aLIGO Guardian Overview

Jameson Graef Rollins

DetChar F2F
LIGO Hanford

July 16, 2014

Introduction and Overview

Introduction

Guardian is the new aLIGO automation system.

Its job is to control the global state of the interferometer, i.e. manage and automate the locking sequence by coordinating the states of all interferometer subsystems.

It is also designed to be a useful tool for interferometer commissioning. It should aid in the commissioning process, and allow for easy reconfiguration of the interferometer to enable various commissioning tasks.

Overview

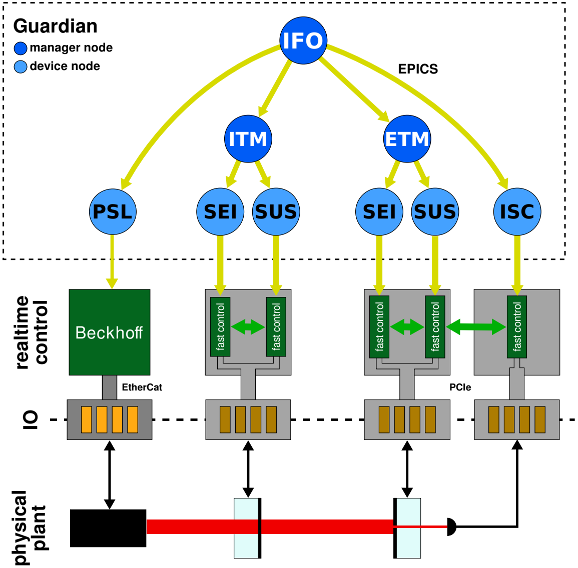
Guardian is a hierarchical **state machine**.

- Distributed guardian processes (**nodes**) oversee particular domains of the interferometer.
- Each node understands **states** for its domain. State code describes actions on the domain, such as changes to the plant and/or verification of the configuration.
- A hierarchy of nodes control the full IFO, with upper level **manager** nodes controlling sets of **subordinate** nodes, down to lowest level **device** nodes that talk directly to the real-time front-ends and Beckhoff.

Overview

Manager nodes control subordinates, with lowest level device nodes that talk directly to the real-time system.

A single IFO manager node will sit at the top coordinating the state of the entire interferometer.



System and state behavior

Core state machine engine daemon

The core of Guardian is the **node daemon**. It is essentially a state machine execution engine.

Node daemons execute **system modules** that describe the state graph that the daemon follow and the code to be executed during each state.

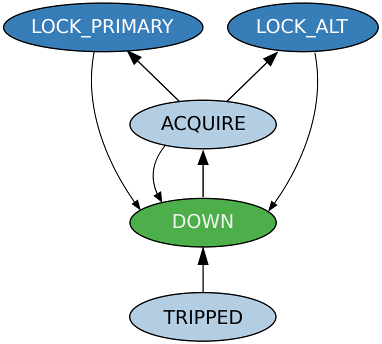
They run continuously, responding to system changes and user input.

System behavior

Each guardian system is represented as a **directed graph**.

The system **states** are represented by nodes in the graph, each consisting of executable code.

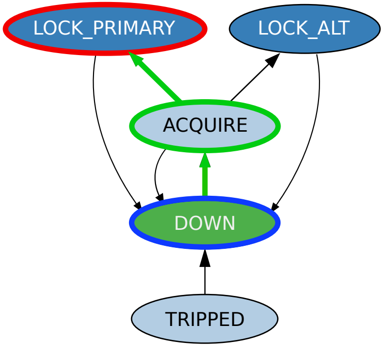
Graph **edges** represent possible transition between states. Edges have zero content, representing only which states are accessible from each other.



System behavior

Guardian accepts commands in the form of a **request state**.

Guardian then looks at the **current state** and calculates the shortest **path** to reach the request.

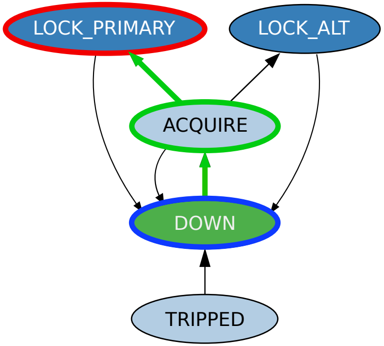


System behavior

Code for the current state is executed in a loop until a completion condition is returned.

