Project Description

In this project, small teams of students will utilize two adaptation approaches to manage/adapt a run-time robotics system’s architecture. This architecture is similar to that of the system you developed in HW3.

The first approach, introduced in [1], relies on the Layered Adaptation Style. This style differs from the conventional Layered Style: in the Layered Style, components in higher layers use the services and interfaces provided by components in the lower layers; on the other hand, in the Layered Adaptation Style, components in higher layers monitor, analyze, and adapt the components and architectures in lower layers [1]. Figure 1 depicts a simple two-layer adaptation architecture. The Collector meta-component in this architecture monitors and collects data about the components and architecture in the layer below. The Analyzer interprets the data produced by Collectors to evaluate some adaptation policies and creates adaptation plans. The Admin controls and manipulates components at the layer below according to the adaptation plans produced by Analyzer.

The second approach you are required to implement in this project is the Policy-Based Approach to Architectural Adaptation Management (PBAAM), introduced in [2,3]. Figure 2 shows three core parts and three core components of PBAAM. The three fundamental parts of PBAAM are as follows: An architectural model specifying the system’s structure, a set of adaptation policies capturing how the structure changes, and executable units of code.
corresponding to each architectural element. These three parts are managed at runtime by elements of the PBAAM infrastructure: the Architectural Model Manager (AMM), the Architectural Adaptation Manager (AAM), and the Architecture Runtime Manager (ARM) respectively [3]. AMM receives monitoring observations from the run-time system and when applicable sends adaptation responses to the AMM unit. AMM adapts the architecture model of the system and sends model change notifications to the ARM unit. Finally, ARM uses these notifications to adapt the run-time architecture of the system. As a result, the model and the run-time implementation of the system in PBAAM are always synchronized.

![Figure 2: PBAAM Core Elements [3]](image)

This project involves the following steps:

1) Design and implement a C2-style robotics application in which a leader and several follower robots form a convoy. Unlike in HW3, each follower robot ONLY uses its camera to follow the robot in front of it and should reform the convoy in the case of robot failures. You need to assign a separate architecture for each robot in your application. You need to design the C2SADDEL models first and generate the Prism code using DRADEL. You are free to reuse your architectures from HW3; however, you need to design separate component(s) for camera following so that you can proceed to the next steps smoothly. Addendum 1 describes how applications with multiple architectures are modeled in C2SADDEL. It also describes how to implement such an application using DRADEL.

2) Implement a two-layer adaptation architecture as described in [1] to adapt the architecture you designed in step 1. To do this step, you also need to design the required component(s)
to perform the IR following method for the follower robots. You should provide TWO sets of policies for your adaptation architecture, one to perform each of the following tasks:

a. Adapt the architecture you developed in step 1 for each robot (at run-time) to include the IR following related components. In this step, the meta-layer in your layered architecture adds the IR following components to the application layer as soon as they are available. The resulting architecture should be able to switch to IR following when the Camera following fails without any further architectural adaptations.

b. Adapt the architecture you developed in step 1 for each robot (at run-time) ONLY when their camera following method fails. In other words, when the camera following fails in a robot, the camera following related components are replaced with the IR following components as a result of adaptation. This step differs from the previous step in two ways: (1) in step a, architecture is adapted as soon as new components are available; however in step b, architecture adaptation happens after a failure. (2) in step a, the architecture after adaptation contains both IR and camera following related components and switching to IR following happens without further architectural adaptation; however in step b, the architecture after adaptation only includes the IR following related components.

3) Implement the PBAAM approach to adapt the architecture you designed in step 1. You need to use C2SADEL for architectural modeling in PBAAM. Please refer to [1] to find out how you can implement PBAAM in the Layered Adaptation Style. You should provide TWO sets of policies for your adaptation architecture, to perform the steps 2.a and 2.b.

To design your architectures you are not limited to the ideas and approaches introduced in the referenced papers. You are encouraged to attempt to devise other, better solutions; if effective, such solutions will be rewarded with extra-credit.

