An Intuitive and Flexible Architecture for Intelligent Mobile Robots

By Xiao-Wen Terry Liu
Dept. of Computer Science
The University of Manitoba
Winnipeg, Manitoba
November 17 2005

Outline

- Introduction & Motivation
- Background – Robot Architectures
- Design
- Evaluation
- Conclusion

Introduction

- Goal: Develop an intuitive, adaptive, and flexible architecture for controlling intelligent mobile robots
- An ***architecture*** is a unifying, coherent form or method of construction, which provides the foundation for creating powerful intelligent systems.
- Intelligent :
  - Pragmatic definition of intelligence
  - Must act autonomously
  - Must perform appropriate action in controlled and uncertain situations

Intelligent behaviour

- Performing appropriate actions demonstrating behaviours that are working towards completing the system's objectives
- Difficulties:
  - Selecting the correct action among a very large set of possibilities
  - Real-time constraints
    - Reaction to danger and other events
    - Noisy sensors and imprecise actuators

Motivation

- Developing, maintaining, and modifying systems to control intelligent mobile robots in the real world can be a daunting task.

Motivation (II)

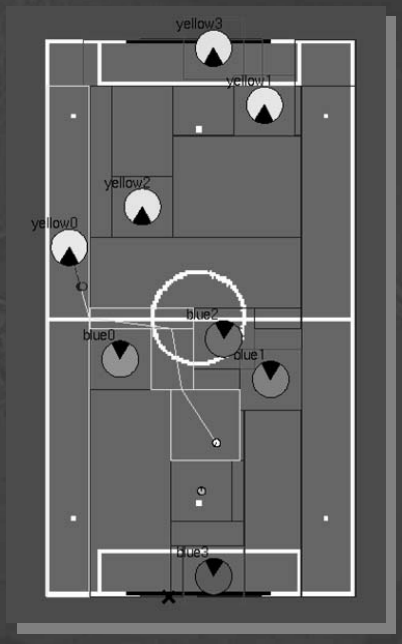
- Systems often limited by initial design
- New architecture's focus:
  - Adaptable: deals with noise and uncertainy
  - Flexible: add new tasks, change & remove existing tasks
  - Extensible: add new sensors/actuators
  - Intuitive: make the things above easy

Test Domain

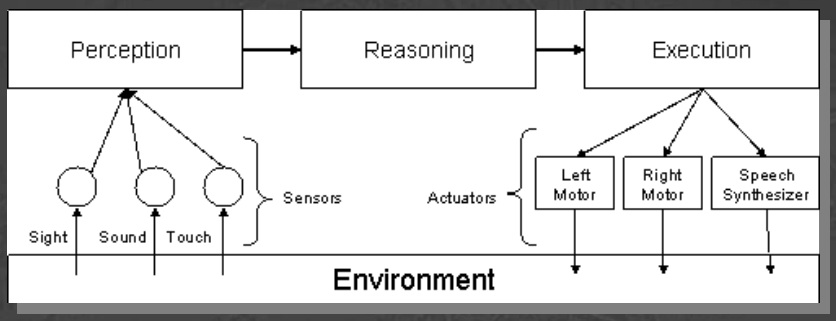
- Robotic soccer
  - Requires common fundamental skillset for mobile robots
    - Real-time control
    - Perception
    - Awareness
    - Planning
    - Coordination and Communication

Robotic soccer research

- Excellent testbed for research
  - Dynamic and complex domain
  - Large community of researchers



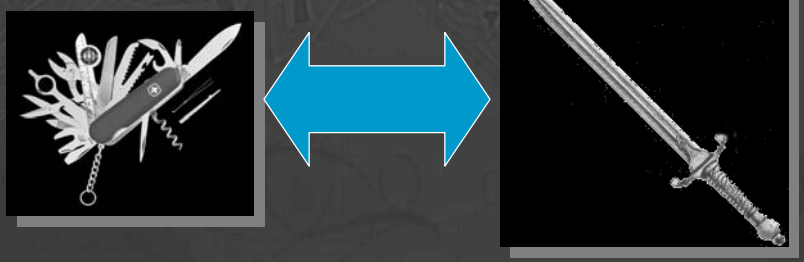
Background



- Architecture classifications
  - Domain Relevance vs. Domain Independence
  - Analysis vs. Synthesis
  - Top-down vs. Bottom-up
  - Deliberative vs. Reactive

Domain Relevance vs. Domain Independence

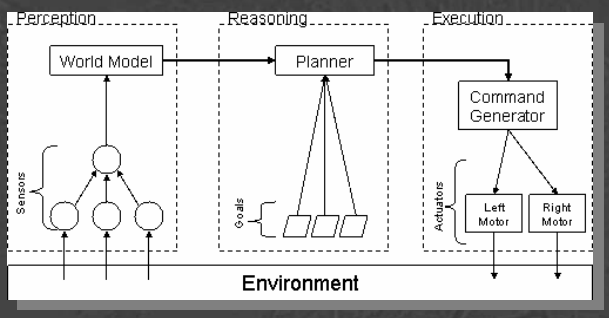
- Domain Relevance: specialized for a domain/task
- Domain Independence: functions for multiple domains
- Matter of Utility vs. Efficiency



Analysis vs. Synthesis

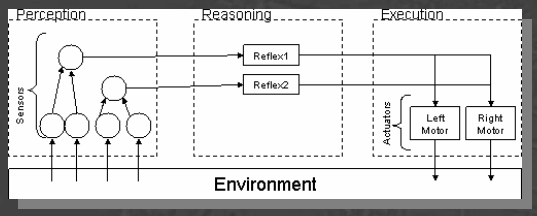
- Analysis: starts with an intelligence model
- Synthesis: starts with a basic unit/component of intelligence (unified field theory)

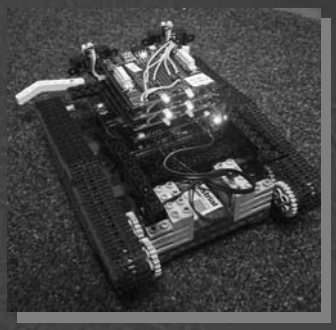
Top-down architecture



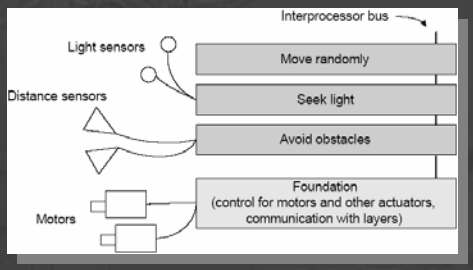
- The first type of architecture
- ***Knowledge-driven*** – takes a problem and decomposes it to further sub-problems
- Good for simple/routine tasks, however cannot cope with errors well.

