determine how difficult it will be to realize them.

8 Operational Issues

The ALMA software development group will try to design a system that, by virtue of the flexibility of its basic structure, should be able to accommodate significant variations in the still yet-to-be-defined operational model for ALMA. Nevertheless, there are certain issues whose resolution in one way or another can have important impacts on the complexity of the resulting system, and therefore also on its cost.

a) Depending on the flexibility to be accorded to the Time Allocation Committee, the software may need to support the assignment of differing priorities and time allocations to pieces (Observing Units) of an Observing Project. This document has already defined the necessary conceptual infrastructure, but whether such differential assignment will actually be done is a policy issue.

b) Use Cases are needed for subarray examples, in particular for case of splitting array into two or more subarrays and the synchronization requirements for reuniting them.

c) There will be an effect on the design of the Dispatcher and Scheduler if "filler" programmes are to be scheduled when some antennas are not needed by the primary programme.

d) If the simulator is needed early to allow automatic vetting of non-standard scripts (which by hypothesis would be more frequent in the early years of ALMA operations), this will affect the order of software development.

e) A mechanism that authorizes proposers to submit non-standard scripts is probably needed.

f) The SSR believes that evaluation of a test source (*e.g.*, a calibrator) will provide sufficient data quality evaluation for standard modes. We haven’t seen this done in software before, and the effort needed is hard to predict.

g) If the scientific staff will have to adjust the array configuration schedule to reflect the needs of accepted programs, some support software will be needed.

h) Should stringent control of Phase I/II consistency be needed, supporting software of undefined complexity (undefined partly because explicit requirements are missing and we don’t know the scope of the work) will be needed.

9 Appendix: Revised Use Cases

9.1 *Use Case: Operate ALMA System*

This is the top-level executive program in the ALMA Observing System [AOS]. It initializes, operates and supervises the interfaces, services, observations and hardware.

Role(s)/Actor(s):
Primary: System Operators
Secondary: Observers, Staff Astronomers, Maintenance

Priority: The top priority process in the ALMA system.

Performance: Must be immediate and maximum. In addition, no outside programs or requests should be able to prevent or degrade the control system from operating the instrument.

Frequency: Continuous operation - Constantly used, available, *and checked*.

Preconditions: Initial hardware and software installed

1. The entire ALMA system is powered up
2. All major software and hardware devices are operational.

Basic Course: Startup and Normal operations

1. On the main ALMA control computer, the AOS executive program is started as a process, either automatically or manually. A progress and status display is maintained during this process for the operator and system monitor logs.
2. The executive program then starts:
    1. **System Initializer and Loader Processes:**
        1. Communications
        2. Hardware Initialization
    2. **Information Services**
        1. Equipment and Service Monitors and Logs
        2. Configuration Control
        3. Archives:
            1. Catalogs
            2. Calibrators
            3. Science and Image Data - Public

4. User Data:
    1. Status
    2. Proposals
    3. Programs
    4. SBs
    5. Science and Image Data - Private or Proprietary

3. **Processing Services**
    1. Calibrator Pipeline
    2. Quick Look Pipeline
    3. Imaging Pipeline
4. **Operator Interfaces**
    1. System and Equipment Status Displays
    2. Environmental Status Displays
    3. Schedule Status Displays
    4. Observation Status Displays
5. **Observation System Services:**
    1. Dispatcher
    2. Subarray Allocator
        1. Antenna Manager
        2. Correlator Manager

3. From the services and displays, the operator may start and stop any services, observations or the entire ALMA system.
4. The system is then operating in its standard mode - dynamically scheduled mode - automatic workload processing with dynamic scheduling.
5. The operator may also start other processes manually to perform system operation and maintenance, including calibrations, antenna relocation or backup operations. Or the operator may alter the mode of operation of the instrument as follows.

