AutoAPMS: Lightweight and versatile integration of behavior trees into the ROS 2 ecosystem

FOSDEM 2026 Robotics and Simulation

Robin Müller
Research Associate | PhD Candidate
Technical University Darmstadt

Why Behavior Trees?

AutoAPMS

Intelligent robotics require **decision making** capabilities and **reactive mechanisms** for

- Navigation in dynamic environments
- Interactive manipulation tasks
- Resilient operations in general

Behavior trees promise composability, modularity, and reactivity

Framework Comparison

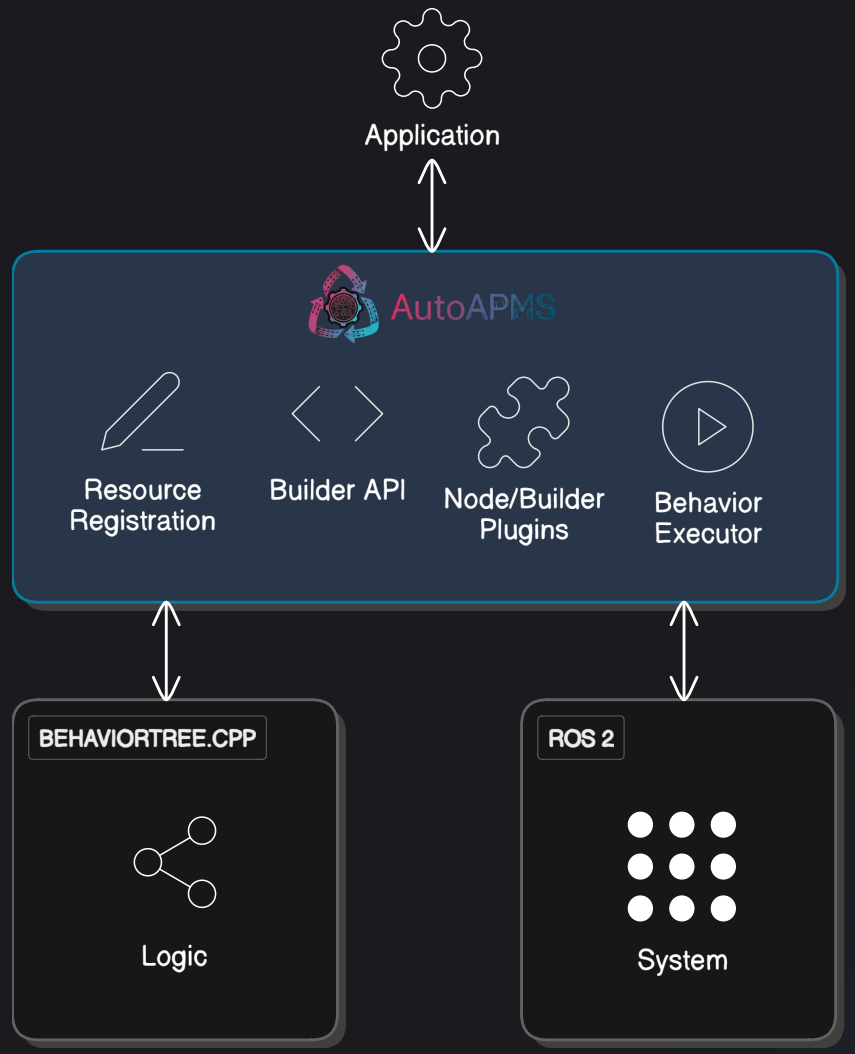
AutoAPMS

ROS 2 Integration	Core Library	Language	Versatility	Functional Extensibility	Developer Experience	Scalability
py_trees_ros	py_trees	Python	🟢	🟢	🟢	🟡
ros2_ros_bt_py	*(Monorepo)*	Python	🟡	🟢	🟢	🟡
BehaviorTree.ROS2	BehaviorTree.CPP	C++	🟢	🟡	🟡	🟡
nav2_behavior_tree	BehaviorTree.CPP	C++	🔴	🟡	🔴	🟡
AutoAPMS	BehaviorTree.CPP	C++	🟢	🟢	🟢	🟢

