64-424
Intelligent Robotics

https://tams.informatik.uni-hamburg.de/lectures/2018ws/vorlesung/ir

Marc Bestmann / Michael Görner / Jianwei Zhang

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Technical Aspects of Multimodal Systems

Winterterm 2018/2019

Outline

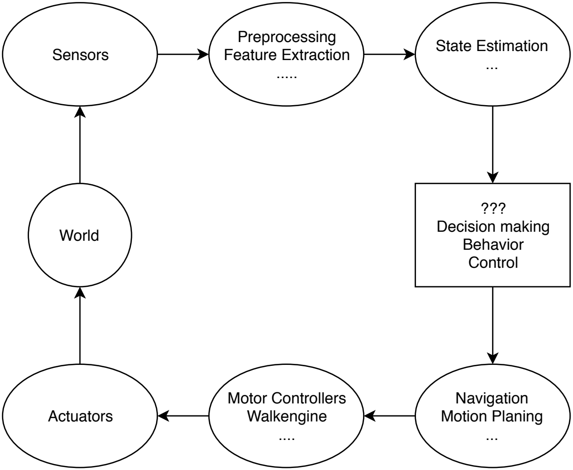
Outline

1. Control Architectures
   - Finite State Machines (FSM)
   - Hierarchical State Machines (HSM)
   - Subsumption Architecture
   - Decision Trees (DT)
   - Behavior Trees
   - Dynamic Stack Decider (DSD)
   - Summary

Motivation

- Robot needs to achieve a goal
- Sensors and filters give estimation of the world
- We need to decide which actions to take
- We have to react on changes of the world
- In trivial cases this can be programmed directly
- In general an architecture is needed

General Principle



Agenda for Today

- Talk about design principals
- Look at the most used classical architectures
  - Finite State Machine (FSM)
  - Hierarchical State Machine (HSM)
  - Subsumption
  - Decision Tree (DT)
- Talk about more recent architectures
  - Behavior Trees (BT)
  - Dynamic Stack Decider (DSD)
- Compare advantages and disadvantages

Design Principles of Control Architectures (CA)

- Hierarchical organization
  - Some subtask may be more important than others
  - Ex: recharging when empty > navigating to goal
- Reusable code
  - The same subtask is maybe needed multiple times
  - Ex: turning sensors to specific location
- Modular design
  - Splitting a task into subtasks makes development easier
  - Ex: divide "grasp" into "open hand", "position hand", "close hand"
- Maintainability
  - Changes to the behavior has to possible without general restructuring
  - Ex: adding "lift hand" to "grasp" should only require changes in this part

Design Principles of Control Architectures (CA) (cont.)

- Human readable
  - Structure has to be readable for developing and debugging
  - Ex: GUI with graph structure and current state
- Stateful
  - The current state of the system should be clear
  - Ex: clear if ball is currently in hand or not
- Fast
  - Low latency between sensor input and action
  - Ex: Bumper is hit -> immediate stop of wheels to prevent damage
- Expressive / scalable
  - CA must be able to encode a large variety of tasks
  - Ex: a soccer player with different strategies

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017
Poppinga, Martin and Bestmann, Marc. "ASDS - Active Self Deciding Stack", 2018

Design Principles of Control Architectures (CA) (cont.)

- Suitable for automatic synthesis
  - Synthesis, e.g. by machine learning, for action ordering
  - Ex: using NNs in some parts to decide which action is to be taken
- Understandability of the concept
  - It should not take to much time to understand the concept
  - Ex: FSM is very simple, BT is complex
- Implementation effort
  - Effort to implement the used concept
  - Not important if fitting library is available

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017
Poppinga, Martin and Bestmann, Marc. "ASDS - Active Self Deciding Stack", 2018

