Project Kilo - Progress Report

Josh Bambrick, Davide Cultrera, Howard Guo,
Rupert Horlick, Raahil Shah & Jerry Yan

February 10, 2015

Contents

1 Current Progress **2**
1.1 Code Written 2
1.2 Testing & Test Harnesses 3

2 Work by Individual Members **4**
2.1 Joshua Bambrick 4
2.2 Davide Cultrera 4
2.3 Howard Guo 4
2.4 Rupert Horlick 4
2.5 Raahil Shah 5
2.6 Jerry Yan 5

Chapter 1

Current Progress

1.1 Code Written

The current source code is our first iteration of a working end-to-end application, that allows us to scan a barcode, send it to a Java WebApp, lookup and parse information from multiple data sources, perform (very basic) NLP and format and display the results on the Glass.

The first module is a refactoring of an existing Glassware, BarcodeEye, that performs the scanning part of our application. Having this resource is very useful, because it allows us to focus on the more challenging part of our application, NLP. We refactored the code to send its result, a barcode number and type, to our Java WebApp for further processing.

The second module, the first part of our Java WebApp, is the application server. This is the code that handles the incoming HTTP request sent by the Glassware, and sends the parsed parameters into the rest of the modules. It creates an ItemInfo object, which is a custom datatype class to hold all of the information about a product. It includes the title of an object, authors/artists, descriptions from multiple sources, and reviews from multiple sources. The server then creates new threads for each data source and allows them to populate the ItemInfo before passing it to the NLP module for further processing.

The third module, already mentioned, is the lookup and parsing module, this has two phases, and the relative importance of each phase depends on the datasource. The first phase is the initial collection of data, performed by making calls to the APIs of our datasources. The second phase is parsing; some of our APIs return complex HTML structures, including embedded widgets and iframes. In order to parse these structures, we have created a dedicated node.js application, because JavaScript is much more powerful when it comes to manipulating HTML. For simpler APIs this application can be ignored and the data can be extracted within the Java WebApp.

Once the third module has bundled all of the gathered data into a single ItemInfo object, the NLP module can operate on this. Right now the module is not very advanced and simply takes the first sentence from a description, along with the title and the author/artist, and passes this on to be formatted for display on the Glass. This module will be heavily extended in the next two weeks.

The fifth module is an HTML formatter, which prepares the information processed by the NLP module for display on the Glass. We have a well defined HTML structure which the Glassware is expecting.

The sixth module overlaps with the first and is the second half of the Glassware. While all of the server-side processing is occurring we will display a loading screen, but once that is completed and the Glassware receives an HTML response we are able to display a new card containing the card the the WebApp has already defined for us.

1.2 Testing & Test Harnesses

Our application is split over several parts and modules within those parts. Each part will require a slightly different testing environment.

For the Glassware we do not currently have formal tests set up, but are primarily relying on running our application on random barcodes from the internet and seeing what our application can produce. The positives from this are that the application does indeed scan barcodes and can successfully display lookup results from the WebApp.

The Java WebApp has its own test harness, built on JUnit, this tests a number of test cases for each module, and this will be extensively expanded as the application is made more complex.

The node.js application has a very basic test harness at the moment, but this will be extended significantly in the coming weeks.

Chapter 2

Work by Individual Members

2.1 Joshua Bambrick

Written a node.js service for parsing web pages and return product reviews contained. This service should be used by the Java server to extract user text from online sources. It interacts with the Java server via standard in and out. The current state of this is that it can detect errors in the request and return a canned response on valid queries (currently just to amazon).

Set up a basic test harness for this system and I would like to improve this by using a proper unit-testing plugin.

2.2 Davide Cultrera

Refactored BarcodeEye Glass app: pruned out unnecessary classes (e.g. the ResultProcessorFactory, and some website-specific look up functions), and readapted existing ones (e.g. ResultProcessor) to allow sending requests to the backend server

Set up Tomcat server on the Student-Run Computing Facility

Tested components of the backend (local server on Vagrant, Maven project)

Researched possible NLP libraries

Defined ItemInfo class

2.3 Howard Guo

2.4 Rupert Horlick

Set up a Vagrant VM running a Tomcat server. This allows us to all run the same server test environment on our own machines and sync it via GitHub. To do this it was necessary to define a Vagrantfile, specifying the nature of the VM, and a collection of Chef cookbooks, used to provision the VM with Tomcat and its dependencies, and install the WebApp produced by our Java application.

Designed the interface of our Glassware, including defining an HTML format for the WebApp to return information in. This allows us to translate from our Java representation to a native Glassware card extremely easily.

2.5 Raahil Shah

2.6 Jerry Yan