If there are more states in the path, the system transitions to the next state and immediately starts executing the new code.

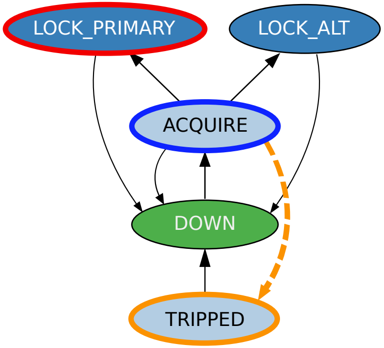
If no exit condition is achieved, or the current state is the requested state, the code continues execution.



System behavior

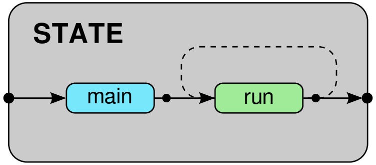
States can also specify **jump** transitions that bypass the normal dynamics of the graph.

Jumps represent transitions that are in some sense "undesirable" (e.g. lock loss, watchdog trip, etc.). They are ignored in normal path calculations



State behavior

The states themselves have very simple behavior. There are two state *methods* (i.e. functions).

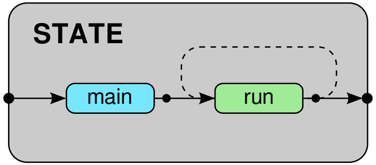


They are essentially identical except that:

main	executed **once** immediately upon entering state.
run	executed **in a loop**. Used to continuously check for state completion conditions.

State behavior

The states themselves have very simple behavior. There are two state *methods* (i.e. functions).

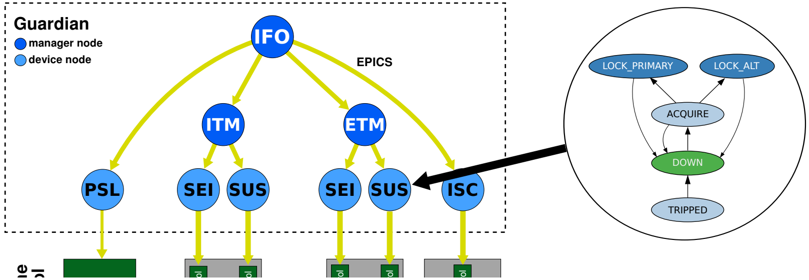


If either state returns `True` the state **completes** and guardian transitions to the next state ("edge transition").

Either method can also return the name of a state in which case guardian will **jump** immediately to that state ("jump transition").

Nodes

Each system is a **node** in the full IFO automation environment:



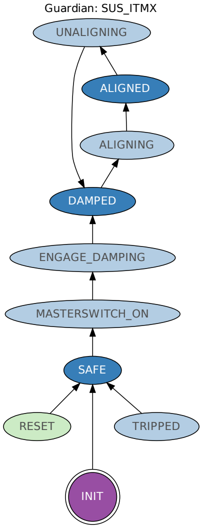
Manager nodes control subordinates by requesting states and monitoring their progress towards achieving that request.

Commissioned systems: SUS

SUS was the first subsystem automated (also the simplest). All suspensions nodes have identical guardian structure, replying upon a base SUS.py module and a sustools.py library interface.

Fully handles recovery to aligned state from watchdog trips.

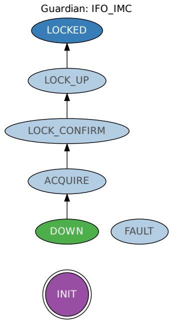
Currently (as of this week at LLO) *not* touching the alignment offsets. Will need to revisit how alignment offsets are managed.

