Project Kilo - Functional Specification

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

January 27, 2015

Contents

Chapter 1

General Investigation of Problem

1.1 Background

Our original project brief was as follows:

> Google Glass is an amazing product, but can make the wearer seem a little geeky. The goal of this project is to compensate for that, by using the Google Glass camera to identify a barcode on a non-geeky cultural product you are given in a social situation (say a book or vinyl LP), and then provide a private display with cues of clever things to say about it. This will probably come from online reviews, but your software will need to spot the barcode, find the reviews, and distil them into a few words that make the wearer seem as smart as possible.

From this we outlined the problem as follows:

> Create a Google Glass app that will scan a scene, pick out a barcode, determine the barcode number, send this to a server to: determine the type of product, lookup information and parse it, use NLP to process the text and generate concise information of conversational value and finally display this on the Google Glass.

1.2 Points of Complexity

- Barcode scanning
  - Scanning under different conditions (angle, distance, lighting)
- Barcode lookup API integration
  - Using different APIs for different product
- Information aggregation
  - Parsing API return data into a common data format
- Natural language processing
  - Summarising data
  - Picking out most important information
- Presentation of results
  - Preparing the information for display
  - Displaying it on a confined screen size

Chapter 2

Facilities to be Provided

The goal of the project is to deliver a standalone application for Google Glass (a “Glassware”), which does not require installation of paired smartphone apps. The Glassware will provide the user with textual cues about cultural items that have a scannable barcode. In order to offer this functionality, the following facilities will be provided:

- Real-time barcode detection and scanning by capturing camera input
- Automatic UPC/ISBN code search and item retrieval
- Selection of one best source out of a closed set of options, e.g.:
  - Generic search: Amazon, Wikipedia
  - Books: Goodreads
  - Films: Rotten Tomatoes, IMDb (via the OMDb API)
  - More categories may be offered in subsequent iterations, e.g. CDs, video games, wine etc.
- Two types of useful content have been identified, which require different processing:
  - **Product descriptions and plot summaries.** This will be the genre tackled in the first prototype. The baseline approach will consist of extracting the first few sentences of the text; a more sophisticated approach, where viable, will involve automatic summarisation and/or keyphrase extraction, in order to provide more effective and information-dense cues.
  - **Critical/customer reviews.** As a possible augmentation of the basic functionality, the app could offer extraction of most helpful reviews, and graphical display of overall ratings.
- The user will ultimately be presented with informative snippets of texts which should aid him/her to gain a better understanding of the content and social relevance of the item.

Chapter 3

Major Components of the System

3.1 Barcode Scanning

A barcode typically refers to a representation of a number or string as a shape which is easily machine readable. There are many common types, such as Universal Product Code (UPC), a very widely used type, International Standard Book Number (ISBN) which is specific to books and QR codes.

Since this is a very common and useful problem to solve, there is a wide variety of free tools which are already available for the purpose. ZXing, a popular barcode scanning library, will scan a scene in real-time, detect a barcode and return the barcode number and format.

This component of the system will mostly be based on integrating one or many existing libraries into our project so that it will perform to a tolerance. We need to account for many factors such as orientation, distance, luminance and obscurity. Existing libraries will have their own techniques of dealing with some of these things but we will look to optimise them for our purpose and our hardware.

3.2 Information Collection & Parsing

The information collection process is the first step in the back-end portion of the system. It is intended to take a barcode type (UPC, ISBN) and the barcode number and perform a search in numerous libraries on the product and return back a block of text (either XML or JSON format) that contains all relevant information on the product page (reviews, description, rating, product images).

In the first stage of our product, we intend to just implement books, ISBNs, as these will provide a sample for the kind of review and information we would like to see in the finished product. In later stages, this will be expanded to include movies, wines, and various other products. The goal of the completed product is to search through different libraries based on the type of barcode and the type of product. For example, ISBNs will be able to use the GoodReads library as it is most suited for books and provides the most complete information. Each implemented library will have its own specialized case.

Movies Rotten Tomatoes

Books GoodRead or Amazon

Wines Wine.com

Other products Amazon, Wikipedia, Other sources TBD

In this specification, we will outline the relevant methods from the Amazon and Goodreads APIs

3.2.1 Amazon

We will be using the Amazon Product Advertising API (v 2013-08-01) to retrieve information about the product from its barcode number and type. This will be done using REST requests to the API with the ItemLookup operation.

