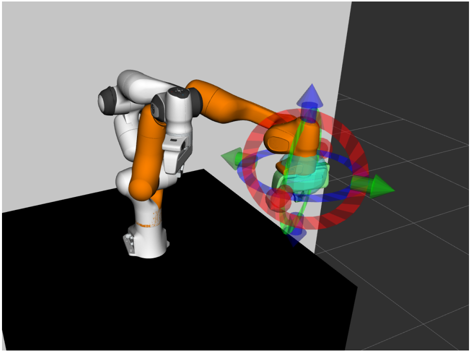
The following presents an overview over a complex architecture with many details simplified in an attempt to provide a comprehensible structure.

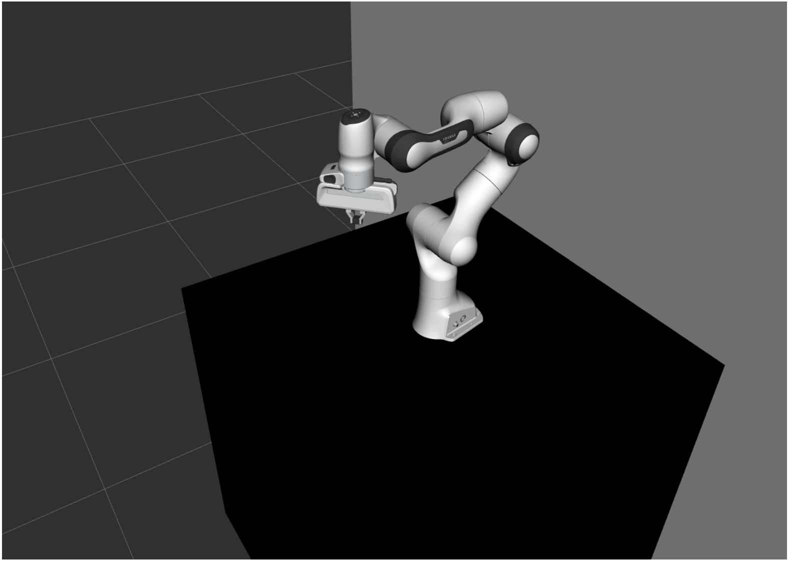
Beware “**Lies for Children**”.

MotionPlanningDisplay



- Interactive Markers added for **chain tips** and **end_effectors**
- Only available if IK solver is available for **JointModelGroup**
- Drag&Drop fails if IK solver fails

PlanningScene



PlanningScene

```
$ rosmsg show moveit_msgs/PlanningScene
```

```
moveit_msgs/RobotState robot_state
  sensor_msgs/JointState joint_state
  sensor_msgs/MultiDOFJointState multi_dof_joint_state
  moveit_msgs/AttachedCollisionObject[] attached_collision_objects
  bool is_diff
geometry_msgs/TransformStamped[] fixed_frame_transforms
moveit_msgs/AllowedCollisionMatrix allowed_collision_matrix
moveit_msgs/ObjectColor[] object_colors
moveit_msgs/PlanningSceneWorld world
  moveit_msgs/CollisionObject[] collision_objects
     ...
    string[] subframe_names
    geometry_msgs/Pose[] subframe_poses
  octomap_msgs/OctomapWithPose octomap
bool is_diff
```

Motion Planning Interface

```
$ rosmsg show moveit_msgs/MotionPlanRequest

moveit_msgs/RobotState start_state
moveit_msgs/Constraints[] goal_constraints
  string name
  moveit_msgs/JointConstraint[] joint_constraints
  moveit_msgs/PositionConstraint[] position_constraints
  moveit_msgs/OrientationConstraint[] orientation_constraints
  moveit_msgs/VisibilityConstraint[] visibility_constraints
moveit_msgs/Constraints path_constraints
  ...
string group_name
string planner_id
...

moveit_msgs/PlanningOptions options
```

