Behavior-based code generation for robots and autonomous agents

[ & related to AI in Games ]

Terrance Medina, Maria Hybinette, and Tucker Balch

University of Georgia
Georgia Institute of Technology

Aphaenogaster cockerelli while they forage for, subdue, and collect Drosophila melanogaster (fruit flies).
Motivation & Big Picture

- Enabling Agent-Based Modeling for non-programmers

- **Bigger** ‘Big Picture’:
  - Tracking system (animals)
  - Create Model
    - individual and multi agent based models
  - Run Model / Simulate and
  - Observe & Experimental validation of models
Motivation: Enabling ABM

- **Agent Based Modeling** is an essential tool for communities ranging from:
  - traffic analysis
  - military planning
  - social animal research
Motivation: Enabling ABM

- Big roadblock: Biologists are not programmers
- How can we unlock ABM for non-programmer researchers?
- How can we bridge this gap?
Motivation: Enabling ABM

Idea: Simplify the access to ABM

- Intermediate language: XML
- Supports behavior library
- Supports high performance simulation engine
Motivation: Enabling ABM

What does this enable? (in the future)

- GUI-based interfaces – ease of access
- Multiple “back end” languages
- Multiple “back end” platforms
The Rest of This Talk

- Some examples of ABM applications in biology: Ants, Fish (Big Picture)
- Details of the XML implementation
Ant HUNT Domain: Observe & Track
Behavior Model: Hybrid Controllers:

http://gritslab.gatech.edu/Droge/2012/01/behavior-based-mpc/
Simulation of Controller: Simulate (BioSim)
Test & Validate

- **Phase 1:** Initial test of model and refinement to align with live animal results (calibrated perception from 1 cm to \(\frac{1}{2}\) cm)
- **Phase 2:** Perturb (added obstacles) the environment and assess the predictive value of model.
Experimental Results: Phase 1
Challenge

- Constructing accurate animal behavior models is difficult because:
  - it is time intensive
  - it requires domain specialists (ethologists) to also be capable programmers
Our approach

- “on-the-fly” automatic behavior-model generation
- We combine behavior-based robot control architectures with an automatic code-generation framework
- XML used as an intermediate language
Advantages of our approach

- **Language-neutral**
  - we generate Java code from the XML, but we could choose from many suitable languages

- **Human readable/writable**

- **Machine readable/writable**
  - XML is a structured document, which can be created and modified programatically through an object model
Automatic code generation

- template based approach
- produce code based on some regex pattern
  - Style sheet (XSLT) match patterns in XML
- essentially the same problem as language compilation
  - e.g. YACC, Bison
Concept

Convert XML into executable Java

- use XSL (eXtensible Stylesheet Language)
- Transform(xml, xsl) --> output format
- A 'template' type of system
  - uses XPath to match element patterns
  - produces code snippets based on matched patterns
  - think of XPath like RegEx for XML docs
  - Example files next ...
Configuration XML

- Specifies the parameters of the simulation. (e.g., the number and types of agents, which controller modules they use, number and placement of physical objects)

- This simple example defines a single ant placed in the simulation, driven by a SpiralAntController controller, which is defined separately

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <appName>SpiralAnt</appName>
  <targetArch>Clay</targetArch>
  <targetKernel>MASON</targetKernel>

  <!-- AGENTS -->
  <agents>
    <agent>
      <type>Ants</type>
      <body>
        <name>PredatorBody</name>
      </body>
      <controller>
        <name>SpiralAntController</name>
      </controller>
      <defined_placement
        locX="0.10"
        locY="0.15"
        dirX="1"
        dirY="1"/>
      <radius>0.007</radius>
    </agent>
  </agents>
</config>
```
Controller Specification

1. Perceptual Schemas
2. Behaviors
3. Agent Schema
4. Triggers
5. FSM
1. Perceptual Schemas

- objects, locations and other agents
2. Behaviors

- simple reactive behaviors
- Motor Schemas transformed into instances of Clay behaviors
  - library of behavior based primitives
  - support of controlling a state machine.

