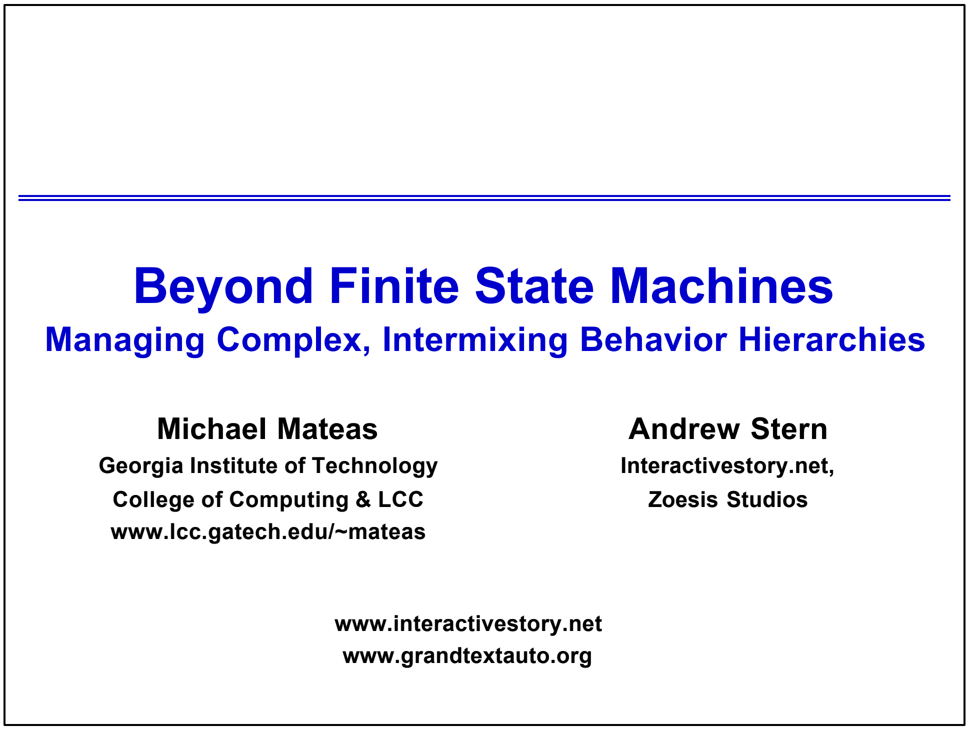
<< RETURN



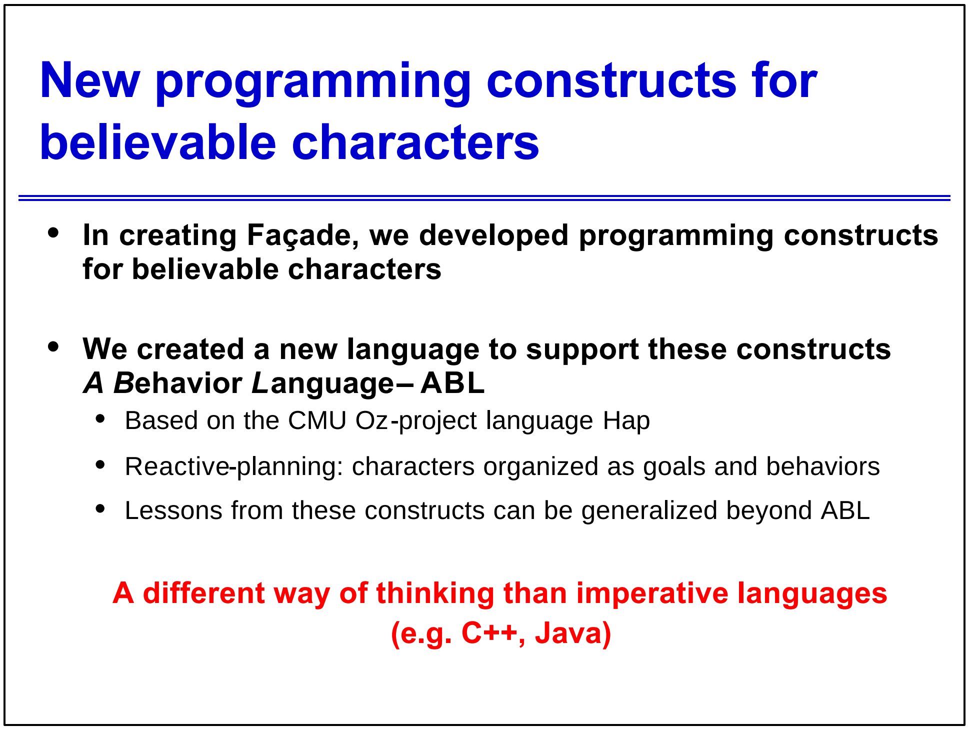
Beyond Finite State Machines

Managing Complex, Intermixing Behavior Hierarchies

Michael Mateas
Georgia Institute of Technology
College of Computing & LCC
www.lcc.gatech.edu/~mateas

