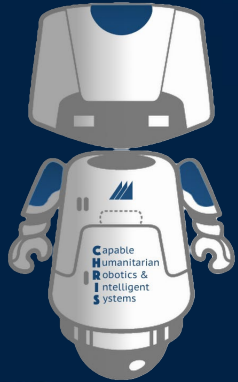


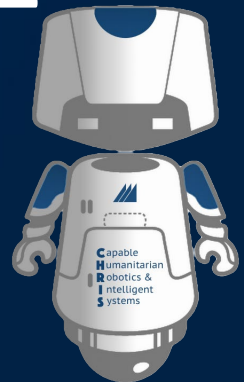


CHRISTOPHER NEWPORT UNIVERSITY
DEPARTMENT OF PHYSICS,
COMPUTER SCIENCE & ENGINEERING

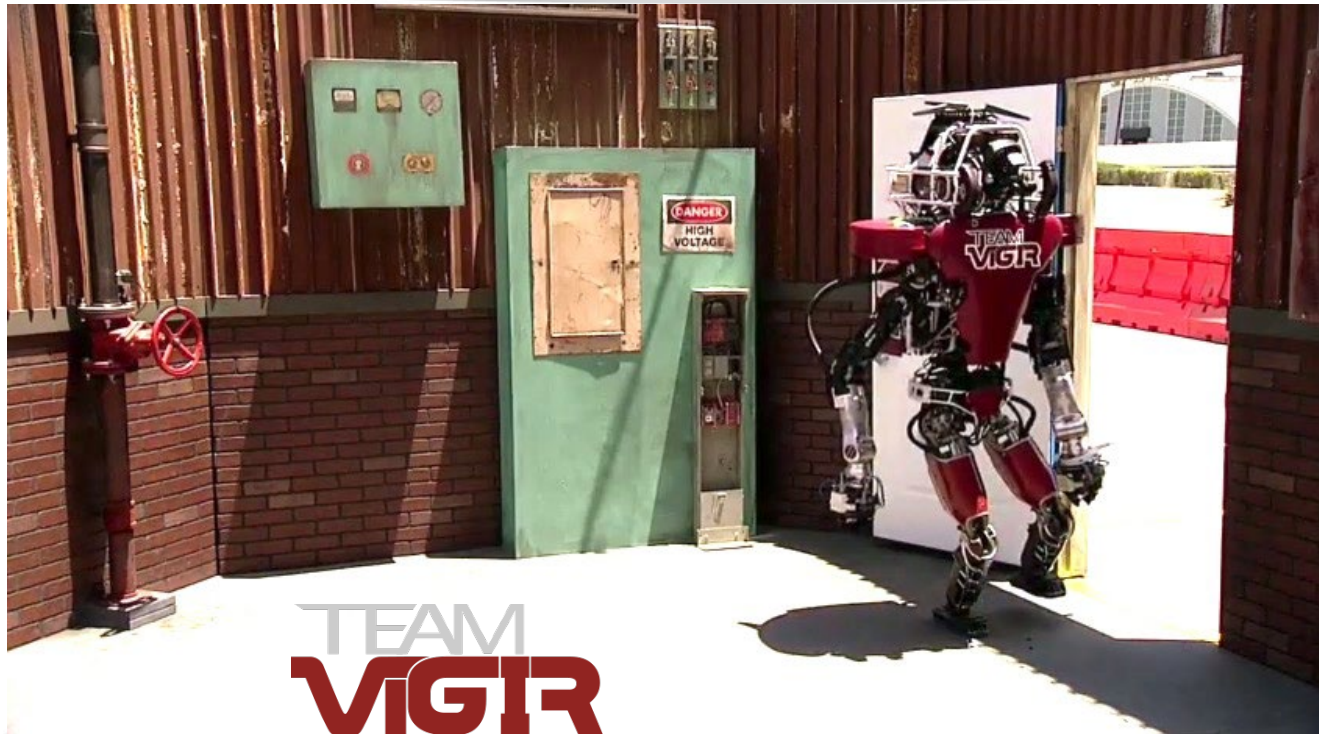
FlexBE -The Flexible Behavior Engine: Collaborative Autonomy for ROS 2