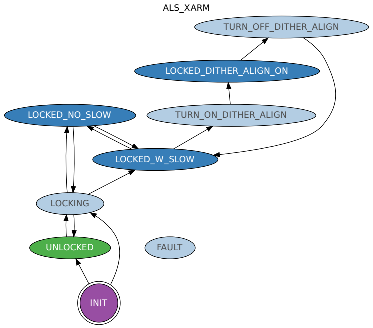


Commissioned systems: IMC and ALS

The **IMC** was next, and was the first "non-trivial" control. It is now deployed at both sites (although still being tweaked).

The **ALS** system under active development at LHO.





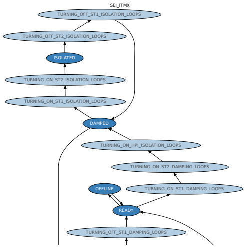
Commissioned systems: SEI

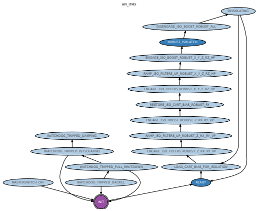
Next was **SEI** at LHO, the most complicated subsystem from an automation perspective. Fully handles recovery to full isolation, coordinating HPI and ISIs. Chamber seismic system consists of multiple nodes managed by a single “chamber manager” (HAM: 3 nodes, BSC: 4 nodes):

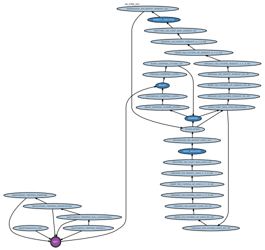
SEI_ITMX
(manager)

HPI_ITMX

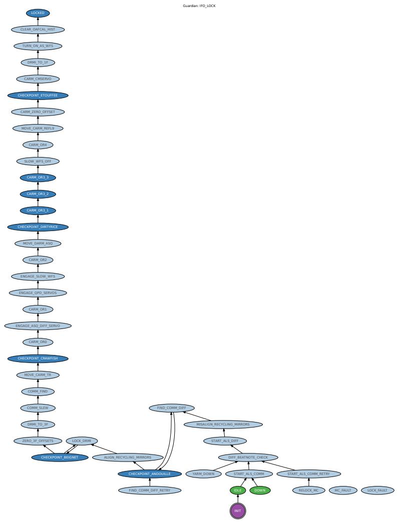
ISI_ITMX_ST{1,2}


SEI_ITMX
TURNING_OFF_ST1_ISOLATION_LOOPS
TURNING_OFF_ST2_ISOLATION_LOOPS
ISOLATED
TURNING_ON_ST2_ISOLATION_LOOPS
TURNING_ON_ST1_ISOLATION_LOOPS
DAMPED
TURNING_ON_HPI_ISOLATION_LOOPS
TURNING_ON_ST2_DAMPING_LOOPS
OFFLINE
TURNING_ON_ST1_DAMPING_LOOPS
READY
TURNING_OFF_ST1_DAMPING_LOOPS


HPI_ITMX
DEISOLATING
DISENGAGE_ISO_BOOST_ROBUST_ALL
ROBUST_ISOLATED
ENGAGE_ISO_BOOST_ROBUST_X_Y_Z_RZ_HP
RAMP_ISO_FILTERS_UP_ROBUST_X_Y_Z_RZ_HP
ENGAGE_ISO_FILTERS_ROBUST_X_Y_Z_RZ_HP
RESTORE_ISO_CART_BIAS_ROBUST_RY
ENGAGE_ISO_BOOST_ROBUST_Z_RX_RY_VP
WATCHDOG_TRIPPED_DAMPING
RAMP_ISO_FILTERS_UP_ROBUST_Z_RX_RY_VP
WATCHDOG_TRIPPED_DEISOLATING
ENGAGE_ISO_FILTERS_ROBUST_Z_RX_RY_VP
WATCHDOG_TRIPPED_FULL_SHUTDOWN
LOAD_CART_BIAS_FOR_ISOLATION
MASTERSWITCH_OFF
WATCHDOG_TRIPPED_DACKILL
READY
INIT


