Tekkotsu Behaviors & Events

15-494 Cognitive Robotics
David S. Touretzky & Ethan Tira-Thompson

Carnegie Mellon
Spring 2008
Behaviors

- Behaviors are classes defined in .h files
  - Add them to the ControllergGUI “Mode Switch” menu by calling `addItem` in `StartupBehavior_SetupModeSwitch.cc`
  - Double click on the “Mode Switch” menu item to instantiate
  - Instantiated by `BehaviorSwitchControl()` when requested by the ControllerGUI
  - When you stop a behavior (double click on the menu item), the instance is deleted
Five Behavior Components

#include "Behaviors/BehaviorBase.h"

class PoodleBehavior : public BehaviorBase {

  • Constructor
    
    PoodleBehavior() : BehaviorBase("PoodleBehavior") {}

  • DoStart() activates the behavior
    
    virtual void DoStart() {
      BehaviorBase::DoStart();
      cout << getName() << " is starting up." << endl;
    }


Five Behavior Components

- **DoStop()** deactivates the behavior

  ```
  virtual void DoStop() {
    cout << getName() << " is shutting down." << endl;
    BehaviorBase::DoStop();
  }
  ```

- **processEvent** processes requested event types

  ```
  virtual void processEvent(const EventBase &event) {
    cout << getName() << " got event: "
        << event.getDescription() << endl;
  }
  ```
Five Behavior Components

- `getClassDescription()` returns a string displayed by ControllerGUI pop-up help

```cpp
virtual std::string getClassDescription() {
    return "Demonstration of a simple behavior";
}
```

}; // end of PoodleBehavior class definition
Behaviors are Coroutines

• Behaviors are coroutines, not threads:
  – Many can be “active” at once, but...
  – Only one is actually running at a time.
  – No worries about mutual exclusion.
  – Must voluntarily relinquish control so that other active behaviors can run.

• BehaviorBase is a subclass of:
  – EventListener
  – ReferenceCounter

• Behaviors will be deleted if they are deactivated and the reference count goes to zero.
Tekkotsu Releases

- Tekkotsu.org holds the current *stable release* and accompanying documentation.

- tekkotsu.no-ip.org holds the latest *(bleeding edge)* version of Tekkotsu, and the latest version of the documentation.

- This class will be using bleeding edge software.

- The “Reference” link on the course home page points to the bleeding edge documentation.
Browsing the Documentation

• “Class List” in left nav bar
  - Click on class name (BehaviorBase) to see documentation page
  - Click on method name (DoStart) to jump to detailed description
  - Click on line number to go to source code

• “Directories” in left nav bar shows major components
  - Look at the Behaviors and Events directories
Searching the Source

- Use the search box in the documentation pages to search for any identifier.
  - Example: NearIRDistOffset

- Use the “ss” shell script to grep the source code:
  
  > cd ~/Tekkotsu
  
  > ss NearIR
Events

- Events are subclasses of `EventBase`
- Three essential components:
  1. Generator ID: what kind of event is this?
     - `buttonEGID`, `visionEGID`, `timerEGID`, ...
  2. Source ID: which sensor/actuator/behavior/thing generated it?
     - `RobotInfo::HeadButOffset`
  3. Type ID, which must be one of:
     - `activateETID`
     - `statusETID`
     - `deactivateETID`
Where are these Defined?

- EventGeneratorID_t defined in EventBase.h
  
- EventTypeID_t defined in EventBase.h

```c
enum EventTypeID_t {
    activateETID,
    statusETID,
    deactivateETID,
    numETIDs
};
```

- Event source ids are specific to the event type:
  - HeadButOffset defined in ERS7Info.h
  - visPinkBallSID defined in ProjectInterface.h
Subscribing to Events

`addListener(listener, generator, source, type)`

```cpp
#include "EventRouter.h"

virtual void DoStart() {
    BehaviorBase::DoStart();
    erouter->addListener(this, 
    EventBase::buttonEGID, 
    RobotInfo::LFrPawOffset, 
    EventBase::activateETID);
}
```
virtual void processEvent(const EventBase &event) {
    switch (event.getGeneratorID()) {

        case EventBase::buttonEGID:
            cout << "Button press: " << event.getDescription() << endl;
            break;

        default:
            cout << "Unexpected event: " << event.getDescription() << endl;
            break;

    }
}
Types of Events

- What are some subclasses of EventBase?
Vision Object Events

- VisionObjectEvent is a subclass of EventBase

- The vision pipeline includes an “object detector” that looks for pink blobs, like the AIBO's ball.

- The center and area of the largest blob are reported by posting a VisionObjectEvent (if anyone's listening.)
  - visObjEGID
  - visPinkBallSID
  - activate, status, deactivate ETIDs
The Event Router

- Runs in the Main process.
- Distributes events to the Behaviors listening for them.
Subscribing to Vision Events

#include "Events/VisionObjectEvent.h"
#include "Shared/ProjectInterface.h"

virtual void DoStart() {
    BehaviorBase::DoStart();
    erouter->addListener(this,
        EventBase::visObjEGID,
        ProjectInterface::visPinkBallSID);
}
void processEvent(const EventBase &event) {
    switch (event.getGeneratorID()) {

    case EventBase::visObjEGID: {
        const VisionObjectEvent &visev =
            dynamic_cast<const VisionObjectEvent&>(event);

        if (visev.getTypeID() == EventBase::activateETID ||
            visev.getTypeID() == EventBase::statusETID)
            cout << "Saw pink ball at (" << visev.getCenterX() << ", " << visev.getCenterY() << ")" << endl;
        else // deactivate event
            cout << "Lost sight of the ball!" << endl;
    }
    break;

    case EventBase::buttonEGID: ...
    }
Text Message Events

You can send text messages to the AIBO via the ControllerGUI's “Send Input” window:

!msg Hi there

This causes the behavior controller to post a textmsgEvent.
Subscribing to TextMsg Events

#include “Events/TextMsgEvent.h”

virtual void DoStart() {
    BehaviorBase::DoStart();
    erouter->addListener(this, EventBase::textmsgEGID);
}

The source ID is meaningless (it's -1).

The type ID is always statusETID.
void processEvent(const EventBase &event) {
    switch (event.getGeneratorID()) {

    case EventBase::textmsgEGID:
        const TextMsgEvent &txtev =
            dynamic_cast<const TextMsgEvent&>(event);
        cout << "I heard: '" << txtev.getText() << "'" << endl;
        break;

    case EventBase::buttonEGID:
        ...
    }
}
The Event Logger

- Root Control
  - Status Reports
  - Event Logger

- Outputs to console: telnet to port 59000 to see the log
Timers

Timers are good for two kinds of things:

• Repetitive actions: “Bark every 30 seconds.”
  – Whenever a timer expires and a timer expiration event is posted, the timer should be automatically restarted.

• Timeouts: “If you haven't seen the ball for 5 seconds, bark and turn around.”
  – One-shot timer. Will need to be cancelled if we see the ball before the time expires.
**addTimer**

- **addTimer**(listener, source, duration, repeat)
  - listener is normally this
  - source is an arbitrary integer
  - duration is in milliseconds
  - repeat should be “true” if a sequence of timer events is desired
- Starts timer and automatically listens for the event.
- Timers are specific to a behavior instance; can use the same source id in other behaviors without interference.
- Behaviors can receive another's timer events if they use addListener to explicitly listen for them.
- **removeTimer**(listener, source)
#include "Behaviors/BehaviorBase.h"
#include "EventRouter.h"

virtual void DoStart() {
    BehaviorBase::DoStart();

    erouter->addListener(this,
        EventBase::buttonEGID,
        RobotInfo::LFrPawOffset,
        EventBase::activateETID);

    erouter->addListener(this,
        EventBase::buttonEGID,
        RobotInfo::RFrPawOffset,
        EventBase::activateETID);
}
Timer Example

virtual void processEvent(const EventBase &even) {
    switch (event.getGeneratorID()) {

        case EventBase::buttonEGID:
            if (event.getSourceID() == RobotInfo::LFrPawOffset)
                erouter->addTimer(this, 1234, 5000, false);
            else if (event.getSourceID() == RobotInfo::RFrPawOffset)
                erouter->removeTimer(this, 1234);
            break;

        case EventBase::timerEGID:
            cout << "Timer expired!" << endl;
    }
}

What does this behavior do?
The Tekkotsu Simulator

- Really useful for debugging vision code, but...

  > cd ~/project  
  (or whatever your project name is)
  > make sim
  > ./tekkotsu-ERS7

- In another terminal tab:
  > ControllerGUI localhost
ControllerGUI Can Post Events to the Simulator

- Type this command in the “Send Input” box:

```plaintext
!post buttonEGID LFrPaw A
```

- Monitor the result using the Event Logger
Tekkotsu Architecture: Main

- **Main Process**
  - Vision Pipeline
  - erouter
  - System sends state information (via Motion, ~32ms)
  - System sends camera frames (~30fps)
  - Behaviors can play sounds anytime

- **WorldState**
  - Can access state anytime for reactive/open loop control
  - Created by currently active Behaviors

- **MotionManager**
  - Registered with Motion Manager
  - Requests joint positions
  - Returns positions based on current MotionCommands

- **Motion Process**
  - Requests joint positions (~32ms)
  - Sends new joint positions to system
  - Requests sound buffer (~32ms)
  - Returns 32 ms of sound to system

- **Sound Manager**
  - Returns sound buffer by mixing current sounds

- **MotionCommands** (dynamically created)
  - Can play sounds at any time

- **Sound Process**
  - Aibo-only, allows you to FTP files during run time. Other platforms use their own FTP server.

**Key**
- Pre-emptive Process
- Shared Memory Region
- Unshared Global Variable
World State

• Shared memory structure between Main and Motion
• Updated every 32 msec
• sensorEGID events announce each update
• Contents:
  - joint positions, duty cycles, and PID settings
  - button states: state->buttons[LFrPawOffset]
  - IR range readings: state->sensors[NearIRDistOffset]
  - accelerometer readings
  - battery state, thermal sensor
  - commanded walking velocity (x,y,a)
Sensor Observer

• Root Control
  > Status Reports
  > Sensor Observer

• Try monitoring the accelerometers.

• Then pick up the robot and wave it around.
Control of Effectors

• How do we make the robot move?

• Must send commands to each device (head, leg, and tail joints, ear servos, LED display, etc.) every 32 ms.

• This is real-time programming.

• Can't spend too long computing command values!

• Best to do all this in another process, independent of user-written behaviors, so motion can be smooth.