🟢 High/Good | 🟡 Medium/Problematic | 🔴 Low/Bad

AutoAPMS Design Goals

1. **Domain-agnostic C++ development framework** for behavior-based control and automated planning
2. **Reduced configuration overhead** and lower entry barrier
3. **Focus on modularity and reusability** for distributed workspaces and large projects



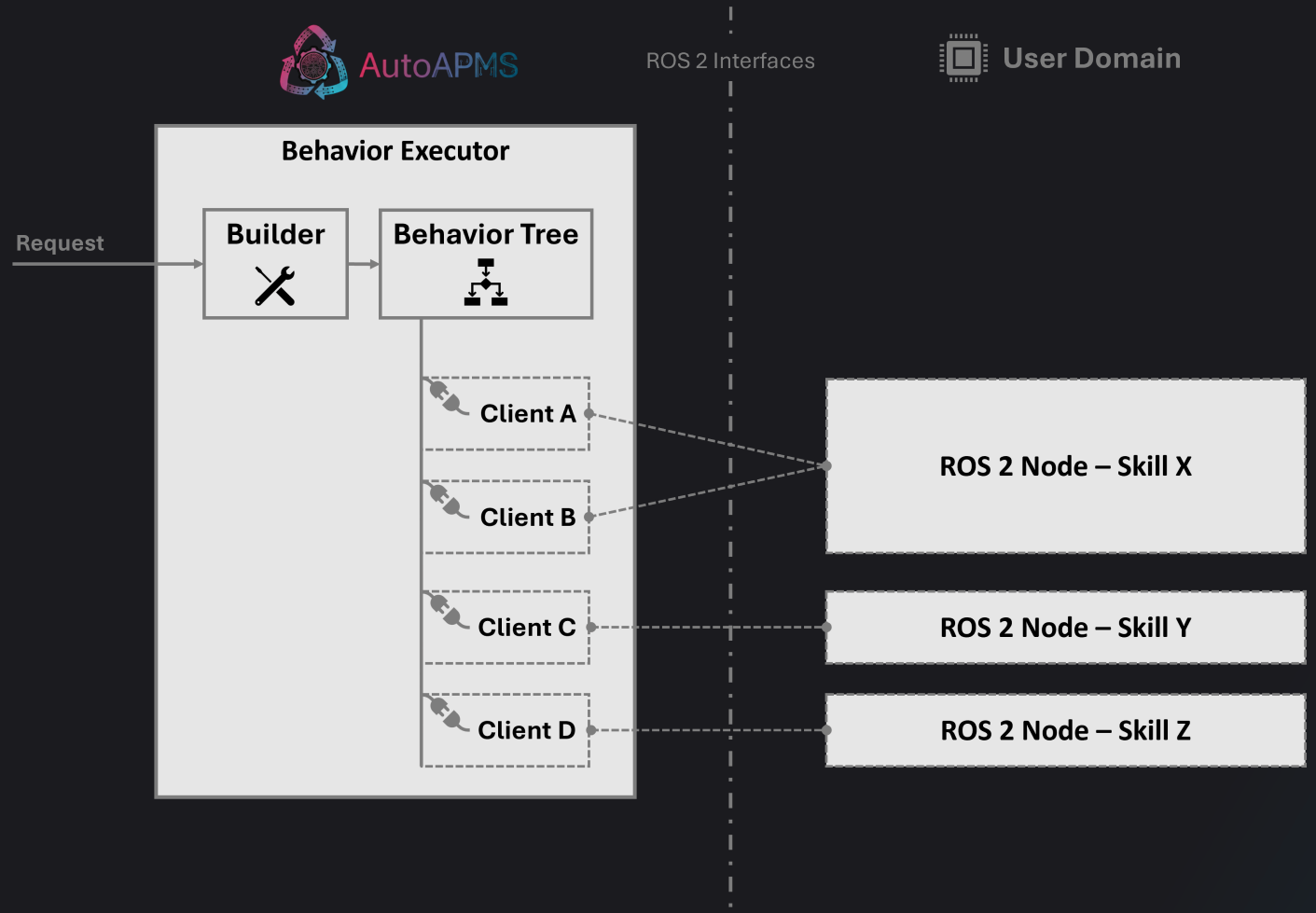
So how does it work?

