The autonomy implemented in this repo is based on a hierarchical mission control framework (Figure 1), which is a modified version of the traditional MAPE-K loop. Here, we add a Mission Control componenet (MCC) to decide which task to do next for the mission, and leave the Analysis component (AC) to act as a low-level control center to fulfill the task. For each task, the AC might need to create multiple plans to execute in order to fulfill it.

Considering the available PLEXIL Execution engine provided by NASA teams, we consider synthesizing PLEXIL plans to fulfill a task. Each PLEXIL plan is consist of multiple lander operations, e.g., Grind and Deliver. And some operations may contains several primitive lander actions, e.g. moving one arm joint. The provided PLEXIL Execution engine wraps up primitive lander actions into PLEXIL library actions (renamed as operations in our design), which can be used to compose more high-level PLEXIL plans.

To sum it up, the proposed hierarchical mission control decompose a mission into four layers, mission, task, plan and operation, as shown in Figure 2.

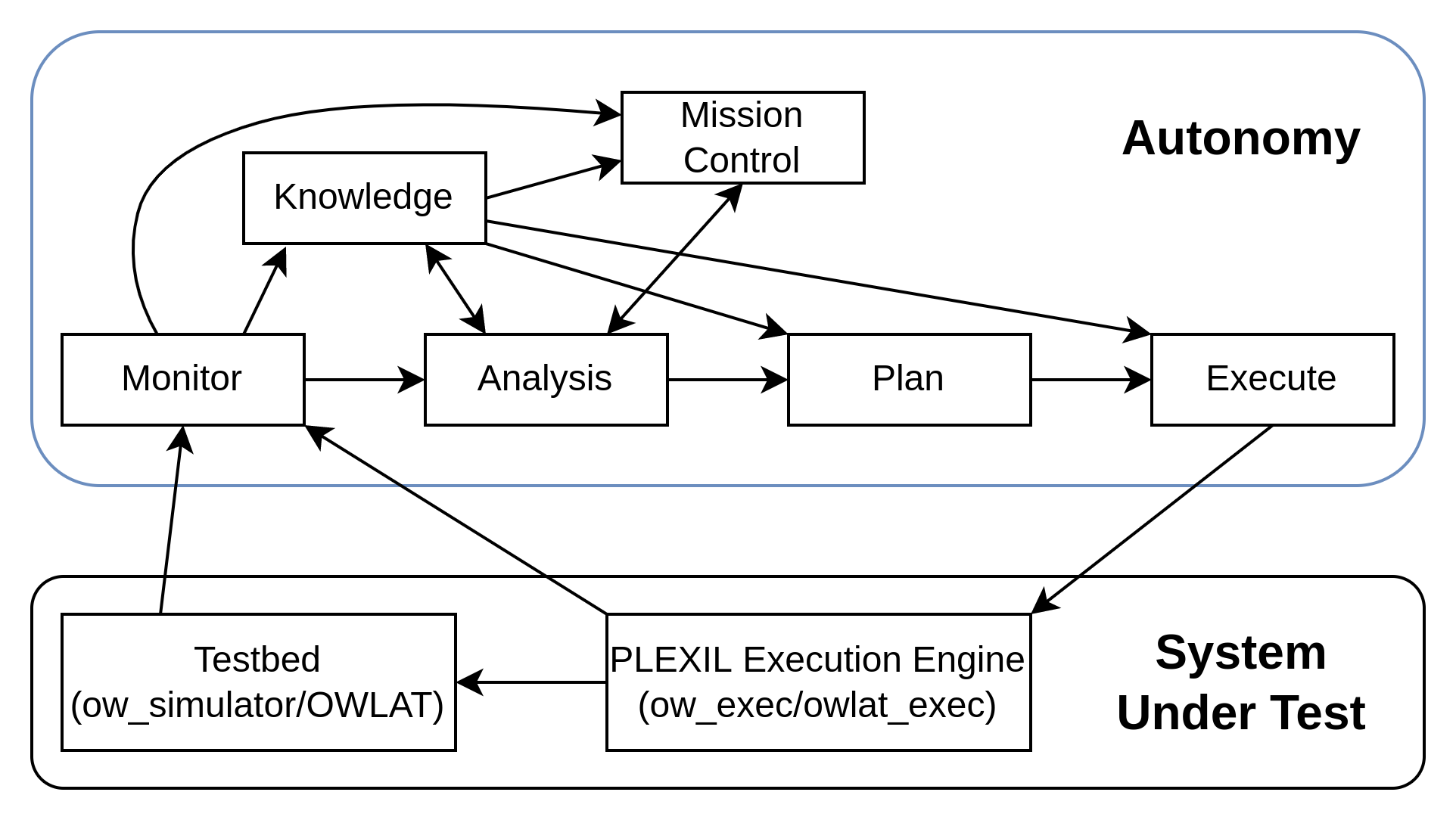


Figure 1 Autonomy Design - Hierarchical Mission Control.

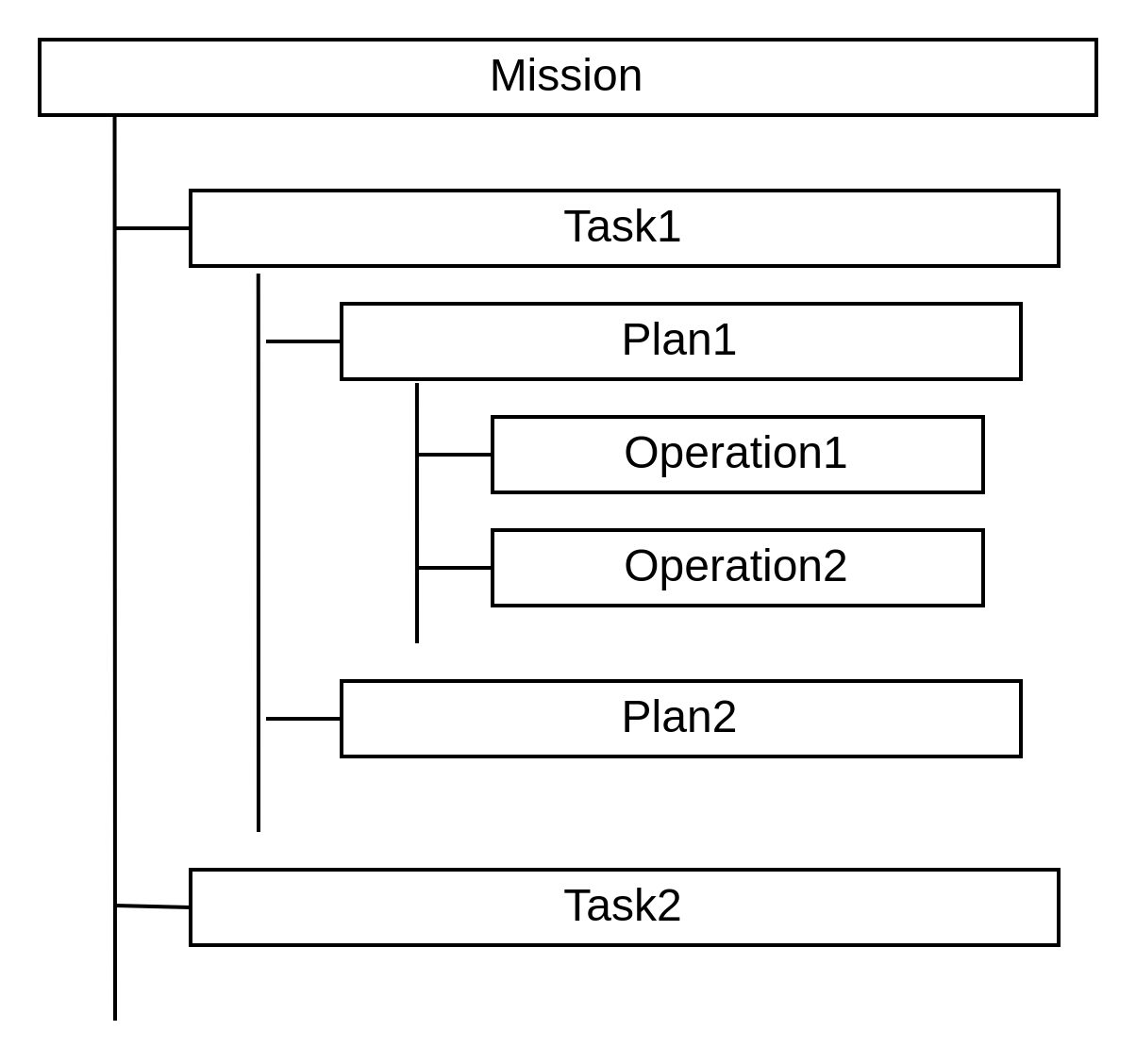


Figure 2 Mission Decomposition

The responsible entities for monitoring the status of each of the four layers are as follows:

* **Mission Status**: it is monitored by Mission Control. To determine it, it may need to collect information from Monitor, Knowledge and Analysis. In current implementation, it reports the status by only checking the task status returned by Analysis.
* **Task Status**: it is monitored by Analysis. By design, it is determined by collectively checking the completeness of the objective(s) of the task, available resources and constraints. Current implementation realizes a simplified version without a consideration of available resources and constraints. Therefore, Analysis will not stop requesting a new plan for a task until the recent plan fulfill the task or Mission Control sends a termination signal for the task.

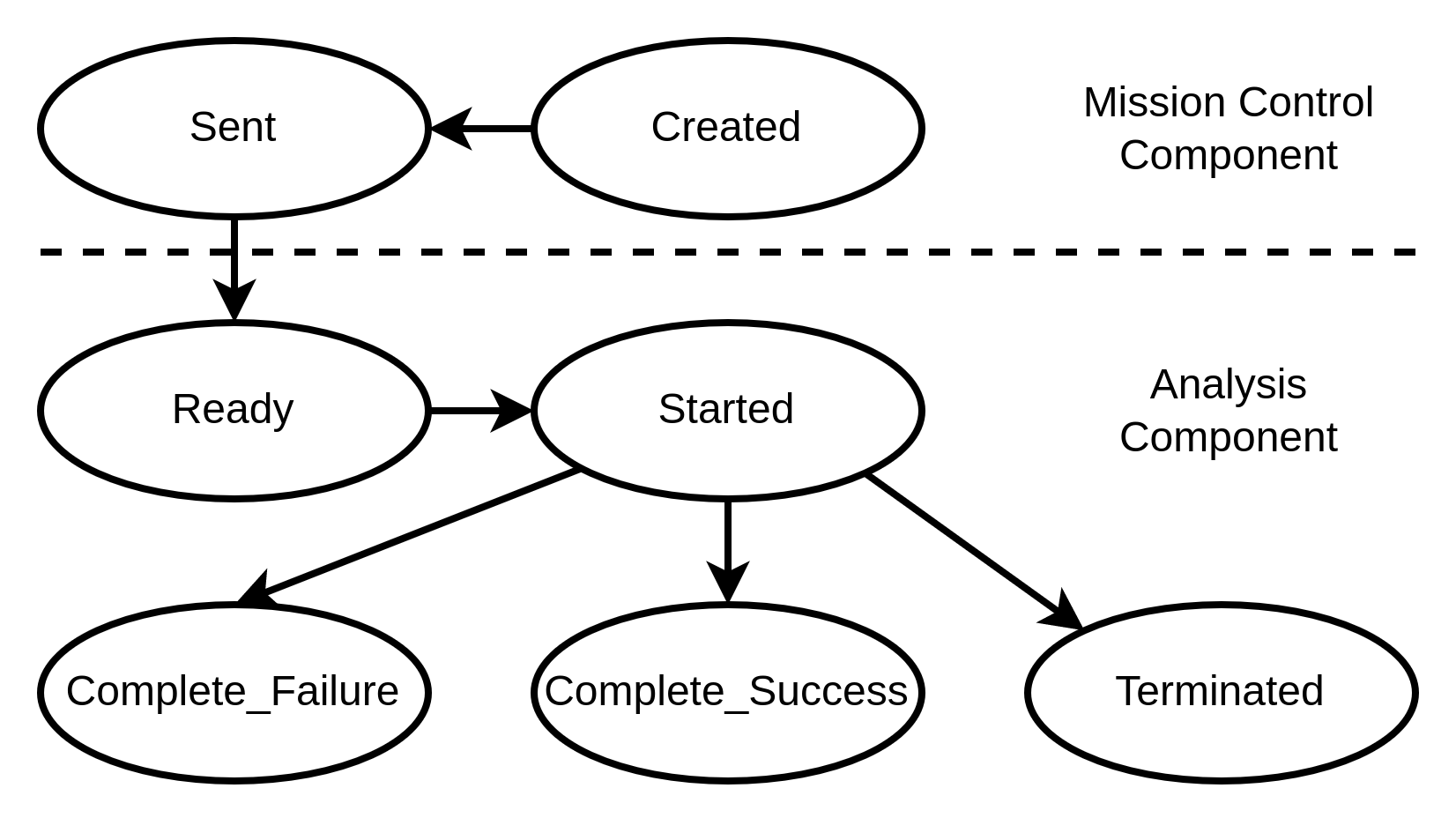


Figure X The transition diagram of task status.

* **Plan Status**: it is monitored by a component (ow\_exec or owlat\_exec ROS node) in the PLEXIL Execution Engine, whose role is to maintain the PLEXIL executive and arriving PLEXIL plans. Let’s call it ow\_executor.The Plan Status here is somehow a high-level state of a plan but does not have details about how each operation goes. The status corresponds to actions performed by ow\_executor. For an incoming PLEXIL plan, it will be added in a waiting list, then registered for starting, run and terminated (We added the termination action to assist the realization of a task termination signal from Mission Control). The status of a running plan could also be influenced by the SUSPEND and RESUME actions by ow\_executive, which target to the PLEXIL executive.