6. *Alternate Course: One or more of the observing subsystems is operated in interactive mode.* This mode provides for a guest or staff astronomer to directly control an observation through a GUI.
   The scheduler allows the operation of an observation activity from a user

terminal.
Script and other parameters in use may be altered by the observer at the terminal.
Postcondition: The system may be returned to any other mode.

7. *Alternate Course: One or more of the observing subsystems is* operated *in manual mode. This mode provides for direct control of the instrument through its command language.*
   Dynamic Observation Scheduling is suspended.
   All activity is initiated manually.
   Postcondition: The system may be returned to any other mode.

8. *Alternate Course: One or more of the observing subsystems is operated in technical mode. This mode is provided for engineers for debugging and maintenance purposes.*
   Dynamic Observation Scheduling is suspended.
   All user observation activity is suppressed.
   All activity is initiated manually.
   Postcondition: The system may be returned to any other mode.

Subflow: System Shutdown

1. The Scheduler can suspend operations under command or clock control, preventing any new observations from starting. Any existing observations may be allowed to run to completion or be manually preempted.
2. The Executive can terminate the ALMA processes enumerated above.
3. The system platform may then be halted and powered off.

Alternate Course: System Error Processing

1. Error Detection - At any time, the system may detect errors and require a response from the exec. Errors are detected by two methods: 1) Errors are detected by direct response from the command and control system to commands issued by the exec or related processes. 2) They are also detected by other monitoring processes which evaluate data being published by the equipment monitor stream. These error sources are two independent processes. Errors are classified according to severity and urgency. Some errors may be so catastrophic that these detection processes do not work. An example is the failure of the exec itself, the command and control system, or the monitor system. The instrument is unusable if this happens.
2. Minor System Problems - The exec is expected to diagnose, respond and recover automatically to these errors if observing parameters can be met. - These are errors which do not prohibit continued operations. They may range from a temporary loss of pointing or communications to the permanent loss of one or more antennas or image processing. They may also include software errors that can be temporarily fixed by restarts.
3. Major System Problems - The exec is expected to diagnose these errors and

await manual intervention. - These are severe errors that prohibit continued operations. They include failure of the control, communications, correlator and archiving system. This also includes software errors which are unfixable, even temporarily, by a restart. Obviously one of these major errors is the failure of the exec itself so a simple restart cannot compensate, even temporarily.

4. Recovery - Recovery from errors may range from:

    1. unnecessary because later processing compensates as in correlator output flagging, pointing flagging, etc.;
       [- exec remains on line. - Observation continues - No action by exec is necessary.]

    2. restarting of a failed component, possibly giving a temporary fix;
       [- exec remains on line. - Observation may be interrupted and resumed. - Some items such as a crashed program can be restarted by the exec, automatically or manually.]

    3. replacement of a failed component;
       [- exec may go offline. System , or part thereof, goes offline. - Time and a maintenance period is required.]

    4. fixing a design flaw in hardware or software.
       [no fix by the exec or operator is possible. Engineering development must handle this case, perhaps requiring scheduling observations around the problem until it is fixed. The exec is only useful to manipulate the instrument to perform manual intervention for component replacement or possible temporary manual actions to work around a failure.]

If the exec itself fails, recovery ranges from a restart of the top level processes wherein they recapture their orphan processes, or otherwise the entire system may require a restart.

Exception Course:

1. Instrument operation requires the exec for observations. Some maintenance activities may be performed in manual processes without the entire instrument operating.

Postconditions:

1. There are no postconditions. The system does not operate without the exec.

Issues to be Determined or Resolved:

1. Differences between Interactive and Manual modes.

9.2 Use Case: Manage ALMA Facility

The goal of this Use Case is to illustrate a set of general management tasks for the ALMA facility. They include the generation of reports on the status and performance of the facility, initiation of maintenance actions, verification of local supplies, and management of visitors and staff (*e.g.*, creation of schedules, reservations).

Role(s)/Actor(s):

Primary: System administrator, System Operator

Secondary: InfoServices

Priority: major

Performance: minutes

Frequency : several times per week

Preconditions :

1.Information Services are available.