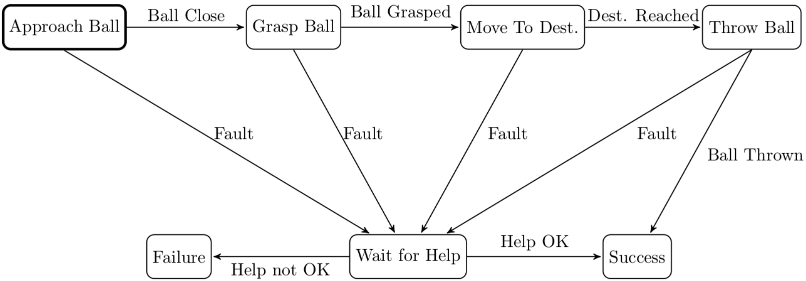
Design Principles

- A lot of things to keep in mind
- Sometimes contradictory
  - Ex: fast <-> expressive
  - EX: maintainability <-> understandability
- Highly depended on the domain and goal
- Spoiler alert: there is no silver bullet

Finite State Machine

- Very common in computer science
- Often implicitly implemented
  - State is encoded in multiple variables of flags
- Good theoretical foundation
- Working principal
  - List of possible states
  - Transitions between those states
  - Start state
  - Check for transition conditions
  - Change state if condition is true
  - Act according to current state

FSM - Example



Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

FSM - Pseudo Code - Transition in States

```
class AbstractState:
    def run(self, blackboard):
        raise NotImplementedError
    def next(self, blackboard):
        raise NotImplementedError
class ApproachBall(AbstractState):
    def run(self, blackboard):
        # send some walking commands
    def next(self, blackboard):
        if blackboard.ball_distance < 1:
            return GraspBall()
        if blackboard.fault:
            return WaitForHelp()
        return self
class GraspBall(AbstractState): ...
class BallStateMachine:
    def __init__(self, blackboard):
        # the blackboard is some kind of object holding all information
        self.blackboard = blackboard
        self.current_state = ApproachBall()
        while true:
            self.run()
            sleep(0.1)
    def run(self):
        self.current_state.run()
        self.current_state = self.current_state.next()
```

FSM - Pseudo Code - Transition in Machine

```
class ApproachBall(AbstractState):
    def run(self, blackboard):
        # send some walking commands
class GraspBall(AbstractState):
    ...
class BallStateMachine:
    def __init__(self, blackboard):
        # the blackboard is some kind of object holding all information
        self.blackboard = blackboard
        fsm = StateMachine(inital=ApproachBall, states=[ApproachBall, GraspBall, ...])
        fsm.add_transition(from=ApproachBall, to=GraspBall, if=ball_close)
        fsm.add_transition ...
        while true:
            self.run()
            sleep(0.1)

    def ball_close(self):
        return blackboard.ball_distance < 1

    def run(self):
        self.fsm.get_current_state().run()
        self.fsm.check_transition()
```

FSM - Defining Transitions

- Transitions can be defined by the state (version 1)
- Or in the statemachine (version 2)
- This has pros and cons
- Pro in state
  - Decision can depend on "state of the current state"
    - Ex: "Wait5Sec" remembers time when state started
  - Simpler to implement
  - Easier to see to which state you go from one state
- Con in state
  - Danger of putting too much "state into a state", leading to an implicit HSM
  - Transitions are all distributed across states
  - More difficult to use if you have events
- Both versions can be found in libraries

FSM - Advantages and Disadvantages

Advantages:

- Commonly used in computer science
- Intuitive structure
- Ease of implementation

Disadvantages:

- Maintainability
- Scalability ("state explosion")
- Reusability
- No standardization

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

FSM - Conclusion

- Simple to understand and implement
- Very wide spread
- Use for small/trivial scenarios
- Stateful

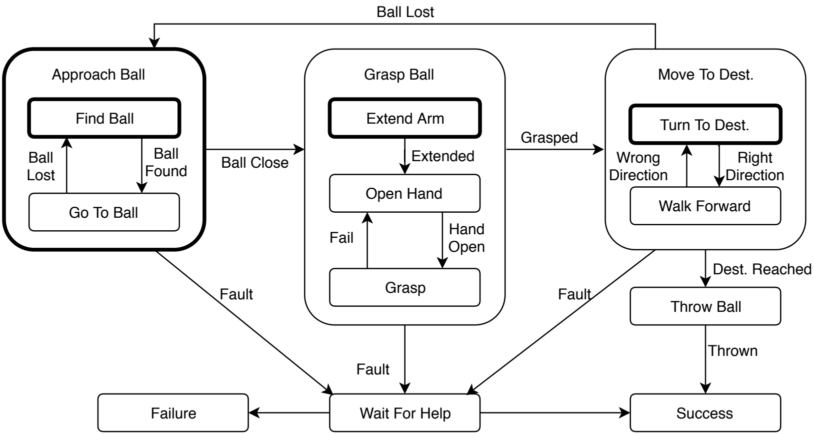
Libraries

- To many to list
- I recommend picking one which gives you graphical output for better debugging