Bottom-up architecture

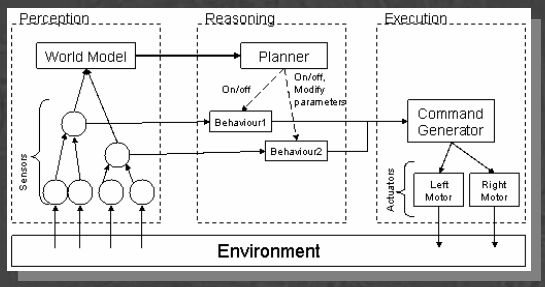




- More reactive by design
- Good for dealing with unexpected situations
- Example: Brooks' Subsumption Architecture



Hybrid Architecture



- Incorporates the advantages of both Top-down and Bottom-up design
  - But also the disadvantages
  - Importance is on finding a balance
- Examples: 3T, Atlantis, Aura, RCS, SSS
- Belief-Desire-Intention (BDI) Architecture
- Blackboard Architecture
- Ubiquitous Robot Architecture

Behaviour Specification

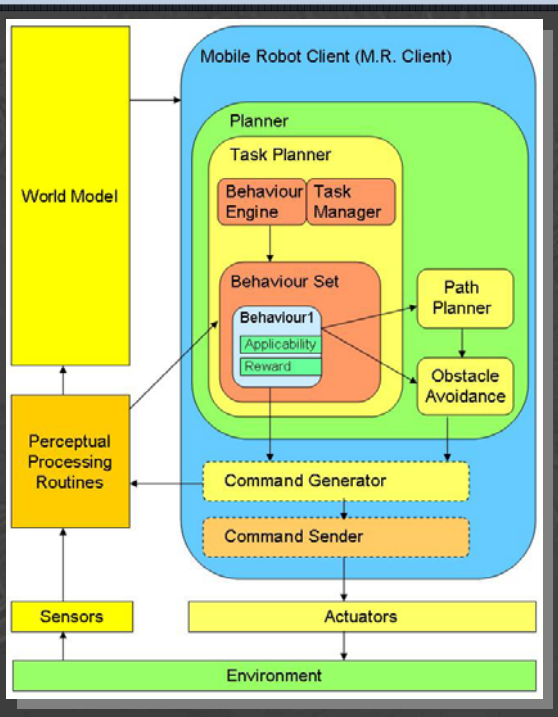
- There are some specialized languages for behaviour language
  - eXtensible Agent Behavior Specification Language (XABSL)

```
<option xsi:schemaLocation="http://www.ki.informatik.hu-berlin.de/XABSL2.1
../../../Xabsl2/xabsl-2.1/xabsl-2.1.option.xsd" name="striker" initial-state="initial">
 - <state name="initial">
    - <subsequent-basic-behavior ref="go-to">
       - <set-parameter ref="go-to.x">
          - <minus>
               <decimal-input-symbol-ref ref="ball.x"/>
               <decimal-value value="8"/>
            </minus>
         </set-parameter>
       - <set-parameter ref="go-to.y">
            <decimal-value value="11"/>
         </set-parameter>
      </subsequent-basic-behavior>
    - <decision-tree>
         <transition-to-state ref="initial"/>
      </decision-tree>
  </state>
</option>
```


Design

- Requirements: adaptable, flexible, extensible, and intuitive
- Archangel architecture
  - Sensor and Actuator modules
  - World Model
  - Sequencing
  - Timing Constraints
  - MRClients
    - Flexible Behaviour Selection Mechanism
  - Explicit Representation of behaviour specifications

Design Overview

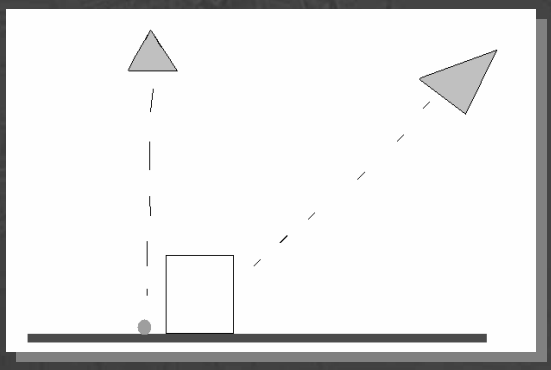


Sensors and Actuators

- Sensors and actuators use a loosely-coupled methodology to allow for extensibility
  - i.e. perceptual processing routines and command generator abstracts hardware from logical
- Use perceptual processing routines to link sensors to World Model
  - Filtering of object coordinates – e.g. obstacles, ball
  - Useful to deal with errors

World Model

- Very useful for purpose-driven (proactive) behaviours
- Can be used when sensors fail
  - For example, when the ball gets hidden behind another robot from the camera sensor