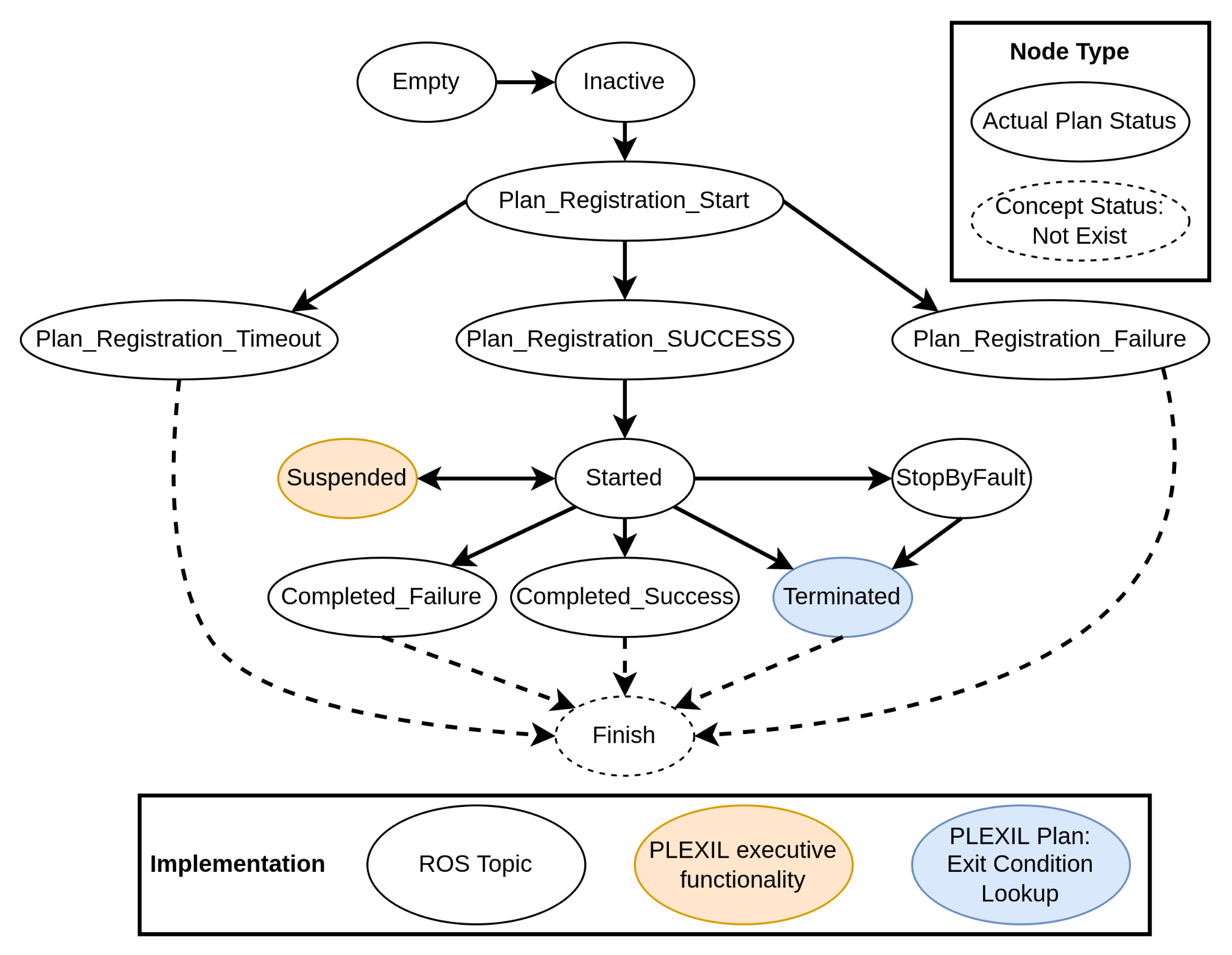


Figure X The transition diagram of plan status

* **Operation Status**: it is handled by PLEXIL Executive and the designers of PLEXIL plan templates. One task has one PLEXIL plan template. The template is designed by injecting PLEXIL Update nodes in interesting points in a PLEXIL plan. In the runtime, the interested information could be communicated from the PLEXIL executive to the autonomy through the PLEXIL Update node and a ROS topic (“/CurrentOperation”). Figure 3 shows the idea of how we enhance the current ow\_plexil package to support it. Figure 4 shows an examples of using Update nodes. In Figure 4, anything in black belongs to the template while the stuff highlighted in blue are filled in with runtime information during planing. The four Update nodes here show four interesting points around the Digging operation in the plan.

