
# Teaching Tool Management System: Generating A Web Front End

**Andy Berry, 0606562**

**Supervisor: Richard Cooper**

**Honours Project (CS4H)**

**March 2010**

**A dissertation submitted in part fulfilment of the requirement of**

**the Degree of BSc in Computing Science**

**at The University of Glasgow**

**Department of Computing Science**

**University of Glasgow**

**Sir Alwyn Williams Building**

**Lilybank Gardens**

**Glasgow, G12 8QQ**

# Abstract

With the growth of the Internet and its prevalence, software teaching tools are widely becoming used within teaching establishments to aid learning. These teaching tools take the form of virtual learning environments, virtual reality teaching environments and other software tools. Staff and students at the University of Glasgow have been developing one such tool to allow the teaching of software development concepts. This application allows users to create documents to represent diagrams and textual documents to describe these concepts. It also provides the ability to create processes to manipulate these documents and describe the diagram creation steps. This functionality of displaying and manipulating documents is currently only available through the newly created application and no other 'views' of the information are available. The aim of this project is to create this new 'view', more specifically to design and implement a website representation of the documents and its processes.

The creation of a new module allowed a lecturer to create a website representation of the application and mirror all viewing functionality of this application. This website can then be copied to a users 'webspace' allowing others, such as students, to view the documents created via the internet.

# Acknowledgements

I would like to acknowledge the University of Glasgow and Dr. Richard Cooper for his advice and continued support throughout this project. I would also like to thank Katie Murray and my parents for their help in proof reading this document.

# Contents

# Table of Figures

# Table of Equations



# Table of Tables

# Glossary

| | |
|---|---|
| Applet | A small program which is contained within a web page. Typically written in Java, they perform the same as a desktop application. |
| CLEV-R | Collaborative Learning Environment with Virtual Reality – a virtual learning environment under development by University College Dublin |
| CSS | Cascading Style Sheet – a file which allows styling to be applied to a website |
| ER Diagram | Entity Relationship Diagram - a method of diagrammatically modelling a database |
| HTML5 | A new revision (2010) of the HTML web language. HTML (HyperText Markup Language) is a way of representing website pages. |
| Java | Java is a programming language maintained by Sun Microsystems. |
| JavaScript | A scripting language used to perform client side functions within websites. |
| Open Source | A software philosophy where software is free to use and the source code for the software is also freely available. For more information see http://www.opensource.org/ (Open Source Initiative, 2010) |
| PHP | A server side scripting language used to dynamically create website pages. |
| UML | Unified Modelling Language – a standard way of diagrammatically describing a system or application within Software Engineering. |
| W3C | World Wide Web Consortium – the international standards organisation for the World Wide Web |
| Web 2.0 | A ‘new breed’ of web applications such as forums, blogs, wikis and collaborative environments. |
| XML | eXtensible Markup Language – a language and set of rules for encoding data within text files |
| Scripting Language | A programming language that allows the control of an application. |

# 1.0 Introduction

## 1.1 Overview

Interactive teaching tools have been in use for many years and, as the cost of computer hardware falls, this use is growing. Whilst University lecturers and other educational teachers make use of computers and projectors to provide visual aids these are very often static images, making the explanation of complex algorithms, such as the sorting of an ordered list, and software processes difficult. Tools and applications are now being developed that allows the creation of 'moving' images to help describe these complex structures.

Within the University of Glasgow such a tool is being developed to allow lecturers to illustrate software engineering concepts such as UML diagrams, XML documents and algorithms. Within software engineering processes a variety of documents are created in order to understand a given problem, typically each of these documents are inter-related with several others. When the solution comes to be implemented; very often many of these documents can be directly translated into code. An example of this is a class diagram for a system where the classes within the diagram directly translate into classes within the system code. The tool developed at the University of Glasgow aims to help users understand this inter-relation between documents and their uses by allowing users to create graphical and textual representations of these documents, such as UML diagrams, to be used within lectures. The tool, however, currently allows the use of these diagrams within lectures but does not allow them to be exported to another format.

Another use of technology which has seen a recent growth is the Internet. Many organisations now have an online presence and many popular tools now have the ability to export data to an HTML format to be displayed as a website. This functionality can be utilised to provide a teaching tool which can be used within the classroom, and also allow the data to be exported to an HTML web page which can be viewed outside of the classroom using a web browser.

The aim of this project is to develop a 'web front end' for the existing interactive teaching tool which is currently in development at the University of Glasgow. This web front end will be generated by an additional module that will process and create the files required to make a website. It is imagined the existing application will be used by lecturers to create teaching aids which can then be exported to a website for students to view in their own time.

This project follows on from the work undertaken by staff and students at the University of Glasgow. Details and functionality of the existing system appropriate to this project are outlined later.

## 1.2 Project Context

With the growth of the Internet has come the use of online resources to aid teaching within Universities and other teaching establishments. These uses have expanded to form concepts such as Virtual Learning Environments (VLEs), Virtual Reality Learning (VRL) and Learning Management Systems (LMS). The system being developed aims to provide a similar service allowing lectures to create resources which can be used in lectures. From these resources a web representation can be created which students can access outside of lectures.

The adoption of e-Learning and online learning environments is increasingly becoming an explored option within Universities and other learning establishments. Online learning environments such as Blackboard (Blackboard Inc., 2010) and Moodle (Moodle Trust, 2010) are now widely used to provide teaching materials online that are accessible outside of the normal teaching hours. With the increase in usage of these systems there are now many services, ranging from fee paying subscriptions to open source, all aiming to provide the next generation of online learning services. The advance in internet web services, such as Web 2.0 & HTML5, have aided the development in such systems and nowadays many websites make use of JavaScript and other web technologies to provide the user with an enhanced web experience.

There have been many studies performed regarding how students learn and the use of electronic resources and ‘active learning’ (Bonwell & Eison, 1991). These studies have shown that students participate in classes and learn more by actively taking part in their teaching. The use of online learning resources also facilitates this. An example is the recent web link, funded by the Connecting Classrooms Project (British Learning Council, 2010), between several schools in Devon and schools in Iraq (BBC News, 2009) where English students communicated with other foreign students and learned about different cultures around the world. A similar use of technology is the utilisation of computer games, such as Wii Fit, within the classroom (BBC News, 2010). Whilst perhaps the use of games is not seen to be as beneficial as other learning resources some schools are investigating the use of educational games to aid teaching.

### 1.2.1 E-Learning

Currently there are several projects being carried out which aim to provide learning tools online and offer a large variety of resources for students to make use of. One such project is the Virtual European School (Bouras, Fotakis, Kapoulas, Koubek, Mayer, & Rehatschek, 1999). The VES is a project funded by the Educational Multimedia Task Force Initiative within the European Union whose aim it is to develop a system which provides online resources of teaching materials for secondary school education. The VES also aims to provide

an innovative learning system to reduce teacher's reluctance to use computers and electronic resources within the classroom (Bouras, Fotakis, Kapoulas, Koubek, Mayer, & Rehatschek, 1999). In addition to this the system facilitates interaction between teachers and pupils from different areas of the world through communication tools, games and collaborative environments. Similar to the module which is to be created as part of this project, the VES system is accessed through a standard web browser and makes use of Internet technologies to deliver and display content to the user.

The CLEV-R (Collaborative Learning Environment with Virtual Reality) (Gavin McArdle, 2004) is another project which takes virtual learning a step further. This project, developed at University College Dublin, makes use of the availability of broadband access and the advances in virtual reality to "*deliver a VR learning environment, mimicking a conventional university setting"* (Gavin McArdle, 2004). Students are represented by avatars, a virtual representation of themselves, and can interact with other avatars as if they were interacting within the real world. This system allows users to 'attend' lectures in a way they would do in real life using the internet.

## 1.3 Problem Definition

While these tools can be incredibly useful and allow students to learn without attending any University building they can be very costly to develop, build and maintain. A basic interactive teaching tool is much cheaper to develop and, to a certain extent, can provide the same basic functionality as a virtual reality system. The tool developed by The University of Glasgow provides this functionality of an interactive immersive teaching tool at a fraction of the cost of a virtual reality system.

