- Learning Goals
- Legacy Software
- Requirements
  - Actors
  - Components breakdown and responsibilities
  - Interfaces
  - Communication Diagrams
    - Proposal lifecycle
    - Automatic Proposal Execution
  - Success requirement
- Development Lifecycle

The main goal of this project is to develop a complete example of a working end-to-end system built on top of ACS in the five days time frame allocated for the workshop course. The software development cycle will include the system integration and tests.

Some of the ACS concepts illustrated with this project are:

- Component/Container model
- ACS Services: Logging and Error systems
- ACS Components simulation
- Language transparency and interoperability
- Distributed deployment
- Test driven development (unit testing)
- Continuous Integration

The development will have a fixed deadline (last day of the course) and will be distributed in several teams. De-scoping will be allowed if needed to meet the deadline. Interface changes will have to be negotiated between the interesting parties and informed in due time to the rest of the developers.

The following components coming from external projects are available and will be used. They are all located inside the `AWV/EXTERNAL` GIT repository.

- Camera (real, remote)
- Telescope DevIO (real, remote)

Actors

There are two main external actors for this software project: Astronomers and Operators.

- **Astronomers** will interact with the system through a Database component. They should be able to store a proposal, to query for the status of a given proposal, and to retrieve the proposal observations once the proposal has been executed.
- **Operators** will interact with the system through a Console component. They should be able to start and stop automatic observations, and to get direct access to low level components for maintenance purposes.

Components breakdown and responsibilities

The system will be composed of 5 components. They will have the following responsibilities (top-down):

- **Console:** This is the system entry point for the Operators. It allows the Operator to start/stop the Scheduler's automatic mode, and provides them manual access to the low level components. It should provide a component that implements all methods, and a TUI client to access those methods.
- **Database:** This is the system entry point for the Astronomers. Besides allowing an astronomer to store a target list, query for the status and retrieve the proposal observations, it provides methods to get the proposals currently inserted into the Database, to set a status to a given proposal and to insert a given observation into the Database.
  - A *Proposal* consists of one *TargetList* (which is a list of one or more *Target*), an identifier and a status (0 - queued, 1 - running, 2 - ready). A unique identifier is assigned by the database component and returned to the client after storing its *TargetList*.
  - A *Target* consists of a *Position* specification, an exposure time and a target identifier assigned by the astronomer. The target identifiers should be unique per proposal.
  - A *Position* is simply the telescope position to be reached for that observation. It has two coordinates: elevation *el* and azimuth *az*.
- **Scheduler:** The Scheduler is responsible to select a proposal from the Database, execute it, store the observations, and to manage the proposal lifecycle. The observations are scheduled automatically according to some scheduling algorithm as soon as the scheduler is requested to start. On stop it will complete the proposal before suspending the automatic mode.
- **Telescope:** This component communicates with the low level hardware access layer, and executes an observation; i.e. moves the telescope to a given position and acquires the image from the Instrument for a given exposure time once the telescope is in position.
- **Instrument:** This component sets the CCD camera on and off, and it allows to take an image with a given exposure time.

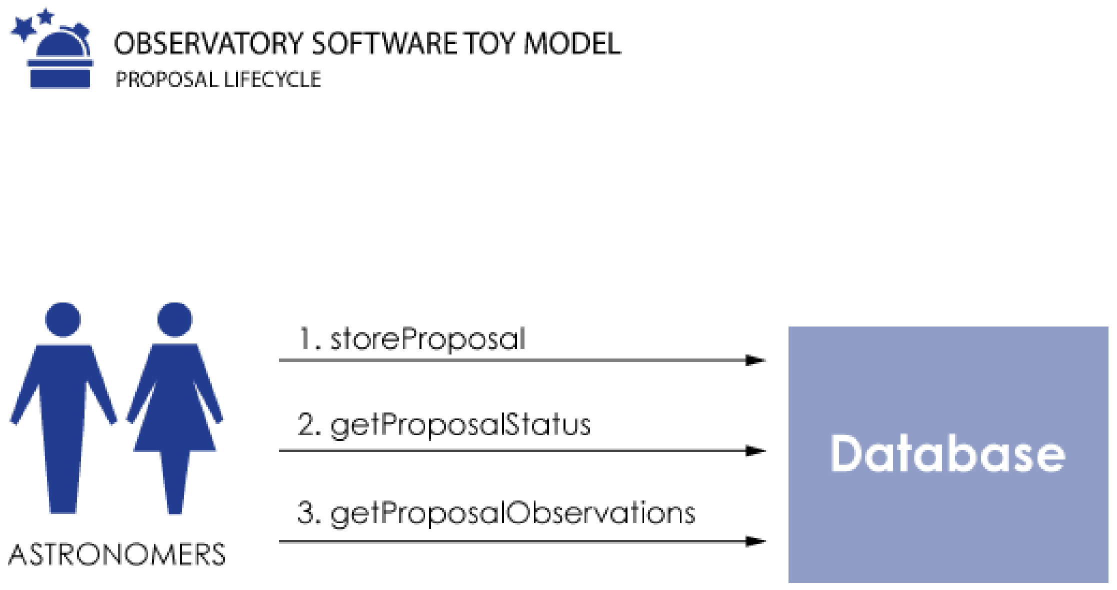
Interfaces

The initial interfaces and data types are defined and documented under `AWV/ICD/idl`. They could eventually be modified during the development cycle, but this will have to be negotiated on a case by case basis between the interested parties and will have to be noted down in a change request publicly available.

