BT-Studio: a ROS Behavior Tree webIDE

Asociación de Robótica e Inteligencia Artificial JdeRobot

CIF G88145909

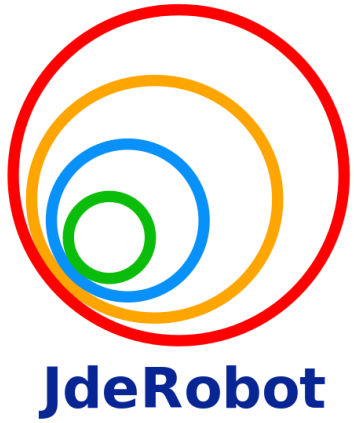
josemaria.plaza@gmail.com, oscar.robotics@tutanota.com,
javizqh@gmail.com

Contents

- Who we are
- Introduction
- BT-Studio tool
  - Features
  - Graphical User Interface
  - Edit: Visual BT editor
  - Edit: Action files
  - Run Monitored execution
  - Examples
- How is it done?
- Conclusions

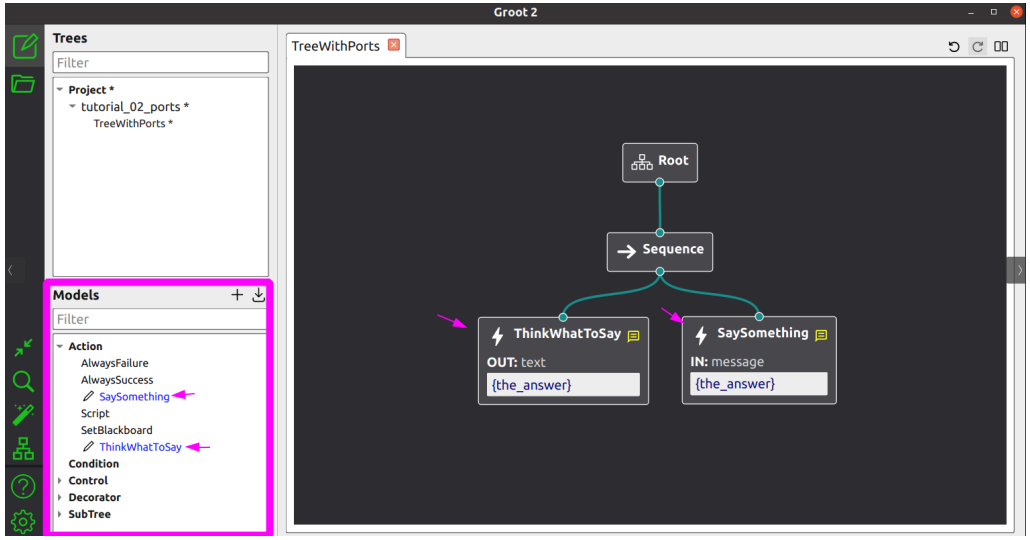
JdeRobot: who we are

- International open source robotics organization, 2014-
- `https://jderobot.github.com`
- Projects
  - Robotics education: RoboticsAcademy
  - Robot programming tools: Unibotics, BT-Studio...
  - AI driven robotics
- Activities: Google Summer of Code, internships...
- Community: 20+



Introduction

- Reactive approach does not scale up to complex robotics applications
- "Planned execution": Finite State Machines, Behavior-Trees (BT)...
- Simplify and speed up the development of BT robotics applications
- Maximize compatibility with state of the art technologies: `BehaviorTrees.CPP` (+groot2), `Py_Trees`