Technical walkthrough

Robot Architecture

Behavior Trees represent **policies/plans**

They utilize the robot's capabilities through **clients/nodes**



Behavior: Wave Hand

Sensor

for reading hand position

```
float64 position
```

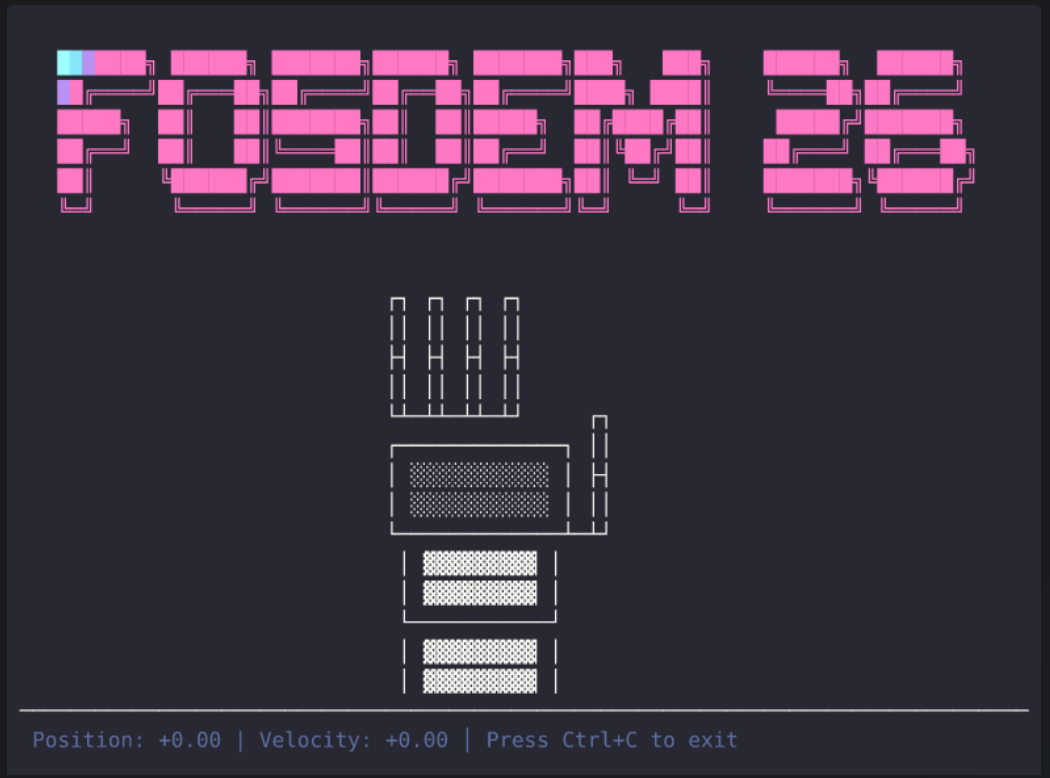
receive ⟵

Actuator

for moving hand translationally

```
float64 velocity
```

send ⟶

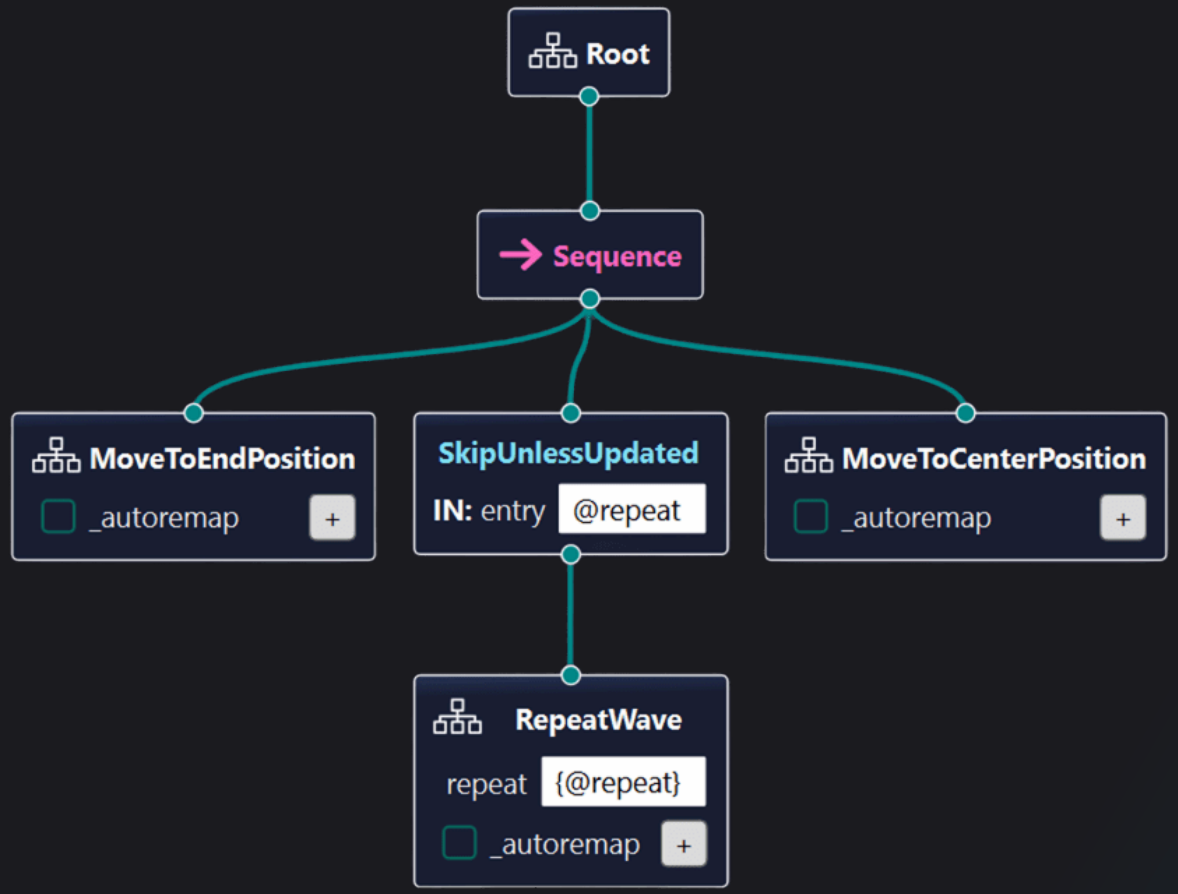


The Build Request

AutoAPMS

Encoded using BT.CPP XML schema

```
<BehaviorTree ID="DoWave">
  <Sequence>
    <SubTree ID="MoveToEndPosition"/>
    <SkipUnlessUpdated entry="@repeat">
      <SubTree ID="RepeatWave"
               repeat="{@repeat}"/>
    </SkipUnlessUpdated>
    <SubTree ID="MoveToCenterPosition"/>
  </Sequence>
</BehaviorTree>
```



The Node Manifest

AutoAPMS

Two underlying implementations are made reusable through YAML configurations

MoveToTheSide

```yaml
MoveToTheSide:
  class_name: fosdem26_autoapms_behavior::VelocityPub
  topic: /robot/velocity_cmd
  port_default:
    # Negative for moving left - positive for right
    velocity: -1.0
```