```xml
<behaviors>
  <behavior name="MOVE_TO_HOMEBASE_1">
    type="linear_attraction"
    target="HOMEBASE_1"
    controlled_zone="1.1"
    dead_zone="0"
  </behavior>
  <behavior name="MOVE_TO_HOMEBASE_2">
    type="linear_attraction"
    target="HOMEBASE_2"
    controlled_zone="1.1"
    dead_zone="0"
  </behavior>
  <behavior name="MOVE_TO_HOMEBASE_3">
    type="linear_attraction"
    target="HOMEBASE_3"
    controlled_zone="1.1"
    dead_zone="0"
  </behavior>
  <behavior name="NOISE" type="noise" timeout="2"/>
  <behavior name="AVOID_OBSTACLES" type="avoid" p1="2.0" p2="2.0" target="OBSTACLE"/>
  <behavior name="SPIRAL" type="spiral" reset_when="TIMEOUT_5SEC"/>
  <behavior name="STOP" type="stop"/>
</behaviors>
```
3. Agent Schema

- Aggregated into groups of behaviors
- Serve as **states** in the finite state machine that control the transitions

```xml
<!— AGENT SCHEMA -->
<!— Behaviors co-ordinated by a FSM -->
<br/>&lt;states&gt;

  &lt;state name="MAKE_DECISION"&gt;
    &lt;coordinator type="weighted_sum"/&gt;
    &lt;behavior weight="1.0" embedded="STOP"/&gt;
  &lt;/state&gt;

  &lt;state name="GO_HOME_1"&gt;
    &lt;coordinator type="weighted_sum"/&gt;
    &lt;behavior weight="1.0" embedded="MOVE_TO_HOMEBASE_1"/&gt;
    &lt;behavior weight="5.0" embedded="AVOID_OBSTACLES"/&gt;
    &lt;behavior weight="1.5" embedded="NOISE"/&gt;
  &lt;/state&gt;

  &lt;state name="GO_HOME_2"&gt;
    &lt;coordinator type="weighted_sum"/&gt;
    &lt;behavior weight="1.0" embedded="MOVE_TO_HOMEBASE_2"/&gt;
    &lt;behavior weight="5.0" embedded="AVOID_OBSTACLES"/&gt;
    &lt;behavior weight="1.5" embedded="NOISE"/&gt;
  &lt;/state&gt;

  &lt;state name="GO_HOME_3"&gt;
    &lt;coordinator type="weighted_sum"/&gt;
    &lt;behavior weight="1.0" embedded="MOVE_TO_HOMEBASE_3"/&gt;
    &lt;behavior weight="5.0" embedded="AVOID_OBSTACLES"/&gt;
    &lt;behavior weight="1.5" embedded="NOISE"/&gt;
  &lt;/state&gt;

  &lt;state name="SPIRAL_CONT"&gt;
    &lt;coordinator type="weighted_sum"/&gt;
    &lt;behavior weight="1.0" embedded="SPIRAL"/&gt;
  &lt;/state&gt;

  &lt;state name="DEFAULT"&gt;
    &lt;coordinator type="weighted_sum"/&gt;
    &lt;behavior weight="1.0" embedded="STOP"/&gt;
  &lt;/state&gt;

&lt;/states&gt;
```
4. Triggers
   edges in a
   Finite State
   Machine

5. FSM
   States = Agent Schemas
   edges = Triggers

```xml
<configuration>
  <start_state>MAKE_DECISION</start_state>
  <transition from="GO_HOME_3" to="MAKE_DECISION" trigger="CLOSE_TO_HOME_3"/>
  <transition from="GO_HOME_2" to="MAKE_DECISION" trigger="CLOSE_TO_HOME_2"/>
  <transition from="GO_HOME_1" to="MAKE_DECISION" trigger="CLOSE_TO_HOME_1"/>
  <transition from="GO_HOME_3" to="SPIRAL_CONT" trigger="ANT_BUMP"/>
  <transition from="GO_HOME_2" to="SPIRAL_CONT" trigger="ANT_BUMP"/>
  <transition from="GO_HOME_1" to="SPIRAL_CONT" trigger="ANT_BUMP"/>
  <transition from="SPIRAL_CONT" to="MAKE_DECISION" trigger="TIMEOUT_10SEC"/>
  <transition from="MAKE_DECISION" to="GO_HOME_1" trigger="CHOOSE_1"/>
  <transition from="MAKE_DECISION" to="GO_HOME_2" trigger="CHOOSE_2"/>
  <transition from="MAKE_DECISION" to="GO_HOME_3" trigger="CHOOSE_3"/>
</configuration>
```
An even simpler XML behavior tree

```xml
<!-- BEHAVIORS -->
<behaviors>
  <behavior name="MOVE_TO_HOMEBASE" type="linear_attraction" target="HOMEBASE"/>
  <behavior name="NOISE" type="noise" timeout="2"/>
  <behavior name="AVOID_OBSTACLES" type="avoid" p1="1.0" p2="0.9" target="OBSTACLE"/>
</behaviors>

