Preprocessing stage of data mining for ACS logs

FINAL REPORT

Javier Fuentes Muñoz

Internship, Alma Observatory

`https://github.com/javierivanov/ObsSM`

This work was done under the supervision of Tzu-Chiang Shen TZU-CHIANG.SHEN@ALMA.CL and Juan Pablo Gil JUAN.GIL@ALMA.CL within a total of 24 weeks, from January 18th to June 18th of 2016.

Contents

1. Introduction

1.1 Project Overview

The objective of this project is to find an approach useful enough to reduce the amount of log data produced during an observation. In order to narrow the project, it was decided to analyze only the Observing Modes package, which is part of CONTROL subsystem. The actual state in ALMA related to logs is interesting for this project. Nowadays logs are sent to Elastic Search, which is a big database with a quick response to specific queries, being more efficient in data retrieval work

1.1.1 Duration and Location

This internship is designed to have a duration of 6 months. The starting date is Monday, January 18th, 2016. The student will need full dedication to the project during the whole period. The main location will be ALMA Santiago Central Offices (SCO) with a potential trip to the Observatory site.

1.1.2 Resources

One student fully dedicated for the whole internship period, with administrative supervision of Tzu-Chiang Shen and technical supervision by Juan Pablo Gil.

1.2 Objective

1.2.1 Goal

- Document the observing mode process by using a state machine model fed through software logs.

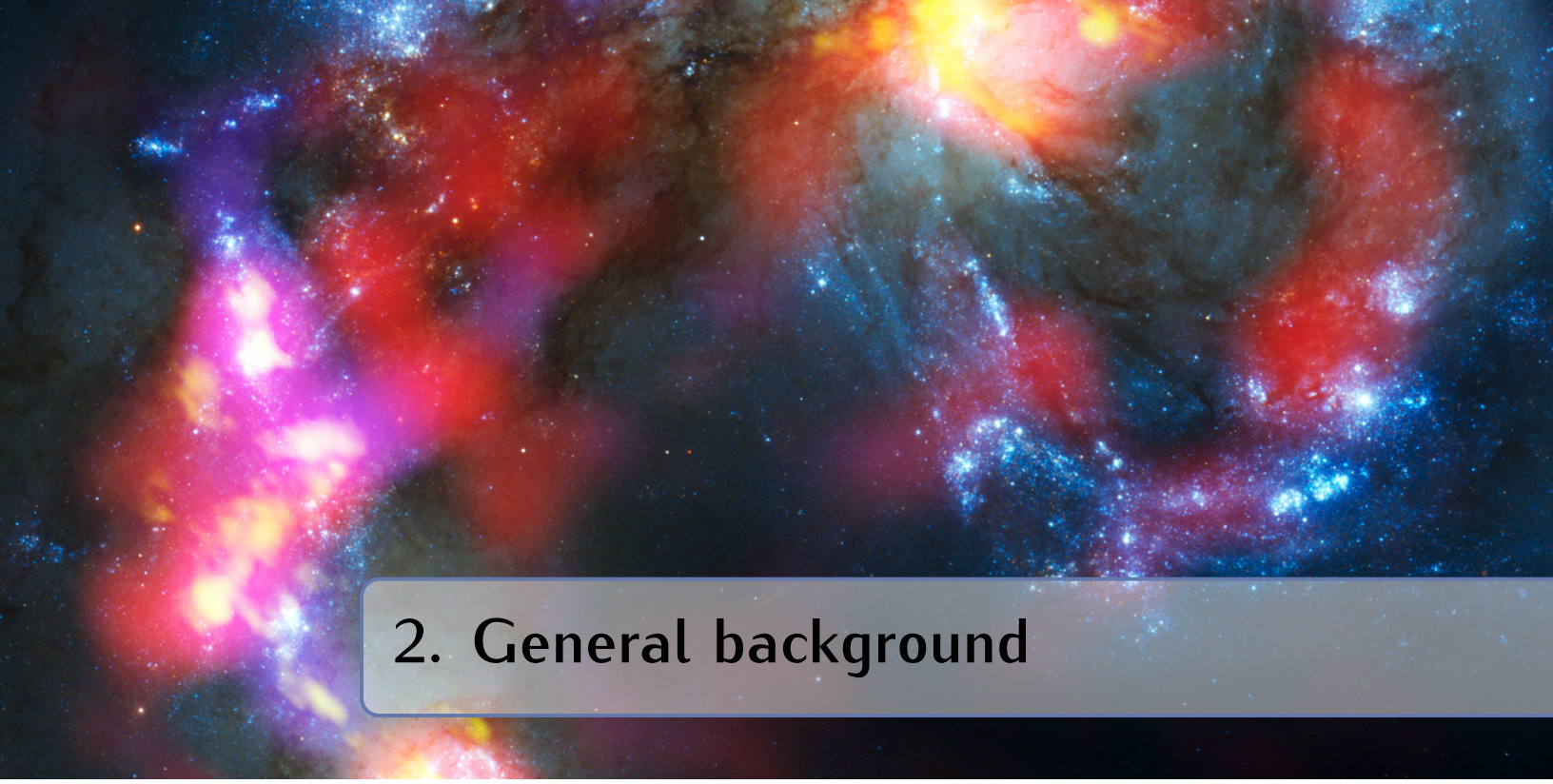
Sub Goals

- Generate detailed documentation regarding ALMA's observing modes to be used by future work. This must include at least a report with relevant logs, the observation process model and tools.
- Model a state machine based on logs that reproduce the observation process. This must include high level processes as well as the result of hardware configuration stages, but not including hardware inner work.
- Develop a tool that is able to process the model and interpret the log, generating an input for a possible learning machine.

1.2.2 Project milestones

#	Milestone/Deliverable	End Date	Status
1	Defining objectives and planning	20th Jan	DONE
2	Studying kibana, logtash and elasticsearch (summary)	22th Jan	DONE
3	Definition of the problem	29th Jan	DONE
4	Analyze the observation process.	15th Feb	DONE
5	Search, compare(Comparative table) and define the Implementation of the state machine model and the standard to use.	5th Feb	DONE
6	Develop a prototype for Logs Interpretation using a SM Model	26th Feb	DONE
7	Develop a prototype to identify transitions through the logs, generating a SM Model	10th March	DONE
8	Develop stable releases of created prototypes (packages and testing)	1st April	DONE
9	Integrate both applications with their respective documentation (package and testing)	12th April	DONE
10	Increase the complexity of the Model (More details are needed)	10th Jun	DONE
11	Documenting information and recommendations about logs into the code	10th May	DONE
12	Test the model and applications	10th June	DONE
13	Model documentation scxml and instructions to use it.	10th June	DONE
14	Finals tasks	17th June	DONE

Table 1.1: Milestone and Deliverable



2. General background

2.1 Finite State Machines (FSM)

There are many ways of modeling the behavior of systems, and the use of state machines is one of the oldest and best known. State machines allow us to think about the "state" of a system at a particular point in time and characterize the behavior of the system based on that state. The use of this modeling technique is not limited to the development of software systems. In fact, the idea of state-based behavior can be traced back to the earliest considerations of physical matter.

2.1.1 Mathematical Definition of a Finite State Machine

A finite automaton M is defined by a 5-tuple $(\sum, Q, q_0, F, \gamma)$, where:

- $\sum$ is the set of symbols representing input to M
- Q is the set of states of M
- $q_0 \in Q$ is the start state of M
- $F \subseteq Q$ is the set of final states of M
- $\gamma : Q \times \sum \to Q$ is the transition function.

2.2 State Chart extensible Markup Language: SCXML

This section outlines State Chart XML (SCXML), which is a general-purpose event-based state machine language.

SCXML provides a generic state-machine based execution environment based on CCXML and Harel State Tables.

The most basic state-machine concepts which are required on this work are: *state*, *transition* and *event*. Each state contains a set of transitions that define how it reacts

to events. Events can be generated by the state machine itself or by external entities. In a traditional state machine, the machine is always in a single state. This state is called the active state. When an event occurs, the state machine checks the transitions that are defined in the active state. If it finds one that matches the event, it moves from the active state to the state specified by the transition (called the "target" of the transition.) Thus the target state becomes the new active state.

