



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SHARAREH NAGHDI
DEVICE AGNOSTIC ADAPTABLE USER INTERFACE FOR
CLOUD MUVIS

Master of Science thesis

Examiner: Prof. Moncef Gabbouj,
Prof. Yevgeni Koucheryavy
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineer-
ing on 4th June 2014

ABSTRACT

SHARAREH NAGHDI: Device Agnostic Adaptable User Interface for Cloud MUVIS

Tampere University of technology

Master of Science Thesis, 50 pages

December 2014

Master's Degree Programme in Information Technology

Major: Wireless Communication Circuits and Systems

Examiner: Prof. Moncef Gabbouj, Prof. Yevgeni Koucheryav

Keywords: Adaptable, User Interface, Responsive Web Design, C-MUVIS

In recent years, adaptable user interfaces have been broadly considered as the main interactive parts of a software system because of variety of smart devices. New smart devices with different screen sizes are produced every month. For this reason, adaptability of UIs has become one of the big challenges for developers.

In this thesis, an adaptable web based UI is developed for a Cloud Multimedia Video Indexing and Retrieval System (C-MUVIS). Since C-MUVIS is a content-based image retrieval system and will act as a web service, a wide range of screen sizes should be considered when designing the UI.

The adaptable UI has two applications, DbsEditor and MBrowser. In DbsEditor mode, the users can modify the attributes of a database, for instance, to create a new database, add media to a database, delete existing media from the database, or add or remove visual features. The MBrowser Mode is used for browsing and displaying the media in the database. Also, the users can query for a specific media item. The query operation returns the items most relevant media to the queried media.

In order to make the UI adaptable, we introduced Adaptive Web Design (AWD) and Responsive Web Design (RWD) methods. After comparing the advantages and disadvantages of both approaches, RWD was chosen. RWD is executable with Fluid Grid and Media Queries techniques.

As a result, we show that decreasing the screen resolution from the main width (1280px) and height (1024px) to the breakpoints resolution (width 768px and height 1024px) modified the layout of the UI smoothly. But, the layout of the UI will be changed completely for the breakpoint with different scales, colors and images size.

PREFACE

This work has been conducted at the Department of Signal Processing of Tampere University of Technology within D2I project.

I would like to express my sincere gratitude to my supervisor, Prof. Moncef Gabbouj who gave me the opportunity to be part of the MUVIS research group. It has been an honor to work with him. This thesis owes its existence to his help and feedback. Special thanks to Dr. Iftikhar Ahmad for his technical supervision during the work.

I would like to thank Prof. Yevgeni Koucheryavy especially for acting as an examiner.

I would also like to thank Dr. Payman Aflaki not only for guiding me in writing the thesis, but also for being one of my best friends in Finland.

I would also like to thank my dearest friends Dr. Shadi Ahmadi and Dr. Shahram Karami who have always supported me.

My warm thanks go to my friends Zahra Abbaszadeh and Golnaz Sabet Nejad whose company made these years very memorable for me.

Last, but not least, I would like to thank my parents for their support and love. I owe more than thanks to my sister, Dr. Shamim Naghdi, for supporting and encouraging me throughout this experience. Finally, I dedicate this thesis to my family.

Tampere, 9.12.2014

Sharareh Naghdi

CONTENTS

1.	INTRODUCTION	1
2.	BACKGROUND AND RELATED WORK	2
2.1	MUVIS Framework.....	2
2.2	Other Image Search Engines	3
2.3	Mobile MUVIS	4
2.4	Adaptable User Interface.....	4
3.	CLOUD MUVIS AND ITS UI	5
3.1	Cloud MUVIS Architecture	5
3.2	Block Diagram of the System	5
3.3	Client-Server Paradigm	8
3.4	Front-end User Interface Software.....	9
3.4.1.	HTML5 and Device APIs	9
3.4.2.	Cascading Style Sheets 3 (CSS3).....	9
3.4.3.	JavaScript Framework.....	9
3.5	Functional Requirement for UI	10
3.6	Requirement for Adaptive UI.....	10
3.7	UI Responsibilities	11
4.	IMPLEMENTATION PLAN	13
4.1	User Interface Design.....	13
4.1.1.	Fluid Grid.....	13
4.1.2.	Flexible Images and Media Queries.....	14
4.1.3.	Implementation Plan for Adaptable UI.....	14
4.2	Client-Server Architecture Style	15
4.3	Web Application Technology	16
4.4	Error Handling.....	17
4.5	Usability of C-MUVIS UI.....	17
5.	C-MUVIS UI VIEWS AND RESULTS	19
5.1	Introduction	19
5.2	Design and Implementation of Adaptable C-MUVIS UI.....	19
5.3	Implementation Architecture.....	26
5.4	DbEditor	30
5.4.1	Create New Database	30
5.4.2	Load Created Database	32
5.4.3	Appending Images into Database	32
5.4.4	Removing Images from Database	33
5.4.5	Extracting Visual Feature.....	33
5.4.6	Removing Visual Feature.....	36
5.4.7	ADD HTC Indexing.....	38
5.4.8	Remove Indexing	40

5.5	MBrowser.....	41
5.5.1	Get List of Database.....	41
5.5.2	Query by Indexing	42
5.5.3	Query by Media	42
6.	CONCLUSIONS AND FUTURE WORKS	44

LIST OF FIGURES

<i>Figure 1: DbsEditor Graphical User Interface (GUI) PC-Base</i>	<i>2</i>
<i>Figure 2: MBrowser GUI PC-Base</i>	<i>3</i>
<i>Figure 3: General C-MUVIS Architect.....</i>	<i>6</i>
<i>Figure 4: General Master and Worker Architect</i>	<i>6</i>
<i>Figure 5: C-MUVIS Application Block Diagram</i>	<i>6</i>
<i>Figure 6: DbsEditor Application UI Layout.....</i>	<i>7</i>
<i>Figure 7: Mbrowsers Application UI Layout</i>	<i>8</i>
<i>Figure 8: Client-Server Architect</i>	<i>8</i>
<i>Figure 9: The Role of C-MUVIS UI.....</i>	<i>12</i>
<i>Figure 10: The Role of UI.....</i>	<i>13</i>
<i>Figure 11: Two-Tier Client-Server Architecture</i>	<i>16</i>
<i>Figure 12: Three-Tier Client-Server Architecture.....</i>	<i>16</i>
<i>Figure 13: The DbsEditor UI on 1280×1020 Resolution.....</i>	<i>22</i>
<i>Figure 14: The DbsEditor UI on 768×1020 Resolution.....</i>	<i>22</i>
<i>Figure 15: The MBrowser UI on 1280×1020 Resolution.....</i>	<i>25</i>
<i>Figure 16: The MBrowser UI on 768×1020 Resolution.....</i>	<i>25</i>
<i>Figure 17: Screen resolution is measured by the Sizer v3.34 on the MBrowser UI.....</i>	<i>26</i>
<i>Figure 18: The DbsEditor UI Implementation Architecture.....</i>	<i>27</i>
<i>Figure 19: The MBrowser UI Implementation Architecture</i>	<i>28</i>
<i>Figure 20: Architecture of C-MUVIS UI Connection to Master.</i>	<i>29</i>
<i>Figure 21: Create a New Database</i>	<i>30</i>
<i>Figure 22: Append Visual Features.....</i>	<i>35</i>
<i>Figure 23: ADD HCT Indexing</i>	<i>38</i>
<i>Figure 24: MBrowser with No Current Action.....</i>	<i>41</i>

LIST OF SYMBOLS AND ABBREVIATIONS

AAC	Advanced Audio Coding
AWD	Adaptive Web Design
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CBIR	Content-Based Images Retrieval
CSS	Cascading Style Sheets
C-MUVIS	Cloud Multimedia Video Indexing and Retrieval System
DLL	Dynamically Linked Library
DOM	Document Object Model
FeX	Feature extraction
GUI	Graphical User Interface
HTC	Hierarchical Cellular Tree
hdbs	Hybrid Databases
HTML5	Hyper Text Markup Language 5
HTTP	Hypertext Transfer Protocol
idbs	Image Databases
mp3	Picture Experts Group Layer-3
MPEG-4	Motion Picture Experts Group Layer-4
MUVIS	Multimedia Video Indexing and Retrieval System
M-MUVIS	Mobile MUVIS
MS	Microsoft System
QBIC	Query By Image Content
RWD	Responsive Web Design
UI	User Interface

URL	Uniform Resource Locator
vdfs	Video Databases
WUI	Web User Interface
WWW	World Wide Web
XML	Extensible Markup Language
YUV	Luminance-Bandwidth-Chrominance 4:2:0

1. INTRODUCTION

The user of a software system interacts with it via software's User Interface (UI). Considering the increasing number of variety of devices adaptation, the UI has become a big challenge. The first step in the development of a UI is the creation of UI architecture [45], [36].

Image retrieval is a method and process to search among images in a dataset [40], [50]. The exponential growth in the digital world has led to increased amounts of media data [1] and storage facility capacity. This growth has led to a need of retrieval of media information from large databases, based on which, retrieval of media becomes a challenge [1]. For many years, most image retrieval systems have performed the search process based on keyword and text generated by humans which is referred to as text-based [1], [20]. In the text-based image retrieval system, the text label is created by human observation and may lead to many difficulties [37], [48].

Content-based images retrieval (CBIR) was introduced to overcome the problem of text-based image retrieval [37], [48]. The CBIR system retrieves images based on similarities in their content such as textures, colors, shapes etc. [47], [55]. One of the content-based indexing and retrieval techniques that has been developed is "Multimedia Video Indexing and Retrieval System" (MUVIS) [40].

This thesis explores the adaptable web based UI development for a Cloud Multimedia Video Indexing and Retrieval System (C-MUVIS). Since C-MUVIS is an expanding system based on the MUVIS, the adaptable UI can run locally or remotely on the network by the users. UI clients are connected by Web-Socket server to the back-end.

The structure of this thesis is as follows. Chapter 2 introduces MUVIS, a framework for content-based multimedia indexing and retrieval, which can be considered as the foundation of C-MUVIS. In this chapter, other client-server architecture of MUVIS, M-MUVIS are introduced and compared to C-MUVIS. Chapter 3 gives an overview of framework architecture and the related functional and adaptability requirements. Chapter 4 presents possible technologies for an adaptable UI implementation while chapter 5 reviews the design for adaptation framework and results. Finally, Chapter 6 states the conclusions of the thesis by summarizing the achievements and reporting the potential future working areas.

2. BACKGROUND AND RELATED WORK

2.1 MUVIS Framework

MUVIS is a content-based indexing and retrieval framework for large multimedia databases. The first MUVIS system was developed in the late 90's to support indexing large image databases and retrieval based on visual and semantic features such as color and texture [49].

Recently, based on this first system, an MS windows MUVIS system has been developed, which is more capable of indexing and retrieval of video, audio and several types of images [40]. Current varieties of file formats, codec types, and encoding parameters could affect retrieval systems [49]. For this reason MUVIS is developed to cover a wide range of multimedia formats [50]. MUVIS consists of novel methods in progressive query, indexing and browsing which provides a faster system [50]. The MUVIS system is based upon several applications. Each of these applications has different responsibilities [49].

DbEditor is the application for creating, editing, and indexing of the multimedia databases. A snap shot of this application is depicted in Figure 1. As shown in Figure 2; another application is MBrowser that integrates browsing and retrieval into a MUVIS database [40].

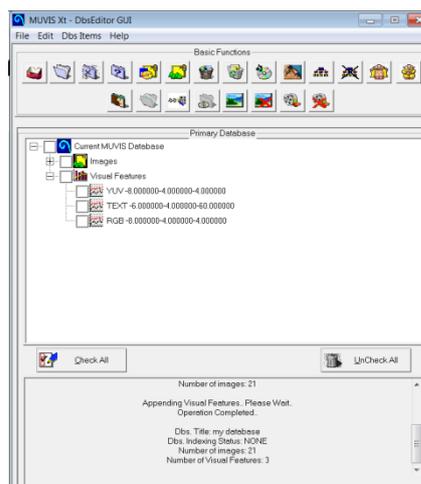


Figure 1: DbEditor Graphical User Interface (GUI) PC-Base

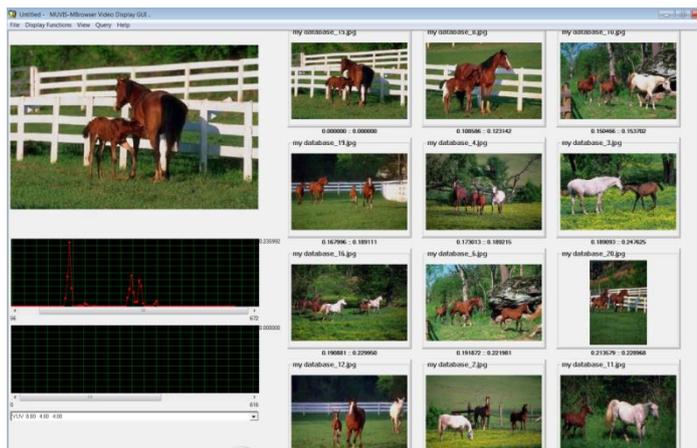


Figure 2: MBrowser GUI PC-Base

2.2 Other Image Search Engines

Recently, several web image search engines have been developed such as WebSEEk [26], Webseer [10], ImageRover [53], MIRROR [32], and WebMARS [42].