Andrew Stern
Interactivestory.net,
Zoesis Studios

www.interactivestory.net
www.grandtextauto.org

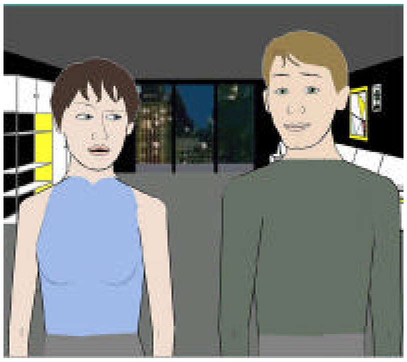


New programming constructs for believable characters

- In creating Façade, we developed programming constructs for believable characters
- We created a new language to support these constructs
  A **B**ehavior *L*anguage– ABL
  - Based on the CMU Oz-project language Hap
  - Reactive-planning: characters organized as goals and behaviors
  - Lessons from these constructs can be generalized beyond ABL

A different way of thinking than imperative languages
(e.g. C++, Java)

Façade



- **Dramatic world inhabited by computer controlled characters (believable agents)**
- **The user (player) plays a protagonist within the story, first-person point of view**
- **The player experiences a story with a dramatic arc**

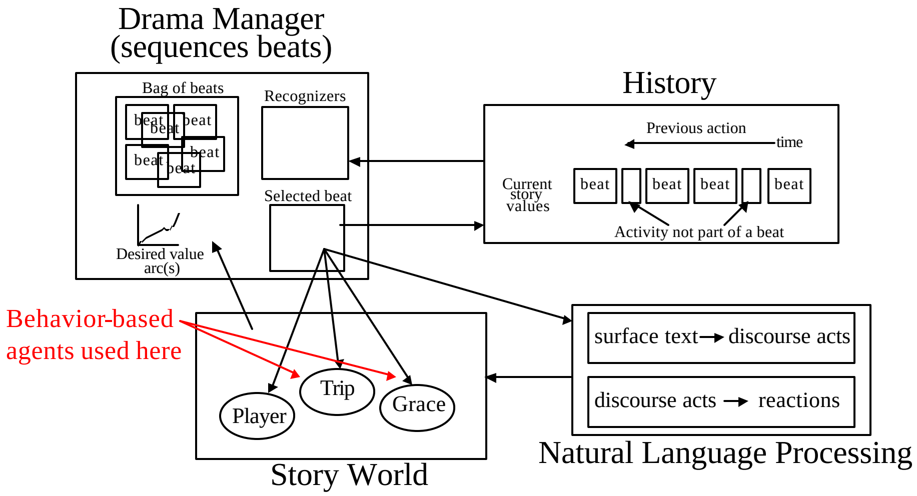
Façade character requirements

- **Moment-by-moment believability**
  Body movements, facial expression, behavior mixing
- **Tightly coordinated action**
  Characters work closely together to perform story
- **Conversational behavior**
  Longer-term, non-linear dialog flow that preserves reactivity

Code support for character requirements

- **Goals and behaviors**
  Sequencing + reactivity, behavior mixing, hierarchy
- **Joint goals and behaviors**
  Protocol supporting multi-character teamwork
- **Meta-behaviors**
  Canonical behavior sequences are modified by player interaction

Façade architecture



Drama Manager
(sequences beats)

Bag of beats

beat beat beat beat beat beat

Recognizers

Selected beat

Desired value arc(s)