Basic example:

```
<state id="idle">
  <transition event="array.creation" target="ArrayCreatedState"> </transition>
</state>

<state id="ArrayCreatedState">
  <transition event="interferometry.start" target="InterferometryStartedState">
    </transition>
  <transition event="array.destruction" target="ArrayDestroyedState"> </
    transition>
  <transition event="singledish.start" target="SingledishStartedState"> </
    transition>
</state>
```

Listing 2.1: SCXML Example

- For further information please see: `https://www.w3.org/TR/scxml/`
- SCXML Visualization and editing software: `https://github.com/fmorbini/scxmlgui`

2.3 Apache Commons SCXML

The Apache Commons SCXML libraries provides a generic state machine execution environment. The following are some features:

- Reads a SCXML file.
- Checks the model.
- Executes the events.
- Allows following the execution flow.
- Written in Java

3. Work Description

The next sections outline the results of this project. As a summary as you can see in Figure 3.1, it aims to explain the complete model. There are 3 different systems, ALMA-ELK, which is the current structure for handling the log search and archiving. The two others systems are Discovery and ObsSM, which are a part of the results of this work. Their objectives are to generate transition models in the case of Discovery and interpret logs using the generated models in the case of ObsSM.

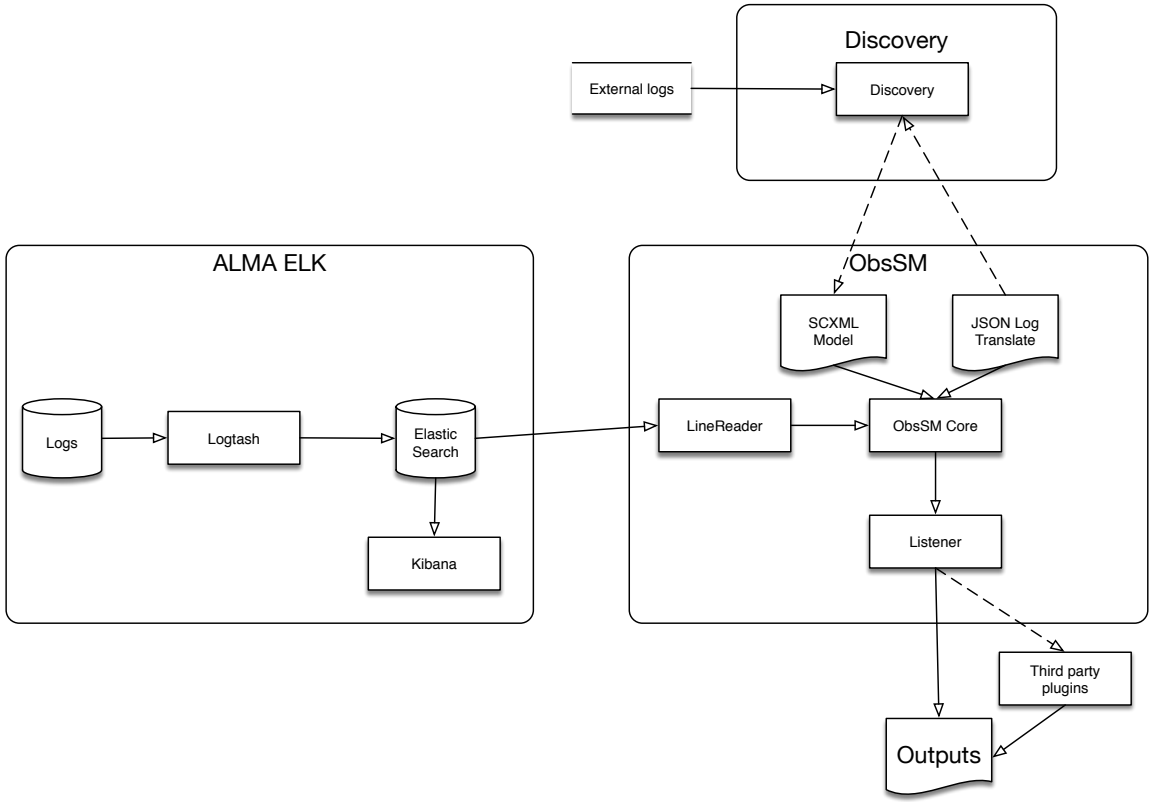


Figure 3.1: Data Flow

3.1 Observing Mode Analysis

This documentation was obtained through conversations with software engineers, documents of the control system and reading the source code. ACS is a big software product and as others, documentation about processes is not the best. For that reason it was important to generate some useful kind of document, to capture the logic behind an observation.

The first issue was to identify a good kind of diagram to explain how the Observing Mode works. This question was answered by the Data Capturer documentation. In this documentation it is possible to see three important diagrams: a sequence diagram, a dependency diagram and a state machine diagram.

In Figure 3.2 you can see the activity diagram for an Interferometry observation.

In Figure 3.3 you can see the dependency diagram for the ObservingMode package

3.2 SCXML Model

This model was designed for the Observing Mode package. The model is based directly on the logs and the relation to methods and classes, the idea is to understand the logic behind the Observing Mode and with this relation between states and log