- Actions
- Sequence, Fallbacks
- Decorators

BT Studio tool

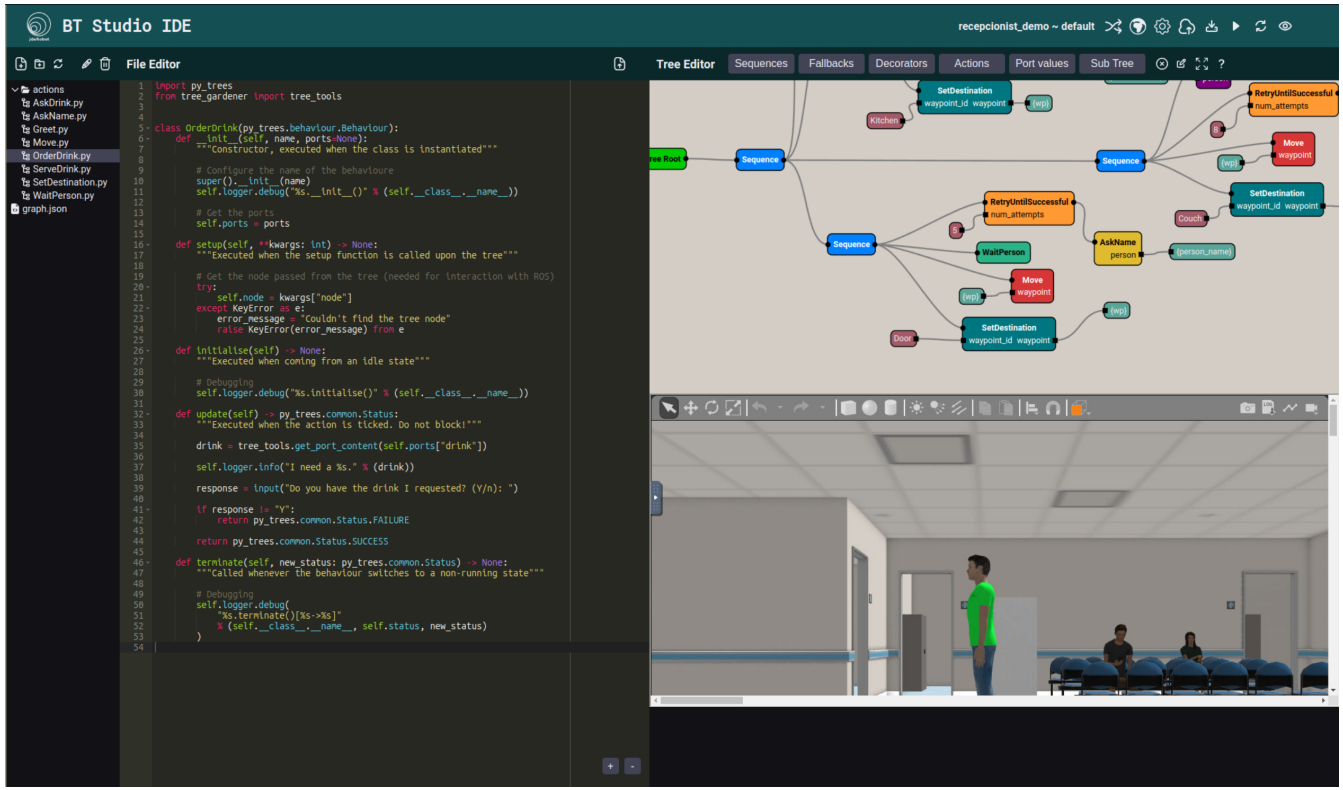
Web based IDE: edit, run, debug robotics applications from the browser

Features

- Crossplatform (Linux, Windows, MacOS)
- Python applications
- ROS2 Humble
- Simulated (Gazebo, Webots...) and real robots
- Open-source: `https://github.com/JdeRobot/bt-studio`
- Each user has a set of robotics projects, each project several files

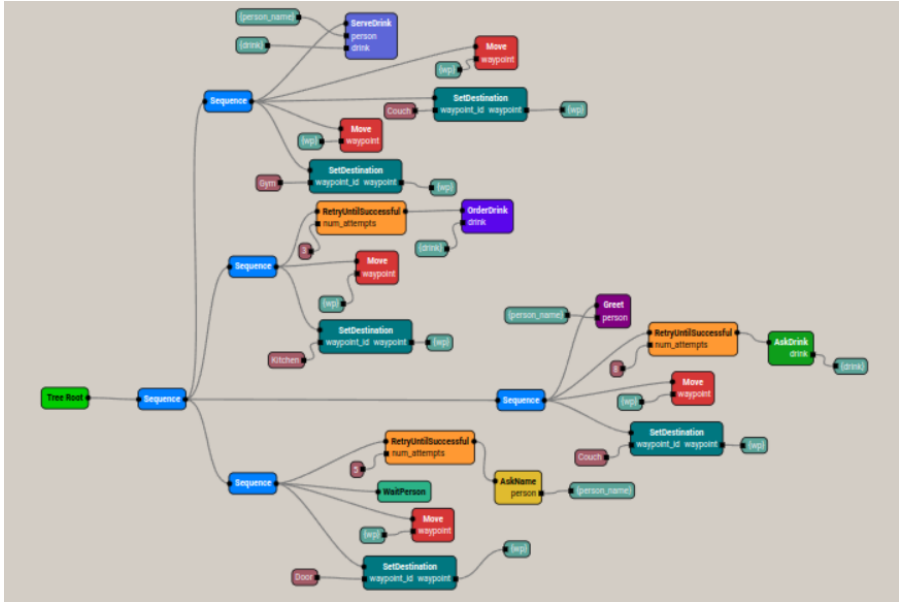
User Interface

- Files (Application [actions, trees], Universes...)
- *Text editor* for Python Actions, *Visual editor* for BT
- Execution monitoring: *VNC viewer*



