**Software Design Specification**

WikiMapper

Ashley Revlett

Everett Rush

Sarah Mullins

April 24, 2014

Version 2

**Software Design Document Revisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Revision** | **Editor** |
| 2/20/14 | Initial draft | 0 | Ashley Revlett |
| 4/24/14 | Major revisions to reflect current design | 1 | Sarah Mullins |

**Table of Contents**

[Introduction](#h.mndw8sauqjxn)

[Purpose](#h.gupk2y2x3sny)

[Scope](#h.aw9h6286ya3k)

[Definitions, Acronyms, Abbreviations](#h.46i2g9o7ynnh)

[Design Overview](#h.2p3yl59phb6e)

[Description of Problem](#h.vd335mrz8d0u)

[Technologies Used](#h.jdpu72nt06s7)

[Constraints](#h.ou7wg0nm0bdj)

[System Architecture](#h.eum3ow4r67tq)

[System Operation](#h.yxw4qd64kl3l)

[Detailed Design](#h.bid4lemixdjm)

[Display Module](#h.xh1oskety8n8)

[Controller Module](#h.j9lv7euy40l6)

[Model Module](#h.ecrlbsqchofj)

[Outside Libraries](#h.2svavjsc23l8)

# Introduction

## Purpose

The purpose of this document is to describe the implementation of the WikiMapper software as described by the WikiMapper Software Requirements Specification, version 2. The WikiMapper software allows users to visually navigate related topics in Wikipedia.

## Scope

This document describes the implementation details of the WikiMapper software. The software consists of one major process – a user enters a search term, then views a node-edge graph of the term they entered connected to other related pages. The user can click a node to display additional related nodes, and click the node again to view more detailed text and larger image for that topic.

## Definitions, Acronyms, Abbreviations

**Edge** – A connection between nodes, depicting a “related” relationship. Created based upon links found in a keyword/page’s text.

**Graph** – For our purposes, the collection of nodes and edges which depict a keyword’s related content, and is displayed by the UI.

**Keyword**– The word entered by the user into the search input box

**Node** – A vertex in our graph of content representing a keyword. Depicted in the UI by a circular image and label.

# Design Overview

## Description of Problem

Our challenge is to depict the relationship between content on Wikipedia in a visually stimulating manner, and to allow users to discover new information in a novel way, and to share that information with their friends. Figure 1 depicts the major use cases covered by the application:

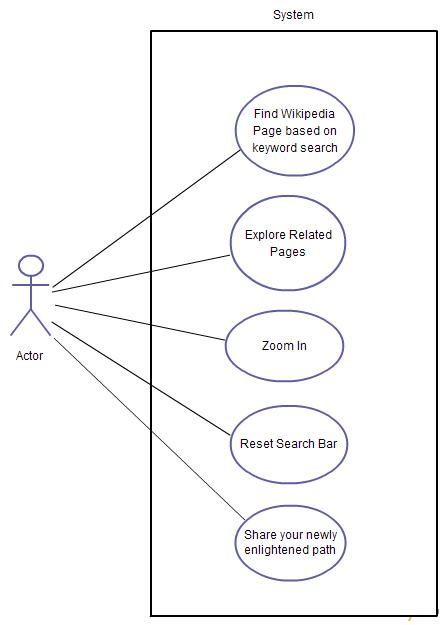


Figure 1: WikiMapper Use Cases

## Technologies Used

HTTP will be used to connect to Wikipedia. Python and the Kivy framework will be used for the application logic and GUI. Kivy also uses Cython for resource-intensive modules. The Beautiful Soup library will be used for text parsing. For target platform, please see WikiMapper Software Requirements Specification.

## Constraints

Because our data comes from a network connection to Wikipedia, our application’s performance is reliant upon the network’s performance. To maximize the responsiveness of our application, we will use asynchronous network calls to retrieve the data, and continue the UI event loop in a separate thread. New nodes are to be displayed incrementally, as they arrive via the network.

## System Architecture

Figure 2 depicts the high-level system architecture. The system is constructed from three top-level packages: Display, Model, and Controller. Those packages contain the classes below:

* **Display** – The UI for the sytem; draws the GUI and the Graph representation
* **Controller** – The main business logic for the system
  + **Network** – An interface for interacting with network resources
    - **Page** – Class used by the Network to represent page data
  + **TextParser** – A module to encapsulate text parsing functionality
* **Model** – The model for the data we show
  + **Node** - Vertice in the models’ graph
  + **Edge** - connection between 2 edges in graph

Figure 2 illustrates the relationships between these components:

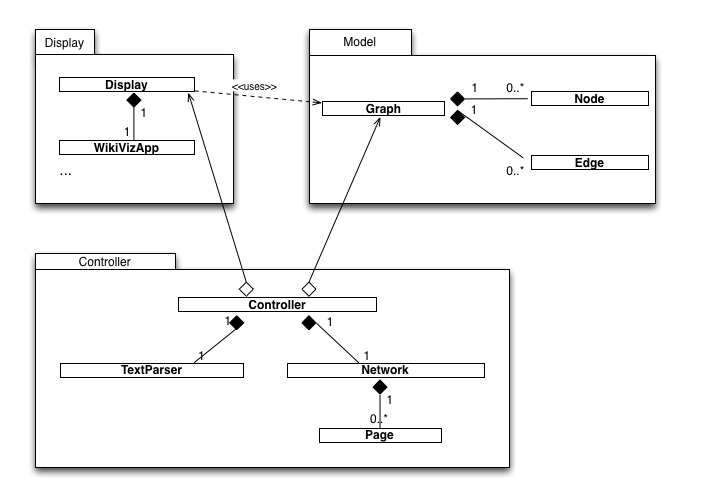


Figure 2. WikiMapper Architecture

The details of these classes are explained in part 3 of this document.

## System Operation

Figure 3 is the typical sequence of events that occur during an initial WikiMapper session. Asynchronous network communication is used to update the model’s Graph. The Display is notified when the update occurs, at which time the Display updates its drawing of the nodes.

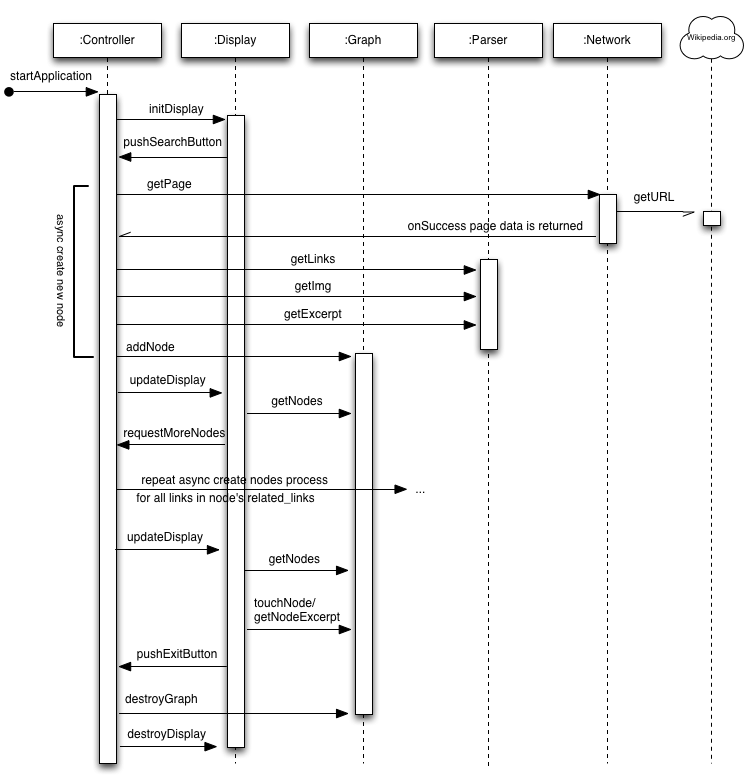


Figure 3: WikiMapper Sequence Diagram

During this session, the application can enter several states:

* Waiting
* Retrieving
* Searching
* Requesting
* Error
* Erasing
* Drawing
* Parsing
* Closing
* Resetting

Figure 4 depicts the states and transitions that occur during a WikiMapper session.

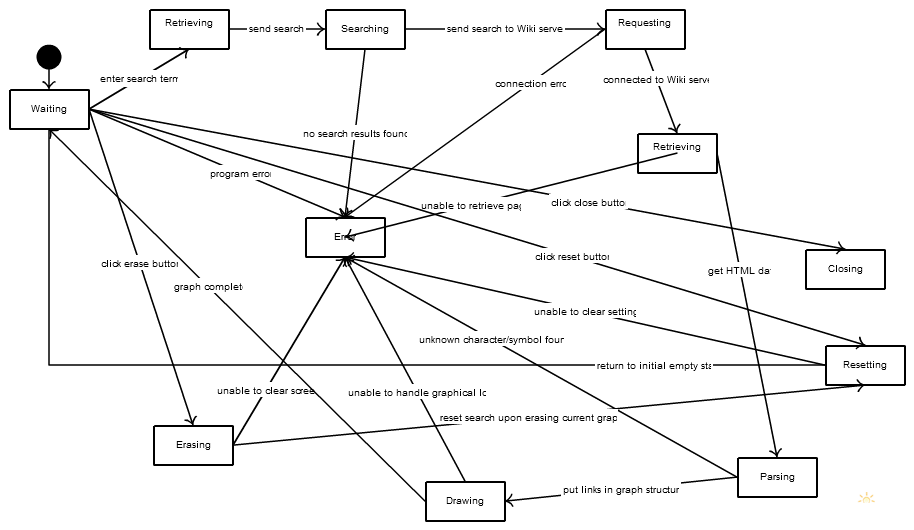


Figure 4: WikiMapper State Diagram

# Detailed Design

## Display Module

The Display Module’s responsibility is to:

1. Draw the user interface
2. React to user events (taps, button presses) by notifying controller
3. Read data from the Model and display it on screen

The top-level Display class is a singleton which encapsulates all Display logic. It manages several Kivy-specific UI classes, along with custom Node classes for the application.

