Public Transportation Model over Opportunistic Networks

Final Report

Shuai Yue

Internet Real-Time Lab

Columbia University

sy2342@columbia.edu

Supervised by SeGi Hong, SungHoon Seo and Prof. Henning Schulzrinne

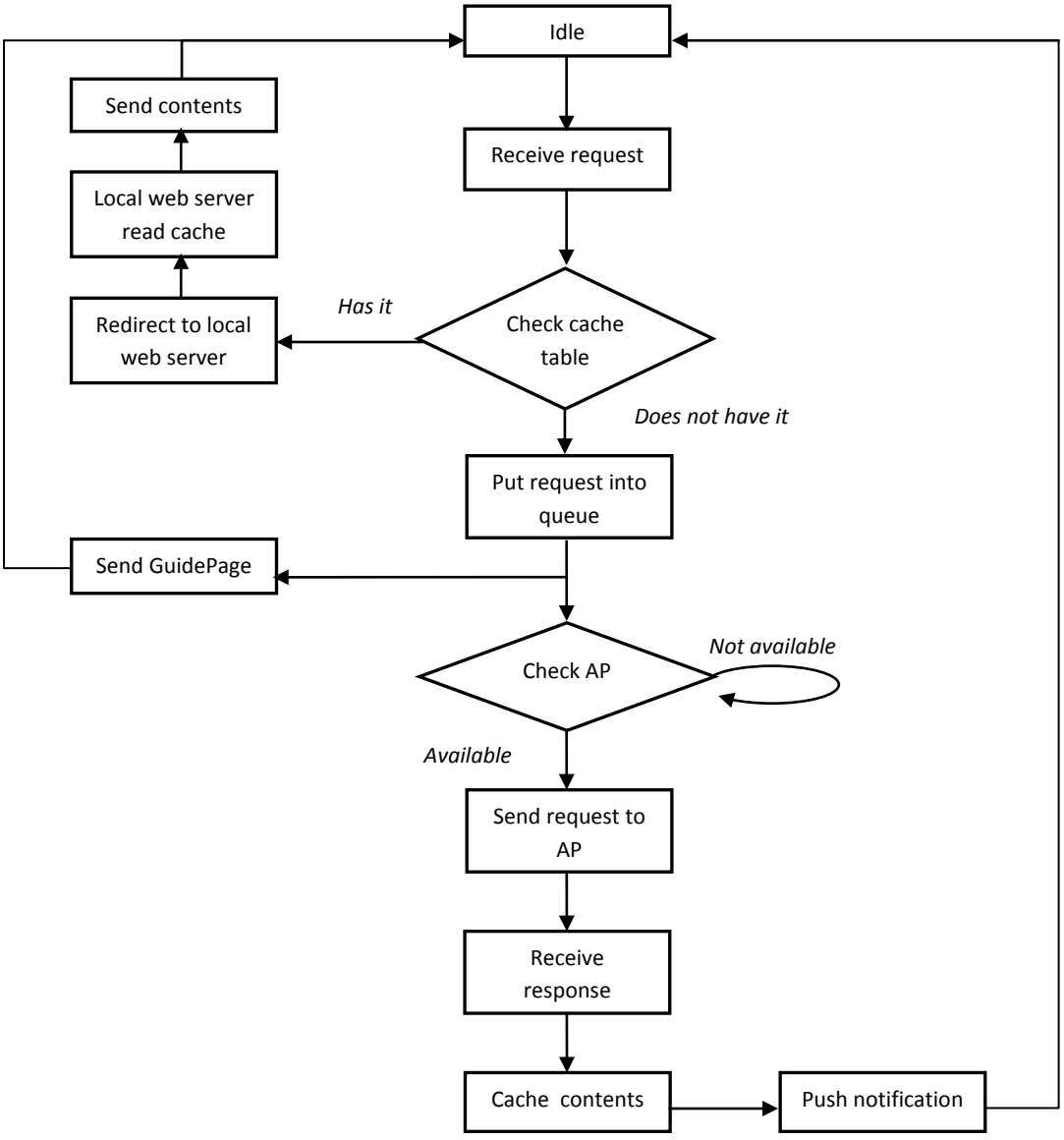
December, 2010

Introduction

We have built a system that provides Web content to passengers in an intermittently connected network. Our system is a proxy-cache-based system in which a proxy node caches Web content requested by passengers during the Internet connection period, so other passengers can obtain the cached content regardless of Internet availability

Workflow

The following flow chart describes the workflow of our system.



Environment Requirement (for Linux)

1 Make sure you have two wireless network interface cards on your machine, so you can use one to set up an Ad Hoc network and use the other to connect to access point (AP).

2 Make sure you have installed **JDK** (Our Version is 1.6).

3 Make sure you have installed **Tomcat** (Our Version is 6.0). For Linux machine, you should follow the default installation directory, i.e. under the path "/usr/local/tomcat/"

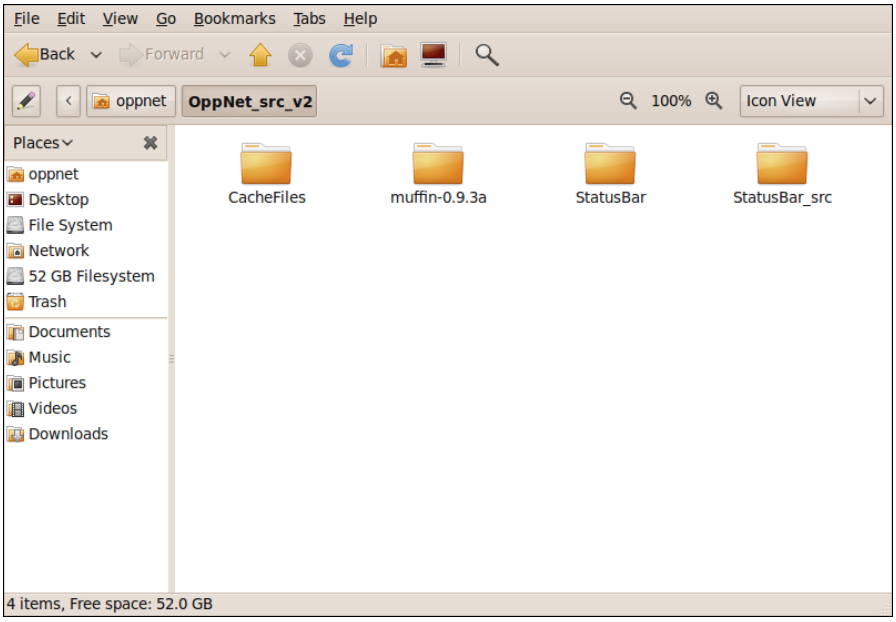
4 For development purpose, you also need to install **Eclipse** (Our Version is 3.5). But this is not necessary if you just want to run our application.

Just for reminding, for Windows system, you have to install Windows SDK because it uses different method to detect network connection. Also, you may need to modify the paths in some source files we wrote to let them satisfy Windows directory style. This document is for Linux system.

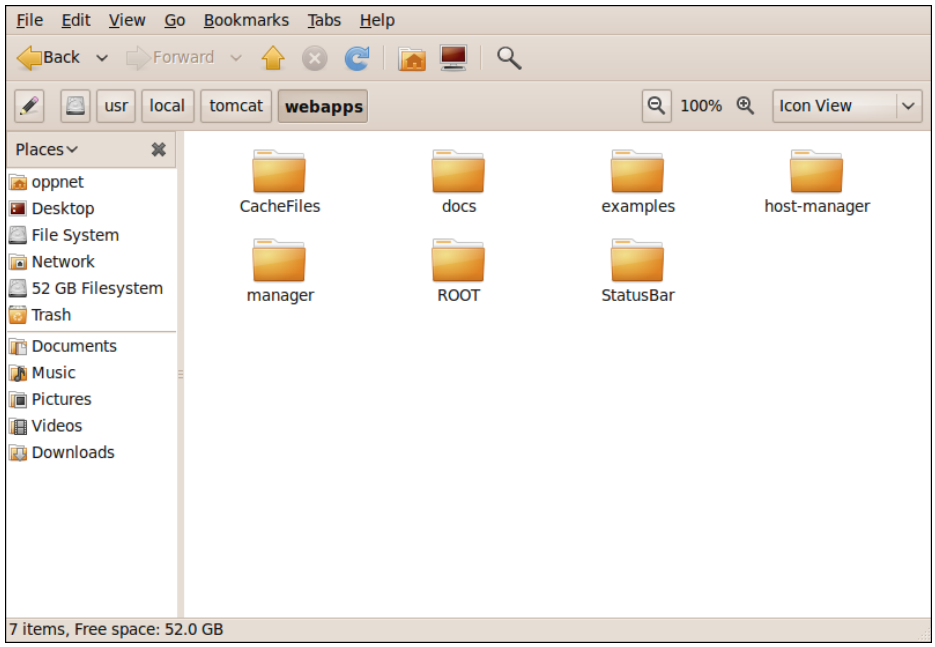
Install Instruction

Step 1 Set up an Ad Hoc network on your proxy server machine, and manually set its IP address for Ad Hoc to *10.42.43.10*. Use a client machine to join this Ad Hoc network.

Step 2 Unzip and open folder ***OppNet_src_v2*** and you can see the following folders.



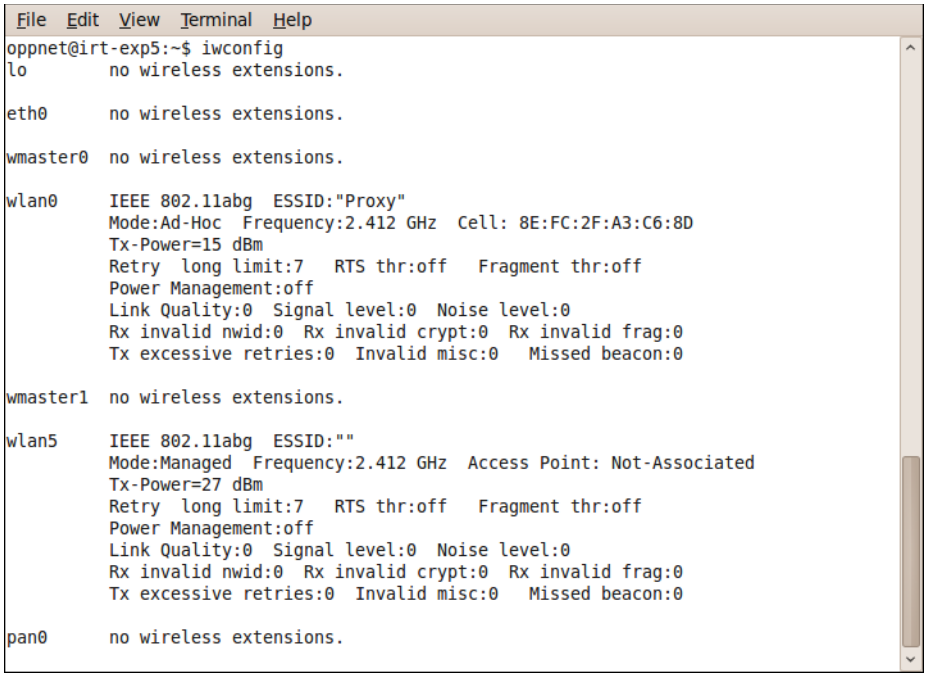
Step 3 Copy folder ***CacheFiles*** and ***StatusBar,*** and then paste them under directory "/usr/local/tomcat/webapps/" as shown below.



Step 4 Open a prompt window and start up Tomcat using the following command:

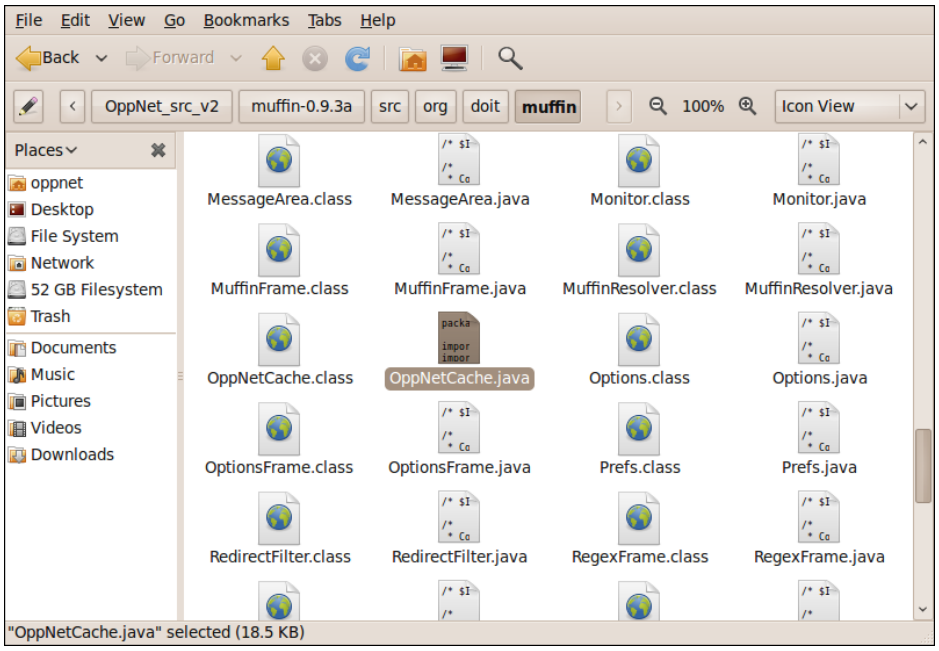
```
oppnet@irt:~$ cd /usr/local/tomcat/bin
oppnet@irt:/usr/local/tomcat/bin$ ./startup.sh restart
```

Step 5 Use command `iwconfig` to check the name of your wireless NIC which is used to establish connection to the access point (AP). For example, in the following figure, wlan5 is used for AP while wlan0 is used for Ad-Hoc.

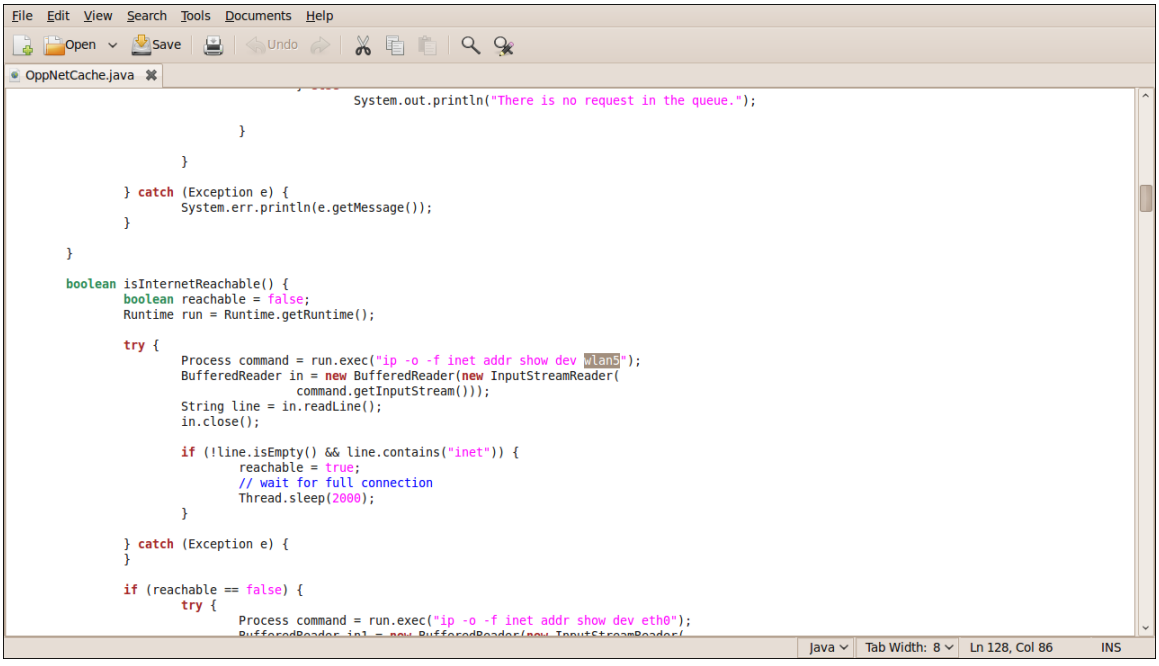


If you also use wlan5 as the wireless NIC name for AP connection, skip the following steps and read Step 9 directly. If not, remember your wireless NIC name for AP, for example wlanX.

Step 6 Open file ***OppNetCache.java*** under the path "OppNet_src_v2/muffin-0.9.3a/src/org/doit/muffin/"



Step 7 Go to Line 128, replace "wlan5" (as highlighted below) with the name wlanX in last step. Save and exit.



Step 8 Open another prompt window, set the path to "OppNet_src_v2/muffin-0.9.3a/". For example, if you unzipped ***OppNet_src_v2*** under path "/home/oppnet", you can use the following command:

```
oppnet@irt:~$ cd /home/oppnet/OppNet_src_v2/muffin-0.9.3a/
```

Then, input the following configure command

```
oppnet@irt:/home/oppnet/OppNet_src_v2/muffin-0.9.3a$ ./configure
```

Then, input the following make command

```
oppnet@irt:/home/oppnet/OppNet_src_v2/muffin-0.9.3a$ make
```

Step 9 Set the path to "OppNet_src_v2/muffin-0.9.3a/src" and start up Muffin. If you are from Step 8, use the following command to set path:

```
oppnet@irt:/home/oppnet/OppNet_src_v2/muffin-0.9.3a$ cd ./src
```

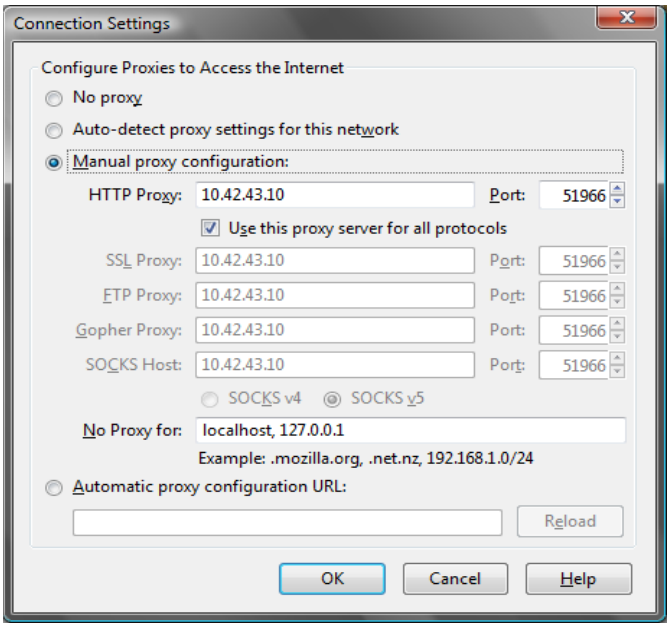
Or, if you are directly from Step 5, open another prompt window and use the following command to set path:

```
oppnet@irt:~$ cd /home/oppnet/OppNet_src_v2/muffin-0.9.3a/src
```

Step 10 Run Muffin using the following command:

```
oppnet@irt:/home/oppnet/OppNet_src_v2/muffin-0.9.3a/src$ java Muffin
```

Step 11 The Muffin is running now and you can monitor what's happening on the proxy server side. On your client machine, open a web browser and configure the proxy setting. (For better performance, please use Mozilla Firefox and disable its cookies, history or cache). Set the proxy's IP address as 10.42.43.10 and port number as 51966 and save configuration. As shown below.



Step 12 Use Firefox to send an HTTP request for a webpage. And you will see on proxy server side that this request is in the queue right now. If the proxy server side has network connection to AP, it will send out the request and cache the response then push notification to client, if it doesn't, it will keep checking network status.

Step 13 To stop Muffin, press key Ctrl+C; to shut down tomcat, use command `./shutdown.sh`

Development Instruction

This section is for developers who want to explore our system. There're two major functional components of this system: the proxy server program and the local web server program.

Proxy server program

The proxy server program is based on Muffin, a java-based open source proxy server solution. Please visit http://muffin.doit.org/ to learn about how Muffin works. You can download and install the original Muffin to try its filtering functionality.

Usually developers can write their own customized filters for Muffin simply by using the filter interfaces that Muffin has provided. But in our application, besides adding a new filter, we also modified the source codes of Muffin to make things work. As a result, it not only handles all the incoming and outgoing packets, but also merges with web caching tool and queue scheduling functionality.

Folder "OppNet_src_v2/muffin-0.9.3a/src/org/doit/muffin/" is the most important folder and it contains about sixty core source files of Muffin. It has a sub folder ***filter*** which contains all the customized filters.

We have modified three source files: ***Server.java***, ***Handler.java*** and ***FilterManager.java***, and wrote a new source file ***OppNetCache.java***. In the ***filter*** folder, we added three files: ***OppNet.java***, ***OppNetFilter.java*** and ***OppNetFrame.java***.

The ***Server.java*** creates a server socket and listens for requests from client sockets. It will open a new thread (a *Hander* object) for each request it gets. It also creates a paralleling thread (an *OppNetCache* object) with all the other threads.

The ***Hander.java*** is a thread class that processes a request and its response. Before sending out the request or after obtaining the response, it will go through all the selected filters, including our own customized filter.

The ***FilterManager.java*** lists all the filters in Muffin.

The ***OppNetCache.java*** is also a thread class; it manages the queues, controls scheduling, check network connection, caches (pre-fetches) Web contents and maintains a ***CacheTable.txt***.

The ***OppNet.java***, ***OppNetFilter.java*** and ***OppNetFrame.java*** are the filter classes; they process the request, check cache and add new items into the queue.

To modify our proxy server program, you can directly modify these source codes or add new ones. Each time after modification, remember to compile the source files with following commands:

```
oppnet@irt:/home/oppnet/OppNet_src_v2/muffin-0.9.3a$ ./configure
oppnet@irt:/home/oppnet/OppNet_src_v2/muffin-0.9.3a$ make
```

Local web server program

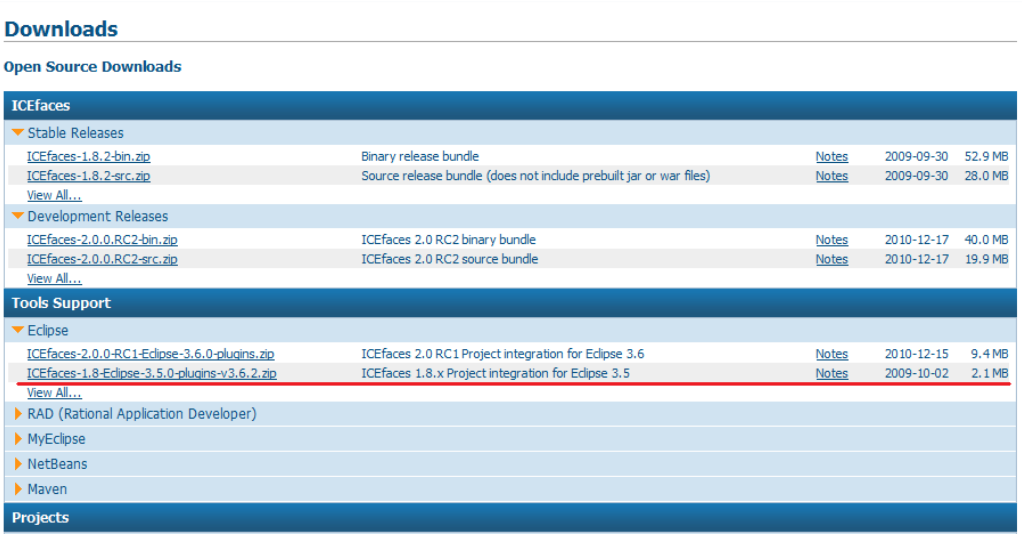
The local web server program is based on Tomcat, providing cached web contents for clients, deployed with AJAX Push technique. To modify this part, you need to use IDE like Eclipse since the AJAX Push is an ICEfaces Facelet Project. If you are not familiar with AJAX Push, please refer the following two links to learn about how it works.

http://www.digit.lk/09_sept_eclipse

http://facestutorials.icefaces.org/tutorial/easy-ajax-push.html

For our application, it's more complicated. To modify our local web server program, follow the next steps:

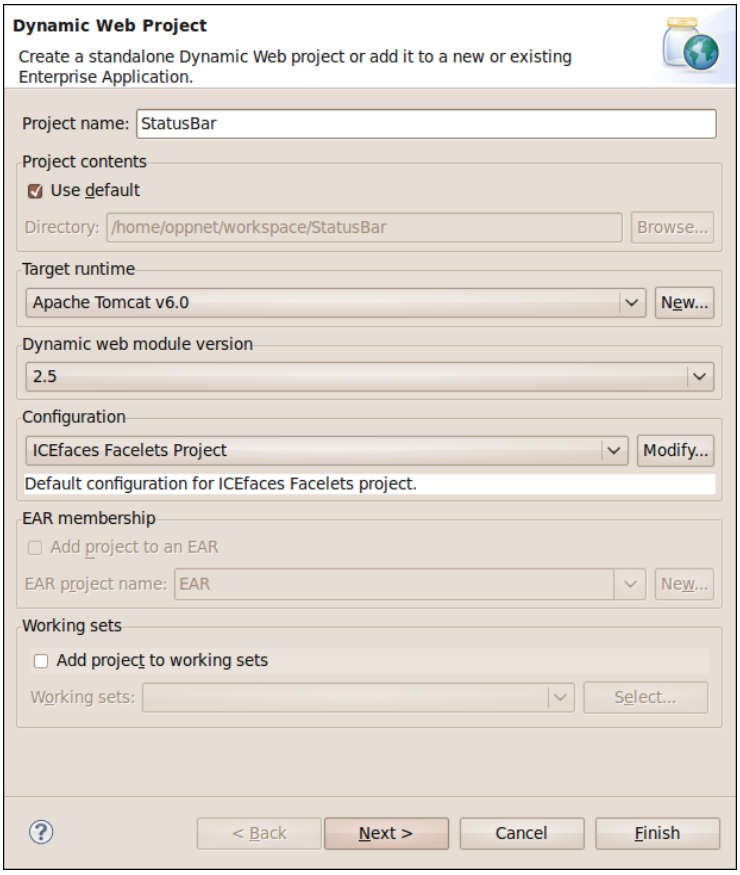
Step 1 Download "ICEfaces-1.8-Eclipse-3.5.0-plugins-v3.6.2.zip" from www.icefaces.org downloads page and unzip it in directory.



Step 2 Run Eclipse and select "Help" → "Install New Software". Then select "Add" → "Local". Input the ICEfaces-plugin directory. Click "OK".

Step 3 Uncheck “Group items by category”. Select all the items in the list. Then click “Next”. Accept license and finish. Restart Eclipse.

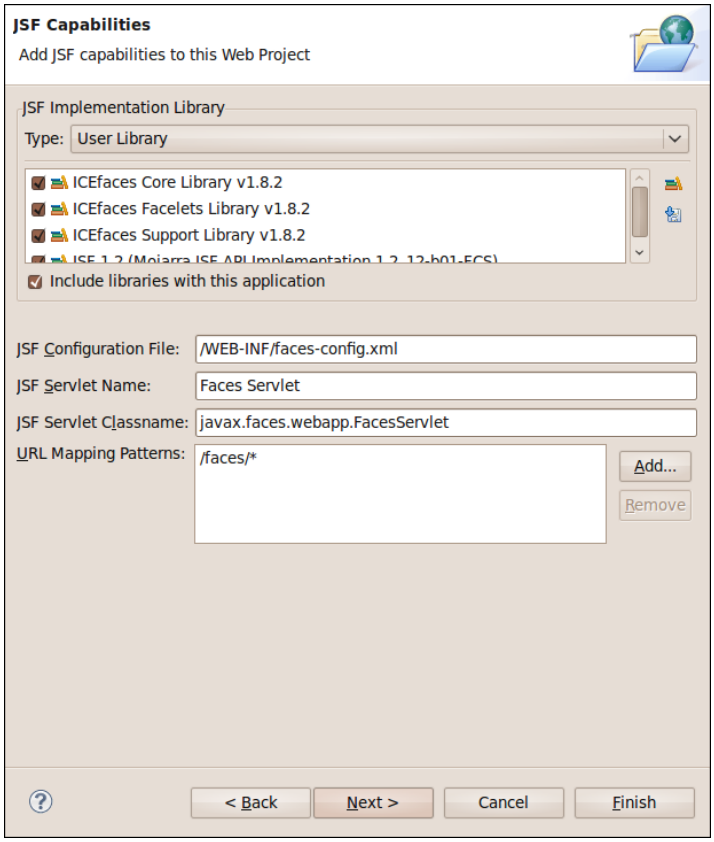
Step 4 Create a new “Dynamic Web Project” as shown below, name it as “StatusBar”. Choose “ICEfaces Facelets Project” in the “Configuration”. Click “Next”.

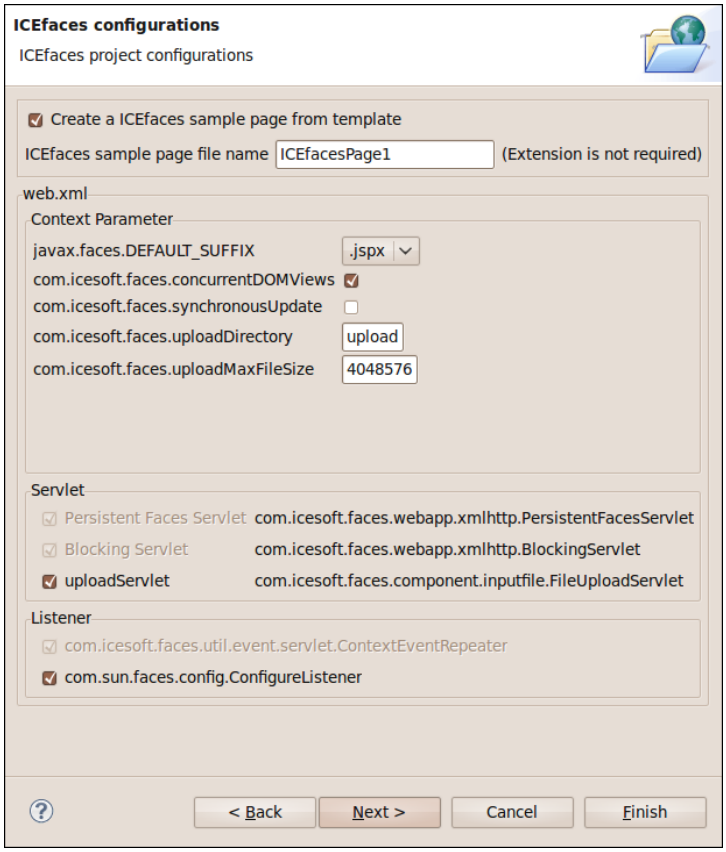


Step 5 In “Java”, click “Next”. (Not shown here).

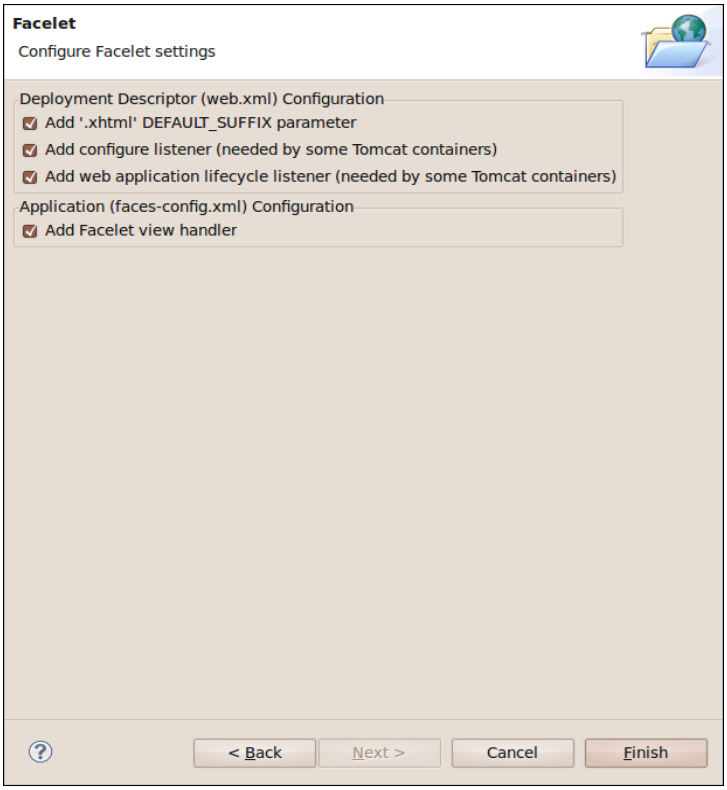
Step 6 In “JSF Capabilities”, click the “Download Library” icon, there is a list of libraries to download, select ICEfaces Core Library, ICEfaces Facelets Library, ICEfaces Support Library and JSF 1.2. You can only download one library at a time. After download all libraries, check them all and click “Next”. (See figure in next page)

Step 7 In “Iceface Configurations”, make sure to check “com.icesoft.faces.concurrentDOMViews” and “com.sun.config.ConfigureListener”. Click “Next”. (See figure in next page)

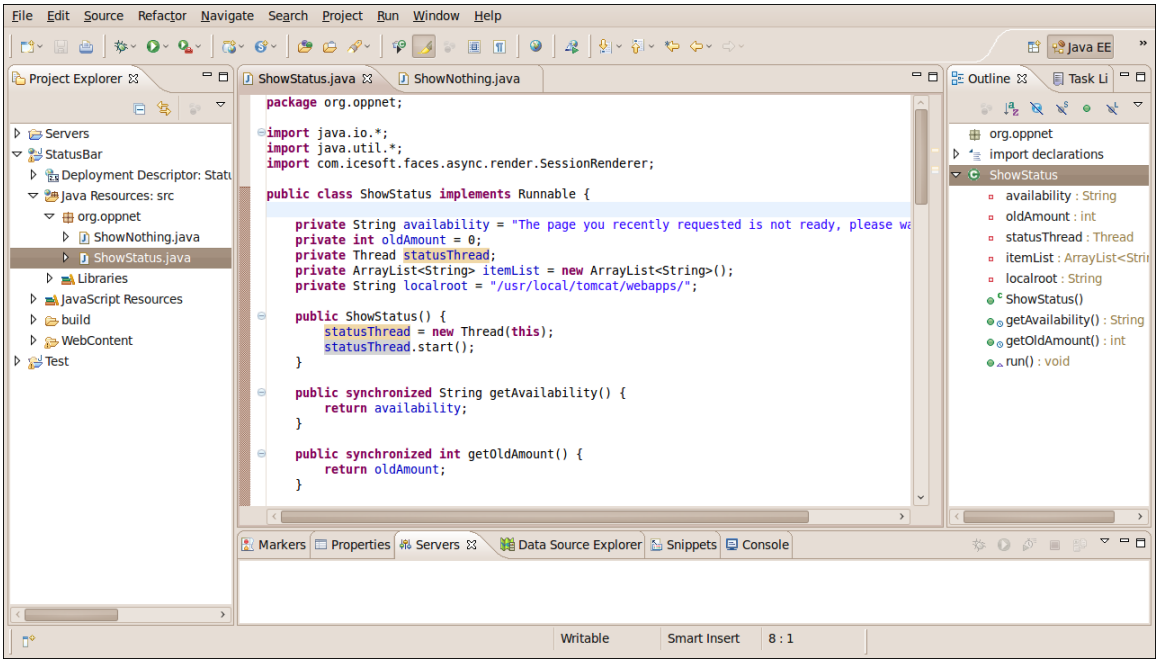

JSF Capabilities
Add JSF capabilities to this Web Project
JSF Implementation Library
Type: User Library
ICEfaces Core Library v1.8.2
ICEfaces Facelets Library v1.8.2
ICEfaces Support Library v1.8.2
Include libraries with this application
JSF Configuration File: /WEB-INF/faces-config.xml
JSF Servlet Name: Faces Servlet
JSF Servlet Classname: javax.faces.webapp.FacesServlet
URL Mapping Patterns: /faces/*
Add...
Remove
< Back
Next >
Cancel
Finish


ICEfaces configurations
ICEfaces project configurations
Create a ICEfaces sample page from template
ICEfaces sample page file name ICEfacesPage1 (Extension is not required)
web.xml
Context Parameter
javax.faces.DEFAULT_SUFFIX .jspx
com.icesoft.faces.concurrentDOMViews
com.icesoft.faces.synchronousUpdate
com.icesoft.faces.uploadDirectory upload
com.icesoft.faces.uploadMaxFileSize 4048576
Servlet
Persistent Faces Servlet com.icesoft.faces.webapp.xmlhttp.PersistentFacesServlet
Blocking Servlet com.icesoft.faces.webapp.xmlhttp.BlockingServlet
uploadServlet com.icesoft.faces.component.inputfile.FileUploadServlet
Listener
com.icesoft.faces.util.event.servlet.ContextEventRepeater
com.sun.faces.config.ConfigureListener
< Back
Next >
Cancel
Finish

Step 8 In “Facelet” as shown below, make sure to check all the items. Click “Finish”. Then you should see the project “StatusBar” in project explorer view.

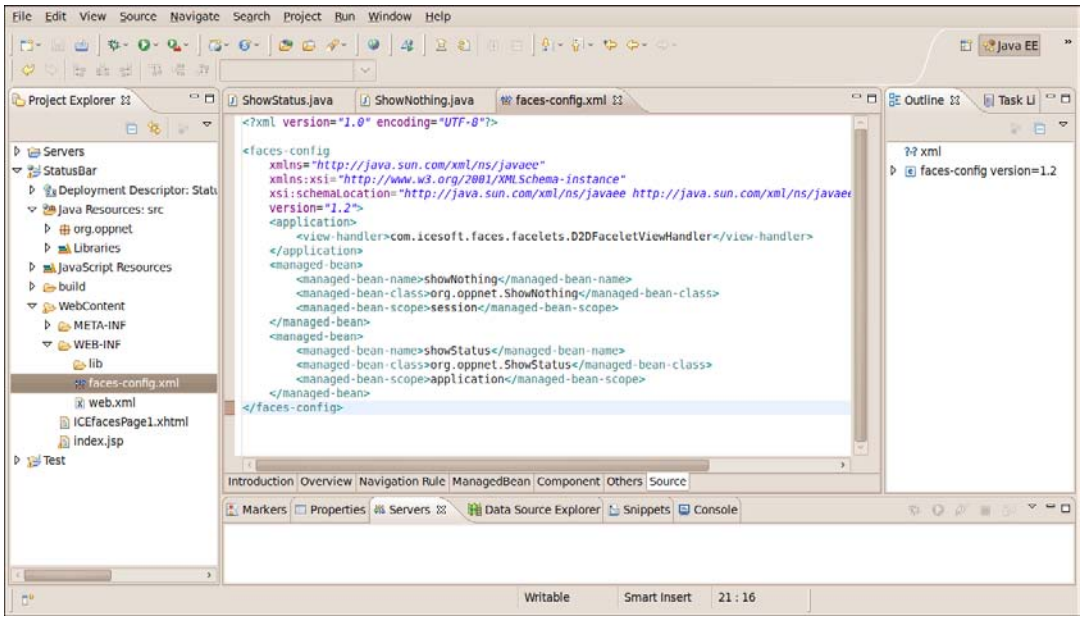


Step 9 Under “Java Resources: src”, create a package “org.oppnet”, then create two classes “ShowNothing.java” and “ShowStatus.java”.

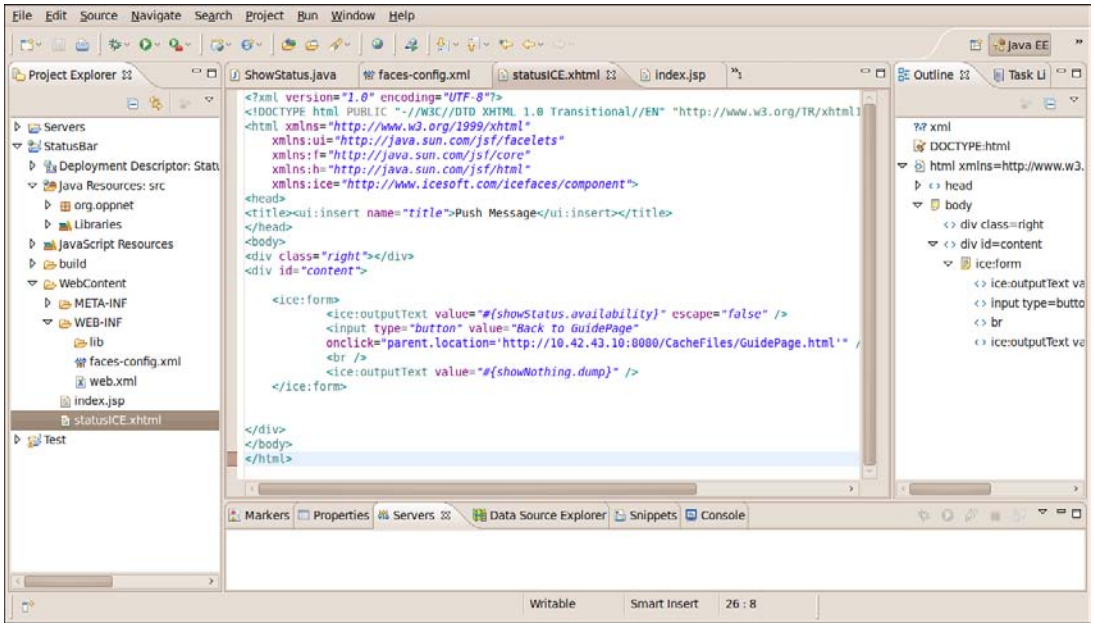


Go back to the source code folder of our application, namely ***OppNet_src_v2***. Open sub folder ***StatusBar_src/***. Find two files "***ShowNothing.java***" and "***ShowStatus.java***". Copy the content of these two files and paste them in the corresponding new class you just created in Eclipse project. Save "ShowNothing.java" and "ShowStatus.java".

Step 10 In Eclipse, expand folder "WebContent" → "WEB-INF", open "faces-config.xml", choose the "source" view of this xml file as below. Use the content of ***OppNet_src_v2/face-config.xml*** to replace the content of current file. Save it.

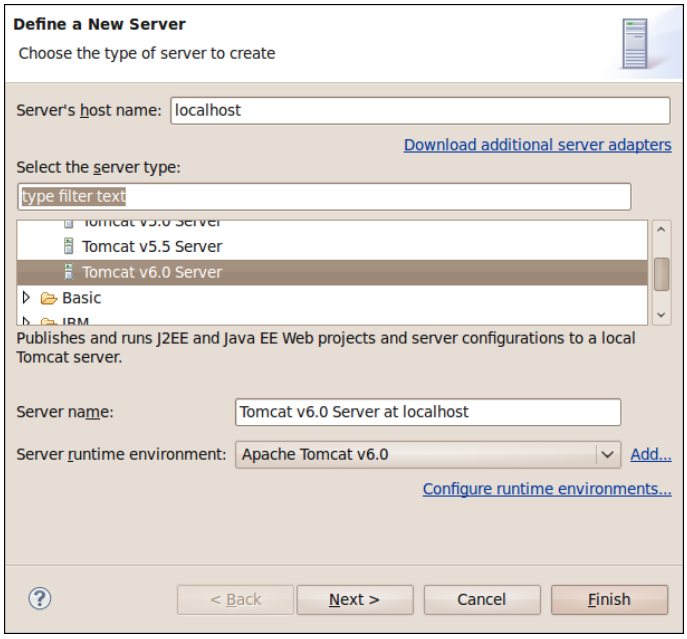


Step 11 In Eclipse, find the file "ICEfacesPage1.xhtml" under folder "WEB-INF" as below, rename it as "statusICE.xhtml"and open it. Use the content of ***OppNet_src_v2/statusICE.xhtml*** to replace the content of current file. Save it.

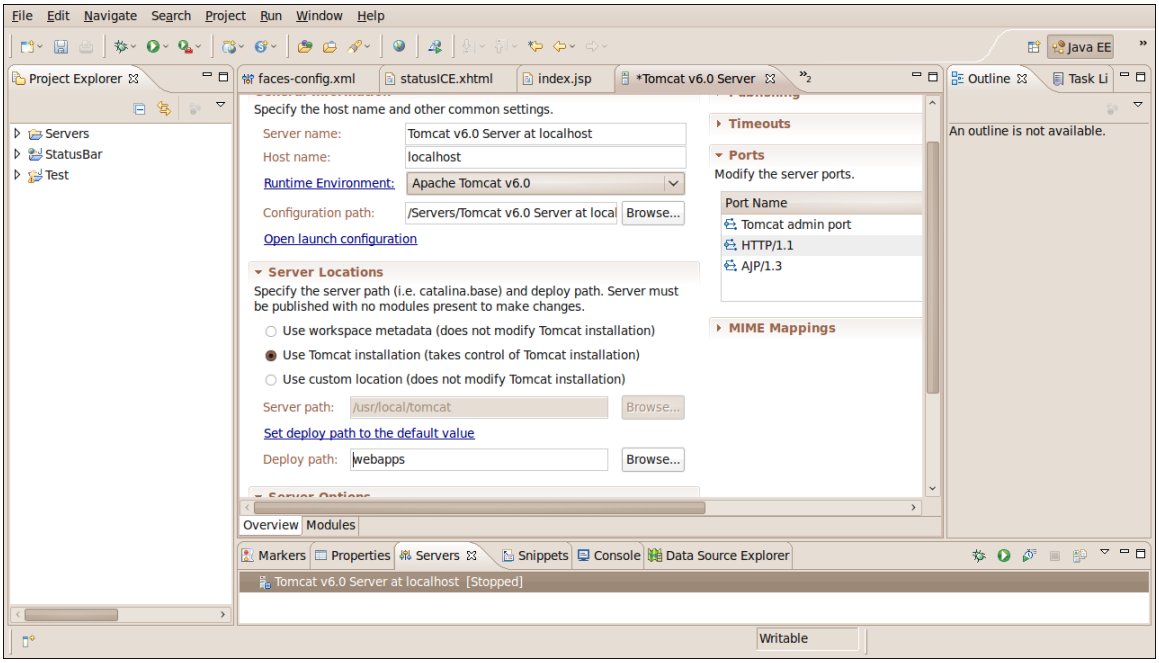


Step12 In Eclipse, find the file “index.jsp” under folder “WEB-INF” and open it, Use the content of ***OppNet_src_v2/index.jsp*** to replace the current file. Save it.

Step 13 Note that in the bottom panel of Eclipse, there is a “Servers” label. Click the label, move the cursor the blank zone under the label. Then Right click the mouse to add a new Server. Select “Tomcat v6.0 Server” and click “Finish”.



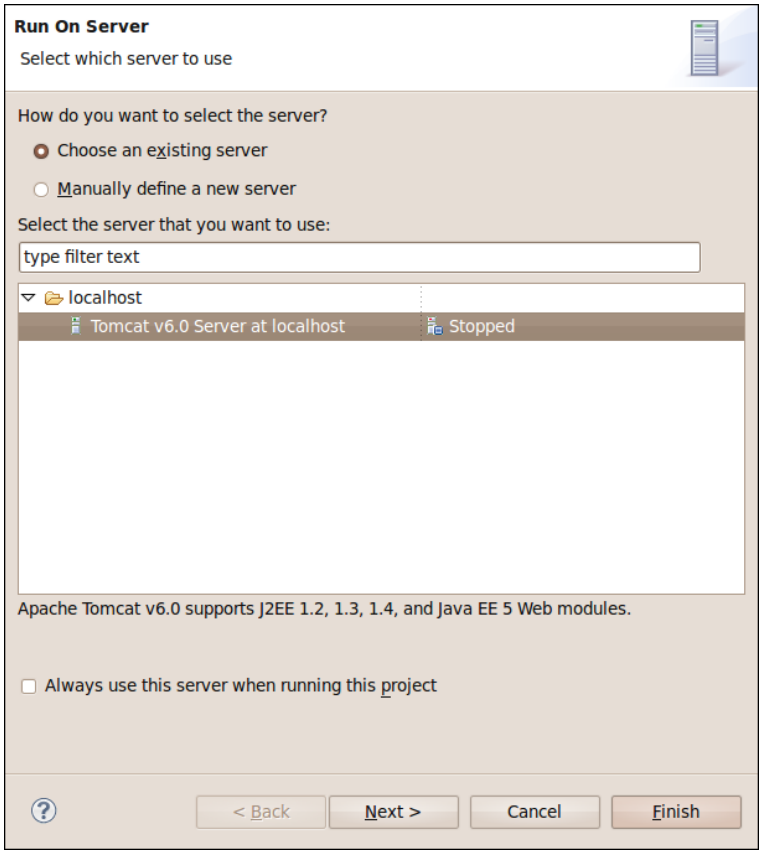
Step 14 Double click the new server “Tomcat vc6.0 Server at localhost” and you can see the page below.



For Server Location, check “Use Tomcat installation (take control of installation)”. Change “Deploy path” to “webapps”. Save it.

Step 15 Before deploying the new web project to Tomcat server, Check the folder “usr/local/tomcat/webapps/” and make sure that there is no previous folder ***StatusBar*** that exists. If there is one, delete it. This is very important. Otherwise the Tomcat will report error.

Step 16 In Eclipse, right click the project “StatusBar” and click “Run” → “Run it on Server”. The following window will appear, click “Finish”. Then the Facelet project “StatusBar” is deployed and running in the Tomcat.



You can stop it by clicking the stop button. In this way, you can use Eclipse to start or stop Tomcat conveniently.

There is another way to manipulate Tomcat. After the first successful deployment via Eclipse, you can also find that a ***StatusBar*** folder was created under path “usr/local/tomcat/webapps/”. This means next time, you don’t need use Eclipse to manipulate Tomcat. You can follow the

```
oppnet@irt:~$ cd /usr/local/tomcat/bin
oppnet@irt:/usr/local/tomcat/bin$ ./startup.sh restart
```

command to start the Tomcat and it will automatically run the ***StatusBar*** project. But make sure that you can only choose a way to manipulate Tomcat at a time.

Note that after you close Eclipse IDE, next time if you want to deploy the ***StatusBar*** project again, you need to delete the previous ***StatusBar*** project and its files, as well as the Tomcat server you created last time. Then repeat Step 4 to Step 15.

Lastly, the two java beans "***ShowNothing.java***" and "***ShowStatus.java***" are very import because they control the business logic here. The ***ShowNothing.java*** does nothing but register a user to a certain group, ***ShowStatus.java*** will check the event and update (about a new available websites) by comparing the ***MappingTable.txt*** and ***CacheTable.txt*** (the usage of these two files will be covered in next section), and render notification to the users that request for that website.

About Cache

We use file systems to store cached files.

Currently, we use a text file to maintain a cache table (we will use database in future). In the "OppNet_src_v2/CacheFiles/" folder, there is a ***CacheTable.txt***, this file keeps all the records of cached requests and is maintained by ***OppNetCache.java***.

There is also a ***MappingTable.txt*** recording which user (IP address) has requested for which websites. This file is for push purpose.

The ***GuidePage.html*** in the same folder is a catalog showing all the already cached websites. And this page will be returned when user's requests are not yet available. This file is updated whenever a new website becomes available.

The ***InvalidPage.html*** in the same folder is a page that will be returned if the HTTP request is invalid.

Note that by default, the ***MappingTable.txt*** and ***CacheTable.txt*** are empty. The ***GuidePage.html*** contains no items. After you copy the whole folder to Tomcat, if you run the application, these three files will be updated. Therefore, if you want to start over again the application from the state when there is nothing in the cache. Please delete the ***CacheFiles*** folder in Tomcat and copy it again from "OppNet_src_v2/CacheFiles/".

Performance Evaluation

We evaluate our system by analyzing the performance of Web page caching and counting the number of sub-links. Note that the following tables only show the results of a certain period.

Since each websites updates now and then and the network signal strength varies, the data can be slightly different (more or less), but within a reasonable range.

Cache a front page and its relevant files (images, css, js etc.)

Mode	Speed*	Website	Size	Num of Files	Caching Time
WiFi	54Mb/s	www.cnn.com	537.8KB	64	2.178s
WiFi	54Mb/s	www.msn.com	471.8KB	62	1.332s
WiFi	54Mb/s	www.wsj.com	968.3KB	74	1.941s
WiFi	54Mb/s	www.nytimes.com	1.1MB	190	1.765s

* Signal Strength: 94%

Sub-links

Website	Total Num of Sub-links In Front Page	Total Num of Sub-links ends with .html	Total Num of Sub-links to pre-fetch *
www.cnn.com	360	94	81
www.msn.com	281	3	0
www.wsj.com	471	280	122
www.nytimes.com	461	374	291
www.time.com	257	102	100
www.bbc.com	306	0	0

* To reduce the total number of sub-links, we will only pre-fetch the sub-links that satisfy:

1 Within the same domain (e.g., www.cnn.com/world/1215unconference.html is allowed)

2 Does not contains symbols like quote, semicolon, bracket etc.

3 Not a duplicate of previous cached or pre-fetched links