David C. Conner, PhD
Associate Professor
robotics@cnu.edu



Team ViGR's Atlas at the 2015 DARPA Robotics Challenge Finals



TEAM
VIGR

October 2023

CHRISTOPHER NEWPORT UNIVERSITY

© 2023 David C. Conner

3



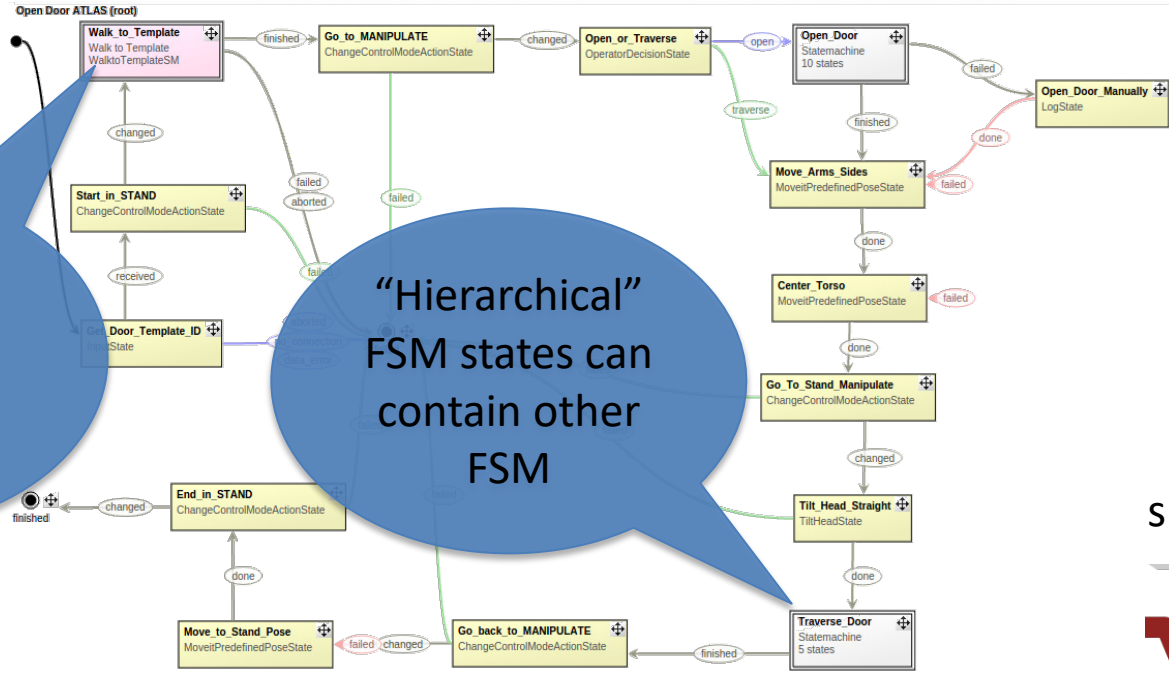
DARPA Robotics Challenge (DRC)

- Human-robot teams
- Supervised autonomy
 - Operators can inject information
 - Operators can preempt behaviors
- Constrained communications

TEAM
VIGIR



Hierarchical Finite State Machines



Compose behaviors within behaviors

“Hierarchical” FSM states can contain other FSM

2015 DRC Open Door Task High-level Behavior

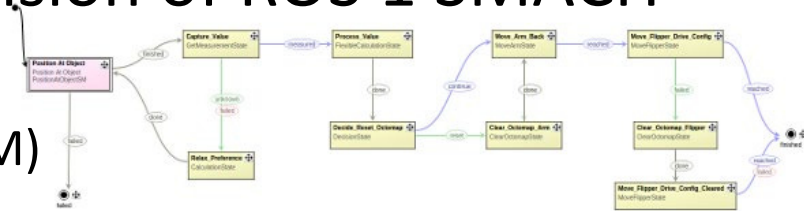
HFSM acts as a script for supervisors/robot



FlexBE : The Flexible Behavior Engine

- Originally conceived as an extension of ROS 1 SMACH

- <http://wiki.ros.org/smach>
- Hierarchical Finite State Machines (HFSM)
- Python-based state implementations



- Initially developed by Philipp Schillinger

@ TU Darmstadt (Germany)



- ROS 1 open-source release in Fall 2015

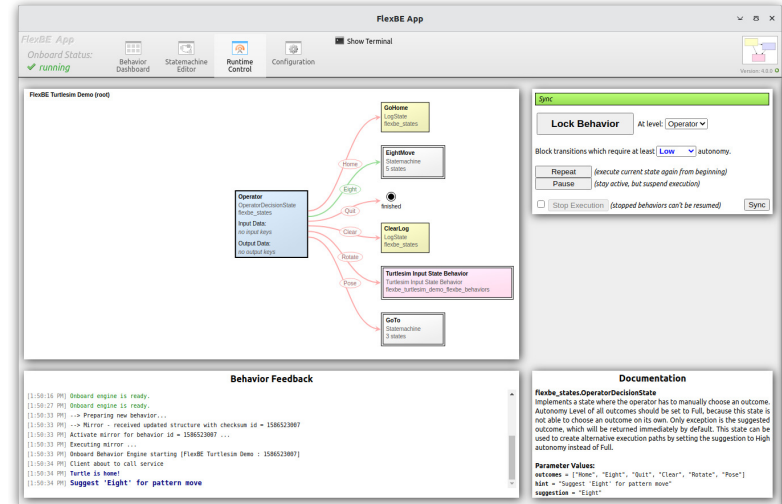
- <http://wiki.ros.org/flexbe>
- <http://github.com/FlexBE>