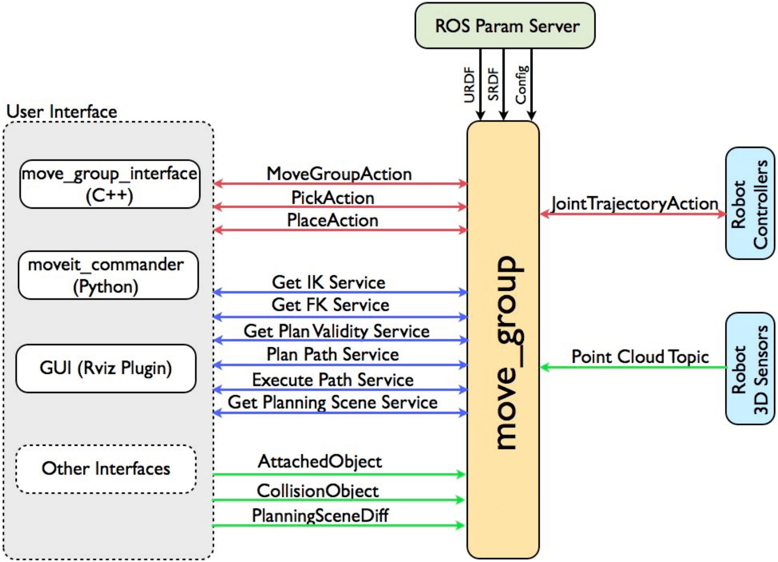
Specifying Goals

Typical Usage

- Joint Target
  - RViz display
  - Pre-specified poses
  - Default: 0.02 deg tolerance
- Pose Targets
  - Most-requested application
  - `MoveGroupInterface::setPoseTarget()`
  - Default: 1mm^3, 0.1 deg tolerance

Tolerances are intrinsic in the **planning problem, not controlling**.

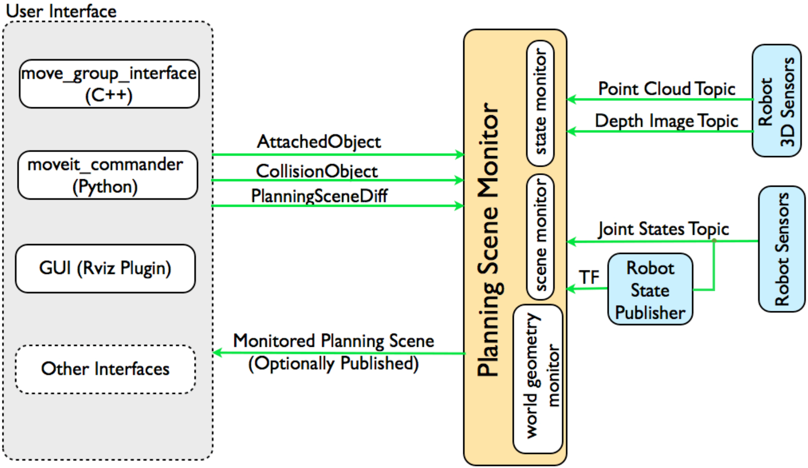
move_group node



MoveGroupContext

```
struct MoveGroupContext
{
  bool status() const;

  planning_scene_monitor::PlanningSceneMonitor psm;

  planning_pipeline::PlanningPipeline pipeline;

  plan_execution::PlanExecution execution;
  trajectory_execution_manager::TrajectoryExecutionManager tem;

  plan_execution::PlanWithSensing plan_with_sensing_;

  bool allow_trajectory_execution_;
};
```