<!-- AGENT SCHEMA -->
<!-- Behaviors co-ordinated by a FSM -->
<states>
  <state name="GO_HOME">
    <coordinator type="weighted_sum"/>
    <behavior weight="1.0" embedded="MOVE_TO_HOMEBASE"/>
    <behavior weight="10.0" embedded="AVOID_OBSTACLES"/>
  </state>
</states>

<!-- TRIGGERS -->
<!-- Trigger transitions between FSM states -->
<triggers>
  <trigger name="START_MOVE" type="probability" p1="1"/>
</triggers>
```
This corresponds to ...

A simple model with one state

- Two behaviors
  - MOVE_TO_HOMEBASE
  - AVOID_OBSTACLES
- coordinated by a weighted average
- a single trigger, START_MOVE keeps us in the same state
XSL snippet

```xml
<xsl:for-each select="/behaviors/behavior">
  <xsl:choose>
    <xsl:when test="/type='avoid'">
      <xsl:value-of select="/name"/>!([CDATA[ = new v_Avoid vbCrLf(avoidPara_]])><xsl:value-of select="/name"/>!([CDATA[[0],
        avoidPara_]])><xsl:value-of select="/name"/>!([CDATA[[1],
        ]]>PS_<xsl:value-of select="/target"/>!([CDATA[, cb]); ]]></xsl:when>
    </xsl:when>

    <xsl:when test="/type='noise'">
      <xsl:value-of select="/name"/>!([CDATA[ = new v_Noise_()]]><xsl:value-of select="/timeout"/>!([CDATA[, ab]); ]]></xsl:when>
  </xsl:when>

  <xsl:when test="/type='linear_attraction'">
    <xsl:value-of select="/name"/>!([CDATA[ = new v_LinearAttraction_y(ab,
        1, 0.0, )]>PS_<xsl:value-of select="/target"/>!([CDATA[]]); ]]></xsl:when>
  </xsl:when>

  <xsl:otherwise></xsl:otherwise>
</xsl:choose>
</xsl:for-each>
```
And the resulting Java code

```java
//=============================
// MOTOR SCHEMAS
//==============================

MOVE_TO_HOMEBASE = new v_LinearAttraction_v(ab, 
  1, 0.0, PS_HOMEBASE);

NOISE = new v_Noise_{
  2, ab};

AVOID_OBSTACLES = new v_Avoid_va(
  avoidPara_AVOID_OBSTACLES[0],
  avoidPara_AVOID_OBSTACLES[1],
  PS_OBSTACLE, ab);
```

- The XSL has correctly generated the MOVE_TO_HOMEBASE and AVOID_OBSTACLES motor schemas for us
Executing the new code

- Since the code was dynamically generated, we'll compile and inject it to the JVM on the fly
- We use the ANT build tool for compilation
- Java's ClassLoader will load it for us
ANT (Another Neat Tool)

- We use a dynamically generated build.xml file to control the compilation process
- same XSL technique as previously
- a config xml file specifies simulation parameters
  - number of agents, controller types, and more
  - we use that config file to generate build file
  - ANT is invoked programmatically
- this also means an agent could (potentially) modify its own xml controller, and invoked ANT to recompile itself
Machine Generation of controllers

- JAXB
  - XML schema (XSD) describes the semantic structure of an XML controller
  - the xjc compiler generates an object model, a set of Java classes to create and populate XML elements
  - we use this object model with a random number generator, to generate random controllers
Machine Generation of controllers

XML is machine readable/writable
We have used this to randomly generate agent controllers using the JAXB framework (Java Architecture for XML Binding)
This is the “Agent Schemas” section of a randomly generated controller.
identifiers have a UUID appended to ensure uniqueness
Future work

- **Evolving controllers**
  - implement a genetic crossover mechanism, and hook the generation, crossover and **evaluation of the controllers** into an evolutionary computation framework, e.g., ECJ (George Mason)

- **GUI interface for creating/editing XML controllers**

- **Self adapting agents**
  - develop agent controllers with the ability to modify themselves and swap out controllers while executing
Future Target Domains: Fish Schooling

Video kindly provided by Iain Couzin and the Collective Animal Behavior Lab at Princeton University.


- Avoid: Separation
- Align
- Attract: Cohesion
Target Domain: Dolphins
Target Domain: Monkeys (Yerkes)

- Dominance Behavior
Target Domain: Apple Snails
Bigger Picture: Apple Snails.