Hierarchical State Machines

- Also known as State Charts (UML)
- Solve some of the problems of FSMs
- Introducing a hierarchical layout
- Each state can consist of substates
- States with substates are called superstates
- Generalized transitions connect superstates
- Each superstate has a start substate
- The number of overall transitions is reduced

HSM - Example


Ball Lost
Approach Ball
Find Ball
Ball Lost
Ball Found
Go To Ball
Ball Close
Grasp Ball
Extend Arm
Extended
Open Hand
Fail
Hand Open
Grasp
Grasped
Move To Dest.
Turn To Dest.
Wrong Direction
Right Direction
Walk Forward
Dest. Reached
Throw Ball
Thrown
Fault
Fault
Fault
Failure
Wait For Help
Success

HSM - Advantages and Disadvantages

Advantages:

- Modularity
- Behavior inheritance

Disadvantages:

- Maintainability
- Non intuitive hierarchy

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

HSM - Conclusion

- Still relative easy to implement
- Stateful
- Still comparably wide spread
- Useful in medium complex scenarios

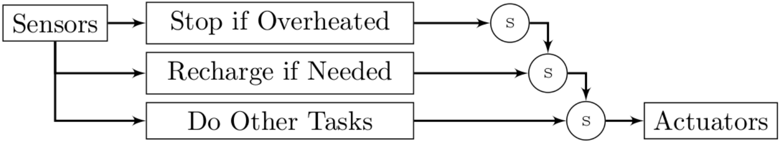
Libraries

- smach (ROS)
- pysm (Python)

Subsumption Architecture

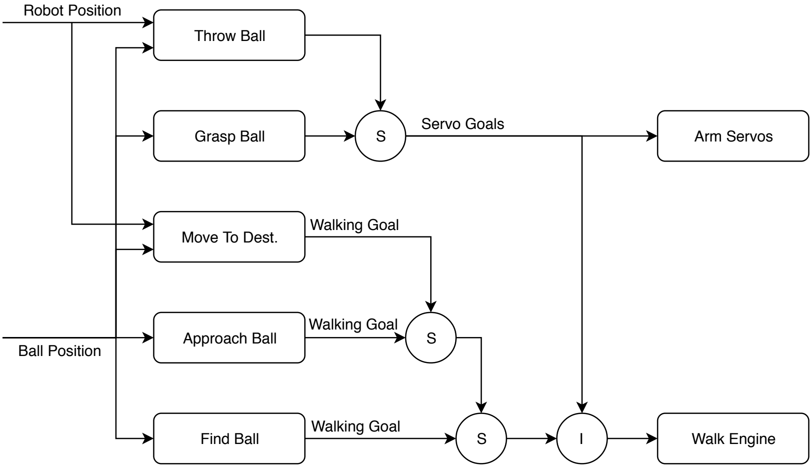
- Several modules
- Each implements one task
- All run in parallel
- Module are ordered by priority

Subsumption - Example



Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

Subsumption - Robot Example



Subsumption - Advantages and Disadvantages

Advantages:

- Modularity
- Hierarchy
- Reactivity

Disadvantages:

- Scalability
- Maintainability
- Not stateful

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

Subsumption - Conclusion

- Good for reactive systems
- Hard to handle time dimension
- Usable for small to medium complex systems
- Not widely used