The existing system (discussed in section 2.0) allows users to design different graphical and textual document structures; for example CSS files, UML diagrams and ER (entity relationship) diagrams. It also enables users to combine several documents into an 'application' and associate a 'process' with an application which allows users to step through the creation of a document. The functionality of generating a web front end will be provided by a new module generating the required output for a web based client.

This new module, for which the requirements are to be specified and then implemented, will allow a web interface to be generated which will mirror the look of the existing application. It is perceived that the teaching tool could be used by lecturers and university staff to demonstrate the use, and creation, of graphical and textual documents. For example a lecturer may use the tool to create a UML document and a process to step through the creation of the document. The new module could then be used to create a web representation of the

document to be placed on the lecturer's website and allow students to view the process in their own time.

## 1.4 Assumed Knowledge

It is not assumed the reader has any technical knowledge regarding software practices or specific software implementation languages. It is however useful to have an elementary knowledge of software languages and web technologies. Any specific details regarding implementation languages or web technologies will be explained. It is also useful to have a fundamental understanding of simple algorithms, although not essential.

## 1.5 Existing Application

The existing application provides several key functions; the creation and display of textual and graphical documents; the creation and display of applications; and the creation and execution of processes. Within the application a document is used to represent a collection of fragments, or objects. Each fragment is a part of a diagram or textual document, such as a group of letters or text. There are two types of documents, textual and graphical, each of which is made up of many of these fragments. Each document can be contained within many applications with each application having zero or more processes associated with it. These relationships are represented in Figure 1. The functions and relationships between the parts of the existing application are discussed in section 2.0.

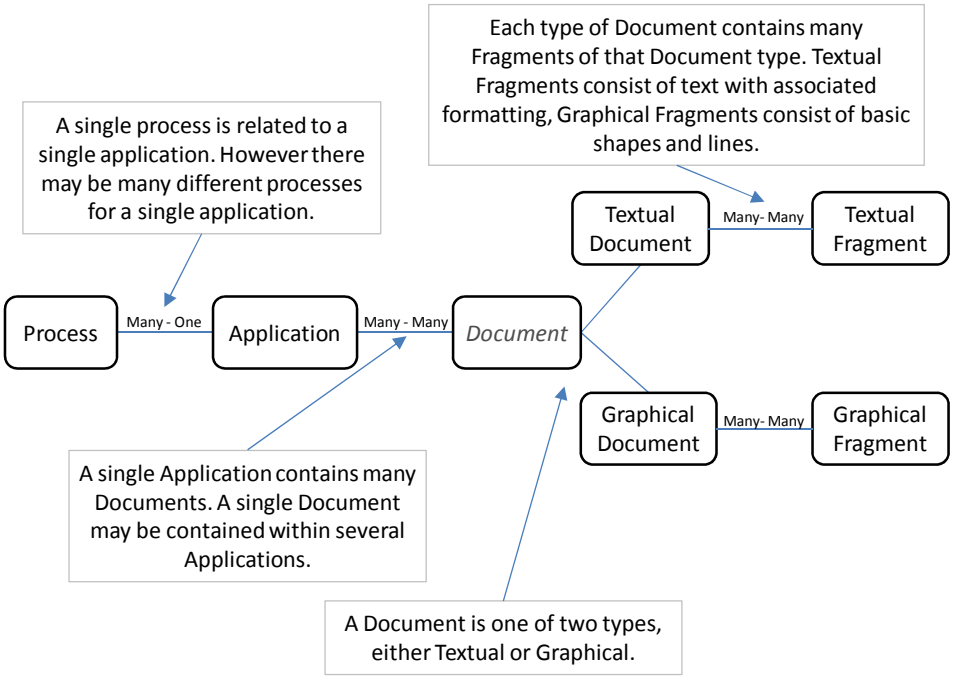


***Figure 1 - Relationship between objects within the existing application***

## 1.6 Report Outline

This document is a report on work done within the project. It outlines the structure of the existing teaching tool and the requirements for the new module which will generate a web representation of the documents. It then explores the different methods that can be used to implement the specified requirements and the design decisions made when designing and implementing the module's requirements. The implementation details are then discussed and the implemented module reviewed. Finally testing and software metric details and areas for improvement or extension are discussed.

# 2.0 Existing Application

This section outlines the existing application and the representation of different entities within the application. Any specific implementation details needed when designing the web output module are discussed where relevant.

### 2.1.1 Graphical and Textual Documents

Graphical documents are comprised of one or more graphical fragments; a fragment is simply an object which is contained within a document. Fragments within graphical documents (graphical fragments) can either be an arc or a node. Arcs are lines connecting two nodes with an optional label for the arc. This label may have some styling associated with it such as font style, font colour and size. A node can be one of several simple shapes, such as rectangles; ellipses; and triangles. Similar to an arc a node may have a label associated with it, which again may have some styling associated. The node itself can have styling applied, such as fill colour, line colour and line thickness. The example in Figure 2 shows a graphical document with 3 nodes, with the Person and Name nodes visible; and 2 arcs, Person-name and Person-age, connecting the nodes. Each of the nodes and arcs also has a label associated with it with various styling applied to the fragments.

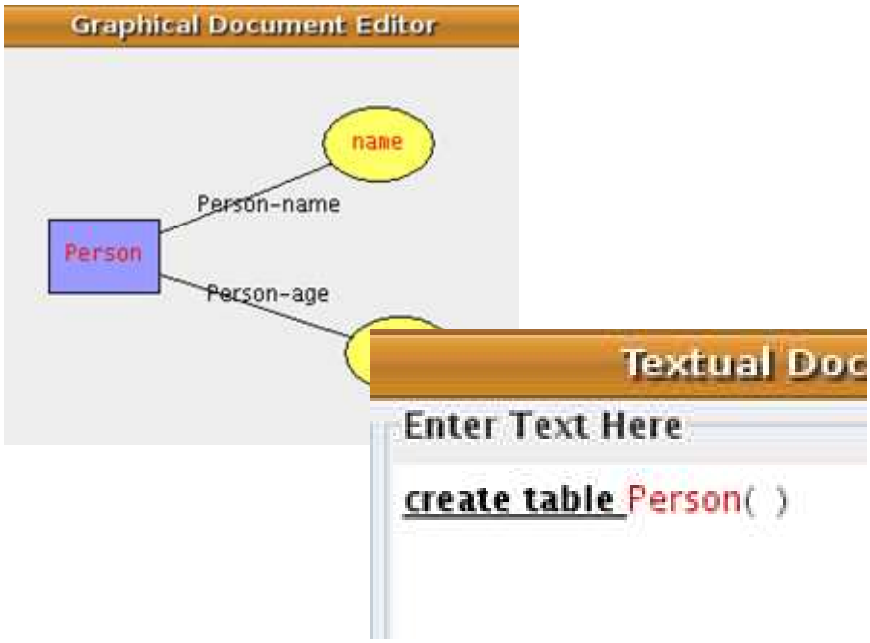


***Figure 2 – Example of graphical and textual documents***

Like graphical documents, textual documents are made up of one of more textual fragments. Within textual documents a fragment is a piece of text which has some styling. Again, like graphical fragments, this styling contains details such as font style; font colour; and font size.

It also allows underlined, bold, and italic styles to be added. Figure 2 shows a textual document with different styling applied to each fragment.

### 2.1.2 Applications

An application defines a layout of one of more documents contained within panels, chosen from a selection of available layouts (Figure 3). Once the layout has been selected the user can then select a document to be contained within each panel. These documents can be of different types and the same document may appear twice within a given application. An example of an application is shown in Figure 3. This application contains a single graphical document with two textual documents. Other than showing many documents together an application provides no other functionality above displaying a single document. The main use of an application is to display two or more documents side by side and allow a process to manipulate the documents within the application.