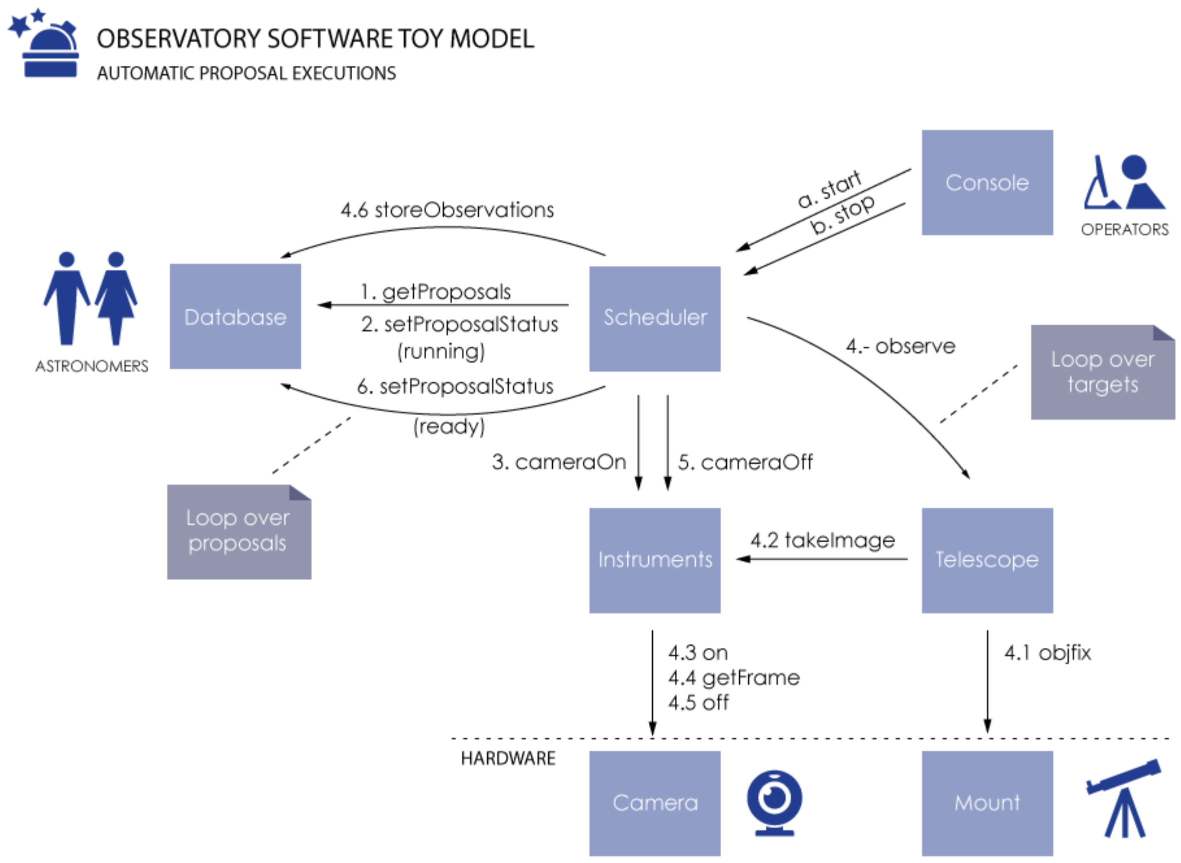
Communication Diagrams

The following two communication diagrams illustrate the two main use cases to be implemented for this project.

Proposal lifecycle



Automatic Proposal Execution



Success requirement

To declare the project a success the software should be able to perform an automatic observation of at least one proposal composed of one target.

The project will be guided in simultaneous steps to be developed by the teams defined below. The development lifecycle is test-driven and is defined as follows. The project will be continuously integrated by the Integration, Test & Support team, and it is the responsibility of each development team to have working versions in GIT.

The basic preparation considers the following steps:

1. **Development environment:** Create (empty) development module and do initial import to GIT.
2. **ACS environment:** Set up the configuration database for the ACS environment and start/stop the system and (empty) component.
3. **Components simulation:** Set up simulated components for each needed interaction. This is important to test each component, as other components are also under development.
4. **Unit tests development:** Create unit tests within a given TAT environment, to test the component input/output and functionality.

These steps will have to be iterated several times by each development team:

1. **Component functionality development:** Implement the component internal logic and functionality.
2. **Adding logging and error handling:** Add log messages and exception handling to the component.
3. **Individual component v/s simulation interaction testing:** Test the component's interaction with other components against local simulations.
4. **Bug fixes?**
5. **Component v/s component interaction testing:** Test component's interaction with other components against other development teams' real implementations.
6. **Bug fixes?**

These steps will be prepared by ITS during the entire week, and concluded on the last day with support by all development teams:

1. **End-to-end integration and testing:** Overall system deployment, integration and use case testing.
2. **Bug fixes?**

Continuous integration is an important part of the project lifecycle. A Jenkins build and test server has been set up to automatically manage the project upon GIT commits. The following is the URL: http://138.68.61.143:8080/

The Git repository can be found at: https://bitbucket.alma.cl/projects/ACSWS/repos/awv/browse