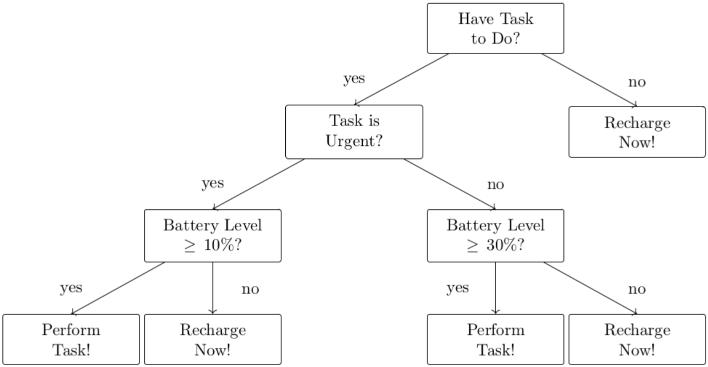
Libraries

- subsuMeLib (C++) -outdated-

Decision Trees

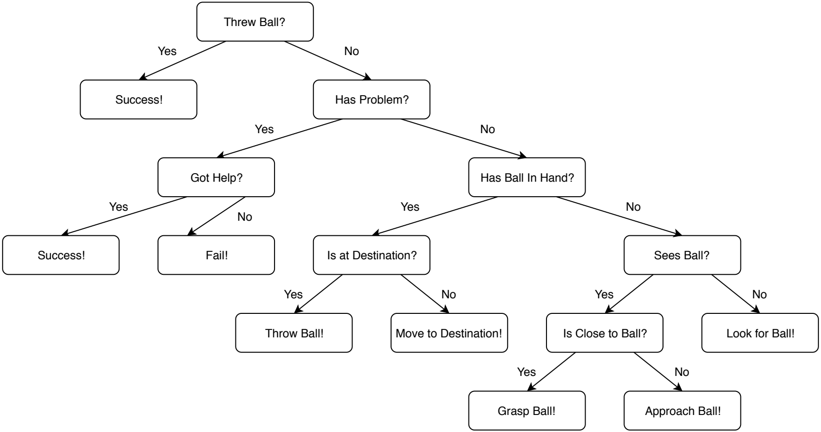
- Representation of nested if-then clauses
- Tree structure
- Internal nodes are predicates
- Leaf nodes are actions

DT - Example



Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

DT - Robot Example



DT - Advantages and Disadvantages

Advantages:

- Modularity
- Hierarchy
- Intuitive structure
- Clear division between actions and decisions

Disadvantages:

- Repetitions
- Maintainability
- Not stateful

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

DT - Conclusion

- Trivial to implement
- Using a framework rather than if-else can help with larger trees
- Widely (implicitly) used in computer science
- Easy to use with machine learning
- Good for domains which have no time dimension

Libraries

- scikit-learn + dtreeviz (Python)

Behavior Trees

- Tree of nodes
- Internal nodes are control flow nodes
  - Sequence
  - Fallback
  - Parallel
  - Memory
- Leaf nodes are execution nodes
  - Action
  - Condition
- Root node sends out *ticks* in fixed frequency to its children
- Only nodes that receive a tick are executed
- Children return *Running*, *Success* or *Failure*

BT - Sequence Node

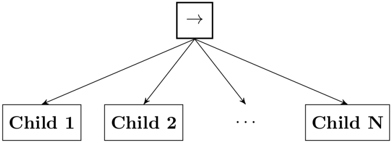


Fig. 1.2: Graphical representation of a Sequence node with N children.

Algorithm 1: Pseudocode of a Sequence node with N children

```
for i ← 1 to N do
    childStatus ← Tick(child(i))
    if childStatus = Running then
        return Running
    else if childStatus = Failure then
        return Failure
return Success
```

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

BT - Fallback Node

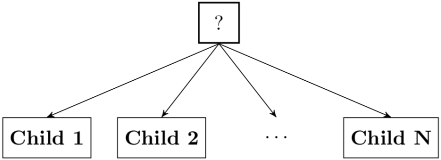


Fig. 1.3: Graphical representation of a Fallback node with N children.

Algorithm 2: Pseudocode of a Fallback node with N children

for $i \leftarrow 1$ **to** N **do**
    childStatus $\leftarrow$ `Tick` (*child(i)*)
    if *childStatus* $=$ *Running* **then**
        return *Running*
    else if *childStatus* $=$ *Success* **then**
        return *Success*
return *Failure*

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

BT - Parallel Node

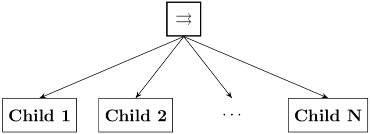


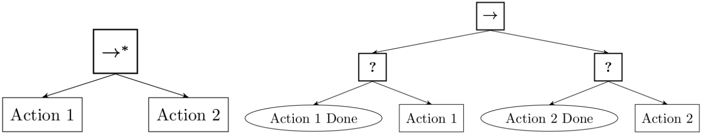
Fig. 1.4: Graphical representation of a Parallel node with N children.