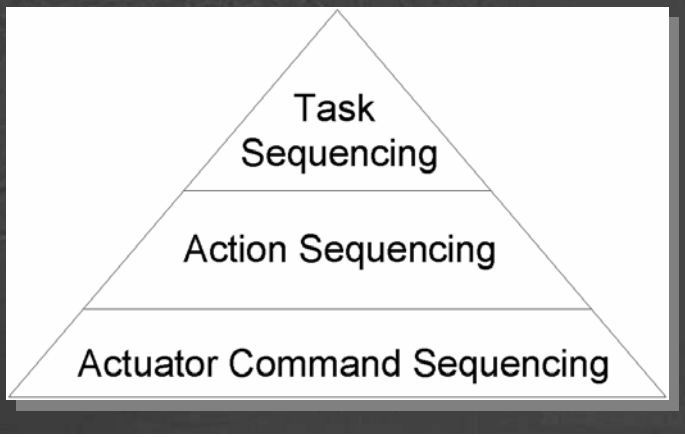


Sequencing

- Different levels of sequencing
  - Task Sequencing – ordering subtasks to complete the goal
    - E.g. Steal the ball, go behind the ball, then kick
  - Action Sequencing – e.g. a set of waypoints to move to destination
  - Actuator Command Sequencing – more performance related
    - E.g. help smooth out turns

Sequencing (II)

- Trend
  - Units more abstract up the pyramid
  - Fewer units queued up the pyramid
- More queuing usuals means better efficiency, however less reactivity

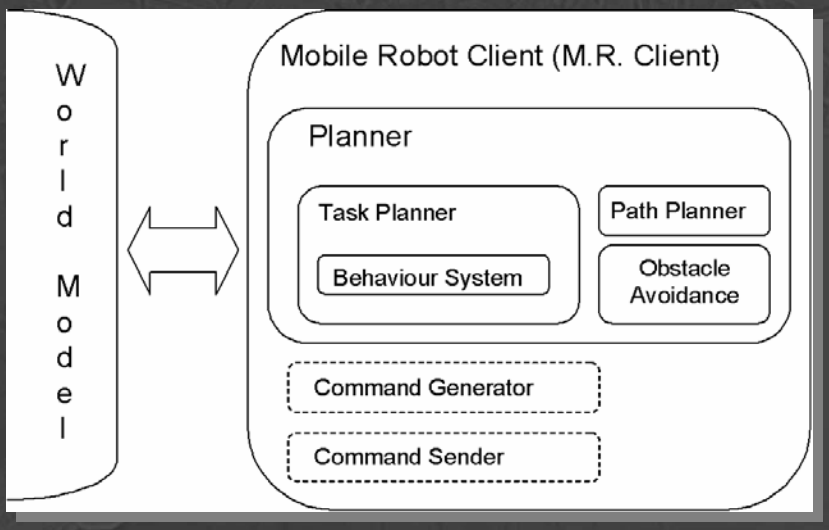


Timing Constraints

- Necessary to deal with real-time constraints
- Minimum time allowed for behaviours
  - Useful to avoid behaviour oscillation
- Maximum time allowed for behaviours
  - Useful to avoid local minima/maxima situations

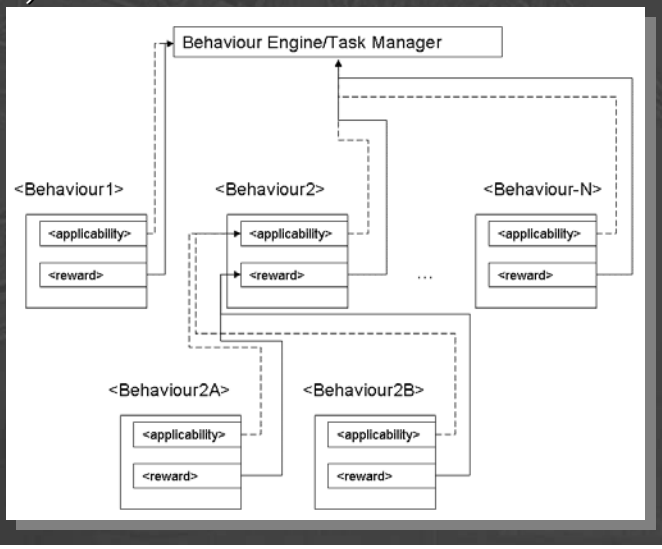
Mobile Robot Client

- Controls one (physical) robot



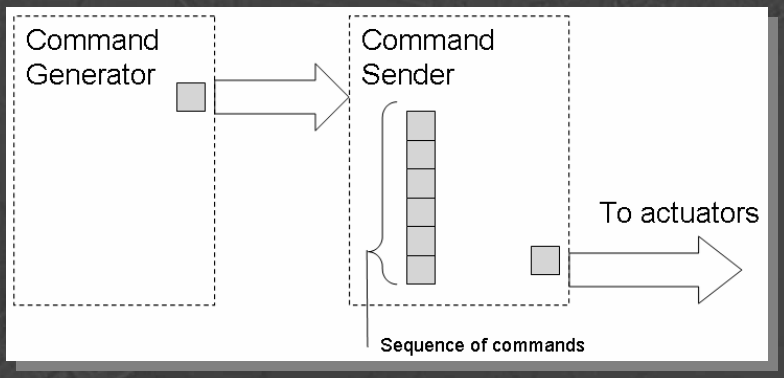
Behaviour System

- Uses competition as behaviour selection
- Behaviour with the largest activation (applicability+reward) value:



Command Generator & Sender

- Relates to the actuator command sequencing
- Abstracts the physical actuators



Explicit Representation

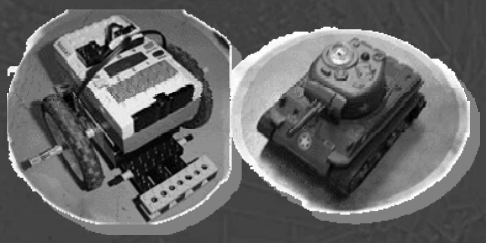
- Using eXtensible Markup Language (XML) to describe behaviours
- Different types of behaviours
- High-level complex behaviour representation
  - Behaviour-tree Composition
  - Finite State Machine
- Low-level action behaviours