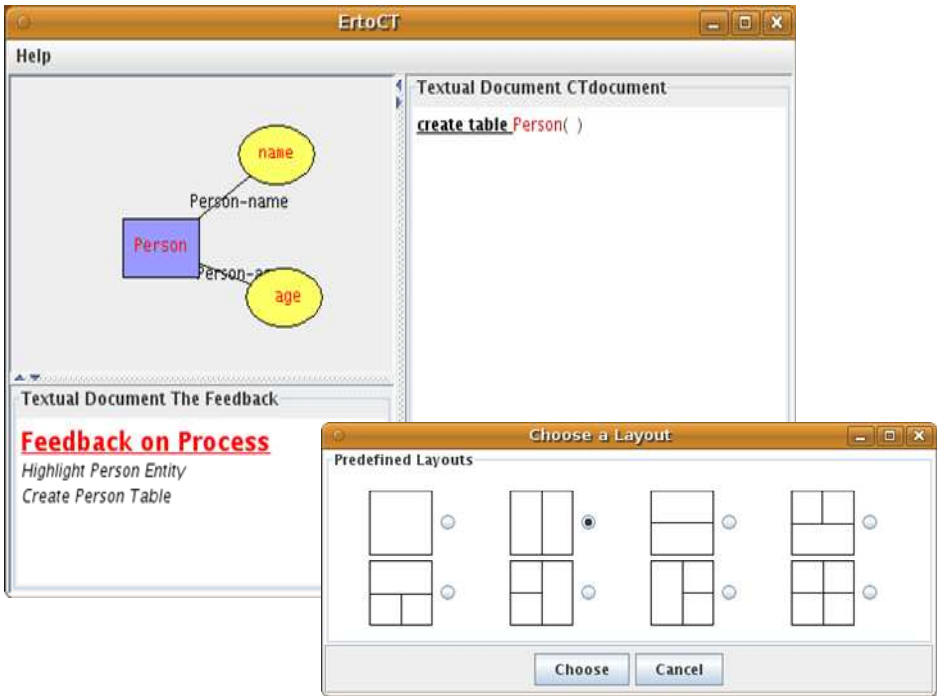


***Figure 3 - Example of an application***

### 2.1.3 Processes

A process is associated with an application and allows the manipulation of fragments within documents associated with the application. This allows a user to define a process to step through several documents describing the creation process at each step. A process is made up of one of more steps and each step is made up of one of more changes. At each step every change for that step is made, allowing complex operations to be performed. The changes available are insert; delete; move; highlight and show whole document. Figure 4 shows an

application where a highlight operation is applied to a graphical fragment and text is inserted describing the operation performed.

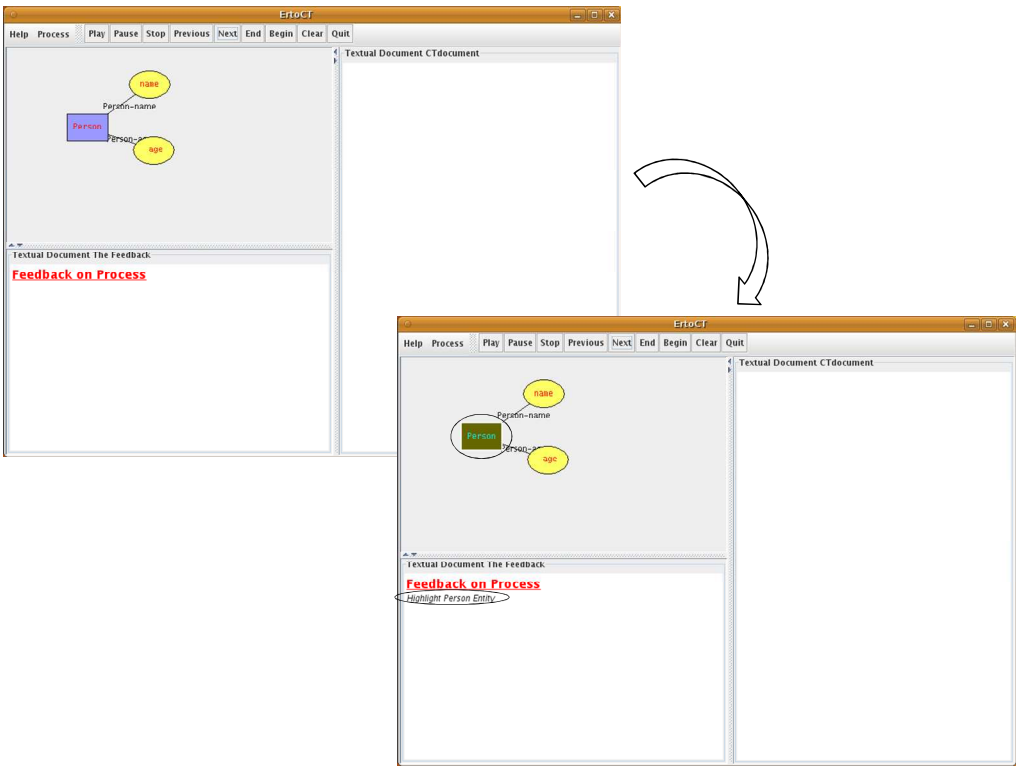


***Figure 4 - Example of a highlight operation within a process***

### 2.1.4 Data Storage

As the application's 'state' must be persistent (saved between sessions) a method of data storage is used. The data held within the application is stored using XML files. These XML allow the data to be represented in a way that many applications can understand and make use of the data. Four files are used to store the data needed by the application; Documents.xml, Doc Types.xml, Applications.xml and Processes.xml. The specific details of these documents are described in the Design & Implementation section.

# 3.0 Requirements

The requirements for the project were gathered from several meetings with the 'client', and project supervisor, Dr. Richard Cooper. During these meetings the problem was explained along with the specifications of the current application and the requirements for the application extension. A requirements specification was then created based on the users requirements. Within the requirements specification there are two sets of requirements, functional and non-functional. Functional requirements deal with functions or services the system must provide while non-functional requirements deal with any constraints on the project, such as time and performance requirements.

The system, or module, is concerned with the generation of document, application and process display and does not involve any changes to be made to the underlying functionality of the existing application. The generated output will 'mirror' the existing application.

This section is a report on work to be done within the project; it contains the requirements of the module which is to be created. The section was created as a separate document during the design process and is included here.

## 3.1 Single Statement of User Need

A single statement of user need is a useful way of expressing the requirement at the highest level. The following is an SSUN for this project. *The user requires to ability to convert the display functions of the current Java application, such as viewing documents and executing process, to HTML format in order to allow presentation over the web. A key requirement is to make use of the existing application as-is and provide the additional functionality in an extra 'module' so no changes to the existing application are required.* This is what the project needed to realise and against which success can be judged.

## 3.2 Feasibility Study

Before specific requirements were defined the existing application was reviewed in order to establish whether the implementation of the user needs was possible. This involved establishing whether the technology was available to implement the requirements, such as the use of dynamic graphic generation and delivery over the web. It also involved ensuring the requirements were realistic and could be implemented in the given time frame.

## 3.3 Functional Requirements

Each of the functional requirements is described below. Each requirement follows on from the previous in terms of the functionality it provides. As each requirement relies on the

proceeding requirements functionality they will be implemented in the order defined. An alternative ordering could be used by creating ‘dumb’ modules, or stubs. This is discussed in the Design & Implementation section.

### 3.3.1 Textual Document Representation

The user must be able to generate the relevant web page for a given textual document. The generated web page shall be similar, ideally identical, to the display given by the Java *application* from which the web page is generated. This similarity shall include both textual formatting, font style; colour; and size, and page layout.

### 3.3.2 Graphical Document Representation

The user must be able to generate the relevant web page for a given graphical document. Like web pages for textual documents this graphical web page shall be similar to the display give by the Java application. This similarity shall include shape size; colour; style and positioning.

