

ROS2 Navigation

Introduction, Overview and Status
8/2/18

Target Use Cases

- 2D Navigation
 - Warehouse robot
- 2D Navigation + elevation
 - Search and rescue robot
- Indoor Navigation in multi-story building
 - Room service robot
- Outdoor Navigation
 - Campus delivery robot

Stretch Target

- 3D Navigation
 - Drones

More info: https://github.com/ros-planning/navigation2/tree/master/doc/use_cases

Target Improvements (based on feedback)

- Modularity / ability to easily replace planners & localization algorithms
 - Use (language neutral) ROS Nodes and Actions instead of plugins and C++ API calls
- Map Abstraction - Easy to use different map types
 - ex: OccupancyGrid, grid_map, octomap
- Ability to add special properties / zones to maps
 - ex: keep out zones, slow zones, directional lanes
- Flexible / dynamic state machines and recovery methods
- Ability to support different robot types
 - ex: Ackerman steering
- Extensibility for tasks such as docking, moving into elevators, etc

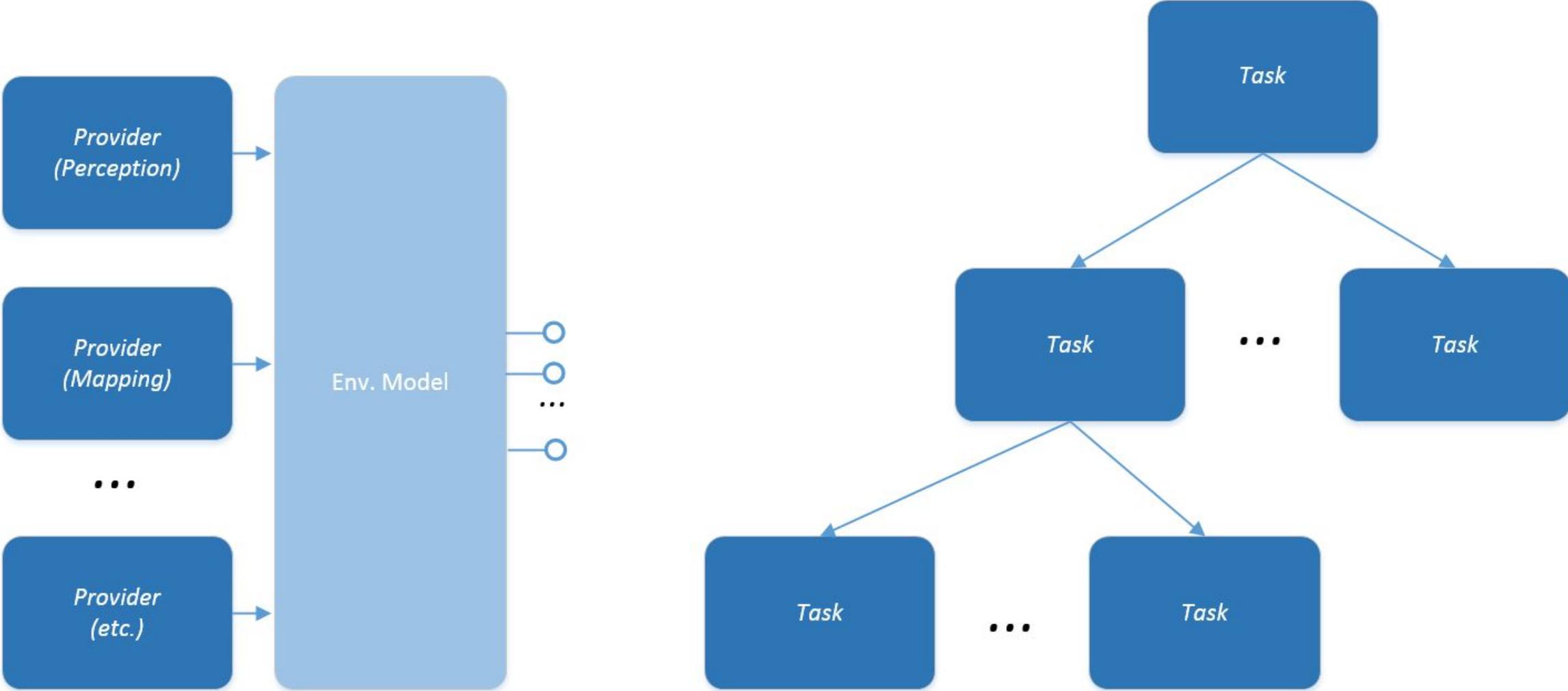
Additional Improvements

- Modular, extensible framework
- Allow for integration various kinds of tasks, both navigation and non-navigation
- Generalize the base code as a task dispatch engine (e.g., Behavior Tree)
- Integrate recovery behaviors as tasks (ROS Actions)
- Allow for different deployment architectures
 - For example, central management of delivery robots bringing work items to robot at a work pod
- Provide an Environmental Model
 - Extend the framework to provide interfaces to Perception, Prediction, Mapping, etc.
 - Make the development of task code simpler and more uniform

High-Level Architecture

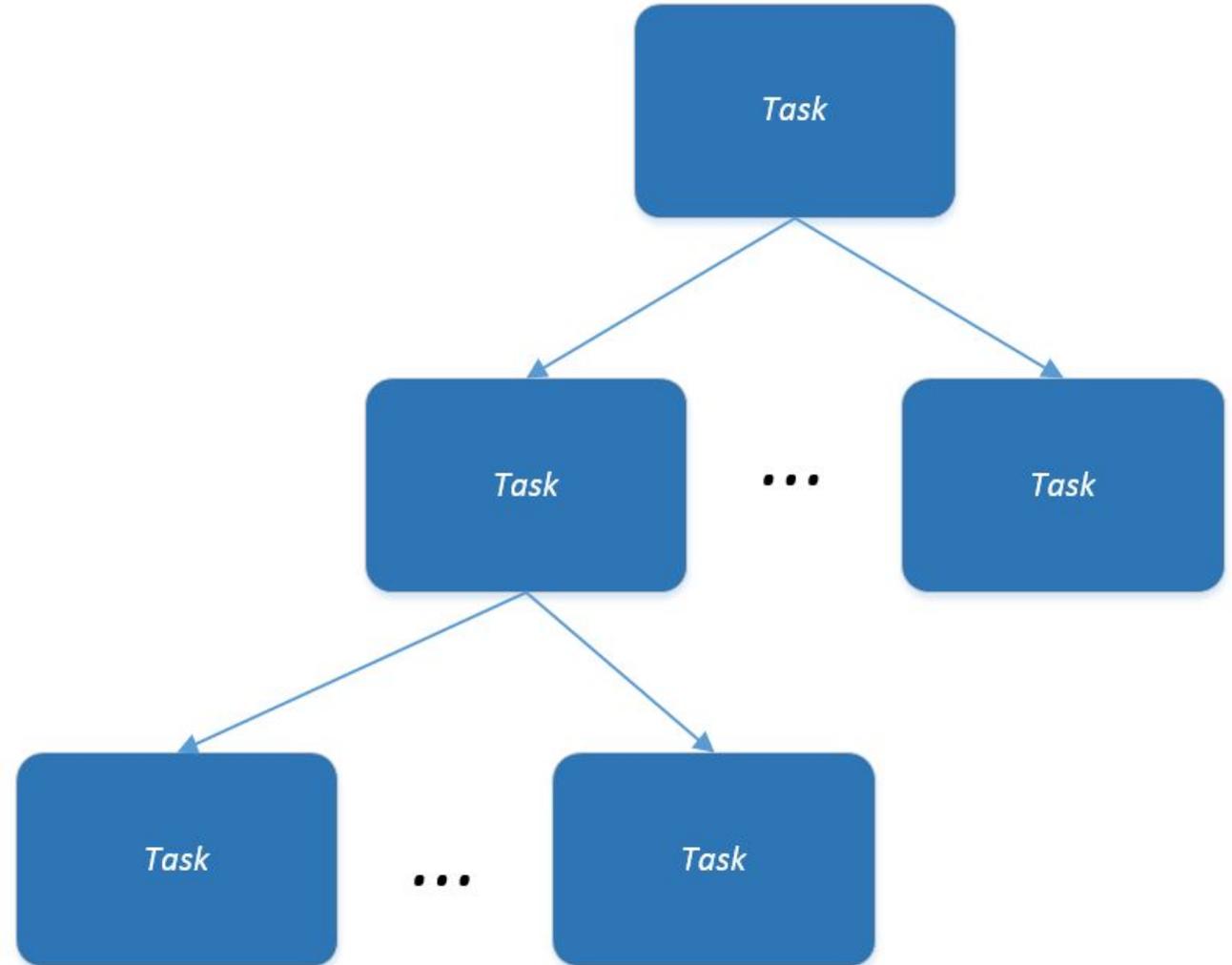
- Command Chain
 - Robot Management System (“orchestration”)
 - Mission Planning
 - Mission Execution
 - Tasks
 - Robot
- Support Infrastructure
 - Perception
 - Prediction
 - Mapping
 - Localization