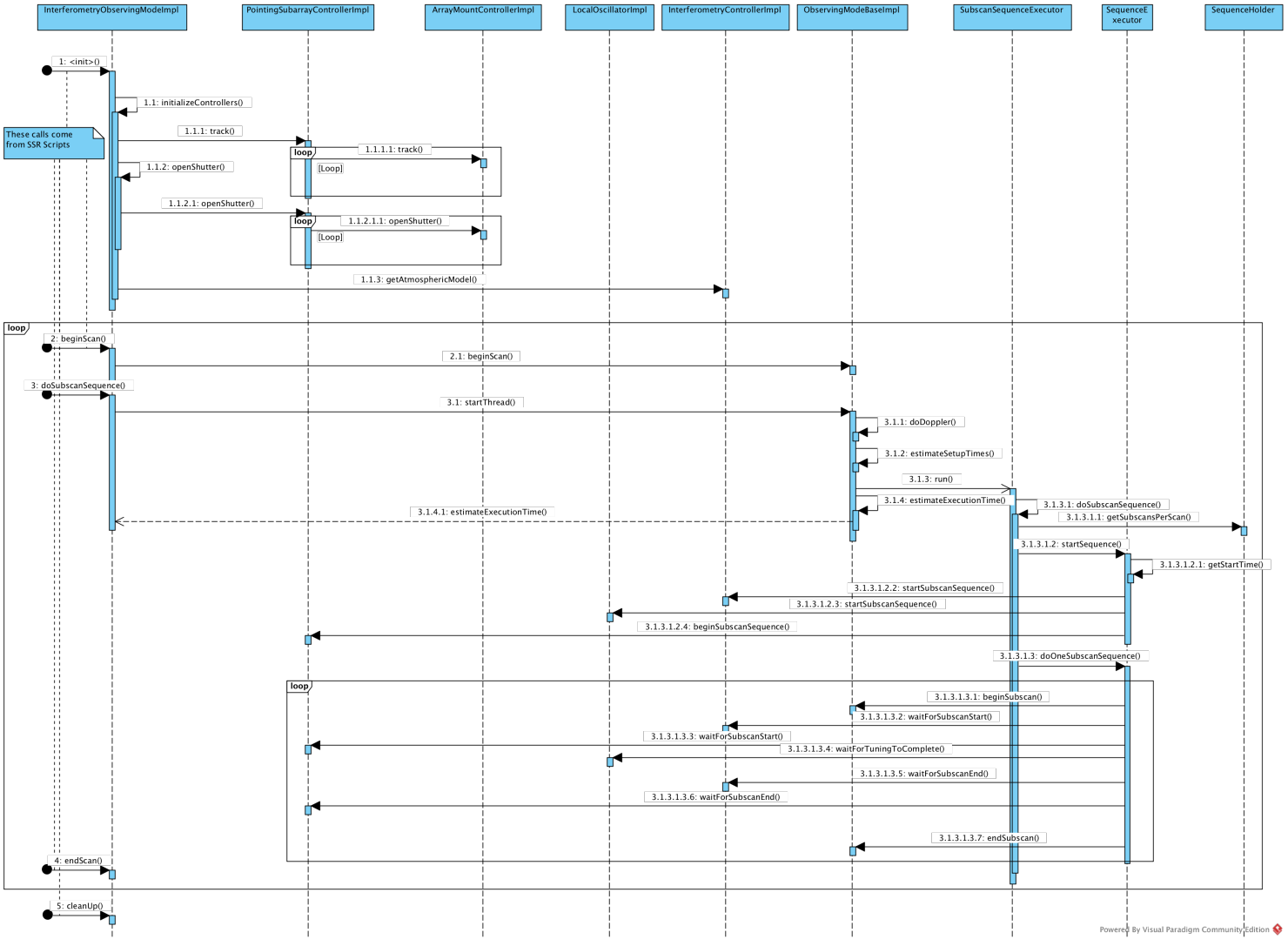


Figure 3.2: Interferometry Sequence Diagram

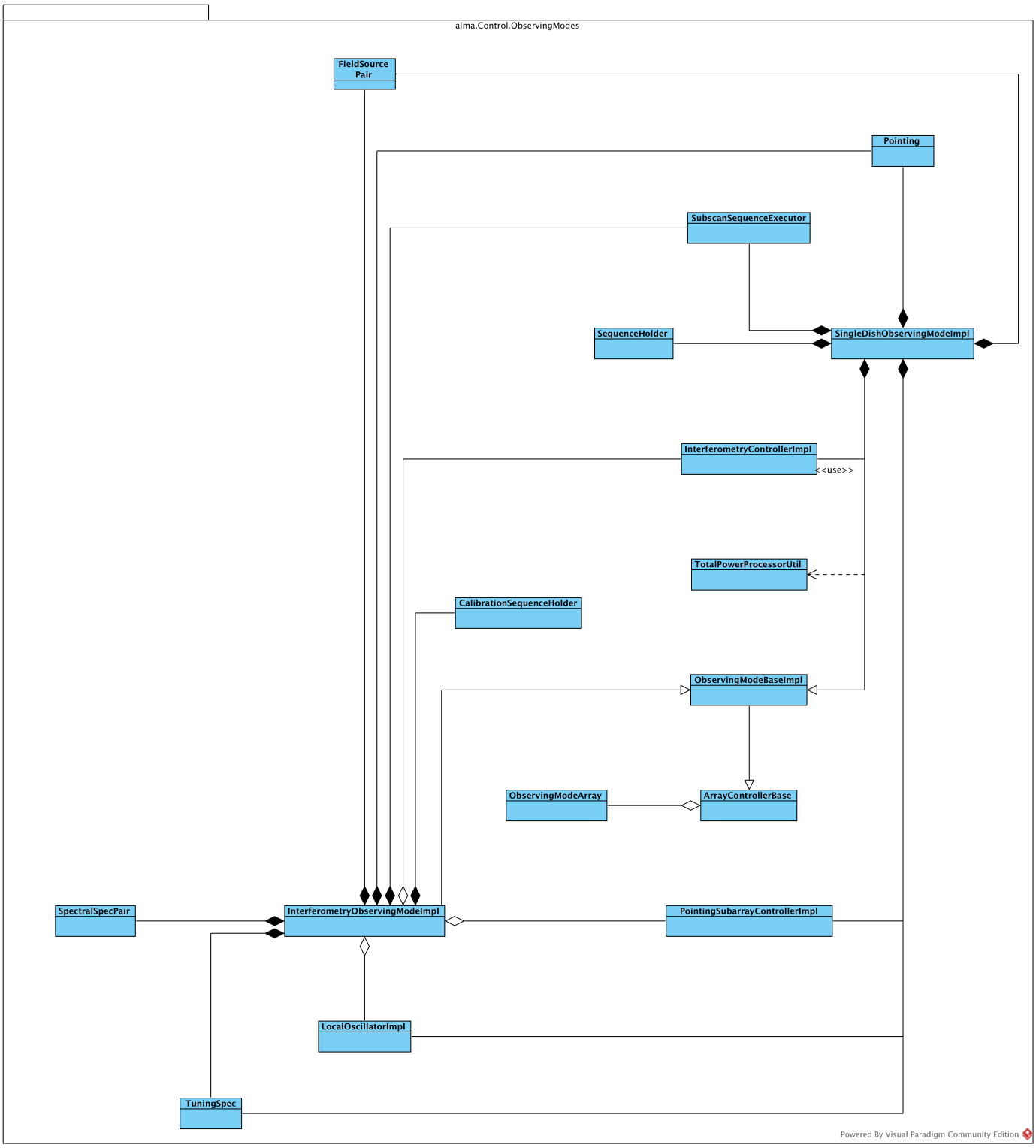


Figure 3.3: Class Dependency Diagram

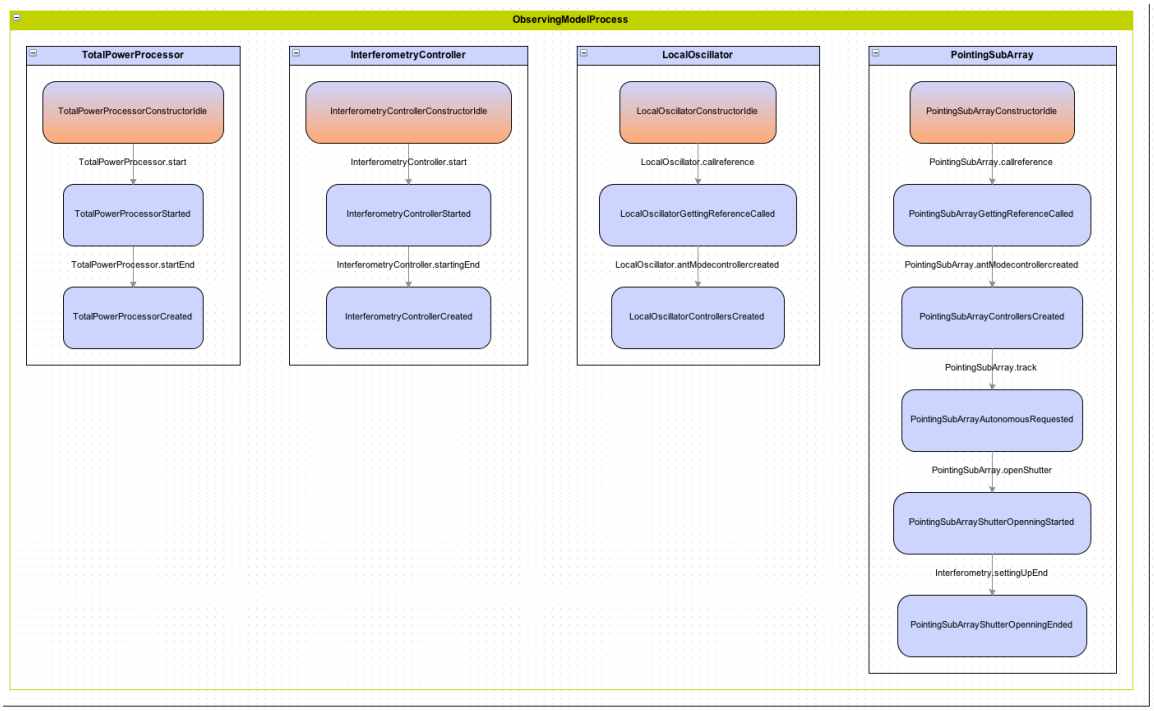


Figure 3.4: SCXML Example

events it is possible to follow executions and therefore, debug them. Also, this model is really interesting to generate a machine learning approach that could recognize patterns and/or predict bad states or exceptions, which represents a huge impact on the debugging and operation processes.

To generate a base model, the Discovery application could be used which aims to search for transitions. This could be helpful to start designing a model.

This model was developed thinking about the possibility of having asynchronous processes, and for that matter it is possible to use the `parallel` element described on SCXML notation. All the children of the parallel element can happen simultaneously therefore, the state machine will have many states. See the example in Figure 3.4.

The name of the first state always contains the word "idle". These states mean that they are waiting for an event and are not a real state. This is to respect the notation. Every child of a state must have an initial state, so the `initial` element described on the SCXML notation is required.

This model was designed using classes as sub-states inside of the `parallel` element, which means an easier way to visualize and develop improvements on the diagram. See Figure 3.4. The main flow process is described in Figure 3.5.