History

Previous action

time

Current story values

beat beat beat beat

Activity not part of a beat

Behavior-based agents used here

Player

Trip

Grace

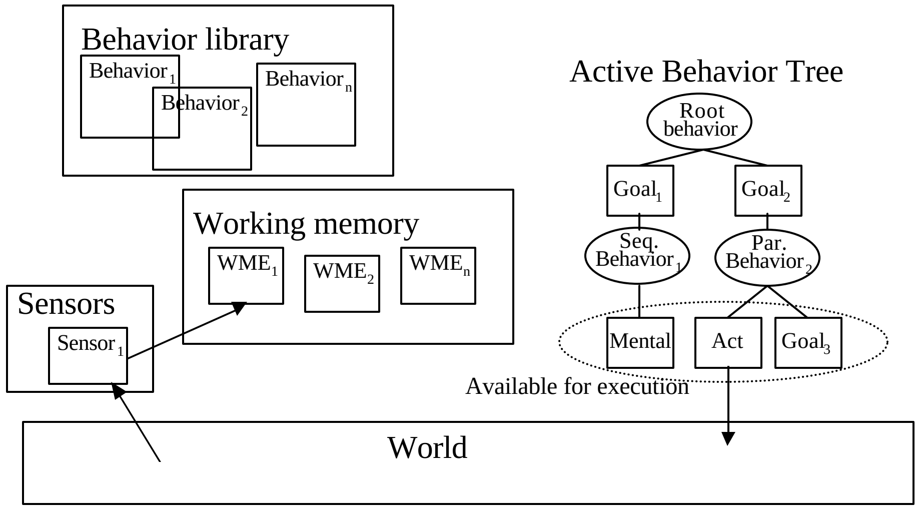
Story World

surface text → discourse acts

discourse acts → reactions

Natural Language Processing

A behavior-based agent



Behavior library

Behavior$_1$

Behavior$_2$

Behavior$_n$

Working memory

WME$_1$

WME$_2$

WME$_n$

Sensors

Sensor$_1$

Active Behavior Tree

Root behavior

Goal$_1$

Goal$_2$

Seq. Behavior$_1$

Par. Behavior$_2$

Mental

Act

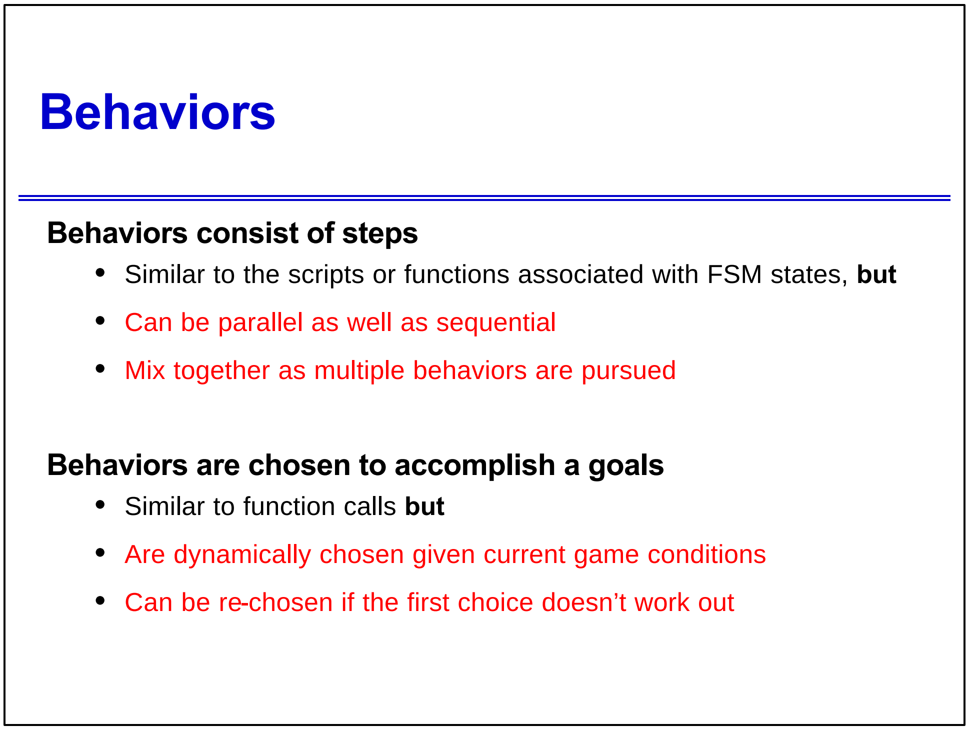
Goal$_3$

Available for execution

World

Features of our behavior-based agents

- Characters organized as goals and sequential & parallel behaviors
- Joint (synchronized) goals and behaivors
- Reflection (meta-behaviors)
- Generalization of sensory-motor connections
- Multiple named working memories
- Atomic behaviors (useful for atomic WM updates)



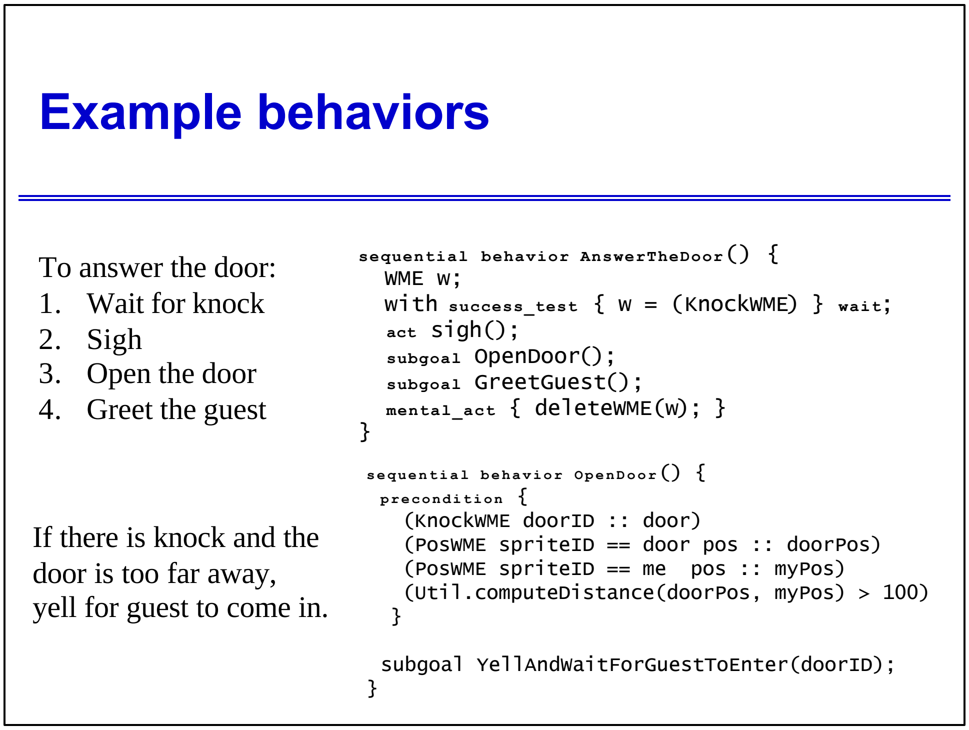
Behaviors

Behaviors consist of steps

- Similar to the scripts or functions associated with FSM states, **but**
- Can be parallel as well as sequential
- Mix together as multiple behaviors are pursued