Edit: Visual BehaviorTree editor

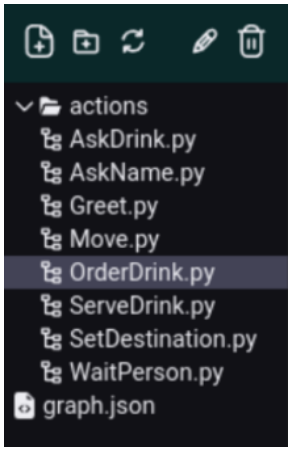
- Intuitive reactive REACT editor
- Customizable colors for each action
- Configurable order:
  bottom→top, top→bottom, ...
- Actions
- Sequence, Fallbacks
- Decorators



Edit: Action files

```python
import py_trees

class TemplateAction(py_trees.behaviour.Behaviour):

    def __init__(self, name, ports = None):

        """ Constructor, executed when the class is instantiated """

        # Configure the name of the behaviour
        super().__init__(name)
        self.logger.debug("%s.__init__()" % (self.__class__.__name__))

        # Get the ports
        self.ports = ports

        ...

    def setup(self, **kwargs: int) -> None:

        """ Executed when the setup function is called upon the tree """

        # Get the node passed from the tree (needed for interaction with ROS)
        try:
            self.node = kwargs['node']
        except KeyError as e:
            error_message = "Couldn't find the tree node"
            raise KeyError(error_message) from e

        ...

    def initialise(self) -> None:

        """ Executed when coming from an idle state """

        ...

    def update(self) -> py_trees.common.Status:

        """ Executed when the action is ticked. Do not block! """

        ....

        return new_status

    def terminate(self, new_status: py_trees.common.Status) -> None:

        """ Called whenever the behavior switches to a non-running state """
        ...
```