StopMovement

```yaml
StopMovement:
  class_name: fosdem26_autoapms_behavior::VelocityPub
  topic: /robot/velocity_cmd
  port_default:
    # Zero velocity to stop the hand
    velocity: 0.0
```

HandReachedEndPosition

```yaml
HandReachedEndPosition:
  class_name: fosdem26_autoapms_behavior::PositionInRange
  topic: /robot/position
  port_default:
    # Negative for end on the left - positive for right
    low: -1.0
    high: -0.95
```

HandIsCentered

```yaml
HandIsCentered:
  class_name: fosdem26_autoapms_behavior::PositionInRange
  topic: /robot/position
  port_default:
    # Centered around 0.0
    low: -0.1
    high: 0.1
```

The Node Manifest

AutoAPMS

1. Register reusable C++ implementations

```
auto_apms_behavior_tree_register_nodes(
  behavior_tree_nodes  # Target library
  "fosdem26_autoapms_behavior::VelocityPub"
  "fosdem26_autoapms_behavior::PositionInRange"
)
```

2. Register specific manifests for your use-case

```
set(SIDE left)  # and/or right
auto_apms_behavior_tree_register_nodes(
  wave_${SIDE}  # Alias for node manifest
  NODE_MANIFEST
  "config/common_nodes.yaml"
  "config/wave_${SIDE}_nodes.yaml"
)
```

Easy configuration using CMake macros in an `ament_cmake` package

Key benefits:

- Workspace-wide reusable node registrations
- Compile-time validation avoids runtime errors
- Automatic node model generation

  → XML file for visual editors

  → C++ header for builder API

Deploying the Behavior Tree

AutoAPMS

Registration: Two behaviors — One definition

```
auto_apms_behavior_tree_register_trees(
  "behavior/tree/generic_wave.xml"
  ALIAS_NAMESPACE wave_${SIDE}
  NODE_MANIFEST "fosdem26_autoapms_behavior::wave_${SIDE}"
)
```

Deployment: `ros2 behavior` CLI tool

```
ros2 behavior run <package>::<namespace>::<tree> --blackboard <key>:=<value> ...
```

`<package>` Name of registering package

`<namespace>` Namespace defined during registration

`<tree>` Behavior Tree ID from XML definition

Deploying the Behavior Tree

Deployment:

Wave left

```
ros2 behavior run \
  fosdem26_autoapms_behavior::wave_left::DoWave \
  --blackboard repeat:=2
```