Sample Behaviour Representation

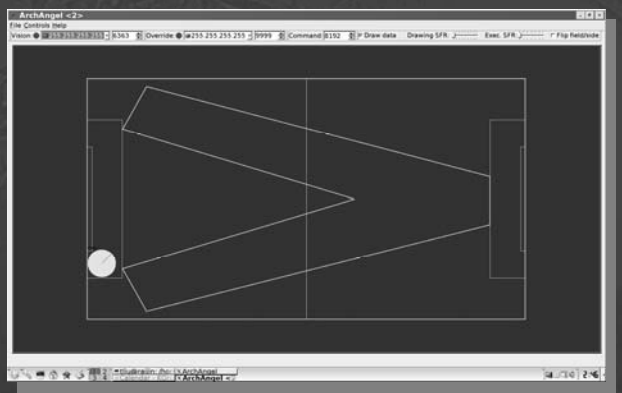
```
- <behaviour name="sampleBehaviour">
  - <init>
      <target_list ofType="obstacle" src="World::videoObjects"/>
    </init>
  - <draw_env>
      <!-- Home location -->
      <pen colour="red"/>
      <rect x="360" y="40" width="20" height="20"/>
    </draw_env>
  - <behaviour_list>
      <behaviour_ref name="chaseTarget"/>
      <behaviour_ref name="goHome"/>
      <behaviour_ref name="dance"/>
    </behaviour_list>
    <reward value="0.5"/>
  - <applicability>
    - <robot fartherThan="50">
        <reference_position reference="closestTarget"/>
        <add value="0.8"/>
      </robot>
    </applicability>
    <execute useBehaviourList="true"/>
</behaviour>
```


Implementation

- Robots



- System: Linux, QT3



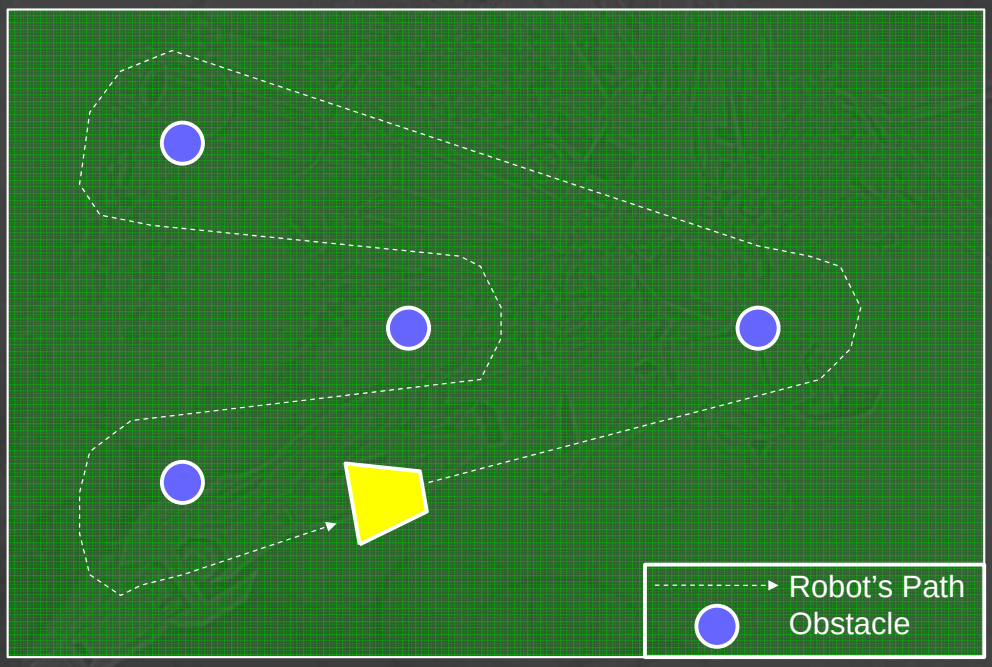
Evaluation

- Difficult to evaluate robot architecture.
  - (No universally accepted standard)
- User study – costly (time, money, and other resources)
- *Evaluation between architectures are more of a question of efficiency rather than computability* – R.C. Arkin
- Anecdotal evidence on several challenge tasks
  - Used soccer domain
  - RoboCup challenge tasks
- Bias

Challenges

- Simple tasks required of mobile robots applicable to many domains
  - Path Tracking (Racetrack)
  - Obstacle Avoidance (Obstacle Run)
  - Path Planning (Treasure Hunt)
  - Goal-Scoring (Shooting)
  - Robotic Interaction (Ball-Passing)

Path Tracking



Path Tracking

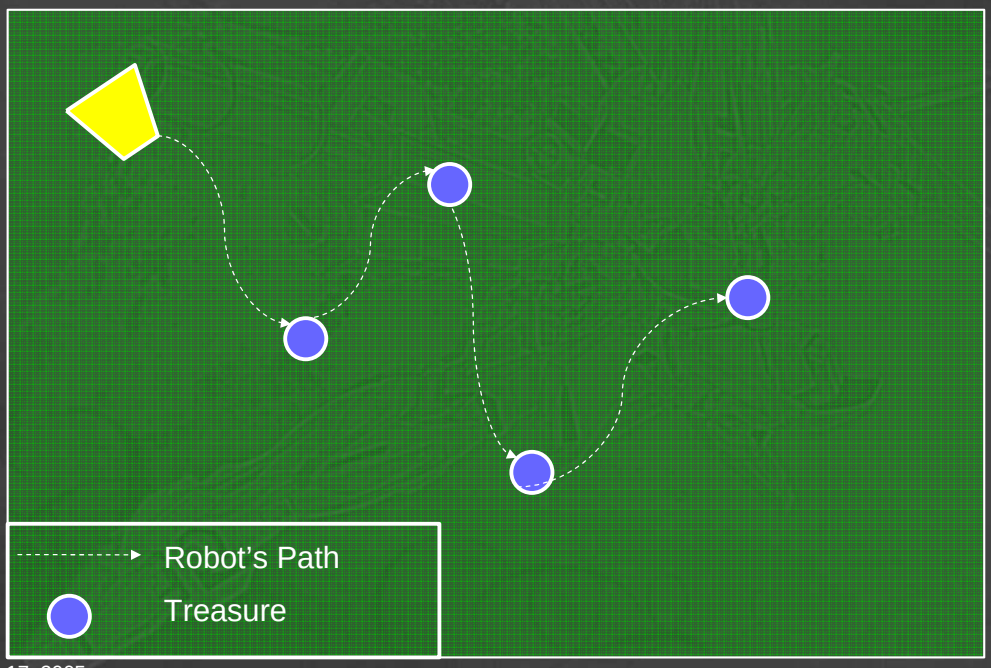
- Sample state

```
- <state name="state1">
  - <draw_env>
      <pen colour="red"/>
      <rect x="360" y="40" width="20" height="20"/>
    </draw_env>
  - <goto>
      <absolute_position x="370" y="50"/>
    </goto>
  - <next_state name="state2">
    - <condition>
      - <robot within="60">
          <absolute_position x="370" y="50"/>
        </robot>
      </condition>
    </next_state>
  </state>
```

VS.

