|  |
| --- |
| UBC MAGIC |
| Coffee Shop Container Application |
| Technical Documentation |
|  |
|  |
|  |
| **4/20/2010** |

**Jay Wakefield**

|  |
| --- |
|  |

Table of Contents

[List of Figures 3](#_Toc259999276)

[1.0 Introduction 4](#_Toc259999277)

[2.0 Server Side Structure 5](#_Toc259999278)

[3.0 Communication Protocols 7](#_Toc259999279)

[4.0 OSGiBroker Integration 10](#_Toc259999280)

[5.0 Event Processing 11](#_Toc259999281)

[5.1 Context Switches 11](#_Toc259999282)

[5.2 Custom Connector Events 11](#_Toc259999283)

[5.3 Java Beans 12](#_Toc259999284)

[6.0 Plug-in Ability 13](#_Toc259999285)

[6.1 XML and JAXB 13](#_Toc259999286)

[6.2 Custom Connectors 13](#_Toc259999287)

[7.0 Future Work 15](#_Toc259999288)

[Appendix A – Main Class Diagram 16](#_Toc259999289)

[Appendix B – Servlet Class Diagram 17](#_Toc259999290)

[Appendix C – JAXB Class Diagram 18](#_Toc259999291)

# List of Figures

[Figure 1 - Communication Servlet Messages 8](#_Toc259999292)

[Figure 2 - SMS Protocol parsing 9](#_Toc259999293)

# Introduction

This report is intended to be a technical description of the development work completed on the Coffee Shop Container application for the PSPI project, and EECE 496. This report is an amendment to the English report which was written as a high level description of the functionality and description of the project. As opposed to the English report, this document contains class diagrams, a description of the protocols used, event processing algorithms, and the design used to make the application highly configurable.

This report continues with the following sections; server side structure, communication protocols, OSGiBroker integration, event processing algorithms, plug-in design, and items to be completed.

# Server Side Structure

The server side logic of the container application has been designed so all information and main functionality and business logic of the application are centered in on main class. As such, there is a “CoffeeShop” class, which has been designed to be the main class for business logic of the application. It holds reference to objects describing pluggable applications that are configured, as well as references to main application configuration properties. It is also responsible for performing business logic for the application that pertains to what application is in viewing context, what is queued to be run, connecting with the OSGiBroker. The actual business logic for these responsibilities will be described in a section later in this report.

Since the CoffeeShop class contains configuration information for the entire scope of the application, it is needed to be accessible to all other classes, to allow them to invoke business logic, interact and potentially change configured information. To do this, the class has been designed with the singleton design pattern, where only a single instance of this class will ever be available at one time during the life of the application. This is important, because all of the information contained in the class must keep state, and therefore we do not want several instances of the same information which can then be changed and possibly would not reflect to other parts of the container.

In addition to the CoffeeShop class, there is a MessageBoard class, which has the only responsibility of holding state with the OSGiBroker for the message board of the container, and providing a method of sending messages to the board. There is also a simple linked list that is used as the container queue, to queue applications that users wish to run on the large display.

Instances of the Application class store information about external applications that have been configured to run in the application. These classes are special, because they are generated by the JAXB Framework, to allow XML unmarshalling. This class and JAXB will be described in more detail later in this report, but for now it is sufficient to realize that they allow the container to easily get information about applications that are configured be able to be loaded to the large display. Having this information in an object is much easier to work with than reading from XML configuration files.

Three main servlets exist to provide a way for the large display and users to interact with the container. The Message Board Servlet contains an instance to a MessageBoard object, and is only responsible for returning new messages for the message board to the large display. The Communication Servlet is responsible for providing GUI configuration and change information. This information includes what application should be running in context, what applications are configured to be able to run, and what applications are in the queue. The communication servlet is the main interface for communication between server and large display. Finally, there is a “CoffeeShop” servlet, which is used for receipt of messages from OSGiBroker topics that have been configured to send messages to a URL. A simple example of this is the SMS topic. When the container subscribes to the SMS topic, it tells the OSGiBroker to send the message to this servlet and from there can be decoded and forwarded to the appropriate part of the application.