### 3.3.3 Application Representation

The user must be able to generate the relevant web page for an application for both graphical and textual documents. This application view must be the same as the view given by the Java application and must be capable of displaying several different displays of either the same document or different documents.

### 3.3.4 Application Processes

The user should be able to generate pages to display and ‘execute’ an application process. As with previous requirements the output should be a similar to the existing application as possible. Users should be able to step forward and backward through steps. In addition to this users should be able to play through a process; while playing a process the process is automatically stepped through without any user intervention required.

### 3.3.5 Document Modification

Users may be able to interact with documents allowing them to edit existing documents and create new documents. This functionality could then be extended to allow users to save documents they have created using the web page.

*This requirement is an additional requirement that may be implemented if time allows.*

### 3.3.6 Highlight Similar Objects

The user may be able to select a document object and that object along with any associated objects will be highlighted. This will allow users to, for example, select a graphical node

representing a table within an entity relationship diagram and be shown the corresponding SQL command to create that table.

*This requirement is an additional requirement that may be implemented if time allows.*

## 3.4 Non Functional

1. The code for generating the web output must be in Java in order to provide an add-on module for the existing teaching tool application. This add-on module must not require any functionally changes to be made to the existing *application*.
2. All output files making up the web representation must conform, conform as much as possible, to XHTML and CSS specifications outlined by W3C (World Wide Web Consortium, 2010).
3. Graphical output should be in a format that conforms to W3C standards (World Wide Web Consortium, 2010).
4. Output generation must be completed within a 1 minute period. Output time, from the user selecting the generate option to completion, should ideally be completed within 10-20 seconds.
5. The user should not require any technical knowledge regarding web technologies and their uses.
6. The specified functionality must be contained within an additional module that can be used with the existing 'Teaching Tools Project' application.
7. The generated text and 'code' must be in a legible and human readable format.
8. The generated web pages should be compatible with the latest version of the two most popular web browsers, Firefox 3.5 and Internet Explorer 8. This may also be extended to other popular browsers such as Opera, Google Chrome and Safari.
9. Additional files that may be required for data storage must be in an XML data format (World Wide Web Consortium, 2010).
10. The project must be fully tested and integrated before Friday 2$^{6th}$ March 2010, the deadline for Honours projects.

## 3.5 Requirements Validation

Once defined, the requirements were validated to ensure they were both realistic and could be implemented within the time constraints. The requirements were then put forward to the client, Dr. Richard Cooper, to ensure they met client requirements.

# 4.0 Use Cases

The following section takes each requirement in turn and outlines the process in which the requirements will be satisfied. These processes are represented as Use Cases with each Use Case containing details such as pre-conditions, success and failure scenarios and the actors associated with the use case.

Pre-conditions must be satisfied in order to the Use Case to succeed. Scenarios are generated to ensure possible outcomes are indentified and the steps performed during these scenarios are outlined. Actors, 'people' or systems, which interact with the Use Case, are also identified.

The Use Case diagram created from the Use Cases is shown in Figure 5. The diagram shows two actors, Lecturer and Student, each interacting with a Use Case as described below.

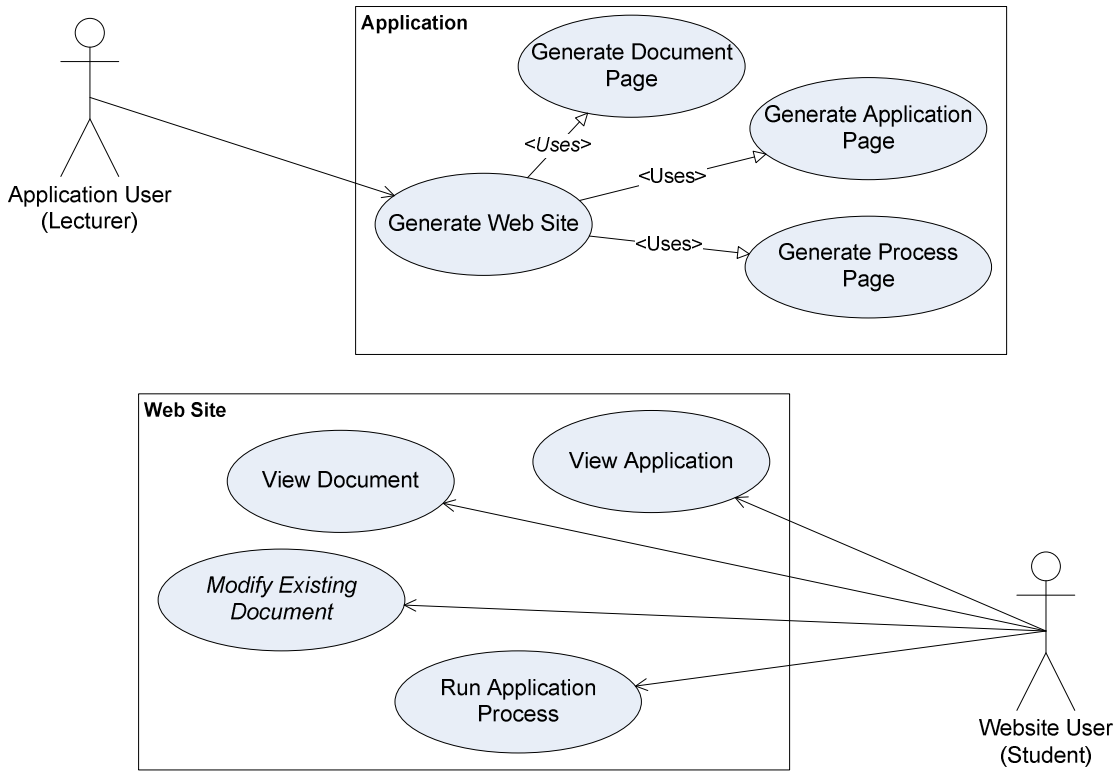


***Figure 5 - Use Case diagram for the new functionality***

## 4.1 Generate Web Site

### 4.1.1 Rationale

Figure 1 shows the main Use Case for the generation of the web output. This Use Case is divided into three sub Use Cases in order to better identify the requirements and their details. These Use Cases are discussed later.

### 4.1.2 Details

| | |
|---|---|
| Actors: | Application User (Lecturer) |
| Preconditions: | The user is using the Java *application*.<br>Documents already exist for which to create a website. |
| Post-conditions: | The website is created.<br>The user is required to copy the generated files to their own 'webspace' in order to view the website. |
| Extension Points: | None |
| Include Use Case: | Generate Document Page<br>Generate Application Page<br>Generate Process Page |

### 4.1.3 Flow of Events – User Interaction

1. The user selects the option to generate the web output.
2. The user is then prompted for a path for which to save the files.
3. A confirmation is shown to the user confirming the files have been generated.

### 4.1.4 Flow of Events – System Actions

1. The system records the path entered by the user.
2. The required files are then generated and written to the path selected by the user.

## 4.2 Generate Document Page

### 4.2.1 Rationale

The user can generate a page for a selected document, either graphical or textual. The user does not directly interact with this Use Case, this Use Case is used through the generate web site Use Case.

### 4.2.2 Details

| | |
|---|---|
| Actors: | Application User (Lecturer) – through generate web site Use Case |
| Preconditions: | A path for which to output the files has been selected |
| Post-conditions: | The files required to display a document are generated |
| Extension Points: | None |
| Include Use Case: | None |

### 4.2.3 Flow of Events – System Actions

1. The generate document page function is invoked.

2. The system then generates the required files to display a document.

### 4.2.4 Scenarios

***Success***

Web output is generated and the user is notified of success.

***Generation Failure***

User must be notified that the generation process has failed. This failure scenario will only be brought to action in the event of an error writing to the path specified by the user. This could be caused by incorrect file permissions or an invalid path.

### 4.2.5 Other Requirements

A textual document must have been previously created by the Teaching Tools *application*.

## 4.3 Generate Application Page

### 4.3.1 Rationale

The user can generate a page for a selected application. The user does not directly interact with this Use Case, this Use Case is used through the generate web site Use Case.