370 50

Treasure Hunt



Treasure Hunt

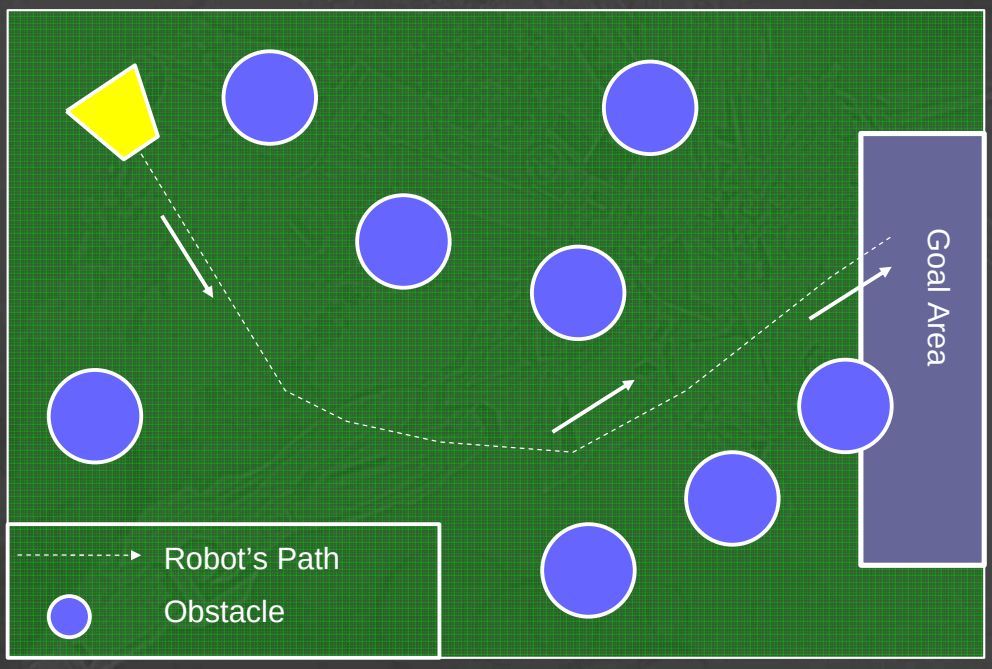
```
- <behaviour name="treasureHunt">
  - <init>
      <target_list ofType="obstacle" src="World::videoObjects"/>
    </init>
  - <behaviour_list>
      <behaviour_ref name="treasureHuntChase"/>
      <behaviour_ref name="treasureHuntTurn"/>
    </behaviour_list>
    <reward value="1"/>
    <applicability value="1"/>
    <execute useBehaviourList="true"/>
</behaviour>
```

```
- <behaviour name="treasureHuntChase">
    <init/>
    <reward value="1"/>
  - <applicability>
    - <condition>
      - <robot fartherThan="50">
          <reference_position reference="closestTarget"/>
          <add value="0.8"/>
        </robot>
      </condition>
    </applicability>
  - <execute>
    - <goto within="40">
        <reference_position reference="closestTarget"/>
      </goto>
    </execute>
</behaviour>
```

Treasure Hunt

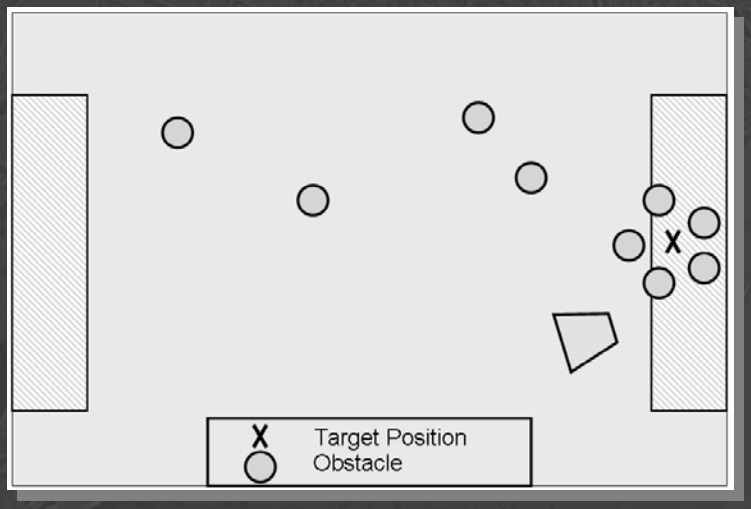
```
- <behaviour name="treasureHuntTurn" minExecMicroSecs="5000000">
    <init/>
    <reward value="1"/>
  - <applicability>
    - <condition>
      - <robot within="50">
          <reference_position reference="closestTarget"/>
          <add value="0.8"/>
        </robot>
      </condition>
    </applicability>
  - <execute initialState="stateTurn1" autoResetFSM="true">
    - <state name="stateTurn1">
      - <turn direction="towards">
          <reference_position reference="closestTarget"/>
        </turn>
      - <next_state name="stateMark">
        - <condition>
          - <robot within="50">
              <reference_position reference="closestTarget"/>
            </robot>
          </condition>
        </next_state>
      </state>
    - <state name="stateMark">
        <mark_complete target="closestTarget"/>
      - <next_state name="stateTurn2">
          <condition met="1"/>
        </next_state>
      </state>
    - <state name="stateTurn2">
      - <turn direction="towards">
          <reference_position reference="closestTarget"/>
        </turn>
      - <next_state name="stateTurn2">
          <condition met="1"/>
        </next_state>
      </state>
    </execute>
</behaviour>
```