Behaviors are chosen to accomplish a goals

- Similar to function calls **but**
- Are dynamically chosen given current game conditions
- Can be re-chosen if the first choice doesn't work out



Example behaviors

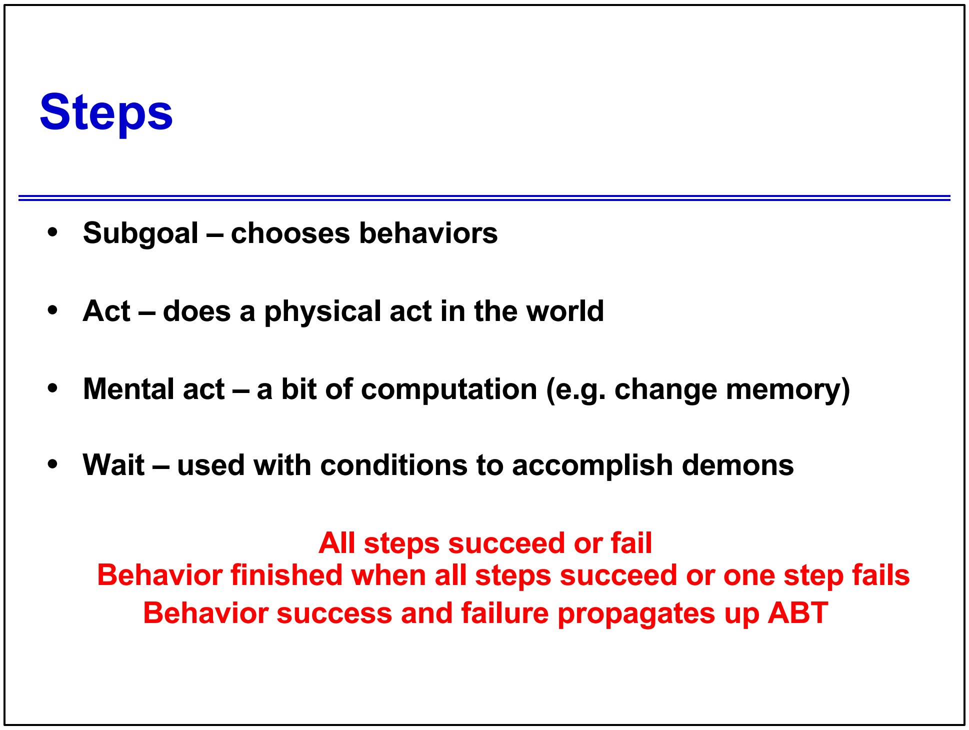
To answer the door:

1. Wait for knock
2. Sigh
3. Open the door
4. Greet the guest

```
sequential behavior AnswerTheDoor() {
  WME w;
  with success_test { w = (KnockWME) } wait;
  act sigh();
  subgoal OpenDoor();
  subgoal GreetGuest();
  mental_act { deleteWME(w); }
}
```

If there is knock and the door is too far away, yell for guest to come in.