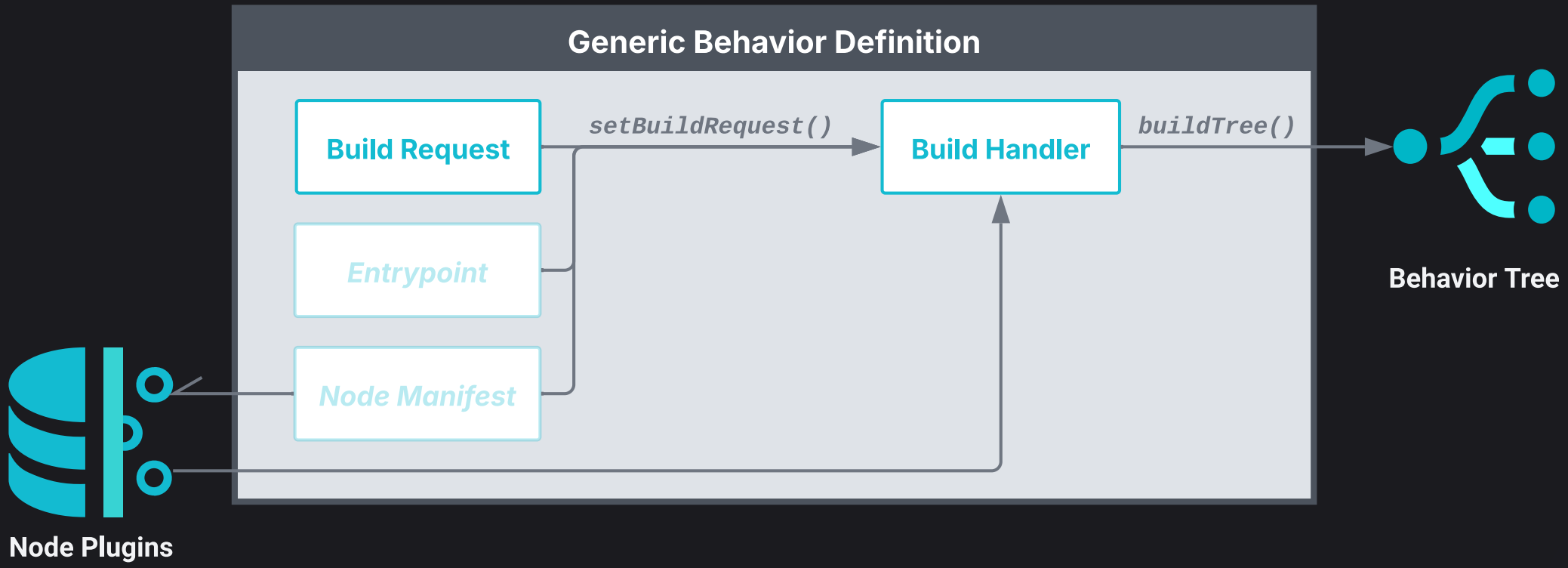
Wave right

```
ros2 behavior run \
  fosdem26_autoapms_behavior::wave_right::DoWave \
  --blackboard repeat:=2
```

Customizable Behavior Generation

More than just plain behavior trees

Customizable Behavior Generation



The Build Handler

Concept Idea

User-defined behavior tree generation logic

What if we want the hand to move like this?

Left → Left → Right → Right → Left → Right

→ New build request message format

- `l` means left — `r` means right
- E.g. `l;l;r;r;l;r`

C++ Behavior Tree Builder API

AutoAPMS

1. Load predefined subtrees

```
TreeDocument doc, doc_left, doc_right;
doc_left.newTreeFromResource("fosdem26_autoapms_behavior::wave_left::MoveToEndPosition")
  .setName("MoveLeft");
doc_right.newTreeFromResource("fosdem26_autoapms_behavior::wave_right::MoveToEndPosition")
  .setName("MoveRight");
```

2. Create the custom tree by parsing the build request

```
TreeDocument::TreeElement tree = doc.newTree("CustomBehavior").makeRoot();
model::Sequence sequence = tree.insertNode<model::Sequence>();
for (const std::string dir : auto_apms_util::splitString(build_request, ';')) {
  if (dir == "l")
    sequence.insertSubTreeNode(doc_left.getTree("MoveLeft"));
  else if (dir == "r")
    sequence.insertSubTreeNode(doc_right.getTree("MoveRight"));
}
return tree;
```

Deploying Custom Behavior Definitions

AutoAPMS

Registration: Register build handler and assign to behavior

```
auto_apms_behavior_tree_register_build_handlers(
  custom_build_handler  # Target library
  "fosdem26_autoapms_behavior::CustomBuilder"
)
auto_apms_behavior_tree_register_behavior(
  "l;l;r;r;l;r"  # May also be defined at runtime
  ALIAS "custom_wave"
  BUILD_HANDLER "fosdem26_autoapms_behavior::CustomBuilder"
  CATEGORY "custom"
)
```

Deployment: `ros2 behavior` CLI tool

```
ros2 behavior run <package>::<alias>
```

`<package>` Name of registering package `<alias>` Behavior alias defined during registration

Deploying Custom Behavior Definitions

Deployment:

Static resource

```
ros2 behavior run \
  fosdem26_autoapms_behavior::custom_wave
```

Dynamic request

```
ros2 behavior run \
  --build-request "l;l;r;r;l;r" \
  --build-handler fosdem26_autoapms_behavior::CustomBuilder
```