Basic Course:

1. Actor specifies the period and ALMA systems for which a report should be generated.
2. Logs for this period are retrieved.
3. Log records are scanned and relevant information extracted.
4. A report is generated.
5. Potential maintenance tasks are initiated which include:
    - reserving the equipment for maintenance
    - scheduling personnel and time for maintenance
6. Verify if supplies are in stock and, if not, initiate orders of new supplies.
7. Establish a general plan for availability of antennas and their configurations considering information from Observing Program Catalog and maintenance schedule.
8. Generate schedule for interactive observation
9. Generate schedule for staff considering the schedule for interactive observations, calibration plan and preventive maintenance.
10. Allocate infrastructure resources to accommodate staff and visitors.

Postconditions:

1. Requested reports on the general ALMA status and performance are generated.
2. Required maintenance actions are taken.
3. Supplies are checked.
4. Plan for antenna and configurations is generated.
5. Staff and visitor schedules and associated arrangement are made.

9.3 Use Case: Administer Observing Programs

The purpose of this Use Case is to perform a set of tasks required to administer the execution of Observing Programs. This includes emission of e-mail to PI's at specific Program events (*e.g.*, reaching a Breakpoint), migration of user data, creation of final user data packages, and generation of reports on the status of Observing Programs.

Role(s)/Actor(s) :

Primary: System Operator, Staff Astronomer

Secondary: E-mailService, InfoServices

Priority : Major

Performance : minutes

Frequency : several times per day

Preconditions :

1. Observing Program Catalog is available.
2. SB Repository is available.
3. Raw Data and Science Archives are available.

Basic Course:

1. The user starts the tool which checks if e-mail service and communication are available and subscribes to Observing Program change event.
2. Status information for all active Programs is obtained from the Observing Program Catalog.
3. A status report for all active Programs is generated
4. The actor may explicitly initiate migration of data and e-mail messages to PI.
5. When Program change events are received the Program information is retrieved.
6. If a breakpoint is reached or similar event which requires that the PI is informed (e.g. Program complete), the following tasks are executed:
   - data and logs associated with the Program are identified
   - relevant data are migrated to a server to which the PI has access.
   - an e-mail is sent to the PI to inform her/him of the event and the availability of the data. Receipt of this e-mail by the PI is verified by the system.
7. Status of Program is updated.

Postconditions:

1. PI is informed by e-mail of changes in the Observing Program status.

2. Relevant user data are made available to PI
3. Requested reports on the status of Observing Programs are generated.

Issues to be Determined or Resolved:

1. Policy: at which Program status changes should a PI be informed?
2. Policy: when can a PI request data?

9.4 Use Case: Dispatch Scheduling Block

The goal of this use case is to obtain a ranked list of Scheduling Blocks (SBs) from the Dynamic Scheduler or from an interactive observer (via the Observing Tool) and pass this to the Sequencer. When a group of SBs from the same Observing Programme is executed contiguously, an observing session is said to have been executed. The system may initiate some data processing activities at the end of a session.

Role(s)/Actor(s):
Primary: Operator, Scheduler, Sequencer
Secondary:

Priority: Critical

Performance: Seconds to hours

Frequency: Several times per minute/hour/day; One at a time per Sub-Array

Preconditions:

1. Need to have Scheduling Block(s) available in the Phase II SB Repository

Basic Course:

1. System requests a priority-ordered list of SBs from the Dynamic Scheduler in either "atomic" or "snapshot" dynamic scheduling mode. (see UC_ScheduleSB)

   Alternate course: System requests an SB from the Observing Tool in Interactive mode.