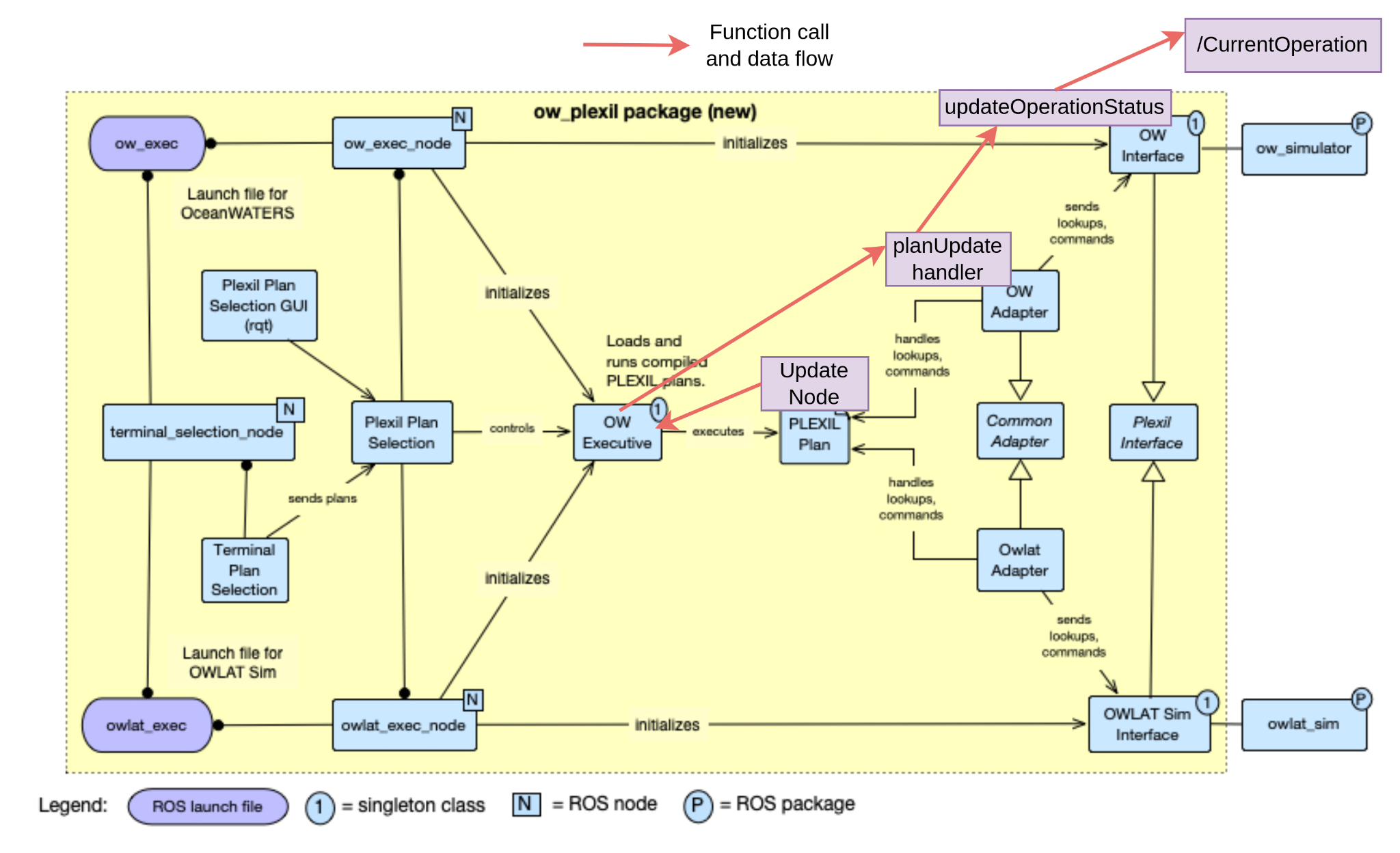


Figure 3 Support