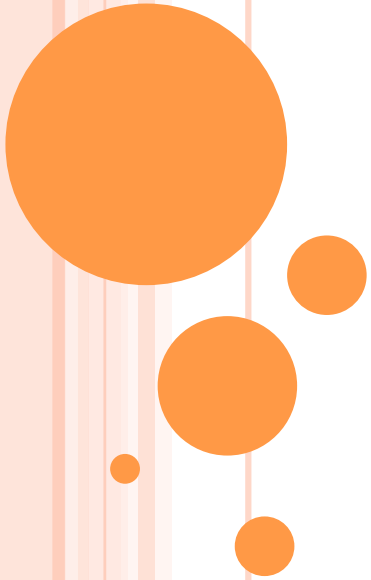


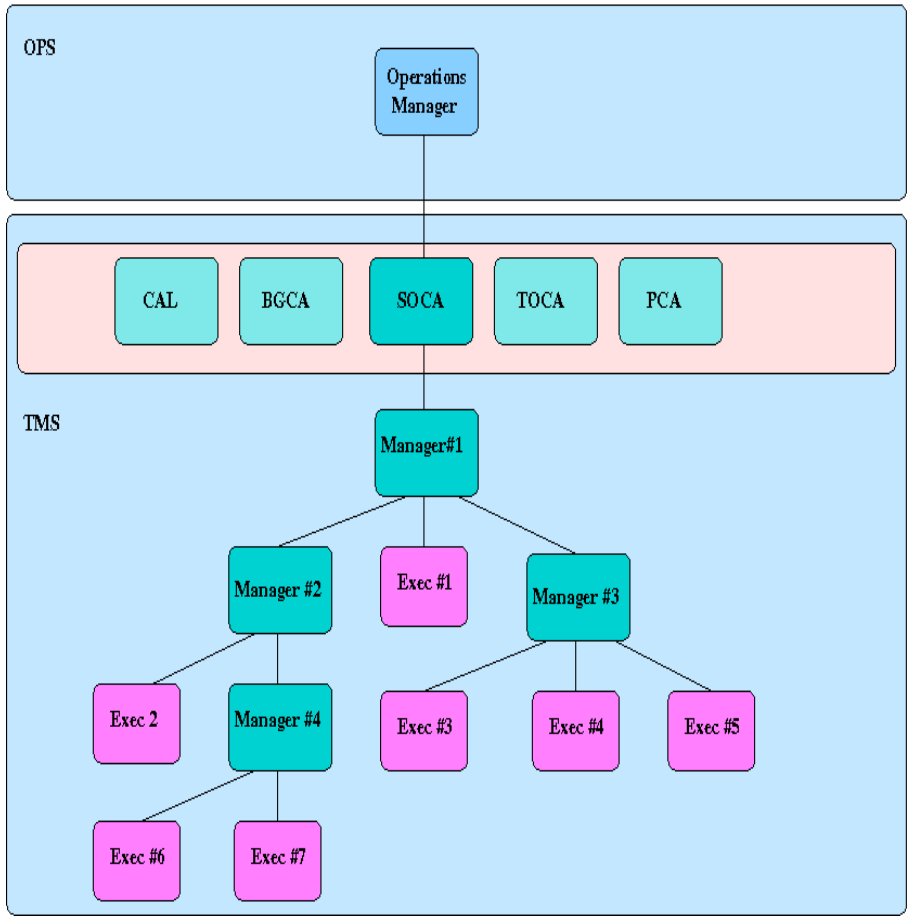
ROBOTIC CONTROL SYSTEM

TASK MANAGEMENT SYSTEM

Task Management System

- The TMS is made up of a large number of Tasks.
- They are all either *managers* or *executives.*
  - Arranged in a vertical control hierarchy.
  - Horizontal control of sequencing.
- Executive tasks perform some sort of action.
  - Generally on an external system.
  - Using a socket connection or similar.
  - Typically they send a command and receive and process a reply.
- Manager tasks control other managers and executives