Project Report

As part of this project, you are to generate a final written report that contains the following elements:

A) Document the design of each software architecture that you develop (the results of Step 1, Step 2a/b, and Step 3a/b).

B) Compare the adaptation results for Steps 2a and 2b to answer following questions:
   a. What is the estimated time to design the policies for each step?
   b. What is the estimated time to run the adaptation policies in each step?
   c. In which approach does the system react to the camera failure faster?
   d. How many components have been added to the initial architecture after adaptation in each step?
e. What are the other changes made to the initial architecture after adaptation in each step? (Include any component update, deletion or change in the architecture topology)
f. What are the pros and cons of each step?

C) Repeat the above analysis (B a-f) for Steps 3a and 3b.

D) Compare the adaptation results for Steps 2 and 3 to answer following questions:
   a. What is the estimated time of development for each step?
   b. What are the pros and cons of each approach?

E) Are there adaptation approaches that would be superior to the two used in this paper? If so, can you outline such an approach?

Deliverables

<table>
<thead>
<tr>
<th>Deliverable (to be submitted just like a regular submission)</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 set of Java and C2 files related to step 1</td>
<td>April 16, 2010</td>
</tr>
<tr>
<td>2 sets of Java and C2 files related to steps 2.a and 2.b</td>
<td>April 24, 2010</td>
</tr>
<tr>
<td>2 sets of Java and C2 files related to steps 3.a and 3.b</td>
<td>May 3, 2010</td>
</tr>
<tr>
<td>Final Written Report</td>
<td>May 6, 2010</td>
</tr>
</tbody>
</table>

References

Addendum 1

C2SADEL Extension to support distributed applications

In a distributed application, different architectures in different hosts interact over a network. This addendum describes how a distributed application can be modeled in C2SADEL. It also describes how to set up the distributed application by plugging the port numbers and host names in the DRADEL generated Prism code. The C2SADEL models and Prism implementation of the provided example in this addendum are included in the project auxiliary files.

C2SADEL Extension

Figure 1 depicts a simple distributed application with two architectures on two different hosts.

![Figure A1: A simple distributed application](image)

The extended C2SADEL specification provides a special connector type required for connecting different architectures over a network. This type of connector is called Extensible Connector. An Extensible Connector can be defined either as a Reply or a Request Connector. A Reply Extensible Connector receives request events from the architecture on the other side of the connection and sends back replies. On the other hand, a Request Extensible Connector sends request events and receives reply events.

In Figure A1, the connector in Architecture 1 is a Reply Extensible Connector and the connector in Architecture 2 is a Request Extensible Connector. Figures A2 shows how these connectors are defined in a C2SADEL specification.
Note that C2SADEL only describes each of the architectures in a distributed application separately and does not model the connectivity between different architectures. The connections between different architectures will be specified by architect only in the implementation.

**Prism Implementation**

In Prism, connectors and components get connected through their ports. There are two types of ports: *Reply* and *Request*. *Reply ports* receive request events and send reply events. *Request ports* send request events and receive reply events. In a distributed Prism application, components and connectors in different architectures are connected using *Extensible Reply* and *Request ports*.

![Figure A2: (a) connector type for Architecture 1 from Figure A1, (b) connector type for Architecture 2 from Figure A1](image)

An *Extensible Reply port* has a socket distribution object which listens to an assigned **port number** and accepts incoming connections from *Extensible Request Ports* on the other hosts. An *Extensible Request port* needs the **port number** and the **host name** of a target *Extensible Reply Port* to be able to connect to it.

The Prism code generator in DRADEL provides the required code to instantiate an *Extensible Request or Reply Port* for each *Extensible Connector* in a C2SADEL architecture. However, it does not provide the port numbers and host names for these ports to connect them. The architect of the system needs to complete this code and provide the same port numbers for the matching *Extensible Request* and *Reply ports* so that they may be connected properly. The architect should also provide the host name of the target *Extensible Reply Port* for each *Extensible Request Port*. 