Executive Layer: Tasks and TDL

Brad Hamner
Robotics Institute – Carnegie Mellon University
April 8, 2008
About the Executive Layer

- Concerned with accomplishing tasks
- Breaking tasks into subtasks
- Sequencing tasks to achieve larger goals
- Recovering if a task fails
  - By launching more tasks
About the Executive Layer

- The base unit of the executive layer is...
  - The task!

- Tasks are written in TDL
  - Task Description Language
TDL

- Written on top of C++
  - Like C++, but with extra keywords and syntax
- Allows
  - Tasks to be spawned on any agent
  - Sequencing of tasks
    - A starts after B ends
    - A starts 10 seconds after B ends
    - A starts after B starts
    - A starts after B starts and B ends when A ends
    - A and B end together
    - ...
    - ...
Task Example

- src/Scenarios/Ace/LeaderAgent/Feb2008AceDemoTask.tdl

- What you should learn here:
  - Task header
  - spawn
  - with
  - serial
  - On: Spawn a task on another agent
  - Naming a task in the tree
More on Task Sequencing

- See the manual
- http://www.cs.cmu.edu/~tdl/tdl.html
Goal or Command?

- Goal and Command keywords essentially mean the same thing.
- Use Goal when task spawns other tasks.
- Use Command otherwise.
Distributed Keyword

- Put Distributed before task header to make it possible to spawn this task from another agent
On `agentname`, you know

- But the `agentname` isn't really `agent name`
- Agent has executive and behavioral layer
- Need to tell TDL to use executive layer
- So really it's On `agentname:exec`
Frequently, you want to make something happen, then wait for results

**Use** POSTPONE

Then you make “resume clauses”
- Functions that say what happens next
Resuming from Handlers

- Task is postponed
  - Waiting for results or error notification
  - Notification comes from events or IPC messages
- Need handlers to receive these
- Handlers will call a resume clause
See IPC slides for basics on handlers

For now, see:

```
src/Scenarios/Ace/LeaderAgent/RoughBaseMoveTask.tdl
```

- Spawns PoseEstimationTask and postpones
- Handler catches pose result
  - Resumes Task
- Resume clause spawns base move
Handler needs to spawn a specific task
  ▪ Could be more than one of a task in the tree
  ▪ Therefore, you need task reference

In main task clause, create reference to self
  ▪ Pass self reference to handler
  ▪ Handler uses reference to resume right task
Note on Resuming

- Handler calls resume clause with task reference
- Resume clause’s header doesn’t have task reference
- TDL does some automatic compiling there
Another Note on Resuming

- Have to check the task didn’t disappear while waiting for event or message
  - It’s that big if statement in example handlers
  - Just copy that when you write a handler
- Task’s parameters are persistent between main function and resume clauses
- Anything else you want is declared PERSISTENT

- Task reference needs to be accessed by handler
  - So it’s persistent
Subscribing to Block Events

- Create an event reference
  - SkmInterface::EventRef<
    BlockNameBlockStub>
  eventRefName

- Subscribe to the event
  - eventRefName = skm->subscribeEvent
    <
    BlockNameBlockStub>("BlockName");

- Connect the event to a handler
  - eventRefName().connect(bind(slot(&
    eventCallbackName),
    taskSelfRefName));
Subscribing to Block Events

- Event callbacks similar to IPC handlers
  - See src/Components/MobileManipulator/GoToGoalTask.tdl

- Make the event ref persistent if you want to unsubscribe later
  - skm->unsubscribeEvent(eventRefName);
Enabling and Disabling Blocks

Here's an example

```cpp
Distributed Goal GoToGoalTaskAction(GoToGoalTaskParams params)
ON TERMINATE SPAWN GoToGoalTaskActionCleanup(TIL_REF(THIS))
{
    int bcp = MONITOR_STATUS;
    // Hmm. Debugging.
    cout << "GoToGoal;Action: Setting up base control block..." << endl;
    skm->changeParams<BaseControlBlockStub>("baseControl", &bcp);
    skm->enable<BaseControlBlockStub>("baseControl", &bcp);
    
    cout << "GoToGoal;Action: Enabling goToGoal block..." << endl;
    GOALPOSE_TYPE goal;
    params.fillGoalPose(&goal);
    skm->enable<GoToGoalBlockStub>("goToGoal", &goal);
    cout << "GoToGoal;Action: Postponing." << endl;
    POSTPONE;
}

// Fired when the auto task is terminated (on monitor completion or switching)
Command GoToGoalTaskActionCleanup(TCM_Task_Tree_Ref taskRef)
{
    MARKUSED(taskRef);
    #define _disable disable
    skm->_disable("goToGoal");
    // baseDead is always enabled
    #undef _disable
}
Local vs. Remote

- TDL creates different header files for tasks to be spawned on same agent (locally) vs. tasks to be spawned on different agent (remotely)
- Creates task.tdl.loc.H and task.tdl.rem.H
- Be sure to include the right one when you need to spawn that task
I haven’t used these yet, but I’ll explain what I know.
Defining Exceptions

- Somewhere you declare a type of exception
- Use the `Exception` keyword and give it a name
  - Also give it a data type
    - `Exception exceptionName (DataType dataName);`

- See
  `src/Components/Tasks/VisualServo/VisualServoWaypointTask.tdl`
Failure

- Then throw the exception
  - `FAIL exceptionName (data);`
  - This keyword ends the task as well
Recovery

- Somewhere up the task tree
  - I.e. A task that spawned the failing task or its parent
- Create an exception handler
Exception Handlers

- **See** src/Components/Tasks/VisualServo/VisualServoTask.tdl
- **Declare the handler in the task header**

```c
Goal VisualServoTaskAction(VisualServoTaskParams params)
PERSISTENT TCM_Task_Tree_Ref* selfRef,
PERSISTENT TaskSwitch cbaTaskSwitch,
PERSISTENT VisualServoTaskActionInternalData data,
EXCEPTION HANDLER VisualServoTaskActionGenericErrorHandler(TDL_REF(THIS)),
EXCEPTION HANDLER VisualServoTaskActionBaseDeadHandler(TDL_REF(THIS)),
EXCEPTION HANDLER VisualServoTaskActionWaypointFailureHandler(TDL_REF(THIS))
{
    MARKUSED(params);

    // Subscribe to incoming switch requests.
    selfRef = new TCM_Task_Tree_Ref();
    *selfRef = TDL_REF(THIS);
```
Exception Handlers

- Then in the handler header say what it handles

```c++
// Handle generic waypoint errors - for now, the recovery is the same for all of them
ExceptionHandler_VisualServoTaskActionWaypointFailureHandler(TCM_Task_Tree_Ref taskRef)
    HANDLES VisualServoWaypointFailure
{
    // Start from the beginning again
    if(taskRef.notNull() && !TDL_TASK_IS_COMPLETED(taskRef)) {
        cout << "VisualServoTaskActionWaypointFailureHandler: Failed. Doh. Starting from scratch; resuming VSHTA." << endl;
        TDL_RESUME_VisualServoTaskAction(taskRef, VisualServoWaypointFailure, waypointNotAchieved);
    } else {
        cout << "VisualServoTaskActionWaypointFailureHandler: Task is gone, not doing anything." << endl;
    }
}
```

- Resuming the task is usually a good thing to do in the handler

Remember the exception data?
If parent task doesn’t handle exception, system searches its parent task, etc., until it finds one that does

If no task handles exception, everything goes boom
  ▪ (I think)
Action/Monitor

- Common model is action task paired with monitor task
- Action task tries to actually complete the task
- Monitor task checks status info and makes sure task completes