<http://philserver.bplaced.net/fbe/applications.php>



Key Design Concepts

- Support for high-level behavior control
 - Hierarchical Finite State Machines
 - Natural interaction with system capabilities
 - Concurrent state execution
 - Adjustable/sliding autonomy levels
 - Support for unsupervised fully autonomous mode
 - Runtime modifiable behaviors
- Intuitive GUI
 - State machine (behavior) editor
 - Interactive operator/supervisor runtime interface
 - Enable “Collaborative Autonomy”



<https://onlinelibrary.wiley.com/doi/full/10.1002/rob.21671>

Continuing Development @ CNU

- Christopher Newport University

Dept. of Physics, Computer Science and Engineering

- FlexBE used in “Introduction to Robotics”

- Recent releases

In collaboration with Philipp Schillinger

- Final ROS 1 Noetic release May 2023

- ROS 2 Conversion

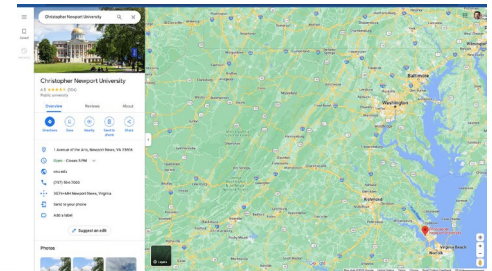
- Initial source released summer of '22

- Humble and Iron binaries summer '23



<https://cnu.edu>

4500 students in SE VA




Key Features and Design

Key Features and Design



FlexBE is

- Python-based
 - Easy state implementation development
- NOT for high-rate control
 - Desired update rate in tens of Hz
- NOT for real-time control
- NOT for verifiable safety critical systems



FlexBE can easily interact with such systems

The purpose of FlexBE is high-level behavioral control systems.



FlexBE GUI : Behavior Dashboard

FlexBE allows parameters to be configured at runtime

The screenshot shows the FlexBE GUI Behavior Dashboard for a ROS Turtlesim demo. The dashboard is divided into several sections:

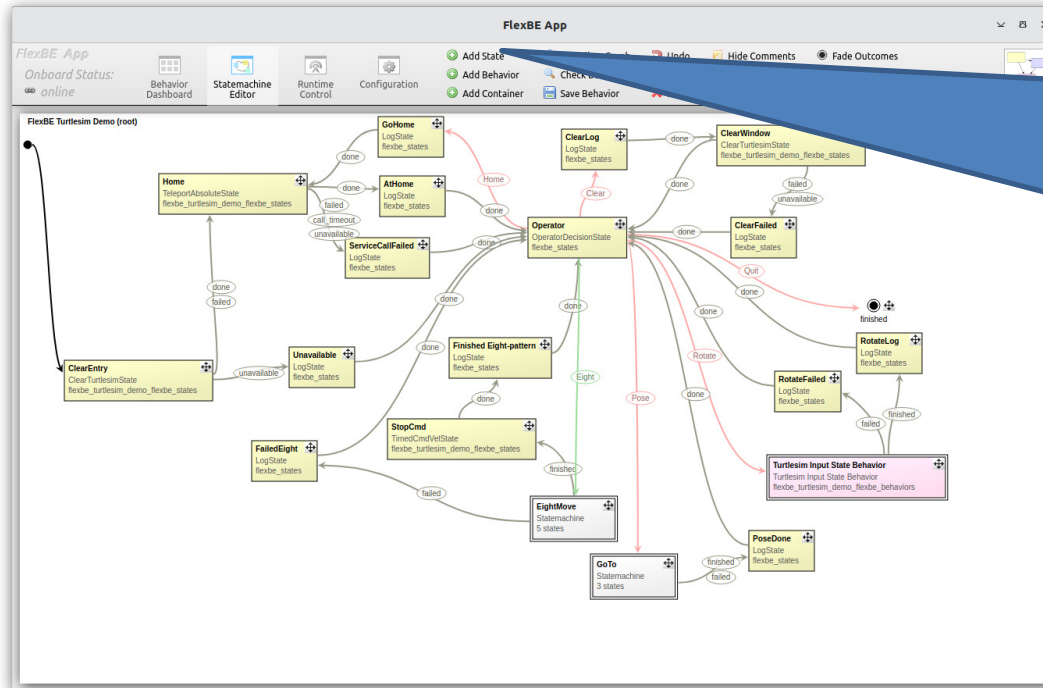
- Overview:** Package: flexbe_turtlesim_demo_flexbe_behaviors, Name: FlexBE Turtlesim Demo, Description: FlexBE demonstration using ROS Turtlesim from ROS tutorials.
- Private Configuration:** Variables enable easy configuration of constant internal values which are used multiple times. They are read-only and cannot be used in private functions. Example: cmd_vel = 'turtle1/cmd_vel'.
- State Machine Userdata:** The userdata of a state machine can be used to pass any data from one state to another. Userdata values may be changed by states during runtime. Example: home = [5.5, 5.5, 0.1].
- Behavior Parameters:** Parameters can be set by the operator when this behavior is started. Each parameter is identified by a unique variable name and displayed by using a label and providing usage advice. Example: Enum dropdown.
- Private Functions:** Defines which functions can be referenced by states that accept functions as parameters. Example: self.function_name.
- State Machine Interface:** Defines how the state machine of this behavior can be accessed when embedded in another behavior. Example: Outcomes: finished, Input Keys, Output Keys.