The authors of [26] have described the WebSEEk as a content-based image and video query tool for the web. WebSEEk utilizes text and visual information synergistically to search the images and videos. The system has several functionalities such as searching using image content-based techniques, query modification using content-based relevance feedback, and automated collection of visual information. In this system, the user interface is coded in HTML to facilitate its access from a web browser [27].

In [10], authors explain that the Webseer system is another World Wide Web (WWW) image search engine. The system uses image content, in addition to associated text to index. The Webseer system detects faces within the images and lets users search by the number of faces. In this system, all executable parts are written in C++ and run on a M.S. Windows NT 3.51 platform with the exception of the URL server, which is written in Java.

ImageRover is a content-based image browser for WWW. A separate query server running at the ImageRover WWW site and user interface is coded in HTML. The UI shows a set of randomly selected images to the user to be marked as ‘relevant’ and asks the system for searching. Each thumbnail image is linked to the original image and the user can retrieve the image from its original home website by clicking on the thumbnail[53].

The MIRROR system works based on user’s relevance feedback and MPEG-7 Experimentation mode. To start the search, the user chooses a reference image and a search method, and the system returns the images with the highest ranks at the center of the screen in similarity descending order. The query image is shown at the top of the screen.

The system allows users to select relevant images triggering the search process for the next search [32].

WebMARS is a multimedia web search engine that utilizes both textual and visual information for HTML document retrieval. One of the important parts of this system is the collection of relevance feedback data in a simple and intuitive user interface [42].

2.3 Mobile MUVIS

Previous MUVIS client-server architecture is Mobile MUVIS (M-MUVIS) that implement to bring the MUVIS into a wireless area device [1], [20]. The client application runs on a mobile phone which is handled by a servlet. The UI is displayed on the mobile phone with the capability of showing just one image at a time, then the user can view the images one by one [1], [20].

Comparing C-MUVIS UI to M-MUVIS UI, several similarities and differences can be mentioned. Firstly, we consider similarities such as both of them are implemented as client-server of MUVIS system and both applications DbsEditor and MBrowser are available on them [21]. On the other hand, differences show C-MUVIS UI is adaptable and works with web server as it will be implemented during next chapters with markup languages. However, M-MUVIS UI was supported by Java, and because normally this type of phone has a small screen, the user can view images in the query operation back and forth, and one by one [21].

2.4 Adaptable User Interface

One of the main reasons for the increasing interest in user interface adaptation is that applications interact with a larger and larger variety of smart environments [29], [18]. Adaptation occurs in the layout and elements based on the needs of the users or context [44]. By technological evolution in smart devices, the users engage in different types of screens depending on the screen size they are using such as: a TV screen, a laptop, a tablet, or a smartphone [29], [7]. New devices with new screen sizes are stepping forward every day, and the users prefer the same user interface of an application on the different devices but not a small user interface on a desktop monitor or a large user interface on a small device [18], [52]. For this reason, the user interface should be designed to adapt to screen size and resolution [24]. Some developers implement different styles of UI for different sizes of displays. However, it is impossible to design a wide range of UI, and no one can confidently describe the potential screen sizes that will be produced in the futures [24], [15].

3. CLOUD MUVIS AND ITS UI

3.1 Cloud MUVIS Architecture

MUVIS as a PC-based system which supports indexing, browsing and querying on various multimedia types is a type of Content-Based image retrieval system[21]. The MUVIS system is developed as a framework for capturing, recording, content-based indexing and retrieval, combined with browsing and various other visual and semantic capabilities [40].

The Cloud MUVIS (C-MUVIS) system aims to achieve MUVIS beyond the desktop environment which would have client-server computing architecture and is able to deal with very large image databases, even Big Data.

The proposed C-MUVIS system is based upon several objectives and ideas. First of all, it is intended to support huge databases for query and indexing. Second, it is able to connect remotely to applications without any limitation on data capacity. Finally, it is intended to be accessible from Windows and Linux operation systems.

C-MUVIS consists of client and server applications. The C-MUVIS server has C++ code consisting of two vital parts, the Master and Worker(s). The Master is the communication part and the intermediary between the Worker and client-sides. The Master will distribute received tasks from the client-side to Workers. Moreover, it combines Worker's results and updates them on the frontend. Each Worker is run separately with assigned partitioned data working by the Master. Feature extraction (FeX) module(s) for calculating similarity distances in query operations

The C-MUVIS client application consists of two components. The primary component is User Interface (UI) that is responsible for presenting a graphical area for creating a database, adding or removing a feature, adding or removing media, querying media, and displaying of results from the server-side. The other component is communication protocol between the C-MUVIS client and server. This protocol uses Hypertext Transfer Protocol (HTTP) to exchange information. HTTP is also used as the underlying protocol [21].

3.2 Block Diagram of the System

Figure 3 and Figure 4 illustrate the basic C-MUVIS framework architect.

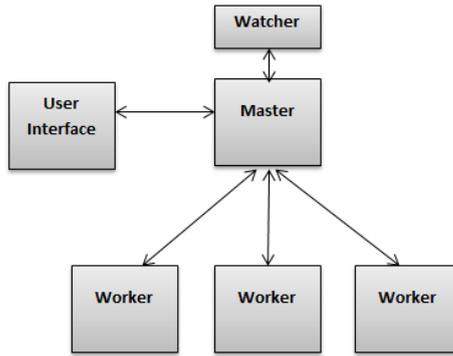


Figure 3: General C-MUVIS Architect

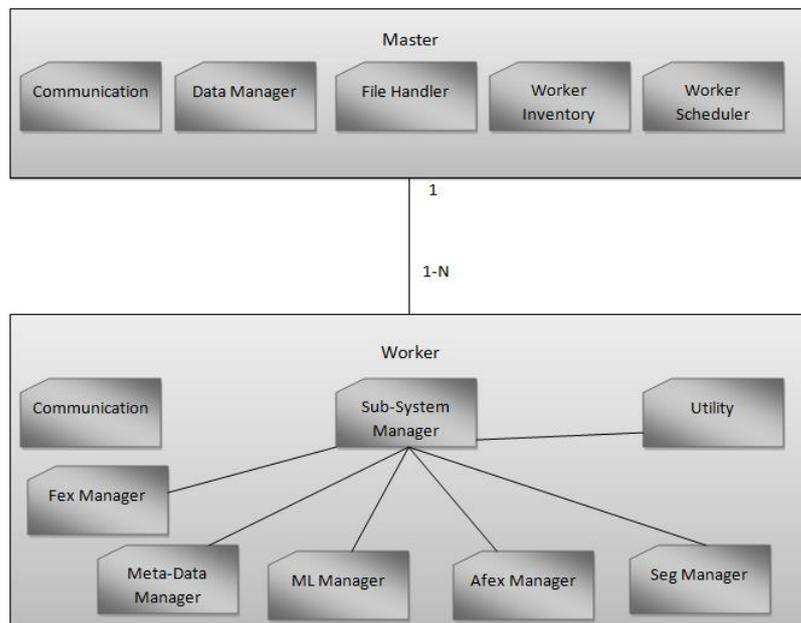


Figure 4: General Master and Worker Architect

Figure 5 illustrates the basic applications within the C-MUVIS framework.

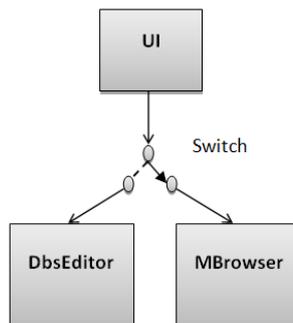


Figure 5: C-MUVIS Application Block Diagram

The DbsEditor application is deployed on the user interface to edit database to add or remove indexing, visual features and images. One of the most important tasks of this application is to create a C-MUVIS database which can add or remove media to any other C-MUVIS database in any path that the user wishes. Another important task is feature extraction; as a result, DbsEditor is capable of adding and removing features to and from the C-MUVIS database [50].

The MBrowser application is designed for browsing and retrieving media within the C-MUVIS database. Its query works based on similarity distance(s) between feature vector(s) of queried items with feature vector(s) available in the database. However, this is not valid when a classification algorithm is utilized, e.g. CNBC.

The DbsEditor application UI is partitioned into three views. The layout of the interface is depicted in Figure 6. The main view is the area in which the databases and all popup windows are presented to the user. All drag-drop and progress bar events also occur in the same area. The menu view is the selection menu with links to various parts of the DbsEditor application UI. In this part, the user can choose a link or button where actions are to be performed. The mode view is a button for changing the mode between DbsEditor application UI and MBrowser application UI.

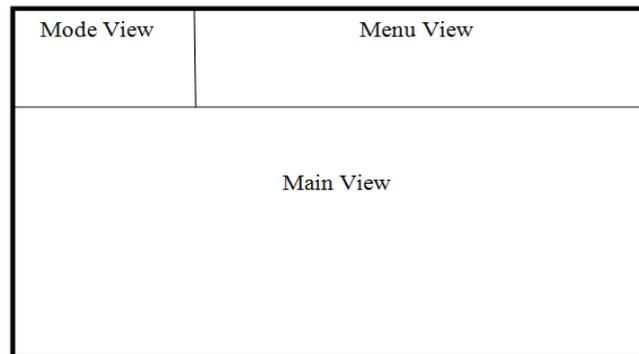


Figure 6: *DbsEditor Application UI Layout*

The MBrowser application UI is partitioned into five view areas, as shown in Figure 7. As explained previously, the mode view and menu view are for changing the mode of application and finding the menu respectively. The Image holder view is a box for holding the selected image for the query. The Uniform Resource Locator (URL) view is also a box for a URL of an image to be queried. The main view is the area in which the databases and the results of the query are presented.

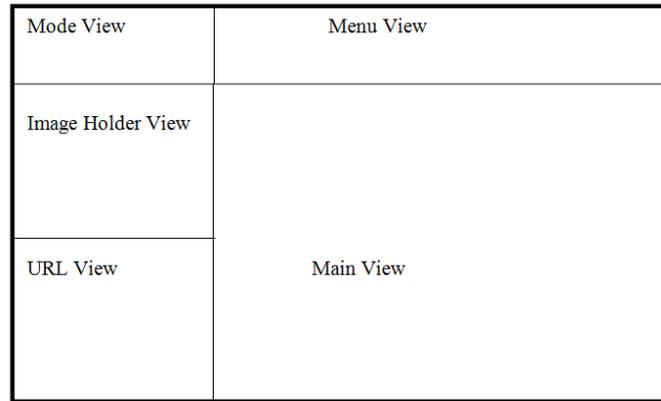


Figure 7: Mbrowser Application UI Layout

3.3 Client-Server Paradigm

The client-server application describes a piece of software that runs on a client computer and requests an action or service from a remote server. Both the client and server model can be used by programs within a unique computer. In the case of remote access, the client connects to the server over a local or wide area network (LAN, WAN) although the client needs not be aware of the physical location of the server. Received requests from the client will be processed according to the input data, and the results of the processing are sent back to the client in response, as illustrated in Figure 8.

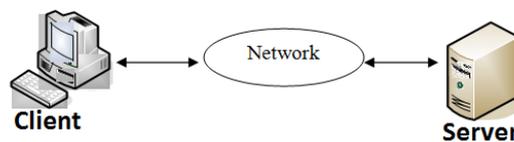


Figure 8: Client-Server Architect

As explained before, the C-MUVIS server consists of Master and Worker(s) which are native C/C++ processes communicating over socket. When the client sends a request to the server, the Master will parse and distribute the task to the Worker(s). All the tasks will be in Extensible Markup Language (XML) format and consist of a unique task ID tag for performing a global API task. After the processing is complete in the Worker(s), the results will be sent back in XML format with the results state tag showing the state of the results. The Master configures these results and sends them back to the client-side. In all the communication between the Master and Worker, the database configuration (dbconfig) tag will appear and consist of the database file, database root, and database title.

3.4 Front-end User Interface Software

Recently, development of front-end is a mixture of HTML, CSS, JavaScript and jQuery which are controlled by a browser[8]. Different types of the above-mentioned programming languages are described in the following sub-sections.

3.4.1. HTML5 and Device APIs

Hyper Text Markup Language 5 (HTML5) has become one of the main foundations for creating user interface web applications. HTML5 is the fifth major revision of HTML, a markup language used for structuring and presenting content on the web. In October 2009 the first draft of HTML5 specifications was released, introducing HTML5 – 2010.

One of the main goals of HTML5 is to make development easier for web applications. Hence, HTML determines new Document Object Model (DOM) Application Programming Interfaces (APIs) for server-sent events, media, location, drag and drop and many other events. The HTML5 is becoming a reasonable solution to create applications with good browser support because new interfaces are implemented to JavaScript via objects and have made it easier to write web applications [38].

In adaptable C-MUVIS UI, HTML5 language is selected as the markup language to allow the browser to read the front-end and display it. The techniques will be explained later for adaptability in UI will be supported by HTML5 [9].

3.4.2. Cascading Style Sheets 3 (CSS3)

Cascading Style Sheets 3 (CSS3) is an extremely powerful style sheet language which is used for controlling the look and feel of the content written in a markup language. CSS3 is the third version of CSS specification. During design of websites, CSS allows developers to create layouts quickly. CSS gives the ability to keep every visual aspect of the website completely separated from the content, and to control all elements from a style sheet.

As mentioned above, a big issue in our framework is adaptability that will be handled by CSS. It is an essential ingredient to an adaptable UI design, and CSS3 modules enable us with previously unseen levels of flexibility.

3.4.3. JavaScript Framework

For many years, the front-end application in JavaScript has been written with difficulty. Recently, many frameworks offer the advantage of using HTML, CSS, and JavaScript. Such frameworks provide structure to the front-end code and help to decrease loads on the network by minimizing the number of HTTP requests. The JavaScript framework is

a collection of pre-written JavaScript code which makes the development of applications easier. Among the most well-known client-side JavaScript frameworks are jQuery and Asynchronous JavaScript and XML (AJAX). The most notable server-side JavaScript frameworks are AppJet and Node.js [19]. Choosing a framework is important to consider, as it sets the requirement list, size, durability, and performance of a project [6].

Another powerful tool in our project is JavaScript which will handle all tasks including drag and drop, pop-up windows, paging, etc. It is an interactive part of our framework at the back-end.

3.5 Functional Requirement for UI

Through the C-MUVIS UI the users are able to handle image indexing and retrieval on big data. In this case, creating a database, appending images, removing images, appending features, removing features, editing a database, querying by media, and querying by name have been vital functions that the UI needed. For all these functions the UI should communicate with the Master.

Sending information and receiving a response are provided for the user by clicking and the drag and drop action. Informative feedback messages are provided for the users when a process is successfully completed.

There are some blocked functions for restricting user activity when the UI and the Master are in communication processes. These functions will appear as a progress bar on the UI. Also, other functions of UI are: showing the UI in full screen mode, Hierarchical Cellular Tree (HTC) indexing and finding a query image from a URL. C-MUVIS is a retrieval framework that functions through the web. HTTP is used to communicate between the UI and the Master.

3.6 Requirement for Adaptive UI

In this thesis, an adaptable client-side of content-based image retrieval framework is developed. It was a major challenge to provide an adaptable user interface for a wide range of screen sizes. In order to overcome to this problem, there are two popular approaches to develop adaptable interactive applications: Adaptive Web Design (AWD), and Responsive Web Design (RWD) [18], [25], [23], [3].

AWD is an approach where focus is on the user and not the browser [18], [3]. The AWD approach uses a predefined set of layout sizes based on screen size of device [39]. In other words, the AWD adapts to the detected device. In this approach, the designer controls all changes on the layout and determines how elements are located for large desktop screens, tablet devices, and mobile phones, etc. [25], [3], [43], [4]. As an exam-

ple, the website Ask.com introduces an adaptive site that alters the layout and functionality according to the end-user requirements and capabilities. It should be noted that, as introduced in [6] fast response times are more important than content to Ask.com mobile users.

RWD is another approach that changes a UI's layout fluidly to fit any screen size of device [18], [25], [23], [3]. In the RWD method, layout expands while all elements of a page change their placement to display properly on the current screen size [18], [25], [23], [3]. Microsoft.com is a responsive site example. The small size screen users are more interactive and visually-based, because users check out their product rather than just hunt for quick information [11].

As explained above, both AWD and RWD address the adaptability issue for our adaptable client-side of content-based image retrieval framework. However, there are differences between those approaches. Understanding these differences will help choose the best approach to meet our requirements. Some of these similarities and differences are reported in the next paragraph.

Both of these approaches try to modify the UI look and experience for different screen sizes. The structure of adaptive and responsive methods is different. The adaptive method relies on devices with predefined screen sizes, and the responsive method relies on fluid grids. In other words, the responsive method utilizes a single design and only the layout shall be adapted to the change of screen size, while the adaptive methods has optimal templates for each device screen. In the AWD approach, images are optimized for screen resolution but in RWD, the images are resized to fit the device.

For several reasons, the RWD is recommended. Three of such examples comprise: 1) a single URL for the content, 2) less required page loading time, and 3) flexible images. Google, as the most popular search engine, has recommended preferential use of RWD to deliver content from a search engine perspective [22], [35].

The focus in our framework is to display images as results of a retrieval system and resize images in different screen sizes. For these requirements, and the above reasons, adaptability is employed along with responsive web design.

3.7 UI Responsibilities

When a user starts the UI, locally or remotely, the UI checks if the Master is running and if so, then the UI connect to the Master. After connecting to the Master, the UI displays the connection message to the user. The Master will take all tasks from the UI, parse it and send it to the available worker(s). All responses received from the workers will be sent back to the UI by the Master and during the response processing handled by Master, the UI will wait for the results.

All the tasks and data fetching from the back-end in the C-MUVIS framework is controlled by the UI. The UI controls how many images are loaded and how the data is displayed to users. Figure 9 shows the role of the C-MUVIS UI.

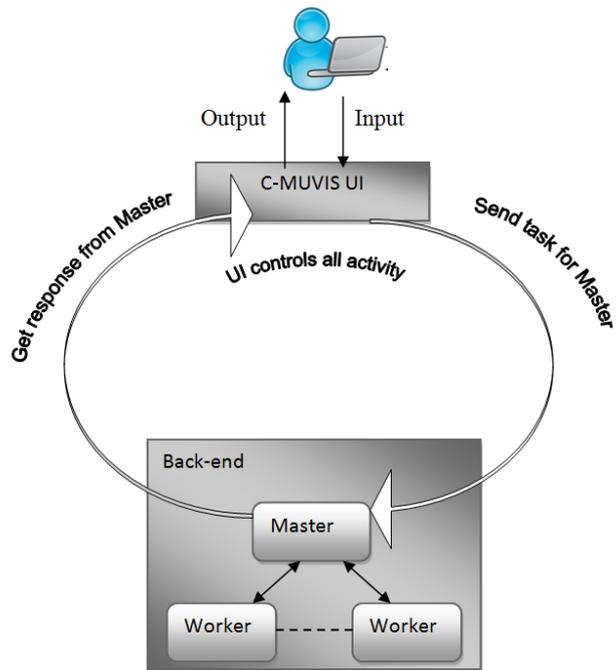


Figure 9: The Role of C-MUVIS UI

There is no separate path (port) for request and response.

4. IMPLEMENTATION PLAN

In chapter 3, C-MUVIS requirements were discussed. Here, several possible technologies for implementation of these requirements are described. The choice of the implementation plan will completely affect the design and performance of the C-MUVIS application.

4.1 User Interface Design

A UI aims to achieve an easy way to implement interaction between application users and devices. There are several types of UIs such as: graphical user interface, web user interface (WUI), voice user interface, or touch user interface etc. Figure 10 shows the role of a UI in a computer system.

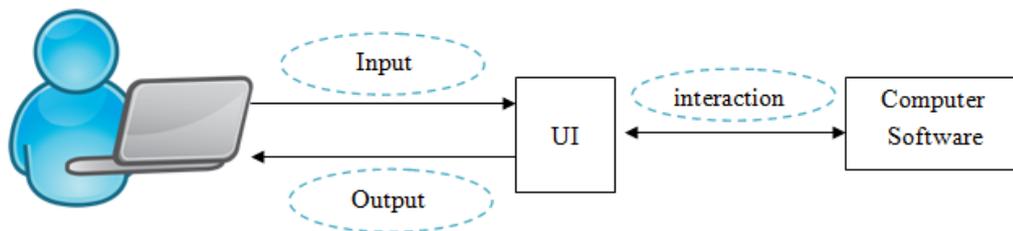


Figure 10: The Role of UI

As explained in Chapter 3, our framework will be implemented by RWD approach. Let us look at the implementation of responsive web design. Implementation of RWD starts with fluid grid, flexible images, and media query [15], [1], [17], [16].

4.1.1. Fluid Grid

One of the techniques to achieve responsive web design is fluid grid. Before the fluid grid technique, most client-servers were implemented with a fixed width style layout. Recently, with wide ranges of screen resolution devices, a fixed width style implementation has a resizable limit [1], [17].

The fluid grid concept is to create a layout where all elements are calculated relative to one another and based on the percentage width. To calculate an element's width, the width of elements is divided by the size of the parent element and the result is multiplied by 100 to get the percentage value.

If N represents the size of the element within pixel and M represents the size of the parent element width in pixel then eq. 4.1 simply reports the percentage value of width for the respective element.

$$\text{Percentage value of width in one element} = (N/M) \times 100 \quad (4.1)$$

However, the disadvantages come out even with fluid-layout design. There is need for min-width or max-width which every browser cannot support these techniques. Component can get too small on small screens. Web developers also add horizontal scrolling if required [31], [30].

4.1.2. Flexible Images and Media Queries

In the adaptable UI, one of important parts is to adjust the images. In fixed terms, widths of images are resized for different screen sizes but not dynamically. For dynamic resizing there are some techniques like CSS cropping, Max-Width and Multiple Image with HTML5, CSS3, and Media Queries [15], [17].

In order to resize images dynamically and retain optimization of loading time [42] the Media Queries technique is considered in our proposed system. In this technique, the users screen resolution will be checked and the proper source will be chosen. High screen resolutions have one source and lower ones have another source [15], [17]. As mentioned earlier, Media Queries are supported by HTML5 and CCS3 while the Syntax of media queries is:

```
@mediascreen and (max-width: ..value..) and (max-height : ..value..){...css code..}.
```

Max-width and max-height are used for checking the resolution range under and above the certain breakpoints [51]. There is a huge list of common device breakpoints [34], but some of them are major common breakpoint for responsive methods [12].

4.1.3. Implementation Plan for Adaptable UI

Next chapter described the process by which we are going to implement our framework on the 768px×1024px breakpoint along with the respective achieved results. First, the focus is on a pc screen with the resolution 1280px×1024px, then, the resolution will be reduced.

As mentioned before in section 3.3, DbsEditor Application UI Layout is partitioned into three views. For implementing Fluid grid and Media Queries techniques on DbsEditor Application UI, our main focus is on the partitions and the image databases. For this reason, the width and height of the main view are calculated relying on the screen resolution. Resizing the images is calculated based on what is the respective proportion of the image size compared to the main view. The height of the menu and the font size of that are also calculated based on what is the respective proportion of the menu and its font size compared to the main body.

When changing the resolution size, the Media Queries technique will be effective to change the CSS style such as size of the images, the width and height of the main view, the width and height of the menu and its elements. The results and calculations will be explained in the next chapter.

For implementation, the MBrowser application UI that is partitioned into five views areas(the main view, URL view, Image holder, menu and image), are calculated with the same approach exploiting Fluid grid and Media Queries technique. However, the screen width is divided into two parts between the URL view and the main view.

4.2 Client-Server Architecture Style

Client-server architecture is a message-based, distributed and modular framework that involves a separate client and server framework and a connecting network. Client and server modules would not necessarily need to be located on the same memory space. On the other hand, client and server modules can also be located on a same platform unit.

The client is a program that sends data and tasks to the server program. The server will perform accordingly to respond to the requests received from the client-side. The UI will be managed by the client program. All inputs will be received and validated by clients from the user then delivered as requests to servers. The server program basically receives requests from the client-side. Also, the server performs databases retrieval and dispatches responses to the client [2].

Figure 11 and Figure 12 show the sample of two-tier and three-tier client-server architecture, respectively.

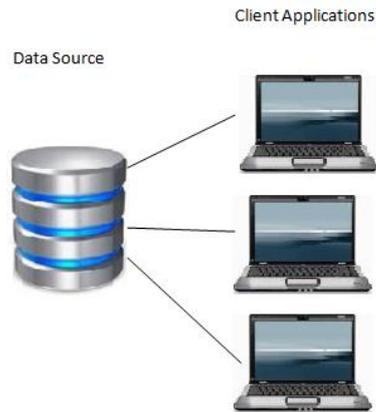


Figure 11: Two-Tier Client-Server Architecture

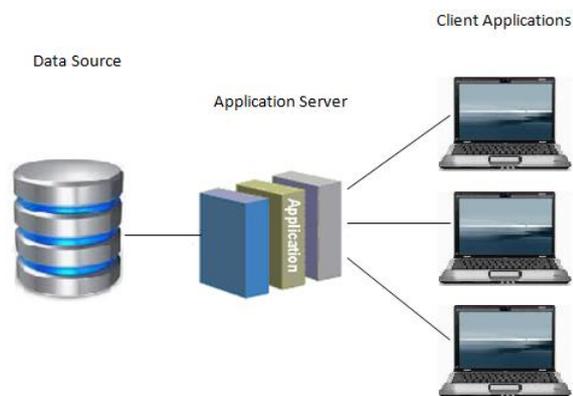


Figure 12: Three-Tier Client-Server Architecture

C-MUVIS is implemented as a three-tier client-server application. The web browser and the HTML page is the top tier. The middle tier is Master and Worker(s) while the database is the third tier. There are dummy fully transparent proxy between the UI and the Master. This proxy does not modify any request or response. For this dummy transparent proxy is needed to use static IP.

4.3 Web Application Technology

From a technical view-point, a web application is a software application that is stored on a remote server and is accessible using a web browser. Web application design is completely different from the traditional systems.

In order to keep user data separately, users of web applications should be identified within a unique session. There are huge numbers of different web browser implementations, and different internet connection speeds that the users of a web application are

using. A web application should be implemented and performed according to standard web technologies such as web hosting or markup languages [5].

In our framework, front-end and back-end can work on a local framework or a remote framework. When the Master is running, the UI is displayed on a browser while all requests and responses between UI and Master are designed as XML format.

4.4 Error Handling

When a web application is running, it might encounter faults that should not occur in the normal operational situations. For example, when a file is missing or a user has entered unsuitable values, or the network is down. A well-designed web client must recover skillfully from such problems. C-MUVIS UI error handling refers to the following three types of errors: UI errors, communication errors, and synchronization errors.

In case of errors that happen inside the C-MUVIS client side, it is of utmost importance that understandable feedback is provided to the user. User friendly error messages should be displayed, for example, browser is not supported.

In a scenario that a user provide a wrong input, the user should be well informed of the wrong input and asked to try again. The error messages should be comprehensive and yet understandable for the user. In the C-MUVIS web client application, there can be many reasons for such mistakes e.g. dragging and dropping improper files in the DbsEditor or Mbrowsers, opening a feature extraction window, adding images before creating or loading a database, or typing a wrong path for creating a database.

In web client, communication error happens during interacting the UI and Master or in the case of potential network breakdown. After running the UI and connecting it to the Master through a socket server with a Master on a port, a pop-up windows message is displayed to inform the user about the connection status. If any errors occur, then a respective error message will appear and the connection fails. As the client is using socket but just state (task-based), it is a stateless protocol that helps you to maintain state.

In case of synchronization errors, there exists a temporal miss alignment between the UI and Master. Such error will be handled by means of restarting the connection for a well synchronized framework. It is also possible to inform the user with a pop-up message saying that such re-synchronization has been done, this is a subject for future work.

4.5 Usability of C-MUVIS UI

In the C-MUVIS UI usability should consider as one of the important factors. One of the usability issues in the C-MUVIS UI is drag and drop. For security reasons browsers do not allow fetching full path of file from a file systems in JavaScript and hence, the

browser has no access to the file system [46]. That is why we have to drag and drop XML files and images in the frameworks instead of opening the full path of files. In the C-MUVIS UI the full path for saving the database is asked the user to type it and following this, the server creates a 'dbs' file and an XML file after creating a database in this path. All tasks in the C-MUVIS framework are in the XML configuration.

In C-MUVIS UI, the first step is to read the dragged file and to determine its format. For XML and image files, the drop event will activate the following steps of the process while any other format results into activation of error handling. The drop event is implemented by two methods of 'event.stopPropagation' and 'event.preventDefault' while the former method stops the eruptive of an event to parent elements and the latter method stops the default action of an element from happening.

One of the other usability is preparing appropriate feedback. Feedback for all images should be provided as thumb up and thumb down that shows the relevancy of query images with the given image. During the implementation of UI the back-end was not prepared for the score of relevancy and for this reason this is a topic for future work.

5. C-MUVIS UI VIEWS AND RESULTS

This chapter goes through the technology specific design and implementation of the C-MUVIS UI concept presented in the previous chapter. Moreover, the calculations and the results of implementation are presented.

5.1 Introduction

The C-MUVIS architecture framework which enables indexing, browsing and querying the images is comprised of client and server architecture. This framework is designed to work as a local application or a client-server application used as a web service on big data. We present the framework for adaptive user interface generation; that adaptation occurs during resolution size change. The adaptable UI layout will fluidly move when changing the screen resolution.

As explained previously, the C-MUVIS framework consists of server-side and client-side. The client-side is designed to implement two applications in the same user interface. One of these applications is DbsEditor for creating initial image databases in C-MUVIS. This application could append and remove features from the database as well as appending or removing external images to any C-MUVIS database. Another one of these applications is Mbrowse that supports multi feature queries. The feature extraction module generates feature vectors and stores them in their associated feature files.

For selecting DbsEditor or Mbrowse applications, a push button is designed in the user interface. With the selection, one application will be activated and the other one will be blocked.

5.2 Design and Implementation of Adaptable C-MUVIS UI

The breakpoint is a width less than 768px and height less than 1024px as screen resolution. As described in Chapter 4, in the DbsEditor mode main view is resized on the screen resolution then:

$$WMV = SW - MP - TP$$

Where:

WMV represents width of the main view

SW represents the screen width (1280px)

MP represents the main padding (2×10 px)

TP represents table padding ($2 \times 8\text{px}$)

WMV = 1244px

$$\text{H MV} = \text{SH} - \text{MP} - \text{TP} - \text{HM} - \text{P}$$

Where:

H MV represents height of the main view

SH represents the screen height (1024px)

MP represents the main padding ($2 \times 10\text{px}$)

TP represents table padding ($2 \times 8\text{px}$)

HM represents the height menu (40px)

P represents padding (168px)

H MV = 800px

According to the equation (4.1):

Percentage value of WMV = $(\text{WMV}/\text{SW}) \times 100 = 96\%$

Percentage value of H MV = $(\text{H MV}/\text{SH}) \times 100 = 78\%$

Percentage value of HM = $(\text{HM}/\text{SH}) \times 100 = 4\%$

The size of displayed images is calculated in relation to width and height of the main view e.g. if there exists 40 images on one page with size 150px×144px then:

Width of image = $(150\text{px}/1244\text{px}) \times 100 = 12\%$

Height of image = $(144\text{px}/800\text{px}) \times 100 = 18\%$

Calculated above parameters in resolution screen size less than width 768px and height 1024px are:

$$\text{W MV} = \text{SW} - \text{MP} - \text{TP}$$

Where:

W MV represents width of the main view

SW represents the screen width (768px)

MP represents the main padding ($2 \times 15\text{px}$)

TP represents table padding ($2 \times 10\text{px}$)

W MV = 718px

$$\text{H MV} = \text{SH} - \text{MP} - \text{TP} - \text{HM} - \text{P}$$

Where:

H MV represents height of the main view

SH represents the screen height (1024px)

MP represents the main padding ($2 \times 10\text{px}$)

TP represents table padding ($2 \times 8\text{px}$)

HM represents the height menu (40px)

P represents padding (168px)

HMV = 800px

According to the equation (4.1):

Percentage value of WMV = $(WMV/SW) \times 100 = 93\%$

Percentage value of HMV = $(HMV/SH) \times 100 = 78\%$

Percentage value of HM = $(HM/SH) \times 100 = 4\%$

Number of images is assumed 42 per page with 115px \times 112px.

Width of image = $(115px/718px) \times 100 = 16\%$

Height of image = $(112px/800px) \times 100 = 14\%$

These width and height are presented at CSS style such “cssmenu” and “images” (Program 1) located at style.css file.

```
#cssmenu {
  height: 4%;
  background-color: #A4A4A4;
  width: auto;
}

.images {
  height:18%;
  width:12%;
  margin:0 0.5% 0 0;
}
```

Program 1: Example of CSS style

As discussed before, to implement a responsive web design approach, there are some techniques such as Fluid grid and Media Queries. The above calculations present the fluid grid, and the next step is the effect of Media Queries technique. According to our breakpoint on width 768px and height 1024px the class will be changed (Program 2). Figure 13 and Figure 14 show the results.

```
@media screen and (max-width: 768px) and (max-height: 1024px) {
  .images {
    height:14%;
    width:16%;
    margin:0 0.5% 0 0;
  }
  #cssmenu {
    height: 4%;
    background-color: #A4A4A4;
    width: auto;
  }
}
```

Program 2: Example of CSS style with Media Query technique

In addition, the number of images will be changed during the resizing process applied to the resolution screen size. Here, as a paging technique we use jQuery, and now with the match media technique, Media Query will be supported for any resolution resizing.



Figure 13: The DbsEditor UI on 1280×1020 Resolution

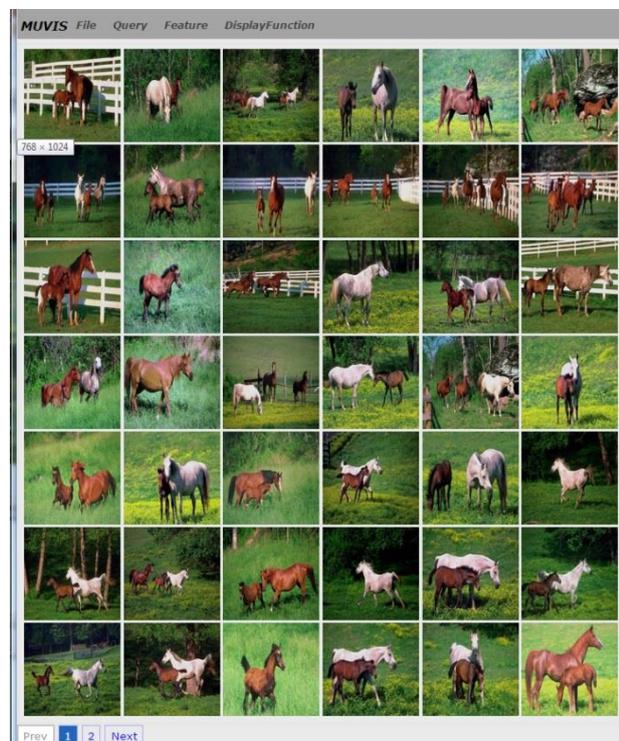


Figure 14: The DbsEditor UI on 768×1020 Resolution

The Mbrowser application UI is implemented with fluid grid technique as below:

$$WMV = SW - MP - IHC$$

Where:

WMV represents width of the main view

SW represents the screen width (1280px)

MP represents the main padding ($2 \times 8\text{px}$)

IHC represents image holder column (251px)

$$WMV = 994\text{px}$$

$$HMV = SH - MP - TP - HM - P$$

Where:

HMV represents height of the main view

SH represents the screen height (1024px)

MP represents the main padding ($2 \times 10\text{px}$)

TP represents table padding ($2 \times 8\text{px}$)

HM represents the height menu (40px)

P represents padding (168px)

$$HMV = 800\text{px}$$

According to the equation (4.1):

$$\text{Percentage value of WMV} = (WMV/SW) \times 100 = 78\%$$

$$\text{Percentage value of HMV} = (HMV/SH) \times 100 = 78\%$$

$$\text{Percentage value of HM} = (HM/SH) \times 100 = 4\%$$

$$\text{Percentage value of IHC} = (IHC/SW) \times 100 = 19\%$$

The size of displayed images is calculated in relation to the width and height of the main view. Considering a page with 30 images in it where the page size is 150px×144px then:

$$\text{Width of image} = (150\text{px}/1011\text{px}) \times 100 = 15\%$$

$$\text{Height of image} = (144/800) \times 100 = 18\%$$

Parameters in MBrowser mode and resolution screen size less than width 768px and height 1024px are:

$$WMV = SW - MP - IHC$$

Where:

WMV represents width of the main view

SW represents the screen width (768px)

MP represents the main padding ($2 \times 25\text{px}$)

TP represents image holder column (178px)

$$WMV = 540\text{px}$$

$$HMV = SH - MP - TP - HM - P$$

Where:

HMV represents height of the main view

SH represents the screen height (1024px)

MP represents the main padding ($2 \times 10\text{px}$)

TP represents table padding ($2 \times 8\text{px}$)

HM represents the height menu (40px)

P represents padding (168px)

$$HMV = 800\text{px}$$

According to the equation (4.1):

$$\text{Percentage value of WMV} = (WMV/SW) \times 100 = 70\%$$

$$\text{Percentage value of HMV} = (HMV/SH) \times 100 = 78\%$$

$$\text{Percentage value of HM} = (HM/SH) \times 100 = 4\%$$

$$\text{Percentage value of IHC} = (IHC/SW) \times 100 = 23\%$$

Number of images is assumed 24 per page with $119 \times 128\text{px}$.

$$\text{Width of image} = (119\text{px}/540\text{px}) \times 100 = 22\%$$

$$\text{Height of image} = (128\text{px}/800\text{px}) \times 100 = 16\%$$

“imagesMBrowser” is a CSS style for MBrowser mode (Program 3).

```
.imagesMBrowser {
    height:18%;
    width:15%;
    margin:0 0.9% 0 0;
}

@media screen and (max-width: 768px) and (max-height: 1024px) {

    .imagesMBrowser {
        height:16%;
        width:22%;
        margin:0 0.9% 0 0;
    }
}
```

Program 3: Example of CSS style and Media Query technique

The results are illustrated in Figure 15 and Figure 16.