Obstacle Run



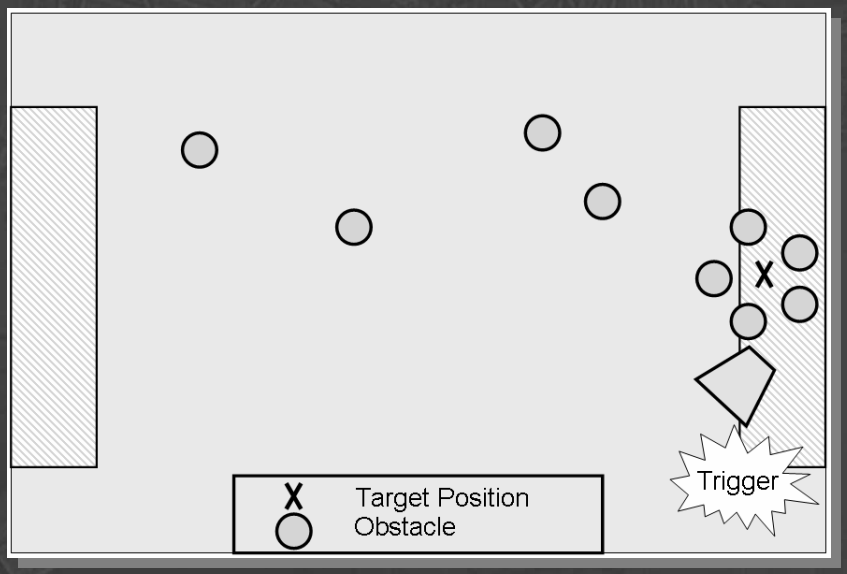
Obstacle Run

- Impossible scenario using single destination point



Obstacle Run

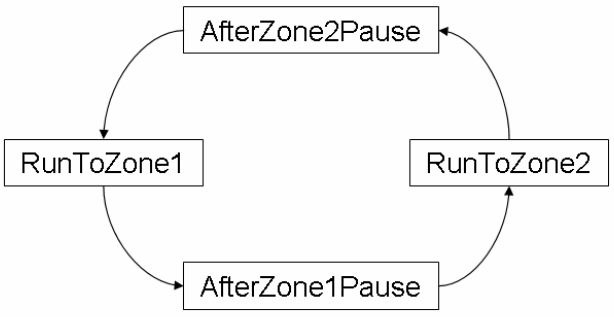
- Explicit trigger mechanism for zone



Obstacle Run

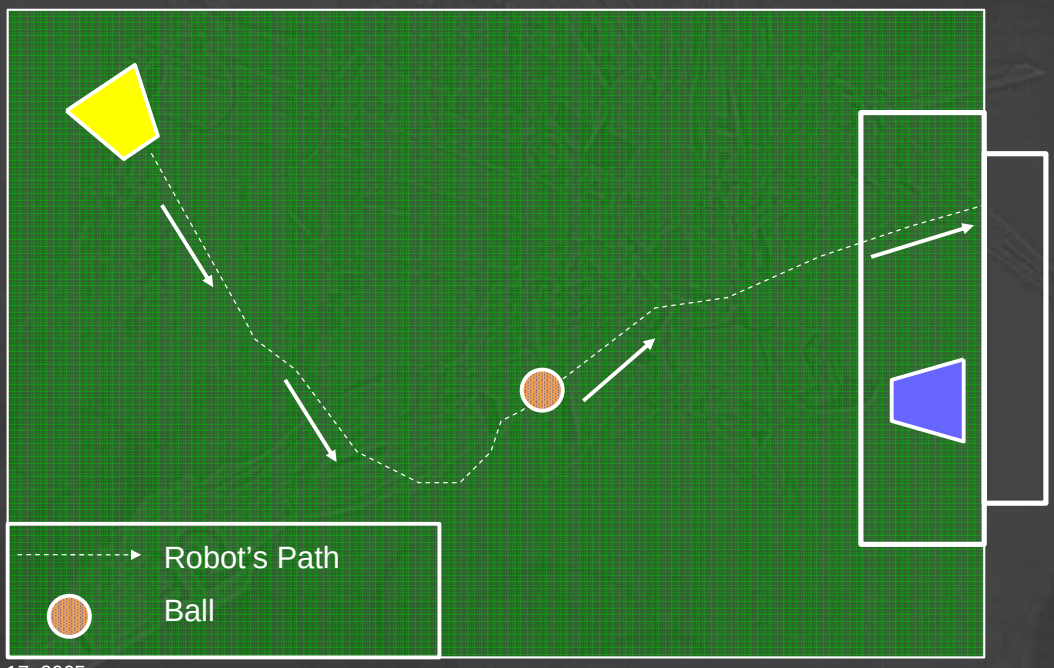
```
- <behaviour name="obstacleRun">
    <init/>
    <reward value="1"/>
    <applicability value="1"/>
  - <execute initialState="RunToZone1">
    - <state name="RunToZone1">
      - <goto>
          <control_command avoidBall="true"/>
          <absolute_position x="2640" y="760"/>
        </goto>
      - <next_state name="RunToZone2">
        - <condition>
          - <robot within="50">
              <absolute_position x="2640" y="760"/>
            </robot>
          </condition>
        </next_state>
        <!-- trigger if reach end zone 1 -->
      - <trigger_set>
        - <trigger_next_state targetName="RunToZone2">
          - <condition>
            - <robot>
                <within_rect x="2520" y="260" width="220" height="1000"/>
              </robot>
            </condition>
          </trigger_next_state>
        </trigger_set>
      </state>
```