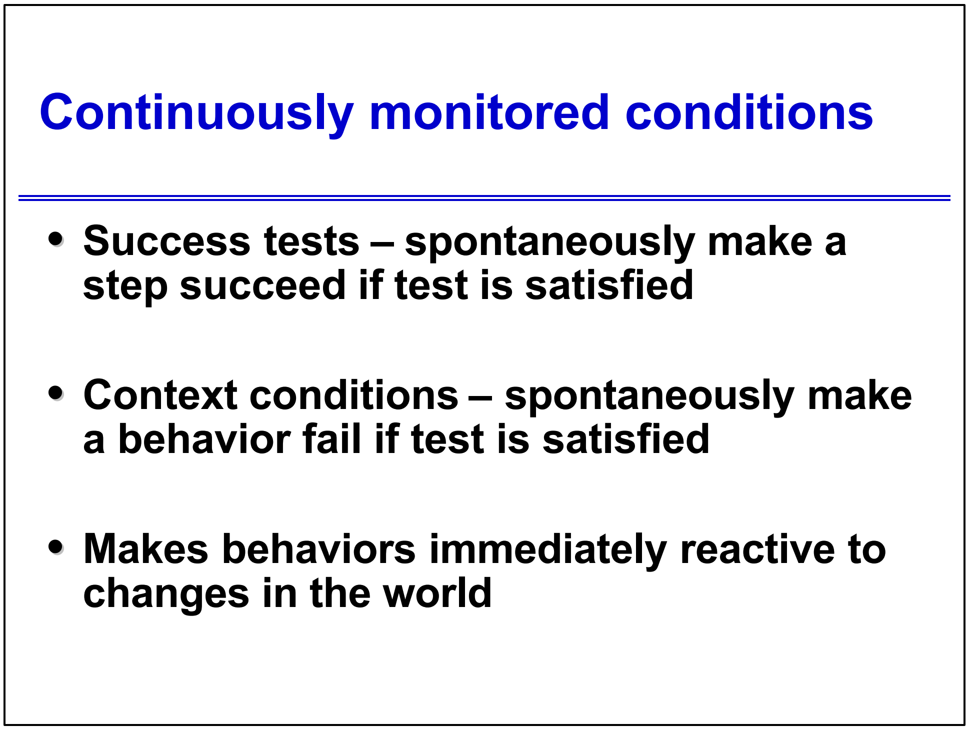
```
sequential behavior OpenDoor() {
 precondition {
   (KnockWME doorID :: door)
   (PosWME spriteID == door pos :: doorPos)
   (PosWME spriteID == me  pos :: myPos)
   (Util.computeDistance(doorPos, myPos) > 100)
  }

 subgoal YellAndWaitForGuestToEnter(doorID);
}
```



Steps

- Subgoal – chooses behaviors
- Act – does a physical act in the world
- Mental act – a bit of computation (e.g. change memory)
- Wait – used with conditions to accomplish demons

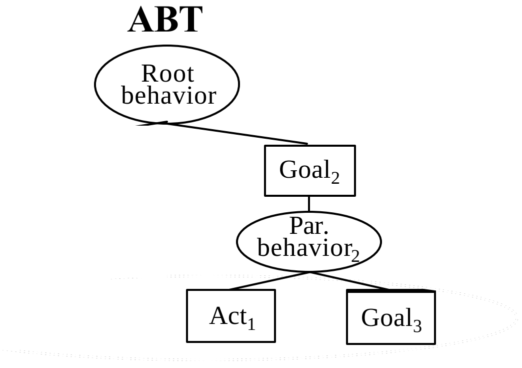
All steps succeed or fail
Behavior finished when all steps succeed or one step fails
Behavior success and failure propagates up ABT



Continuously monitored conditions

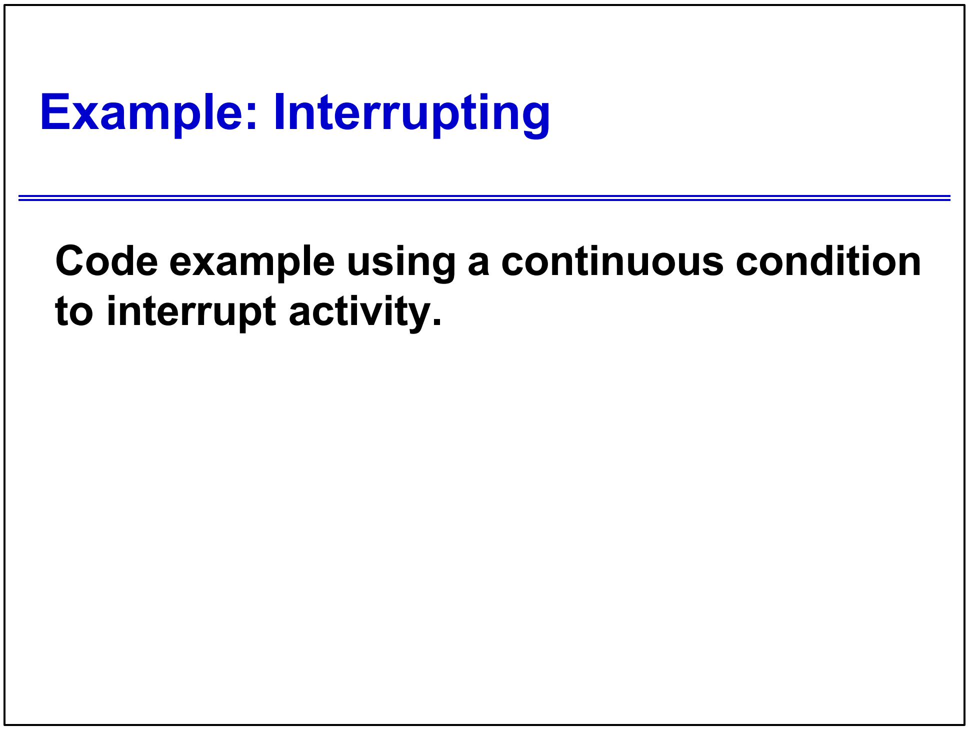
- Success tests – spontaneously make a step succeed if test is satisfied
- Context conditions – spontaneously make a behavior fail if test is satisfied
- Makes behaviors immediately reactive to changes in the world