### 4.3.2 Details

| | |
|---|---|
| Actors: | Application User (Lecturer) – through generate web site Use Case |
| Preconditions: | A path for which to output the files has been selected |
| Post-conditions: | The files required to display an application are generated |
| Extension Points: | None |
| Include Use Case: | None |

### 4.3.3 Flow of Events – System Actions

1. The generate application page function is invoked.
2. The system then generates the required files to display the application.

### 4.3.4 Scenarios

***Success***

Web output is generated and the user is notified of success.

***Generation Failure***

User must be notified that the generation process has failed. This failure scenario will only be brought to action in the event of an error writing to the path specified by the user. This could be caused by incorrect file permissions or an invalid path.

### 4.3.5 Other Requirements

An application must have been previously created by the Teaching Tools *application*.

## 4.4 Generate Process Page

### 4.4.1 Rationale

The user can generate a page for a selected application process. The user does not directly interact with this Use Case, this Use Case is used through the generate web site Use Case.

### 4.4.2 Details

| | |
|---|---|
| Actors: | Application User (Lecturer) – through generate web site Use Case |
| Preconditions: | A path for which to output the files has been selected |
| Post-conditions: | The files required to display a document are generated |
| Extension Points: | None |
| Include Use Case: | None |

### 4.4.3 Flow of Events – System Actions

1. The generate process page function is invoked.
2. The system then generates the required files to display a document.

### 4.4.4 Scenarios

***Success***

Web output is generated and the user is notified of success.

***Generation Failure***

User must be notified that the generation process has failed. This failure scenario will only be brought to action in the event of an error writing to the path specified by the user. This could be caused by incorrect file permissions or an invalid path.

### 4.4.5 Other Requirements

An application process must have been previously created by the Teaching Tools application.

# 5.0 Design & Implementation

After reviewing the requirements, it was apparent that an Extreme Programming (Wells, 2009) approach to design and implementation was perhaps a more appropriate method than the traditional Rational Unified Process (IBM, 2009) or other alternative frameworks. Extreme Programming is an iterative process where a single requirement, or a very small group of requirements, are analysed; designed; implemented and tested before the process is repeated again with another set of requirements. This provides several small releases within a project rather than a large integration and deployment process producing a final release towards the end of the project.

The decision to use an Extreme Programming was made by reviewing the requirements and identifying that each of the functional requirements requires one, or more, of the preceding requirements to be implemented. The requirement to view an application, for example, can only be completed once the functionality of viewing an individual document is available.

An alternative approach to the project could have been to implement the largest, or most complex, part of the solution first. This would result in the most complex part being fully implemented and tested without any issues before the end of the project. This could be facilitated by creating stubs, 'dumb' parts of the implementation which simulate the functions and the output of that part. The complex functions could then be implemented and the dumb functions used. These dumb functions could then be fully implemented in a future release.

The following section follows the process used within the design and implementation stage. The design, implementation and testing details are discussed for each requirement in turn. Below is a brief outline of the order in which the requirements were implemented.

1) Textual Document Representation – page 31
2) Graphical Document Representation – page 33
3) Application Representation – page 34
4) Application Processes – page 36
5) Highlight Similar Objects – page 37

Due to the time constraints imposed on the project the final two requirements, Dynamic Editing of Documents and Highlighting Similar Elements, could not be implemented before the deadline for projects. However these requirements could be implemented in a following release.

## 5.1 Implementation Overview

As the current tool is written in Java it was originally thought that the simplest approach would be to alter the existing tool to allow it to be run as an applet, a small web program. This applet would be embedded within a web page and, apart from needing very minor changes to ensure it would be compatible running as an applet, would be a very simple modification, This approach was very quickly discounted as an applet would require web users to install additional software, process additional information and require knowledge of the tool itself in order to view the documents. For this reason, the method chosen was to create a website using the same structure as contained within the tool.

The first choice to make when designing the website solution was whether to generate static HTML pages within the Java module or to create PHP scripts which would dynamically display pages based on different arguments passed to the script. Each of these methods has its own advantages and disadvantages which are discussed on the following pages.

### 5.1.1 The Advantages of Static HTML Pages

Static pages are typically either HTML or XHTML files which once created simply present the same output regardless or when they are viewed, the state of the system or any arguments passed to the web server. These static pages are relatively fast to load compared to dynamic pages, as they don't have to execute prior to sending, however as mentioned they only display a single output which cannot be changed without changing the file itself.

Using static pages to display the output for the existing Teaching Tool System would require the new module to calculate the pages needed to make up the website and then in turn generate each of these pages. Typical files required by the website would be a Home Page; a file representing styles used by the website, perhaps one for every document used; any files required to support the representation of processes and the files to display each documents and each application. Using the above estimations the following calculation (Equation 1) demonstrates the number of files required to produce a website for 5 documents, 3 applications and 2 processes. More importantly it also estimates the time required to generate this website.

*Home Page + Style Sheet + 5 * (Document Page + Style Sheet) + 3 * (Application Page + Style Sheet) + 2 * (Process Functions + Style Sheet) = 22 files*

***Equation 1 – Calculation of static number of website files required***

Each of these files must be created by finding the appropriate data held by the existing system and, if the system holds a large amount of saved documents and applications, it could take several seconds to create each file. If small files were to take 5 seconds to generate and larger

files up to 15 seconds an overall time for generation could be in the region of 2.5 to 4 minutes. Although this is a small amount of time users may not be prepared to wait this length of time to generate, what seems to be, a very small and simplistic web representation.

### 5.1.2 Server Side and Client Side Scripting

Server side scripting, such as PHP or dynamic pages, is the term used to describe programs running on a web server. This program is inaccessible by any clients and executes, typically creating an output in the form of a web page. A well used example of this is a Google search, where a 'program' runs in order to find the results and an output is generated showing the results to a user. Client side scripting is usually used to enhance functionality on the client, or web browser. Again this program, and its data, is separate to the web server and the two cannot interchange information directly. A typical example of client side scripting is JavaScript.

### 5.1.3 The Advantages of Dynamic PHP Pages

PHP files are small programs, or scripts, which execute on a web server to produce a webpage or similar output. They typically interact with a database or similar data store, such as XML files, to create pages whose content can change based on the data received and any parameters passed to the script. An example of this is a typical Google search. The URL http://www.google.com/search?q=Computing Science relates to a dynamic page *search* which displays its output based on data stored within Google's servers and the argument *Computing Science* which is the search term.

It can be seen from the example that a single PHP file can generate a large variety of output based on a data source, a simple Google search can return a variety of results that differ on a daily, or even hourly, basis. This will allow for a simplified file structure and shorter website generation time. The disadvantage of dynamic pages is the time taken to send a page from a web server to a client is slightly increased as the script first has to be executed. In most circumstances this change is unnoticeable, especially if a very connection to a data source is used and the script is optimised for speed such as using faster searching algorithms for example. In addition to this many web servers are built for the purpose of executing and delivering dynamic pages and will therefore decrease the time taken to execute scripts.

In addition to the advantages already mentioned using dynamic files allows for greater flexibility when extending the functionality of the output. If another requirement was added this new requirement could easily be added by simply creating a function within the PHP scripts to parse the data used by the existing system and display the required output.

### 5.1.4 Approach Chosen

It was decided that due to the flexibility for current & new requirements allowed and the simplification of the website provided, all files would be dynamic and would change their output based on arguments passed to the file. This included both pages for documents and applications, and style sheets for documents.

Making use of PHP and the XML data files creates a multiple tiered architecture (Figure 6), which allows for greater flexibility within the website.