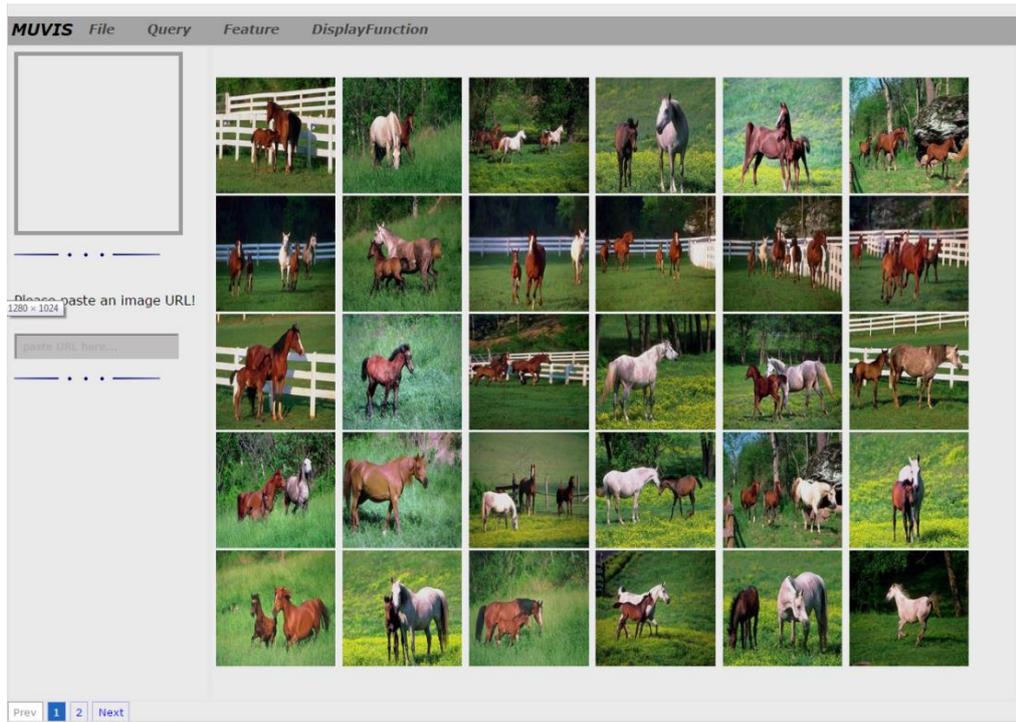


Figure 15: The MBrowser UI on 1280×1024 Resolution



Figure 16: The MBrowser UI on 768×1024 Resolution

Changing the on screen resolution is measured by the Sizer v3.34 (Appa 1997-2012) application [8]. The Sizer is a utility that allows you to see how the browser will look when viewed at different resolution sizes [28].

The resolution size will appear as a small box on the mouse when the size of the browser is changing. This is shown in Figure 17. Decreasing the browser width from 1280px to 769px will happen smoothly and fluently, and when less than 768px, the class style will be different. In the below figure, the CSS style is changed first and then the font size, holder size, and other elements are adapted to the new size accordingly.



Figure 17: Screen resolution is measured by the Sizer v3.34 on the MBrowser UI

5.3 Implementation Architecture

The functional design of C-MUVIS UI and layout architecture were introduced in Chapter 3. In this section implementation architectures of the C-MUVIS UI based on HTML, JavaScript, CSS and XML are illustrated in Figure 18 and Figure 19. The main differences between these figures are based on the XML requests. As shown in the following, XML requests in the DbsEditor mode are consisted of Create Database, Add media, Remove media and etc. But XML requests in the Mbrower mode are consisted of query by media, query by id and etc.

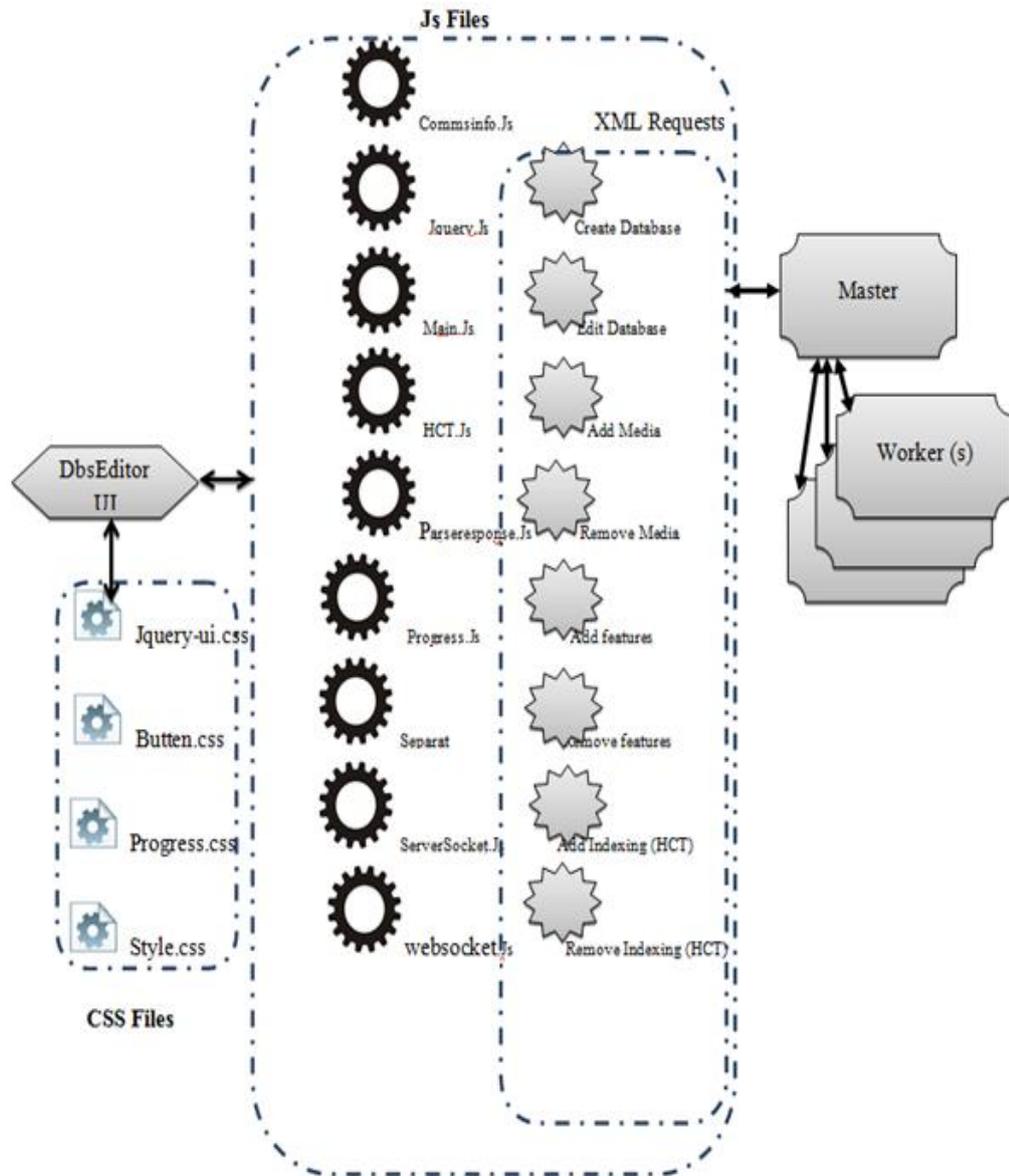


Figure 18: The DbsEditor UI Implementation Architecture

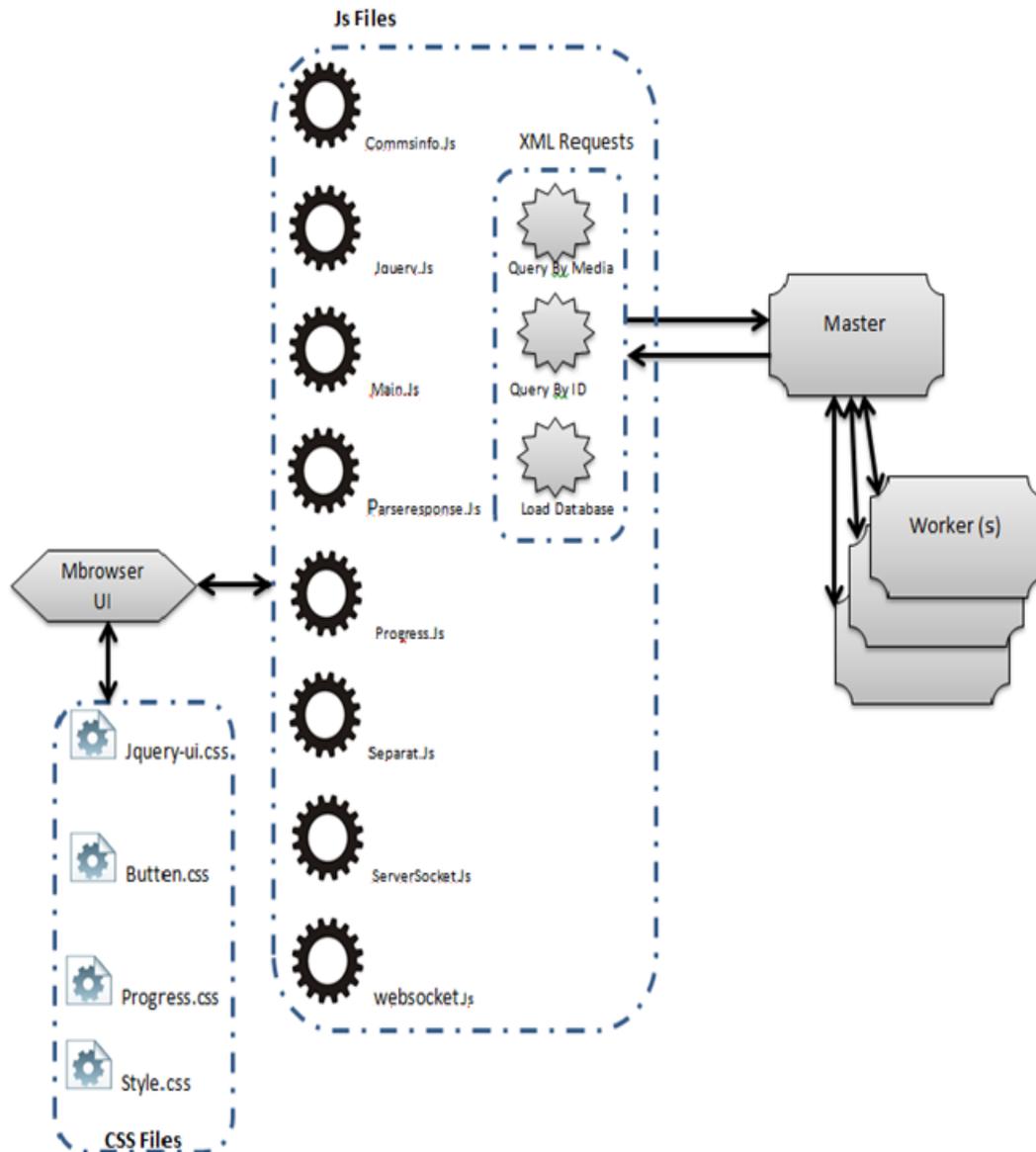


Figure 19: The MBrowser UI Implementation Architecture

An overview of the completed architecture of C-MUVIS UI connection to Master is illustrated in Figure 20. The HTML page would call the `commsinf.js`, `ServeSocket.js` and `websocket.js`. These JavaScript files can connect the UI to the socket server, handle the errors and transmit data to the server. The `commsinf.js` contains a Master IP and a Master port. Basic components of the architecture are described in Table 1.

When UI is started by the Master, firstly, the UI checks the Master is running and then the UI interacted with the Master. The connection between UI and Master is running until the user closes the UI or the Master has closed.