Algorithm 3: Pseudocode of a Parallel node with N children and success threshold M

for $i \leftarrow 1$ **to** N **do**
  $childStatus(\mathrm{i}) \leftarrow$ `Tick` $(child(i))$
if $\Sigma_{i:childStatus(i)=Success} 1 \geq M$ **then**
  return *Success*
else if $\Sigma_{i:childStatus(i)=Failure} 1 > N - M$ **then**
  return *Failure*
return *Running*

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

BT - Memory Node



(a) Sequence composition with memory. **(b)** BT that emulates the execution of the Sequence composition with memory using nodes without memory.

Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

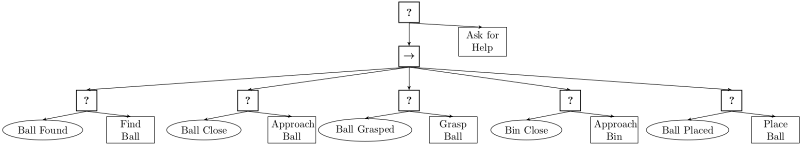
BT - Nodes

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	$\rightarrow$	If all children succeed	If one child fails	If one child returns Running
Parallel	$\rightrightarrows$	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	If impossible to complete	During completion
Condition	text	If true	If false	Never
Decorator	$\Diamond$	Custom	Custom	Custom

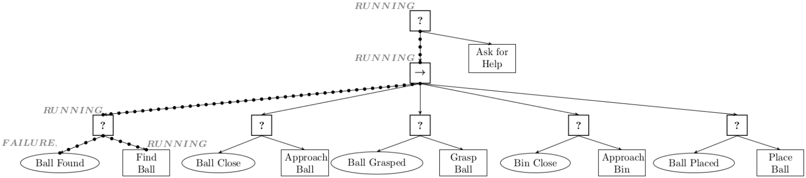
Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

BT - Example

Structure of the BT



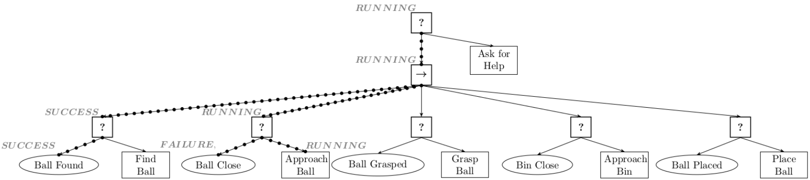
Start with searching the ball



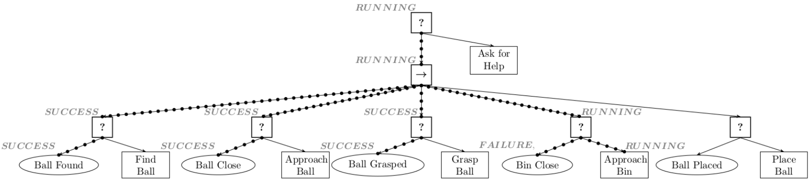
Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

BT - Example

Robot is approaching the ball

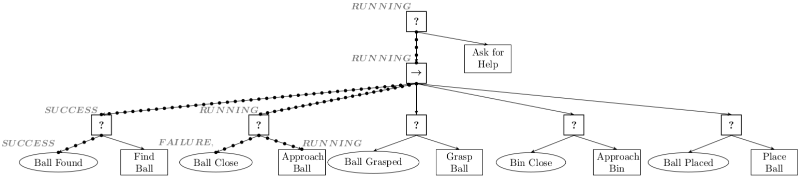


Robot is approaching the bin



BT - Example

Ball was removed from the hand



Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

BT - Real World Example

Video

Advantages and Disadvantages

Advantages:

- Good modularization and code reuse
- Good maintainability
- Frameworks and knowledge from usage in game industry
- Well formalized elements
- Parallelism possible