R Figures 3.4 and 3.5 are reduced models, in order to make them easier to visualize on a page. If you want to see the more complete model, you can download it from: https://raw.githubusercontent.com/ javierivanov/ObsSM/master/ObsSM/src/main/resources/model.xml

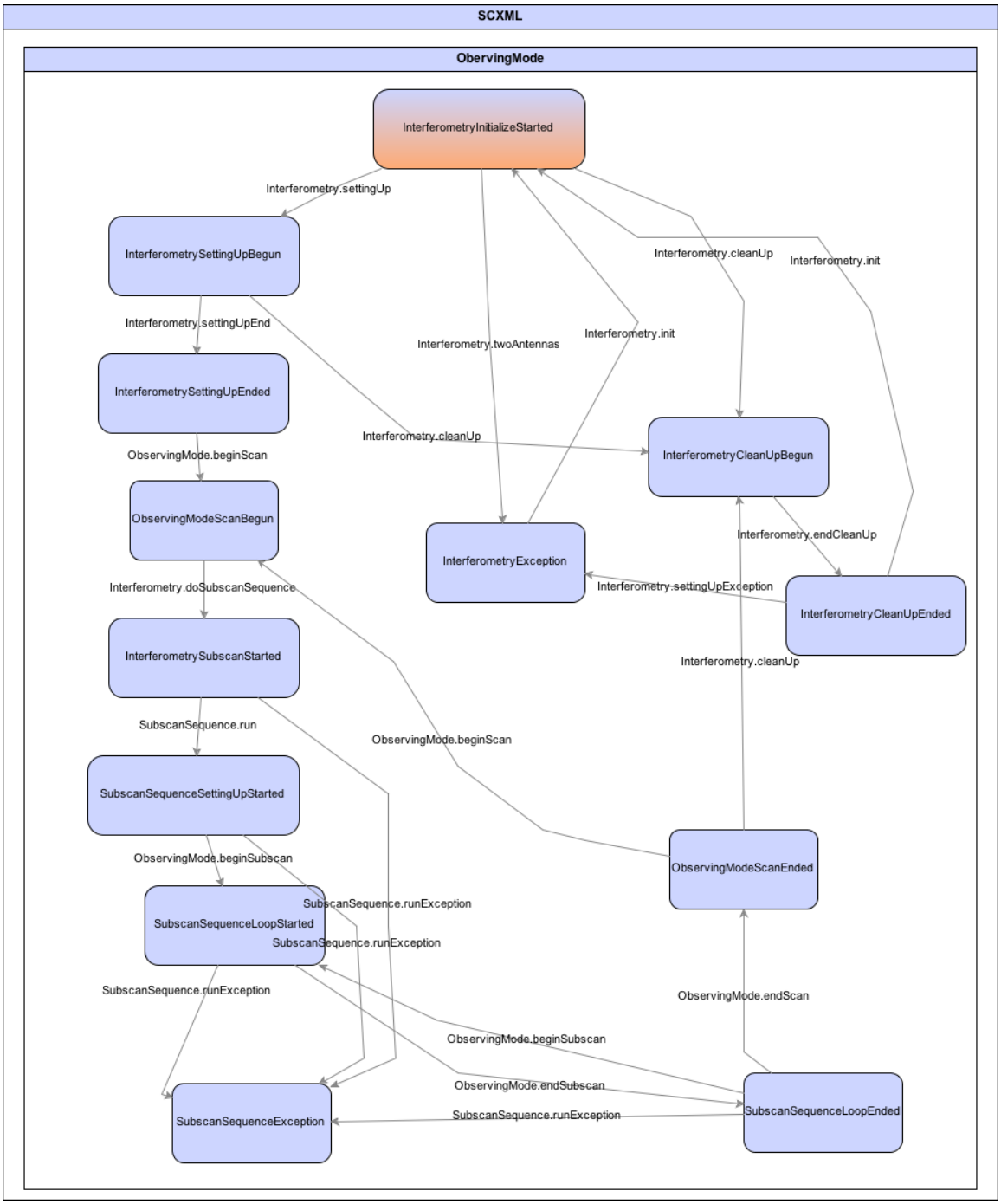


Figure 3.5: SM Model

3.3 Discovery

Discovery is a learning application which discovers the transitions between the events and creates a SCXML model. Discovery uses a log translate file to identify events from the log, and Discovery automatically select a range of days set in the `main.py` file.

The strategy is to considerer every *Array* like an *Agent* and this *Agent* has its own *event-state* and every transition to another *event* is recorded into a *transition list* calculating the probability between them (Markov's chains). Discovery only records logs, and does not have any logic idea about correct process. Therefore, it is possible to get bad logic transitions but which really happened on the log, triggered by time or systems errors. For that reason it is highly recommended to check the consistency of the generated model, to assure its quality.

```
========================START=TRANSITION=========================
state_from: array.creation
states_to:  [u'interferometry.start', u'subscan.start']
T:    [0.4, 0.6]
counter:  5
=========================END=TRANSITION=========================
========================START=TRANSITION=========================
state_from: interferometry.start
states_to:  [u'mount.release', u'scan.start', u'subscan.start']
T:    [0.4444444444444444, 0.2222222222222222, 0.3333333333333333]
counter:  9
=========================END=TRANSITION=========================
========================START=TRANSITION=========================
state_from: mount.release
states_to:  [u'mount.release', u'observation.stop']
T:    [0.9661016949152542, 0.03389830508474576]
counter:  177
=========================END=TRANSITION=========================
========================START=TRANSITION=========================
state_from: observation.stop
states_to:  [u'interferometry.start', u'array.destruction', u'subscan.start']
T:    [0.5, 0.16666666666666666, 0.3333333333333333]
counter:  6
=========================END=TRANSITION=========================
========================START=TRANSITION=========================
state_from: subscan.start
states_to:  [u'scan.start', u'mount.release', u'subscan.start', u'interferometry
    .start', u'array.destruction']
T:    [0.225, 0.05, 0.625, 0.075, 0.025]
counter:  40
=========================END=TRANSITION=========================
========================START=TRANSITION=========================
state_from: scan.start
states_to:  [u'subscan.start', u'interferometry.start', u'scan.start']
T:    [0.8181818181818182, 0.09090909090909091, 0.09090909090909091]
counter:  11
```

```
=========================END=TRANSITION=========================
```

Listing 3.1: Example result from Discovery

As you can see, there are some transitions without logic due to a possible system failure, or timing troubles with logs. Therefore is necessary to check the reality and logic of the transitions. Never the less, it is a good tool to discover unknown possibles transitions.

3.4 Log Translate JSON Document

This section outlines the basics about the Log translate structure. The Log translate structure is a dictionary that translates a log line into an event (defined in the SCXML standard).

The documents contain a list of events or transition objects, each of these objects has the following attributes:

- `eventName (string)`: This is the ID of the transition and will be used on the model like a transition event.
- `search_list (list of (string))`: This is a list which contains the regular expressions that have to match in the log message. All of the elements must match.
- `keyName (string)`: This is a regular expression which aims to find an ID, like the array number of the log.
- `eventType (string)`: This is a custom variable. It could be used to identify if a log message is an event: initial, final, exception or error for example.

```
[
  {
    "eventName":"array.creation",
    "search_list": ["(component CONTROL\/Array\\d{3} activated and initialized)"
    ],
    "keyName":"(Array\\d+)",
    "eventType": "initial"
  },
...
]
```

Listing 3.2: Log Translate JSON Docuement example

R Using the *eventType* it is possible to use another special flag, like exception, error, etc.

R You can also review some information on these links:

- JSON Document location: ObsSM/ObsSM/src/main/resources/log_translate.json

- About JSON: http://www.json.org
- About Regular Expressions: http://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap09.html

3.4.1 Nomenclature notes

In order to simplify name creations, all the *events* and *states* names have to be respectively on the form below:

- [system,object,procedure].[action] e.g.: `observation.start`
- [SystemObject][Procedure][ActionOnPast] e.g.: `ObservationProcessStarted`

3.4.2 Log table reference

This table contains the log lines used on the JSON documents to identify events on the logs.