Class diagrams of what has just been described in this section can be found in Appendx A and Appendix B.

# Communication Protocols

Two main protocols have been developed for communication. One is intended for communication between the server and large display GUI, and the other for routing SMS messages to their desired location.

The communication between GUI and server happens on a regular basis, and is initiated by the large display. The large display polls the URL of the communication servlet with either no parameters, or a “type” parameter that can either be “configuration” or “queued.” All messages returned in the HTTP response to the large display are in an XML format, so that the large display can simply parse the message and get relevant information. there is no parameter, the server either returns a configuration message if there a new page configuration is a available, or returns a context configuration message if a new application is in context to run, or an empty message if no new information message is available. If the large display asks for a “configuration” type message, the server will return a new configuration message, and if asked for a “queued” message the server will return a list of applications that are currently queued to run. Figure 1 below contains a flowchart describing the decision process as to what message to return.

The actual information that is returned depends on what the large display needs from the message. For example, a context message must contain the URL of the application to run, while a configuration messages includes a list of applications that are configured, along with their names, and display images.

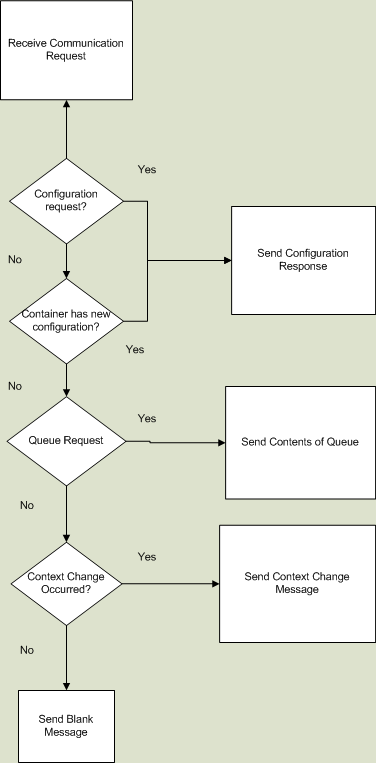


Figure - Communication Servlet Messages

Since there are three possible destinations for an SMS message in the container, we have developed a protocol that should be used when users with to interact with the application via SMS. The SMS message may be intended for the message board, or to select an application to run, or it could be intended for the application that is running. To allow for this, the user must add keywords to the beginning of their message. If they wish to post to the message board, they just send “say <message>.” For example, if they wish to say “Hello World” they must send “say Hello World.” If they are making an application selection, they must send “select” followed by the number of the application they wish to run. Any other message that does not contain these keywords can be assumed to be intended for the currently running application, and is forwarded to the appropriate OSGiBroker topic. Figure 2 below shows this logic in a flowchart.

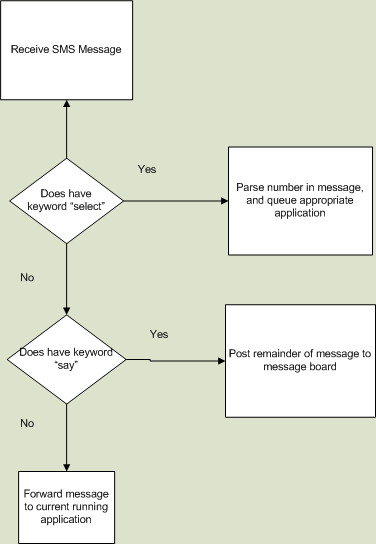


Figure - SMS Protocol parsing

# OSGiBroker Integration

The OSGiBroker plays an integral part the logic of the container application. Each external application is intended to run normally, or how it would even if it was not configured to run within the container. As such, it should use the OSGiBroker as it would normally. However, the business logic of the container also uses the broker to determine how much activity the currently running application is experiencing, and makes application context changes based on that.

We intended to use the OSGiBroker as a monitoring tool because of its subscription and broadcasting capabilities. It provides a simple way for the container to see events that have occurred on any given application’s topic, and allows us to be notified when certain events occur, such as the receipt of an SMS message.