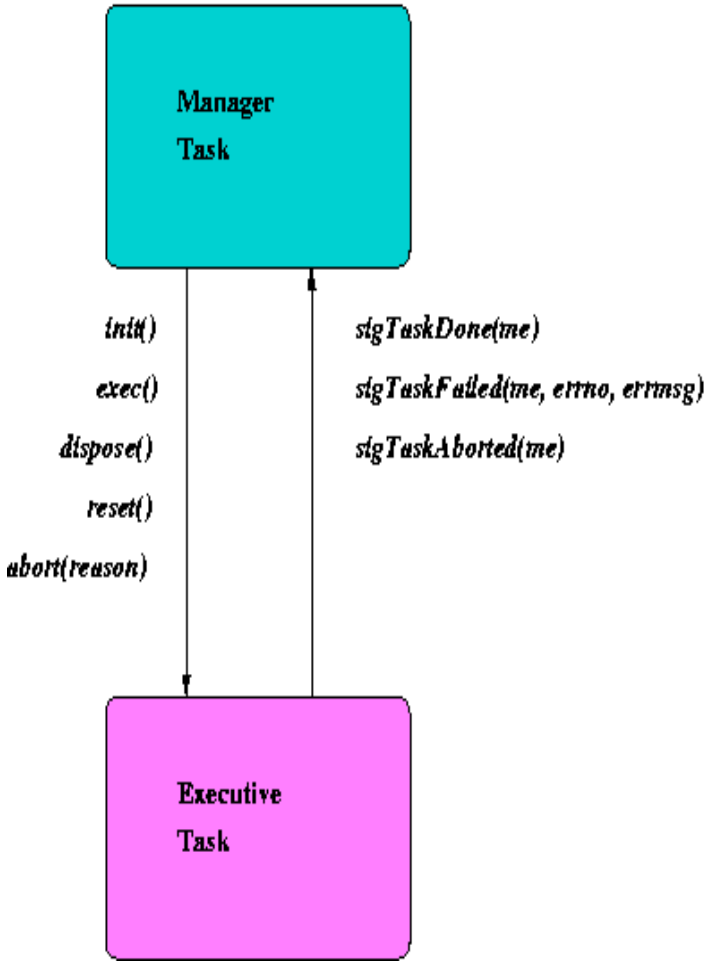
TMS Hierarchy



TOP LEVEL TASKS

- OperationsManager
  - receives action commands from StateModel.
  - creates and manages top-level TMS tasks.
- Types of top-level task
  - Transients - invoked during state-model state changes.
    - Open, Close, Startup, Stop, Initialize, Finalize.
    - Run once, then disposed of when complete.
  - Mode controllers - invoked while state-model in OPERATIONAL state.
    - BGCA, SOCA, TOCA, CAL, PCA.
    - Run several jobs one after another on request from OperationsManager

Task Control



Task Control - (downstream)

- init ()
  - initialize task.
- exec()
  - assigns worker thread to task and executes.
- abort()
  - task stops what its doing.
- dispose()
  - task disposes any resources e.g. sockets.
- reset ()
  - resources reset to pre INIT values so can be re-run under new worker .