File	Method	Message
AcsContainer	activate_component	component CONTROL/Array001 activated and initialized in 151 ms.
ContainerServicesImpl	releaseComponent	client 'CONTROL/MASTER' has successfully delivered a component release request for curl=CONTROL/Array001
InterferometryObservingModeImpl	<init>	Interferometry observing mode starting.
InterferometryObservingModeImpl	<init>	This array has been set up to use a sum antenna and cannot be used with this observing mode.
InterferometryObservingModeImpl	<init>	The interferometry observing mode needs an array with at least two antennas.
InterferometryObservingModeImpl	initializeControllers	Enabling delay compensation and fringe tracking.
InterferometryObservingModeImpl	initializeControllers	Failed to cleanly activate the Observing Mode
InterferometryObservingModeImpl	<init>	Completed starting the interferometery observing mode.

ObservingMode BaseImpl	logBeginScan	Starting scan 1 with scan intent CALIBRATE_POINTING using CHANNEL_AVERAGE_CROSS.
ObservingMode BaseImpl	endScan	Scan 33 was completed at 02:21:22.936
ObservingMode BaseImpl	beginScan	Cannot publish a scan started event.
Interferometry ObservingModeImpl	doSubscanSequence	Starting doSubscanSequence
SubscanSequence Executor	run	The SubscanSequenceExecutor.run function is starting.
SequenceExecutor	doOneSubscanSequence	Cannot start a sub-scan sequence.
SubscanSequence Executor	toCompletion	Problem executing a subscan sequence containing
SequenceExecutor	doOneSubscanSequence	Cannot complete sub-scan
ObservingMode BaseImpl	beginSubscan	Scan 1, sub-scan 1 has an intent of ON_SOURCE, takes 5.760 seconds from 02:37:25.728 to 02:37:31.488
ObservingMode BaseImpl	endSubscan	Scan 1, sub-scan 1 was completed at 02:37:31.488
Interferometry ObservingModeImpl	cleanUp	Interferometry observing mode shutting down.
Interferometry ObservingModeImpl	cleanUp	Completed shutting down the interferometry observing mode.
ArrayController Base	call	Trying to get a reference to a component called CONTROL/PM02/ MountController thats collocated with the component called CONTROL/PM02
ArrayController Base	createAntModeControllers	Creating and initializing all the alma.Control. MountControllerHelper controllers took 0.017 seconds.

ArrayController Base	createAntModeControllers	Creating and initializing all the alma.Control. AntLO-ControllerHelper controllers took 2.448 seconds.
ArrayController Base	createAntModeControllers	Creating and initializing all the alma.Control. AntLO-ControllerHelper controllers took 2.448 seconds.
Interferometry Control-lerImpl	<init>	Interferometery controller starting.
Interferometry Control-lerImpl	<init>	Completed starting the in-terferometery controller.
TotalPower Proces-sorUtil	createTotalPowerProcessor	Created the CONTROL/Ar-ray032/ TotalPowerProces-sor component.
TotalPower Proces-sorUtil	createTotalPowerProcessor	Creating the CONTROL/Ar-ray032/ TotalPowerProces-sor component
ArrayMount Control-lerImpl	track	Ensuring enough antennas are in autonomous mode (re-leasing the brakes).
ArrayMount Control-lerImpl	openShutter	Opening the shutter on all antennas
LocalOscillator Thread	abortSubscanSequence	Aborting the tuning se-quence.
Interferometry Control-lerImpl	stopSubscanSequence	Stopping the subscan se-quence
Interferometry Control-lerImpl	startSubscanSequence	Sequence timing pro-vided to the correlator. Seq#1 [setupTime=2.640, duration=10.368, process-ingTime=0.000]
Interferometry Control-lerImpl	waitForSubscanEnd	Subscan 1 contains 18 in-tegration(s) (240672 total bytes) and 1 subintegra-tion(s)/integration
Interferometry Control-lerImpl	cleanUp	Interferometry controller shutting down.
Interferometry Observ-ingModeImpl	cleanUp	Completed deactivating the interferometery controller.

LocalOscillator Thread	run	Completed a sequence of 1 tuning changes
LocalOscillator Impl	cleanUp	Shutting down delay compensation and fringe tracking.
Interferometry ObservingModeImpl	cleanUp	Completed deactivating the local oscillator.
ArrayController Base	cleanUp	Completed the cleanUp function
Interferometry ObservingModeImpl	cleanUp	Completed deactivating array mount controller offshoot

Table 3.1: Log Origins

3.5 ObsSM

The interpreter is a program which allows to parse a log input with a SCXML model. This program can read directly from ElasticSearch for log intervals. It can also wait for data in near real time (depends on ElasticSearch).

For more details see the Wiki: https://github.com/javierivanov/ObsSM/wiki

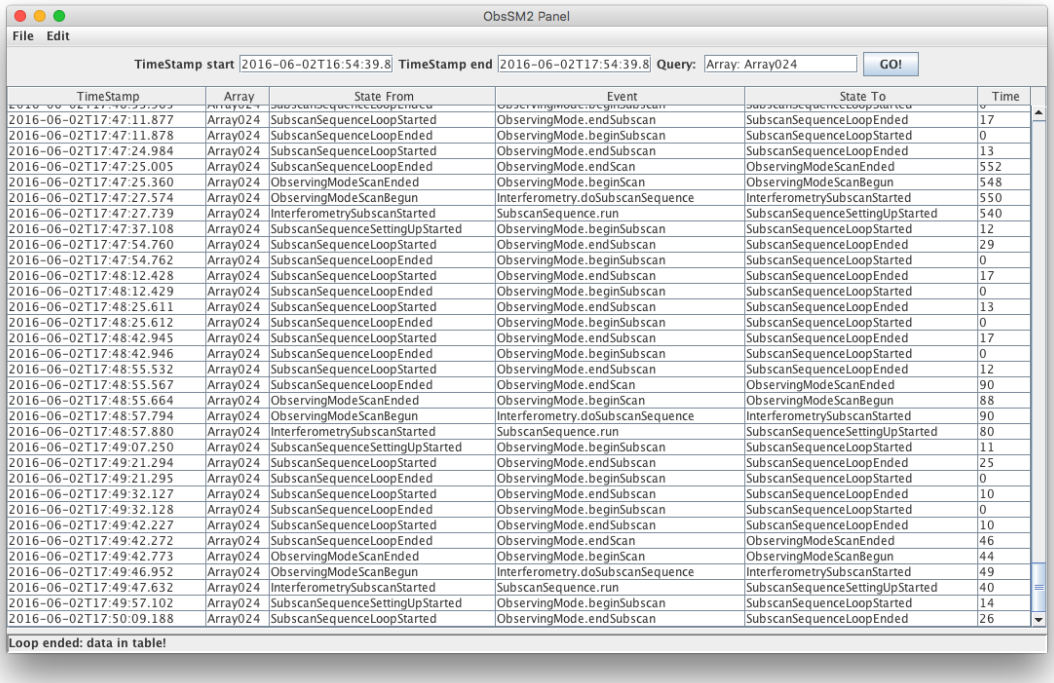


Figure 3.6: Default Graphical Interface ObsSM Panel

3.5.1 Architecture

The next sections aim to explain the behavior of the application.

SCXML Framework

The SCXML notation is a big document and requires a lot of constraints to be programmed. Therefore, a search for an SCXML Framewrok was performed. The only one which was documented well was the Apache implementation, and it is part of the common applications provided by them. This Framework is written in Java, thus the application was developed in Java as well.

Activity diagram

This diagram is intended to explain the flow of data and procedures within the application.