High-Level Architecture

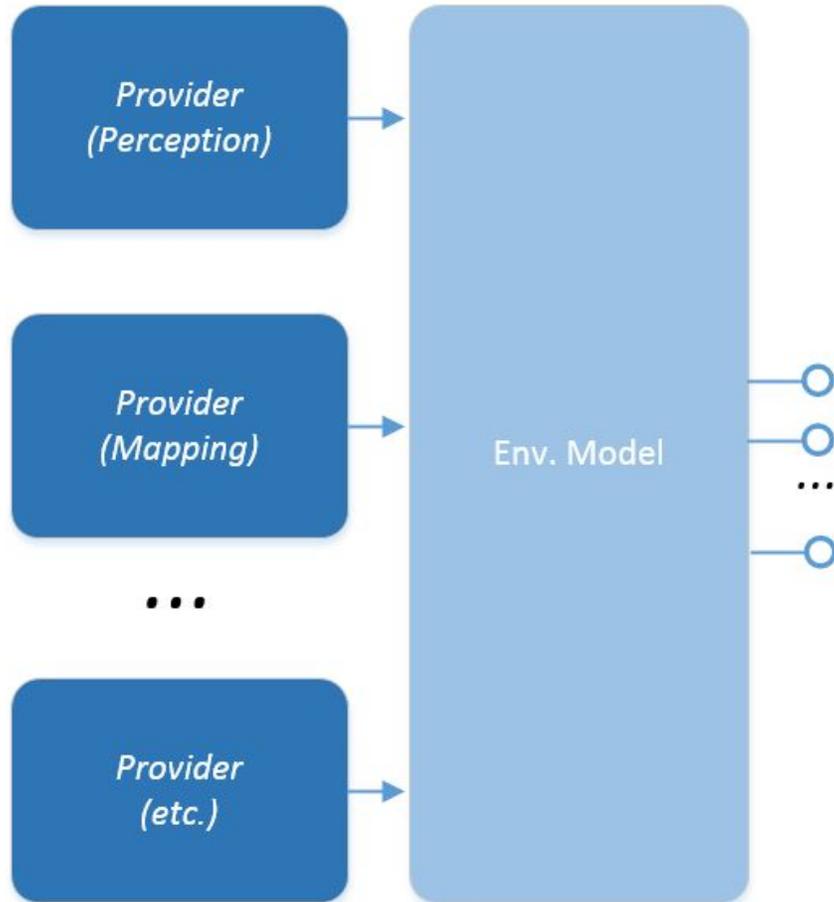


High-Level Architecture

- **Task Hierarchy** - Tasks can be composed from other (sub-)tasks
- **Task Coordination** - Can coordinate tasks w/ simple code, state machine, or Behavior Trees, depending on complexity
- **Programming Language Independence** - As ROS Actions, tasks can be implemented in various languages, encouraging experimentation