Disadvantages:

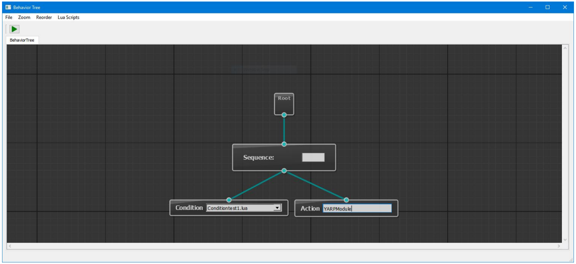
- Concept less intuitive
- Engine is complicated to implement
- Due to parallel activation current state difficult to see
- Testing preconditions indirect

BT - Conclusion

- Very powerful
- Good for large scenarios
- Also usable in smaller scenarios, but a bit overkill
- More complicated to learn but worth it

BT - Libraries and Tools

- Implemented in most big game engines
  - Unity
  - Unreal
  - ...
- behavior_tree (ROS package)
- YARP-Behavior-Trees (YARP Library)



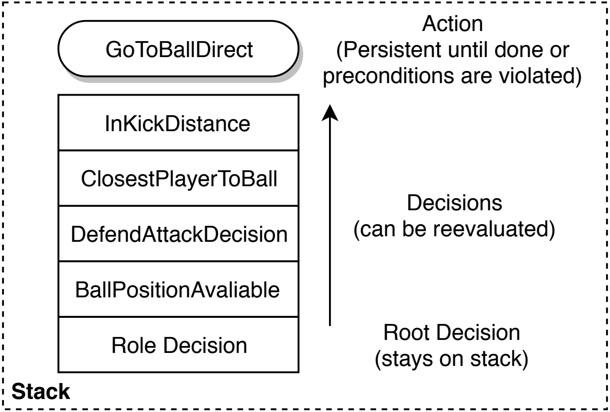
DSD - Motivation

- FSMs/HSMs not usable for non trivial problems
- BTs complicated to understand/implement, expensive to run
- Why not try to combine the advantages of the existing approaches
  - Tree structure (DT, BT)
  - Decision as internal nodes (DT)
  - Clear state (FSM, HSM)
  - Semantic transitions (FSM, HSM, DT)

Dynamic Stack Decider

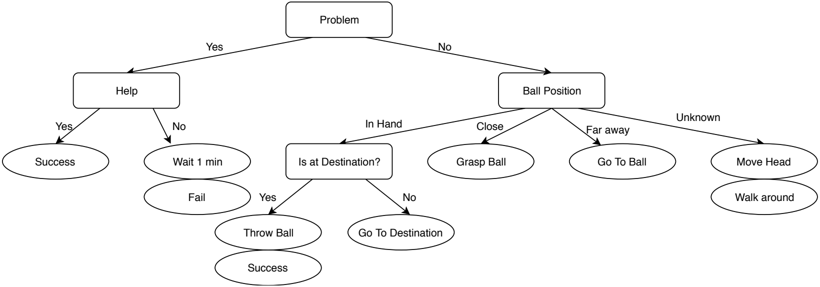
- Tree like structure
  - Internal nodes are decisions (no time component)
  - Leaf nodes are actions (time component)
- Decisions provide a semantic outcome (string)
- Tree structure is defined by a simple language
- State consists of current active action and previous decisions
- Decisions can be reevaluated to easily recheck preconditions
- But not all decisions should be redone every time
- Action can be concatenated as sequences

DSD - Stack



Poppinga, Martin and Bestmann, Marc. "ASDS: A Lightweight Approach for Dynamic Decision Making." (2018)

DSD - Example


Problem
Yes
No
Help
Ball Position
Yes
No
In Hand
Close
Far away
Unknown
Success
Wait 1 min
Fail
Is at Destination?
Grasp Ball
Go To Ball
Move Head
Walk around
Yes
No
Throw Ball
Success
Go To Destination

DSD - Tree Script

```
-->BallBehavior
$Problem
    YES --> $Help
        YES --> @Success
        NO --> @Wait1Min, @Fail
    NO --> $BallPosition
        IN_HAND --> $IsAtDest
            YES --> @ThrowBall, @Success
            NO --> @GoToDest
        CLOSE --> @GaspBall
        FAR_AWAY --> @GoToBall
        UNKNOWN --> @MoveHead, @WalkAround
```