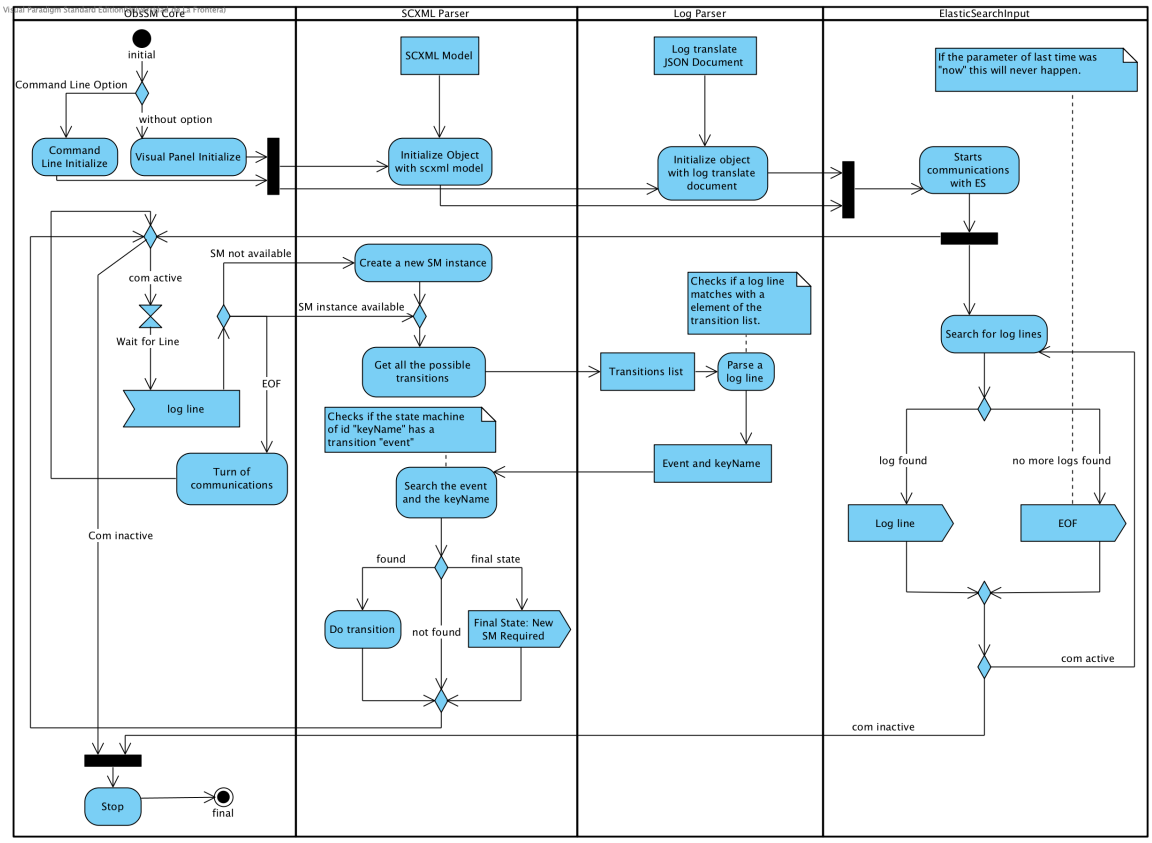


Figure 3.7: Activity Diagram

Class Diagrams

The packages of ObsSM are shown below.