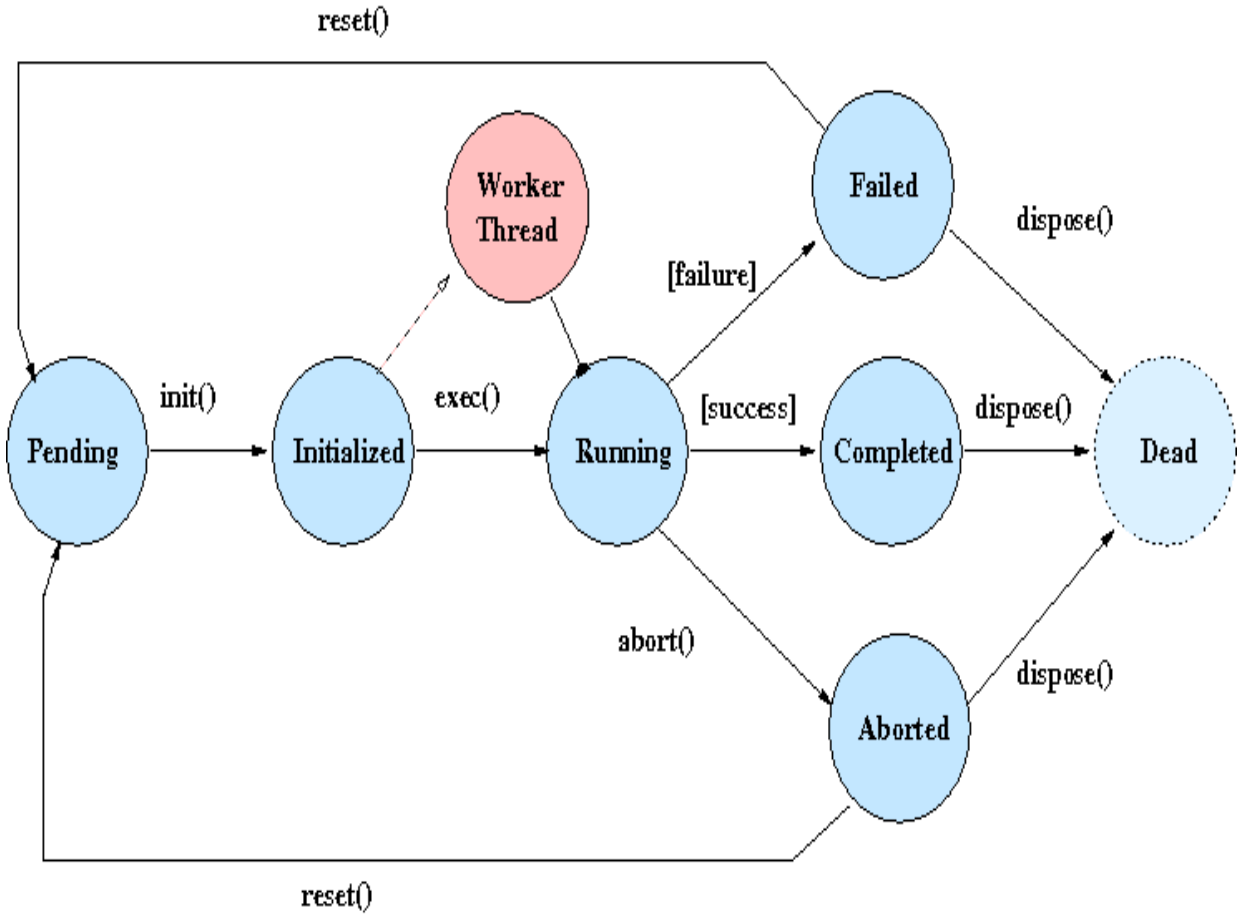
Task Control - (upstream)

- sigTaskDone ()
  - notification that task has completed successfully.
- sigTaskFailed (errno, errmsg)
  - notification that task has failed with error-code and reason.
- sigTaskAborted ()
  - notification that task has completed aborting.
- sigMessage(msgtype, msgvalue)
  - notification of some useful information.

Task information

- A manager has not completed until all its sub-tasks are completed.
- On sub-task failure, a manager can decide to:-
  - skip or restart a failed subtask.
  - take some corrective action.
  - fail itself.
- When a manager is aborted, it must abort any pending sub-tasks before signalling to its own manager that it has completed aborting.