FlexBE allows modifiable “userdata” to be passed from state to state

<http://wiki.ros.org/flexbe/Tutorials> and https://github.com/FlexBE/flexbe_turtlesim_demo



FlexBE GUI : Statemachine Editor



Add New State

Name: Print_Greeting
Class: LogState

Class Selection

Filter: lo

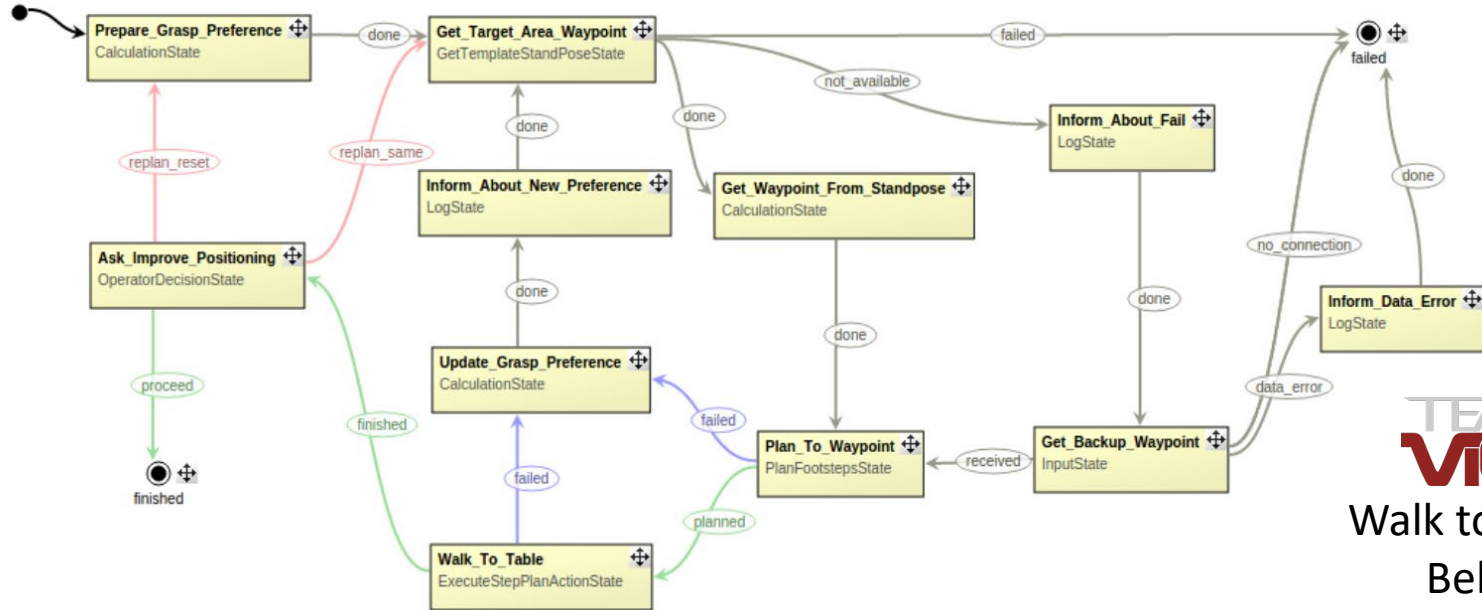
- LocomotionPosePerceptionState**
Extracts a pose of interest to walk to from environment data.
- LogState**
A state that can log a predefined message to precisely inform the operator about what happened to the behavior.
- StartRecordLogsState**
A state that records the contents of the specified ROS topics in a bag file.
- StopRecordLogsState**
A state that records the contents of the specified ROS topics in a bag file.

List of available Python state implementations that interact with system capabilities

<http://wiki.ros.org/flexbe/Tutorials> and https://github.com/FlexBE/flexbe_turtlesim_demo



FlexBE Behavior Example



TEAM
VIGIR

Walk to Template
Behavior

A state machine realizes a desired “behavior” by invoking system capabilities;
behavior state machines can be composed.



FlexBE GUI : Runtime Control

The screenshot displays the FlexBE GUI Runtime Control interface, which is divided into several panels:

- Configure Behavior Execution:** A panel for setting parameters and starting behavior. It includes a "Start Execution" button and a "Reset to Default" button. A message states: "The selected behavior supports no parameters."
- Behavior Feedback:** A log panel showing system messages such as "Onboard engine just started." and "Turtle is home!".
- Operator Decision State:** A central panel showing the current state and available transitions. The "Operator" is currently in the "OperatorDecisionState" with "no input keys" and "no output keys". Transitions include "Home", "Eight", "Quit", "Clear", "Rotate", "Pose", "GoHome", "EightMove", "ClearLog", and "GoTo".
- Lock Behavior:** A control panel with a "Lock Behavior" button and a dropdown menu set to "Operator". It includes options for "Repeat" and "Pause".
- Documentation:** A panel providing details about the "flexbe_states.OperatorDecisionState", including its purpose and parameter values.