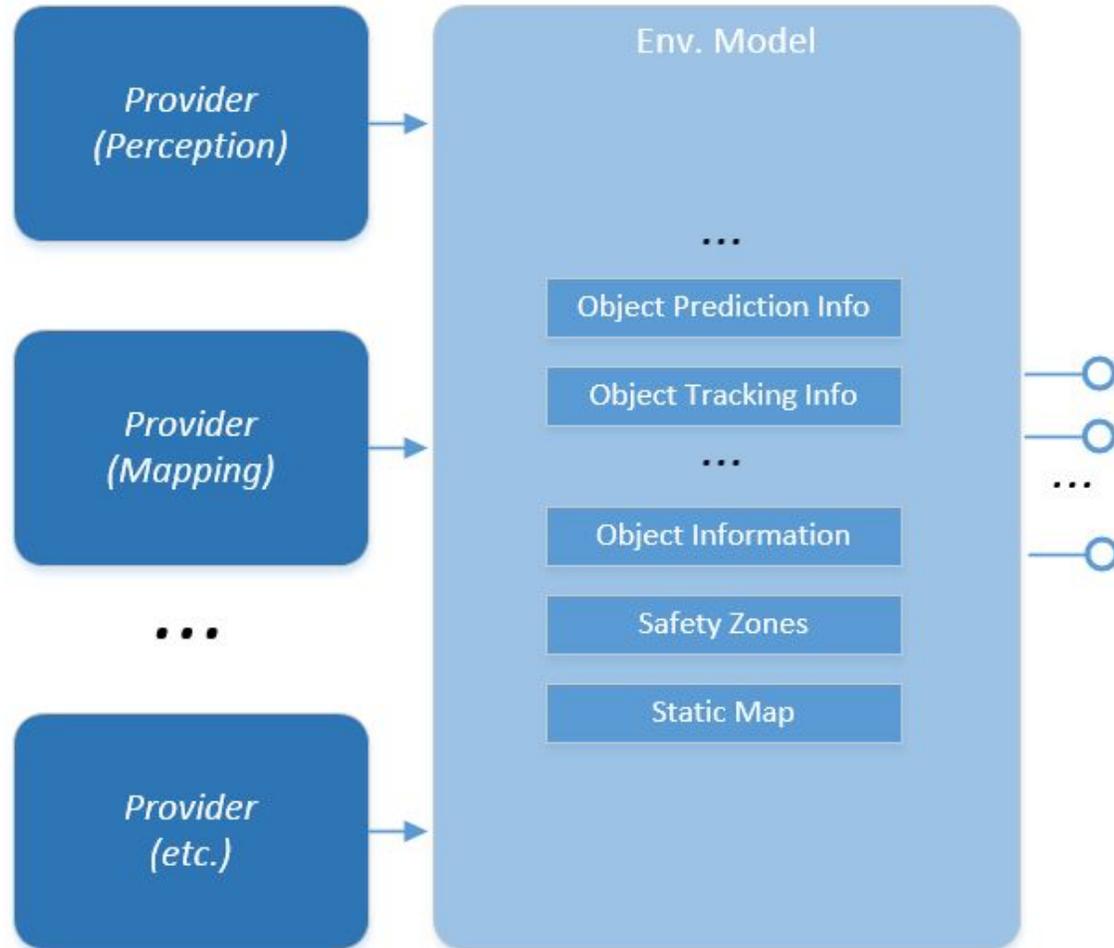


High-Level Architecture



- **Environmental Model** - Aggregates knowledge of the environment.
- **Various Input Sources** - Input from Perception, Mapping, Prediction, etc.
- **Service Interfaces** - Provides several service interfaces to clients to retrieve info.

High-Level Architecture

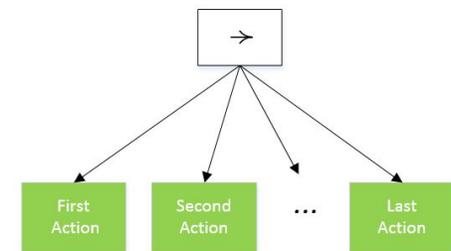


- **Abstraction** - Provide higher-level abstractions of what task implementers need.
- **Encapsulation** - Internal representation for the environmental model may vary

A Mission Plan could be a detailed Behavior tree, incorporating recovery nodes.