PLEXIL Update node in the ow\_plexil package

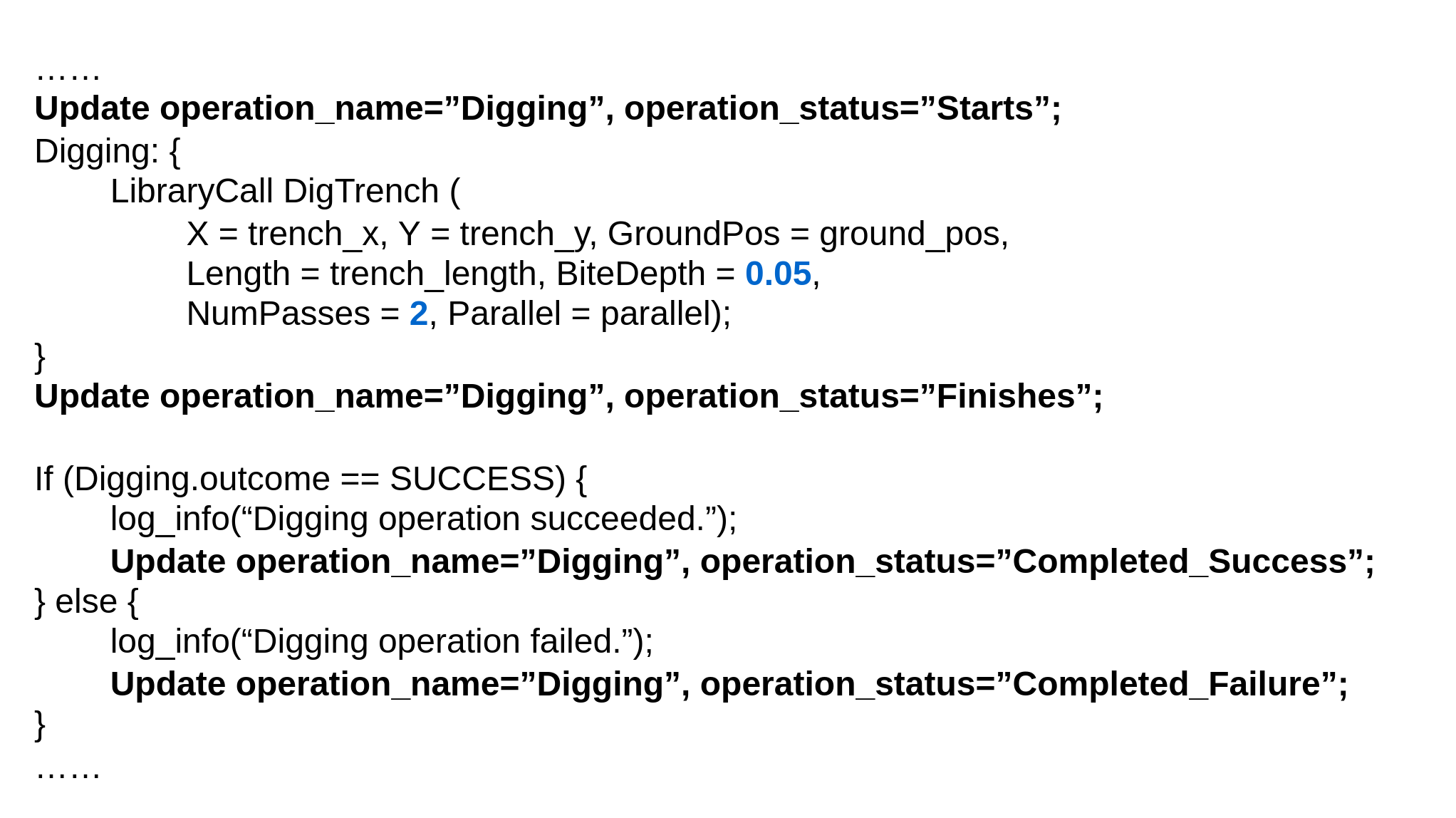