Four blue callout bubbles highlight key features:

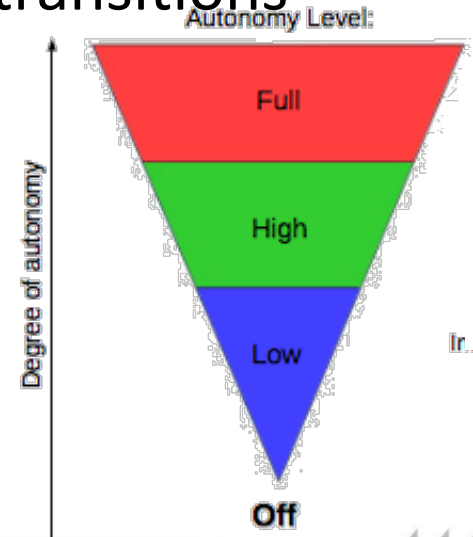
- Potential outcomes:** Points to the transition options in the Operator Decision State.
- Current active state:** Points to the "Operator" box in the Operator Decision State.
- Operator can select outcome to preempt state and force a transition:** Points to the transition options in the Operator Decision State.
- Autonomy level:** Points to the "Lock Behavior" dropdown menu.

<http://wiki.ros.org/flexbe/Tutorials> and https://github.com/FlexBE/flexbe_turtlesim_demo



Key Design Concepts

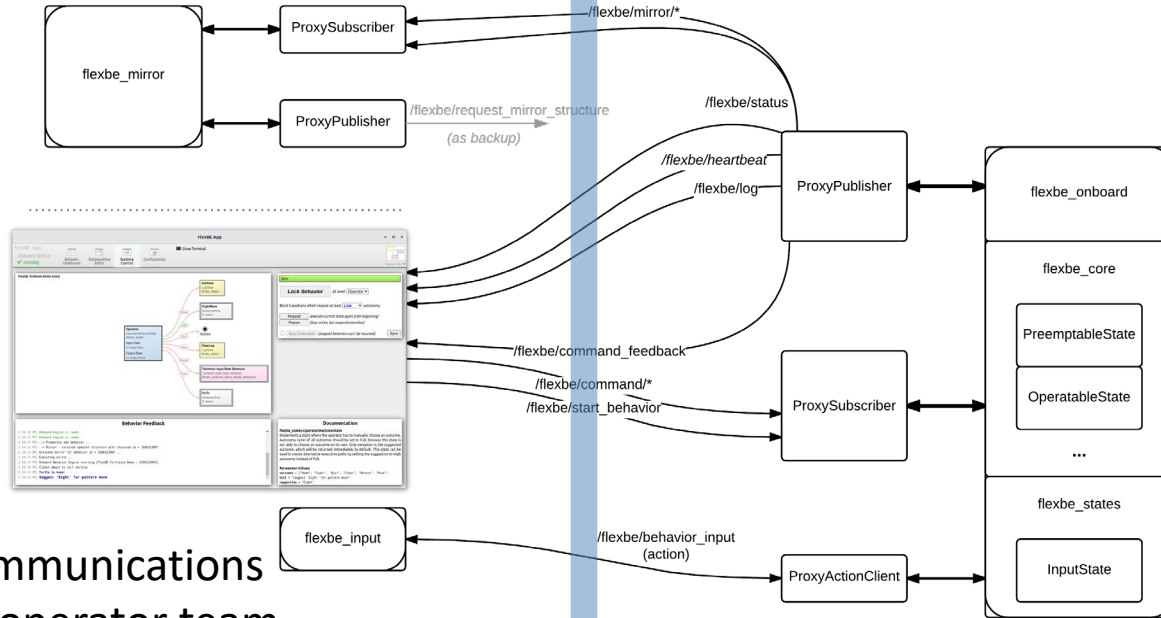
- Sliding autonomy levels
 - Low requires operator to confirm some transitions
 - i.e., it blocks exit transition
 - Full allows fully autonomous transitions
- Lockable states and edit on the fly
- Enable operator forced transitions



FlexBE Communications

Operator Control Station (OCS)

Onboard Robot Software



States share
pub/sub/action
interfaces via
“proxies”

DRC constrained communications
between robot and operator team



FlexBE States and ROS 2 Actions

- States interact with system capabilities
- Commonly implement an action client interface
 - Send goal **on_enter**
 - Monitor feedback and result in **execute**
 - Return outcome and **on_exit** transition on action **result**

For example, see code for topic-, service-, and action-based state implementations at

https://github.com/FlexBE/flexbe_turtlesim_demo/tree/ros2-devel/flexbe_turtlesim_demo/flexbe_states/flexbe_turtlesim_demo/flexbe_states