Success and failure propagation



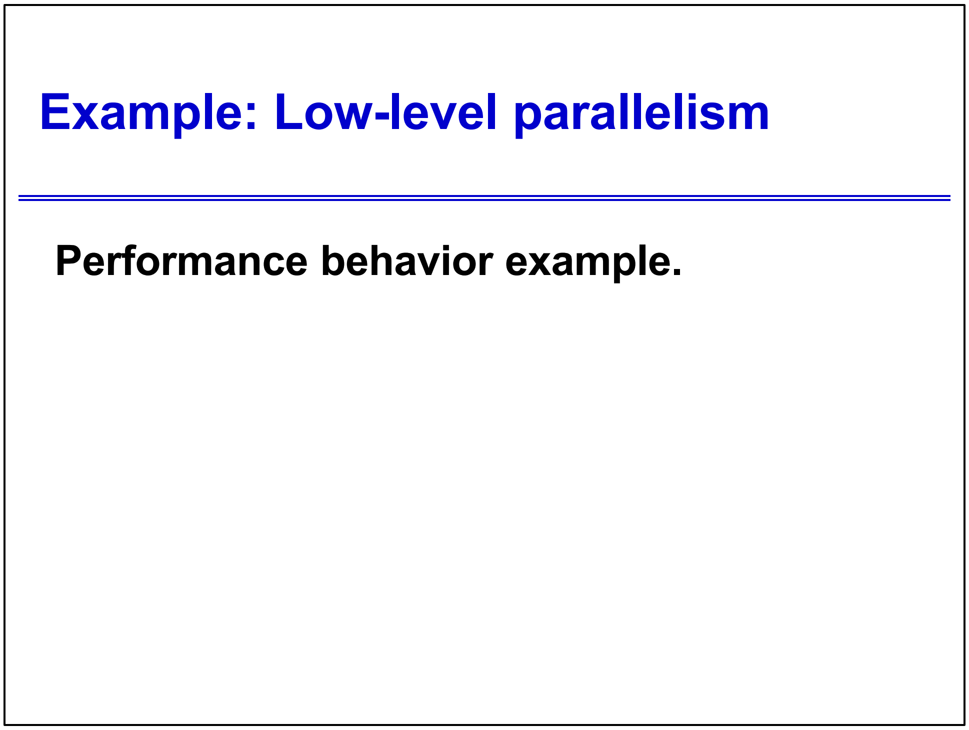
Example: Making a drink

Code example showing basic sequential behavior plus hierarchical subgoaling.



Example: Interrupting

Code example using a continuous condition to interrupt activity.



Example: Low-level parallelism

Performance behavior example.

Example: High-level behavior mixing

Example showing two high level behaviors blending together (making a drink + dialog performance). Demonstrates conflicts and priorities.

Joint goals and behaviors

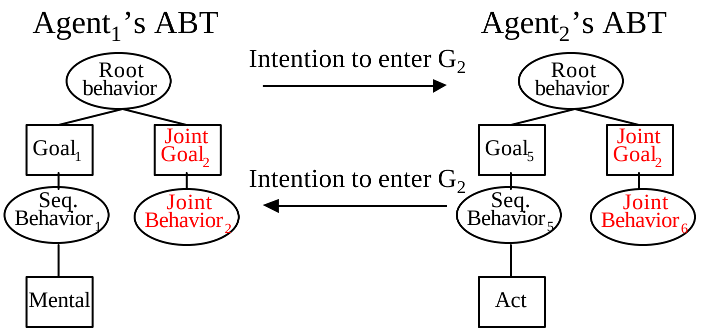
Characters sometimes need to coordinate action

Some approaches

- Coordinate through sensing (but plan recog. hard)
- Explicitly communicate (but ad hoc)
- Build it into architecture (but not flexible)

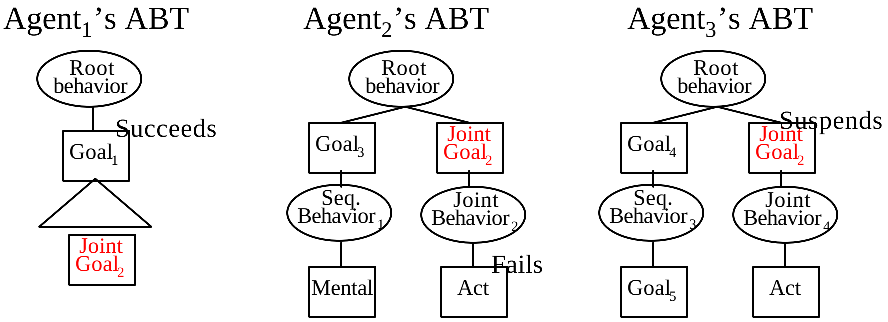
Architecture coordinates author-specified joint action