Runtime-Configurable Executor

AutoAPMS

ROS 2 Action

```
string build_request
string build_handler
string entry_point
string node_manifest
```

Goal ↓

```
uint8 tree_result
```

↑ Result

Many behavior definitions — One powerful ROS 2 Node

```
ros2 run auto_apms_behavior_tree tree_executor
```

Now on ROS Index 🎉

AutoAPMS

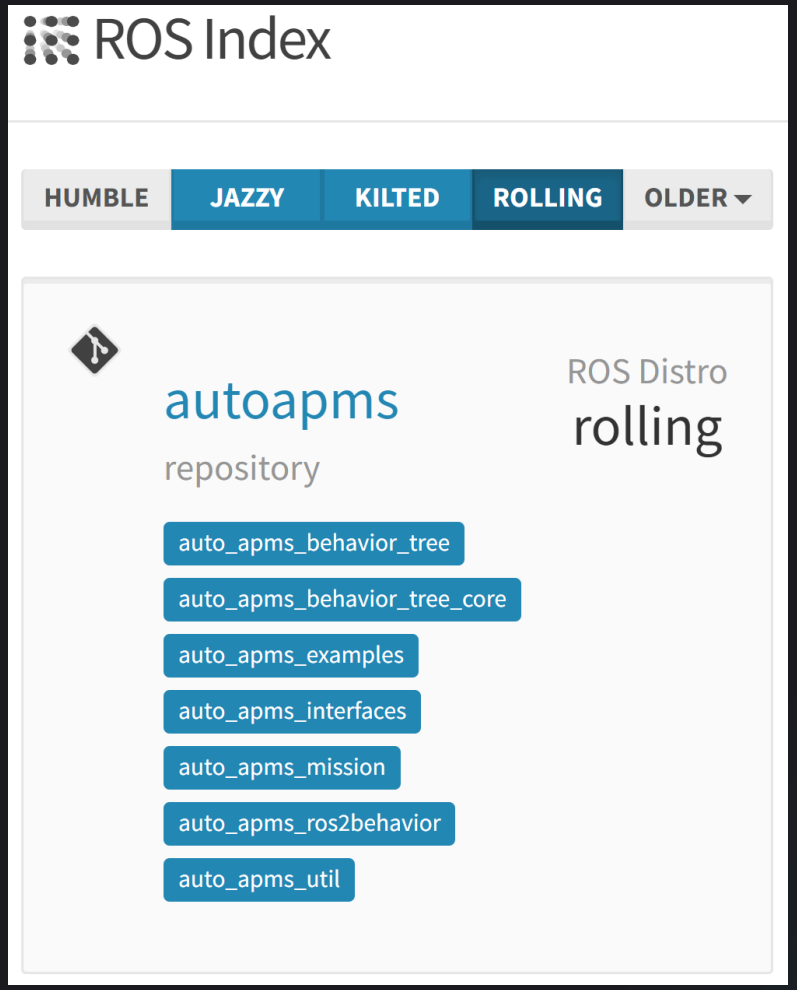
Core tools

```
sudo apt install ros-$ROS_DISTRO-auto-apms-behavior-tree
```

ROS 2 CLI integration

```
sudo apt install ros-$ROS_DISTRO-auto-apms-ros2behavior
```