2. The system displays this list to the Operator, who can choose to override it, moving a different SB to the top of the list and furnishing a reason for this decision to the log.
3. The system passes the SB at the top of the list to the Sequencer for execution
4. Upon return from the Sequencer, the system updates persistent SB / OP (Observing Program) parameters that have changed and saves SB / OP

status

5. The system repeats Steps #1 and 2. If the newly selected SB requires that the previous session be ended (because it is from a different Observing Programme, for example), the system initiates the appropriate session-processing activities on the Science Data Pipeline.

6. The system loops on Steps #3, #4 and #5 as long as the Scheduler or the Observer can furnish an executable SB.

Exception Course:

No more SBs are available to be scheduled.

1. Stop SB execution
2. Notify operator / observer; request repopulation of SB-Repository

Postcondition: System waiting for input of SB or observing commands

Postconditions:

1. SB's have been successfully dispatched

Issues to be Determined or Resolved:

- Which calibrations can be shared across SBs from different programs?
- What happens in snapshot mode when conditions change rapidly?

Notes:

- Must have access to persistent program parameters.

Owner: Joseph Schwarz
`Last updated by $Author: jschwarz $ on $Date: 2001/06/21 11:19 $`

9.5 Use Case: ScheduleSB (Revised)

Retrieve SBs from Phase II SB Repository, assign priorities to Observing Programme SBs and return prioritized list of SBs to the Dispatcher.

The Phase II Repository for a given Programme contains SBs as well as their associated configuration and calibration requirements. The Scheduler will take account of the time required to bring the array to the necessary calibration state when assigning a rank to each SB.

In "local scheduling mode", the Scheduler will consider each SB independently. In "global scheduling mode", on the other hand, the Scheduler will attempt to look ahead, building a queue of SBs -- possibly from different Programmes -- that can share a significant (in terms of observing time needed) amount of calibration operations. The main goal here is to accommodate "snapshot" programmes, short observing programmes that would be inefficient to schedule independently because of their relatively high calibration time-to-target time ratios.

The Programme may contain relational links between SBs, in the sense that a given SB may only be scheduled if specified other SBs have been previously executed, and if some condition on their results (as indicated by the Observing Programme's status) is fulfilled.

The programme may contain Breakpoints, *i.e.* conditions in the Observing Programme's status that will inhibit further execution of SBs in that Programme, pending release of the Breakpoint by the Observer.

Role(s)/Actor(s):
Primary: Dispatcher
Secondary: Phase II Repository, Array Observing System

Priority: major

Performance: order of seconds

Frequency: order of minutes

Preconditions:

1. The Repository of active Programmes from Phase II

Basic Course:

1. The system determines the current array configuration, in particular, that part that is available for use
2. The system determines the current observing conditions
3. The system acquires all Phase II SBs that can be run with the current:
    1. Array configuration
    2. Observing conditions

3. LST range

and that:

4. fulfill any conditions imposed by their Programme (relational links between SBs).

5. are not on hold because of a breakpoint.

4. The system determines the starting calibration requirements of each ready-to-run SB and the time necessary to fulfill them.

5. System calculates SB priorities based on rules involving:

    1. Initial scientific priority rating.

    2. Environmental parameters (weather, LST, UT, ...)

    3. System parameters (is the Programme started, is it currently in execution, ...)

    4. Pipeline results (current phase rms if available from calibrators, possibly science results,...)

    5. Time to execute SB and all necessary calibrations.

6. System returns priority-ordered list of SBs.

7. Whenever an SB makes it into the list of (TBD) ten top ranked ones or the list of SBs likely to be executed within the next 24 (TBD) hours, an e-mail is sent to the PI.

Alternate Course: "Global scheduling mode"

1. The system matches SBs that can share time-consuming calibration operations and constructs separate queues for groups of these.

2. The system returns these queues

Postconditions:

1. SBs are passed to the Dispatcher.

Issues to be Determined or Resolved:

1. The actual set of rules to calculate priorities.

2. How to resolve the conflict between the "local" way of ranking SBs (considering each one individually) and the "global" scheduling mode (where the assumption is made that the observing conditions will remain constant enough to allow more than one SB to be executed using a common set of calibrations).

