UMLTool

An online UML modelling tool.

umltool.heroku.com

Alex Cowell

BSc Computer Science

Submission Date: 6th May 2011

Supervisor: Dr Graham Roberts

Executive Summary

This project aims to design and develop a web-based UML modelling tool using HTML5/CSS3, JavaScript, and Ruby on Rails. It should support class, sequence, use case, and activity diagrams and should allow offline working, provide diagram management features and have a sleek yet intuitive and user-friendly interface. UMLTool aims to blur the line between a simple web application for drawing, and a complex, feature-rich, UML modelling desktop program (such as MagicDraw UML).

After an initial literary review into the problem domain, initial requirements and use cases were gathered and developed. Analysis of the requirements lead to working out ways to implement some of the key features of the application and also identifying those which would be out of scope for the project. Development followed an iterative process, where the next requirement to be implemented was chosen based on the value it would add to the application. Many Agile methodologies were also made use of in order to make the most out of the time available.

This project was a little too ambitious in its scope, and only class diagrams are supported by the application. However, UMLTool was designed and developed in such a way that it would be simple to extend the core functionality too cope with other UML diagram types. Taking the application offline was a major challenge, but one that was successfully completed, allowing the user to edit UML diagrams any time, anywhere.

Contents

1. Introduction

Computer Scientists at every level, from students to professional software developers, will require the use of a UML modelling application at some point. While many will turn to a trusted desktop program when such a need arises, this is not always the most convenient nor efficient way to create a UML diagram.

Perhaps someone wants to quickly whip up a fairly abstract, whiteboard-style sequence diagram to get an idea of how a particular feature will work, or maybe they need a set of detailed, complex UML diagrams modelling an entire system's behaviour to distribute to their team members. There is a gap in the market for an online UML tool capable of adapting to such a wide spectrum of user needs which is also lightweight, instantly available any time, anywhere, and rich with forward-thinking features and cutting edge technologies.

This project aims to create such an application.

UMLTool is web-based in order to be as convenient and accessible as possible, while also supporting offline working so it can be used any time, anywhere. It has an intuitive user interface, allowing for quick and easy UML diagram creation and editing, and lets the user decide the level of detail they would like to work to. It is built using the advanced, powerful features of technologies such as HTML5, JavaScript and Ruby on Rails, allowing it to be both lightweight and fast.

Throughout the duration of this project, one of the main challenges has been simultaneously learning JavaScript and Ruby on Rails, two technologies which were completely new to me at the outset, and balancing development time between the two. In addition, developing the application so that it would work exactly the same when the user was offline proved to be a major obstacle, but one which was later overcome.

1.1 Aims and Goals

Aims:

- Conduct a literary review, reading into the area, researching the problem domain and related issues within the field. This is to get a good grasp of what features are expected from an application such as this and to gain an understanding as to how they are currently being implemented.
- Define initial functional and non-functional requirements based on results of requirements gathering and target user consultations. UMLTool will be developed using Agile methodology, so the requirements will likely change as the project proceeds. However, having a set of requirements defined to begin with focusses development and gives an overview of the project's scope.
- Search and review various technologies that would be suitable for the project and could facilitate development of the project. This is to ensure that the right tools for the job are used: lightweight, fast, yet powerful and flexible technologies.
- Construct small, experimental programs to prototype required functionality. This is done to both validate the scope of the project and to gain experience with the chosen technologies.
- Learn the technologies identified from the technology review. This is a continuing objective which will allow me to get the most out of the chosen technologies and ensure that the product will be finished to a high standard.
- Iteratively develop using Agile methodology in order to make the most of the time available and keep the final vision in mind with regular releases of working software.

Goals:

- Design and develop a fully functional version of UMLTool, supporting Class, Use Case, Sequence and Activity Diagrams.
- UMLTool should allow offline working, provide diagram management features and have a user-friendly interface.
- UMLTool should blur the line between a simple web application for drawing, and a complex, feature-rich, UML modelling desktop program.

- Documentation for UMLTool including a User Manual and a System Manual.
- An evaluation of the results of UMLTool.

1.2 Report Overview

Chapter 2 will cover the background work and research into the field that was undertaken prior to starting work on the project. The sources of information that were drawn upon will be outlined and referenced, followed by a survey of similar software, encompassing both desktop programs and web-based applications, to see what can be learned from some UML modelling tool stalwarts. This survey will also try to pick out mistakes and potential pitfalls to be avoided.

Chapter 3 will present the detailed problem statement for this project and break down the requirements and use cases that have been identified. Chapter 4 will analyse the gathered requirements and extract information from them such as potential database models or the different views that will be needed. There will then be analysis of the key features as determined from the requirements and a technology review to search for and identify specific code libraries, frameworks and software to be used.

Chapter 5 will focus on the actual design and implementation of UMLTool, first giving a high level overview of how the application works, then concentrating on the reasoning behind certain design choices and how they were implemented. Chapter 6 will then examine the testing strategies used for unit, functional and acceptance testing throughout development.

Finally in Chapter 7 the results of the project will be evaluated, addressing the aims and goals set out in Section 1.1, and possibilities for future developments will be outlined.

2. Background Information and Related Work

In order to better understand the problem domain and scope of the project, a literature review was conducted which explored both relevant issues within the domain and features that would be expected from a tool such as the one this project aims to develop. This is then reinforced by conducting a survey of similar applications, addressing their key features, and strengths and weaknesses.

2.1 Literature Review

- UML modelling:
  - » www.agilemodeling.com
    A fantastic resource for both Agile methodology and the construction and breakdown of UML diagrams.
  - » www.ibm.com/developerworks
    The "UML Basics" series of articles gave a great insight into how an industry standard UML modelling tool, IBM's Rational Rose, approaches UML diagrams.
- JavaScript and web development:
  - » developer.yahoo.com
    The Yahoo! Developer Network is a vast sea of useful information on best practices, design patterns and learning resources, specifically the YUI Theater where I learned JavaScript by watching the two series of videos by Douglas Crockford, "Crockford on JavaScript" and "The JavaScript Programming Language".
  - » developer.mozilla.org
    The Mozilla Developer Network has in-depth articles and documentation on HTML5, JavaScript and AJAX, providing a voluminous, valuable resource for current trends in web development.
  - » diveintohtml5.org
    Presents a very accessible and easy to understand collection of articles on all the new HTML5 features and links to further reading for each topic. These

articles were mainly referenced when analysing the advantages and disadvantages of the HTML5 canvas element, and when assessing how to implement offline working with local storage.

2.2 A Survey of Similar Applications

Here we will compare similar existing applications, both desktop and web-based, in order to determine where UMLTool will fit in. To begin with, we shall define the assessment criteria used to survey the applications. These will be the expected features that UML modelling tools might support[1].

- Diagramming – Creating and editing UML diagrams.
- Round-trip engineering:
    - » Code generation – Deriving part or all of the source code for a software system given a set of user-created UML diagrams.
    - » Reverse engineering – Deriving model data and UML diagrams given source code of a software system.
- Templating – Creating and loading from model templates.
- Collaboration – Working with other users on the same model or diagram.

These features are common across UML modelling software regardless of the platform or technology the application is developed on/with.

For brevity, 4 applications of different types have been examined in detail here, but many more were looked at during research (for example, Microsoft Visio, gModeler, EasyUML Editor and more).

2.2.1 MagicDraw UML 11.5

MagicDraw is a desktop program developed in Java and is available cross-platform. The price ranges from $149 for a Personal License, to almost $6,000 for a Teamwork Server License and has a minimum disk space overhead of 500 MB for a personal installation.

1 Modified from the "Kinds of Functionality" section of the "UML tool" Wikipedia page. http://en.wikipedia.org/wiki/UML_tool

Test computer specifications:

- Dell OptiPlex 980
- Intel i5 CPI @ 3.47GHz
- 4GB RAM
- NVIDIA GeForce GT240

Key Features:

- Supports all UML diagrams as well as a multitude of other diagrams for business and other systems modelling.
- Reverse engineering source code to automatically generate UML diagrams (from Java, C++, C#, and more).
- Generate source code from UML models.
- Real-time collaboration through using MagicDraw's Teamwork Server.

Pros:

- The page size automatically increases as needed.
- The main sidebar is very compact yet clear. The context sensitive menus that appear when different objects are selected are very useful and reduce the time needed to create a diagram.
- The quick diagram layout tool is very handy in some cases. It allows the user to quickly arrange the diagram in a number of ways to keep it organised and well presented.
- The round-trip engineering facilities that MagicDraw provides, which are also integrated with popular IDEs like IntelliJ IDEA, NetBeans, Eclipse, and many more.

