SmallADE Documentation

Tom Williams

October 12, 2014

1 What is smallADE?

The Agent Development Architecture (ADE) is an implementation of the DIARC Cognitive Architecture. ADE consists, mainly, of a set of components that provide useful functionality. The LRFComponent provides an interface to a Laser Range Finder. The CreateComponent provides an interface to the iRobot Create. An instance of ADE consists of a set of components whose connections and status are managed by one or more Registries.

This document details how to use a subset of ADE referred to as "smallADE". smallADE includes the following components:

- LRFComponent: a component for interfacing with a Laser Range Finder
- CreateComponent: a component for interfacing with the iRobot Create
- ADESim: the ADE Simulator
- KeyboardJoystick: A controller for driving a robot with the computer keyboard

The code for these components can be found by navigating to [ade]/com. In your particular smallADE directory there may be some other components lurking around. Ignore them. Those are not the droids you’re looking for.

2 ADE Quickstart

In this section, I’ll detail how to get some simple ADE setups up and running.

2.1 Compiling

./ant main

2.2 Running the Simulator

Let’s try to get a simple simulator setup running.

First, run the registry: ./ant run-registry
Next, start the simulator: ./ant run-adesimenvironmentcomponent
Finally, add a simulated robot: ./ant run-simpioneercomponent

To drive the simulated robot around the simulated environment, use the arrow keys.
2.3 Running the Robot

To run the robot, the process is similar:

First, run the registry: `./ant run-registry`

Next, start the create: `./ant run-createcomponent -Dargs="-port /dev/ttyACM0"

Then, start the LRF Component: `./ant run-urglrfcomponent -Dargs="-port /dev/ttyAMA0 -g"

Finally, start the component: `./ant run-keyboardjoystickcomponent`

(This all *should* work correctly, but I may need to do a wee bit of testing to verify.)

2.4 Running Distributed smallADE

Now let’s look at running parts of the system on the robot and parts on another computer. Here’s how we do that:

On the remote computer:

Run the registry: `./ant run-registry -Dk=config/hosts/matt.hosts -Dl=[your ip]`

On the robot:

Start the create: `./ant run-createcomponent -Dargs="-port /dev/ttyACM0" -Dr=[your ip] -Dl=[the robot’s ip]`

Start the LRF Component: `./ant run-urglrfcomponent -Dargs="-port /dev/ttyAMA0 -g -Dr=[your ip] -Dl=[the robot’s ip]"

3 Writing an ADE Component

In order to actually interact with the create and lrf, you’ll want to write your own ADE Component.

3.1 Writing the Component

Let’s say you want to write a new component, JimsEKFComponent. To do so, you’ll want to do three things:

1. Create a new folder in /com to house your component (e.g., /com/ekf
2. Create your component interface (JimsEKFComponent.java)
3. Create your component implementation (JimsEKFComponentImpl.java)

The interface file should contain something like:

```java
package com.ekf;

import ade.ADEComponent;
import java.rmi.RemoteException;
```
public interface JimsEKFComponent extends ADEComponent {
    public void doThatEKFThang() throws RemoteException;
}

Note that each function you expose in your component needs to throw RemoteException, since ADE uses RMI.

The implementation file should contain something like:

package com.ekf;

import ade.Connection;
import ade.SuperADEComponentImpl;
import java.rmi.RemoteException;
import com.vslam.JNI.polar_laser_scan;

public interface JimsEKFComponentImpl extends SuperADEComponentImpl implements JimsEKFComponent {
    Connection lrfConn;
    boolean doneConstructing;

    public JimsEKFComponentImpl() throws RemoteException {
        super();
        lrfConn = connectToComponent("com.lrf.LRFComponent");
        doneConstructing = true;
        System.out.println("Ready to go.");
    }

    @Override
    protected void init() {
        lrfConn = null;
        doneConstructing = false;
    }

    @Override
    protected boolean localServicesReady() {
        return doneConstructing && requiredConnectionsPresent();
    }

    @Override
    protected void readyUpdate() {
        polar_laser_scan ps = (polar_laser_scan)hwConn.call("getPolarScanData");
        System.out.println("Scan ID "+ps.scanID+" received.");
    }

    @Override
    protected String additionalUsageInfo() {
        return "The component has no arguments.";
    }

    @Override
    public void doThatEKFThang() throws RemoteException {
        System.out.println("Doin’ that EKF Thang");
    }
}
Important things to note in here:

1. init() is required, and is called before any code in the constructor is called, and after additional arguments (if any) are parsed. This is where you should set the default values of any class variables, unless you want terrible things to happen.

2. readyUpdate is called every update cycle once the component is ready (i.e., once the criteria in localServicesReady are met), which is by default once every 100 ms. This time can be changed by calling this.setUpdateLoopTime(this, SOME_TIME_IN_MS)

3. To call a method advertised by another component, you must have a connection to that component, and you can then do THATCONNECTION.call("METHODNAME", TYPERETURNEDBYPYMETHOD, ANYARGSTOSTOPASSTOMETHOD, making sure to cast the returned object to the appropriate type.

3.2 Building the Component

Examine build targets like create in build.xml for examples for creating ant build targets for building your component. Perhaps yours will look something like this:

```xml
<target name='ekf' depends='init, core' extensionOf='java-only' description='Builds Jims terrible, terrible ekf.'>
  <build-java package.dir='com/ekf'/>
</target>
```

Examine run targets like run-createcomponent in build.xml for examples for creating ant run targets for running your component. Perhaps yours will look something like this:

```xml
<target name='run-ekf' depends='ekf' description='Runs Jims terrible, terrible ekf.'>
  <run-component component='com.ekf.JimsEKFComponent'/>
</target>
```