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

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

• History and background
• Key features and design
• Related ROS packages
• Ongoing research and development
Team ViGIR’s Atlas at the 2015 DARPA Robotics Challenge Finals
DARPA Robotics Challenge (DRC)

- Human-robot teams
- Supervised autonomy
  - Operators can inject information
  - Operators can preempt behaviors
- Constrained communications
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
  – 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://github.com/FlexBE](http://github.com/FlexBE)
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”

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

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

FlexBE can easily interact with such systems.
FlexBE allows modifiable “userdata” to be passed from state to state

FlexBE allows parameters to be configured at runtime

FlexBE GUI: Statemachine Editor

List of available Python state implementations that interact with system capabilities

A state machine realizes a desired “behavior” by invoking system capabilities; behavior state machines can be composed.
FlexBE GUI: Runtime Control

- Potential outcomes
- Autonomy level
- Current active state
- Operator can select outcome to preempt state and force a transition

References:
- http://wiki.ros.org/flexbe/Tutorials
- 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

DRC constrained communications between robot and operator team

States share pub/sub/action interfaces via “proxies”
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
  - State implementations
  - Special Nav2 compatible nodes

ROS 1 version shown

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 HFSMBTH” – HFSM-BT Hybrid
  – from a 2017 Game Developers conference talk by Bobby Anguelov
    [https://www.youtube.com/watch?v=Qq_xX1JCreI&t=1159s](https://www.youtube.com/watch?v=Qq_xX1JCreI&t=1159s)
  – Combine each method’s strengths
    - BT: Reactive decisions, high-speed
    - HFSM: cyclical/repetitive behaviors, collaborative autonomy
- Flexible Behavior Trees: The “Mythical HFSMBTH” with FlexBE
  - The code: [https://github.com/FlexBE/flexible_behavior_trees](https://github.com/FlexBE/flexible_behavior_trees)
  - The demo: [https://github.com/FlexBE/flex_bt_turtlebot3_demo](https://github.com/FlexBE/flex_bt_turtlebot3_demo)
Flexible Manipulation

• FlexBE interface to MoveIt!
  
  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

© 2023 David C. Conner

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

Formal Task Specification

Workspace

System

Environment

Reactive Synthesis

Discrete Strategy

(Automaton)

Grounding to System Capabilities

(e.g. FlexBE state implementations)

Maniatopoulos et al., "Reactive high-level behavior synthesis for an Atlas humanoid robot," *ICRA 2016*,

“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) -> X!farmer

https://link.springer.com/chapter/10.1007/978-3-319-41540-6_18
Manually created state machine from SLUGS synthesis
Synthesized statemachines placed in statemachine container
Realization of SLUGS synthesized automata from GR1 specs
Parameters defined by Mealy machine outputs

Run time execution monitor

Confirm transition in low autonomy
Operator preempt
Switch to Full Autonomy
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
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

  - `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

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

[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

# Not moving leaves the environment alone
!move_goat -> ((goat & goat') | (!goat & !goat'))
!move_wolf -> ((wolf & wolf') | (!wolf & !wolf'))
!move_corn -> ((corn & corn') | (!corn & !corn'))
### [SYS_TRANS]

# Allowable commands from our controller

# Can only move one thing at a time

\(|\text{move\_goat} \& \text{move\_wolf}|\)

\(|\text{move\_goat} \& \text{move\_corn}|\)

\(|\text{move\_goat} \& \text{move\_empty}|\)

\(|\text{move\_corn} \& \text{move\_wolf}|\)

\(|\text{move\_corn} \& \text{move\_empty}|\)

\(|\text{move\_wolf} \& \text{move\_empty}|\)

# What we need our controller to enforce

# Farmer must stay when goat and corn are together

\((\text{goat}' \& \text{wolf}') \rightarrow \text{farmer}'\)

\((!\text{goat}' \& !\text{wolf}') \rightarrow !\text{farmer}'\)

# Can only move one thing at a time

\((\text{goat}' \& \text{corn}') \rightarrow \text{farmer}'\)

\((!\text{goat}' \& !\text{corn}') \rightarrow !\text{farmer}'\)

# Farmer must stay when goat and corn are together

# Cannot move unless boat on same side

\((\text{goat} \& !\text{farmer}) \rightarrow !\text{move\_goat}\)

\((!\text{goat} \& \text{farmer}) \rightarrow !\text{move\_goat}\)

\((\text{wolf} \& !\text{farmer}) \rightarrow !\text{move\_wolf}\)

\((!\text{wolf} \& \text{farmer}) \rightarrow !\text{move\_wolf}\)

\((\text{corn} \& !\text{farmer}) \rightarrow !\text{move\_corn}\)

\((!\text{corn} \& \text{farmer}) \rightarrow !\text{move\_corn}\)

### [ENV_LIVENESS]

# Nothing

### [SYS_LIVENESS]

# Let's get across infinitely often

\(\text{goat} \& \text{wolf} \& \text{corn}\)