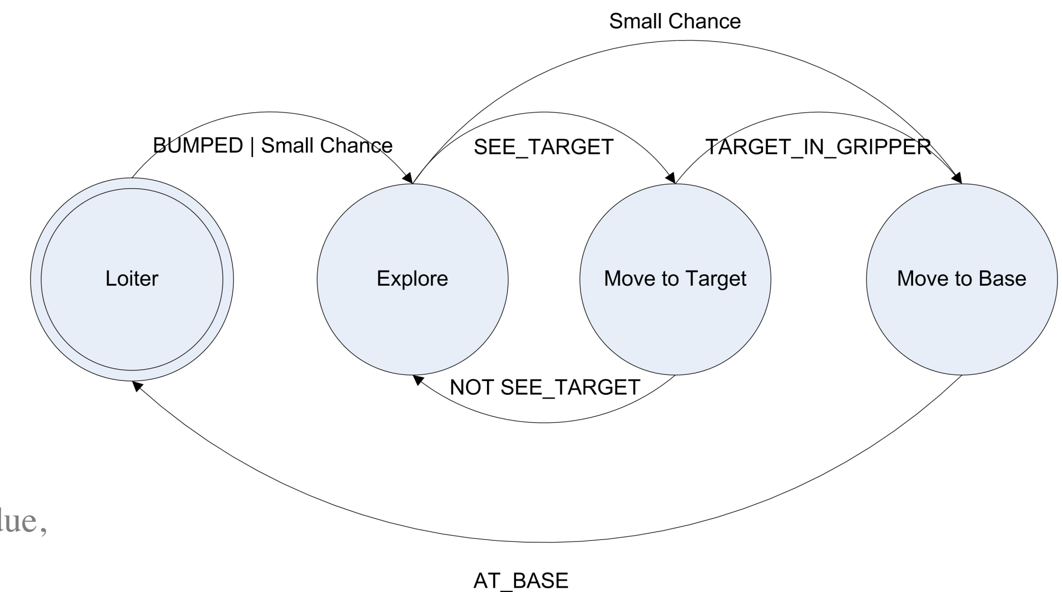
Behavior-based code generation
for robots and autonomous agents

[& 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).

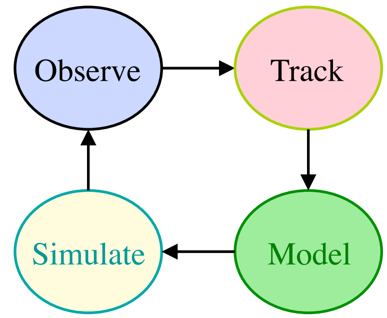
The University of Georgia

Georgia Institute of Technology

Ethogram

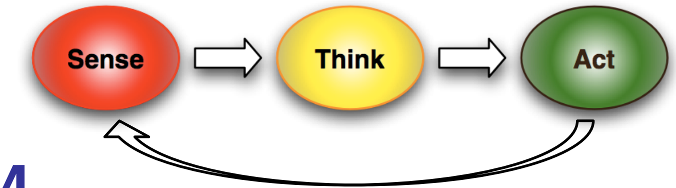
Motivation & Big Picture

- 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*



The University of Georgia

Georgia Institute of Technology



Motivation: Enabling ABM

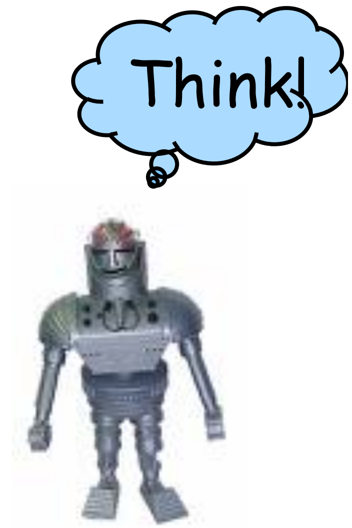
- **A**gent **B**ased **M**odeling is an essential tool for communities ranging from
  - traffic analysis
  - military planning
  - social animal research

The University of Georgia

Georgia Institute of Technology

Motivation: Enabling ABM

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



The University of Georgia

Georgia Institute of Technology

Motivation: Enabling ABM

Idea: Simplify the access to ABM

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

The University of Georgia

Georgia Institute of Technology

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 University of Georgia

Georgia Institute of Technology

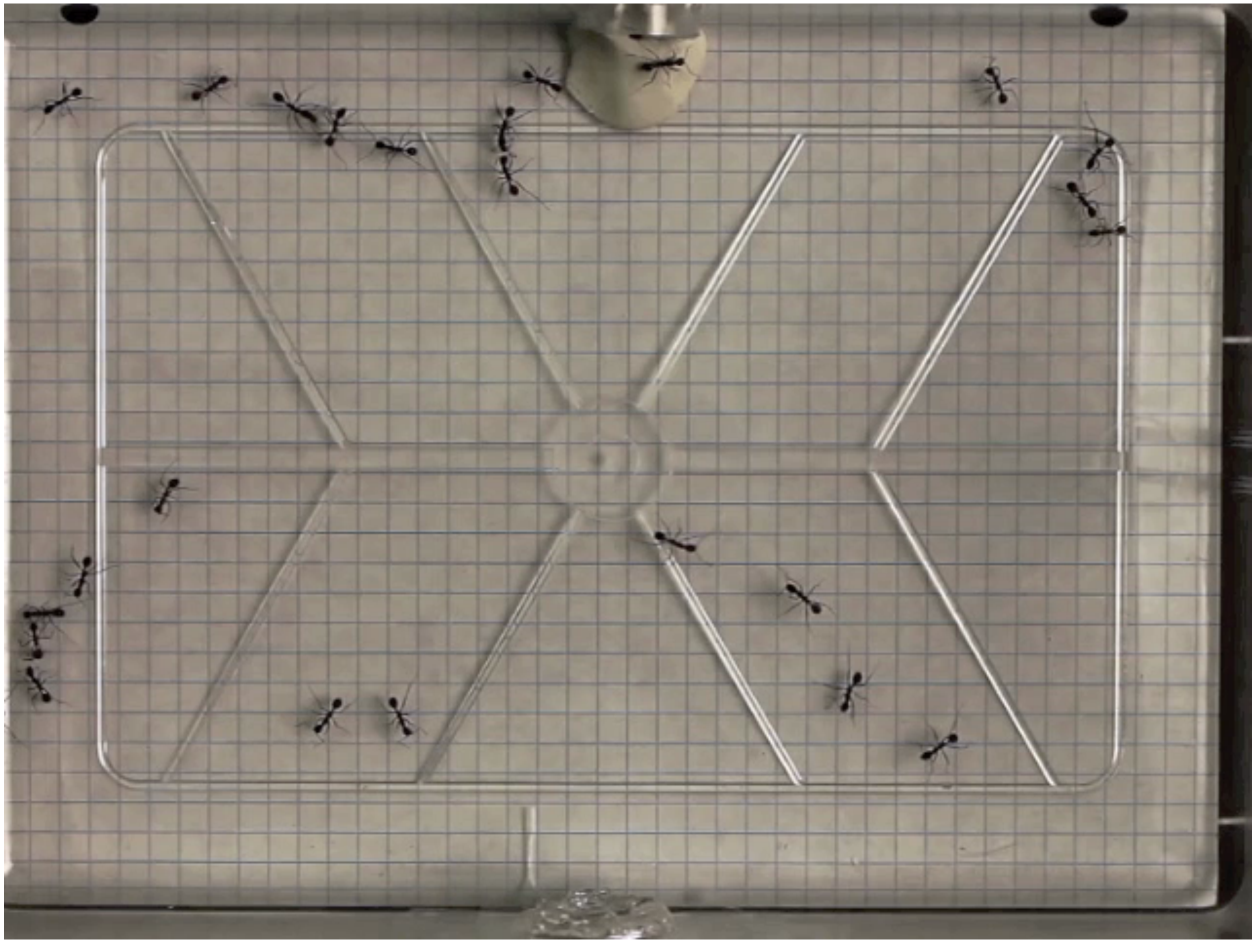
The Rest of This Talk

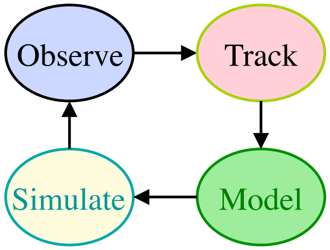
- Some examples of ABM applications in biology: Ants, Fish (Big Picture)
- Details of the XML implementation

The University of Georgia

Georgia Institute of Technology

Ant HUNT Domain: Observe & Track

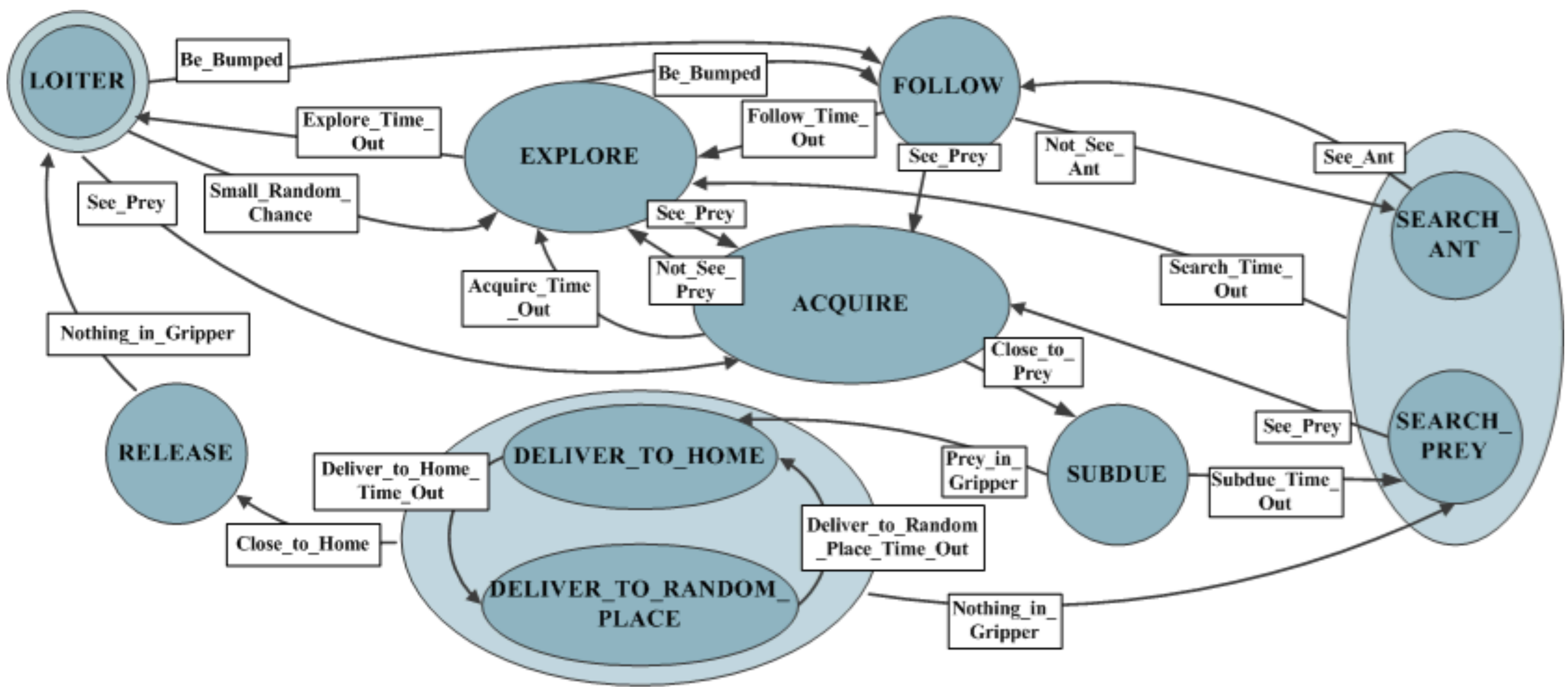




The University of Georgia

Georgia Institute of Technology

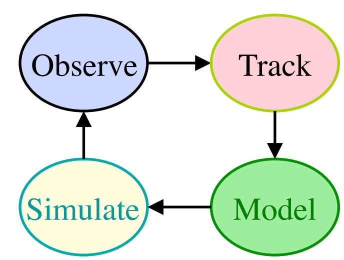
Behavior Model: Hybrid Controllers:



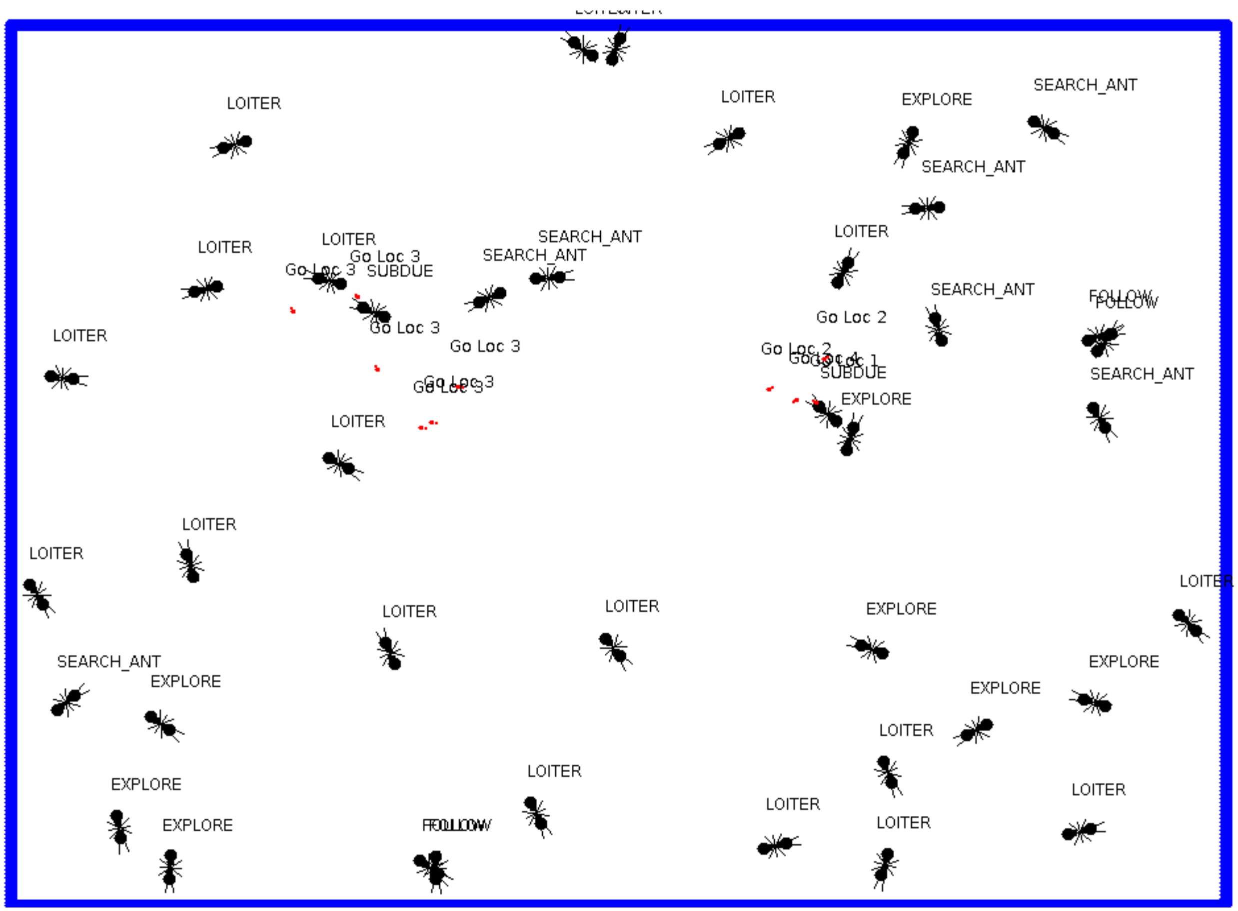
http://gritslab.gatech.edu/Droge/2012/01/behavior-based-mpc/

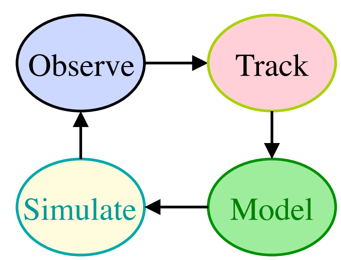
The University of Georgia

Georgia Institute of Technology



Simulation of Controller : Simulate (BioSim)



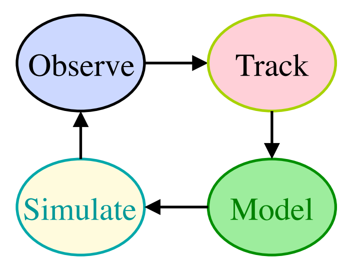


The University of Georgia

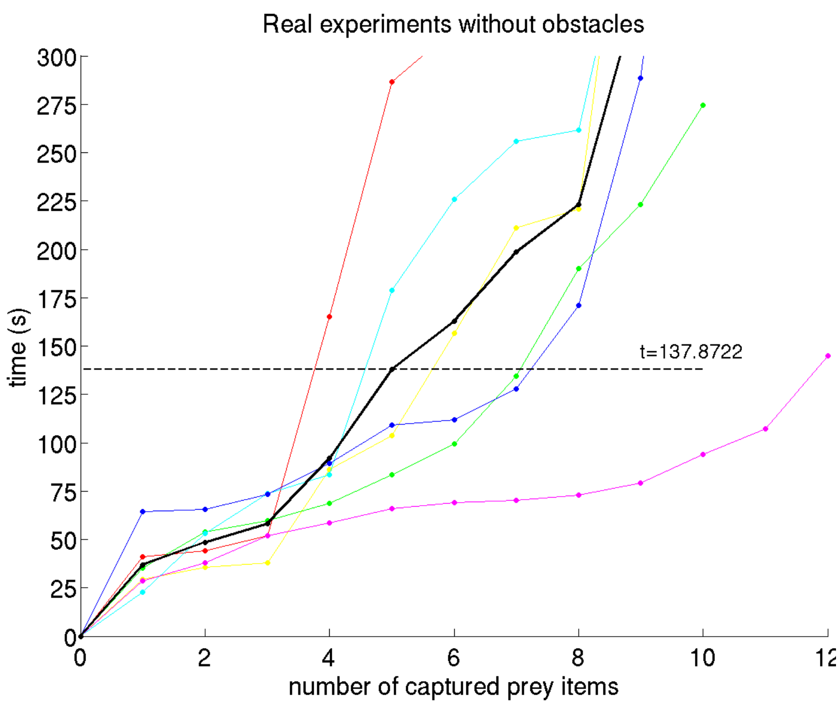
Georgia Institute of Technology

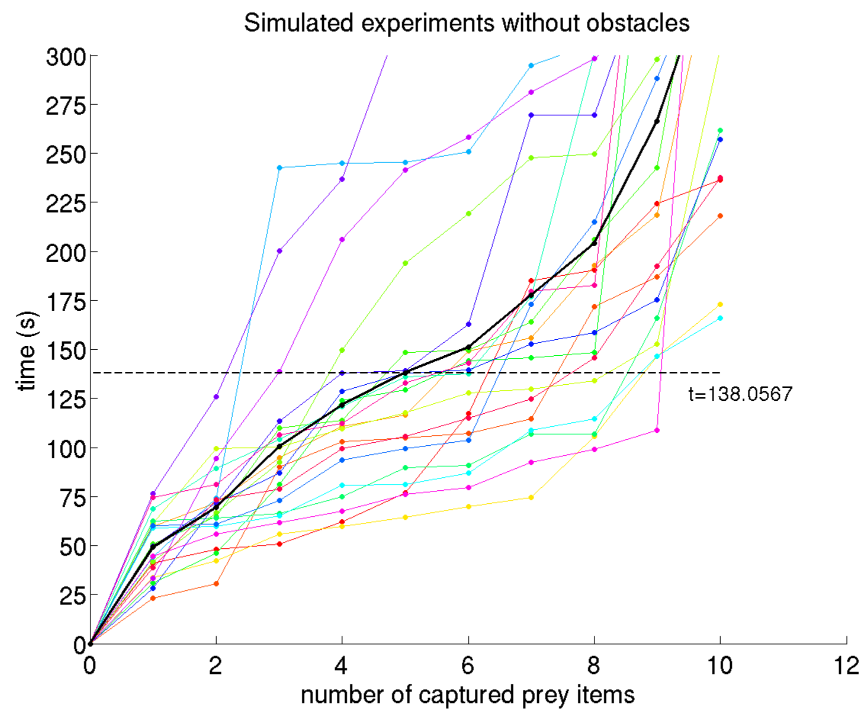
Test & Validate

- **Phase 1:** Initial test of model and *refinement* to align with live animal results (calibrated perception from 1 cm to ½ cm)
- **Phase 2: Perturb** (added obstacles) the environment and assess the predictive value of model.



Experimental Results: Phase 1





The University of Georgia

Georgia Institute of Technology

Challenge

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

The University of Georgia
Georgia Institute of Technology

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

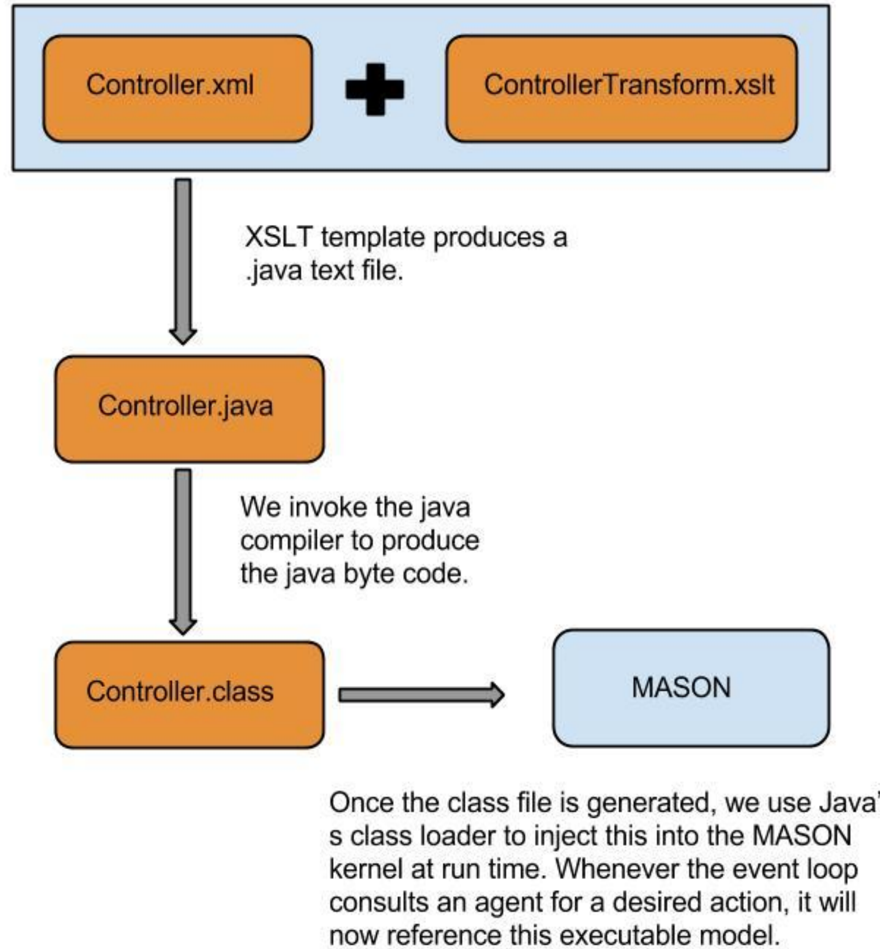
- template based approach
- produce code based on some regex pattern
  - Style sheet (XSLT) match patterns in XML
- essentially the same problem as language compilation
  - e.g. YACC, Bison

The University of Georgia
Georgia Institute of Technology

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>
```

The University of Georgia 1785

Georgia Institute of Technology

Controller Specification

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

1. Perceptual Schemas

- objects, locations and other agents

```
<!-- PERCEPTUAL SCHEMAS -->
<!-- Things that the agent can perceive in its environment-->
<objects>
    <object name="OBSTACLE" type="obstacle" size="3"/>
    <object name="HOMEBASE_1" type="fixed" x="0.02" y="0.05"/>
    <object name="HOMEBASE_2" type="fixed" x="0.22" y="0.02"/>
    <object name="HOMEBASE_3" type="fixed" x="0.19" y="0.15"/>
    <object name="CLOSEST_ANT" type="closest" size="3"/>
</objects>
```

2. Behaviors

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

```
<!--    BEHAVIORS   -->
<behaviors>

    <behavior
                name="MOVE_TO_HOMEBASE_1"
                type="linear_attraction"
                target="HOMEBASE_1"
                controlled_zone="1.1"
                dead_zone="0"
    />
    <behavior
                name="MOVE_TO_HOMEBASE_2"
                type="linear_attraction"
                target="HOMEBASE_2"
                controlled_zone="1.1"
                dead_zone="0"
    />
    <behavior
                name="MOVE_TO_HOMEBASE_3"
                type="linear_attraction"
                target="HOMEBASE_3"
                controlled_zone="1.1"
                dead_zone="0"
    />
    <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

```
<!-- AGENT SCHEMA -->
<!-- Behaviors co-ordinated by a FSM -->
<states>

    <state name="MAKE_DECISION">
        <coordinator type="weighted_sum"/>
            <behavior weight="1.0" embedded="STOP"/>
    </state>

    <state name="GO_HOME_1">
        <coordinator type="weighted_sum"/>
        <behavior weight="1.0" embedded="MOVE_TO_HOMEBASE_1"/>
        <behavior weight="5.0" embedded="AVOID_OBSTACLES"/>
        <behavior weight="1.5" embedded="NOISE"/>
    </state>

    <state name="GO_HOME_2">
        <coordinator type="weighted_sum"/>
        <behavior weight="1.0" embedded="MOVE_TO_HOMEBASE_2"/>
        <behavior weight="5.0" embedded="AVOID_OBSTACLES"/>
        <behavior weight="1.5" embedded="NOISE"/>
    </state>

    <state name="GO_HOME_3">
        <coordinator type="weighted_sum"/>
        <behavior weight="1.0" embedded="MOVE_TO_HOMEBASE_3"/>
        <behavior weight="5.0" embedded="AVOID_OBSTACLES"/>
        <behavior weight="1.5" embedded="NOISE"/>
    </state>


    <state name="SPIRAL_CONT">
        <coordinator type="weighted_sum"/>
        <behavior weight="1.0" embedded="SPIRAL" />
    </state>

    <state name="DEFAULT">
        <coordinator type="weighted_sum"/>
            <behavior weight="1.0" embedded="STOP"/>
    </state>

</states>
```

4. Triggers
edges in a Finite State Machine

# 5. FSM
States = Agent Schemas
edges = Triggers

```
<!-- TRIGGERS -->
<!-- Trigger transitions between FSM states -->
<triggers>
        <trigger name="CLOSE_TO_HOME_1" distance="0.009" type="close" subject="self" object="HOMEBASE_1" />
        <trigger name="CLOSE_TO_HOME_2" distance="0.009" type="close" subject="self" object="HOMEBASE_2" />
        <trigger name="CLOSE_TO_HOME_3" distance="0.009" type="close" subject="self" object="HOMEBASE_3" />
        <trigger name="CHOOSE_1" type="probability" p1="0.333"/>
        <trigger name="CHOOSE_2" type="probability" p1="0.333"/>
        <trigger name="CHOOSE_3" type="probability" p1="0.333"/>
        <trigger name="ANT_BUMP" type="close" distance="0.1" subject="self" object="CLOSEST_ANT" />
        <trigger name="TIMEOUT_5SEC" type="timeout" duration="5.0"/>
        <trigger name="TIMEOUT_10SEC" type="timeout" duration="10.0"/>
</triggers>


<!-- FSM CONFIG -->
<!-- The FSM transition table -->
<!-- FSM = {states, transition table} -->
<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>
```

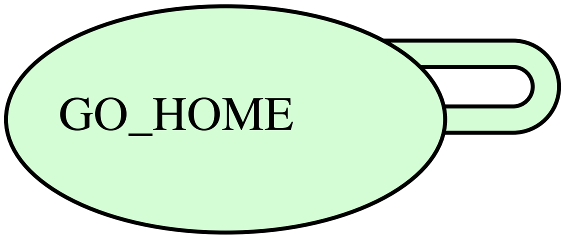
The University of Georgia

Georgia Institute of Technology

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



The University of Georgia

Georgia Institute of Technology

XSL snippet

```
//=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
//      MOTOR SCHEMAS
//=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
]]>
<xsl:for-each select="/*/behaviors/behavior">

        <xsl:choose>


        <xsl:when test="./@type='avoid'">

                <xsl:value-of select="./@name"/><![CDATA[ = new v_Avoid_va(
                avoidPara_]]><xsl:value-of select="./@name"/><![CDATA[[0],
                avoidPara_]]><xsl:value-of select="./@name"/><![CDATA[[1],
                ]]>PS_<xsl:value-of select="./@target"/><![CDATA[, ab); ]]>
                </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 test="./@type='linear_attraction'">

                <xsl:value-of select="./@name"/><![CDATA[ = new v_LinearAttraction_v(ab,
                1, 0.0, ]]>PS_<xsl:value-of select="./@target"/><![CDATA[); ]]>

        </xsl:when>


        <xsl:otherwise></xsl:otherwise>
        </xsl:choose>

</xsl:for-each>
```

The University of Georgia

Georgia Institute of Technology

And the resulting Java code

```
//=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
//      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

The University of Georgia

Georgia Institute of Technology

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 invoke ANT to recompile itself

The University of Georgia

Georgia Institute of Technology

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

The University of Georgia

Georgia Institute of Technology

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.