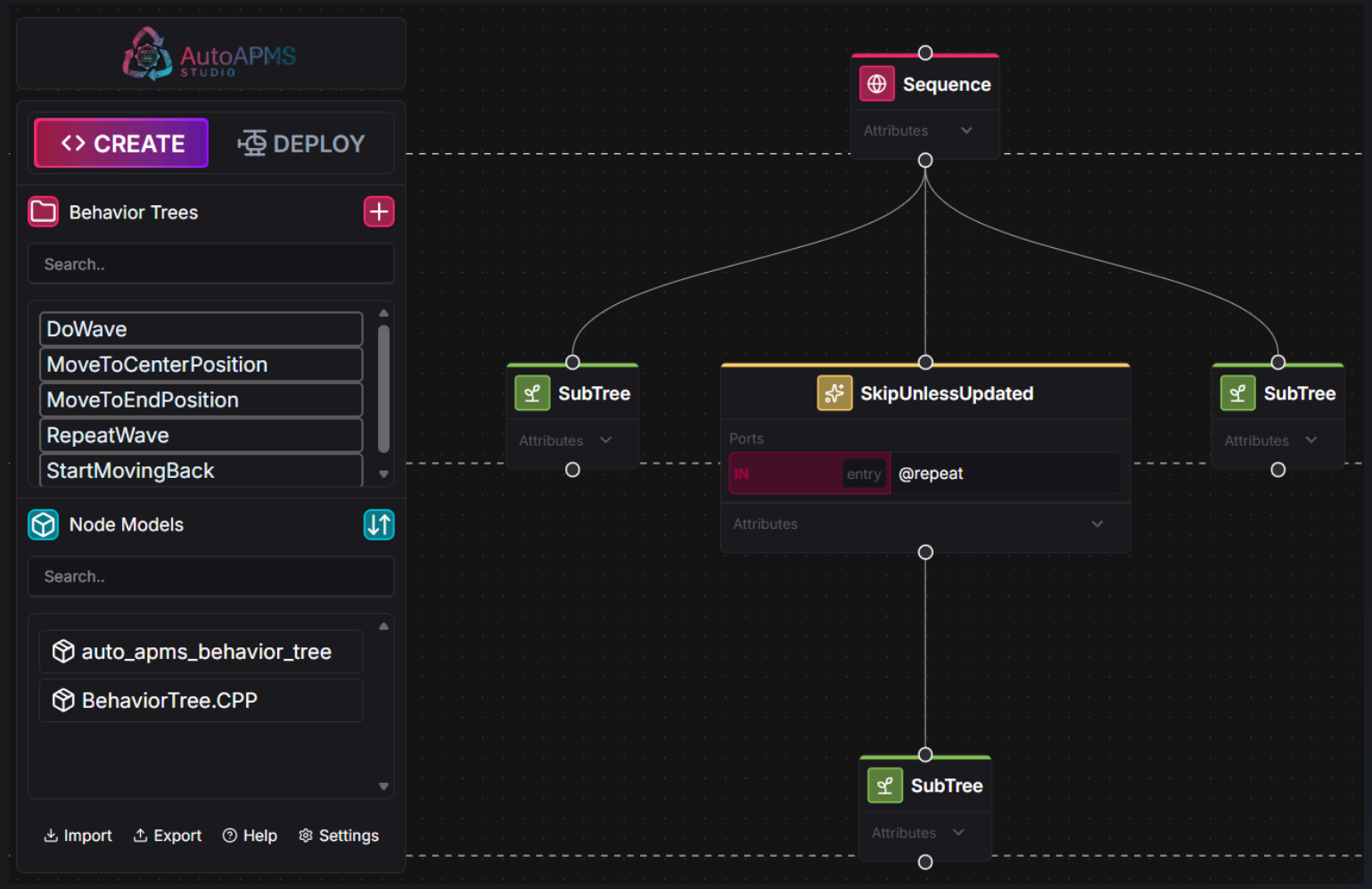
Give it a try and streamline your ROS 2 project with AutoAPMS!



Coming Soon

AutoAPMS

AutoAPMS STUDIO



Thank You

FOSDEM 2026 Robotics and Simulation

Full example code on GitHub
fosdem26-devtalk-autoapms

User Guide & API Docs
autoapms.github.io/auto-apms-guide

AI Generated Docs & Interactive Guide
deepwiki.com/AutoAPMS/auto-apms

This work has been funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center [LOEWE/1/12/519/03/05.001(0016)/72]

TECHNISCHE
UNIVERSITÄT
DARMSTADT

emergenCITY

LOEWE

Exzellente Forschung für
Hessens Zukunft