PlanningSceneMonitor



MoveGroupCapability

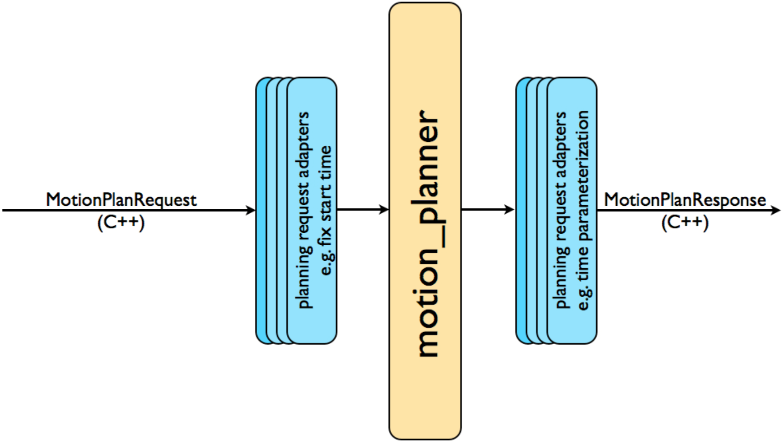
```
class MoveGroupCapability
{
  virtual void initialize() = 0;

  MoveGroupContextPtr context;
};
```

- ClearOctomapService
- ExecuteTrajectoryAction
- ApplyPlanningSceneService
- StateValidationService
- CartesianPathService
- MoveAction

MoveGroupAction

1. Lock Planning Scene for planning
2. Overlay Scene if requested
3. **Invoke PlanningPipeline**
4. Unlock Scene
5. If solution is found, **execute solution**

PlanningPipeline



PlanningPipeline

Standard Pipeline:

- FixStartStateBounds
  - Adjust reported current state (soft limits)
- FixStartStateCollision
  - Wiggle around “minor” collisions
  - Parameter `jiggle_fraction`
- FixStartStatePathConstraints
  - Satisfy path constraints before planning on
- Hand over to PlannerManager

OMPL Planning

- Select **Planning Space** based on constraints:
  - Orientation Path constraint: SE(3) with IK projection
  - Otherwise: Joint Space
- Create **custom ConstraintSampler** for goals
  - Run asynchronously
  - Uniform distribution
  - Pose goals are sampled **in SE(3)** before IK
    - IK receives "`bool isValid`" callback to test solutions
- Build (RRT) Sampling Trees
  - Parallelized planning
  - **State-based collision checking**
  - Interpolated trajectories might contain collisions
    - Parameter: `longest_valid_segment_fraction`
    - "Continous" collision checking does **not** resolve this problem
- Post-Processing: Refine / Smooth solution

Time Parameterization

Usually, the **kinematic solution** is time-parameterized afterwards

- Iterative Parabolic Time Parameterization (IPTP)
  - traditional implementation in MoveIt
- Iterative Spline Parameterization (ISP)
  - Refined approach based on local quintic splines
  - might slightly adjust start and end states
- Time Optimal Trajectory Generation (TOTG)
  - Only became available upstream this year
  - Might adjust whole trajectory within bounds to generate optimal parameterization w.r.t. speed parameters

MoveIt Controller Interfaces

- Controller integration defined through abstract `MoveItControllerManager` class interface
- In practice: almost everyone uses `moveit_simple_controller_manager` which maps this to `control_msgs/FollowJointTrajectoryAction`
- Either implemented by custom nodes or via a `ros_control` `HardwareInterface`
- If actions are not available (or not correctly configured) trajectories can not be executed

Execute Planned Trajectory

- Verify start state is valid for trajectory
  - Not all hardware controllers check for this!
  - Parameter `allowed_start_tolerance`
- Split trajectories among controllers as required
  - Bimanual manipulation
  - Arm & Hand control
- Monitor unexpected changes in the PlanningScene
  - Stop & Replan if future collision expected
- Monitor runtime of controllers
  - Abort if execution takes longer than expected
  - Parameters: `allowed_execution_duration_scaling`, `allowed_goal_duration_margin`

Some Unmentioned Aspects

- `PlanWithSensing`
  - Move visual sensors to get information about potential collisions of planned motions
- 3D Perception
  - Octomap updating via `PointCloud` or `DepthImage` data
  - GPU accelerated
  - Carefully consider update rates during setup
- `computeCartesianPath`
  - Interpolation in SE(3) with IK projections and bounds check
- Realtime Jogging
- ...