- Same structure as py_trees actions
- Setup
- Initialise
- Update
- Terminate

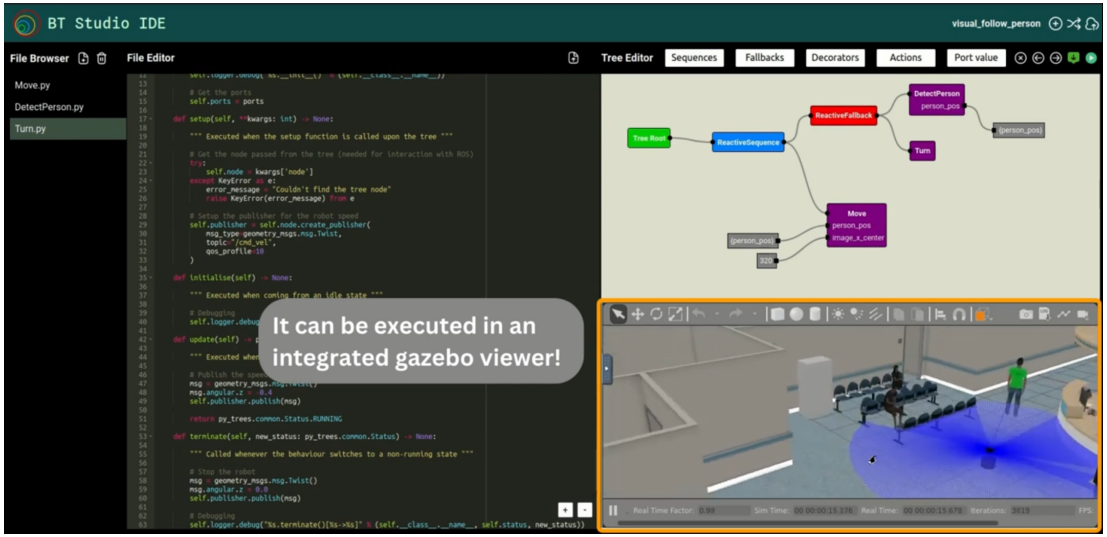


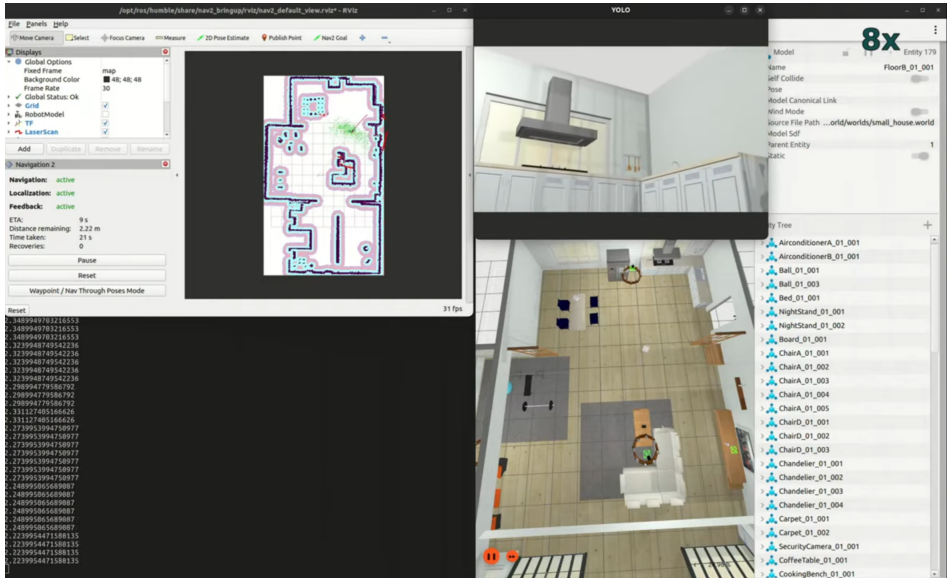
Run: Monitored execution

- (1) Dockerized execution (*Robotics Backend*)
  - All dependencies, assets, etc... are already pre-installed
- (2) Local execution creating a ROS2 package
  - ROS2 Humble is required installed locally
  - A test enviroment is provided with Webots simulator and a tree execution visualizer as thirdparty repos

- Control the flow of execution: Run, Pause and Restart
- Simple selection of Universes
  (simulated worlds, robot models, launchers...)

Examples



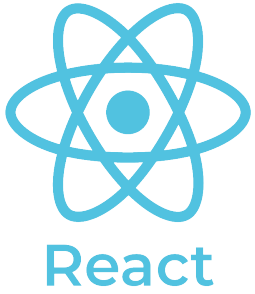


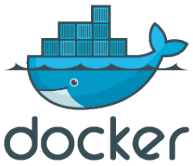
- Follow Person application

- RoboCup2022 recepcionist

How is it done?

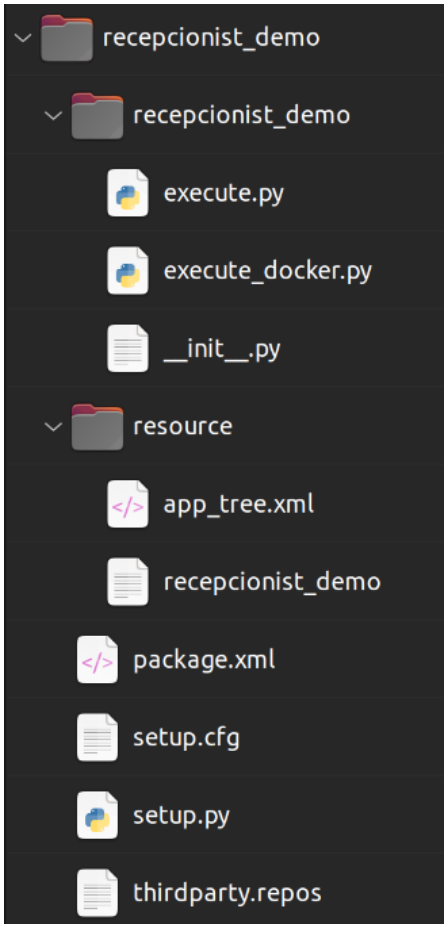
- Web tecnologies
  - backend: Django
  - frontend: React, HTML5, CSS
- Robotics tecnologies
  - ROS2
  - Py_trees
  - Gazebo, Webots simulators
- DevOps tecnologies
  - Docker
  - VNC





Internal files

- `app_tree.xml`: BT and source code
- `execute.py`:
  launcher for the application
- `execute_docker.py`:
  launcher for dockerized execution
- Auxiliary files as a basic ROS2 package



Translation process

- *From the user Python code for the Actions and the visual BT diagram to executable Python files*
- It is done in the backend
- Both are combined into a single XML file with 2 sections:
  - BehaviorTree section with the same structure as Groot2 BT
  - Code section is used instead of external files

Conclusions

- Context: Flowstate (Intrinsic), MoveIt Pro (Picknik), TheConstruct...
- Faster and simpler development of Behavior-Tree robotics applications
- Edit, run and debug BT robotics applications from the web browser

- Integration in Unibotics, our robot programming website
- Library of reusable subtrees (Google Summer of Code 2024)
- Library of universes