Task Life-cycle



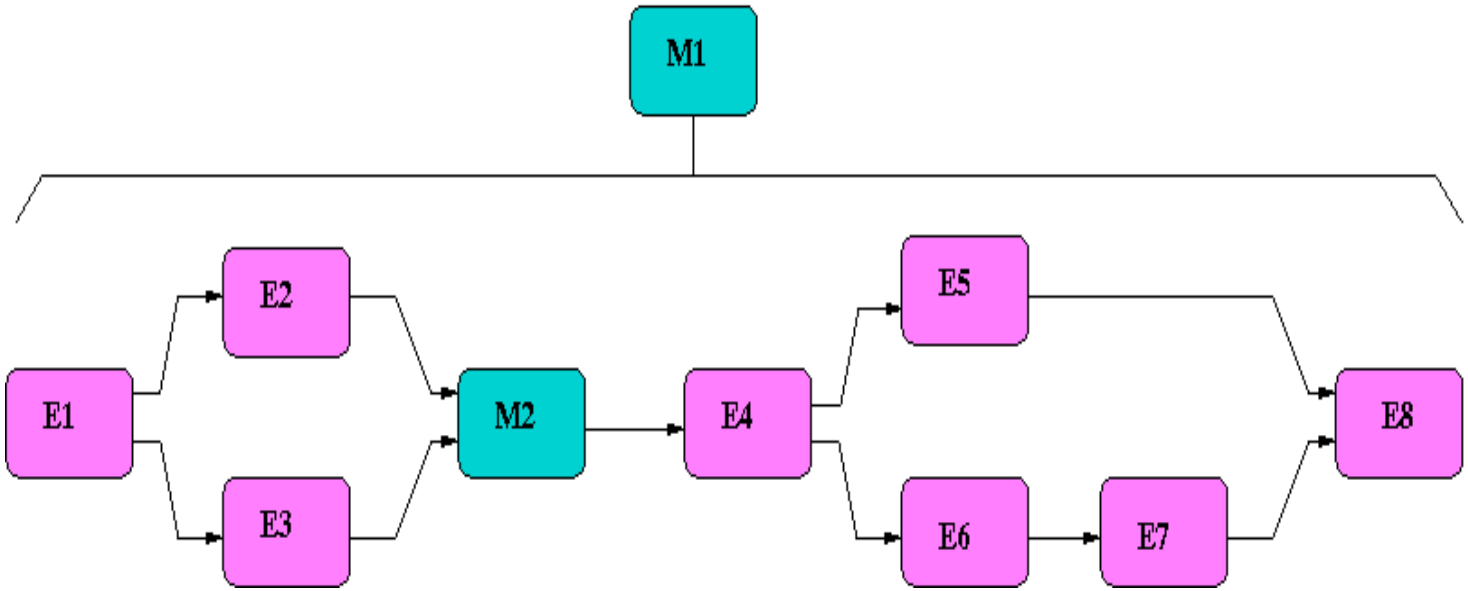
Life-cycle detail (executive)

- Manager creates task -> PENDING
- Init () -> INITIALIZED
  - setup connection and command.
- Exec () -> RUNNING
  - worker thread assigned.
  - send COMMAND
  - await and process ACK(s)
  - await and finally process COMMAND_DONE
  - close streams and socket.
  - signal completion or failure to manager.
- Abort () -> ABORTED
  - break connection if open.
  - signal abort completed to manager.
- Dispose ()
  - resources cleared up.

Life-cycle detail (manager)

- Manager creates task -> PENDING
- Init () -> INITIALIZED
  - pre-init behaviour.
  - create list of immediate sub-tasks.
  - link sub-tasks in sequence.
  - post-init behaviour.
- Exec () - RUNNING
  - worker thread assigned.
  - loop until all sub-tasks are done, failed or aborted.
  - process notifications from sub-tasks. onSubtaskFailed, onSubtaskCompleted, onSubtaskAborted behaviours.
  - dispose of failed and completed subtasks.
  - start (i.e. int () and worker.exec()) any sub-tasks in sequence which are pending and for which all predecessors are done.
- Abort () -> ABORTING
  - send abort () to any PENDING sub-tasks.
- Dispose ()
  - destroy task list.

SEQUENCING



SEQUENCING

- Sub-tasks under a manager can be run sequentially or in parallel or a complex combination.
- Sometimes it is easier to create a new sub-manager to handle part of the sequence.
  - e.g. on the previous slide the tasks E2, E3 could have been put under a manager M1.1 which runs them in parallel. The sequence would then be:- E1 -> M1.1 -> M3 ...
- This is more useful when the number and type of sub-tasks to sequence is variable and dependant on some parameter passed to the manager.
  - See:- *~dev/src/rcs/java/ngat/rcs/oldscience/ObservationSequenceTask.java*
- Use of BarrierTask to act as an assembly point for task sequences.
  - Does nothing much except wait until its predecessors have completed.
  - Very little used nowadays but still useful.
  - See: *~dev/src/rcs/java/ngat/rcs/tms/manager/InitializeTask.java*

Manager task implemention

- Most managers extend an abstract base class:-
  - *ngat.rcs.tms.manager.ParallelTaskImpl*
- Provides skeleton behaviour to create and manage the sequencing of sub-tasks.
- Life-cycle points where specialized behaviour can be performed:-
  - onAborting()
  - onCompletion()
  - onDisposal()
  - onFailure()
  - onInit()
  - onStartup()
  - onSubtaskAborted()
  - onSubtaskDone()
  - onSubtaskFailure()

