Project 3 Improving the Quality of MUSE Services Spring 2007
CSE4/586 © 2007 Bina Ramamurthy Due Date: 5/1/2007

1. Introduction

In this project we will extend the Project MUSE by adding features that will improve the quality of the application developed in Project 2. Features you will add should (i) publish the service developed for consumption (WSDL), (ii) add features such as concurrency, loose coupling, reliability, and persistence and (iii) employ unit testing and performance metrics to evaluate the overall performance improvement.

2. Objectives

To design and develop a meaningful and useful high quality mash up application that leverages the web services from at least two sources.

3. General Requirements

1) Make sure your MUSE application is reasonably novel application that leverages at least two sources of web services (say, Amazon ECS and EBay) in such a way the mash-up leverages the two constituents in a creative fashion. Data created by the mash-up should involve processing by meaningful business logic. For example, zip code from a data set obtained from a service could define the filter for the information extracted from the other source(s) in the mash-up. Another example is creative use of dynamic nature of the data from a source such as EBay. An example of an unacceptable mash-up is the one that literally combines the data from the sources and lists it. Moreover the service that you create should not repeat a functionality that is easily available from any of the constituent web services.

2) When dealing with web services or any remotely accessible services, dependability and reliability are realistic issues. We address this and performance issues by caching the recent results. Implement caching for the recent results of the mash-up.

3) Expose the mashed-up complex services for programmatic consumption by remote applications. Create a WSDL for the mash-up.

4) Information created by the mash-up and the associated business logic will have to be persisted in a database using appropriate schema.

5) Create a portal (servlet or JSP or any equivalent) for accessing the mash-up service. Let the requests be processed asynchronously. This can be accomplished by queuing up the requests (using say, Java Messaging Service (JMS) that is a part of J2EE 5 specification) to the portal and posting the answers when they become available. The results can be obtained from the persisted data. Concurrent requests should be handled by session per request (using say, session java bean). Any transactional data should be managed by entity bean and Java persistence API.

6) Use Junit (for Java based systems) and NUnit (for Microsoft's .NET based systems) and use some performance metric to evaluate the performance of the enhancements to the system. For example, evaluate the average performance with and without caching for a sequence of requests.

4. Functional Requirements from Project 2

Make sure all the requirements from Project 2 are met. These are repeated below:

a. The application should be developed a multi-tier distributed system with at least three layers: data, business (logic) and presentation.
b. The application should be a deployable unit.
c. It should have user registration, login, logout and authentication features. Only registered users can use the service.
d. Let the application focus on a single coherent domain even if it uses feeds and services from other domains.
e. The business logic should be separated from the user interface and the data layer. (For example, you do not want to lump all the logic into your JSP).
f. While your application is in general a consumer of web services, you should expose a few useful services through WSDL.
g. The user interface should provide rich and smart features. It should be visually appealing and user-friendly.
h. Data needed for the application should be carefully designed and stored in normalized form.
i. The application is expected to have multiple interaction screens; the navigation among them should be smooth and intuitive.
j. The application will have users in different roles, a casual user, a registered user and an administrator all with various levels of access privileges and capabilities.
k. All the input should be validated and incorrect input should be appropriately communicated.
l. You application should a hierarchy of exception classes to handle exception at various levels.
m. Decide on a catchy and appealing name for your application and an attractive logo. The logo should appear on all the interaction screens at one of the corners or as a background.

5. Technical Requirements

a. You may use Sun Micro systems Java related support for Web services or Microsoft's .NET support.
b. Use a suitable container (JBoss, Tomcat, JSAS and other) for deploying the service and the application should be portable across servers.
c. For the data layer use any well-known relational data base (DB). You may test your early development with a Java DB or local DB. However it is preferable that the final version store the data in the Oracle DB.
d. For the business layer you may use beans, enterprise java beans (latest version) or equivalent in MS .NET.
e. In general, follow strictly the naming convention of Java. You will name the packages according to these naming convention:

   Web services: org.yourname.ws
   Web services client: org.yourname.wsc
   Forms: org.yourname.forms
   Web elements: org.yourname.web
   Where yourname is your user name; for a group use both your usernames with a hyphen in between.
f. Create a distribution of your application that is runnable on other containers. For demo purposes you may also want to create a distribution with Java DB on your lap top.
g. Use resource references for referring to data sources and other container enabled entities such as message queues.
h. You may use REST-based[7] interactions also.

i. Rich client interfaces can be provided using such software as Macromedia Flash and related software.
j. You may use javadoc and tools such as doxygen[8] to auto generate programmer's manual that can be added to your project report.

6. Design and Implementation

Begin the design with a system model that gives a rough sketch of the various operational blocks. Then refine it to obtain the architectural model. You may use UML [9] use cases to discover various modules that you want to implement. You may start with data modeling and proceed to the user interface in a bottom-up fashion or you may start with the user interface and work on top-down fashion. Pay special attention to container/server configuration at various levels.

7. Report and Submission

You will prepare a professional quality documentation that has a user's manual and a programmer's manual besides a general description of the project. The format of the report is as shown in the enclosed document.

8. Suggested Architectural Model