Significant Upgrades @ CNU

- ROS 2 conversion
 - Initial source released summer of '22
 - Refinements and cleanup in summer '23
 - Humble and Iron binaries summer '23
 - Enhancements to concurrent states in ros2-pre-release branch
 - Planned release to Iron coming soon

Summer '23 work supported under Naval Engineering Education Consortium (NEEC) grant N00174-23-1-0018



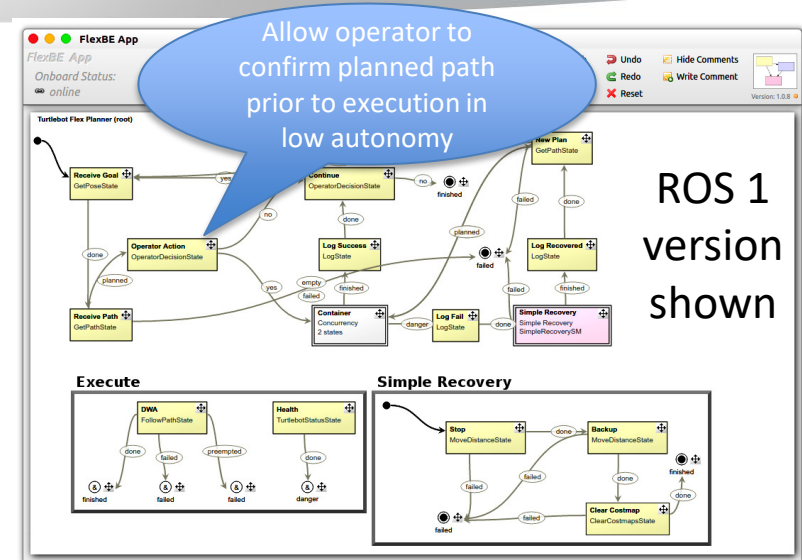
Related ROS Packages

Related ROS Packages



Flexible Navigation Package

- Collaborative navigation
 - Allow approval of plans or replan
 - Separate global and local planners
- FlexBE state implementations interface to Nav2 capabilities
 - State implementations
 - Special Nav2 compatible nodes



https://github.com/FlexBE/flexible_navigation
<https://ieeexplore.ieee.org/document/7925266>
<https://ieeexplore.ieee.org/document/9764047>
https://github.com/FlexBE/flex_nav_turtlebot3_demo

Flexible Behavior Trees

- Behavior trees are popular alternative to HFSM
- In search of the “Mythical HF SMBTH” – HFSM-BT Hybrid
 - from a 2017 Game Developers conference talk by Bobby Anguelov
https://www.youtube.com/watch?v=Qq_xX1JCrel&t=1159s
 - Combine each method’s strengths
 - BT: Reactive decisions, high-speed
 - HFSM: cyclical/repetitive behaviors , collaborative autonomy
- Flexible Behavior Trees : The “Mythical HF SMBTH” with FlexBE
 - The paper: <https://arxiv.org/abs/2203.05389>
 - The code: https://github.com/FlexBE/flexible_behavior_trees
 - The demo: https://github.com/FlexBE/flex_bt_turtlebot3_demo




Flexible Manipulation

- FlexBE interface to MoveIt!

<https://ieeexplore.ieee.org/document/8478933>

https://github.com/CNURobotics/flexible_manipulation

- Currently only ROS 1 (Kinetic) and Python 2
- Planning for ROS 2 conversion late 2024
 - MoveIt! 2 stabilizing 
 - Stable ROS 2 physics-based simulations of robot arms



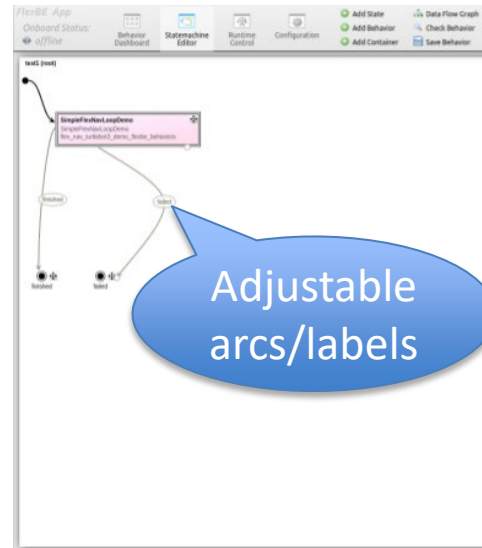
Ongoing Research and Development

Ongoing Research and Development



System Improvements

- Testing, demonstrations, and tutorials for packages
- FlexBE WebUI
 - Improved graphics
 - Simplified Python comms integration with UI
 - Expect alpha pre-release January 2024
 - Planned UI Advancements
 - Improved operator control over concurrent states
 - HFSM Synthesis and debugging tools



Supported under Naval Engineering Education Consortium (NEEC) grant N00174-23-1-0018