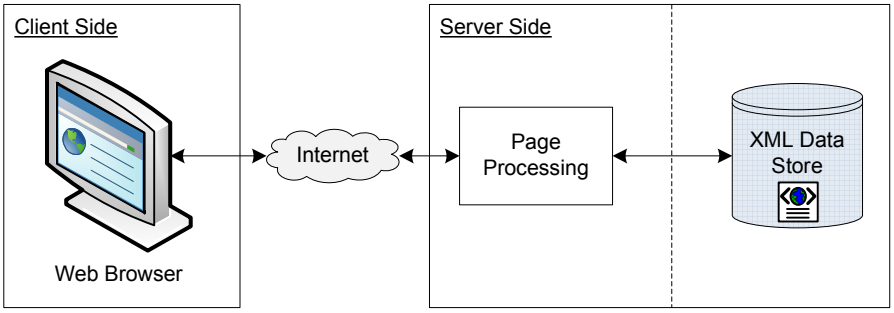


***Figure 6 - Three tiered architecture***

### 5.1.5 Data Storage

It was decided that as the existing application made use of XML to store persistent data the same data files would be used to provide data to the PHP scripts. The current functions to create the XML files would be used to create the data files during the generation process.

Once it was decided these XML files would be used sample data files were created and reviewed to gain an understanding of the structure of these files. Four files are used to store the required data by both the application and the website which is to be created, the data held within these files and their structure is outlined below.

#### *Applications.xml*

The applications file contains any application details which are saved by the user. It contains data such as the name of an application, the layout of the particular application and the documents contained within each panel. Each application element contains the element name along with the layout of that application. It also contains nested elements which give details of the panel number and the contents of that panel. An example of an Applications.xml file is shown below.

```
<applications>
   <application name="Application Name" layout="5">
      <panels>
         <panel number ="0" type ="1" content = "Panel 0"/>
         <panel number ="1" type ="2" content = "Panel 1"/>
         <panel number ="2" type ="2" content = "Panel 2"/>
      </panels>
   </application>
</applications>
```

***Figure 7 – Example Applicatons.xml file***

### *Doc Types.xml*

Any styling details associated with any documents are contained within the Doc Types.xml file. It contains zero or more documentType elements with each documentType element containing the name of the document type and the kind, either Graphical or Textual. Each document type then contains several fragmentType elements which again give the name of the fragment type and the kind. Graphical documents may contain graphical arc or graphical node elements and textual documents may only contain atomic text fragments. Contained within the fragmentType elements are the styling details for each arc, node or text fragment. These styling details contain information such as the fragments foreground and background colour and the text styling and size. An example of such as file is shown below.

```
<doctypes>
   <documentType name = "EntAtt" kind = "Graphical">
      <fragmentType name ="EtoA" kind = "Graphical Arc">
         <from>Entity</from>  <to>Attribute</to>
            <arcstyle style ="Single Line" width ="1" lineColor ="-16777216"
labelColor ="-65536" />
      </fragmentType>
      <fragmentType name ="Entity" kind = "Graphical Node">
         <awayfrom>EtoA</awayfrom>      <towards>EtoA</towards>
         <nodestyle shape ="Rectangle" height ="40" width ="60" borderColor ="-
16777216" fillColor ="-6710785" labelColor ="-65536" />
      </fragmentType>
   </documentType>
   <documentType name = "Feedback" kind = "Textual">
      <fragmentType name ="Heading" kind = "Atomic Text">
         <textstyle font = "Arial Black" size = "18" bold ="true" italic = "false"
underline = "true" background = "-1" foreground = "-65536"/>
      </fragmentType>
   </documentType>
</doctypes>
```

***Figure 8 – Example Doc Types.xml file***

### *Documents.xml*

This file contains details associated with a particular document. It details the fragments contained within the document along with data such as their location and, for graphical fragments, any other fragments they are connected with. It also specifies the style associate with that fragment. Graphical documents contain several node elements which contain details such as the type of fragment, the location, and the text contained within the node. They also contain arc elements which, similar to node elements, contain details associated such as the type and the value of the label. Arc elements also contain the two nodes which the arc

connects in the form and to attributes. Textual documents contain text elements which, similar to graphical node elements, contain the text of the fragment along with their type. An example file is shown below.

```
<documents>
   <document name = "PersonEA" type = "EntAtt" kind = "Graphical">
      <node label ="Person" type ="Entity" X ="118" Y ="116" />
      <node label ="name" type ="Attribute" X ="209" Y ="53" />
      <node label ="age" type ="Attribute" X ="215" Y ="150" />
      <arc label ="Person-name" type ="EtoA" from ="Person" to ="name" />
      <arc label ="Person-age" type ="EtoA" from ="Person" to ="age" />
   </document>
   <document name = "The Feedback" type = "Feedback" kind = "Textual">
      <text type ="Heading" value = "Feedback on Process;" />
      <text type ="Instruction" value = "Highlight Person Entity;" />
      <text type ="Instruction" value = "Create Person Table;" />
   </document>
</documents>
```

***Figure 9 – Example Documents.xml file***

### *Processes.xml*

The Processes.xml file contains details of any processes associated with an application. It contains several *process* elements each with a name and the application they are associated with. It also contains a series of *step* elements, each of which contains several *change* elements. An example file is shown below.

```
<processes>
   <process name ="Transform" app = "ErtoCT">
      <description></description>
         <step num = "1">
            <change num="1">
               <docname>PersonEA</docname>
               <operation>5</operation>
            </change>
            <change num="2">
               <docname>The Feedback</docname>
               <fragname>Feedback on Process;</fragname>
               <operation>1</operation>
            </change>
         </step>
         <step num = "2">
            <change num="1">
               <docname>The Feedback</docname>
               <fragname>Highlight Person Entity;</fragname>
               <operation>1</operation>
            </change>
            <change num="2">
               <docname>PersonEA</docname>
               <fragname>Person</fragname>
               <operation>4</operation>
            </change>
         </step>
         <step num = "3">
         <change num="1">
            <docname>CTdocument</docname>
            <fragname>create table </fragname>
            <operation>1</operation>
         </change>
         <change num="2">
            <docname>CTdocument</docname>
            <fragname>Person</fragname>
            <operation>1</operation>
         </change>
         <change num="3">
            <docname>CTdocument</docname>
            <operation>5</operation>
         </change>
      </step>
   </process>
</processes>
```

***Figure 10 – Example Processes.xml file***

## 5.2 Textual Document Representation

### 5.2.1 Requirement Overview

The user can generate a page for a selected document, either graphical or textual

### 5.2.2 Design

In order to generate the required output for a textual document the Documents XML file needed to be parsed. The structure of this file is described in the previous section. In order to understand the steps required to process the document the sequence diagram in Figure 1 was created. This sequence diagram identifies each step in turn and allows the steps to be seen visually, this was then directly be translated into PHP code to produce the required functionality. During this design stage it was decided that the HTML  element would be used for textual documents in order to apply the styling to a particular document, the creation of this styling is discussed later. The class for a document, which defines the styling

applied, would be generated based on the document type the fragment belongs to along with the type of this fragment. A function to perform this functionality would be created in order to allow for flexibility should the method of class creation change.

### 5.2.3 Implementation

When generating the output certain factors relating to the format of the input data and the way in which HTML is displayed needed to be thought about. One of these factors was the use of the newline character. Within the XML data file the semi-colon character was used to represent a newline, while within HTML the string <*br/*> is used to represent a line break. These semi-colon characters needed to be converted into an HTML break in order to display a similar representation to the Java *application.* Another issue was the use of spaces within the XML data. Within HTML * * represents a space character and this too needed to be catered for.