Relevant request parameters include:

Service AWSECommerceService

Operation ItemLookup

IdType [Type of the barcode, for example ISBN, UPC]

ItemID [Barcode number scanned]

ResponseGroup Small,EditorialReview,Reviews (Small gives information like Title, Authors, EditorialReview gives the product description from Amazon, Reviews give some customer reviews).

AccessKey, SecretKey, Timestamp used to sign the request with a SHA-256 HMAC.

The API response html will be parsed for the plain text data of Title, Author(s), Product Description and Customer Reviews which will then be sent on to the NLP module as a POJO or a lightweight interchange format like JSON.

3.2.2 GoodReads

Get the Goodreads book ID given an ISBN

Get the Goodreads book ID given an ISBN. Response contains the ID without any markup.

URL https://www.goodreads.com/book/isbn_to_id (sample url)

HTTP method GET

Parameters **key** Developer key (required).

isbn The ISBN of the book to lookup.

Get the reviews for a book given a Goodreads book id

Get an XML or JSON file that contains embed code for the iframe reviews widget. The reviews widget shows an excerpt (first 300 characters) of the most popular reviews of a book for a given internal Goodreads book_id. Reviews of all known editions of the book are included.

XML responses also include shelves and book meta-data (title, author, et cetera). The Goodreads API gives you full access to Goodreads-owned meta-data, but it does not give you full access to book meta-data supplied by third parties such as Ingram. Book cover images, descriptions, and other data from third party sources might be excluded.

URL https://www.goodreads.com/book/show.FORMAT (sample url)

HTTP method GET

Parameters **format** xml or json

key Developer key (required).

id A Goodreads internal book_id

text_only Only show reviews that have text (default false)

rating Show only reviews with a particular rating (optional)

Given an IBSN, we will be able to gather the most popular reviews of a book in XML or JSON format. The text from this file will then be extracted, which involves collecting the book title, author, description, and reviews. The re-formatted review will then be passed on to the next stage where key words and ideas are extracted and summarized.

3.3 Natural Language Processing

This is the last module of the server-side application. Because NLP is not an exact discipline, it is not possible to predict which techniques will fit best the available information, therefore the approach towards developing this component will be incremental: beginning from a baseline process, the module will be iteratively refined by experimenting with different algorithms and libraries, while the input and output interfaces will stay constant.

As input from the the information parsing component, the module will receive an ItemInfo object, containing product description and reviews, and process them differently. The latter case will be handled if appropriate text mining algorithms are developed and deemed useful.

In the most basic functionality envisioned, the module will output a short text (regardless of the genre of the input text), which is sent to the Glass to be displayed. If deemed useful, another class with different types of information (e.g. ratings, or images) will be added to the interface.

Within the module, more complex processing will take place in various iterations of the product, depending on the quality of the input and technical feasibility given that processing time is a rather strict requirement. In the first prototype, the processing will simply consist in extracting the first few sentences of the text, until a fixed character threshold is reached; this baseline approach is justified as initially portions of texts are statistically very informative, and this heuristic is adopted by extractive summarisers, as well. The first refinement to be made is to implement an **unsupervised extractive summarisation** algorithm: the choice is justified by the fact that Natural Language Generation (required for an abstractive summariser) is too computationally intensive and technically complex, and so is training a supervised algorithm, which is made further unfeasible by the variety of the texts and the amount of hand-annotation work required. When provided with a compiled collection of reviews, a different algorithm than summarisation could be used, such as **key-phrase extraction**, which should return a list of the most salient phrases in the review.

3.4 UI/UX

Google Glass is a high-end product and users expect a polished, natural feel to every piece of software. Google provides documentation on common UI practices and UX principles, which we will need to apply to our application.

First of all Glass operate on the Fire-and-Forget principle. That means a user should be able to get to their content as quickly as possible and then forget about it once theyre done. As such when the user launches the application we want to begin scanning immediately.

The Glass has a very small display area, so it is extremely important to keep our results as concise as possible. We do not want the reader to appear to be reading during a conversation, but we want to provide them with as much information as possible. Thus, we want to supply them with a large number of points, with maximum information, that we can drip-feed to them over time.