To integrate the OSGiBroker with the container, the CoffeeShop class registers itself with the broker on startup, and subscribes to the appropriate topics that it will use for the lifetime of the application, such as the SMS, and message board topics. During a context change, the container will dynamically subscribe to whatever application is set to run. This process is described later in this report.

# Event Processing

## Context Switches

A context switch is when an application that is running has run the course of its life and needs to be changed. The CoffeeShop object handles this logic, by using a timer and the application’s topic on the OSGiBroker.

When an application is set to run, and a context switch should occur, a few things happen. First, the container unsubscribes from the application that was running previously, and subscribes to the new applications topic. Then, a timer is created and started, that runs for a configurable amount of time. When the timer runs out, it checks for OSGiBroker events that occurred on the application’s topic during that time period. If there were events, the application was being interacted with, and the timer is simply restarted. However, if there were no events, it has been determined that the application is idle, and it is replaced with the application next in the queue of applications to run, or runs the default application if nothing has been queued. The default application follows the same process described as well, the only difference if it will be set to run if no other applications are queued.

## Custom Connector Events

When a custom connector is defined for an application, the OSGiBroker reports the custom application’s events to the Coffee Shop Servlet. This allows us to call the custom connector, and execute procedures whenever something happens on the OSGiBroker. Similarly to the context switches, a timer runs and checks to see if the application has been interacted with in the configured time frame. However, because events are reported to the servlet, there will be no events pending on the OSGiBroker. To deal with this, the servlet informs the timer whenever it calls the custom connectors receipt method. The timer simply checks this flag, and proceeds as described above. For a more detailed description on the custom connector, please see the English report, as mentioned in the introduction of this paper.

## Java Beans

There are a few web pages that have been created and run separately from the large display. These pages are to provide users to interact with the application via HTTP protocol. It also provides an interface for dynamic configuration. The events that occur on these pages are processed by Java Beans, since the pages are developed using the JSF framework. Instead of sending events directly to the OSGiBroker, the beans run methods in the singleton instance of the CoffeeShop object. For example, if a user wishes to queue an application, they may do so on the user input page. The action would invoke the appropriate message on the server side java bean, which would then invoke the queuing method in the CoffeeShop instance. Similarly, posting to the message board via this page is done through the bean and Message Board objects, rather than sending an HTTP message directly to OSGiBroker. This was done to provide consistency, allowing us to interact with the OSGiBroker only via the API, rather than using a mix of API and HTTP protocol.

# Plug-in Ability

The container application has been designed to be extremely configurable, allowing users to configure virtually any web application to run within the container. This is done using XML and JAXB, as well as custom connectors.

## XML and JAXB

JAXB is a framework that has the ability to read XML files, and turn them into structured java classes. This is ideal for our situation, since configurable XML files are created and placed in the classpath for the application when a user defines plug-in applications. Turning these XML files to Java Objects allows us to quickly and easily use the configured information and to differentiate between applications in an object oriented environment. This allows much more flexibility and less latency in the business logic of the container application.

To use the JAXB framework, first an XML Schema was defined as described in the English report. Then, JAXB provides a compiler that reads the schema, and produces classes that represent each element defined. These classes can then be used just like any other java class. These classes can be found in the package named “ca.ubc.magic.coffeeshop.jaxb.”

## Custom Connectors

As described in the English report, we have developed a custom connector interface that should be used if the application developer wishes to perform actions before or after messages are sent or received by the OSGiBroker. This allows ultimate flexibility, since it allows us to user applications that may not work with the OSGiBroker, or does not quite fit with the basic design of the container. It is the intent that this interface forces the programmer to report events to the OSGiBroker, and thus allowing the container’s timing and idle mechanism to perform as if there was no custom connector.

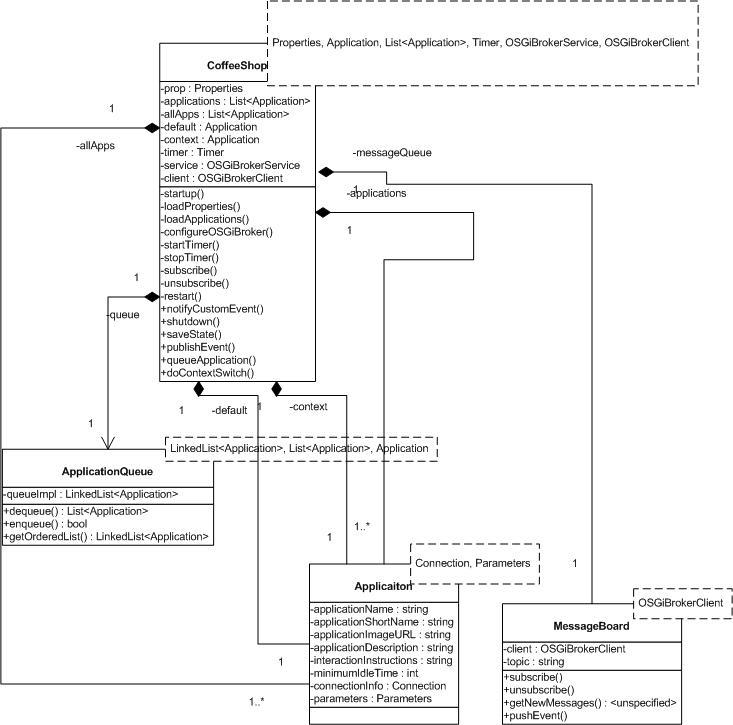
# Future Work

The container application implements all mechanisms required and necessary to plug almost any external application into the large display. However, there will ultimately likely be bugs that are discovered. A description of how to use the container and how to build and run the application is described in the User Guide, which is also provided.

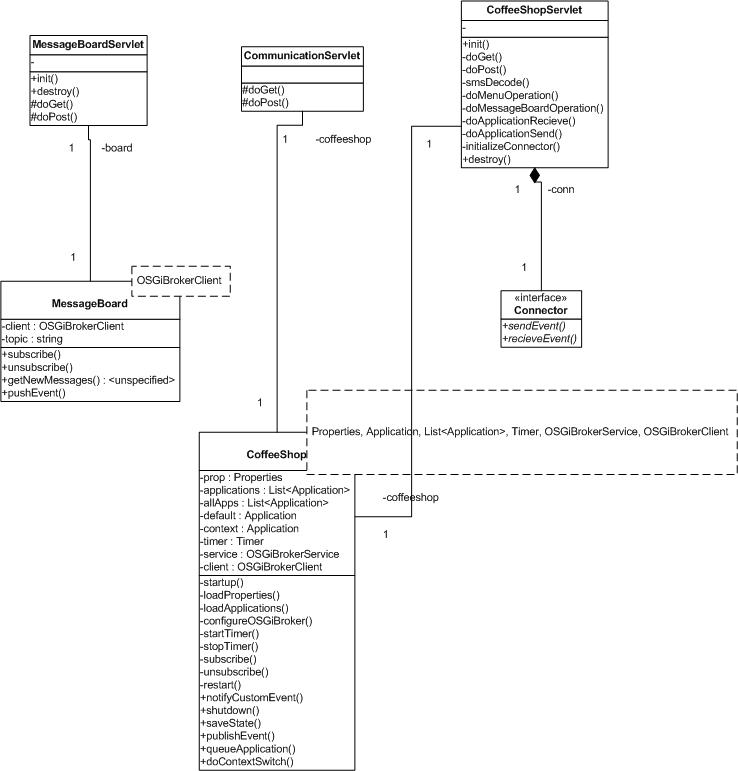
As well, the large display of the application could be greatly improved upon. Since neither student that developed the container has an experience with HCI or design, we have developed the large display to be functionally correct. In the future, changing the design and look of the display would most likely increase the attractiveness of the application.

Finally, any other configuration parameters that we have not thought of could be easily implemented. Perhaps, for example, the instructions or URL listed on the large display could be a configurable value. These parameters could be added to the application XML files, or could be added to the application properties file and easily used anywhere within the container.

# Appendix A – Main Class Diagram



# Appendix B – Servlet Class Diagram



# Appendix C – JAXB Class Diagram