DSD - Code Example

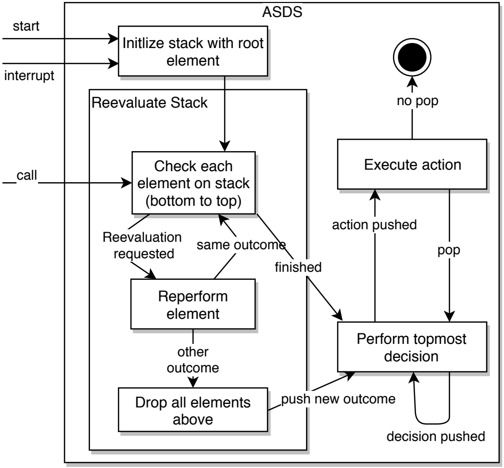
```
def BallPosition(AbstractDecisionElement):
    def perform(self):
        if not self.blackboard.ball_position:
            return "UNKNOWN"
        elif self.blackboard.ball_position < 0.5:
            return "IN_HAND"
        elif self.blackboard.ball_position < 1:
            return "CLOSE"
        else:
            return "FAR_AWAY"

    def get_reevaluate(self):
        return True

def GoToBall(AbstractActionElement):
    def perfom(self):
        # send some walking commands
def GrapsBall(AbstractActionElement):
    def __init__(self):
        # start grapsing animation

    def perform(self):
        if grasping_animation.is_finished():
            self.pop()
```

DSD - Flow


ASDS
start
interrupt
Initlize stack with root element
Reevaluate Stack
call
Check each element on stack (bottom to top)
Reevaluation requested
same outcome
finished
Reperform element
other outcome
Drop all elements above
push new outcome
no pop
Execute action
action pushed
pop
Perform topmost decision
decision pushed

Advantages and Disadvantages

Advantages:

- Clear seperation of decisions and actions
- Semantic transititions
- Time component of actions
- Clear state and previous decisions
- Simple checking of preconditions

Disadvantages:

- No parallelism
- Concept not perfectly intuitive

DSD - Conclusion

- Good for medium to complex scenarios
- Directly designed for robotics
- Not widely used
- Not well suited fo
- Suitable for analysis
  - In safety critical applications, analysis necessary
  - Ex: analyzing if robot stops fast enough when collision is detected

Libraries:

- dynamic_stack_decider (ROS)

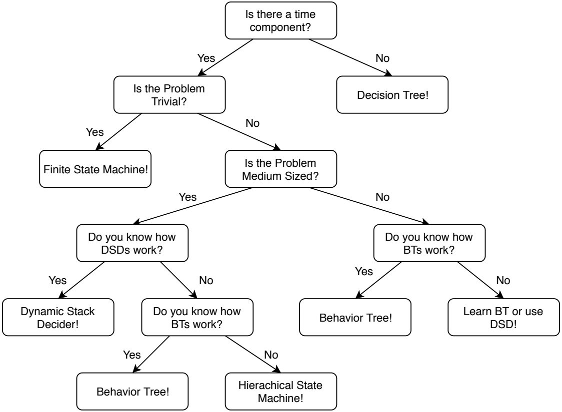
Comparison

	FSM	HSM	Sub.	DT	BT	DSD
Hierarchical organization	-	+	+	+	+	+
Reusable code	-	○	-	+	+	+
Modular design	-	+	+	+	+	+
Maintainability	-	○	-	-	+	+
Human readable	-	○	-	+	○	+
Stateful	+	+	-	-	+	+
Fast	-	-	+	+	○	+
Sufficiently expressive	+	+	-	+	+	+
Suitable for synthesis	+	+	-	+	○	+
Understandability	+	+	+	+	-	-
Implementation effort	+	○	-	+	-	○

(partly) Colledanchise, Michele, and Petter Ögren. "Behavior Trees in Robotics and AI, an Introduction.", 2017

Choosing an Architecture

My personal subjective proposal on choosing an architecture



Summary

- Theoretically any of the CAs can be used
- The choice is still important and very dependent on task and domain