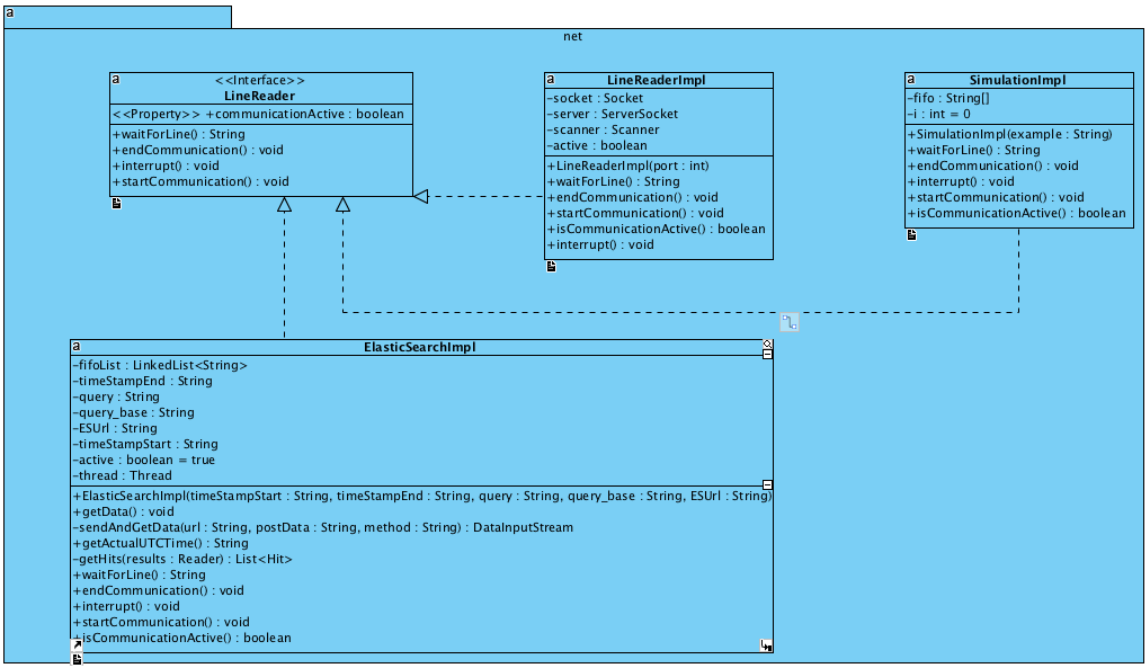


Figure 3.8: Network Package Class Diagram

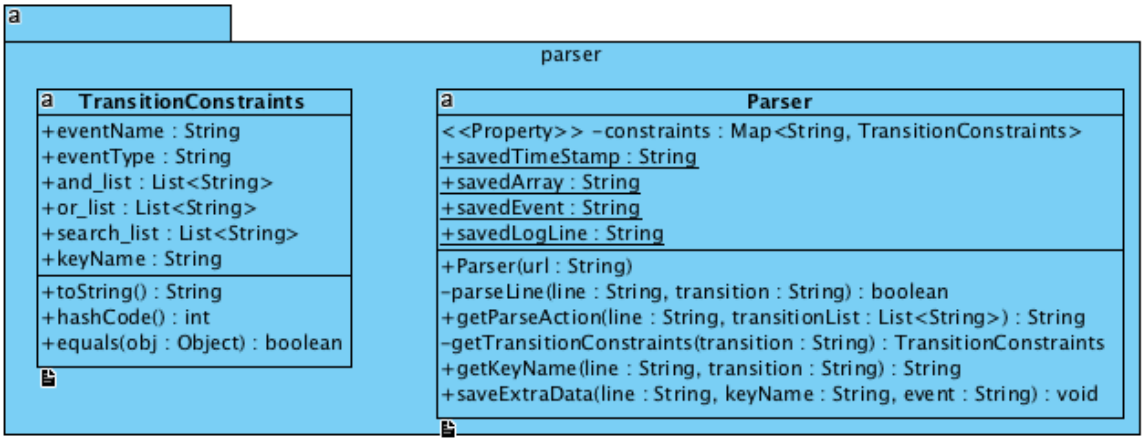


Figure 3.9: Parser Package Class Diagram

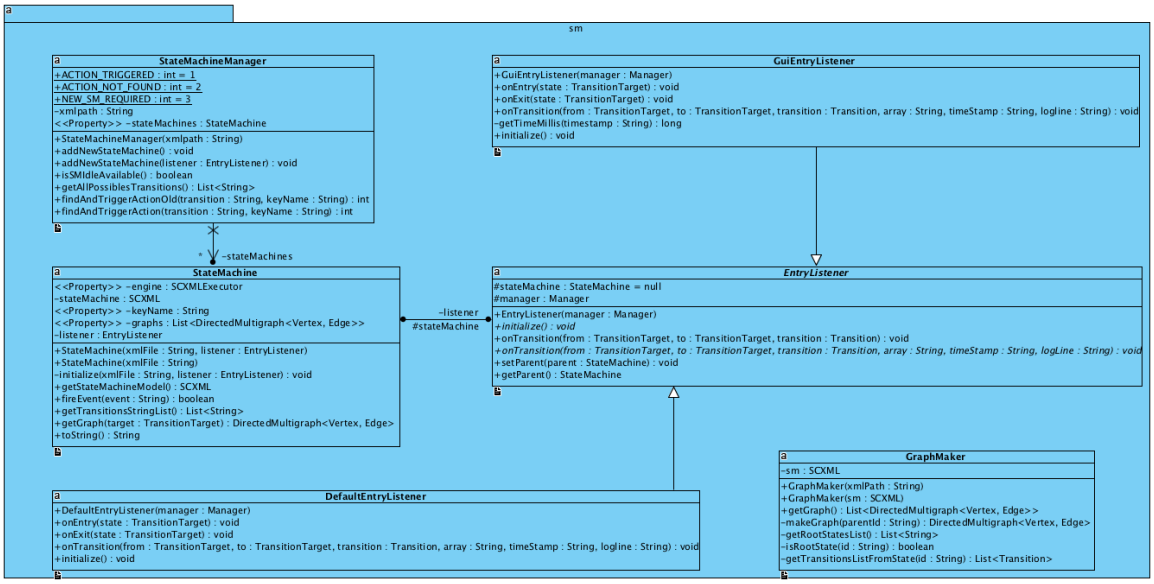


Figure 3.10: State Machine Package Class Diagram

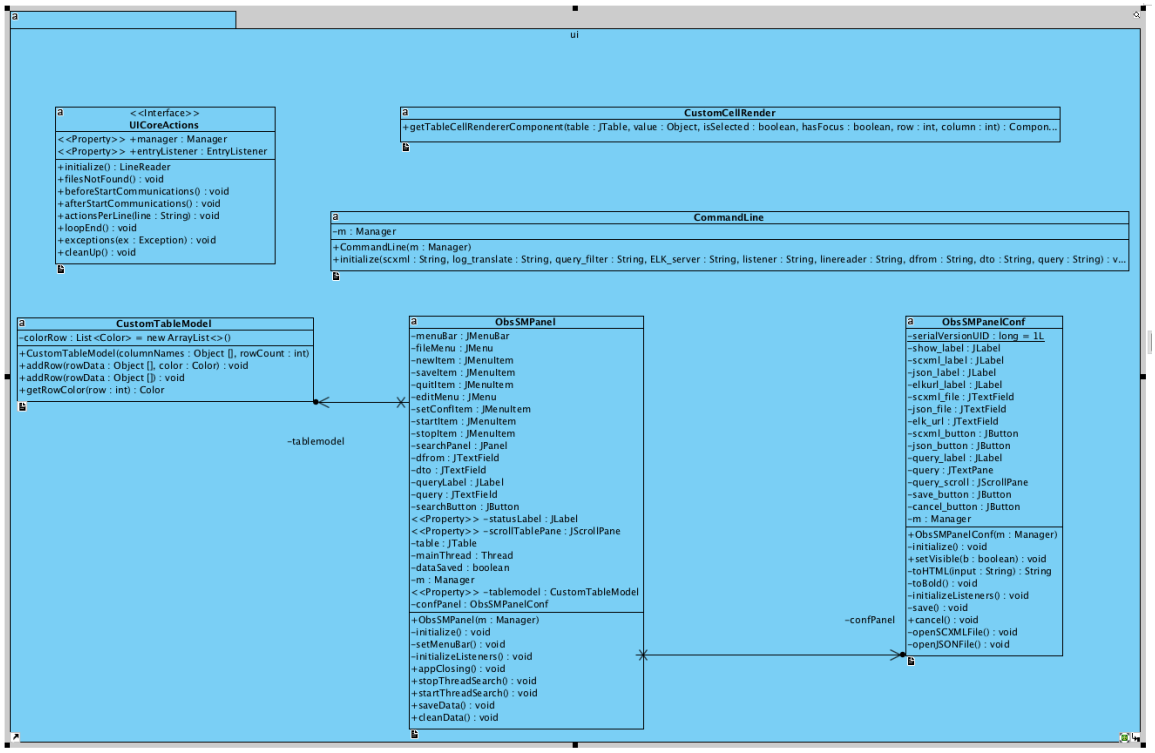


Figure 3.11: User Interface Package Class Diagram

3.6 ObsSM Outputs

This is the most important subject, here are the possible outputs:

3.6.1 Default Graphical Mode

This is the default mode. ObsSM will run in this mode if you do not use a command line option. The Figure 3.12 shows the default data visualization. The Figure 3.13 shows the configuration panel which is by default on the Image.

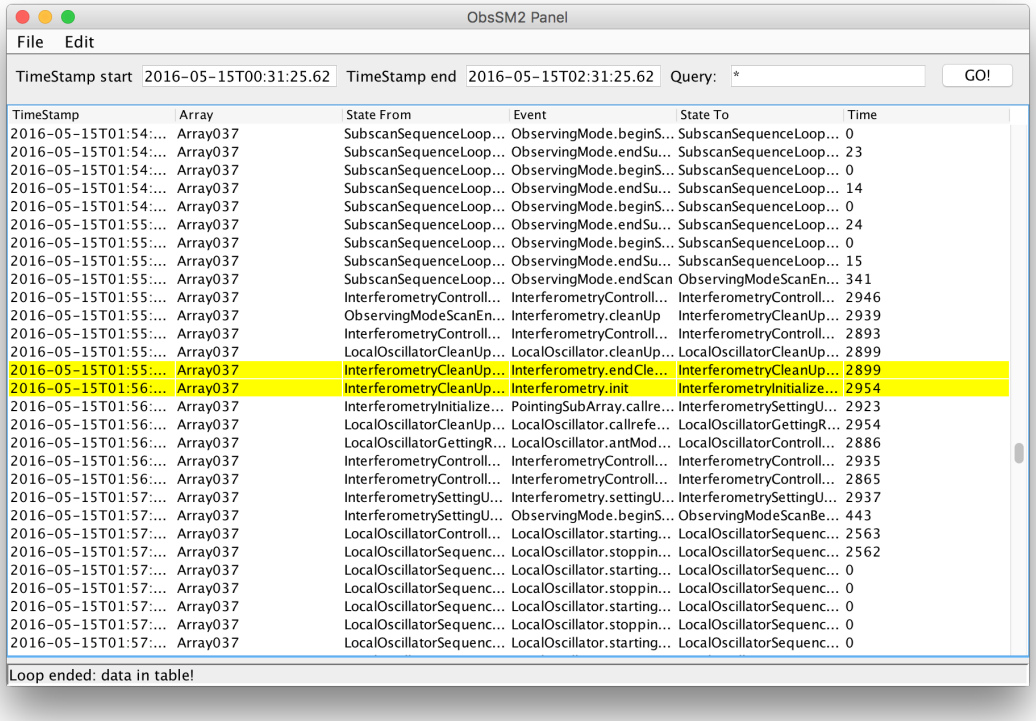


Figure 3.12: ObsSM2 Panel

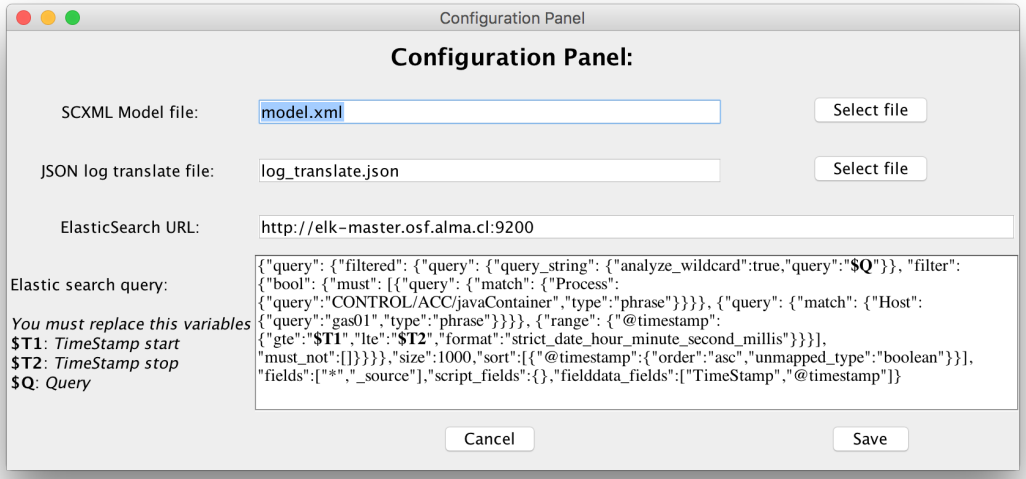


Figure 3.13: ObsSM2 Panel Configuration

3.6.2 Command Line Mode

This mode is useful for automation and custom configurations. As you can see in this command-line options example:

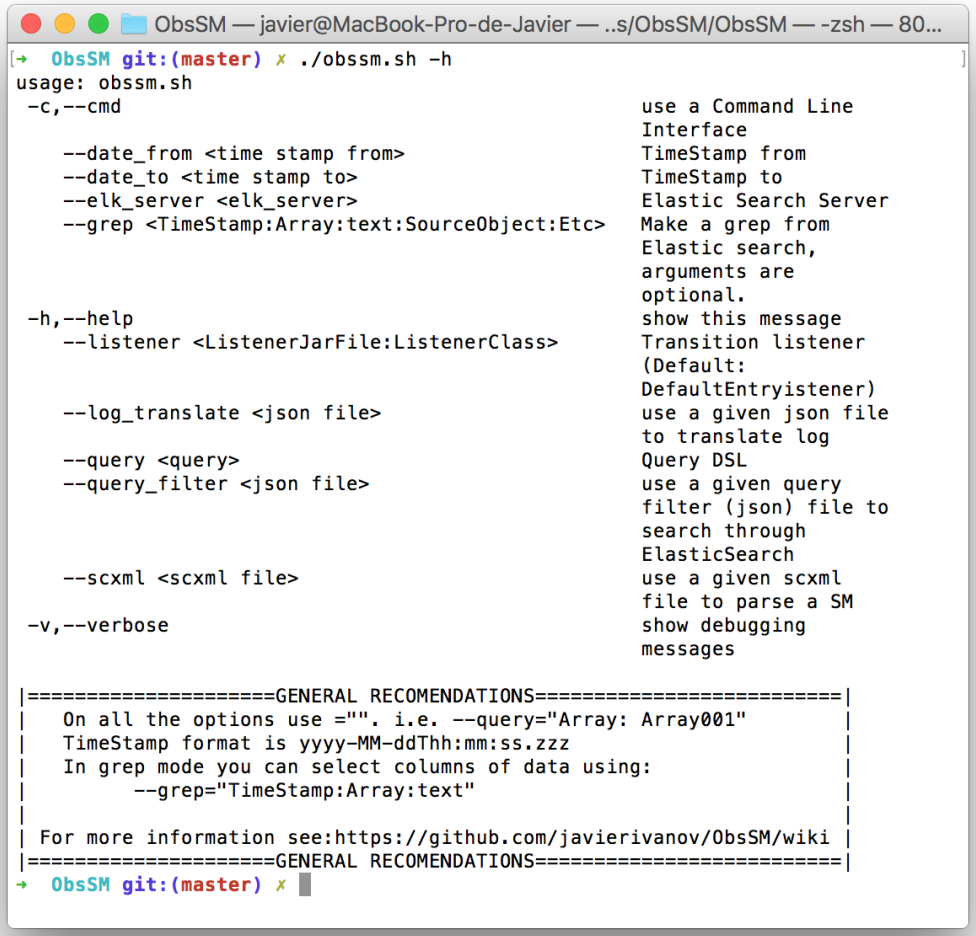


Figure 3.14: Command Line Options

3.6.3 Grep Mode

In this mode is possible to display in the standard output log messages directly from Elastic Search using the command line as it is shown in Figure 3.15.

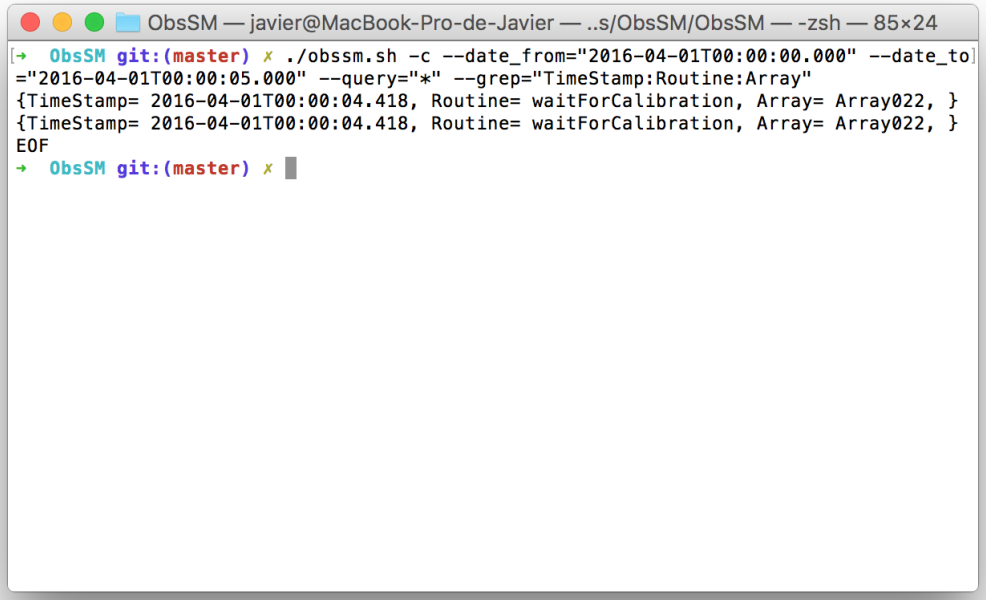


Figure 3.15: Grep Mode Example

This mode allows to obtain raw data.
It is possible also do some filters: `grep="TimeStamp:text:Routine:File"` as an example.

3.6.4 Third Party Plugins

Also ObsSM provides an interface to develop third-party plugins. As an example, an animated transition viewer was developed, as shown in Figure 3.16

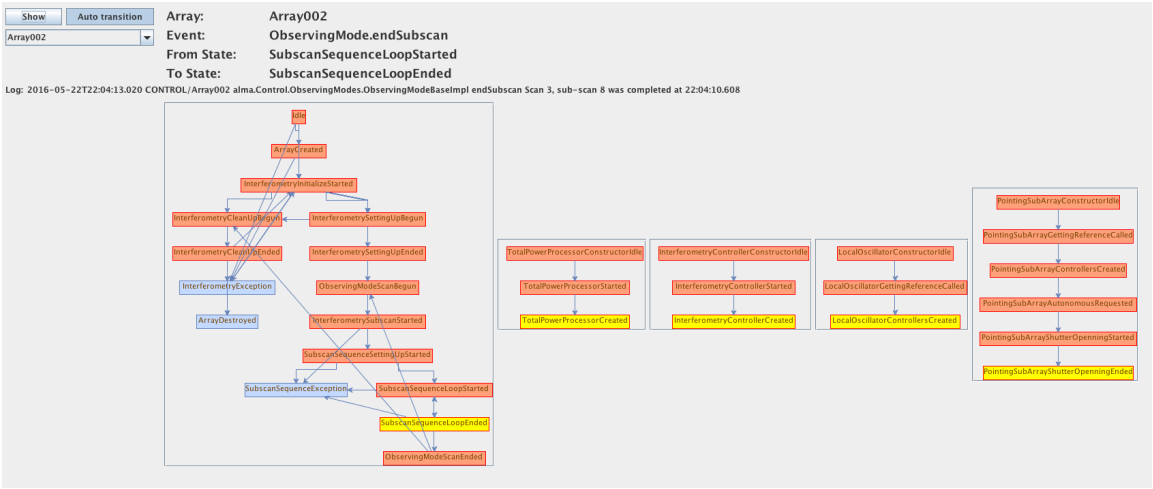


Figure 3.16: Plugin of a Graphical Animated Interface

4. Conclusions

The behavior of the observing mode was documented with several diagrams. These documents will allow to simplify the understanding and studying of this component. This experience will be useful for future analysis of another component or subsystem in order to create documentation.

A State Machine model using the SCXML Standard was also developed. This model describe the sequence of an observation. It could be upgraded and used in other contexts. It can reproduce the methodology to create new state machines regarding others subsystems, as well.

Additionally a tool which works as an interpreter of logs was developed. The output of this application could be a path of states, a SVM, or other kinds of output types that could be found in the course of this work. In the future, using this tool, it could be possible to create an advanced recognition pattern application, making it possible to identify some unusual states.

In addition, the tool provides a visualization of the state machine as a graph and it is shown in near-real time.