Once a user is done with the product we want them to be able to either scan another barcode or leave the app. Glass provides a standard downward swipe gesture for quitting an application, and we can build in a standard contextual menu to return to the scanning phase of the application.

Chapter 4

Acceptance Criteria

In order to be accepted our Glassware needs to perform to the expected level. From our analysis of the problem we would expect the following criteria to suggest success:

- A functional piece of Glassware (i.e. does not crash under normal usage conditions)
- A natural Glassware user experience
- The Glassware should not appear to hang (i.e. should display progress information)
- A mean time from scanning barcodes to displaying results of 8-10 seconds
- The ability to scan barcodes up to 8.5cm away from the glass
- On scanning any ISBN the Glassware should display information that is:
  - Succinct
  - Useful (this is subjective but we can design a usefulness metric)
- All unit tests in the Test Harness should give the expected output

Chapter 5

Management Strategy

We have opted for a horizontal, democratic management strategy, where we reach decisions based on consensus. We will meet as many as times as necessary during any given week, but this will be at least two, two hour meetings. At each meeting we lay down what we want to achieve in that meeting and then discuss all items relevant to achieving those goals. We then divide up the work for that week according to personal preference and expected work time.

For instance, we have split the project into six main modules, but it was decided that four of them constituted the core work and the other two could be kept until later. Of those four we estimated that one would require three people. We each voted for two sections to work on, and then allocated people to one section based on those votes. This was extremely effective and we will use similar systems to divide up work throughout the project.

In order to stay organised we have a couple of systems in place. We use Trello as a central hub for the project where we can store ideas, TODO list and resources. We are using GitHub as our source code management system and JavaDocs for source code documentation. For collaborative, non-programming documents, we are using Google Docs. We use Facebook and mobile phones for more casual discussion and quick organisation of meetings.

Chapter 6

Abstract Classes

6.1 Classes in the Backend

```
public abstract class BarcodeServlet extends HTTPServlet {

  //Handle the POST request getting a barcode and its type and passing
  //it through the steps and adding the results to the response
  public void doPost(HttpServletRequest req, HttpServletResponse resp);

  //Pass the barcode and its type to the LookupHandler object
  public ItemInfo performLookup(String barcode, String barcodeType);

  //Pass the results from Lookup to the NLP module
  public String performNLP(ItemInfo info);

  //Format the results for display on Glass
  public String formatForGlass(String data);

}

public abstract class LookupHandler {

  //Chooses which API to use based on barcodeType
  //and runs the relevant APIClient
  public ItemInfo chooseAPI(String barcode, String barcodeType);

}

public abstract class GenericAPIClient {

final private Vector<String> APIKeys;

  public String APICall(String barcode, String barcodeType);
  public ItemInfo parseResults(String results);

}

public abstract class AmazonAPIClient extends GenericAPIClient {}
public abstract class GoodReadsAPIClient extends GenericAPIClient {}

// may instead handle this in each specialised client
public abstract class TextExtractor {}

public abstract class Summariser {

  public static String summarise(ItemInfo info);

}

public abstract class ItemInfo {

  public String title;
  public String description;
  public Vector<String> reviews;
  public Vector<String> authors;

}

public abstract class GlassFormatter {

  //Format the data returned by the NLP module for display on the Glass
  public String format(String data);

}
```

6.2 Classes on the Glass

```
public abstract class UserInterface implements GestureDetectorBaseListener {

  //May be a constructor (depends on how glass works)
  public void launch();

  //Display a message on the screen (time out or be swiped away)
  public void displayMessage(String msg);

  //Respond to gestures (e.g. swipe down to hide)
  public abstract boolean onGesture (Gesture gesture);

  //Scan for a barcode and send that image to the scanner
  //probably called repeatedly on a timer
  private void findNewBarcode();

  //Request the server with the barcode
  public String doPostRequest(String barcode, String barcodeType);

  //Receive information from the server and request to display it
  private void processItemInfo(String info);
}

public abstract class BarcodeScanner {

  //Receives an image which may, or may not, contain a barcode
  public void tryImage(Image img);

  //Find a barcode in the image (‘null‘ if none found)
  private String findCode(Image img);

}
```