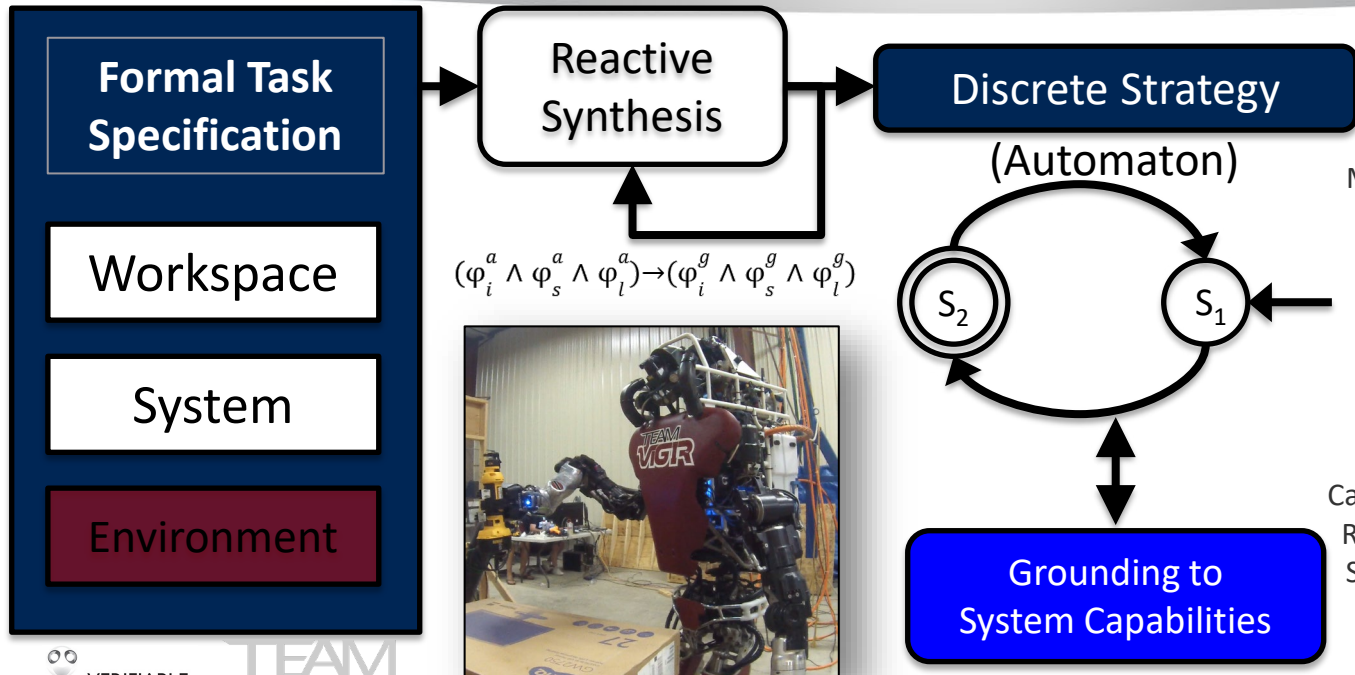


HFSM Synthesis in FlexBE

- Designing HFSM (or BT) is hard
 - Requires significant testing and validation
- Goal: “Correct-by-construction” synthesis tools
 - Specialized research in formal methods community
 - Less accessible to general robotics community



Prior Work with ROS 1 FlexBE and Reactive GR1 Synthesis



Maniatopoulos *et al.*, "Reactive high-level behavior synthesis for an Atlas humanoid robot," *ICRA 2016*, <https://ieeexplore.ieee.org/document/7487613>

Hayhurst and Conner, "Towards Capability-Based Synthesis of Executable Robot Behaviors," *SoutheastCon 2018*, St. Petersburg, FL, USA, 2018, pp. 1-8, <https://ieeexplore.ieee.org/document/8479047>

(e.g. FlexBE state implementations)



GR1 Synthesis Example w/ Slugs



“If in next step the goat and wolf are on the left bank, then on the next step the farmer better be on left bank”
 $X(!goat \ \& \ !wolf) \rightarrow X!farmer$

https://link.springer.com/chapter/10.1007/978-3-319-41540-6_18

Assumptions \rightarrow Guarantees

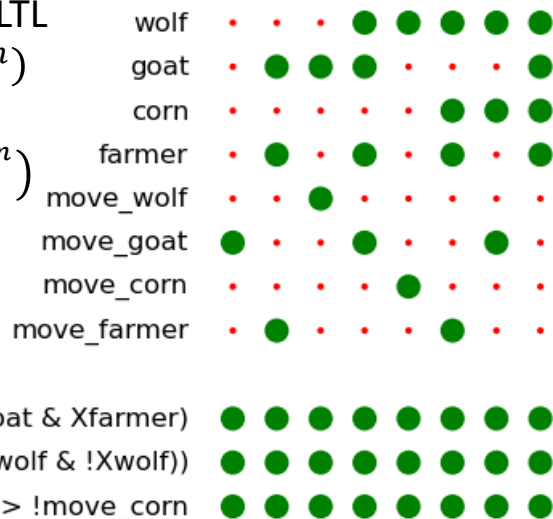
$$(\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a) \rightarrow (\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g)$$

time \rightarrow

GR1 fragment of LTL
 complexity $O(2^n)$

vs.