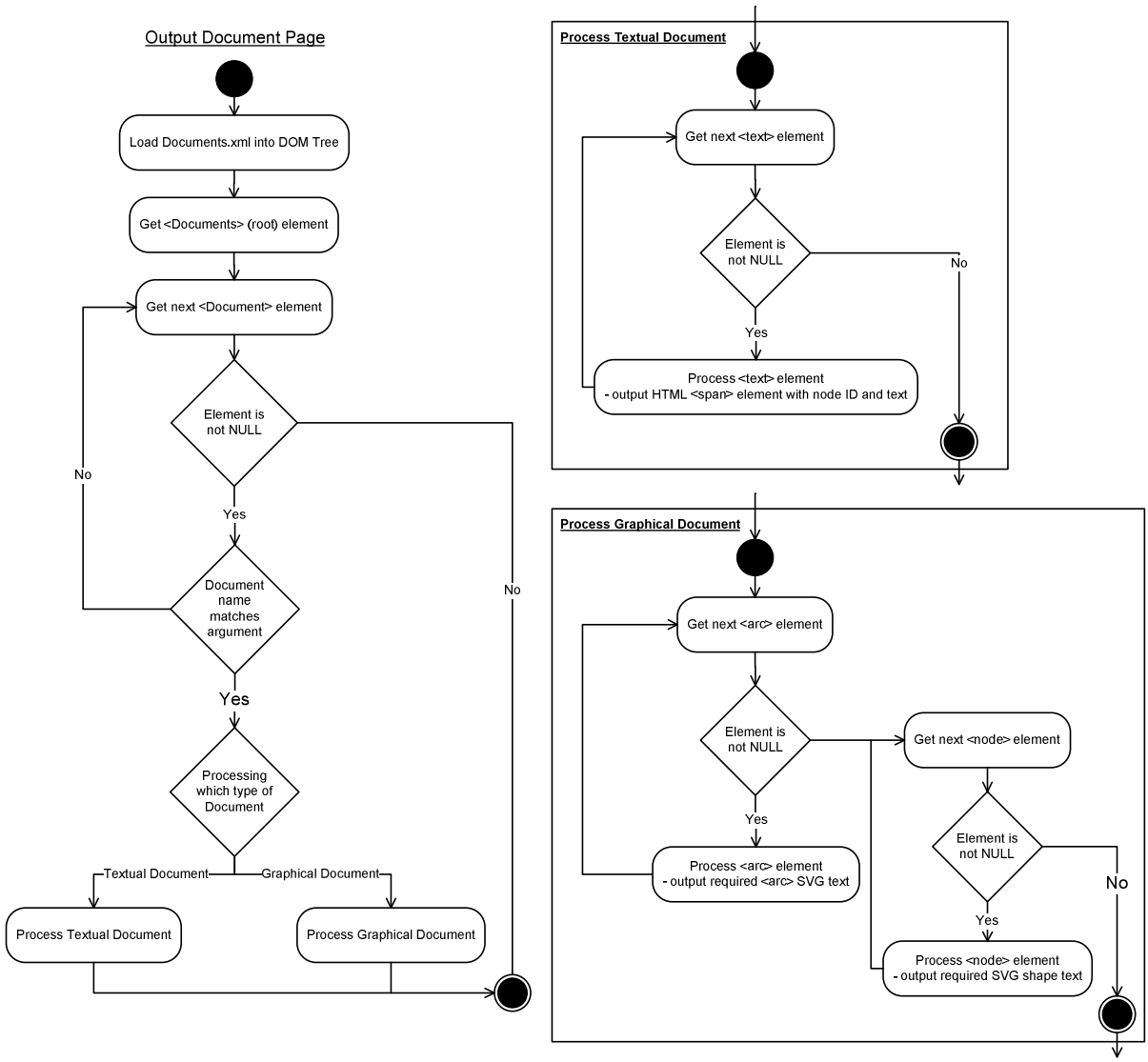


***Figure 11 – Sequence diagrams for Textual and Graphical Documents***

## 5.3 Graphical Document Representation

### 5.3.1 Requirement Overview

The user can generate a page for a selected document, either graphical or textual

### 5.3.2 Design

The first step when determining how graphical documents should be generated was to identify a technology which could be used to describe such a graphical document. One option considered was the use of PHP and the GD library (Joye, 2007) to generate dynamic images based on data from the XML files. Another option was the use of SVG, a language developed by the W3C (World Wide Web Consortium, 2010) to describe two dimensional graphics using XML. It was decided that due to the simplicity and flexibility it provided SVG would be used to produce a graphical document.

The generation of graphical documents was approached in a similar manner to the generation of textual documents. As with the textual documents a sequence diagram (Figure 11) was used to identify the steps required to creating a document. Each of the arcs and nodes in turn must then be processed and the relevant SVG text printed.

### 5.3.3 Implementation

During the output generation the arcs must be printed first to they are painted ‘underneath’ the nodes. If the nodes are printed first, followed by the arcs connecting these nodes, the lines would be visible where they meet the node. However the location of the nodes must first be known before the relevant arcs can be drawn connecting the locations of the nodes. To solve this problem the nodes are first processed and the generated output stored. During this generation a list of node’s locations is created and is then used when drawing the arcs. The output from processing arcs is then printed before printing the stored output from processing nodes.

## 5.4 Document Styling

In order to provide W3C compliance and allow for greatest flexibility it was decided that a style sheet would be used to provide styling for the website. This resulted in the PHP and generated HTML containing no styling details at all and styling is applied by associating a style sheet with the website.

The style sheet for the website would need to be a part static and part dynamic file. It would need to be static to provide styles which would not change regardless of which type of document is displayed, such as heading and table styles. The dynamic section would change

dependant on the documents being displayed and would be generated each time the file was loaded.

The dynamic styling is generated in a similar way to document generation. The Doc Types.xml file is loaded and then traversed, processing each documentType in turn. At each documentType the relevant style is output creating a list of styles for each document.

## 5.5 Application Representation

### 5.5.1 Requirement Overview

The user can generate a page for a selected application.

### 5.5.2 Design

As applications are simply a collection of documents this requirement requires a simple function to output a table with each cell representing a panel containing the required document. The Applications.xml file is traversed to find the application which is being processed. Once the application has been found, each document contained within the panels must then be processed, following this the HTML structure in which to layout the panels must then be output with the panels output in the corresponding layout. The sequence diagram for the generate application page is shown in Figure 12.