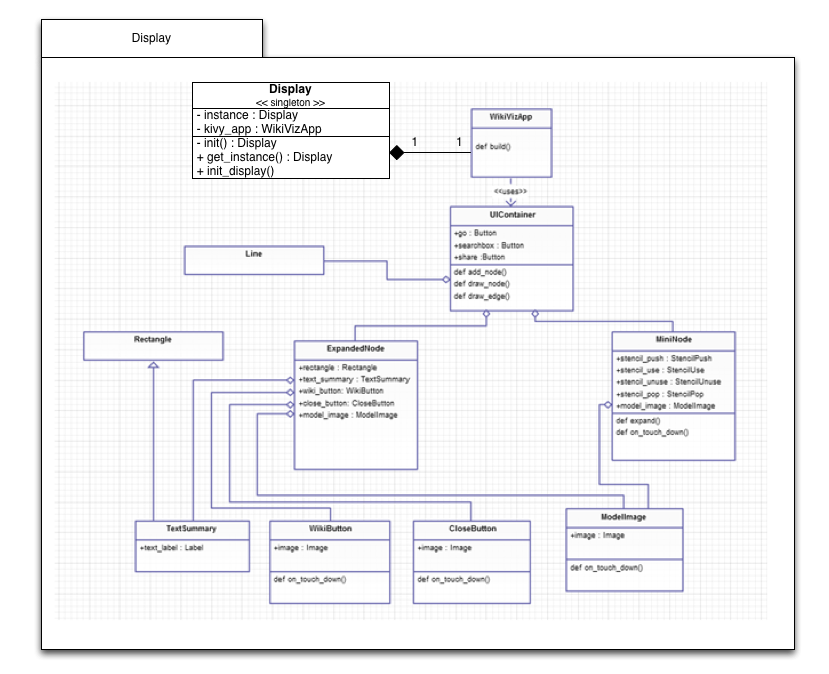


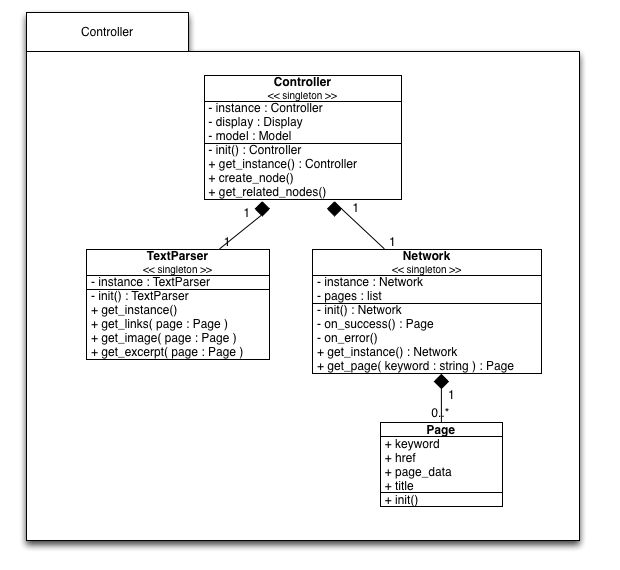
Figure 5: Display package design

## Controller Module

The Controller Module’s responsibility is to:

1. Manage the application lifecycle
2. Coordinate the function of the Network and Parser
3. Update the Model

The top-level Controller class is a singleton which encapsulates all Controller logic. It owns the Parser and Network class. The Network class owns the Page class, which is used to manage the data returned from the network before it has been parsed. The Parser class uses the BeautifulSoup library to parse HTML.

Figure 6: Controller design

## Model Module

The Model Module’s responsibility is to:

1. Store and provide a consistent interface for our application data
2. Protect data from unintended alteration

The Model package encapsulates the Graph structure we’re using. It includes classes for a Graph singleton, Edges and Vertices. The graph is maintained with 2 structures, a list of vertices, and a list of edges as vertice tuples. The Node structure contains all information required to display and interact with the node.

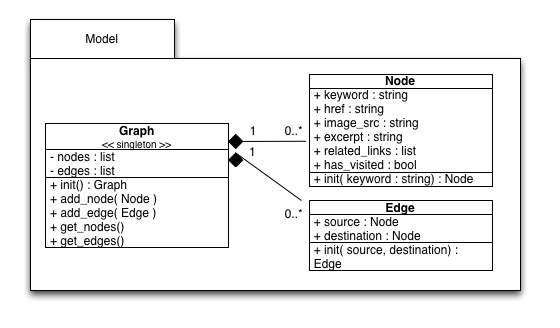


Figure 7: Detailed Model design

# Outside Libraries

Kivy Python Framework – <http://kivy.org/>

BeautifulSoup – <http://www.crummy.com/software/BeautifulSoup/>