Full LTL w/ $O(2^{2^n})$





Ongoing Synthesis Work in FlexBE

- Converting 2018 system into ROS 2 version
- Develop several tutorials and demonstrations
 - Make synthesis more accessible to general community



The paper: coming Spring '24

The code: coming Dec '23

The demo: coming Dec '23

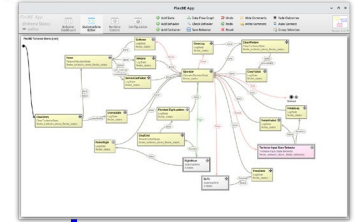
- Refactor and redesign to simplify usage
- Integrate automatic discovery of system capabilities

New work supported under Naval Engineering Education Consortium (NEEC) grant N00174-23-1-0018



Conclusion

- FlexBE is now available for ROS 2
- Quick start demo at https://github.com/FlexBE/flexbe_turtlesim_demo
- Available extension packages
- Development is active and ongoing
- Active research integrating HFSM synthesis



For more information

robotics@cnu.edu





FlexBE State Implementations

- Each state corresponds to a Python implementation of the **EventState** class

https://github.com/FlexBE/flexbe_behavior_engine/blob/ros2-devel/flexbe_core/flexbe_core/core/event_state.py

- **on_start** – invoked when behavior initialized
- **on_enter** – invoked when state becomes active
- **execute** – invoked at (approximately) specified rate
- **on_exit** – invoked when state returns outcome
- **on_stop** – invoked when behavior is shutdown
- **on_pause/resume** – invoked when state is locked/unlocked



FlexBE in Education

- Currently use in CPSC 472/572
“Introduction to Robotics” @CNU
- FlexBE in low-autonomy acts as “script”
 - “Get goal”, “Plan path”, “Execute path follower”
 - Allows users to better see interaction of components
- Teach HFSM-based behavior control
 - Students can write Python-based state implementations
 - Use FlexBE to control high level system behaviors
 - Reinforce use of object-oriented paradigm



WGCF Specs (Slugs format)

[INPUT]

our farmer prefers 4 letter words not cabbages

goat

wolf

corn

farmer

[ENV_INIT]

Everyone on the left bank

!goat

!wolf

!corn

!farmer

[OUTPUT]

move_goat

move_wolf

move_corn

move_empty

[SYS_INIT]

The game solver should figure out that one thing needs
to be moved initially

[ENV_TRANS]

What occurs in environment

transitions due to move

(!goat & move_goat & !farmer) -> (goat' & farmer')

(goat & move_goat & farmer) -> (!goat' & !farmer')

$$(\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a) \rightarrow (\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g)$$

(wolf & move_wolf & farmer) -> (!wolf' & !farmer')

(!wolf & move_wolf & !farmer) -> (wolf' & farmer')

(corn & move_corn & farmer) -> (!corn' & !farmer')

(!corn & move_corn & !farmer) -> (corn' & farmer')

(move_empty & farmer) -> !farmer'

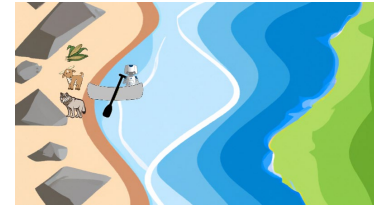
(move_empty & !farmer) -> farmer'

Not moving leaves the environment alone

!move_goat -> ((goat & goat') | (!goat & !goat'))

!move_wolf -> ((wolf & wolf') | (!wolf & !wolf'))

!move_corn -> ((corn & corn') | (!corn & !corn'))



WGCF Spec (continued)

[SYS_TRANS]

Allowable commands from our controller

Can only move one thing at a time

!(move_goat & move_wolf)

!(move_goat & move_corn)

!(move_goat & move_empty)

!(move_corn & move_wolf)

!(move_corn & move_empty)

!(move_wolf & move_empty)

What we need our controller to enforce

Farmer must stay when goat and corn are together

(goat' & wolf') -> farmer'

(!goat' & !wolf') -> !farmer'

(goat' & corn') -> farmer'

(!goat' & !corn') -> !farmer'

Cannot move unless boat on same side

(goat & !farmer) -> !move_goat

(!goat & farmer) -> !move_goat

(wolf & !farmer) -> !move_wolf

(!wolf & farmer) -> !move_wolf

(corn & !farmer) -> !move_corn

(!corn & farmer) -> !move_corn

[ENV_LIVENESS]

Nothing

[SYS_LIVENESS]

Let's get across infinitely often

goat & wolf & corn

$$(\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a) \rightarrow (\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g)$$