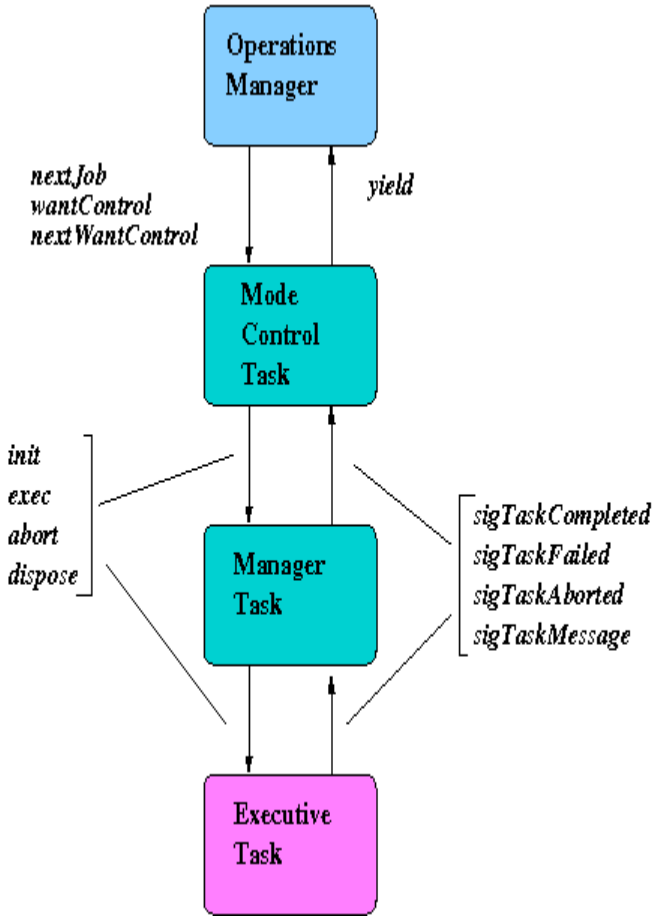
EXECUTIVE TASK IMPLEMENTATION

- Most executives extend an abstract base class:-
  - *ngat.rcs.tms.executive.JMSTaskImpl*
- Provides a basic skeleton for connecting to an external server via JMS protocol, handling ACKs and receiving the final DONE reply.
- Life-cycle methods provide a means of adding specialized behaviour:-
  - calculateTimeToComplete()
  - canAbort()
  - handleAck(ACK)
  - handleDone(COMMAND_DONE)
  - onCompletion()
  - onDisposal()
  - onInit()
- For an example of what might need doing in the handleAck() and handleDone() methods.
  - See:- *~dev/src/rcs/java/ngat/rcs/tms/executive/Exposure_Task.java*

OBSERVING

- Once the StateModel has gone into the OPERATIONAL state, substate OBSERVING, the OperationsManager interrogates each of the ModeControllers in priority order using a call to.
  - *wantControl() : boolean*
- Whichever (highest priority) controller replies in affirmative is then selected to take control and that task is inserted at the top of the task hierarchy.
- It is then requested to perform a single job, one of its own top level tasks, via a call to:-
  - *nextJob() : void*
- Once that job (a high level task) is completed or failed the controller task yields to the Operationsmanager via:-
  - yield() : void
- The cycle then continues. If the same top-level controller is picked, it is simply told to do another job. If a different (higher priority) controller is picked, the current controller is aborted and the new controller installed in its place.

Task Control – (Observing)



Observing Mode priority

- The top-level Mode Control Tasks are, in priority order:-
  - TOCA *(ngat.rcs.tocs.TOControlAgent)*
    - only requests control when a TO interrupt has occured and does so without waiting to be asked, it sends a *yield()* immediately to take control.
  - CAL *(ngat.rcs.calib.CalibrationControlAgent)*
    - only requests control at predefined times for Skyflats, Telfocus, previously also for Standards
  - SOCA *(ngat.rcs.sciops.ScienceControlAgent)*
    - always requests control except very early in the evening, very late in the morning and for a minute or two after a failed scheduling attempt.
  - BGCA *(ngat.rcs.tms.manager.BackgroundControlAgent)*
    - Always requests control except when the sun is too high.