ISI_ITMX_ST1
DISENGAGE_ISO_BOOST_ROBUST_ALL
ROBUST_ISOLATED
RESTORE_ISO_CART_BIAS_ROBUST_RZ
ENGAGE_ISO_BOOST_ROBUST_X_Y_Z_RZ
RAMP_ISO_FILTERS_UP_ROBUST_X_Y_Z_RZ
RAMP_DAMPING_FILTERS_UP
ENGAGE_ISO_FILTERS_ROBUST_X_Y_Z_RZ
ENGAGE_DAMPING_LOOPS
ENGAGE_ISO_BOOST_ROBUST_RX_RY
READY
RAMP_ISO_FILTERS_UP_ROBUST_RX_RY
DISENGAGE_DAMPING_LOOPS
ENGAGE_ISO_FILTERS_ROBUST_RX_RY
RAMP_DAMPING_FILTERS_DOWN
LOAD_CART_BIAS_FOR_ISOLATION
DAMPED
DEISOLATING
DISENGAGE_ISO_BOOST_HIGH_ALL
HIGH_ISOLATED
RESTORE_ISO_CART_BIAS_HIGH_RZ
ENGAGE_ISO_BOOST_HIGH_X_Y_Z_RZ
RAMP_ISO_FILTERS_UP_HIGH_X_Y_Z_RZ
WATCHDOG_TRIPPED_DAMPING
ENGAGE_ISO_FILTERS_HIGH_X_Y_Z_RZ
WATCHDOG_TRIPPED_DEISOLATING
ENGAGE_ISO_BOOST_HIGH_RX_RY
WATCHDOG_TRIPPED_FULL_SHUTDOWN
RAMP_ISO_FILTERS_UP_HIGH_RX_RY
MASTERSWITCH_OFF
WATCHDOG_TRIPPED_DACKILL
ENGAGE_ISO_FILTERS_HIGH_RX_RY
INIT

Under development: Full IFO Locking



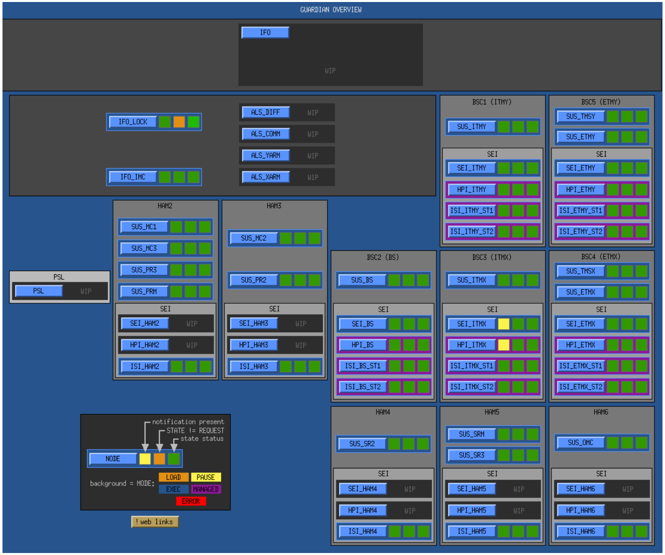
Primary IFO locking node:
IFO_LOCK
Full IFO lock acquisition node at LLO.

This node is also under heavy development, but starts to give a sense of what the full lock acquisition procedure will look like.

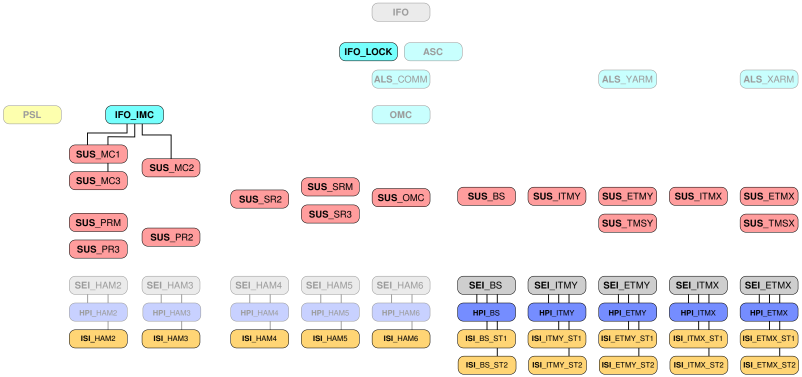
User code status

- SUS deployed at both sites.
- SEI deployed at both sites.
- IMC deployed at both sites.
- ISC is now primary focus, heavy development at LLO this summer.
- Working on improved system monitoring, front end channel monitoring infrastructure now being deployed.