Negotiation



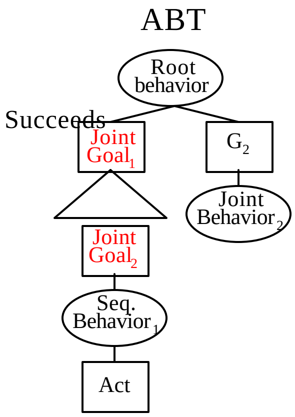
Conflicting intentions

Problem: asynchronous agents enter conflicting states



Resolution: intentions are precedence ordered

Inconsistent subtree execution



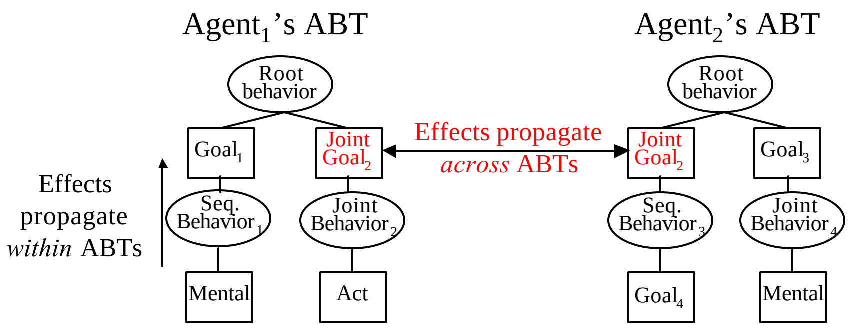
Problem: continuing execution leads to ABT inconsistencies

Resolution: freeze subtree

- Initiate exit intention at the subtree root
- Remove all leaf steps
- Deactivate all monitored conditions
- Negotiate removal of all joint goals
- Commit to exit intention at subtree root

Variably coupled agents

A tunable spectrum between one-mind and many-minds

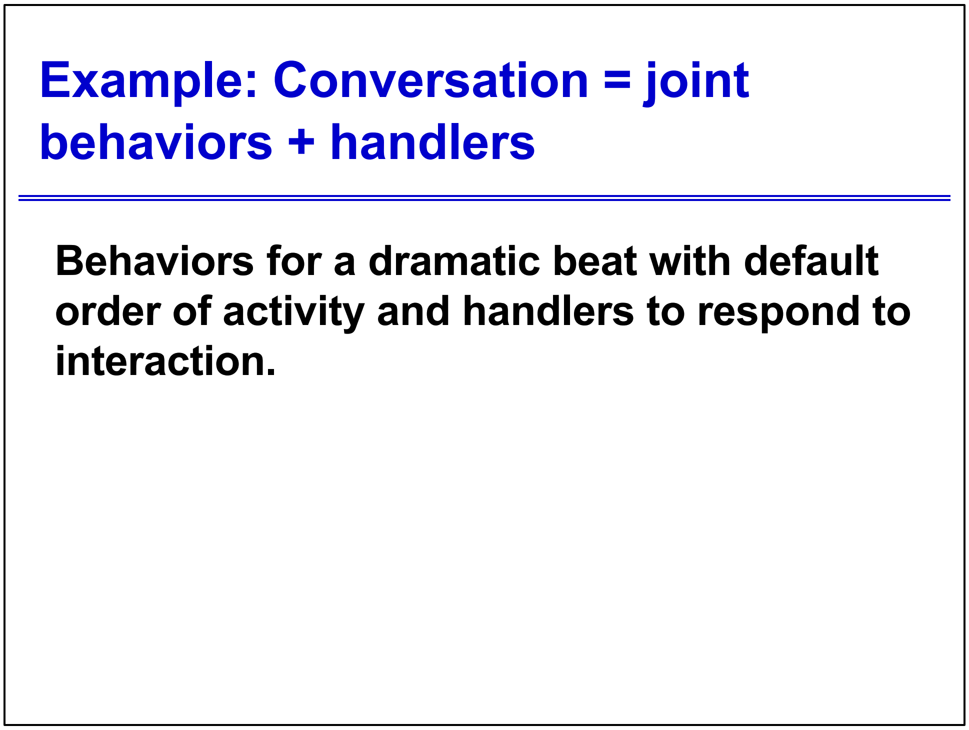


Example: Coordinating dialog

Beat goal showing dialog-line level agent coordination. Difficult to get this level of coordination without negotiation support.