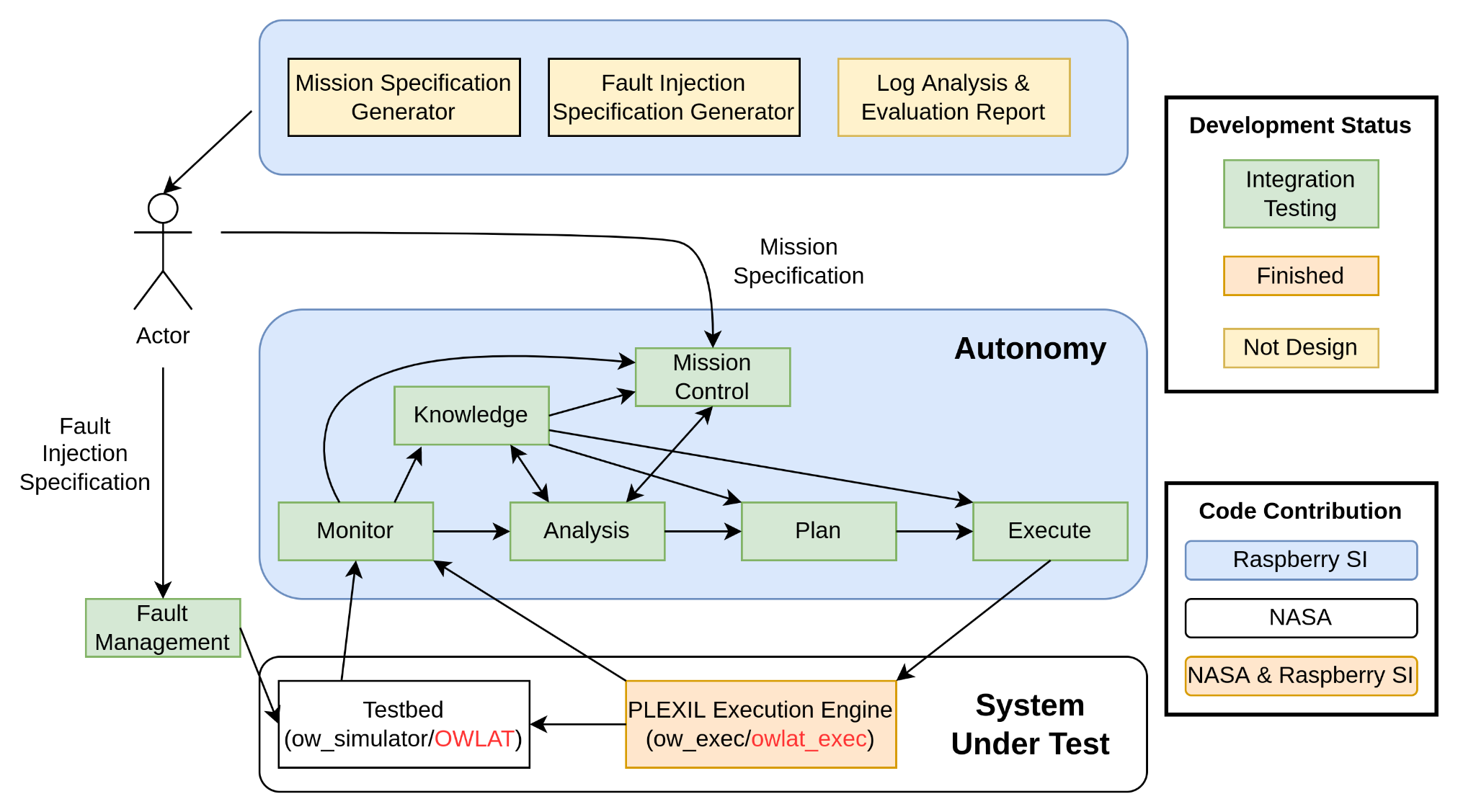


