CS 578 – Software Architectures
Spring 2012
Course Project
Due: *Sunday, April 29, 2012*

In this assignment, you are required to extend the design of your C4 application from homework 2 and implement it in Java. The implemented application needs to run on top of the PrismMW middleware (see http://sunset.usc.edu/~softarch/Prism/). The extended system is a distributed application that is tolerant to communication resource failures. You will also use an Eclipse visualization tool to view your running architecture. The DRADEL tool you supports PrismMW code generation based on C2SADEL models.

Facilities You Will Find Useful

DRADEL Code Generator

DRADEL provides a code generator that creates the structure of your application based on your provided models. The generated code skeleton is intended to run on top of PrismMW, which is implemented in Java. PrismMW is provided to you as a part of the DRADEL distribution. The generated code does not include any application logic. The application logic of a system is reflected in the pre- and post-conditions of the operations in your C2SADEL component models.

For this homework, you need to download a new distribution of DRADEL from the address below:
http://softarch.usc.edu/~farshad/DRADEL_HW3.zip

Architecture Visualization

Download the visualization tool from the following URL and install it in your Eclipse workspace:
http://softarch.usc.edu/~farshad/VisualizerPlug-in.zip

To install this plug-in you need to unzip the file and copy the jar files into the plugin directory of your Eclipse distribution and then run Eclipse. Then, to include the view windows of the visualizer go to the "Menu>Window>Show View>" and add the "Architecture Visualizer>Visualizer Architecture List" and "EclipseGraphviz>Image Viewer" views to your eclipse.

The visualizer plug-in is dependent on Graphviz, an open source graph visualization software. You need to download and install Graphviz software from http://www.graphviz.org/. Then you need to set up the Eclipse Plug-in to point to your Graphviz software installation. To do so, go to the Eclipse Preference menu and choose Graphviz from the left menu. In this page, you need to provide the address to "dot.exe" file, which will be in your Graphviz software installation directory. On Linux/Mac, the file will be installed in the "/usr/local/bin/dot" or "/usr/bin/dot".

PrismMW middleware is delivered to you as an Eclipse Plug-in. It is included in the above zip file and is a jar file. You can find the source code for PrismMW in the URL below, in case you like to take a look at it.

http://softarch.usc.edu/~farshad/PrismMWSource.zip

The generated code by DRADEL can be automatically visualized by this tool. Refer to the examples in the following URL to see some visualizable C2 architectures:
http://softarch.usc.edu/~farshad/prismTest.zip

You will need to fix the jar file references in the test and DRADEL projects after you import them to the Eclipse environment. The references to the jar files need to point to the jar files that you download and put in the …/Eclipse/plugins/ directory.

Assignment Requirements (% of project grade noted)

Step 1: Basic Implementation (40%)

You are expected to implement the application-specific logic for the requirements in homework 2, by "filling in the blanks" of the automatically generated PrismMW code. The entire application may run in a single address space (i.e., OS process). The two DB components must be implemented persistently, but you may use the local file system (i.e., you don't need to create or install an actual database such as MySQL).

Step 2: Make Your Implementation Distributed (20%)

You need to extend the design of your application from homework 2 so that different parts of its architecture run on different hosts (or at least in different processes). Figure 1 depicts the desirable deployment architecture. For modeling of the distributed applications, refer to Addendum 1.

Moreover, your implemented architecture should be tolerant to the

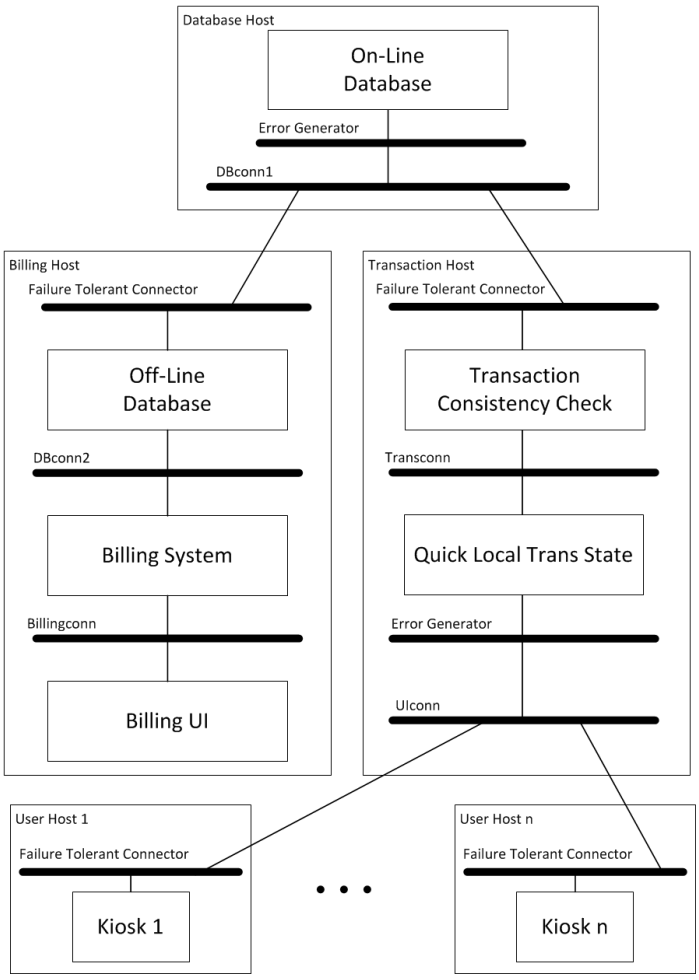

Figure 1 – Distributed Architecture

communication failures between different hosts in the system. The Error Generator connectors in Figure 1 are in charge of simulating communication failures by randomly dropping the request and notification events. You may design a Failure Tolerant Connector (as shown in the figure above) or use your own solution. You are required to simulate temporary communication failures in your system and show that the system tasks are not going to fail because of temporary communication failures.

Step 3: Make Your Implementation Adaptable (10%)

For this part, you are supposed to manually add some components and a new architecture to enable run-time architecture adaptation. The final architecture of your system after this step is shown in Figure 2. An Admin component needs to be added to

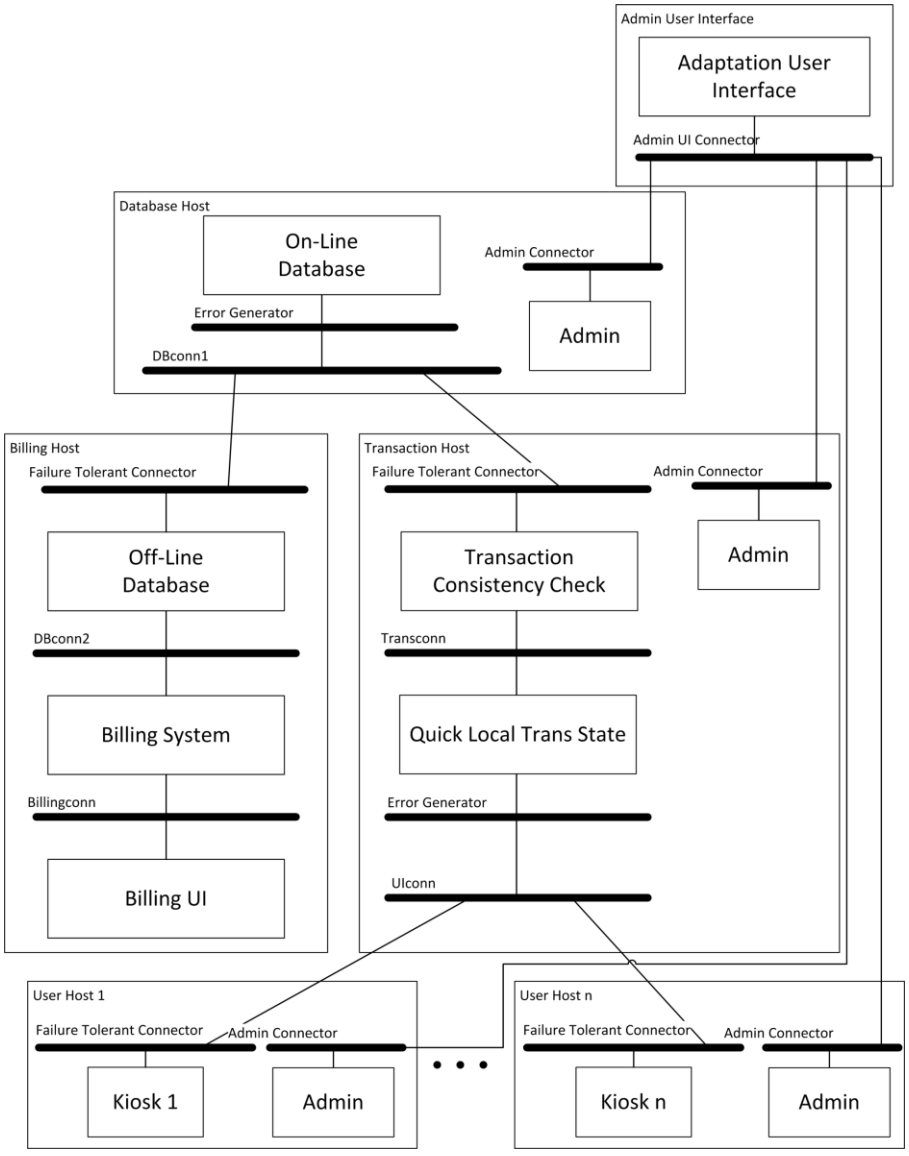

Figure 2 – Adaptable Architecture

the User, Transaction, and Database hosts. For starters, you may borrow the Admin Comp component (*edu.usc.prism.test.components.AdminComponent*) from the examples that you downloaded earlier. This component has a reference to the architecture to which it belongs and can manipulate the architecture at run-time.

You also need to add a new component called Adaptation User Interface. This component is supposed to run in a new host/process (Admin User Interface host in Figure 2). For the Adaptation User Interface component, you may start with the Admin GUI component (*edu.usc.prism.test.components.AdminGUI*) from the provided examples.

Note that you will not need to change your C2SADEL models.

Step 4: Redeploy Your Implementation (15%)

You need to extend the Adaptation User Interface component to send required instructions to the admin components to re-deploy some parts of your system at run-time. The architecture after re-deployment is shown in Figure 3. Feel free to make any needed changes to the Admin Components.

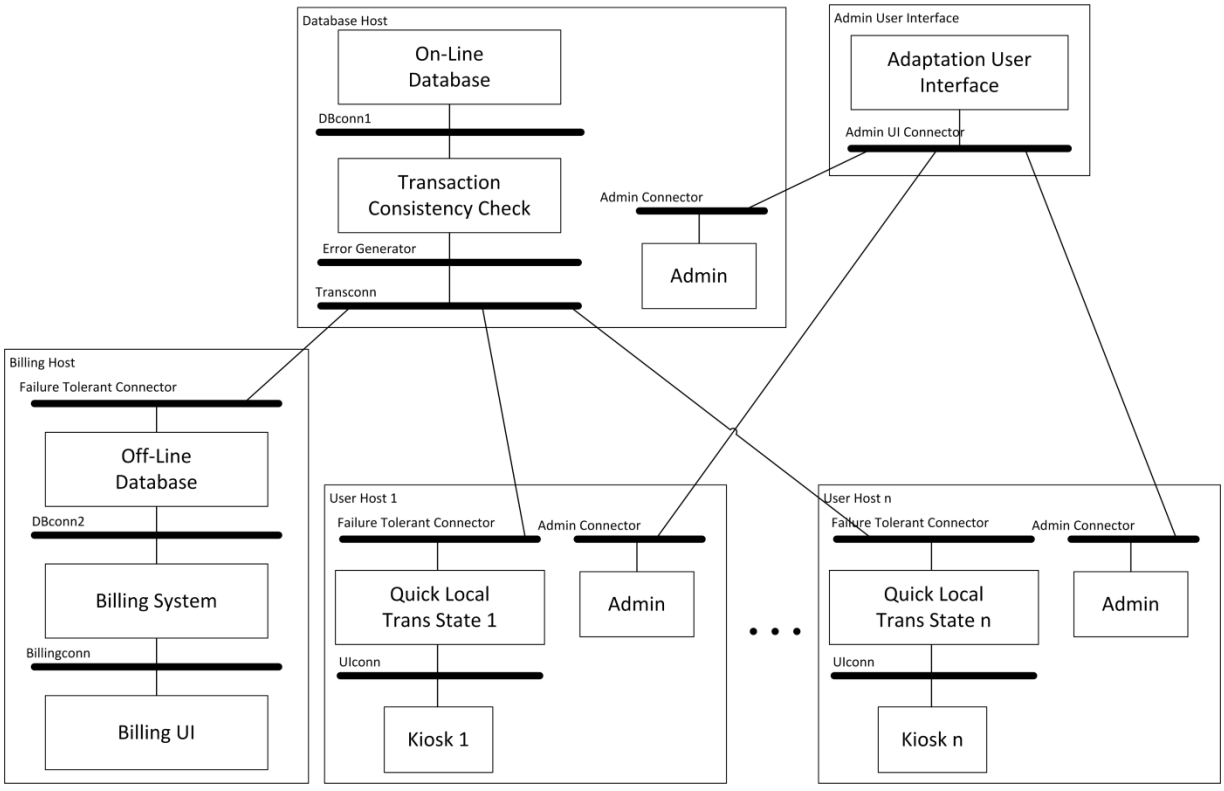

Figure 3 – Redeployed Architecture

Step 5: Add New Components to Your Implementation (15%)

In this step, you need to provide support for sending and receiving text messages between your users. The architecture of the system after this step is depicted in Figure 4. You need to add a Messaging UI component to the architecture of a User Host at run-time. User should be able to use the user interface provided by this component to send and receive text messages to other users in the system.

To do this, a new component, Message Database, needs to be added to the architecture of the Database host. This component should save text messages coming from users and deliver them to the target users whenever they get online.

All the architecture modifications should be initiated and managed using the Adaptation User Interface.

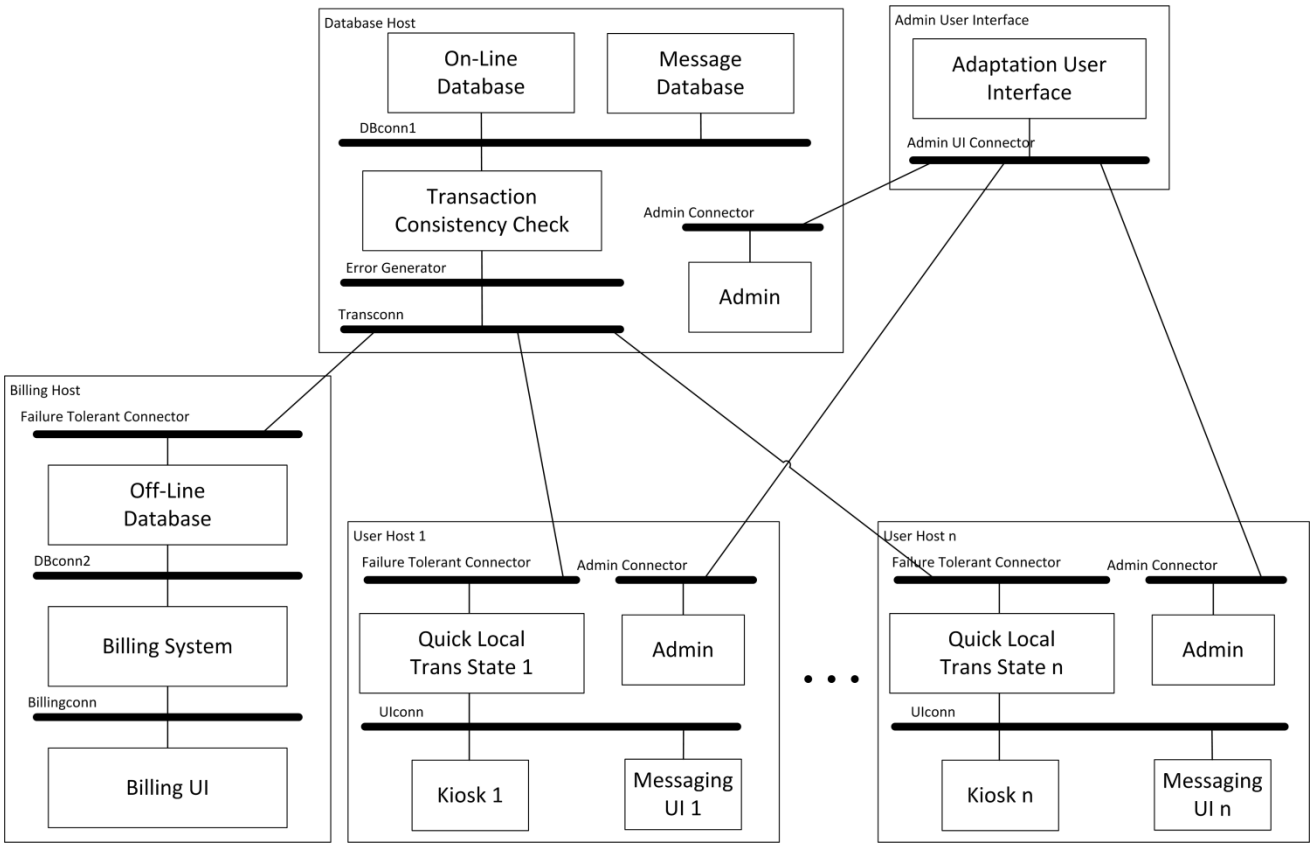

Figure 4 – Extended Architecture 1

Addendum 1

C2SADEL Extension to support distributed applications

In a distributed application, different architectures on 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 PrismMW code. The C2SADEL models and PrismMW implementation of the provided example in this addendum are included in the Project auxiliary file.

C2SADEL Extension

Figure A1 depicts a simple distributed application with two sub-architectures on two different hosts.

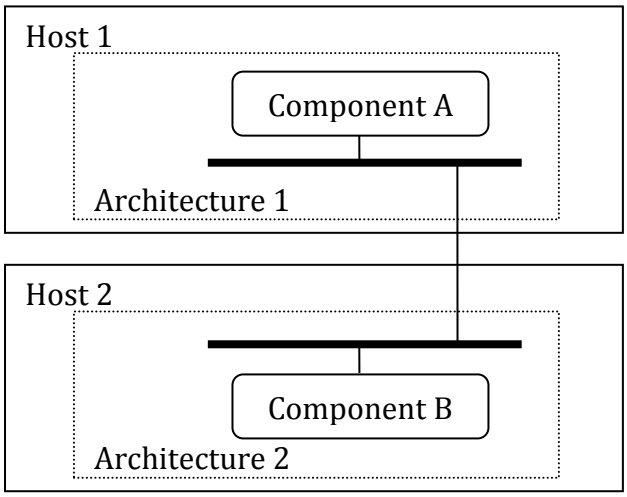

Figure A1: A simple distributed application

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 *Reply* or *Request* Connector. A *Reply Extensible Connector* receives request events from the architecture on the other side of the connection and sends back replies. Conversely, 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*. Figure A2 shows how these connectors are defined in 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 the

architect (you) only in the implementation. You may ignore the type checking errors that are cause by this fact.

PrismMW Implementation of System Distribution

In PrismMW, connectors and components get attached 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 Ports* and *Extensible Request ports*.

(a)

```
connector BottomConn_Type is {
        message_filter no_filtering;
        extensible Reply;
}
```

(b)

```
connector TopConn_Type is {
        message_filter no_filtering;
        extensible Request;
}
```

Figure A2: (a) connector type for Architecture 1, (b) connector type for Architecture 2

An *Extensible Reply port* has a socket distribution object that 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 can get connected properly. The architect should also provide the host name of the target *Extensible Reply port* for each *Extensible Request Port*.