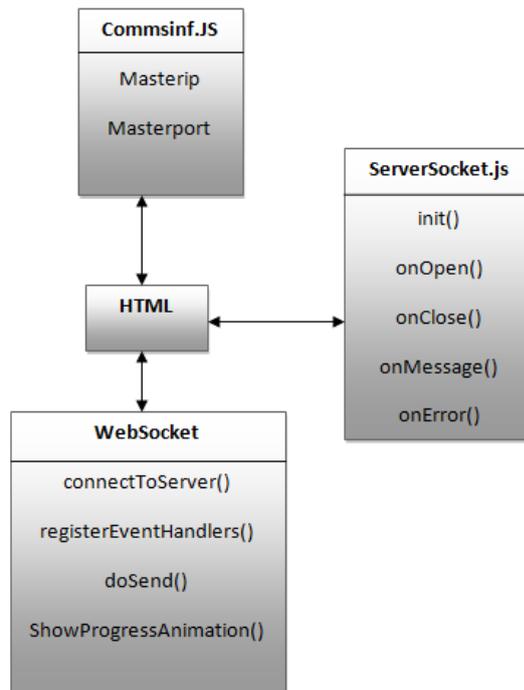


Figure 20: Architecture of C-MUVIS UI Connection to Master.

Table1: Run Events: View Implementation Architecture Components.

Component	Description
init	A function that connects the UI to the Master through a socket server with a Master IP and a Master port.
onOpen	A function that checks the connection between the UI and the Master. If they have a connection it will show an alert that displays 'Connected'.
onClose	A function that frequently checks the connection between the UI and the Master. If they disconnect, it will show an alert that displays 'Disconnected'.
onMessage	This event occurs when client receives response from server, so it stops the progress bar and activates the parse response function.
onError	If any errors occur, this event will appear a displaying the 'Error' message.

connectToServer	A function that creates a new Web-Socket object.
doSend	This event occurs when client transmits data to server using an open connection. It activates progress bar and then sends the data.

5.4 DbsEditor

The aforementioned DbsEditor mode is used for creating, editing and indexing databases. The databases utilized in DbsEditor are Image Databases contain only images and associated features.

The extracting feature is one of the functionalities in the DbsEditor. It will be explained in sub-section 5.4.5 (Extracting Visual Feature). Features are extracted within a database, and before appending and removing any visual features a database should be loaded.

Appending and removing images into the database are other functionalities in the DbsEditor that will be explained in parts 5.4.3 and 5.4.4, respectively. HCT is an indexing operation which can be performed on the databases containing at least one feature. It will be explained in part 5.4.7.

5.4.1 Create New Database

The first step after running the DbsEditor should be the creation process of a new C-MUVIS database or loading process of a previously created database. A C-MUVIS database can be created by selecting the 'Create database' menu item from the File menu. When selecting the aforesaid item, dialog window will pop up, as shown in Figure 21, which are in an overlaid position and are protected from page content. They have a title bar and a content area, and can be moved and resized.

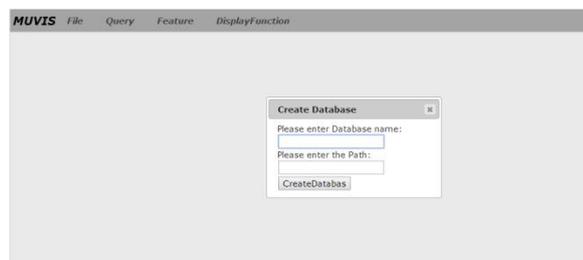


Figure 21: Create a New Database

In the process of creating new databases, the user should fill in a name box and a path box in the content area of the Create Database pop up. The name of a database file is formatted such as: (name given by user). (Four characters specifying the dbs. type). The full path of a database will be typed by the user, because for security reasons browsers have no access to the file system.

When pushing the create database button if the boxes are empty, the UI will administer an error message to request that the boxes are filled. In any other case, the UI will create an XML file such as (Program 4).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<task>
<taskid>2</taskid>
<tasktype>CreateDatabase</tasktype>
<taskdetails>
<dbtitle>MyDatabase</dbtitle>
<dbpath>W:/ Database /</dbpath>
</taskdetails>
</task>
```

Program 4: User's request for creating a database

The xml file will be sent to the server, and then the UI will ask for authority to create a database from the server-side with authentication of the database's title and path. The task ID and task type are constant for every task in XML. With this request, the UI will be waiting for the response of the server then the progress bar will commence and block all activities in the UI until the server gives the response. The response from the server will be similar to Program 5.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <response taskid="2" taskType="CreateDatabase">
3 <message>CreateDatabase Completed:</message>
4 <results>
5 <result>
6 <stat>Done</stat>
7 </result>
8 </results>
9 <taskdetails>
10 <dbconfig>
11 <dbfile> W:/ Database / MyDatabase. idbs</dbfile>
12 <dbroot> W:/ Database /</dbroot>
13 <dbtitle> My Database </dbtitle>
14 </dbconfig>
15 </taskdetails>
16 </response>
```

Program 5: Response of the server for a database creation

The response will be parsed in JavaScript and ‘Done’ will be shown in the state tag to confirm that creation of the database is successful. The display function will switch according to the task ID.

5.4.2 Load Created Database

The DbsEditor can also edit a previously created database. When creating a database, the server will create a dbs. file and an XML file in the selected database’s path. Drag and drop of the XML file to the DbsEditor is the first act for loading the database. With a drag and drop event, the UI will send the below XML file (Program 6) to server-side and the progress bar will start.

```

1
2  <?xml version="1.0" encoding="utf-8"?>
3  <task>
4    <taskid>1</taskid>
5    <tasktype>LoadDataBase</tasktype>
6    <taskdetails>
7  </taskdetails>
8  </task>
9  <dbconfig>
10 <dbfile> W:/ Database / MyDatabase.idbs</dbfile>
11 <dbroot> W:/ Database /</dbroot>
12 <dbtitle> My Database </dbtitle>
13 </dbconfig>

```

Program 6: User’s request for loading a database

The response of the server-side will be handled as is explained in sub-section 5.4.1 with different result tags. There are photo-list and photo tags within the result tags, with which photos are indexed with ID and defined with a path.

5.4.3 Appending Images into Database

External images will be appended to any C-MUVIS database by completing drag and drop of the media to the DbsEditor’s container. With this event an ‘Append Image’ dialog window will open, and an XML file will be made according to the media data. For example, the image will be sent according to base64 coded data. The XML will look like the XML in sub-section 5.4.2 with further tags for the name of media, content, different task ID and task type.

When sending an XML file the progress bar will be active and when receiving a response from the server the progress bar will completed its process. The response will allow the UI to show the media according to a full path that server has portioned and saved, as reported in Program 7.

```

1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <response taskId="4" taskType="AppendMediaItems">
3  <message>Append Media Completed</message>
4  <results>
5  <result>
6  <photolist>
7  <photo id="My Database_0_0">W:/Database/My DatabaseRoot/My Database/My Database0/Images/My
8  Database0_0.jpg</photo>
9  <photo id="My Database_0_1">W:/Database/My DatabaseRoot/My Database/My Database0/Images/My
10 Database0_1.jpg</photo>
11 </photolist>
12 </result>
13 </results>
14 </response>

```

Program7: Response of the server for appending media

According to the response every image will be nominated with a unique id.

5.4.4 Removing Images from Database

To remove image(s) from the database, first they should be selected from the container in the DbsEditor by selecting the images when the database is open. Then CSS will change the view of the selected image with opacity. After one or more image files are selected from the database, the Remove Images button in the Feature menu can be selected.

The progress bar will activate and an XML file such as previously described will be created with different tags. Removing requests consists of ID instead of content of media that is indexed by the server.

The response of the server will consist of a list ID of the images that the user has selected. This response will give the authority to the UI to remove images from screen.

5.4.5 Extracting Visual Feature

When the Append Feature button is pressed from the Feature menu, an XML file will be created according to the requested list of visual features which are existent in the C-MUVIS framework. The requested XML is as explained in sub-section 5.4.2 with a different task (number 35) and task type (GetDbaseAllFeature).

As explained previously, with a request, the progress bar will be active until a response is received from the server. The response will consist of name and parameters of all visual features in the C-MUVIS framework, as reported in Program 8, and be shown in the Append Feature windows Figure 22.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <response taskid="35" taskType="GetAllFeatureList">
3    <message>GetAllFeatureList Completed:</message>
4    <results><result><muvisfeatures>
5      <indexing>HCT</indexing>
6      <feareurelist>
7        <feature>
8          <name>BW</name><featuretype>visF</featuretype><noParam>3</noPara
9          <noSubFeatures>1</noSubFeatures>
10         <subfeature>
11           <param>1000</param><param>512</param><param>256</param>
12         </subfeature>
13         <action>ADD</action>
14       </feature>
15       .
16       .
17       .
18     <feature>
19       <name>YUV</name>
20       <featuretype>visF</featuretype>
21       <noParam>3</noParam>
22       <noSubFeatures>1</noSubFeatures>
23     <subfeature>
24       <param>8</param><param>4</param><param>4</param>
25     </subfeature>
26     <action>ADD</action>
27   </feature>
28 </feareurelist>
29 </muvisfeatures>
30 </result>
31 </results>
32 <taskdetails/>
33 </response>

```

Program 8: Response of Server for All Feature List

The visual feature will be set according to the name and parameters with one radio button for selection. When clicking on the radio button, the visual feature will be selected and hence will appear on the transit part that allows the possibility to change the parameters by typing in the boxes.

There are two buttons with the ADD and Submit value. The ADD button should be pressed after changing the parameters to create a list in the below part of visual feature windows, and the Submit button should be pressed to send the selected visual feature to server for appending.

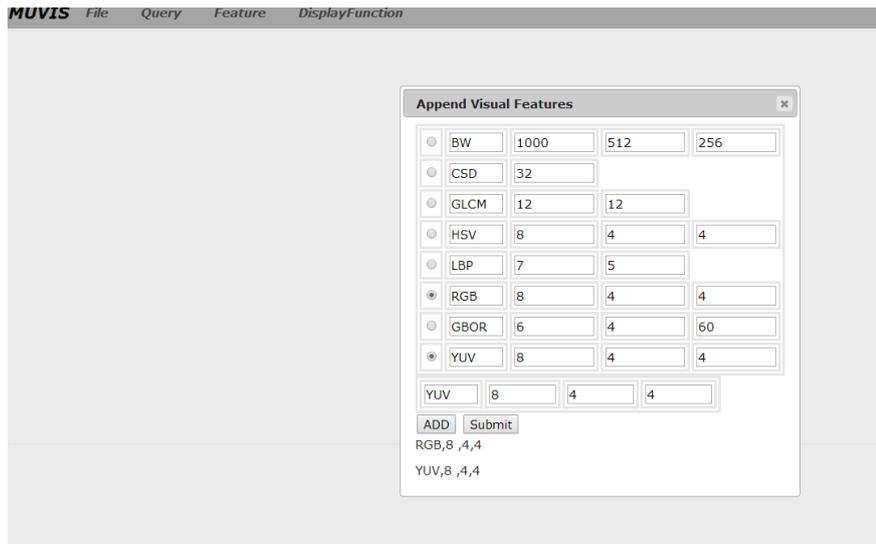


Figure 22: Append Visual Features

After one or more visual features are selected, (here we selected RGB and YUV) by pressing the Submit button, an XML file will be created as shown in Program 9.

```

1  <?xmlversion="1.0"encoding="ISO-8859-1"?>
2  <task>
3    <taskid>6</taskid><tasktype>AppendVisualFeature</tasktype>
4    <taskdetails>
5      <muvisfeatures>
6        <indexing>NONE</indexing>
7        <featurelist>
8          <feature>
9            <name>RGB</name><featuretype>visF</featuretype><noParam>3</noParam>
10           <noSubFeatures>1</noSubFeatures>
11          <subfeature>
12            <param>8 </param><param>4</param><param>4</param>
13          </subfeature>
14          <action>ADD</action>
15        </feature>
16        <feature>
17          <name>YUV</name><featuretype>visF</featuretype><noParam>3</noParam>
18          <noSubFeatures>1</noSubFeatures>
19          <subfeature>
20            <param>8 </param><param>4</param><param>4</param>
21          </subfeature>
22          <action>ADD</action>
23        </feature>
24      </featurelist>
25    </muvisfeatures>
26  </taskdetails>
27 </task>
28 <dbconfig>
29   <dbfile> W:/ Database / MyDatabase.idbs</dbfile>
30   <dbroot> W:/ Database /</dbroot>
31   <dbtitle> My Database </dbtitle>
32 </dbconfig>

```

Program 9: User's Request for Appending Visual Features

After receiving the response from the server, a message will show that the appending of the selected visual feature (here RGB and YUV) is complete.

5.4.6 Removing Visual Feature

To remove a feature from a database, first, the Remove Feature button in the Feature menu should be selected. With this selection an XML file will be created and sent to the server and the progress bar will activate. The XML file asks the server for the list of all the visual features that exist in the current database as shown in Program 10.