Cons:

- Can become unresponsive for short periods of time in places, even on the high end system it was being assessed with. This happens, for example, when scrolling around a diagram or zooming.
- Trying to move objects sometimes selects a component of the object (for

example, some text) and drags that instead, creating new, unwanted diagram elements.

- It is not initially obvious how to do certain tasks, such as adding attributes to objects or multiplicity to associations.
- There's some “dialogue syndrome” where in order to perform certain tasks or edit certain properties, a series of dialogue boxes are needed when such a convoluted path is really not necessary.
- The developers' mantra of “all major commands are reachable through a single click”[2] means that the screen is quite cluttered and there is not too much space left for the diagram.

Conclusion:

When evaluating an application such as MagicDraw, it quickly becomes clear that there is a large difference between a fully fledged UML modelling tool and a UML diagramming application. Overall, MagicDraw does a lot of things right and meets every criterion set out in Section 2.2. However, the drawbacks of having to pay for a license, needing to install the software before it can be used and the large disk space overhead are quite substantial.

2.2.2 Creately

www.creatly.com

Creately is an online diagramming tool that allows users to create flowcharts, mind maps, UML diagrams and more and is built with Flash. It is free to use (some features require user to create an account) or you can upgrade your account to remove certain restrictions (from $5) or alternatively you can buy a desktop version for $75.

Key Features:

- Supports UML class, sequence, use case, state, activity, object, deployment, and component diagrams.
- Clean, simple user interface with context sensitive menus and toolbars.
- Work offline and have diagrams sync with your account when connected again (when using the desktop version).

2 http://www.magicdraw.com/what_is

- Share diagrams with others aided by full revision history being available – but no real-time collaboration.

Pros:

- Can export diagrams to JPG or PNG format.
- Can customize the objects/shapes library to your needs. Integrated search for shapes or images in libraries or the web.

Cons:

- Simple tasks such as moving an object around the diagram or scaling/resizing an object is very slow and can occasionally “break” the diagram, stopping the user from being able to properly position other elements.
- Trying to move or reposition lines will sometimes result in unwanted control points being added which are then not very easy to remove.

Conclusion:

A bulky, slow Flash-based web application. Slow load times and unresponsive interactions blight this tool's user experience. Compared to MagicDraw, Creately is certainly more of a drawing application with some UML symbols added in.

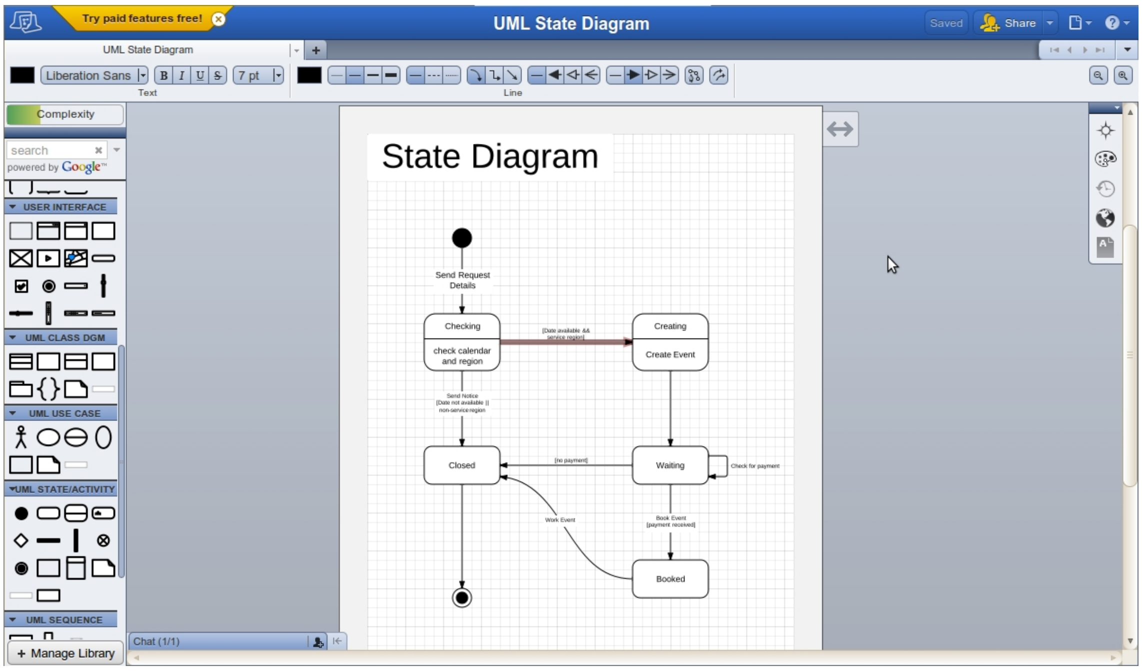
2.2.3 LucidChart

www.lucidchart.com

LucidChart is a relatively new diagramming web application, developed with HTML5, CSS3 and JavaScript. As a free user you are limited in the size of diagrams you can create and upgrading your account costs from $5 to $25.

Test computer specifications:

- Ubuntu Linux 10.04
- Intel Quad Core @ 2.66GHz
- 2GB RAM
- NVIDIA 8800GTX
- Web Browsers: Firefox 3.6; Opera 11.01, Chrome 11



Key Features:

- It officially supports UML class, use case, state, activity, sequence, and component diagrams out of the box, but one could probably make any UML 2.0 diagram using the large library of objects and shapes provided.
- Real-time collaboration with unlimited users (if you pay for it, 2 otherwise), with built in group chat akin to MSN or some other such instant messaging program.
- Intuitive, clean interface with dynamic toolbars that change depending on what you are currently doing/what object/type of object you have currently selected.
- Changes are automatically saved and backed up so you never lose your work.
- Easy to export entire diagrams or only a certain section of a diagram to various formats like PDF, PNG and JPG at various resolutions.
- Integrated search within libraries or across the web for images.
- Fully compatible with all browsers and operating systems.

Pros:

- Access it from anywhere since it is a web application using web standards, so no plugins or anything extra to install. Also compatible with iPad/iPhone and other modern mobile devices.

- Really easy to draw lines/connections by simply dragging one out from the edge of an object. Connections will automatically connect to an object you drag it to and remain fixed when the object is moved around while allowing the user to fine tune the precise positioning of the line on the object. It is also very easy to change the arrows on a line or add multiplicity, roles or labels to associations.
- Objects will snap to the grid, keeping everything neat and tidy. There are also some context sensitive auto-alignment features for certain objects so they can be positioned perfectly.
- Supports the same well known keyboard shortcuts used in many desktop programs and provides a reference of all the supported shortcuts.

Con:

Can be extremely slow and unresponsive, almost to the point where it is unusable – although this does depend somewhat on computer hardware and which browser is being used (Chrome performed much better than Firefox and Opera). Nevertheless, moving even one object around the page can be ridiculously sluggish. With the intention of preventing a similar scenario happening to UMLTool, this problem was investigated. Doing a DOM analysis after starting the editor revealed over 1742 HTML elements which took, on average, 9.9 seconds to load.

Inspecting the source code (the source code which wasn't too severely obfuscated, at least) revealed that every diagram element and their sub-components are HTML5 canvas elements wrapped inside `<div>` tags. This is the main reason why the number of elements is so high and why it can be so sluggish to try and move a diagram element (due to all the DOM manipulation needed). A basic class object, for example, is made up of 47 HTML elements, each of which must be accessed through the DOM every time it is moved or edited.

Conclusion:

Despite the obvious issue of poor performance on certain web browsers, this is a very complete, highly polished and easy to use online diagramming tool. The real-time collaboration features especially make it stand out from the rest, and as browsers evolve, the problem of sluggish user interactions will surely disappear.

2.2.4 yUML

www.yuml.me

yUML is a web application built using Ruby on Rails which allows the user to create and publish simple UML class, use case, and activity diagrams. It is free to use but a private copy with full source code can be purchased from $50. The developers created it for cases where whiteboard or back-of-napkin style UML sketches are needed to visually illustrate a point.

Key Features:

- Supports UML class, use case, and activity diagrams with support for state diagrams currently being implemented.
- Instantly export to PNG, JPG or PDF.

Pros:

- Once created, a unique link to the diagram is provided, so the user can go back and edit it at any time.

Cons:

- Need to learn new syntax for each diagram type which is not very readable or particularly easy to understand just by looking at the source for each diagram.
- Once submitted, all the work to generate the diagram happens on the server side, which can be fairly slow (10-20 seconds for a class diagram with a handful of elements). This also means that there's no way to fine tune the layout, since that is all worked out by the server.
- There is no way to add attributes or methods to classes in a class diagram.

Conclusion:

A pretty decent, fun UML diagram generator. But that's exactly all it is. A generator. You need to learn the markup and then once the diagram is generated, you can't edit elements easily or change the layout, so the lack of user interaction after the diagram is generated hurts the usefulness and productivity of this tool.

3. Requirements

3.1 Detailed Problem Statement

Design and develop a web-based UML modelling tool, focussing on creating and editing UML diagrams but designed in such a way as to allow future extensions for other functionality, such as round-trip engineering. The most basic version of the tool should be aimed at Computer Science students to create simple class, sequence, use case and activity diagrams, while the final version should be a viable alternative for any of the major, popular UML modelling tools.

The tool should be capable of working even when offline, providing the same features as when online, but storing state locally until a connection can be re-established. The tool should allow for some form of user collaboration on diagrams, preferably in real-time. Exporting diagrams to other file types such as PNG, JPG and PDF should be as simple as possible for the user, requiring only a single click to initiate. Users should be able to use the tool without signing up/logging in, but certain features should be restricted. Admin users should be able to manage the user base.

The user interface should be sleek yet simple and intuitive and give the user the feeling of working with a desktop application. The tool should comply with web standards, perform to an acceptable degree across varying hardware and be secure from common exploits and attacks.

3.2 Requirements

The requirements gathering was an ongoing process which took place over almost all of the development period. Being surrounded by your target users almost every day definitely facilitated user feedback on requirements and potential features, as many short but varied user interviews would regularly occur.

Section 3.2.1 lists the final version of the functional requirements and Section 3.2.2 lists the final version of the non-functional requirements. Early in development, the requirements tended to change quite frequently as users' needs were clarified.

The requirements have been prioritised according to the MoSCoW Method in order to easily identify which requirements would add the most value to the project, and implement them.

The capital letters stand for:

- M – MUST have this feature.
- S – SHOULD have this feature if possible.
- C – COULD have this feature if there is time.
- W – WON'T have this time but WOULD like in the future.

Each requirement is assigned a unique ID to facilitate progress analysis and referencing requirements to implement.

3.2.1 Functional Requirements

Supported Diagrams			
ID	Description	Category	Priority
DG1	The system shall allow the user to construct Class Diagrams to the UML 2.0 standard	Diagrams	M
DG2	The system shall allow the user to construct Sequence Diagrams to the UML 2.0 standard	Diagrams	M
DG3	The system shall allow the user to construct Activity Diagrams to the UML 2.0 standard	Diagrams	M
DG4	The system shall allow the user to construct Use Case Diagrams to the UML 2.0 standard	Diagrams	S
DG5	The system shall allow the user to construct State Machine Diagrams to the UML 2.0 standard	Diagrams	C
DG6	The system shall allow the user to construct Component, Deployment, Communication, Composite Structure, Interaction Overview, Object, Package and Timing Diagrams to the UML 2.0 standard	Diagrams	W
Editing			
ID	Description	Category	Priority

ED1	The system shall allow the user to work offline	Editing	M
ED2	The system shall allow the user to edit existing diagrams	Editing	M
ED3	The system shall allow the user to delete existing diagrams	Editing	M
ED4	The system shall allow the user to rename diagrams	Editing	M
ED5	The system shall allow the user to copy, cut and paste diagram elements	Editing	M
ED6	The system shall allow the user to place diagram elements freely	Editing	M
ED7	The system shall allow the user to undo/redo editing steps	Editing	S
ED8	The system shall allow collaborative editing by way of sharing diagrams etc.	Editing	S
ED9	The system shall update local changes upon reconnect	Editing	S
ED10	The system shall automatically expand the drawing area as the diagram increases in size past previous boundary	Editing	S
ED11	The system shall allow the user to snap diagram elements to a grid	Editing	C
ED12	The system shall allow the user to change the scale of the grid	Editing	C
ED13	The system shall allow the user to edit multiple diagrams at once in separate tabs	Editing	W
ED14	The system shall allow collaborative real-time editing	Editing	W
Exporting Diagrams			
ID	Description	Category	Priority
EP1	The system shall allow the user to export a diagram to a JPG/PNG file	Exporting	M
EP2	The system shall allow the user to export a diagram to a PDF file	Exporting	S
EP3	The system shall allow the user to export a diagram to an SVG file	Exporting	C
User Interface			
ID	Description	Category	Priority
UI1	The system shall provide a simple, intuitive user interface	UI	M
UI2	The system shall highlight the currently selected diagram element	UI	M
UI3	The system shall display the currently available diagram elements in a sidebar	UI	M
UI4	The system shall display a menu bar at the top of the screen	UI	M
UI5	The system shall allow the user to drag new elements into the diagram from the sidebar	UI	M
UI6	The system shall support common keyboard shortcuts such as Ctrl/Cmd-Z for undo, Ctrl/Cmd-S for save etc.	UI	S

UI7	The system shall display context sensitive menus	UI	S
UI8	The system shall allow the user to zoom in/out with the mouse scroll wheel	UI	C
UI9	The system shall allow the user to zoom in/out with the plus/minus buttons	UI	C
UI10	The system shall allow the user to view the editing grid	UI	C
UI11	The system shall allow the user to change font styles of text (bold, italic, underline, font size etc.)	Customizing	S
UI12	The system shall allow the user to rearrange UI elements	Customizing	W
UI13	The system shall allow the user to customize the UI's colours	Customizing	W
UI14	The system shall allow the user to change the colours of diagram elements	Customizing	W
UI15	The system shall allow the user to change attributes of diagram elements such as path stroke width	Customizing	W
Saving Diagrams			
ID	Description	Category	Priority
SV1	The system shall allow the user to save a diagram to the server's database	Saving	M
SV2	The system shall allow the user to save a diagram to their web browser's local storage	Saving	M
SV3	The system shall automatically synchronize diagrams stored locally with diagrams stored on the server	Saving	M
SV4	The system shall allow the user to save a diagram to disk	Saving	W
SV5	The system shall allow the user to save a project to disk	Saving	W
SV6	The system shall allow the user to save a project to the server	Saving	W
SV7	The system shall allow the user to save a project to their web browser's local storage	Saving	W
Project Management			
ID	Description	Category	Priority
PM1	The system shall allow the user to create a new project	Projects	W
PM2	The system shall allow the user to merge projects	Projects	W
PM3	The system shall allow the user to add a diagram to a project	Projects	W
PM4	The system shall allow the user to remove a diagram from a project	Projects	W
Printing			
ID	Description	Category	Priority
PR1	The system shall allow the user to print a single diagram	Printing	S
PR2	The system shall allow the user to print a whole project	Printing	W
Help			

ID	Description	Category	Priority
HP1	The system shall provide notifications when a significant event has occurred/is about to occur	Help	S
HP2	The system shall provide some simple tips on getting started	Help	S
HP3	The system shall provide context sensitive tooltips	Help	S
Class Diagrams			
ID	Description	Category	Priority
CD1	The system shall allow the user to edit the class name, attributes and methods of a class element	Classes	M
CD2	The system shall allow the user to resize class elements	Classes	M
CD3	The system shall allow the user to create a new class element	Classes	M
CD4	The system shall allow the user to delete a class element	Classes	M
CD5	The system shall automatically resize class elements upon a user edit to elegantly encompass all data	Classes	S
CD6	The system shall allow the user to resize the inner sections of a class element	Classes	C
CD7	The system shall allow the user to add "getters/setters" to a class element with a single click	Classes	C
CD8	The system shall allow the user to order attributes and methods alphabetically	Classes	W
CD9	The system shall allow the user to create bi-directional and uni-directional associations	Associations	M
CD10	The system shall allow the user to edit the multiplicity of an association	Associations	M
CD11	The system shall allow the user to create aggregation, composition, generalization, realization and dependency relationships	Associations	S
CD12	The system shall allow the user to edit the label of an association	Associations	S
CD13	The system shall allow the user to manipulate association paths by moving control points	Associations	S
CD14	The system shall allow associations to be attached to other diagram elements and stay attached when the diagram element is moved	Associations	S
CD15	The system shall allow the user to edit the direction indicator of association labels	Associations	C
CD16	The system shall allow the user to edit the role of an association	Associations	W
CD17	The system shall allow the user to create and delete package elements	Packages	M
CD18	The system shall allow the user to edit text attributes of package elements	Packages	M

CD19	The system shall allow the user to resize package elements both manually and automatically	Packages	M
CD20	The system shall allow the user to create and delete abstract class/object elements	Abstracts	S
CD21	The system shall allow the user to edit the name of abstract class elements	Abstracts	S
CD22	The system shall allow the user to resize abstract class elements both manually and automatically	Abstracts	S
CD23	The system shall automatically generate a Class Diagram from Java source code	Importing	W
CD24	The system shall automatically generate a Class Diagram from C++ source code	Importing	W
Users			
ID	Description	Category	Priority
US1	The system shall allow the user to create a new user account	Accounts	M
US2	The system shall allow the user to edit their account	Accounts	M
US3	The system shall allow an admin user to delete user accounts	Accounts	S
US4	The system shall allow the user to see all other users	Accounts	C
US5	The system shall create a new session for the user upon login	Sessions	M
US6	The system shall keep the user logged in until they log out	Sessions	S
US7	The system shall clear any data in the user's browser's local storage upon logout	Sessions	S

3.2.2 Non-Functional Requirements

ID	Description	Category	Priority
NF1	The system shall use a web browser as its user interface	Compliance	M
NF2	The system shall be compatible with Firefox 3.5+	Compliance	M
NF3	The system shall be compatible with Safari 4.0+	Compliance	S
NF4	The system shall be compatible with Chrome 5.0+	Compliance	S
NF5	The system shall be compatible with Opera	Compliance	C
NF6	The system shall be compatible with Internet Explorer 9+	Compliance	C
NF7	The system shall perform well across varying hardware	Performance	S
NF8	The system shall be available 24 hours a day, 365 days a year	Availability	S
NF9	The system shall be available offline by way of the web browser's application cache	Availability	S

NF10	The system shall support 100 concurrent sessions	Capacity	S
NF11	The system shall protect against Cross-Site Scripting (XSS) and session spoofing attacks	Security	S
NF12	The system shall protect against Cross-Site Request Forgery (CSRF) exploits	Security	S
NF13	The system shall be accompanied by a User Manual	Documentation	M
NF14	The system shall be accompanied by a System Manual	Documentation	M

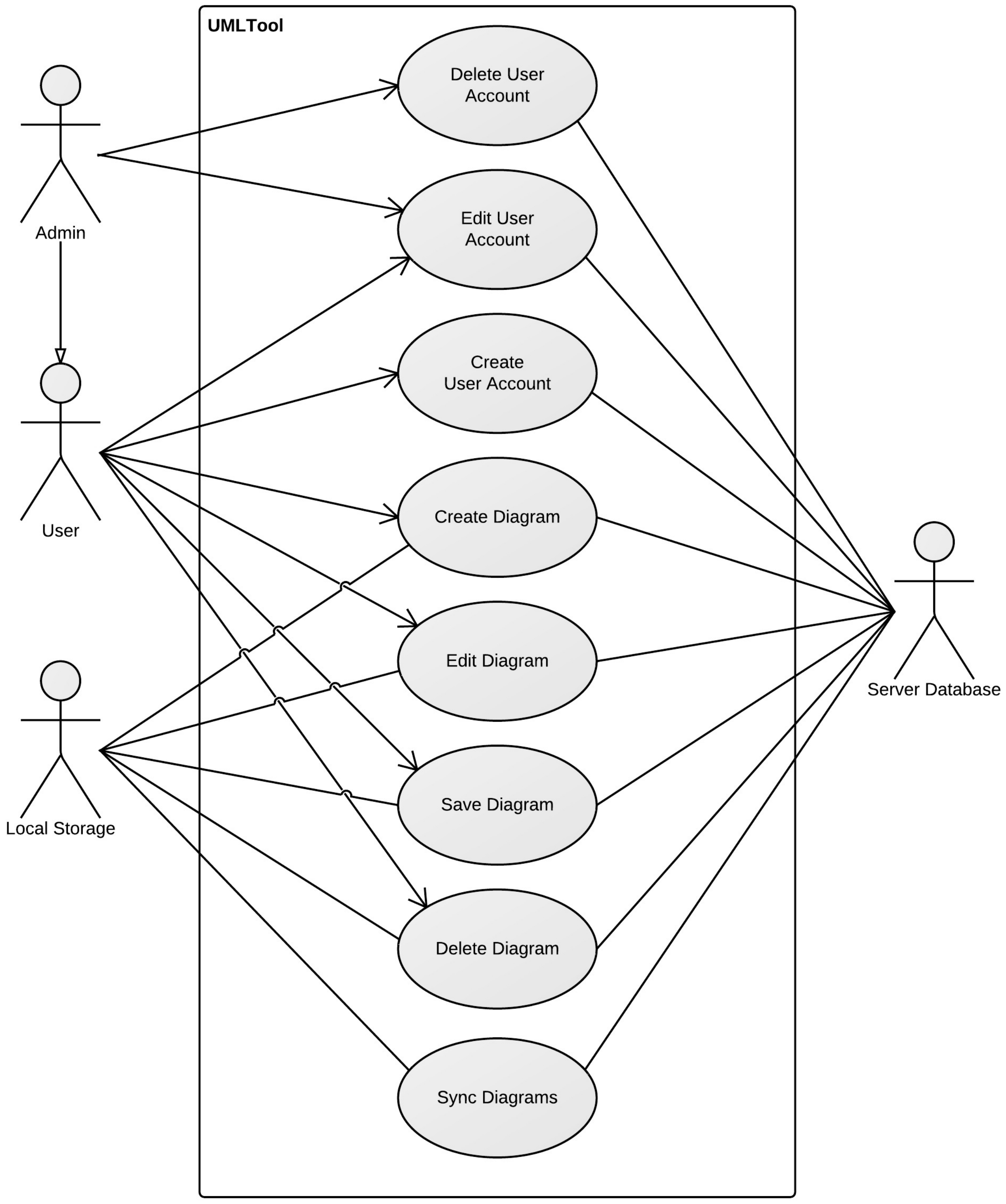
3.3 Use Cases

The use cases were also developed during target user interviews and collaboration, but unlike with the requirements, refinement of use cases stopped once the main implementation had begun. Section 3.3.1 contains a list of use case titles, with the full use cases appearing in the appendix, and Section 3.3.2 contains a use case diagram.

3.3.1 Use Case List

ID	Use Case
UC1	CreateNewUserAccount
UC2	EditUserAccountSettings
UC3	DeleteUserAccount
UC4	CreateNewDiagram
UC5	EditExistingDiagram
UC6	SaveDiagram
UC7	DeleteDiagram
UC8	SyncDiagrams

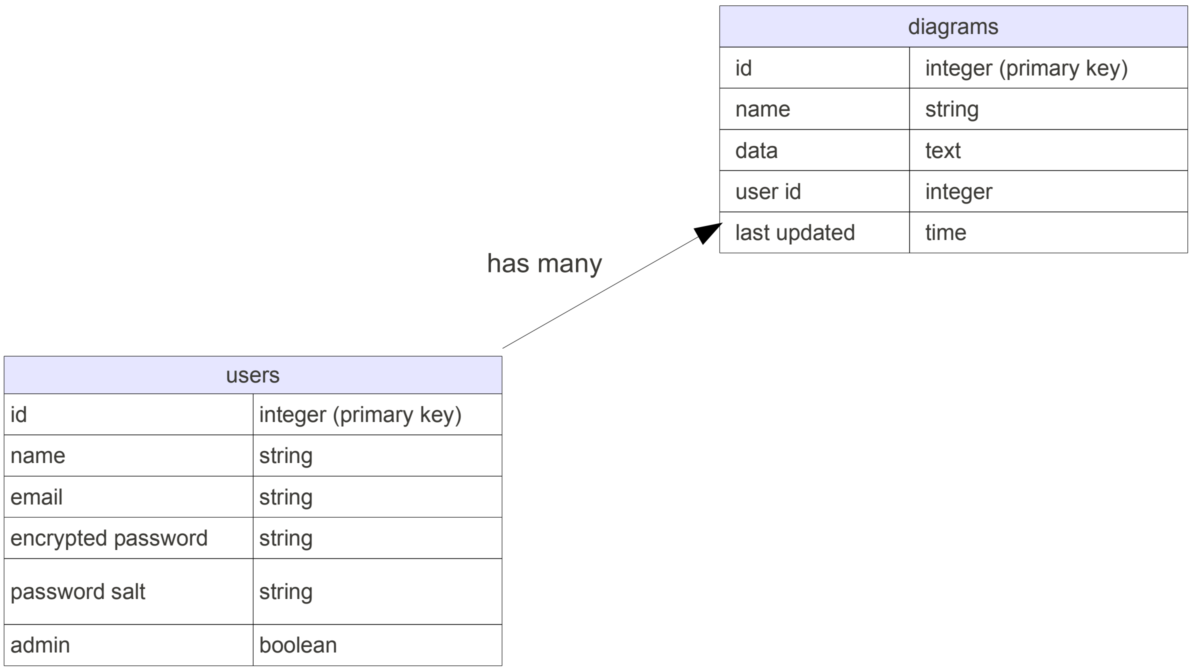
3.3.2 Use Case Diagram



4. Analysis and Design

This chapter will analyse the gathered requirements to extract information in order to determine what data models, technologies and methods will be required in order to implement the application. Section 4.3 will examine the technologies that were chosen for this project based on the results of the analysis and scrutinizing the requirements further.

4.1 Data Modelling



diagrams	
id	integer (primary key)
name	string
data	text
user id	integer
last updated	time

users	
id	integer (primary key)
name	string
email	string
encrypted password	string
password salt	string
admin	boolean

Figure 4.1: Users and diagrams data model and their relationship.

4.2 Functional Analysis

This section analyses key functionality of the application and what will be required to implement them.

Rendering and Editing Graphics

Clearly we will need a way of rendering the UML diagrams on the screen so that the user can see and edit them. My literary review unearthed three possibilities for accomplishing this:

- Flash
- HTML5 canvas
- Scalable Vector Graphics (SVG)

Flash – Reasonable API and widely supported (pretty much everyone who uses the internet has a flash plugin) but it is not native and the overhead is not really necessary for an application like this.

HTML5 canvas – An emerging technology, it is rapidly gaining popularity and is supported in most modern browsers. It has fast rendering capabilities but a canvas is just a bitmap buffer, so you would need to redraw the entire thing when something changes. This makes it undesirable for applications that rely heavily on user interaction with graphical components as everything is drawn with JavaScript commands. Obviously some applications such as LucidChart (as seen in Section 2.1.3) find ways to get around that – but this is not always a good idea as noted during the survey. Canvas is not currently supported in Internet Explorer until IE9, but one can use the exCanvas library to bring the same functionality to IE.

SVG – Supported in most modern browsers (except IE which uses VML) and will be supported in IE9. The crucial deciding point is that you can access SVG elements through the SVG DOM API using JavaScript, so you can add events, dynamic content and manipulate the SVG elements. Since it's scalable, it is ideal for vector image editing which is much closer to UML diagramming than a pixel image editor (using canvas) would be.

Offline Working

Making a web application work offline seems, from the earlier literary review, relatively straightforward. To make resources such as web pages, JavaScript files and images available offline, they are listed in the cache manifest, which is referenced in the HTML. When a user visits a page with the referenced cache manifest, the listed resources are downloaded and stored in the browser's application cache for use when the user is offline.

Similarly, I will need some functionality which stores diagrams locally too for use when the user is offline. This responsibility lies with HTML5 Storage. There are two types of storage: session storage which expires when the browser or window is closed, and local storage which persists even after closing the browser and restarting. Clearly local storage is the winner here – consider the use case where the user creates a diagram offline, then restarts their computer, you'd want the diagram to still be available.

The final consideration for offline working is listening for events that signal the user has either gone offline or come online. Here is a high-level overview of what happens on these events based on the requirements gathering:

- User saves diagram and is online → save locally and send to server.
- User saves diagram and is offline → save locally.
- User is working offline then goes online → synchronize diagrams.
- User loads the page and is online → synchronize diagrams.

There is an issue in using the built-in online/offline events which is that at the time of development, only Firefox and Opera support them, and Opera doesn't support HTML5 drag and drop – which is used to create new diagram elements in the application – and hence won't be supported in this version of UMLTool. Chrome on Windows somewhat supports the online/offline events, but it can be buggy.

Real-Time Collaboration

Another popular requirement, real-time collaboration, is becoming increasingly common in web applications, for example, Google Docs or LucidChart. Incorporating real-time collaboration into UMLTool would definitely distinguish it from the other UML modelling tools – desktop or web-based – as many of them are still single userfacing products. Unfortunately, analysis revealed that this would be out of the scope of this project. There are currently no open source or proprietary real-time collaboration or operational transform libraries for collaboratively editing documents other than text files, so implementing real-time collaboration would require designing and developing a solution from scratch, which would use up far too much development time.

Web Application Framework

In order to easily manage users and diagrams, we would need to use some sort of web application framework to handle user accounts, authorization, diagrams, and server transactions.

4.3 Technology Review

We can now combine what we need from the system's requirements with the analysis results and select the most appropriate technologies for the project. In cases where more than one similar technology would be appropriate, personal preference would be the overriding factor. With that in mind, the following technologies were selected.

4.3.1 JavaScript

Programming Language

JavaScript is referred to as “THE scripting language of the web”[1], which, when I first heard that, rather put me off. I had no experience with JavaScript prior to this project and I must admit I was somewhat thrown by the name, at first taking it to be a simple little language to add some extra functionality to websites. That couldn't be more wrong. After reading into it more I quickly realised it is a very capable object-oriented, functional programming language which would allow me to tap into rich functionality and libraries.

In order to start development as soon as possible and in the right direction, it was imperative that the right code libraries would be selected. One of the main benefits of using JavaScript was also that there is such a wide variety of libraries available.

1 www.w3schools.com/js

Raphael vs. SVGKit vs. jQuery SVG

Raphael – A cross-browser JavaScript vector graphics library using standards compliant SVG and VML (for Internet Explorer) as a base to easily work with vector graphics. As explained in Section 4.2, using vector graphics not only means that elements will scale perfectly and retain their sharpness, but also that each graphical object created is a DOM object, meaning you can target them with JavaScript, listen for events etc. Raphael continues to be updated with version 2.0 coming out soon.

SVGKit – Similar functionality to Raphael but is built as an extension to MochiKit, a large JavaScript framework. Since I was already fairly set about using jQuery, having another large framework in the application was not desirable. In addition, SVGKit hasn't been updated since February 2010.

jQuery SVG – Since jQuery was already a likely candidate to be the main JavaScript framework used in the application, a jQuery plugin for working with SVG could work out well. However, it hasn't been updated since 2007 and jQuery has moved on a lot since then. Also, compared to the interface Raphael provides, this isn't nearly as simple and clean. There is a need to call too many functions to do relatively simple tasks such as drawing a path – compare this to Raphael's easy, intuitive and flexible single method for drawing paths.

jQuery was a name I was familiar with, even though I didn't use JavaScript, which goes to show how popular it has become, and rightly so. It is a fast, powerful, easy to use library that simplifies HTML document traversal, event handling, animating, and AJAX interactions for rapid web development. There is a vast array of plugins available and lots of documentation and resources online to learn from and be inspired by. Other libraries like Prototype, MooTools and Dojo all do more or less the same thing, so personal preference won out on this one. Although after using it, the AJAX functionality of jQuery is incredibly rich and powerful – very useful in a project such as this! jQuery was used on everything from animations and event handling to DOM traversal and AJAX interactions.

One of the more popular requirements was to be able to quickly export a UML diagram in UMLTool to an image file (preferably with a single mouse click). This clearly requires some way of taking the vector graphics and rendering it to an image format. There wasn't much to choose from, most solutions were server based and required large Java libraries or similar to run the conversions. Thankfully there was one client-side solution in the form of canvg, which takes SVG (no VML support however), and renders it onto an HTML5 canvas. The canvas element has a toDataUrl() method which can then be used to get an image representation of the canvas.

4.3.2 Ruby on Rails

Web Application Framework

Rails is a flexible, scalable, web development framework built on the Ruby programming language. It has a RESTful architecture making structuring web applications a simple process. I used Rails as a wrapper for the core of the application (the UMLTool JavaScript libraries) in order to handle user management, diagram modelling and storage, session tracking etc. Lots of plugins (gems) etc. mean practically whatever you want to do has already been done and you can just download and include it.

Other possibilities were Django, which is a Python-based (a language I'm somewhat familiar with) web development framework. And CodeIgniter which is PHP-based (I have little experience but there are huge amounts of resources for it). Rails is highly scalable though, one only has to look at websites such as Twitter, Hulu, GitHub etc. to see that.

I had heard good things about a Test-Driven Development library for Rails called RSpec – specifically a Behaviour-Driven Development tool for Ruby, which proved most useful in testing the application. RSpec was also recommended in many of the Rails resources I had consulted/was learning from.

I also used the Ruby Version Manager (RVM) to work with multiple Ruby environments during development so new updates etc. can be developed for separately, before combining into the main production version.

4.3.3 SQLite and HTML5 Storage

Databases and Storage

SQLite is the default database engine that comes with Rails. Its main goal is to deliver a simple database engine and, as a result, SQLite is small, fast and reliable. This makes it a perfect choice for managing an embedded relational database during development. Since Rails abstracts database interaction through the use of its Active Record library, the application can be developed with SQLite locally, then put into production with a different DBMS more suited towards higher traffic, concurrent actions and so on.

HTML5 Storage is used to store diagrams locally, so the user would have access to them while offline. While limited to 5MB of storage, this should be more than enough for almost all cases. A medium sized UML diagram requires 5-10KB of storage space, which implies an average of around of 500 UML diagrams which can be stored locally by a single user at any point in time.

There are a couple of possible alternatives that were researched. The Web SQL Database API was designed to provide a thin JavaScript wrapper around a locally stored SQL database running on SQLite (essentially sending SQL statements as strings via JavaScript). However, this was not widely supported, most notably by Mozilla, and has now been abandoned. Mozilla instead proposed the Indexed Database API which simplifies the programming model for interacting with databases and has been gaining support from all the main parties.[2] Either of these solutions would be better than using local storage as they are far better suited to storing large amounts of structured data like UMLTool uses to store diagrams.

The web browser's application cache is used to store resources like web pages, JavaScript files and images so the application will display and function correctly offline.

2 Beyond HTML5: Database APIs and the Road to IndexedDB by Arun Ranganathan, 1 June 2010 http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/

4.3.4 Git

Version Control

Version control is essential to a project like this, so it was basically a choice between subversion and Git. While I have lots of experience with subversion, the Rails community almost entirely uses and champions Git, so I went with that. I adjusted quickly to Git and found it was really easy to use.

4.3.5 Heroku

Web Application Hosting

Heroku is a cloud platform for Ruby applications which enables rapid application deployment. I was originally going to host UMLTool on a virtual server in the UCL Computer Science department, but after using Heroku for a while, it seemed great on its own, so there was no need to work out a separate solution. Heroku uses a PostgreSQL database and free users only get 5MB of storage – but for application evaluation, that should be good enough.

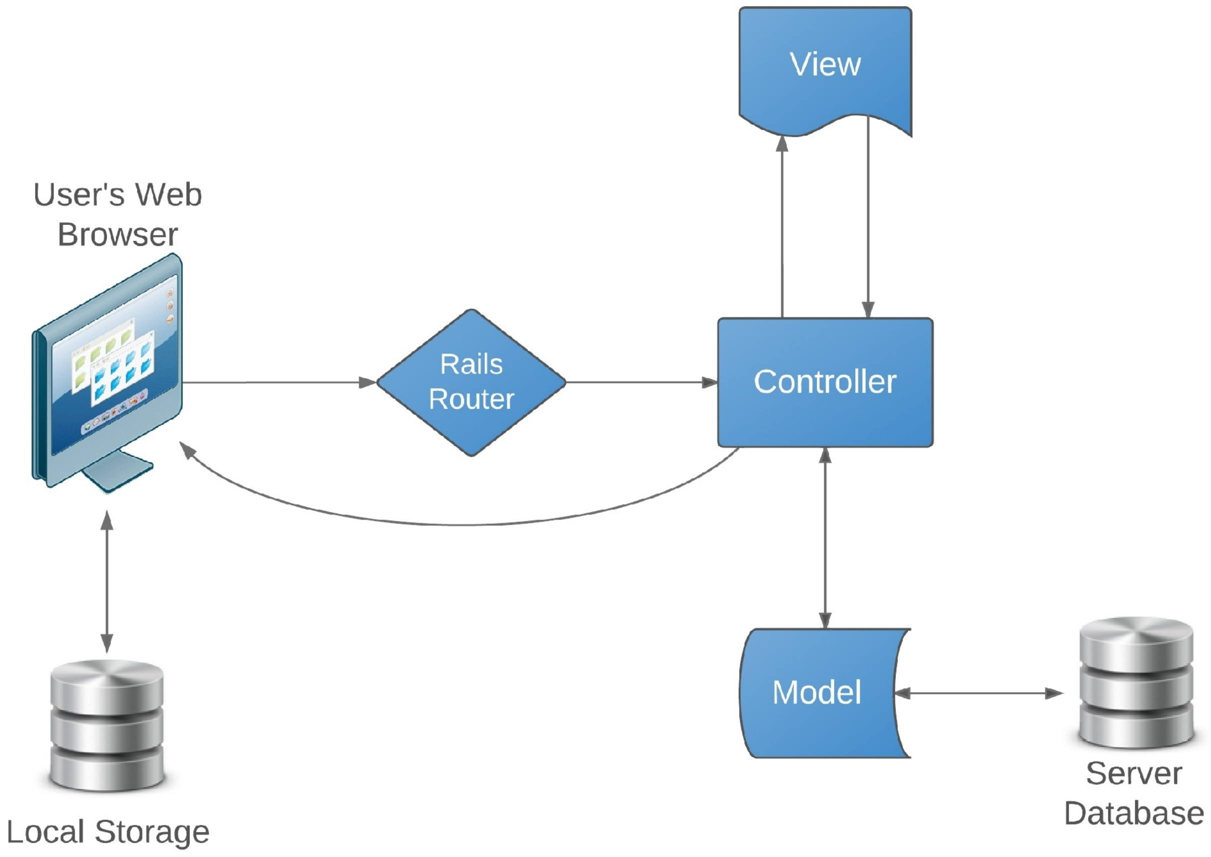
4.3.6 gVim

Editor

I had only recently made the switch to gVim (graphical Vim) after using a combination of text editors (such as gedit, notepad++) and IDEs (NetBeans, Eclipse etc.) but found that it was the perfect medium for me. And with it set up exactly how I like it and with many useful plugins, I found that working with gVim was far more productive than using IDEs, especially on a project of this nature where you're mainly editing JavaScript, CSS, HTML and Ruby.

5. Implementation

5.1 Application Overview



Rails uses a Model, View, Controller architecture, and once that is combined with the local storage of the user's web browser, you get an architecture as visualised in the diagram above.

5.2 Implementation Method

Implementation followed an iterative approach, utilizing Agile methodology. This was a perfect match with a project of this nature, where progress can be measured by releasing new working versions of the application regularly.

When deciding which requirement to implement next, I chose the requirement which would add most value overall to the project in the time I estimated it would take to implement it.

5.3 Implementation

5.3.1 Initial Prototypes

During the analysis phase of the project life cycle, small test programs were designed and developed in order to better understand the scope of the project and to improve understanding of the technologies. Each prototype would consist of one or more JavaScript files, an HTML page to provide a test environment and possibly a CSS file to style HTML elements.

This proved to be a very valuable stage of development as each small prototype provided all or part of a solution for different key features required for the application. This meant that during development proper, I could refer back to the prototypes and see how to implement features ranging from moving diagram elements through drag and drop to editing the text attributes of a diagram element.

5.3.2 Diagram Representation

When working with vector graphics, the Raphael API allows the creation of rectangles, circles, ellipses, paths and text (an abstraction of a path). In the prototypes, each diagram element was made up of a path defining the shape of the element and text positioned relative to the width and height of the shape (see Figure 5.1).

While this meant that the minimum number of vector graphics components was used to represent a UML diagram element, it was also hard to manage and quite inflexible. For example, resizing a diagram element defined like that without scaling it would be a nightmare involving calculations based off of fractions of the elements width and height attributes.

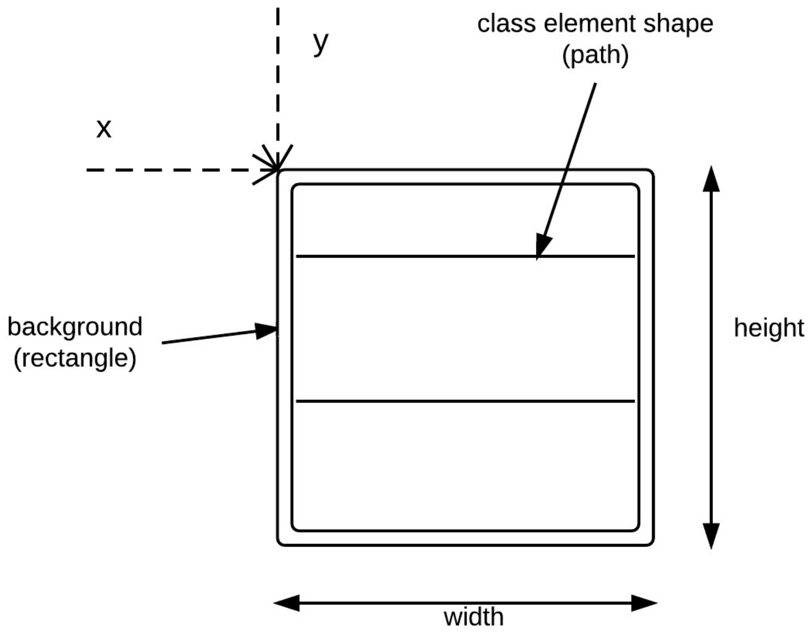


Figure 5.1: The prototype model for a class element.

A new model was conceived which takes a more structural approach. Each diagram element is an object with zero or more containers, a shadow or highlight component, and a background (or a path if the diagram element is a line). This can then be extended upon, depending on the element type; an association element can have arrows at each end of the line, for example.

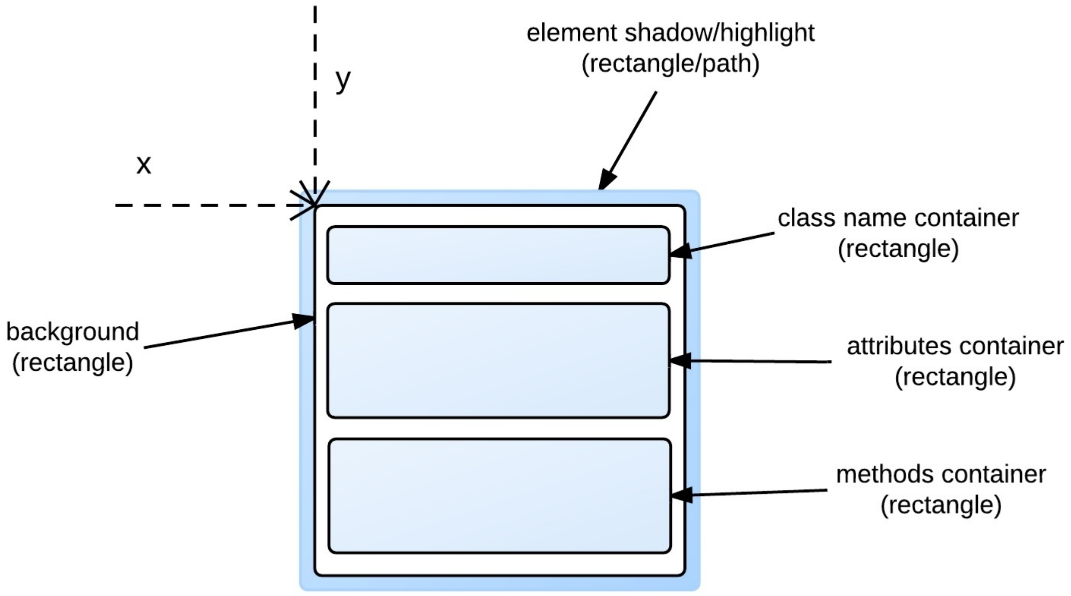


Figure 5.2: The revised model for a class element.

This new model allows for simple element selection, text editing, dynamic insertion and removal of containers and other sub-components, and resizing (see Figure 5.2). Using this model, text can easily be positioned relative to its container rather than through a calculation involving the overall width and height of the diagram element.

This simplifies both manual and automatic resizing of an element. Either a single container can be resized because the text inside it doesn't fit any more – in which case the rest of the element's components resize in response – or the entire element can be resized, resizing all the element's components at once, respecting some minimum size values.

Since every vector graphics component is also a node in the DOM, handling events such as mouse clicks or mouse movement couldn't be simpler – just assign an event handler to the desired component of the diagram element.

5.3.3 Saving and Loading Diagrams

As described in Section 4.1, each diagram entry in the database will have a "data" column with text, serialized by Rails when saved. But how to obtain that data string and how to use it to load a saved diagram?

In the context of UMLTool, a UML diagram is defined as set of zero or more diagram elements and is stored internally as an array. This presented two possibilities to solve the problem:

1. Use JSON.
2. Write custom JavaScript method(s) which convert a given diagram into a data string, and a given data string into a diagram.

Using JSON would certainly be the simpler option of the two. Saving a diagram would require one call to `JSON.stringify()` to convert the array of diagram elements into a string which would then be sent to the server. Loading a diagram would be only slightly more complicated, requiring a single call to `JSON.parse()` on the JSON-ified saved diagram data string, then reassembling the diagram from the resulting array.

That sounds like an ideal solution, however, examining exactly what went into the saved data string revealed that many unnecessary attributes, SVG/VML data and attributes, and any special functions defined for a diagram element were also being saved, resulting in severely bloated data strings in the database.

Addressing each of the aforementioned issues in turn, many diagram element attributes don't need to be saved since they are either reset or defined upon creation. SVG/VML data does not need to be saved since Raphael handles that on element creation (also, there could be problems with saving, say, SVG data, then trying to load a diagram on a browser that doesn't support SVG). Diagram element-specific functions (such as the `update()` function for association control points) are again defined during element creation and so do not need to be saved.

So it came down to a comparison between the simplicity and speed of implementing a JSON solution, and the reduced database space and faster load times of writing custom methods.

Despite the fact that it would certainly use up more development time than the alternative, I decided to go with writing methods from scratch, based around the JSON process, to handle saving and loading diagrams. To accomplish this, the initial idea was for each diagram element type to have a `toString()` method which would return a string representation of itself. Saving a diagram would then comprise of calling `toString()` on each component of the diagram and concatenating the results together into the data string. Loading a diagram would be much more complex, having to parse the data string (probably with a heavy emphasis on regular expressions), then gradually mapping properties to values before finally reconstructing the diagram.

There was, however, a far simpler option. After analysing exactly what information needed to be saved in order to flawlessly recreate a given diagram element, it was discovered that only the vector graphics attributes of the underlying Raphael objects were required. These attributes (accessible by calling the `attr()` method on any

Raphael object) contained all the positional, size, colour and style data that described that object. With this approach, the method for saving a diagram remains relatively unchanged: iterate through each component in the diagram and get a string representation of it, then concatenate them together.

The process of getting the string representation has been vastly simplified now though, resulting in a single, general `toString()` method which, given a diagram element, will return a string of the attributes of the element's sub-components in object literal form, i.e.

“{item$_1$: {attrs$_1$}, item$_2$: {attrs$_2$},..., item$_n$: {attrs$_n$}}”

The key here is that the strings are formatted in object literal form. This means that if they are passed as a parameter into the `eval()` function, an actual JavaScript object will be returned, so no need to parse long strings to extract values, they can be immediately accessed from the object. To then get a data string of the entire diagram, simply concatenate the results of the `toString()` calls together, again retaining object literal form. Now the data string contains only the absolutely essential information needed to recreate a diagram, thus greatly reducing the amount of storage needed per diagram.

This method also allows for fast and easy diagram loading. The entire data string is passed into the `eval()` function, returning an object containing all the diagram elements' attributes. One additional piece of information that gets saved for each diagram element is its type. Using this, the appropriate constructor is called and the attributes for that diagram element are passed as parameters, perfectly recreating the saved diagram.

5.3.4 Exporting Diagrams

Giving users the ability to quickly and painlessly export their created UML diagrams to other formats was a key requirement. Implementing this clearly requires some way of taking the vector graphics of the diagram and rendering them as an image.

There are no native methods to do this but my earlier analysis revealed the canvg JavaScript library, which takes SVG as an input and renders it onto an HTML5 canvas element. Unfortunately this is still not a complete solution as canvg only supports SVG and not VML, so the export feature won't work in Internet Explorer.

The HTML5 canvas element has a `toDataUrl()` method which can be used to get an image representation of whatever is rendered on the canvas. This method also takes an optional MIME type to generate the image as. However, not all MIME types are fully supported in all browsers yet, most notably “image/jpeg” and “image/svg+xml”, so “image/png” was used as the MIME type for all exported diagrams. Once all the modern browsers implement the other MIME types, extending the implementation to include them would be trivial. It would simply be a case of passing a different parameter to the export method depending on which file type was desired.

One issue with this particular implementation was that while converting canvas data to an image data URL was fast, opening it in a new window as a URL was tremendously slow, causing the browser to lock up while the URL was parsed. To resolve this issue there were two clear paths:

1. Open a new window and embed the image data URL as the source of an `<img>` element.
2. Set the image data URL as the current window's location.

They both opened the image data URL in roughly the same amount of time, but the first option meant that application state wouldn't need to be saved before redirecting and thus would require less processing and give better performance, so this was selected.

5.3.5 Taking it Offline

One of the main features which differentiates UMLTool from other web-based UML applications is the ability to use it offline (providing the user has visited the site at least once while online). During the literature review and analysis, getting an already functioning web application to work offline seemed a relatively painless process, merely referencing a cache manifest file in the HTML and listening for online/offline events.

This was far from reality, however. The implementation largely followed what was deduced in the analysis of offline working in terms of handling online and offline events, when to synchronize diagrams with the server and so forth. One of the main issues was trying to work with the cache manifest successfully. Once a cache manifest was introduced, all resources needed for the application were loaded from the application cache (and correctly so, by design). However, when a file listed in the cache manifest changed, the browser would still load from the application cache since it can't tell whether the files themselves have changed, only if something in the manifest as been altered.

To circumvent this issue, a Rails route was mapped to the path for the cache manifest, which would then redirect to a Rails controller action. This action ("pages#manifest"), would then dynamically store a unique hash in the cache manifest file, and return it to the browser. This allowed users to log in to the application, at which point the "manifest" Rails action would embed a hash of the user's unique session token thus forcing the browser to check the manifest and see that something had changed. In this way, the correct views would be rendered for the user while online and all the offline functionality would work as well.

6. Testing

This project raised difficulties when it came to testing the software system. Since it was largely visual, the methods for testing whether the application was performing as expected were limited.

6.1 Unit Testing

Unit testing was by far the hardest testing to do. Ideally all the JavaScript would have been developed test-first, but the few test-driven development libraries and frameworks available were cumbersome and unwieldy. The alternative would have been to go with something like JsUnit to unit test JavaScript functionality. However, the test cases would have been so few and would have had to deal with other JavaScript library dependencies that it didn't seem worth the time doing.

6.2 Functional Testing

Functional testing, on the other hand, was far easier to carry out since most of this involved testing the Rails application, which was much more open and inviting, especially with the Behaviour-Driven Development tool RSpec. Writing tests was almost as intuitive as writing in English, and because RSpec builds on Ruby, it has a whole wealth of useful methods to use when setting up test fixtures etc.

This made it very simple to test not only standard HTTP request actions such as user sign up, logging in and out and so on, but also XmlHTTPRequests (i.e. AJAX requests), simply by prepending the request statement with `xhr`.

All the functional tests written for the Rails side of the application pass.

Testing whether the same features work offline was much more difficult and, in the end, was left to acceptance testing to do that.

6.3 Acceptance Testing

As previously mentioned, acceptance testing was the most immediate method for testing the offline capabilities of the application. But this was not its only use. Acceptance testing was also used to test most of the diagram creation, editing, saving and loading features of the application. This was crucial as it quickly highlighted the unforeseen problems with cross-browser support of UMLTool. Webkit browsers, such as Safari and Chrome, have certain bugs when rendering certain types of SVG which can cause large artefacts to be left on the screen. Raphael does provide some support in this regard, with the `safari()` method, which forces a redraw after SVG elements have been manipulated, but the effectiveness of this was hit-and-miss.

7. Evaluation

7.1 Summary of Achievements

Out of the four initially specified UML diagram types to support, only support for class diagrams was implemented. Nevertheless, the application was designed and developed in such a way that almost all the functionality is there to support other diagram types, they just need specific details such as defining the shapes of other diagram elements, but all the code for saving, loading, working offline and dealing with local storage would work for any extension to the application.

UMLTool supports offline working, diagram management and has an intuitive, user-friendly interface. The ability to delete diagrams didn't quite get implemented and was removed from the source code as, while in its partially implemented state, it was causing bugs for the rest of the application. But again, almost all the functionality is there for deleting diagrams, the only addition would be concerning the way delete requests are handled while offline with regards to the local storage design.

7.2 Evaluation of Results

If I had the chance to start again, possibly the only thing I'd do differently would be to try and find a complete solution for the application wrapper (be it in Rails, Django, PHP, Java or some other language), rather than writing my own from scratch. While that was certainly rewarding and I learned a lot from it, I can only imagine how much more I would have got done on the JavaScript side of things if I could have focussed almost all my time on that. Having to devote almost half my development and implementation time for something which wasn't mission critical specifically to the project was probably a mistake.

7.3 Future Work

Obviously one could fairly easily extend what already exists to allow construction of sequence, use case, and activity diagrams and with that in place could practically make any type of UML diagram.

Other minor things like undo/redo could fairly easily be implemented, although that would require introducing a concept of “actions” so that they can be reversed and/or replayed. That might actually be the first step towards implementing real-time collaboration.

User collaboration could be improved on. From simple features such as giving the user the ability to make diagrams “public” (accessible by all logged in users) or specifying particular users who are allowed to view/edit desired diagrams. However, this could be taken to the next level by implementing real-time collaborative editing of diagrams. This was looked into as it was one of the original requirements, but it turned out to be vastly out of the scope of the project. It could have been feasible if it was the main aim of the project (i.e. to create an online, real-time, collaborative diagram editor). However it certainly seems possible (from my point of view) as the diagrams are rendered as SVG or VML depending on the web browser used, which are both XML-based file formats, i.e. structured documents which is exactly what a real-time collaboration implementation demands.

One feature I'd really like to implement would be the context sensitive menus and toolbars that many other applications are using (as noted in Section 2.2). This would definitely streamline the diagram creation and editing process. For example, in a class diagram, the user could create a new class element, then a new toolbar would automatically be displayed with the various associations that could be created, and selecting one would automatically attach a new one to the edge of the class element.

Another thing, slightly related, would be to give the user more flexibility over associations/lines in general, such as the ability to add/remove control points, have different path types (e.g. squared – what is implemented currently, rounded – would be as simple as replacing the “L” in the association's path with a “C”, or straight – which would allow diagonal lines etc.).

Since the level of detail is quite an important feature in this project, using text to represent what you want is critical. And almost as critical is how that text is formatted. For example, in a class diagram, if an attribute in a class is underlined, it means it is a static attribute. At the moment, the user can either apply a style to all or none of the text in a particular container (i.e. in that previous example, all attributes would also be underlined). One option could be to take the route of LucidChart and have each word/line as a separate element with its own style attributes etc.. But this wouldn't be a good idea as it greatly over-saturates the diagram with extra elements, negatively affecting performance. However, SVG (but sadly not VML) provides a foreignObject property for elements which can be used to embed HTML into the SVG. So anything you can do with text styles in HTML you would be able to do with the text in the diagram.

One request that was fairly frequent during target user research/collaboration was for the tool to take source code as input and generate UML diagrams from it. I envision this as the final stage of the project, something that would be developed last, and once implemented, the application would be pretty much 100% complete. Problems could be: to make it fast would be nice to make use of multi-threaded etc. but not much support for that in JavaScript at the moment (web workers are a step forward though). Perhaps this could happen server-side, in which case, the hardware would need to be drastically improved.

Appendix A: Bibliography

Rails Resources:

railstutorial.org

railscasts.com

railsforzombies.org

JavaScript Resources:

Top SVG JavaScript Libraries Worth Looking At by Dimas Begunoff, 4th February 2010

http://www.farinspace.com/top-svg-javascript-libraries-worth-looking-at/

HTML5 offline webapps: a practical example by Pedro Morais, 7th October 2010

http://blog.cubeanywhere.com/2010/10/html5-offline-webapps-practical-example.html

SVG or Canvas? Choosing between the two by Mihai Sucan, 4th February 2010

http://dev.opera.com/articles/view/svg-or-canvas-choosing-between-the-two/

http://www.crockford.com/