```xml
<states>
    <state name="STATE_d6b80f13_0eaa_4a6d_8ce1_7bc06f29b2e0">
        <coordinator type="weighted_sum"/>
        <behavior weight="0.9063369986056548" embedded="NOISE_030dcbc8_1cac_4b18_aefc_b7cef1670dba"/>
        <behavior weight="0.02907630834143604" embedded="SPIRAL_06db2da1_50e5_46f0_b356_b6f93575953c"/>
    </state>
    <state name="STATE_d9baf5b1_8674_4b26_9c44_9774b499d1be">
        <coordinator type="weighted_sum"/>
        <behavior weight="0.6968335310401979" embedded="NOISE_030dcbc8_1cac_4b18_aefc_b7cef1670dba"/>
        <behavior weight="0.921233444740039" embedded="SPIRAL_06db2da1_50e5_46f0_b356_b6f93575953c"/>
    </state>
    <state name="STATE_e441a603_e46a_402d_a161_3f68feacc480">
        <coordinator type="weighted_sum"/>
        <behavior weight="0.9765308628380032" embedded="SPIRAL_06db2da1_50e5_46f0_b356_b6f93575953c"/>
        <behavior weight="0.36464689653424" embedded="NOISE_70f1c0e0_f8a5_4055_a180_82b7873c1a57"/>
        <behavior weight="0.668687902900071" embedded="SPIRAL_514fabc3_fd8c_43ea_b390_4d764e0db78d"/>
    </state>
    <state name="STATE_11cae3f2_d259_42b5_b9b6_c444d5f24ab3">
        <coordinator type="weighted_sum"/>
        <behavior weight="0.06515032432499868" embedded="SPIRAL_06db2da1_50e5_46f0_b356_b6f93575953c"/>
    </state>
```


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

The University of Georgia

Georgia Institute of Technology

Future Target Domains: Fish Schooling

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

G-Tech BioSim Lab: Brian Hrolenok and Tucker Balch.

- Avoid: Separation
- Align
- Attract: Cohesion

The University of Georgia

Georgia Institute of Technology

Target Domain: Dolphins

The University of Georgia

Georgia Institute of Technology

Target Domain: Monkeys (Yerkes)

- Dominance Behavior

The University of Georgia

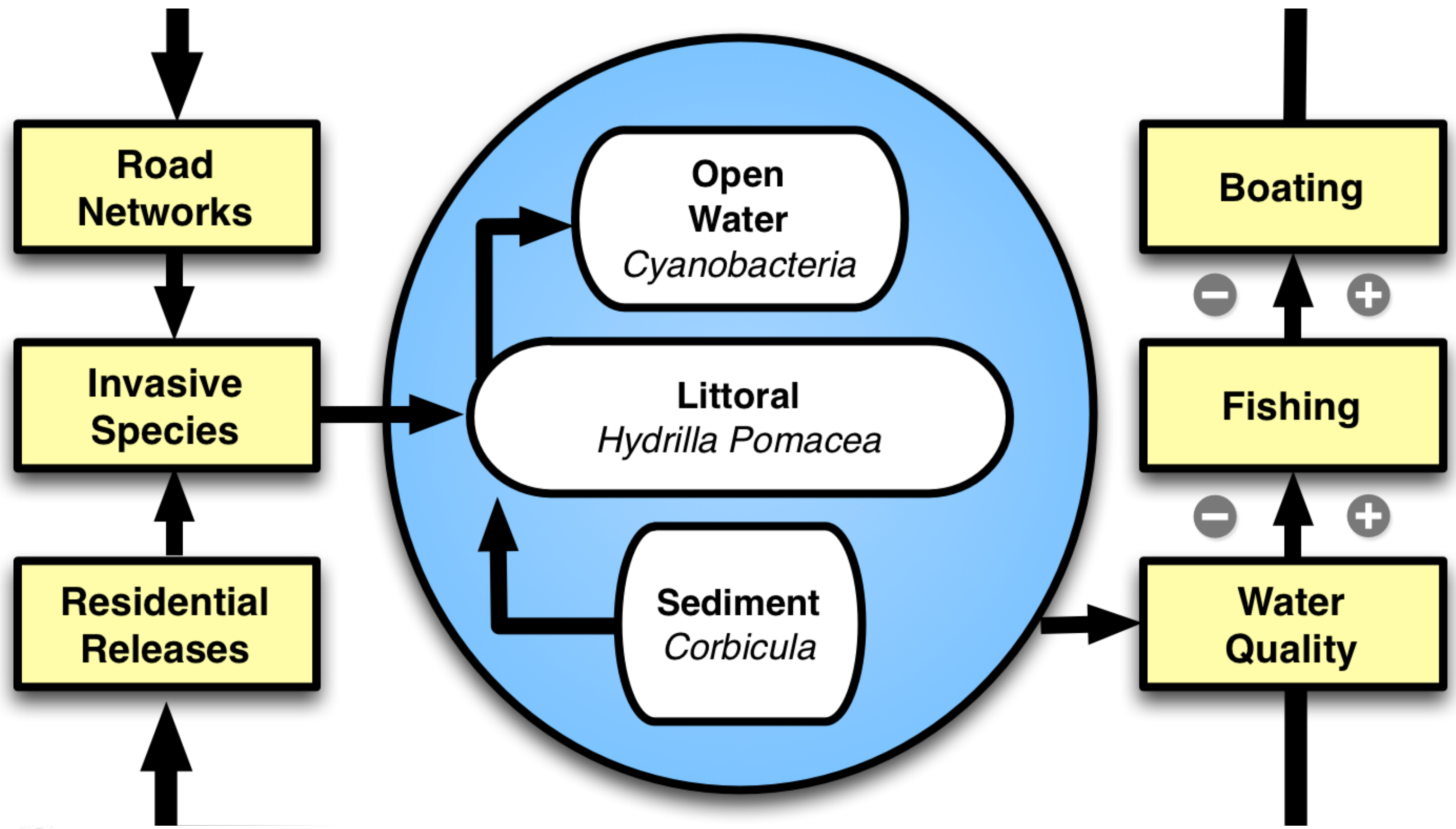
Georgia Institute of Technology

Target Domain: Apple Snails

The University of Georgia

Georgia Institute of Technology

Bigger Picture: Apple Snails.



The University of Georgia
Georgia Institute of Technology