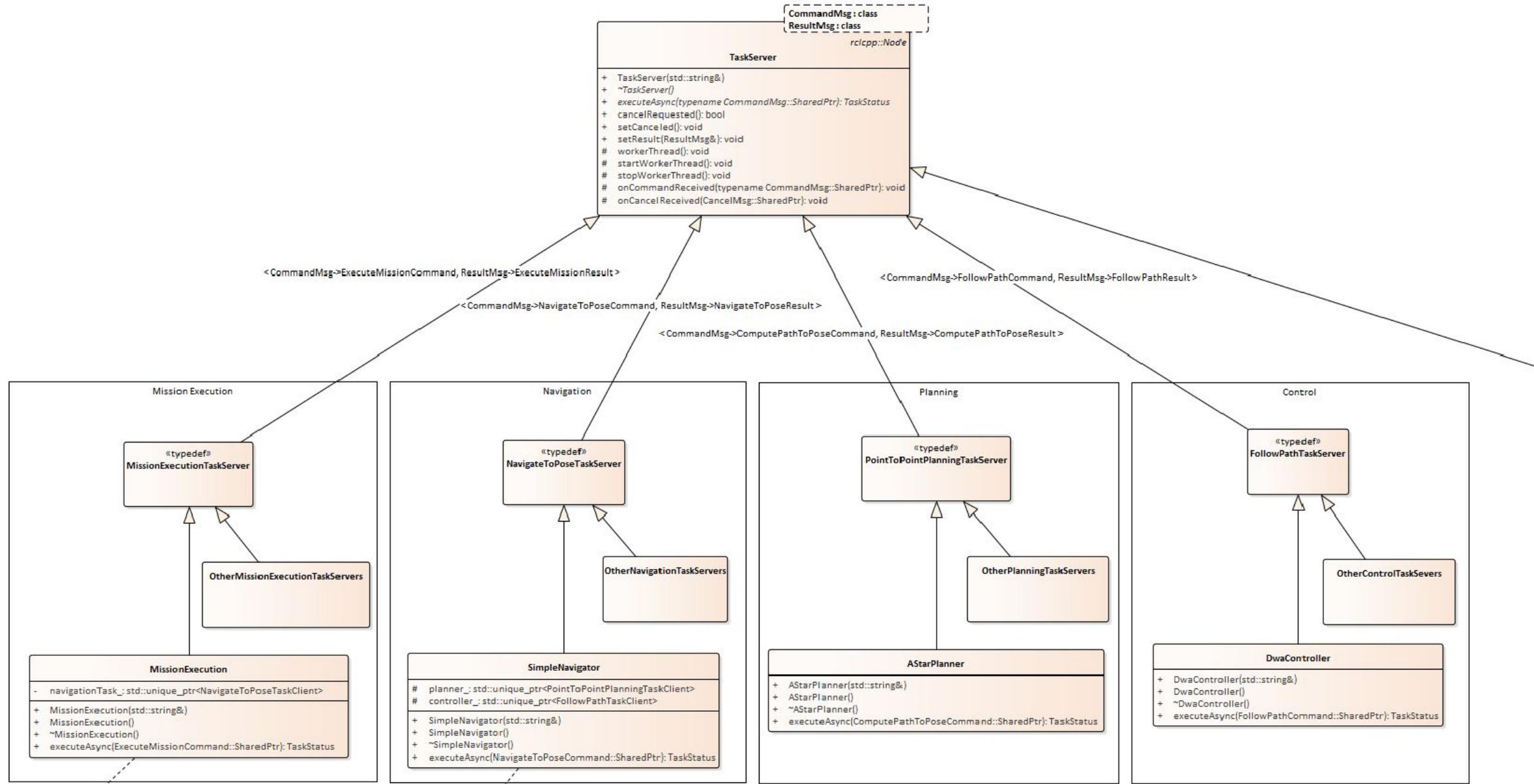
A task can coordinate subtasks using hand-coded logic, state machines, or behavior trees, depending on the complexity of the task coordination. For example, a simple BT sequential node could be used to sequence tasks:



Status

- Posted first revision of requirements:
<https://github.com/ros-planning/navigation2/blob/master/doc/requirements/requirements.md>
- Completed initial design for Command Chain
 - Developed a short-term stand-in for ActionLib, 'til the ROS2 version is ready
 - Implemented an executable shell of the navigation task hierarchy
- Starting a Behavior-Tree-based implementation of Mission Execution
 - Input a Mission Plan (specification of a behavior tree), execute using a BT library
- Implementing some Tasks
 - NavigateToPose using a Point-to-Point Planner (A*) and a Controller (DWA)
 - Porting A* and DWA to ROS2
- Defining services interfaces for mapping, perception, etc.

Backup



Behavior Tree implementation of Mission Execution

Simple sequential execution of planner and then controller