```

1  <?xmlversion="1.0"encoding="ISO-8859-1"?>
2  <task>
3    <taskid>34</taskid>
4    <tasktype>GetDbsVisualFeatures</tasktype>
5    <taskdetails></taskdetails>
6  </task>
7  <dbconfig>
8    <dbfile> W:/ Database / MyDatabase.i<u>dbs</u></dbfile>
9    <dbroot> W:/ Database /</dbroot>
10   <dbtitle> My Database </dbtitle>
11  </dbconfig>

```

Program 10: User's Request for All Visual Features List

The response, as reported in Program 11, will give the name and parameter of the all visual features that exist in the current database (here RGB and YUV as we extend in sub-section 5.4.4).

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <response taskid="34" taskType="GetDbsVisualFeatures">
3    <message>GetDbsVisualFeatures Completed:</message>
4    <results>
5      <result>
6        <muvisfeatures>
7          <featurelist>
8            <feature>
9              <name>RGB</name>
10             <featuretype>VisF</featuretype>
11             <noParam>3</noParam>
12             <noSubFeatures>1</noSubFeatures>
13           <subfeature>
14             <param>8.000000</param>
15             <param>4.000000</param>
16             <param>4.000000</param>
17           </subfeature>
18         </feature>
19         .
20         .
21         .
22       </featurelist>
23     </muvisfeatures>
24   </result>
25 </results>
26 <taskdetails/>
27 </response>

```

Program 11: Response of Server for All Visual Features

After receiving the response progress bar will deactivate and a Remove Feature window will open as is explained in sub-section 5.4.5 (Extracting Visual Feature). When clicking on the radio button, the feature will be selected and by pressing the Remove button a list will be created from the selected features for removal. Program 12 shows that after pressing the submit button the list will be sent to the server for removing the selected features (here RGB are selected).

```

1  <?xmlversion="1.0"encoding="ISO-8859-1"?>
2  <task>
3    <taskid>7</taskid>
4    <tasktype>DeleteVisualFeature</tasktype>
5    <taskdetails>
6      <muvisfeatures>
7        <indexing>NONE</indexing>
8        <featurelist>undefined<feature>
9          <name>RGB</name>
10         <featuretype>visF</featuretype>
11         <noParam>3</noParam>
12         <noSubFeatures>1</noSubFeatures>
13       <subfeature>
14         <param>8.000000 </param>
15         <param>4.000000</param>
16         <param>4.000000</param>
17       </subfeature>
18       <action>remove</action>
19     </feature>
20   </featurelist>
21 </muvisfeatures>
22 </taskdetails>
23 </task>
24 <dbconfig>
25   <dbfile> W:/ Database / MyDatabase.idbs</dbfile>
26   <dbroot> W:/ Database </dbroot>
27   <dbtitle> My Database </dbtitle>
28 </dbconfig>

```

Program 12: User's Ask for Removing Visual Feature

After receiving the response from the server, a message will be shown that removal of the selected visual feature (here RGB) is finalized.

5.4.7 ADD HTC Indexing

The main indexing operation in C-MUVIS is Hierarchical Cellular Tree (HCT). HCT can be performed on the databases with at least one feature. A database will be indexed by visual or aural features; however image databases can only be indexed visually.

When selecting ADD HCT Indexing button from Feature menu, HCT Indexing window will appear. There are 5 parameters with modifiable default values on the HCT Indexing window, as illustrated in Figure 23.

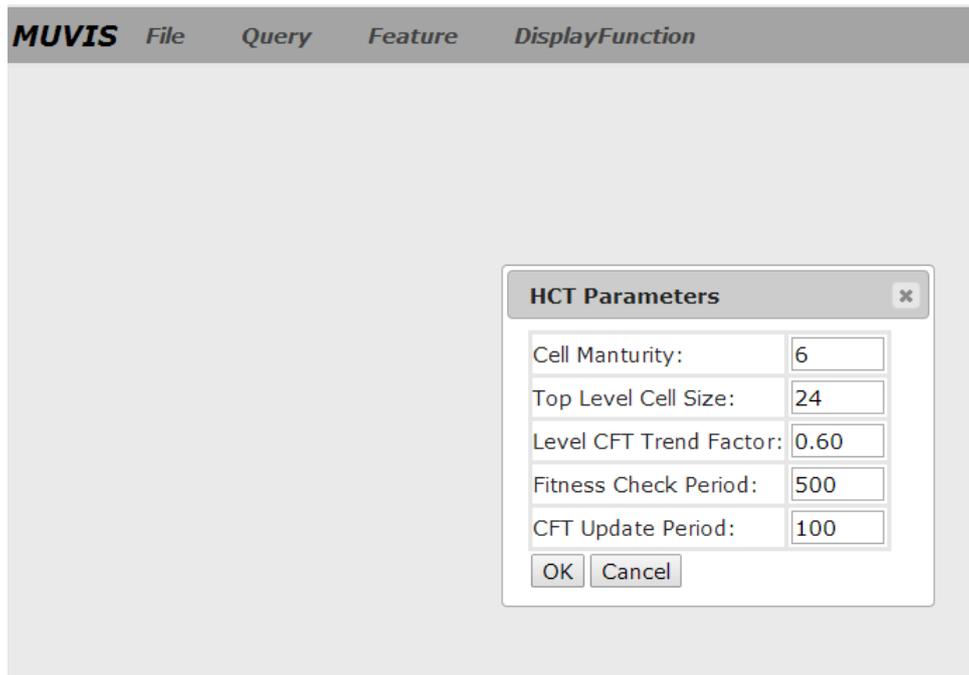


Figure 23: ADD HCT Indexing

To send a request to the server the OK button should be clicked. The requested XML is reported in Program 13.

```

1  <?xmlversion="1.0"encoding="ISO-8859-1"?>
2  <task>
3    <taskid>8</taskid>
4    <tasktype>Indexing</tasktype>
5    <taskdetails>
6      <Indexmode>visualindexing</Indexmode>
7      <CellManturity>6</CellManturity>
8      <TopLevelCellSize>24</TopLevelCellSize>
9      <LevelCFTTrendFactor>0.60</LevelCFTTrendFactor>
10     <FitnessCheckPeriod>500</FitnessCheckPeriod>
11     <CFTUpdatePeriod>100</CFTUpdatePeriod>
12   </taskdetails>
13 </task>
14 <dbconfig>
15   <dbfile>W:/ Database/ MyDatabase.idbs</dbfile>
16   <dbroot>W:/ Database/</dbroot>
17   <dbtitle>My Database</dbtitle>
18 </dbconfig>

```

Program 13: User's Request for Indexing

After receiving the server response, reported in Program 14, the 'Done' message will be shown.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <response taskid="8" taskType="Indexing">
3    <message>Indexing Completed:</message>
4    <results>
5      <result>
6        <stat>Done</stat>
7      </result>
8    </results>
9    <taskdetails>
10     <Indexmode>visualindexing</Indexmode>
11     <CellManturity>6</CellManturity>
12     <TopLevelCellSize>24</TopLevelCellSize>
13     <LevelCFTTrendFactor>0.60</LevelCFTTrendFactor>
14     <FitnessCheckPeriod>500</FitnessCheckPeriod>
15     <CFTUpdatePeriod>100</CFTUpdatePeriod>
16   </taskdetails>
17 </response>

```

Program 14: Response of the Server for Indexing

5.4.8 Remove Indexing

If a database is already indexed, the indexing structure can be totally removed by selecting Remove Indexing from Feature menu. After pressing the Remove Indexing button, the Program 15 will be sent to the server.

```

1  <?xmlversion="1.0"encoding="ISO-8859-1"?>
2  <task>
3    <taskid>33</taskid>
4    <tasktype>RemoveIndexing</tasktype>
5    <taskdetails>
6      <Indexmode>visualindexing</Indexmode>
7    </taskdetails>
8  </task>
9  <dbconfig>
10   <dbfile>W:/ Database/ MyDatabase.idbs</dbfile>
11   <dbroot>W:/ Database/</dbroot>
12   <dbtitle>My Database</dbtitle>
13 </dbconfig>

```

Program 15: User's Request for Removing Index

The server response reported in Program 16 will be to deactivate the progress bar and display the 'Done' message.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <response taskid="33" taskType="RemoveIndexing">
3    <message>RemoveIndexing Completed:</message>
4    <results>
5      <result>
6        <stat>Done</stat>
7      </result>
8    </results>
9    <taskdetails>
10     <Indexmode>visualindexing</Indexmode>
11   </taskdetails>
12 </response>

```

Program 16: Response of the Server for Removing Index

5.5 MBrowser

The MBrowser mode is designed for browsing and retrieval of images. When MBrowser mode is selected, the browser has no current action. This status of the browser is depicted in Figure 24. There is just a container for loading a database, a holder for querying an image, and a text box for pasting a URL. The user should initiate loading of a database as has been explained in part 5.4.2.

For initiating a query, (explained in parts 5.4.2 and 5.4.3) there are three methods; the first one is to drag and drop an image in the holder, the second is to paste a URL in the text box and the last is to click on an image inside the database. The result of the query will be updated on the container according to the relevance of the images in the database to the query image.

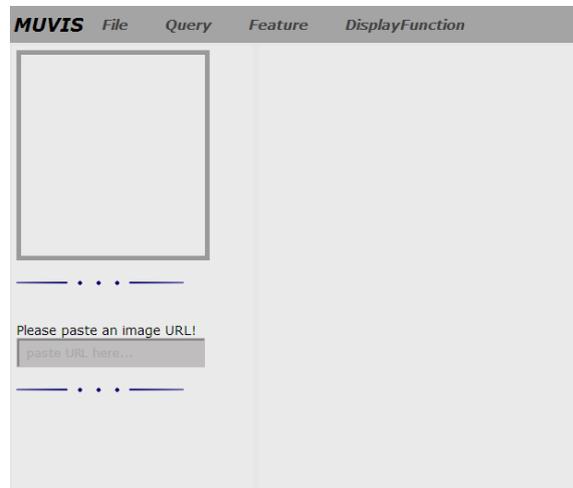


Figure 24: MBrowser with No Current Action

In the mode of Mbrowsers after using drag and drop on a previously created database, the media of the database will be loaded in the container according to its id. There are two methods for a query process; the first is to select an item by clicking on it and the second is to drag and drop an image on the holder or copy and paste a URL in the box. The first method is Query by index and the second method is query by media.

5.5.1 Get List of Database

To get a list of existing databases, first the List of Database button in the File menu should be selected. With this selection an XML file will be created, and sent to the server and the progress bar will activate. The XML file asks the server for the list of all the databases that existing. The response of the server will be consisted of the list of all existing databases.

5.5.2 Query by Indexing

In Query by indexing when a media is selected by clicking on it, an XML file will be created and sent to the server as reported in Program 17. The request is as below in which the Id tag is determines the selected media by the id. This Id is labelled in the server-side.

```

1  <?xmlversion="1.0"encoding="ISO-8859-1"?>
2  <task>
3    <taskid>17</taskid>
4    <tasktype>QueryBymediaindex</tasktype>
5    <taskdetails>
6      <images>
7        <id>My Database_2_0</id>
8      </images>
9    </taskdetails>
10 </task>
11 <dbconfig>
12   <dbfile>W:/ Database/ MyDatabase.idbs</dbfile>
13   <dbroot>W:/ Database/</dbroot>
14   <dbtitle>My Database</dbtitle>
15 </dbconfig>

```

Program 17: User's Request for Query by media Index

In the received response from the server, there are paths of the relevant media that will be parsed and shown in the UI side.

5.5.3 Query by Media

By implementing the drag and drop command to an image in the holder, a request will be sent to the server as is explained previously with basic variations in the image tag. Instead of id, the image data will be sent to the server base on base 64 (Program 18).

```

1  <?xmlversion="1.0"encoding="ISO-8859-1"?>
2  <task>
3    <taskid>16</taskid>
4    <tasktype>QueryByMediaName</tasktype>
5    <taskdetails>
6      <images>
7        <image>data:image/jpeg;base64,/9j/4AAQSkZJRgABAgEAYABgAAD/7g
8        AOQWRvYmUAZAAAAAAAB/+EU...4Xi jpp6SF1imp6qNHgj1VIgrtLCyx111NpaT6An
9        6W92mA8HwkyadMeEZJjEDmvE/t/1f5uv//Z</image>
10     </images>
11   </taskdetails>
12 </task>
13 <dbconfig>
14   <dbfile>W:/ Database/ MyDatabase.idbs</dbfile>
15   <dbroot>W:/ Database/</dbroot>
16   <dbtitle>My Database</dbtitle>
17 </dbconfig>

```

Program 18: User's Request for Query by Media Name

On the other hand, by copying and pasting a valid URL in the URL box the image will first be shown in the holder and then the XML file will be created similar to the above described.