Figure 4 An example of using Update nodes in a PLEXIL plan

Autonomy and Evaluation Infrastructure



[Descriptions of Adaptation Cases]

case 1: digging fails due to an incorrectly estimated excavation probability

Monitoring Information

- status of digging operation

- arm fault signal

(simualte the failure of digging by checking excavation probability p against a random

number p' between 0 and 1. And trigger an arm faul if p' > p since it indicates a

digging failure. The checking will only be checked after the digging operation starts.

Currently, the checking is performed in the analysis componenet.

Arm fault can have other semantics in other cases)

Adaptation Trigger:

- when an arm fault signal is notified during digging operation

(starts but not finishes), it indicates an digging failure.

Adaptation:

- Unstow the arm

- Remove the current excavation location and synthesize a new plan with the rest

locations

case 2: the beliefs of supporting machine-learning models drop below predefined thresholds

Monitoring Information

- the number of digging failures due to incorrectly estimated excavation probability

Adaptation Trigger:

- number of failure == 2, update the excavation probability model

- number of failure == 3, update both excavation probability model andn science model. That is,

a new list of locations will be provided.

Adaptation:

- Unstow the arm (this is specific for the excavation scenario)

- Update models

- Update runtime information (the lists of locations for excavation scenario)

- Synthesize a new plan

case 3: surface vibration caused by earthquake

Monitoring Information

- vibration level

Adaptation Trigger:

- vibration level: (a level of 0 is considered as normal and no adaptation is required for it)

\* level == 1: the terrain has be changed. The models and runtime info need to be updated

Adaptation:

\* level == 1: stop the current plan, unstow the arm, update models and runtime info and

re-planning after the vibration level comes back to 0

case 4: an arm fault is detected during executing a plan for excavation task

Monitoring Information:

- arm fault signal

Adaptation Trigger:

- the arm fault is notified

Adaptation:

- number of failures will be increased

- use the adaptation in either case 1 or case 2 depending on the current number of failures

case 5: a manual plan sent from the Earth center

Monitoring Information

- hasManualPlan

- plan name

Adaptation Trigger:

- hasManualPlan: True

Adaptation:

- Terminate the current plan

- Drive the lander to run the manual plan

case 6: has a new task and need to terminate the current task

Monitoring Information

- has\_new\_task

- terminate\_current\_task

Adaptation Trigger:

- has\_new\_task: True

- terminate\_current\_task: True

Adaptation:

- Terminate the current plan

- Move the next task to the current task (update related variables)

- Synthesize a plan for the new task and run it

case 7: the current task finishes and has a new task needs

Monitoring Information

- has\_new\_task

- terminate\_current\_task

Adaptation Trigger:

- has\_new\_task: True

- terminate\_current\_task: False

Adaptation:

- Move the next task to the current task (update related variables)

- Synthesize a plan for the new task and run it

[The order of adaptation priority]

Order Priority Case

0 High case 5 - Manual Plan, hasManualPlan == True

1 High case 3 - Earthquake, vibration level 1

2 High case 6 - Do a new task

3 Medium case 2 - Decreased Belief, # of failures == 3

4 Medium case 2 - Decreased Belief, # of failures == 2

5 Low case 1 - Digging Failure

6 Low case 7 - Starts New Task

Note: case 4 falls through case 1 and case 2.