Behavior-Based Approach

Sascha Lange, Christian Müller, Stefan Welker
Approach

- architectural requirements
  - behavior-based
  - modular
  - hierarchically

- feasibility requirements
  - purely reactive behavior should possible (right now 90% still is purely reactive)
  - other approaches should not be excluded (e.g. planning)

- more a collection of (unrelated) ideas than a complete theory

"this is not a big theory of behavior specification, but a framework to practically support the implementation. There is no abstract behavior specification language. It is a collection of classes, you make use of or derive your classes from and some „coding guidelines“ you should respect."
Skills / Behavior

- **getCmd**
- **Gain / Loose Control**
- **callbacks**

**Skill: needs parameters, e.g. target position** (DribbleToPos)

**Behavior: no parameters** (DribbleToGoal)

Skill / Behavior

+ getCmd(Time&): DriveVector
+ gainControl(Time&): void
+ loseControl(Time&): void
+ cycleCallback(Time&): void
Example behavior

BDribbleBallToGoal

BDribbleBallToGoal::BDribbleBallToGoal()
: Behavior("BDribbleBallToGoal"),
  skill(new SDribbleBallToPosRL())
{}

DriveVector BDribbleBallToGoal::getCmd(const Time& t)
throw(TribotsException) {

  //use information about the world to calculate
  //target position (in goal)
  FieldGeometry const& fgeom = MWM.get_field_geometry();
  Vec targetPos = Vec(0., fgeom.field_length / 2.);

  //use skill to produce drive commands
  skill->setParameters(targetPos, transVel);
  return skill->getCmd(t);
}
Inheritance is used intensively, build functionality layer by layer (Matryoshka)

Example 1:
- Dribble - Just the handling of the ball, learned with NFQ
- DribbleToGoal - adds obstacle avoidance and sets target to goal

Example 2:
- General defense behavior (cover ball, drive to it if possible)
- Field player behavior derived, position and location to cover is adapted to overall strategy
Arbitration

- Idea borrowed from BDI-architectures (MAS air-traffic controller)
- Interface extension to behaviors (conditions)
  - Invocation Condition (IC)
    - A behavior can take over control for the first time, if IC is fulfilled
    - Example EigenMove: ball possession close to side line
  - Commitment Condition (CC)
    - A behavior can keep control and does not have reached its goal, if this condition is met
    - Example Eigenmove: ball possession
Use IC and CC for the generic arbitration of behaviors

BDI-like Arbitrator

Belief: world model

Desire: drive command

Intention: active behavior
Types of Arbitration

- Highest Priority First (purely reactive, most used)
- Check CC of the active intention (possibly remove intention and signal loseControl)
- Run through list of options up to the active option (iff intention still active, otherwise up to the end) and check IC’s:
  - If IC is true make the currently inspected option to the intention (signal gainControl)
  - Active intention is then called by getCmd()

Algorithm 1 The "highest priority first" arbitration scheme.

**Require:** intention ≠ 0

if not intention.commitment_condition(t) then
  intention ← emergency_stop
endif

for i = 0 to options.length() do
  if options[i] = intention then
    break
  endif
  if options[i].invocation_condition(t) then
    intention ← options[i]
    break
  endif
end for

**Ensure:** intention ≠ 0
Algorithm 2 The "finish plan first" arbitration scheme.

Require: intention ≠ 0

if not intention.commitment.condition(t) then
    intention ← emergency_stop
    for $i = 0$ to options.length() do
        if options[\(i\)].invocation.condition(t) then
            intention ← options[\(i\)]
            break
        end if
    end for
end if

Ensure: intention ≠ 0
Example: Goalie

- Goalie, plain list
  (highest priority first)

Goalie

- BGameStopped
- BGoaliePenalty
- BGoalieGetAwayFromGoalPosts
- BGoaliePositioningChipKick
- BGoalieRaisedBall
- BGoalieFetchBallNearGoalPost
- BGoalieAttackBall
- BGoalieFetchBall
- BGoaliePositioning
- BGoaliePatrol
Sequence (used for complex behaviors)

intention → Option 1 → Option 2 → Option 3 → Option 4

only check IC of next option in list
Types of Arbitration

- **Sequence** (used for complex behaviors)
- **Generalized Sequence**
  - Node has to be activated / can be skipped
  - Present node cedes control / subsequent node grabs control

<table>
<thead>
<tr>
<th>intention</th>
<th>Option 1</th>
<th>Option 2</th>
<th>Option 3</th>
<th>Option 4</th>
</tr>
</thead>
</table>

Brainstormers Tribots, Neuroinformatics Group, University of Osnabrück, 2008  www.tribos.uos.de
Making this whole thing interesting: Nesting

Arbitrators (BDIBehavior) are Behaviors themselves

Behavior Hierarchy

LeftDefender

intention

Behavior

Behavior

BDIBehavior

Behavior

HasBal

intention

Behavior

BDIBehavior

BDIBehavior

Behavior

Skill
+getCmd(Time&): DriveVector
+gainControl(Time&): void
+loseControl(Time&): void
+cycleCallback(Time&): void

Behavior

+checkInvocationCondition(Time&): bool
+checkCommitmentCondition(Time&): bool

BDIBehavior

$options : std::vector<Behavior*>$
-Intention : Behavior*
Example: Decision Tree

- Stack Arbitrators (binary, n-ary, whatever)
- Highest Priority first Arbitration
- nodes are arbitrators
- leaves are behaviors
All our hierarchies can in principle be translated into an equivalent FSM.

However, it's a different way of thinking:

- You would have to spread / C&P IC and CC among the transitions.
- We assume a situation where history is not important.
Think in „Situations“

In this situation, the only correct decision is to shoot the ball.

(ok, obviously, I would have tried to dunk it, but trying to score is the right decision ;-)
Think in „Situations“

How you got there, is not important.

Whether you just dribbled there, ...
How you got there, is not important.

Whether you just dribbled there, ...
Think in „Situations“

... or you ran a really complex set play with several screens and passes...
Think in “Situations”

... or you ran a really complex set play with several screens and passes...
Think in "Situations"

... or you ran a really complex set play with several screens and passes...

Pick'n'Roll
Think in „Situations“

... or you ran a really complex set play with several screens and passes...

Pick'n'Roll
Think in "Situations"

... the correct decision will still be the same.

Shoot it.
Basketball players try to keep decisions as simple as possible.

They train and find solutions for simplified “situations” in “break-down drills“.
Think in „Situations“

So don‘t think about the history or state transitions, just decide what‘s the best action in the present situation, as the smart players do ;-) 

> it‘s an MDP ;-)
Different way of thinking:

- no transitions are specified
- practical benefits:
  - simply insert and delete nodes
  - recurring transition conditions centrally formulated
Different way of thinking:

- no transitions are specified
- practical benefits:
  - simply insert and delete nodes
  - recurring transition conditions centrally formulated
Conclusion

- Behavior based
- Generic arbitration schemes
- Behavior hierarchy using nested arbitrators
- Most behaviors are reactive
- Cut „situations“ from the state space instead of thinking in transition graphs
- more easily separate individual behavior and plug in a RL training setup
- more easily to integrate new behavior in an existing strategy