6. CONCLUSIONS AND FUTURE WORKS

This thesis presented the requirements, design, and implementation of the adaptable C-MUVIS user interface application. C-MUVIS is a framework for content-based retrieval on web services and cloud computing. Currently MUVIS is a framework that provides global solution for content-based multimedia indexing and retrieval on desktop-based systems.

In this thesis, the RWD approach was presented and then its implications on the UI architecture were discussed. Moreover, the benefits and the drawbacks of the RWD and AWD were mentioned. In addition, C-MUVIS was presented as a framework for which a UI has been implemented. Furthermore, the functional and layout requirements for the UI were examined. Several suitable technologies that should be used in the UI implementation were introduced to the reader, such as Fluid Grid and Media Query.

In the design part, the most important point was the adaptability of the layout of the UI. The major breakpoint was chosen at width 768px and height 1024px that media query techniques supported dynamic resolution screen changes at the breakpoint. Decreasing the screen resolution from the main width (1280px) and height (1024px), to the breakpoints modified the layout of the UI smoothly. This smooth movement on the layout of the UI is supported by Fluid Grid technique.

The C-MUVIS UI was divided into two applications: DbsEditor and MBrowser. Furthermore, the layouts of both applications were also divided into several views that compose the C-MUVIS UI. The functionality of the UI was partitioned into three packages and the relevance between each package and the views were illustrated. This architecture simplifies the access to the service from any type of UI device.

Another step in the design process was to define a suitable communication method with server-side. The XML was chosen as markup language for communities between C-MUVIS UI with the Master and the Worker. To implement the UI for web browser, the three following programming languages were used in this thesis: HTML, JavaScript, and CSS.

Utilizing standardized technology has been one of the priorities in this project. Therefore the results have been coded with the base64/XML i.e. a standard reference for multimedia content management.

The presented project has settled the framework for future development of new features that will improve the C-MUVIS user experience. In the next step, one might aim to

speed up the response, by improving the efficiency of communication between UI and the Master by introducing relevant feedback techniques. Another possibility is implementation of RWD approach on the further breakpoints especially to smartphone screen resolution sizes.

The C-MUVIS was tried to use cookies for saving the information of databases such as database name, database root, and etc. on the client-side. For security reasons it was preferred to use global variable inside the JavaScript. In order to have a proper communication when client's browser makes an initial request to the Master, this request notes client's IP address/browser, stores some local session data, and sends a session ID back to the client as 'masterip' and 'masterport'. C-MUVIS client-side sends that same session ID back to the Master on future requests or communications. However, to be prepared for future developments, adding image labels to displayed image name and other information of the image or database is suggested.

REFERENCES

- [1] A. A. Mohamed, W. K. Cheruiyot, and R. Rimiru, "Responsive Web Design in Fluid Grid Concept Literature Survey," *The International Journal of Engineering and Science (IJES)*, vol. 3, pp. 49-57, 2014.
- [2] A. Berson, "Client-Server Architecture," No. IEEE-802, McGraw-Hill, 1992.
- [3] A. Gustafson, "Adaptive web design," *Crafting Rich Experiences with Progressive Enhancement*. Chattanooga, Tennessee, USA: Easy Readers, LLC, 2011.
- [4] A. L. Montgomery and M.D. Smith, "Prospects for Personalization on the Internet," *Journal of Interactive Marketing* 23, no. 2 (2009): 130-137.
- [5] A. Mac Caw, "JavaScript Web Applications," O'Reilly Media, Inc. 2011.
- [6] A. Osmani, "Journey through the JavaScript MVC Jungle," [online]. *Smashing Magazine*; July 2012. Available: <http://coding.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle>. Accessed 7 December 2012.
- [7] A. Stent, J. Dowding, J. M. Gawron, E. O. Bratt, and R. Moore, "The command talk spoken dialog system," *Proc. ACL' 99*, pages 183–190, 1999.
- [8] Apps. Sizer v3.3, "Programs by Brian Apps-Sizer," as published on Nov. 6, 2007, Available: <http://www.brianapps.net/sizer.html>.
- [9] B.FRAIN, "Responsive web design with HTML5 and CSS3," Packt Publishing Ltd, 2012.
- [10] C. Frankel, M. J. Swain, V. Athitsos, "WebSeer: An Image Search Engine for the World Wide Web," University of Chicago, Chicago, IL, 1996
- [11] C. Lowell, "Adaptive vs. Responsive Web Design: Which Is Right for Your Site?," 2014, Available: <http://blog.catchpoint.com/2014/06/02/adaptive-vs-responsive-web-design-right-site/>
- [12] D. S. McFarland, "CSS3: The Missing Manual," O'Reilly Media Inc., Sebastopol, 2009.

- [13] D. Sadoski, "Client/Server Software Architectures - An Overview," Technical report, Carnegie Mellon, Software Engineering Institute, 1997.
- [14] E. Dumbill, "Planning for big data," O'Reilly Media, Inc. 2012.
- [15] E. Harb, P. Kapellari, S. Luong, and N. Spot, "Responsive Web Design," , 2011 Available: <http://courses.iicm.tugraz.at/iaweb/surveys/ws2011/g3-survey-resp-web-design.pdf>
- [16] E. Marcotte, "Responsive Web Design," A Book a part Edition. Jeffrey Zeldman. 2011, ISBN 978-0-9844425-7-7. Available: <http://www.abookapart.com/products/responsive-web-design>.
- [17] E. Marcotte, "Responsive Web Design," a List Apart, May 25, 2010; Available: <http://www.alistapart.com/articles/responsive-web-design/>.
- [18] F. Paterno, "User Interface Design Adaptation," In: Soegaard, Mads and Dam, Rikke Friis (Eds.). "The Encyclopedia of Human-Computer Interaction, 2nd Ed." Aarhus, Denmark: The Interaction Design Foundation, 2013.
- [19] H. Heitkötter, S. Hanschke, and T.A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," In Web Information Systems and Technologies, pp. 120-138. Springer Berlin Heidelberg, 2013.
- [20] I. Ahmad, and M. Gabbouj, "A generic content-based image retrieval framework for mobile devices," Multimedia Tools and Applications 55, no. 3, pp. 423-442, 2011.
- [21] I. Ahmad, S. Abdullah, S. Kiranyaz, and M. Gabbouj. "Content-based image retrieval on mobile devices," In Proc. SPIE, vol. 5684, pp. 255-264, 2005.
- [22] J. Allsopp, "A Dao of Web Design," Retrieved May, vol. 1, 2000.
- [23] J. Bryant and J. Mike. "Responsive web design," In Pro HTML5 Performance, pp. 37-49. Apress, 2012.
- [24] J. Eisenstein, J. Vanderdonckt, and A. Puerta. "Adapting to mobile contexts with user-interface modeling," In Mobile Computing Systems and Applications, 2000 Third IEEE Workshop on., pp. 83-92. IEEE, 2000.

- [25] J. N. Robbins, "Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics," Third edition. Sebastopol, Calif.: O'Reilly, 2012.
- [26] J. R. Smith, S. F. Chang, "Visually searching the web for content," *IEEE multimedia* 4, no. 3 (1997): 12-20.
- [27] J. R. Smith, S. F. Chang. "Image and video search engine for the world wide web," In *Electronic Imaging 97*, pp. 84-95. International Society for Optics and Photonics, 1997.
- [28] J. Wang, M. Wang, X. Yang, H. Zhang, C. Zhang, and X. Zhu. "Multi-panel user interface," U.S. Patent 8,302,026, issued October 30, 2012.
- [29] J. Y. Chai, P. Hong, and M.X. Zhou, "A probabilistic approach to reference resolution in multimodal user interfaces," In *Proceedings of the 9th international conference on intelligent user interfaces*, pp. 70-77. ACM, 2004.
- [30] K. Knight, "Fixed vs. Fluid vs. Elastic Layout: What's The Right One For You?" Available: <http://www.smashingmagazine.com/2009/06/02/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/>
- [31] K. Knight, "Responsive web design: What it is and how to use it." *Smashing Magazine* 12 (2011).
- [32] K. M. Wong, K. W. Cheung, L. M. Po, "MIRROR: an interactive content based image retrieval system," In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 1541-1544. IEEE, 2005.
- [33] K. Porkaew and K. Chakrabarti, "Query refinement for multimedia similarity retrieval in MARS," In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pp. 235-238. ACM, 1999.
- [34] K. V. Natda, "Responsive Web Design," *Eduvantage* 1, no. 1, 2013.
- [35] K. Williams, "The State of Mobile Strategy and Responsive Web Design at Ohio Universities," PhD diss., Bowling Green State University, 2013.
- [36] L. Bass, P. Clements, and R. Kazman, "Software Architecture," in *Practice*, SEI Series in Software Engineering, Addison-Wesley, 1998.
- [37] L. Ying, D. Zhang, G. Lu, and W. Y. Ma, "A survey of content-based image retrieval with high-level semantics," *Pattern Recognition* 40, no. 1, pp: 262-282, 2007.

- [38] M. Clancy, R. Cremin, J. Leonard, "Implementing Your Mobile Strategy," dotMobi, 2012. Available: http://www.patternmatched.com/download/white-papers/industry-papers/Implementing_Your_Mobile_Strategy.pdf
- [39] M. Gabbouj, I. Ahmad, M. Y. Amin, S. Kiranyaz, "Content-based image retrieval for connected mobile devices," In Proceedings of the Second International Symposium on Communications, Control and Signal Processing, ISCCSP 2006, Marrakech, Morocco, 13-15 March 2006
- [40] M. Gabbouj, S. Kiranyaz, K. Aglar, B. Cramariuc, F. A. Cheikh, O. Guldogan, and E. Karaoglu, "MUVIS: A Multimedia Browsing, Indexing and Retrieval System," In Proceedings of the IWDC 2002 Conference on Advanced Methods for Multimedia Signal Processing, 2002
- [41] M. Gabbouj, S. Kiranyaz, K. Caglar, E. Guldogan, O. Guldogan, and F. A. Qureshi, "Audio-based Multimedia indexing and retrieval scheme in MUVIS framework." In Proceedings of 2003 IEEE International Symposium on Intelligent Signal Processing and Communication Systems, ISPACS 2003. 2003.
- [42] M. Ortega-Binderberger, S. Mehrotra, K. Chakrabarti, and K. Porkaew, "WebMARS: A multimedia search engine," In Proceedings of the SPIE Electronic Imaging, pp. 314-321, 1999.
- [43] M. Perkowitz, and O. Etzioni, "Adaptive web sites: an AI Challenge," In IJCAI (1), pp. 16-23, 1997.
- [44] M. Schneider-Hufschmidt, U. Malinowski, and T. Kühme, "Adaptive user interfaces: principles and practice," Elsevier Science Inc., 1993, Available: <http://www.loc.gov/catdir/enhancements/fy0601/93008442-t.html>.
- [45] M. Shaw and D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline," Prentice Hall, 1996.
- [46] P. Carey, and F. Canovatchel, "New Perspectives on JavaScript," Cengage Learning, 2005.
- [47] P. S. HIREMATH, J. PUJARI, "Content based image retrieval using color, texture and shape features," In: Advanced Computing and Communications, 2007. ADCOM 2007. International Conference on. IEEE, 2007. p. 780-784.
- [48] R. Yong, T.S. Huang, M. Ortega, and S. Mehrotra, "Relevance feedback: a power tool for interactive content-based image retrieval," Circuits and Systems for Video Technology, IEEE Transactions on 8, no. 5, pp: 644-655, 1998.

- [49] S. Kiranyaz, C. Kerem, E. Guldogan, O. Guldogan, and M. Gabbouj. "MUVIS: a content-based multimedia indexing and retrieval framework," In *Signal Processing and Its Applications, 2003. Proceedings. Seventh International Symposium on*, vol. 1, pp. 1-8. IEEE, 2003.
- [50] S. Kiranyaz, M. Gabbouj, "Content-Based Management of Multimedia Databases: Advanced Techniques for Multimedia Analysis and Retrieval," Paperback – April 27, 2012.
- [51] S. Mohorovicic, "Implementing responsive web design for enhanced web presence," In *Information & Communication Technology Electronics & Microelectronics (MIPRO), 2013 36th International Convention on*, pp. 1206-1210. IEEE, 2013.
- [52] S. P. Reiss, "Seeking the user interface," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pp. 103-114, 2014.
- [53] S. Sclaroff, L. Taycher, M. L. Cascia. "Imagerover: A content-based image browser for the World Wide Web," In *Content-Based Access of Image and Video Libraries, 1997. Proceedings. IEEE Workshop on*, pp. 2-9. IEEE, 1997.
- [54] T. Kadlec, "Implementing Responsive Design: Building sites for an anywhere, everywhere web," New Riders, 2012.
- [55] Y. Liu, D. Zhang, G. Lu, and W. Y. Ma, "A survey of content-based image retrieval with high-level semantics," *Pattern Recognition* 40, no. 1 (2007): 262-282.