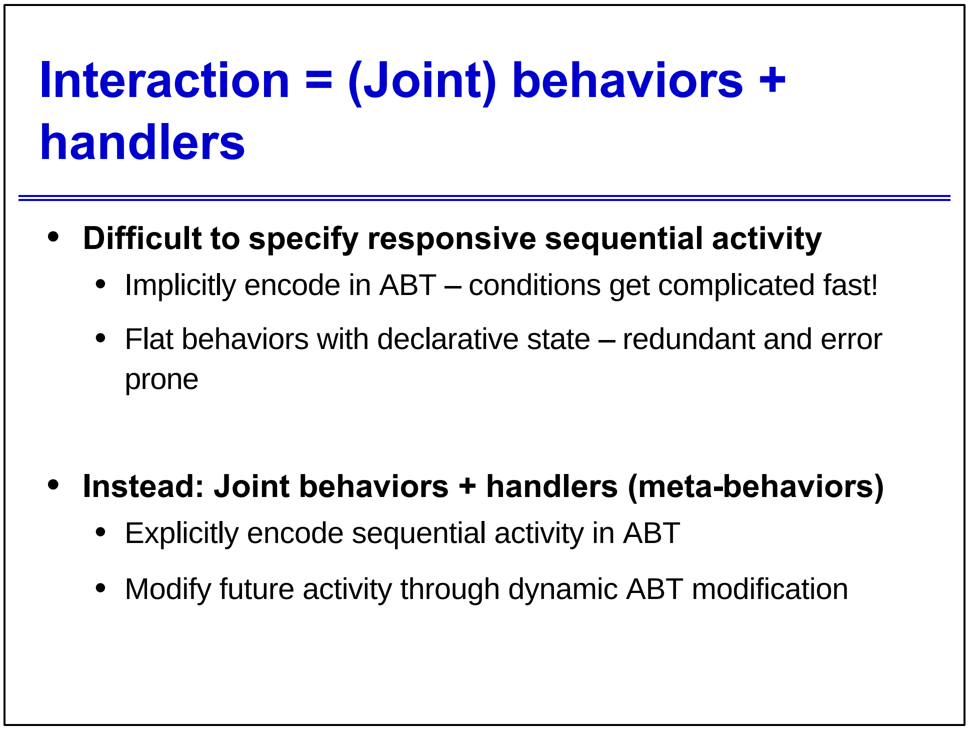
Meta-behaviors

- Meta-behaviors manipulate the runtime state of other behaviors (e.g. succeed or fail steps).
- Ability to match on this runtime state just like it was part of the world (preconditions, context conditions, success tests)



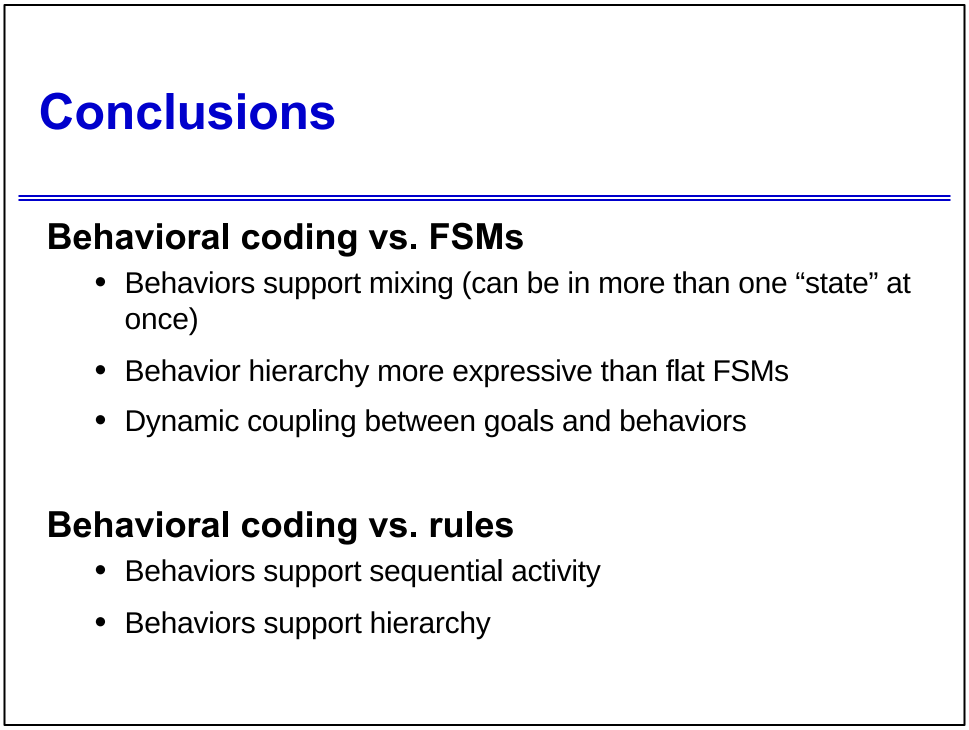
Example: Conversation = joint behaviors + handlers

Behaviors for a dramatic beat with default order of activity and handlers to respond to interaction.



Interaction = (Joint) behaviors + handlers

- **Difficult to specify responsive sequential activity**
  - Implicitly encode in ABT – conditions get complicated fast!
  - Flat behaviors with declarative state – redundant and error prone
- **Instead: Joint behaviors + handlers (meta-behaviors)**
  - Explicitly encode sequential activity in ABT
  - Modify future activity through dynamic ABT modification



Conclusions

Behavioral coding vs. FSMs

- Behaviors support mixing (can be in more than one “state” at once)
- Behavior hierarchy more expressive than flat FSMs
- Dynamic coupling between goals and behaviors

Behavioral coding vs. rules

- Behaviors support sequential activity
- Behaviors support hierarchy

For more info

www.interactivestory.net
Façade project site (includes latest slides)

www.grandtextauto.org
Group blog on games and new media

egl.gatech.edu
Experimental Game Lab (includes projects using ABL)