Obstacle Run

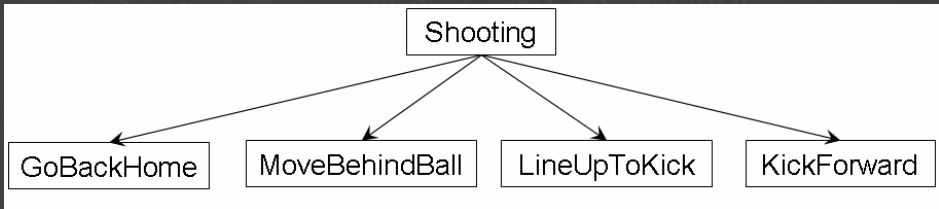



```
- <state name="AfterZone1Pause" minExecMicroSecs="5000000">
  - <goto>
      <reference_position reference="World::self"/>
    </goto>
  - <next_state name="RunToZone2">
      <condition met="true"/>
    </next_state>
  </state>
```

Goal Scoring



Goal Scoring

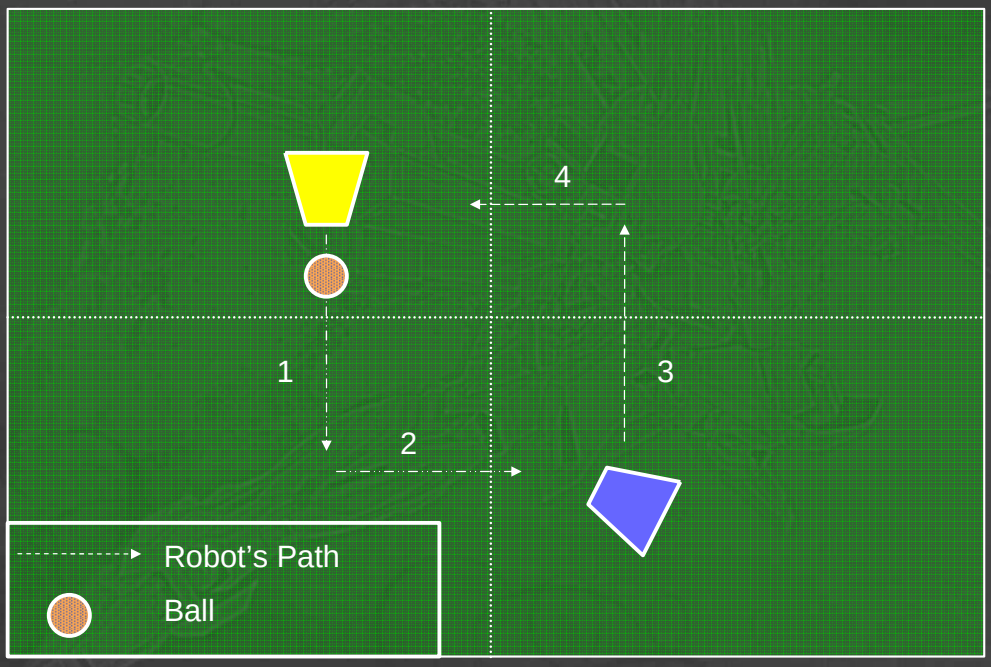



```
- <behaviour name="shooting">
  - <behaviour_list>
      <behaviour_ref name="GoBackHome"/>
      <behaviour_ref name="MoveBehindBall"/>
      <behaviour_ref name="LineUpToKick"/>
      <behaviour_ref name="KickForward"/>
    </behaviour_list>
    <reward value="1"/>
    <applicability/>
    <execute useBehaviourList="true"/>
  </behaviour>
```

Goal Scoring

```
- <behaviour name="kickForward" minExecMicroSecs="150000">
    <init/>
    <reward value="1"/>
  - <applicability>
    - <condition>
      - <robot within="180">
          <reference_position reference="World::ball"/>
          <add value="0.2"/>
        </robot>
      - <shot_on_goal checkFacingGoal="true" probability="high">
          <add value="0.25"/>
        </shot_on_goal>
      </condition>
    </applicability>
  - <execute initialState="kick" autoResetFSM="true">
    - <state name="kick" minExecMicroSecs="70000">
        <kick type="forward"/>
      - <next_state name="backup">
        - <condition>
            <ball isFound="false"/>
          </condition>
        </next_state>
      - <trigger_set>
        - <trigger_next_state targetName="backup">
          - <condition>
            - <ball>
                <within_rect x="2710" y="435" width="30" height="650"/>
              </ball>
            </condition>
          </trigger_next_state>
        </trigger_set>
      </state>
    - <state name="backup" minExecMicroSecs="70000">
        <kick type="reverse"/>
      - <next_state name="kick">
          <condition met="1"/>
        </next_state>
      </state>
    </execute>
</behaviour>
```