Full system overview



Full node projection



Tracking Guardian status

GRD EPICS records

Guardian nodes broadcast their state and status via a set of EPICS channels, all prefixed with:

```
<IFO>:GRD-<NODE>_
```

```
MODE
REQUEST_S
REQUEST_N
STATE_S
STATE_N
TARGET_S
TARGET_N
STATUS
```

```
WORKER
NOTIFICATION
USERMSG
GRDMSG
ERROR
LOGLEVEL
VERSION
```

Primary status channel: `MODE`

`<IFO>:GRD-<NODE>_MODE`

enum record representing the execution mode of the node:

0:INIT	system initializing
1:RELOAD	reloading user code
2:STOP	worker terminated and no new code executed; state preserved
3:PAUSE	no new code executing but current method left to complete; state preserved
4:EXEC	primary code execution
5:MANAGED	primary code execution, managed mode, i.e. no auto-recovery from jump transitions

Primary status channel: `ERROR`

`<IFO>:GRD-<NODE>_ERROR`

enum record representing error condition in node:

0:False	no error
1:True	error present; all code execution stopped
2:Connect	channel access connection error; current state method is re-executed until the channel access error clears

Primary status channels: `REQUEST`, `STATE`, `TARGET`

```
<IFO>:GRD-<NODE>_REQUEST_S
<IFO>:GRD-<NODE>_REQUEST_N
<IFO>:GRD-<NODE>_STATE_S
<IFO>:GRD-<NODE>_STATE_N
<IFO>:GRD-<NODE>_TARGET_S
<IFO>:GRD-<NODE>_TARGET_N
```

These channels, for the requested state, current state, and target state, are the string (`S`) and numeric (`N`) representations of **usercode states**.

States enumeration

The data acquisition “frame writer” is currently only capable of recording numbers in the LIGO frames. This means we need to enumerate all the guardian states (map to number).

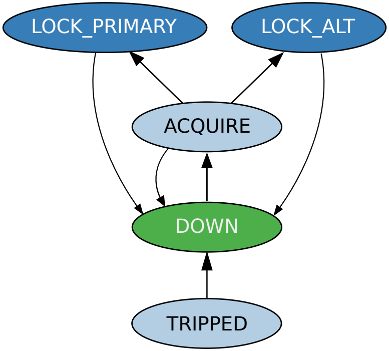
The `guardutil` program (part of the guardian package) is currently the best way to get state information:

guardutil

guardutil has useful functions for developing, analyzing, etc. guardian systems.

```
controls 0$ guardutil graph SIMPLE_LOCK
```

graph	draws system graphs
states	list system state enumeration
code	list all usercode
print	print general system info (combo of above)
etc.	



guardutil

Example run on my laptop:

```
servo:~ 0$ IFO=L1 USERAPPS=~/ligo/userapps
           guardutil states IFO_LOCK
0 INIT *
1 IDLE *
2 DOWN *
3 TURN_ON_PR2_WFS *
4 TURN_ON_CHARD_WFS *
5 DARM_ON_ASQ *
6 CARM_ON_TR *
7 TURN_ON_AS_WFS *
8 CARM_ZERO_OFFSET *
9 ALS_COMPLETE *
10 TURN_ON_INPUT_WFS *
11 DRMI_LOCKED *
12 LOCKED *
13 CARM_10PM *
```

States enumeration issues

Currently, guardian assigns an arbitrary number to each state at load time. As long as the code doesn't change the state enumeration doesn't change. However, if states are added or removed the enumerations will change. This means that the:

The state enumeration for a particular point in time is a function of code loaded at that time.

Not a good situation for tracking state over time. This needs to / will be improved. Currently plan: