Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Ibrahim Olanigan

APPLYING PATTERNS IN WEB-BASED USER INTERFACE FOR DIMENSIONAL ANALYSIS CONCEPTUAL MODELLING FRAMEWORK

Supervisor:	Professor Ahmed Seffah
Employer:	Dynavio

ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Ibrahim Olanigan

Applying Patterns in Web-Based User Interface for Dimensional Analysis Conceptual Modelling Framework

Master's Thesis

54 pages, 6 figures, 4 tables

Supervisors: Professor Ahmed Seffah

Keywords: design patterns, user interface, front-end, modelling, conceptual design

Usability and user experience are the cornerstones for building the web and mobile applications. My thesis research entails the design of a web application for a conceptual system design. The thesis focuses on the usability requirements, the engineering challenges encountered and the implementation using selected design patterns. The process of designing and developing a web interface is detailed.

ACKNOWLEDGEMENTS

I am grateful to my supervisor, Professor Ahmed Seffah and my work superior and collaborator, Professor Eric Coatanéa for their invaluable support and insightful feedbacks.

I extend a special gratitude to my partner, Nimotalahi King for her consistent encouragement and support.

I dedicate this work to my mum and my family who have always stood by me. Thanks to my colleagues at Dynavio and friends, Hossein, Faisal, Shola, Lekan and Larry.

Ibrahim Olanigan
28.11.2016

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

CSS	Cascading Style Sheets
DA	Dimensional Analysis
DACM	Dimensional Analysis Conceptual Modelling Framework
DSM	Design Structure Matrix
DMM	Domain Mapping Matrix
E2E	End-to-End
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IDE	Integrated Development Environment
JS	JavaScript
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVV	Model-View-View Model
MVW	Model-View-Whatever
SI	International System of Units
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
UI	User Interface
WWW	World Wide Web

1 INTRODUCTION

1.1 Background

This report discusses the process of designing a web-based User Interface (UI) for DACM (Dimensional Analysis Conceptual Modelling) based on the work of Professor Eric Coatanéa. I completed this project for Dynavio Cooperative in collaboration with a team headed by Professor Eric, who also serves as the director of the board of the company.

The design of the system focuses on its usability, the usability requirements, and applicable design patterns to fulfil the stated requirements.

1.2 Goals and delimitations

The primary objective of this report is to highlight the usability considerations, appropriate design patterns, the technical implementation as well as the engineering challenges encountered in developing the DACM web UI.

The system consists of the front-end component, which is visible to the user, and the back-end component that handles user authentication, content storage and modelling applications. However, this report focuses only on the design of the front-end part of the system.

This work aims to tackle the following research questions:

- What are the key design patterns for designing web application?
- What are the technologies and tools for actualising design patterns?

1.3 Methodology

This research is a case study of applying patterns to design a web application. The first research question provides the basis for identifying and documenting design patterns for the web application. The second research question provides the basis for identifying available technologies and tools that could be used to actualise the identified patterns to develop a web application.

1.3.1 Case Study as Research Method

A case study is defined by Yin (1994) as "an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident" (Yin, 1994).

The strength of a case study research lies in the ability to investigate a phenomenon in its context without the need to replicate the phenomenon to understand it. This type of research utilises multiple data sources and can be based on a mix of qualitative and quantitative approaches.

This research follows the three fundamental principles of data collection in case study research:

1. Triangulation – The evidence acquired from multiple sources are used to corroborate the same finding.
2. Case Study Database – An organised collection of the pieces of evidence enhances the transparency of the results and the repeatability of the search.
3. Chain of Evidence – Proper citation of the referenced sections of the case study databases allows for easy retrieval of evidence. (Rowley, 2002)

1.3.2 Software Methodology

The design of the DACM web application is based on Agile software development which is characterised by Cho (2010) with:

1. Fast development cycles: Each development cycle lasted for 1-3 weeks depending on the complexity of the developed feature.

2. Iterative and incremental development: The development process is iterative and aimed at improving or modifying available features.
3. Minimal planning: The iterative nature of the development alleviated the need for advanced planning as the development is structured to adapt to changes in requirements.
4. Customer collaboration: The customer (Dynavio) plays an active role in the development process, and is regarded as part of the development team.
5. Frequent delivery: The fast-paced development ensured that features (both new and updated) are delivered on a regular basis.

1.4 Literature review

The NELLI portal of Lappeenranta University of Technology was queried with the phrase "web application design patterns" using sources like SpringerLink, IEEE, Emerald Journals and Web of Science among others. A web-based search was conducted on Google search to compliment the NELLI portal search.

A total of 148 articles and eight web sources (mostly documentations) were retrieved. My supervisor provided additional resources, and after careful consideration for the research scope and relevance of the case study, 20 references were recorded in the research.

1.5 Structure of the thesis

The first chapter introduces the topic of the research. Section 1.1 describes the background for the research, the goals and questions for the research are mentioned in section 1.2, the methods in section 1.3, the literature review in section 1.4 and the thesis structure in 1.5.

The second chapter presents the case study. Section 2.1 lists the usability requirements of the web application, the design patterns selected from various sources to guide the UI design are listed in section 2.1, the Dimensional Analysis Conceptual Modelling (DACM) framework is described in section 2.2, and the design implementation is detailed in section 2.3. Section 2.4 describes the technical development of the web application

The third chapter discusses the challenges encountered before and during the design of the application along with the relevant decisions made. The last chapter discusses possible improvements to the web application.

2 CASE STUDY

This chapter discusses the various sets of information gathered and the technical components employed for the design of the DACM web application. These information include the requirements for the web application, the selected web-related design patterns and the DACM framework.

2.1 Usability Requirements

This section describes the key usability concerns and requirements of the application.

1. General

The application should be a graphical User Interface (UI) that is accessible via the web. It should comply with the requirements of the WWW accessibility standard, and it should be designed based on Material Design, which is a design language created by Google.

2. Presentation

The application should support the minimum resolution of 800 pixels by 600 pixels (800x600) on computer devices. It should run efficiently on modern web browsers like Chrome, Firefox and Edge browsers. It should have excellent readability with clear-to-read and easy-to-understand labels and titles. It should have clear and accessible navigation links.

3. Prevent Errors

The application should allow users to select values from a list of valid values when possible. Use a drop-down list, or an auto-complete text field be used for text and spin control may be used for numeric values.

4. Help Users Recover from Errors

If an error occurs, the application should clearly describe the source of the error and clear instructions on how to fix it. If the error is due to data entry, the application should allow re-entry of the data. The application should allow users to undo or redo recent actions.

5. Avoid High Mental Workload

The application should have an accessible guide or documentation that provides all the necessary information needed to use the application. It should use hints when necessary to provide additional information to the users.
It should allow for multiple views within a window rather than use multiple windows to complete a general task.

6. Minimize Visual Workload

Display the application in well-organized views that ensure easy visual scanning. It should only display relevant views to the user.

2.2 Design Patterns

Based on the requirements collected, several design patterns were identified and gathered to realise the needs of the system. These patterns are primarily from the book, "Web Application Design Patterns" by Pawan Vora and Google Material Design.

Each section in this chapter begins with the main categories where the patterns are applied, followed by sub-categories. Each sub-category lists and describes their associated design patterns.

2.2.1 User Authentication

The system is expected to handle different data and sessions for users, hence the need for an authentication system to secure user information and to prevent unauthorised access. The following design principles have been gathered to realise the requirements for user authentication.

2.2.1.1 Registration

Most applications demand that users register an account before they are granted access. Registration is done primarily to provide a unique experience for each user and identify them uniquely. Registration often requires that the users provide their personal information like name, age and address which the application may use to provide unique user-experience.

Associated Design Patterns

I. **Use CAPTCHA:** Captcha enables the designer to keep out web crawlers from creating pseudo accounts in the system. It is usually placed on the registration page, sometimes as the last field before the primary action button.

II. **State Benefits of Registration:** The application must clearly indicate the advantages of registering an account. Users should be informed about the gains that await them after registration. In cases where registration is not free, users may be granted a "trial" option to explore and examine the application in more depth.

2.2.1.2 Log In

Secured systems require users to identify themselves with a combination of a unique identifier (usually username, user ID or e-mail address) and a password or passcode that the user may provide or generated by the system. These combinations (known only to the user) should help to restrict access and saved information to the users only.

Associated Design Patterns

I. **Echo User's Passwords with Non-characters**: Design the password field to hide the user's password and displays the characters as bullets or asterisks. If the user enters a wrong password, the system alerts the user of the error as an alert, a hint or both.

II. **Offer Secure Login**: Transmission of the user's credentials to the server should be through a secured connection. The connection is secured by adding a Secure Sockets Layer (SSL) certificate to the server domain.

III. **Offer a Registration Option**: New users have the possibility to register to use the application.

IV. **Remember Login Information:** Cookies are used to retain the user’s login credentials on the device used to log in. The credentials may be fully saved (both the unique identifier and password) or partially saved (just the identifier).

V. **Locking User’s Account after Failed Attempts:** The user should be prevented from logging in after a defined number of failed login attempts. The account is temporarily locked to prevent unauthorised users from gaining access to the system through guesses and automated password types.

2.2.1.3 Log Out

The system must provide an easy option for users to end their session and log out of the system. "Logging Out" is mandatory for any secured system to prevent unauthorised access to sensitive information and it is vital to use when users operate multiple accounts in a system.

Associated Design Patterns

I. **Use Labels Consistently:** Terms like sign out, sign off, log out, log off and are the most often used to signify the action of ending user sessions. It is highly recommended to use equal terms to denote the actions of entering and exiting the system to maintain consistency. Hence, Log in with Log Out, sign in with Sign Out to mention a few.

II. **Acknowledge Logout:** The system must clearly indicate to logged-in users when their sessions have ended and thereby logged out of the system.

2.2.1.4 Automatic Logout

The web application keeps track of user's activity and inactivity and should end user's session after an extended period of inactivity. Implementing automatic log out handles security concerns that they arise due to user's distraction or forgetfulness.

Associated Design Patterns

I. **Save User's Information:** An efficient implementation of automatic logout should consider saving users' information before ending their session. It ensures that no vital information is lost while the application is trying to secure it from unauthorised access.

   For small-scale applications, user's information may be saved after the user has made meaningful interactions with the application. These interactions may be known as trigger point between the user and the system. For instance, when a user is writing a new message in Google Mail (Gmail), the message is saved and updated as “draft” with every keystroke and the draft can be edited any other time and stored until the user sends the message to the recipient.

2.2.2 Forms

Forms are essential components of web applications that enable users to interact with the system or perform certain actions. Some common uses of forms include sending electronic messages, uploading files, writing a post among others. Therefore, it is important to clearly state the purpose of the form and make it simple to use.

2.2.2.1 Clear Benefits

The user may not be aware of the benefits of filling a form nor understand how to. Therefore, the benefits of filling a form must be stated.

Associated Design Patterns

I. **Explain the Benefit of Registering on Login Forms:** The login form should describe the benefits of registering into and having access to the web application.

II. **Explain the Benefits Before Leading Users to the Form:** Describe the benefits of filling a form before displaying it to the user.

2.2.2.2 Logical Grouping

Users may be reluctant to fill out long forms, yet these forms may be required to accomplish vital tasks. Hence, divide the form distinct groups or sections that represent different sets of information that is made known to the users. Filling the form as a collection of groups makes the form appear manageable and easier to complete.

2.2.2.3 Label Alignment

Labels are textual indicators for what their corresponding elements represent. Therefore, the association between labels and their fields should be clearly stated to reduce user errors and make the form easier to fill.

Associated Design Patterns

I. **Use Embedded Labels Sparingly:** Labels that appear in their input fields are known as Embedded Labels. Often, they are used for searching as most search application only has one input field (the search bar). An embedded label usually serves as a hint to the input field and is removed the field is in focus.

   These labels should only be used with few input fields and when users are well conversant with the information required by the form.

2.2.2.4 Required Field Indicators

When filling a form, the users may be asked for certain information that is required by the application, along with additional information, which may be used to enhance the user experience. The users cannot identify which information is needed by the application if they are no indicators or pointers to them.

Associated Design Patterns

I. **Show the Required Field Indicator Legend:** A well-designed form should not assume the user's knowledge of what the required field indicator of that form is as designers might use different signs or symbols; rather it should be clearly stated in the form.

II. **Provide Instructive Text for Sensitive Information:** When the form requests for sensitive information, the purpose of such information and how it would be used in the application should be indicated

2.2.2.5 Smart Defaults

When forms become lengthier, a longer time is required to complete and increase the chances of errors. An effective means of reducing the hectic task of filling out a form is to provide some default values, so long as the field is not for sensitive information.

These default values may be assumed based on the previous choices of the users or appropriate contextual estimates like providing the current date as the default value in many time-related tasks (e.g. Date selection for ticket purchases). These defaults may also enlighten the user about the nature and format of the data expected in the field.

Associated Design Patterns

I. **Avoid Default Values for Sensitive Information:** Default values should not be provided for sensitive or personal information like age, race, religion, gender and so on. Providing defaults for this information may give the impression of bias or offend many potential users.

2.2.2.6 Keyboard Navigation

Users often use the Tab key or the directional keys on the keyboard to navigate around when filling a form. When a form does not cater for this navigation, the users would be required to use their mouse to navigate instead. This mouse-only navigation becomes an unnecessary burden for the regular users and unusable for users with assistive technologies.

Therefore, forms must be designed to allow the use of standard keyboard shortcuts like using the Tab key to navigate between fields and the Enter key to submit the form.

2.2.2.7 Input Hints/Prompts

A well-designed form must ensure that the users are well aware of the information expected from them regarding the format, syntax or nature.

Associated Design Patterns

I. **Provide Dynamic Instructions:** Additional instructions may be given in the form of hints or prompts when users select or focus on a form element. In this case, the chosen of a particular field would only be visible to the user when they interact with the field.

II. **Match Field Sizes to Appropriate Data:** Avoid making field sizes more or less than the expected data. The length of a field may inform the user about the specific nature of the data expected in that field and thereby, removing the chances of error while filling the form.

III. **Provide validity indicators:** An efficient way of informing users that they are correctly filling the form is to markers or signs that each field entry meets their requirements. Most commonly used indicators are the green and red colour or symbols in these colours.

   For instance, the appearance of a green symbol beside a field or a green border around the field may indicate that the field is correctly filled. A red indicator either as the colour of the filled entry or border colour or as a symbol near the field may be used to notify the user of an incorrect entry. An incorrect entry should be accompanied by an explanation of the right format or syntax for the field.

2.2.2.8 Action Buttons

Forms require action buttons to continue in its workflow. A simple form may need only two commands, to submit the form when complete and to cancel if the user does not want to fill or submit the form. The commands are invoked with action buttons, which are buttons with the label of the control they invoke like "Submit", "Next", "Cancel" among others.

It is important that every form has the appropriate amount of commands/action buttons needed to achieve a full user experience with the chosen.

Associated Design Patterns

I. **Use Clear Labels for Buttons:** The button labels must be clear and precisely denote the specific actions they are intended to perform. The clarity ensures that the user is not confused about what the buttons do. Therefore, vague terms should be avoided; instead, the buttons should specify their intended outcome.

II. **Use “Enter” Key to Enable Primary Action:** The form should enable the” Enter" key to represent the primary action of the form. This feature is useful for forms with one input field (like Search bar) or many fields.

III. **Align Form Elements with Primacy Action Buttons:** Aligning form elements with action buttons provide a visible path to completing the form. This setup may enable the user to complete the form faster.

IV. **Disable Action Button after First Click:** The action button should be disabled after the first click to avoid repeating the action. This feature is important, especially when dealing with sensitive activities like making a purchase.

2.2.2.9 Error Messages

It should be expected that errors would occur even with the best designs. Users would not always fill forms correctly for a variety of reasons. The most common errors are:

- Incorrect information: Users may provide the wrong information when the instruction is ignored or not well understood or due to forgetfulness.
- Missing information: Users may fail to provide all the mandatory information requested.
- Syntax or formatting errors: This occurs when the user fills information in the wrong format or type a wrong numerical format for phone numbers.

Associated Design Patterns

I. **Provide Clear Instructions to Fix the Error:** The error message should be accompanied by clear and accurate instructions on how to fix the error. The application must be designed with a dynamic error handling approach to cater for a variety of mistakes.

II. **Display Error Messages on the Same Page as the Form:** Showing the error message(s) on the same page as the incorrectly filled form is a simple yet efficient approach. It ensures that the user is not only aware of the errors, but it removes the burden of memorising the errors and corrections.

    The error messages and their corrections should be displayed in a non-blocking way and preferably near the errors.

III. **Organize Error Messages:** When notifying users of multiple mistakes in a form, the error prompts, hints or messages should be displayed in an organised manner that enables the user to see all the errors made. This organisation can be achieved with the use of lists and standard error indicators (text colour or background colour).

IV. **Retain User Input:** Whenever error messages are displayed, the form should keep all the information entered by the user (including the incorrect data). This feature ensures that the user does not re-fill the entire form, rather corrects the invalid data, thereby saving the user from frustration and potentially leaving the form.

V. **Identify Source of the Error:** The application should clearly show the form elements where the errors occurred. It eases the task of identifying the incorrectly filled form elements for the users, which is particularly useful for long forms. (Pawan Vora, 2009)

VI. **Show Character Count:** The application can display a character count for both the input and the data limit of a text field. This number may be shown as a hint under or beside the text field, and it's colour changes when the input exceeds the character limit. This change informs the user that an error has occurred in the field.

2.2.3 Rich Internet Applications

Advancements in web technologies have made it relatively easier to design highly interactive and responsive web applications. Web applications now offer a complete user experience while allowing for stateless communications with the back-end/server.

2.2.3.1 Rich Form

The design of the old forms usually requires them to be validated after submission, and possible errors can only be fixed after the re-submitting and re-validating the form. However, a form should be designed to be interactive and responsive such that the inputs are validated while they are entered, and users are shown only valid choices to select.

Associated Design Patterns

I. **Design Form to Reduce Error:** Use a combination of design patterns like providing validity indicators and auto-completion to create a form that prevents the user from making many errors while filling the form.

   Validity indicators promptly inform the user if they have met the requirements for the concerned input fields while auto-complete feature may be added instead to provide a range of valid choices. These patterns and other related ones can be used as it best suits the designed form.

2.2.3.2 Auto-completion

A partial input may be used to predict a list of possible input choices. This feature is particularly useful when there is a large dataset of options that it becomes impractical to use a drop-down list. As user input data in the field, a list of possible alternatives may be suggested along with the option to select any of these options.

Associated Design Patterns

I. **Enable Use of Keyboard/Keypad for Choice Selection:** Allow users to use keyboard/keypad keys to navigate and select an option from the suggested list of valid choices. The up and down arrow keys can be used to go through the list, and the “Enter” key can be used to select the preferred choice. While navigating the list, the text field displays the focused option.

II. **Highlight the First Match in the List:** The first match in the suggested list could be emphasised and made selectable with the “Enter” key. This feature ensures that the user can easily choose the most likely suggestion without any navigation.

    Another alternative would be to show the entered text while the remaining part of the match is displayed as a hint (usually with a coloured background), which is appended to existing text when the user presses” Enter” key.

2.2.1.3Edit-In-Place

Some web applications may require users to create or edit items with a few properties. Older designs may direct users to a new page or a view and redirect them to the original view with the new or updated item shown. However, this disruption may become frustrating when users have to do this multiple times. A lightweight editor can be used for creating or editing an item.

Associated Design Patterns

I. **Use Text Select for Items with One Editable Property:** Allow users to edit a text property by selecting the text and overwriting it. This feature is a simple yet efficient approach, and it is becoming a standard feature in many web interface libraries or frameworks for graphs, tables, forms and others.

II. **Use Edit Icon/Button for Items with Multiple Editable Properties:** Allow users to edit multiple properties simultaneously by clicking on “Edit” button, which could be a button with “Edit” label or one with a Pen icon (which has become a universal symbol for changing text). These properties become editable when selecting the edit button, and the label or icon changes to signify the “Save” action, such that the item is updated when the user clicks the button again.

2.2.1.4Slider

Value entry in forms might present a challenge for users if they are not aware of the valid range of values required or the correct format expected. While this information can be shown to the user, it presents a formatting problem in the design if they are many of such inputs on a particular page. A slider control provides a solution to this issue as it shows the user the range of expected values along with the correct format.

Associated Design Pattern

I. **Display Selected Value:** Always inform the user of the value(s) chosen to ensure that the user selects the right value(s) intended. (Pawan Vora, 2009)

2.2.1.5Confirmation and Acknowledgement of Action(s)

Informing users about actions performed in the application helps to remove the uncertainty about any changes made and prevents users from making mistakes with the related task.

Associated Patterns

I. **Confirmation:** Confirmation is usually in the form of a dialog or prompt asking the user to verify if they want to proceed with the invoked action. It may be accompanied with a warning that informs the user of the possible outcome(s) of the action. The dialog title should clearly state the intent of the invoked action; this ensures that the user is well informed about the nature and likely consequence(s) of the action. Confirmation should in for irreversible actions.

s

II. **Acknowledgement:** The user is informed about the system's operation, which may have been invoked by the user. This feature ensures that there is clarity about what actions the user had performed or what actions the application had carried out to support or actualize the user's action. This feature may be accompanied by an option to reverse the action. (Google Material design guidelines, 2016)

2.2.4 Navigation

The navigation provides users with the means to move through the various aspects of the UI. Hence, it should be predictable, engaging and accessible. The first step in designing a good navigation is to identify the class of users in the application along with their roles. This setup makes it easier to provide a unique and more appropriate navigation for each user.

2.2.4.1 Tabs

Tabs enable users to transition between views of equal importance easily. It is efficient to use when the number of these views is small and mostly used for a set of sibling/child views. It is the best navigation option to use when users have to switch views frequently.

2.2.4.2 Navigation Drawer

Navigation drawer is used to support an extended or expanding list of targets, and mostly as side navigation. It is usually hidden and shown when invoked by the user. Using this pattern for parent views with siblings is appropriate.

2.2.4.3 Expanding Navigation Drawer

The navigation drawer can be extended such that each level is accompanied by a sub-section or level, which is collapsed by default and the user, can transition easily between the main levels and the sub-levels. These sub-sections or levels can be used for direct navigation or provide additional functionality to the primary targets. (Google Material design guidelines, 2016)

2.3 User Interface

2.3.1 DACM Framework

The Dimensional Analysis Conceptual Modelling (DACM) framework is a mechanism developed for the conceptual modelling of lifecycle systems. It is based on Dimensional Analysis (DA) theory developed by a community of active researchers in the field of physics and engineering.

The DA theory is a research approach for breaking down complex modelling problems into simpler forms by deducing possible constraints on the relationship between variables from the dimensions of these variables.

The DACM process consists of nine (9) steps. This thesis focuses only on three (3) steps, which are:

I. Indicate the Model's Objectives

The modeller provides the rationale for the model by creating the functional representation of the model using appropriate ontology.

II. List the Problem's Fundamental Variables

The modeller highlights the key variables affecting the functional model. They are three categories of variables to be identified in the model:

- Overall system variables (Energy and Efficiency rate)
- Power variables (Effort and Flow)
- State variables (Displacement, Momentum and Connecting variables)

Design Structure Matrix (DSM) and Domain Mapping Matrix (DMM) are used to show the specific properties and interactions within the model. DSM is used to show the interactions between elements of the same domain, while the DMM is used to map elements of different domains, in the case, the mapping of the function to variables and the mapping of the variable with the most popular elementary units of the International System of Units (SI).

III. Develop a Causal Ordering of the Variables

After all the properties of the model problem are clearly defined, the framework uses the DSM and DMM matrixes to compute the cause-effect relationship between the variables.

The study presents a case of a torpedo's movement in the water. The modeller intends to increase the speed of the torpedo in water, and to achieve that, an initial set of objectives is determined:

- Extract the most relevant set of variables and SI units needed to model the torpedo’s interaction with water
- Define the causality between the defined variables. (Coatanéa E. et al., 2016)

2.3.2 The Web Interface Representation

User input developed in the modelling step, yet the interface is designed to minimise the amount of input required from the user. The web UI has two main sections:

I. Functional Modelling (where the user defines the properties of the model problem)
II. Causal Ordering (where the user observes the cause-effect relationship)

The sections are presented in both tabular and graphical forms.

2.3.3 Functional Modelling

This section combines step 1 and 2 of the DACM process by enabling the user to define the functional representation of the system as well as the possible interactions and properties of the system.

The functional graph allows the user to model a system with graphical nodes and links. The functions are defined as nodes while the variables are defined as links (for power variables and system variables) or as node elements (for state variables).

As the user updates the functional graph, the interface automatically generates and updates a series of DSM and DMM matrixes from the graph data showing various mappings of the modelled system in detail.

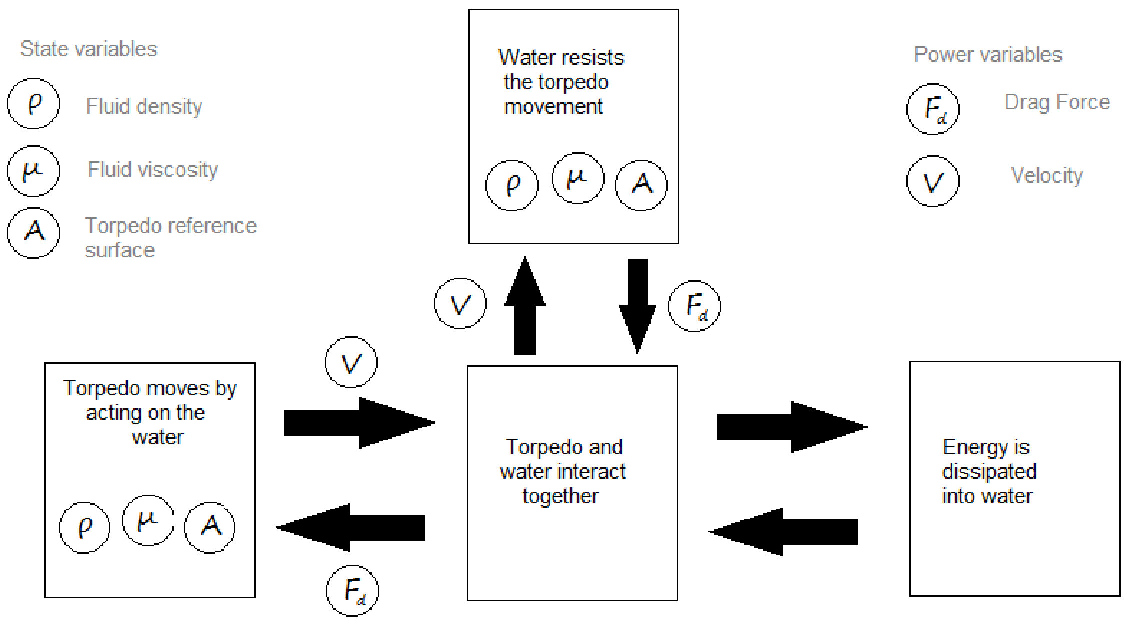

State variables
ρ
Fluid density
μ
Fluid viscosity
A
Torpedo reference surface
Water resists the torpedo movement
ρ
μ
A
Power variables
F_d
Drag Force
V
Velocity
V
F_d
Torpedo moves by acting on the water
ρ
μ
A
V
F_d
Torpedo and water interact together
Energy is dissipated into water

Figure 1. A functional representation of the torpedo case study. (Coatanéa E. et al., 2016)

Table 1. DSM Matrix shows the Function-Function mapping of the torpedo case study

	Torpedo moved by acting on the water	Torpedo and water interact together	Water resists the torpedo movement	Energy is dissipated into the water
Torpedo moved by acting on the water	▨	X		
Torpedo and water interact together	X	▨	X	X
Water resists the torpedo movement		X	▨	
Energy is dissipated into the water			X	▨

Table 2. DMM shows the mapping between the functions and the variables (state and power) in the torpedo case study.

Functions\Variables	A	ρ	μ	F_d	V
Torpedo moved by acting on the water	X	X	X	X	X
Torpedo and water interact together				X	X
Water resists the torpedo movement	X	X	X	X	X
Energy is dissipated into the water					

Table 3. DMM shows the mapping between the variables and the International System of Units (SI) elementary units in the torpedo case study.

Unit\Variables	A	ρ	μ	F_d	V
Mass (*M*)	0	1	1	1	0
Length (*L*)	2	-3	-1	1	1
Time (*T*)	0	0	-1	-2	-1

While the structures of all the matrixes are generated, the modeller may be required to complete some of the matrixes to compute the causal relationship.

2.3.4 Causal Ordering

The DACM framework uses a mathematical algorithm to compute the cause and effect relationship between the variables using the DSM and DMM matrixes generated in the functional modelling section of the UI.

The UI enables the user to compute the causality only when all the relevant aspects of the modelling are complete in both the functional graph and the corresponding matrixes. The UI converts all the matrixes from HTML format to CSV format, and exports them to the back-end where the causality is calculated and returned to the UI in JSON format. If the functional model were done correctly and without conflicts, the response would contain information about the cause and effect relationships among the variables, along with a colour grading to distinguish between the types of variables. The UI generates a causal graph, which shows the visual representation of the causality, and a DSM matrix indicating the relationships in a tabular format.

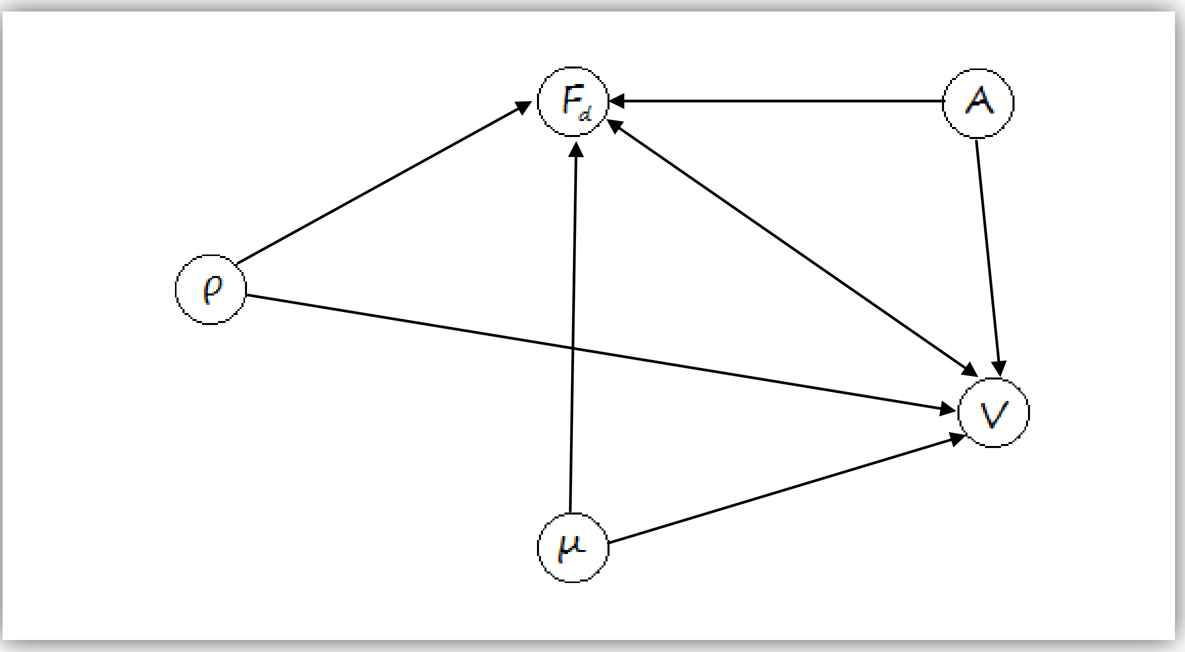


Figure 2. The causal graph in the torpedo case study. (Coatanéa E. et al., 2016)

Table 4. DSM Matrix shows the causal relationship between the variables in the torpedo case study

Variables/Variables	A	ρ	μ	F_d	V
A	■			X	X
ρ		■		X	X
μ			■	X	X
F_d				■	X
V				X	■

The project focused only on the three DACM steps. However, the interface is designed to be extensible for further implementation of the rest of the DACM process.

2.4 Implementation

This chapter describes the various technical concepts, tools and frameworks that were used to design the system.

2.4.1 Development Setup

The application core is built on the traditional web stack of HTML5, CSS3 and JavaScript. JavaScript is an interpreted language and thereby, it does not require a compiler to execute its functions.

2.4.2 Development Environment

Sublime Text (version 3) was the development tool of choice for the following reasons:

- **Lightweight:** The software has a little memory imprint, as it does not have lots of processes or threads running in the background, which may have required lots of memory and CPU usage. This feature ensured that the development is efficient on a reasonably equipped computer system.

- **Multiple Selections:** Sublime Text allows users to select multiple texts at a time and manipulate them simultaneously. This feature is useful for adding, deleting, changing, copying and pasting multiple texts simultaneously.

- **Split Editing:** The editor offers tabbing and split-view. This feature enables the user to edit and compare files side-by-side.

- **Customization:** The editor is very intuitive and allows users to configure the editor to their preference. It also offers a broad range of extension that enhances the development experience.

2.4.2.1 Add-on Libraries

I. **LiveReload:** LiveReload is a simple web server that monitors changes in the development file system and refreshes the web browser when any file under its scope changes. It saves the developer the hassle of manually refreshing a web page every time a change is made to the development files.

2.4.3 Architectural Pattern

The web application is designed based on the Model-View-Controller (MVC) framework. The MVC is one of the oldest design patterns used for implementing user interfaces; it divides a web application into three components:

- Model
- View
- Controller

The Model encapsulates the data set, logic and related behaviours of the data in the application domain. It contains the specification of the data along with appropriate methods needed to retrieve or change the data.

The View represents the visual component of the application. It focuses only on visualising any number of elements contained in the Model.

The Controller provides an interface by which the user may interact with the application (both Model and View). The Controller is synchronised with the application's Model to ensure a smooth and fast workflow between the user and the application. (Ben Smith, 2009)

2.4.4 Core Framework

Due to the increased complexity of web applications, many web application frameworks have been built to cater for the widening demands of web developers as well as providing a structured platform to implement any of the software architectural patterns related to web interface designs like MVC, MVV and others.

The front-end component of the application was developed with AngularJS. AngularJS is a sophisticated JavaScript MVW (Model-View-Whatever) Framework. The MVW pattern is a variation of the MVC pattern discussed above.

2.4.4.1 AngularJS Framework

This project is built on the AngularJS version 1.x. AngularJS is a front-end framework that extends the capability of HTML for designing dynamic web applications. It provides a higher level of abstraction to the developer and offers many features. Some of these features include:

- **Templating:** AngularJS allows developers to create custom DOM elements called templates. Templates are created by combining existing HTML DOM elements with Angular-specific functionalities that may be used to bind data, format data or validate input. Angular renders the dynamic view by combining the template with the controller and model.

- **Data binding:** In traditional template systems, the template and model components are merged one-time into view and subsequent changes to the model is not reflected in the view. However, in AngularJS applications, the model data is automatically synchronised with the view components such that any changes to the model immediately reflects in the view. This feature ensures that the view acts as a real-time projection of the view.

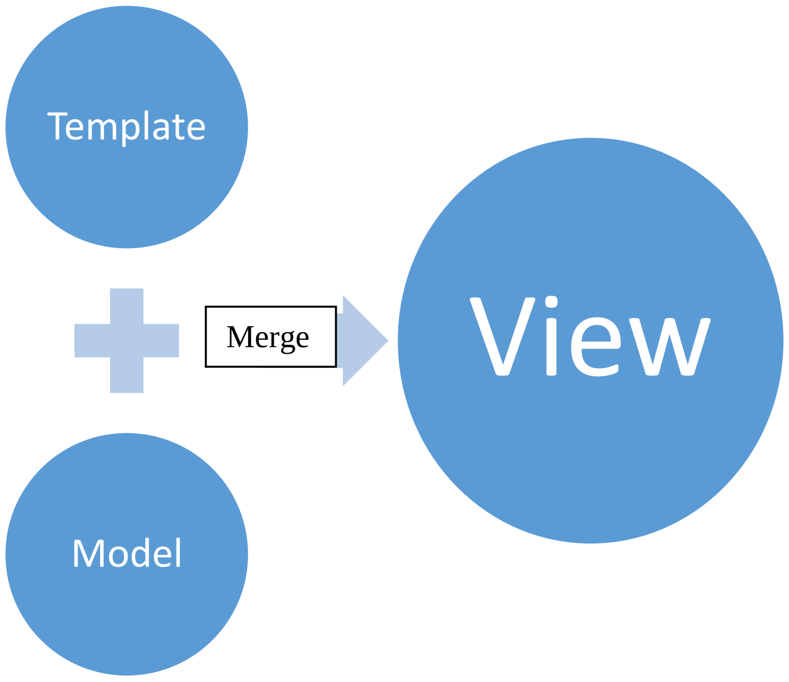


Figure 3. Data binding in Classical Template System (AngularJS: Developer Guide)

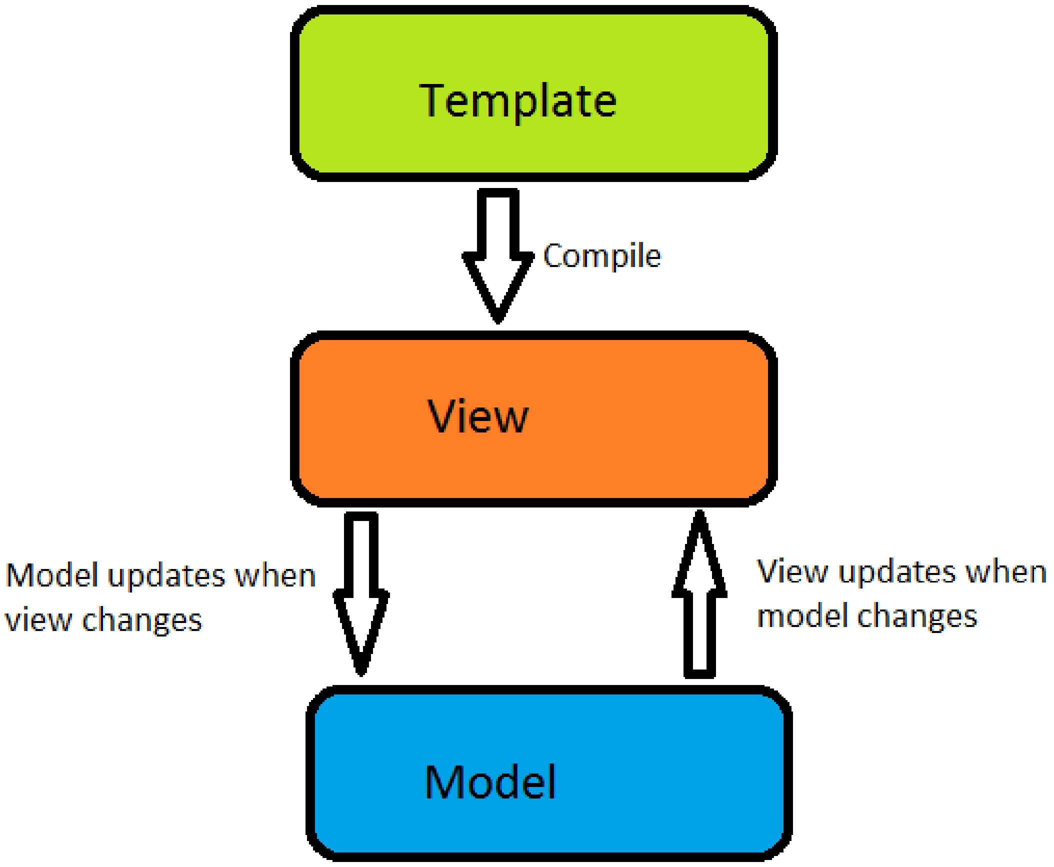

Template
Compile
View
Model updates when view changes
View updates when model changes
Model

Figure 4. Data binding in AngularJS (AngularJS: Developer Guide)

- **Form handling:** AngularJS provides a range of functionalities for handling forms. It allows for user input validation as well as the use of embedded CSS classes to style form controls based on the appropriate validation. AngularJS triggers an update to the model when the view changes, however, these triggers can be customised to define when the model update occurs. This feature is useful when real-time model updates may not be the best option for the application.

- **Routing:** AngularJS provides a routing service that connects URLs (Uniform Resource Locator) to views and controllers. When the user navigates the UI, the view changes based on the route service configuration.

- **Testing support**

AngularJS was written for easy testing; hence, it provides support and is compatible with a broad range of third-party testing tools and frameworks. The most common tests conducted on Angular applications are:

i. **Unit Testing**

AngularJS recommends the use of angular-mock, Karma and Jasmine for testing the correct execution of various operations of the application. Unit testing allows the developer to isolate the function or operation to be tested, and a set of logical tests is executed to assess the correctness of the function. Unit tests are usually used as the first approach to identifying bugs in the application and are written in JavaScript.

ii. **End-to-End (E2E) Testing**

While unit testing is very useful for assessing the correctness of individual features, E2E testing is used to evaluate the level of successful integration of different features and components with one another as well as the general health of the application.
AngularJS provides ‘Protractor’, an E2E test runner that simulates users’ interactions with the application. (AngularJS: Developer Guide)

2.4.5 System Model

The data model of the application is in the form of a JSON Array with two elements: Nodes and Links. The nodes are the data representation of the key elements in a conceptual system, and these nodes have unique and similar properties that can be extended, while the links represent the fundamental interactions between the system nodes.

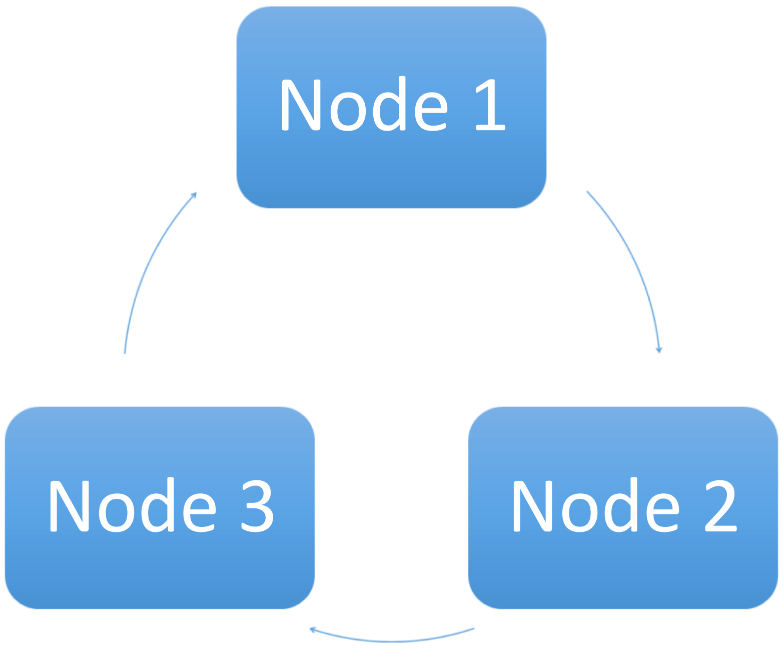
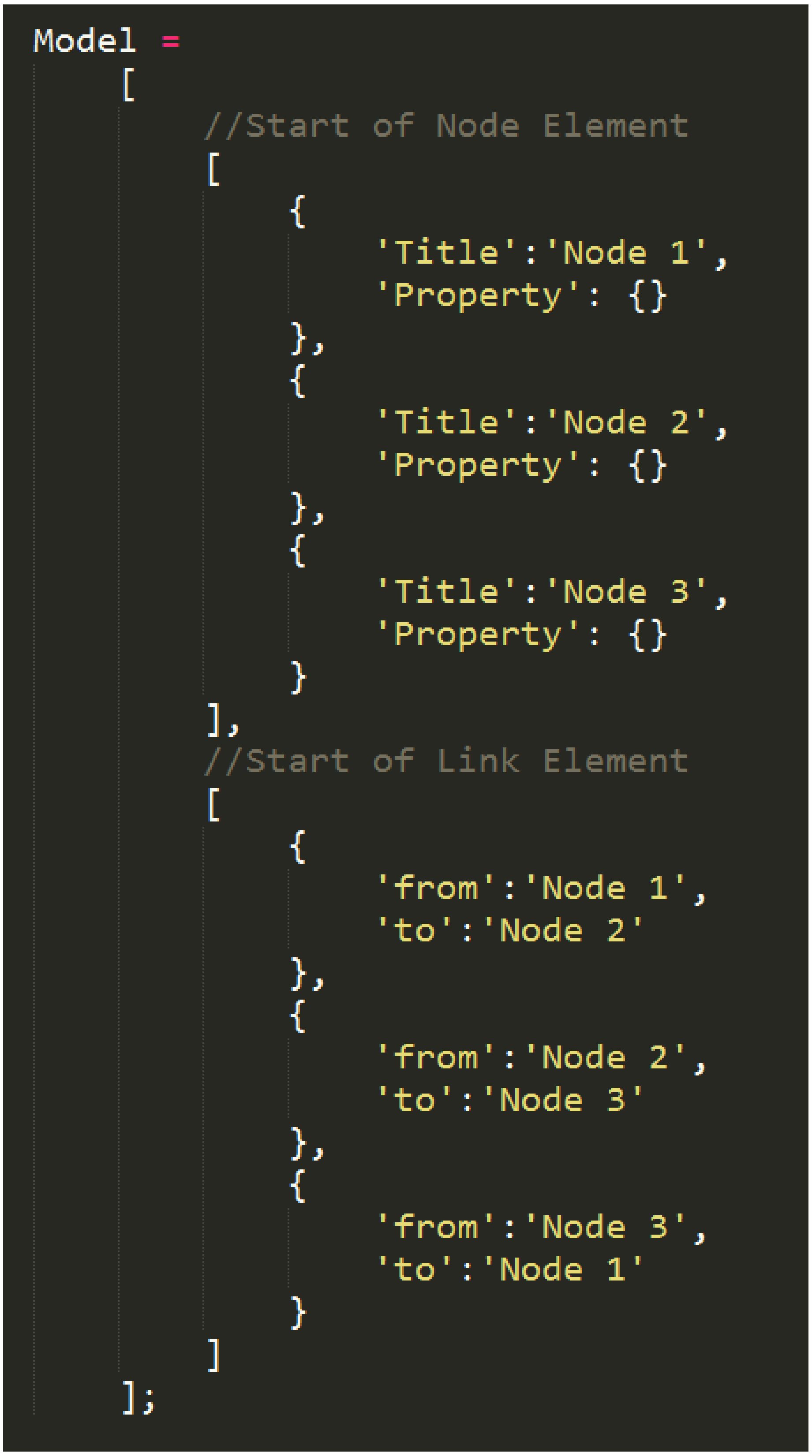


Figure 5. Visual Representation of the System Model



```
Model =
    [
        //Start of Node Element
        [
            {
                'Title':'Node 1',
                'Property': {}
            },
            {
                'Title':'Node 2',
                'Property': {}
            },
            {
                'Title':'Node 3',
                'Property': {}
            }
        ],
        //Start of Link Element
        [
            {
                'from':'Node 1',
                'to':'Node 2'
            },
            {
                'from':'Node 2',
                'to':'Node 3'
            },
            {
                'from':'Node 3',
                'to':'Node 1'
            }
        ]
    ];
```

Figure 6. JSON Representation of Figure 5.

2.4.6 System View

The application should present the system model in a manner that is interactive, responsive and understandable to the users. The target audience for this application includes engineers and designers; hence, the model is presented in both graphical and tabular forms.

A graph offers the most visually understandable view of a system. It shows all the components of a system as well as the relationships (if any) between them. Also, it can be used to display additional information about the system. Tables are often used to show data in more details.

2.4.6.1 Graph Visualization

Data visualisation is becoming popular as a part of front-end design as interest in data processing and analysis has increased. Many open-source libraries offer basic to advanced data visualisation. However, the project mandated the use of a library that is compatible with the chosen front-end and UI frameworks and offers advanced functionalities demanded by the project.

GOJS was chosen for the project. It is a licensed library, which provides graphical templates and data binding of the graphical objects to model data. (GoJS Introduction) It also provides support for the AngularJS framework. A trial version of the software was used for the project.

2.4.6.2 Tabular Presentation

In many Engineering disciplines, matrixes are commonly used to represent data, as they are understandable and appropriate for calculation and analysis. With tables, users observe a more detailed look at the system such as the various kinds of relationship that exist in the modelled system along with the metric of all the system elements.

For further processing of the system design, the tables are extracted and sent to the server.

I. Table Design

Despite the enhancements in HTML5 and variety of custom tabular libraries available, the need for application is quite particular; hence the need for customised DOM elements. AngularJS framework allows and supports the design of customised HTML elements.

The tables are designed with a combination of the basic HTML table markup and AngularJS-specific elements and attributes that were used to bind structure and format data in the table. Multiple data objects were created and maintained (in JSON format) to store and retrieve system details, and these objects are synchronised with the table such that any change to the model is immediately reflected in one or more tables.

The most used Angular directive for designing the tables in the UI is **ngRepeat**; it instantiates a template from a data collection. It provides a tracking function that can be used to prevent duplicating entries. (AngularJS: API). This directive was used to populate the table headers (row and column) as well as the individual cells. Many functions were defined to interpret and convert the graphical data into binary entries (0/1) for the table cells. An example of this essential is one that checks the graph of connections between nodes, and the intersecting cell in the table displays “X” if the nodes are connected and are empty if not connected as shown in Table 1. Table 1-4 above shows some of the matrix generated by the application.

2.4.7 Application View and Styling

The design of the web application has a sturdy with the increased emphasis on usability and user experience. Consequentially, newer web technologies, regarding concept and technicalities, have emerged to cater to these new demands.

Some open-source libraries were used in designing and styling the web application. These libraries are:

- Twitter Bootstrap
- Angular Material

1. Twitter Bootstrap

Bootstrap was created by a team consisting of a developer and a designer at Twitter in 2010 and has grown to become one of the most popular front-end frameworks in the world. (About Bootstrap).

2. Angular Material

Angular Material is designed as both a reference implementation of Google's Material Design Specification and as a companion framework to AngularJS. It provides web developers and designers with a set of reusable UI components that adopts Google's Material Design. Google supports this project and it is licensed under the MIT License (Angular Material – Introduction).

3 DISCUSSION

Applying the patterns to the web application development came with a couple of challenges which were mostly related to making the best technical decisions for realising the requirements. In this chapter, these challenges are discussed along with the decisions made to resolve them.

3.1 Choice of Core Development Frameworks

Web developers and designers enjoy a wealth of readily available and well-supported web development and UI frameworks. Many of these frameworks offer easy-to-follow beginners' guides and boilerplates that a mildly experienced developer can get started with a sample project in a short time.

However, the choice of what frameworks to use in this project was determined by the following factors:

3.1.1 Experience of Developer or Team

When dealing with time-sensitive web UI projects, the easiest decision-maker of what tools or frameworks to use in a project is the level of experience or familiarity with a set of popular choices. In this project, there is hardly any room for sampling or learning a new tool or framework; hence, the team settled for the most familiar options that got the job done.

However, web projects are usually fast-paced nowadays as developers adopt Agile or similar methodologies that ensure quick completion of tasks within projects. Therefore, web developers are expected to be “fast learners" and quickly learn and adapt new technologies in the field. As every project is unique, developers are supposed to learn something new from every project, concerning either theoretical software concepts or experience with practical tools. The selection of the key development frameworks is discussed below from the perspective of my familiarity or experience with them.

I. Core Web Framework

Before this project, my web development skills were limited to the most traditional web stack of HTML5, CSS and JavaScript. As I have worked with a couple of JavaScript frameworks beforehand, I focused my initial list to these frameworks, with AngularJS being my top choice. I settled with AngularJS as the core web framework to build the application based on my positive experience with it as well as its compatibility and support for other famous web libraries and frameworks.

II. Graph Visualization Tool

There are many open-source and commercial JavaScript libraries for visualising data and at the inception of the project; I was only familiar with Vis.js. I was advised to consider D3.js as well; hence, I sampled both libraries to assess if they offered adequate features and support for what the application required.

The tested libraries provided only the basic features that would require additional effort to make it suitable for the application design. Therefore, the focus changed to licensed libraries that not only provided the needed functionalities for the UI design but also did not put too many restrictions on the trial version. This decision led to the eventual selection for GoJS. GoJS is compatible with AngularJS and offers a broad range of features were useful for the project.

3.1.2 Type or Specification of Project

The development/implementation stage of a project comes after arguably more important steps like Requirements and Design. Sometimes during these earlier stages of the software engineering process, some project technicalities are specified that they become a constraint in the choice of frameworks to use to actualize the software.

While there are active efforts to make web technologies universally compatible with one another, they are still many existing proprietary technologies and tools that limit the choices of compatible frameworks to use.

3.1.3 Compatibility with Existing Resources

Arguably, the most important factor when selecting frameworks for web design is the level of compatibility with the available or existing resources. This element is vital to reduce avoidable complications that may arise during development or an operational disaster during release.

Selection of frameworks in these kinds of projects should be primarily based on the seamless operational compatibility with the existing tools and technologies from all the related aspects of front-end and back-end. The application back-end was constrained to work with Java applications. This constraint presented an exciting challenge to identify and select a list of tools and frameworks that would ensure an efficient workflow between the front-end and the back-end.

The primary challenge was to choose a back-end development platform or framework that is compatible with AngularJS from the front-end. The only requirements from the front-end are to be able to upload files to the server and get the response in JSON format. The Spring framework was initially selected as it is a Java-based framework and supports AngularJS. However, the setup was discovered to be too complicated for the project, and the Common Gateway Interface (CGI) provided a leaner option. CGI provides a protocol for executable programs on the server and can dynamically render web pages. The simplicity of CGI ensured there was no need for extended learning to integrate it into the project.

3.2 Choice for Development Environment

The use of Integrated Development Environments (IDEs) has become common practice among developers. These IDEs offer a one-stop solution to vital development needs like code and syntax checking, compilation, versioning, debugging among others. However, their importance varies depending on the type of project.

Using an IDE is recommended practice for any project, yet the choice of the development environment or tool could be dependent on the preference of the developer or team. For this project, a lightweight text editor was used in place of an IDE as the project did not demand the utilisation of the sophisticated features provided by an IDE, rather it provided more useful and adequate functions that ensured a smooth coding experience.

3.3 Modelling Tool Design

Deciding what to model or not is agreed to be the most difficult yet the least understood task in simulation modelling. The modeller is expected to have a thorough understanding of the real system enough to be able to build a simulation model. (Robinson, 2013).

The design of a conceptual modelling tool may not require extensive knowledge of Systems Engineering and modelling concepts, yet the development of this tool would not be possible without the collaboration with trained professionals with knowledge of Systems engineering and mathematical simulation. As the application was designed for skilled or semi-skilled system modellers, the usability aspects were also determined from that perspective to ensure that the tool is intuitive and provides support for the most common modelling tasks.

4 CONCLUSION AND FUTURE WORK

Design material and design patterns provide essential guidelines for developing a web application. Not all the listed patterns have been implemented mostly due to focusing more on the operational aspect of the application. However, the importance of usability and user experience was not neglected. The inclusion of Angular Material made it more efficient to design based on the patterns as it was developed based on Google's Material Design.

Besides, some aspect of the project was designed with basic coding. While this did not create any issue for the project, it is known that using modern frameworks may lead to better overall performance.

4.1 Future Improvements

- **Review and Implement All Relevant Design Solutions**

The identified patterns should be evaluated to assess which design patterns may be added or removed to optimise the usability and performance of the UI. These patterns should be implemented as best suited to the application.

- **Migration from Twitter Bootstrap to Angular Material**

The project started with Twitter Bootstrap due to its familiarity, and Angular Material was discovered later on during the development process. Migrating the Bootstrap coding to Angular Material should bring it closer to fulfilling the design and usability requirements as the UI framework is based on Google's Material Design.

- **Improve Development Processes**

The application has been developed using a simplistic approach that did not require many vital aspects of modern web development like dependency control and code minification. Steps should be taken to integrate essential libraries that offer dependency management and build into the project. These will guarantee a better overall development experience and enable code safety and security.

- **Extensive Testing**

A multitude of UI and code testing should be done to expose the fragilities and the limits of the application. Some of the testing to be done may include:

i. Automated UI Testing: This type of test would focus on identifying the presence of defects on the interface, most importantly transitions between the state and sections of the interface. This testing includes unit testing and E2E testing.

ii. Cross-Browser Testing: This type of test would focus on assessing the performance of the UI on the most popular web browsers like Google Chrome, Mozilla Firefox, and Microsoft Edge. It will also identify browser-related defects concerning the support and compatibility with the UI components.

iii. Usability testing: This testing would focus on how easy and intuitive the UI is for system modellers along with assessing all possible challenges with the user experience.

REFERENCES

About Bootstrap. [Online]. Available at: http://getbootstrap.com/about/. [Accessed October 14, 2016].

Angular Material. [Online]. Available at: https://material.angularjs.org/latest/. [Accessed October 14, 2016]

AngularJS API Reference - ngRepeat. [Online]. Available at: https://docs.angularjs.org/api/ng/directive/ngRepeat [Accessed November 3, 2016]

AngularJS: Developer Guide - Data Binding. [Online]. Available at: https://docs.angularjs.org/guide/databinding [Accessed November 2, 2016]

AngularJS: Developer Guide - E2E Testing. [Online]. Available at: https://docs.angularjs.org/guide/e2e-testing [Accessed November 2, 2016]

AngularJS: Developer Guide: Introduction. [Online].
Available at: https://docs.angularjs.org/guide/introduction. [Accessed October 14, 2016]

AngularJS: Developer Guide - Unit Testing. [Online]. Available at: https://docs.angularjs.org/guide/unit-testing. [Accessed November 2, 2016].

Cho, J, Joey, "An Exploratory Study on Issues and Challenges of Agile Software Development with Scrum" (2010). *All Graduate Theses and Dissertations*. Paper 599. Utah State University.

Coatanéa, E, Roca, R, Mokhtarian, H, Mokammel, F & Ikkala, K 2016, 'A conceptual modelling and simulation framework for system Design', *COMPUTING IN SCIENCE AND ENGINEERING*, Vol 18, no. 4, pp. 42-52.

GoJS Introduction, Northwoods Software. [Online]. Available at: http://gojs.net/latest/intro/index.html. [Accessed October 14, 2016].

Google, Usability Requirements Template, 2016. [Online] Available at:
https://sites.google.com/site/superuserfriendly/templates/usability-requirements-template.
[Accessed on August 11, 2016].

LiveReload. [Online]. Available at: http://livereload.com/. [Accessed October 14, 2016].

Material design guidelines, Patterns – Navigation. [Online]. Available at: https://material.google.com/patterns/navigation.html [Accessed November 1, 2016].

Material design guidelines, Patterns – Confirmation and Acknowledgement. [Online]. Available at:
https://material.google.com/patterns/confirmation-acknowledgement.html [Accessed on November 1, 2016].

Robinson, S. 2013. *Conceptual Modelling for Simulation*. Proceedings of the 2013 Winter Simulation Conference, pp 377 – 388.

Rowley J. 2002. *Using Case Studies in Research.* Management Research News, Vol. 25 Number 1, pp. 17-27.

Smith, B. 2014. *MVC A Compound Pattern*. Advanced Action Script 3, pp. 343-356.

Twitter Bootstrap 2.0.2 Documentation. [Online]. Available at: http://bootstrapdocs.com/v2.0.2/docs/. [Accessed October 14, 2016].

Vora, P. 2009. Web Application Design Patterns, pp. 15-251.

Yin, R.K. 1994. *Case study research: design and methods.* 2nd edition. Thousand Oaks, CA: Sage.