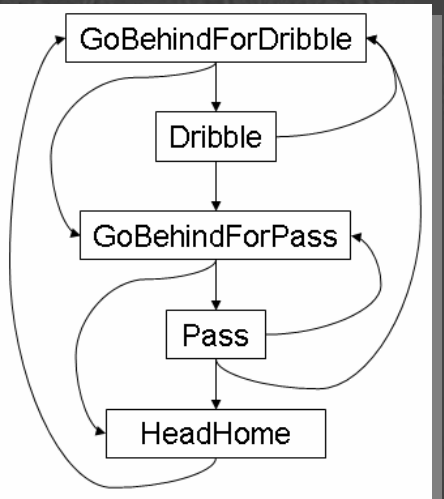

Passing challenge



Passing

- Passing
- Sub-behaviours:
  - GoBackHome
  - PassingDrill

- PassingDrill



Passing

```
<!-- ball is this half of the field-->
- <state name="goBehindBallForDribble">
  - <goto>
      <control_command avoidBall="true"/>
      <relative_abs_focus_position offsetPos="farBehind"
      reference="World::ball" focusPoint.x="600" focusPoint.y="200"/>
    </goto>
  - <next_state name="dribble">
    - <condition>
      - <robot within="50">
          <relative_abs_focus_position offsetPos="farBehind"
          reference="World::ball" focusPoint.x="600" focusPoint.y="200"/>
        </robot>
      </condition>
    </next_state>
   <!-- trigger if ball reach next quadrant -->
  - <trigger_set>
    - <trigger_next_state targetName="goBehindBallForPass">
      - <condition>
        - <ball isFound="true">
            <within_rect width="1370" x="0" y="0" height="600"/>
          </ball>
        </condition>
      </trigger_next_state>
    </trigger_set>
  </state>
```

```
- <state name="dribble">
  - <goto>
      <control_command avoidBall="false"/>
      <absolute_position x="600" y="200"/>
    </goto>
  - <next_state name="goBehindBallForPass">
    - <condition>
      - <ball isFound="true">
          <within_rect width="1370" x="0" y="0" height="600"/>
        </ball>
      </condition>
    </next_state>
  - <!--
     Go back behind ball if we're no longer pushing ball
    -->
  - <trigger_set>
    - <trigger_next_state targetName="goBehindBallForDribble">
      - <condition>
        - <robot>
            <within_rect width="1370" x="0" y="0" height="600"/>
          </robot>
          <!-- AND -->
        - <ball>
            <within_rect width="1370" x="0" y="600" height="920"/>
          </ball>
        </condition>
      </trigger_next_state>
    </trigger_set>
  </state>
```

Passing

```
- <state name="goBehindBallForPass">
  - <goto>
      <relative_abs_focus_position offsetPos="behind"
      reference="World::ball" focusPoint.x="2740" focusPoint.y="200"/>
    </goto>
  - <next_state name="pass">
    - <condition>
      - <robot within="50">
          <relative_abs_focus_position offsetPos="behind"
          reference="World::ball" focusPoint.x="2740"
          focusPoint.y="200"/>
        </robot>
      </condition>
    </next_state>
   <!-- trigger if ball crosses the line -->
  - <trigger_set>
    - <trigger_next_state targetName="headHome">
      - <condition>
        - <ball isFound="true">
            <within_rect width="1370" x="1410" y="0" height="1520"/>
          </ball>
        </condition>
      </trigger_next_state>
    </trigger_set>
</state>
```

```
- <state name="pass">
  - <goto>
      <absolute_position x="2740" y="200"/>
    </goto>
  - <next_state name="headHome">
    - <condition>
      - <ball isFound="true">
          <within_rect width="1370" x="1410" y="0" height="1520"/>
        </ball>
      </condition>
    </next_state>
  - <!--
     Go back behind ball if we're no longer pushing ball
    -->
  - <trigger_set>
    - <trigger_next_state targetName="goBehindBallForPass">
      - <condition>
        - <robot>
            <within_rect width="1370" x="1370" y="0" height="1520"/>
          </robot>
          <!-- AND -->
        - <ball>
            <within_rect width="1370" x="0" y="0" height="1520"/>
          </ball>
        </condition>
      </trigger_next_state>
    </trigger_set>
</state>
```


Evaluation

How easy is it to add/change/remove behaviours?

- Variations in tasks

- Metrics
  - Lines of Code (LoC) for changes
  - Time required for changes

Results

	Original C++ Client	Archangel
Code size		
Racetrack original design	50 lines	88 lines of XML
Racetrack change No. 1	added 1 line	edited 19 lines
Racetrack change No. 2	modified 7 lines	modified 21 lines
Treasure Hunt original design	50 lines	40 lines of XML
Treasure Hunt change No. 1	edited 10 lines	edited 14 lines
Treasure Hunt change No. 2	edited 1 line	remove 6 lines
Obstacle Run original design	40 lines	30 lines of XML
Obstacle Run change No. 1	added 4 lines	Add 18 lines of XML
Obstacle Run change No. 2	edited 30 lines	edited 18 lines
Goal Scoring original design	172 lines	84 lines of XML
Goal Scoring change No. 1	add 8 lines	add 8 lines
Goal Scoring change No. 2	edited 10 lines	add 11 lines
Goal Scoring change No. 3	edited 20 lines	edited 19 lines
Goal Scoring change No. 4	modify 2 lines	add 4 lines
Goal Scoring change No. 5	modify 14 lines	add 20 lines
Passing original design	320 lines	322 lines of XML
Passing change No. 1	modify 4 lines (x2)	modify 1 line (x2)
Passing change No. 2	modify 8 lines	added 12 lines (x2)
Passing change No. 3	modify 1 line (x2)	modify 1 line (x2)
Passing change No. 4	modify 1 line (x2)	add 1 line (x2)
Passing change No. 5	modify 3 line (x2)	modify 2 line (x2)

Results (II)

	Original C++ Client	Archangel
Development Time		
Racetrack original design	7 days	4 days
Racetrack change No. 1	8 minutes	8 minutes
Racetrack change No. 2	15 minutes	18 minutes
Treasure Hunt original design	3 days	3 days
Treasure Hunt change No. 1	20 minutes	8 minutes
Treasure Hunt change No. 2	6 minutes	4 minutes
Obstacle Run original design	2 days	1 hour
Obstacle Run change No. 1	40 minutes	2 hour
Obstacle Run change No. 2	30 minutes	10 minutes
Goal Scoring original design	7 days	5 days
Goal Scoring change No. 1	10 minutes	7 minutes
Goal Scoring change No. 2	9 minutes	7 minutes
Goal Scoring change No. 3	50 minutes	40 minutes
Goal Scoring change No. 4	9 minutes	6 minutes
Goal Scoring change No. 5	30 minutes	13 minutes
Passing original design	14 days	3 days
Passing change No. 1	23 minutes	15 minutes
Passing change No. 2	1 hour	1 hour
Passing change No. 3	10 minutes	6 minutes
Passing change No. 4	15 minutes	10 minutes
Passing change No. 5	25 minutes	15 minutes

Conclusion

- Lines of code is imperfect measurement
  - How much of the code is functional?
- Time required difficult to remove implication of bias
- Future Work
  - Additional domains and tasks

The End

(Questions?)