3. When calibrations are shared among programmes, their

time cost should not be entirely attributed to the first SB, but shared with the SBs in the repository that would benefit from them (in fact only those that may be executed during the validity period of that calibration). That might be difficult to compute. A policy decision to charge time for shared calibrations to the observatory rather than to individual observers might be worthwhile for the simplification it would bring, and for the incentive it might give observers to propose "snapshot" observations.

Notes:
Owner: Robert Lucas
`Last modified by $Author: jschwarz $ on $Date: 2001/06/28 10:57 $`

9.6 Use Case: Execute Scheduling Block (Revised)

The goal of this use case is to execute Scheduling Blocks (SBs) that have been scheduled by the Dynamic Scheduler. SBs are the building blocks of Observing Programs (OPs). They include descriptions of configurations and the initial and final calibrations needed for an Observing Session (an uninterrupted sequence of SBs sharing the same calibration requirements). SBs are the smallest units to which the Dynamic Scheduler assigns priorities. The Dispatcher sends the SB with the highest priority (previously assigned by the Scheduler) to the Sequencer for execution on a Sub-Array. Alternatively, in Interactive Mode, SBs are sent directly from the Observing Tool to the Sequencer. Each SB includes all the project-level calibrations necessary for its output to be processed autonomously. Calibrations that have been performed prior to this SB but that are still valid for this SB's desired hardware configuration/setup are used and not repeated unnecessarily.

Role(s)/Actor(s):
Primary: Dispatcher, Sub-Array (all hardware available to this SB)
Secondary:

Priority: Critical

Performance: Seconds to hours

Frequency: Several times per minute/hour/day; One at a time per Subarray

Preconditions:

1. Need to have Scheduling Block(s) from the Dispatcher

Basic Course:

1. Dispatcher sends SB (see UC_DispatchSB)
2. Perform initial setup and calibration operations. If existing calibrations are still valid for this setup, do not repeat them.
    1. Perform necessary system initializations and/or calibrations, *e.g.*, a bandpass calibration (see UC_ Observe_Interferometric_AstroBandpassCal)
    2. Warn the PI that the observations are started if a previous warning has not been sent within the last 96 (TBD) hours for the same programme.
3. Execute standard Scans by interpreting the corresponding Observing Scripts with the given user parameters (Observation Descriptors) and controlling the antennas, the receiver and the correlator accordingly

   Alternate course 1: For non-standard scan modes interpret the user

supplied Observing Script
Alternate course 2: Alternatively, in manual mode, the user types in commands to be executed directly via a Command Line Interface (CLI).
Exception course: An existing calibration becomes invalid, either because its validity has expired, or because a change of hardware configuration has made it inapplicable. In either case, perform the necessary calibration and proceed.
Exception course: The execution of an observation fails

4. Archive data with time and project tags continuously, *i.e.*, while an observation is being executed (see UC_ArchiveData)
5. For standard observing modes send standard Reduction Script to Calibration Pipeline (see UC_ProcessCalibrations)
6. Perform final calibrations necessary to complete SB; if this SB turns out not to be the last one in a session, these calibrations will still be valid and will not be repeated when the next SB begins execution.
7. Return status to caller (usually the Dispatcher).

Exception Course:

The execution of an observation fails

1. Stop SB execution
2. Notify operator / observer; save status of OP

Postcondition: Execution of SB halted; operator / observer notified; status saved

Postconditions:

1. SB has been successfully executed

Issues to be Determined or Resolved:

- Which calibrations can be shared across SBs from different programs?
- The Observing Tool must know about the possible default scan modes and the necessary Observing Descriptors.

Notes:

- Must have access to persistent program parameters.
- Interactive observing will be setup via SBs that will be directly transmitted to the Sequencer.

Owner: Dirk Muders
`Last updated by $Author: jschwarz $ on $Date: 2001/06/28 11:19 $`