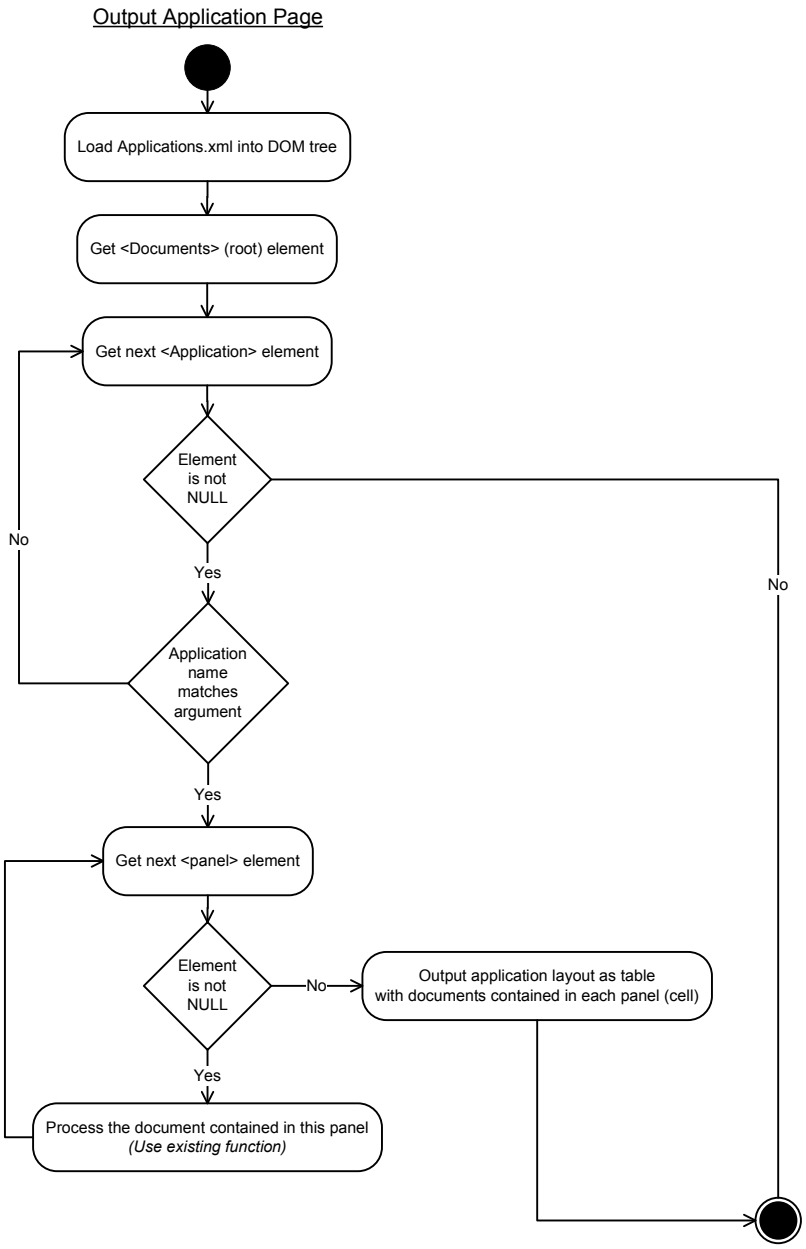


***Figure 12 - Sequence diagram for Applications***

### 5.5.3 Implementation

As this requirement is an extension of the document page the implementation is straightforward. The function created to implement this requirement makes use of the existing generate document functions. It was decided that a table would be used to represent the layout of an application as it best resembled the possible layouts of panels for applications. In order to simplify the structure of the code the function first processes each document held within the application and stores the result. It then calculates the required structure for the panels inserting each output in the relevant place within the table structure.

## 5.6 Application Processes

### 5.6.1 Requirement Overview

The user can generate a page for a selected application process.

### 5.6.2 Design

As a process is associated with an application, the function created to process an application can be used with additional processing implemented to manipulate the fragments within the documents. In order to provide W3C compliance (World Wide Web Consortium, 2010) and provide as much ‘cross browser compatibility’ as possible, JavaScript would be used to implement the fragment manipulation. It was decided this additional JavaScript would be provided in a separate file to that of the application display. This separation would allow for easier flexibility and modularity.

As the JavaScript files would be dynamic and change their output dependant on the XML data associated with the website, a solution was needed to pass the data from ‘server side’ to ‘client side’. As PHP is a server side scripting language, any data structures created within a PHP script are not accessible from a client. JavaScript on the other hand is a client side language and any JavaScript code is executed in the client browser. In order to pass process data between the server and client JavaScript code would be generated creating data structures on the client side to hold the process data. This JavaScript would then be executed on the client machine and create the required data structures.

### 5.6.3 Implementation

The requirement was implemented as discussed previously, creating data structures on the client machine to hold the steps for each process, and the changes within these steps. An example of this data structure is shown in Figure 14 and the code used to create this structure in Figure 13.

The code shown defines a new array (a table with one column) which itself contains an array of step objects. This array within an array has the effect of creating a multi-column table, see Figure 14. Each object defines the document name and the document type that the step is associated with. It also defines the fragment name which is to be manipulated and the type of operation to perform.

```
steps[1] = new Array();
    steps[1][1] = new Object();
        steps[1][1].docName = 'PersonEA';
        steps[1][1].docType = 'Graphical';
        steps[1][1].fragName = '';
        steps[1][1].operation = '5';
    steps[1][2] = new Object();
        steps[1][2].docName = 'The Feedback';
        steps[1][2].docType = 'Textual';
        steps[1][2].fragName = 'Feedback on Process;';
        steps[1][2].operation = '1';
steps[2] = new Array();
    steps[2][1] = new Object();
        steps[2][1].docName = 'The Feedback';
        steps[2][1].docType = 'Textual';
        steps[2][1].fragName = 'Highlight Person Entity;';
        steps[2][1].operation = '1';
```

***Figure 13 - Example process data structure code***

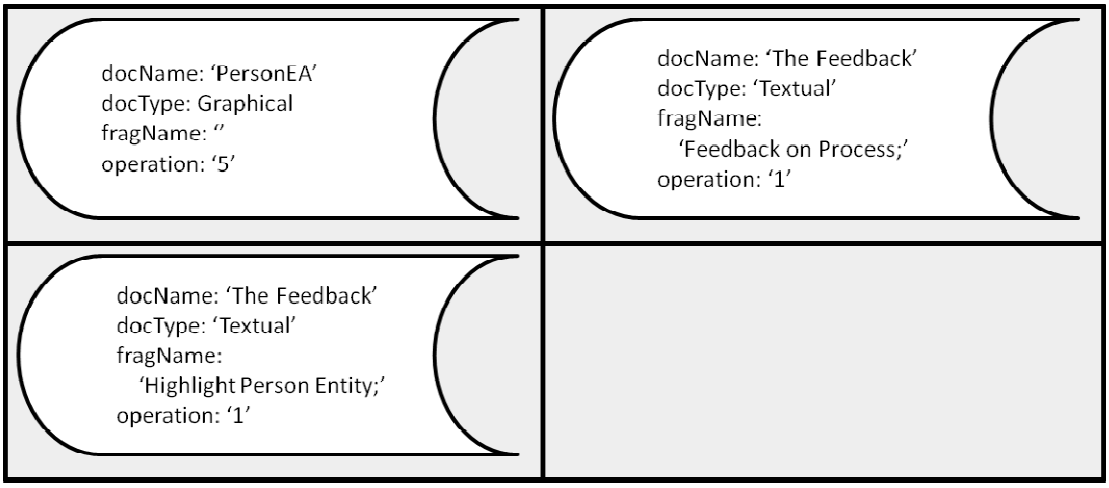


***Figure 14 - Example process data structure diagram***

## 5.7 Highlight Similar Objects

### 5.7.1 Requirement Overview

Users should be able to select a document fragment and the system should then identify related fragments by highlighting those related fragments.

### 5.7.2 Design

Within the existing version of the Java application there is currently no functionality to allow the highlighting of similar objects. This requirement can therefore be designed 'from the ground up'. In order to provide this functionality there must be a method of identifying which fragments are related to one another within the data held in the XML files.

A proposed solution to this is to create a new 'child' element for node, arc and text elements within the Documents.xml file to identify associated fragments within other documents. This element would be called associatedFragment and would contain the two attributes fragment and document. The attributes would identify the related fragment and the document it was

contained in respectively. The existing elements would contain zero or more associatedFragment elements to identify several associated fragments. Using this new data held within the documents.xml file the website, more precisely the JavaScript, would be able to calculate the ID of the associated elements and highlight them.

Due to the nature of the design the changes made to the XML file would not impact on the existing application. This requirement would however require changes to be made to the existing application to be fully implemented.

### 5.7.3 Implementation

Due to the complexity of implementing processes using JavaScript and the time constraints imposed on the project, this requirement could not be implemented within the timescale for honours projects. However due to the iterative nature of the release cycle this could be implemented in a future release.

# 6.0 Testing & Evaluation

This section outlines the testing carried out during the development of the new web output module. It also discusses the usability and acceptance tests carried out once the implementation was complete. Finally the project as a whole is evaluated and any complications or problems identified.

## 6.1 Testing Functionality

As the Extreme Programming development life cycle was used testing was carried out during each implementation phase. It was also carried out after each phase was complete, during the integration. A final testing phase was then carried out once all implementation was finished. This testing involved ensuring the output of the website was identical to that of the application. Any functionality regarding the viewing of applications and processes must also be represented within the website.

### 6.1.1 Display of Documents

Each document type was tested to ensure it was represented in the same way as those documents displayed within the Java application. For graphical documents, documents were used that tested both the display of graphical arcs and graphical nodes. These graphical fragments were tested both with no styling applied and different types of styling, such as font colour and font decorations. Textual documents were tested with several types of fragments and, as with graphical documents, each fragment was tested with several types of styling.

During this testing it was found that the documents displayed within the web pages satisfactorily met the display of the documents within the Java application.

### 6.1.2 Display of Applications

Applications were tested by creating several applications, each with several documents contained in different panel layouts. Each panel layout was then tested to ensure it was identical to the layout of the application. As this functionality used the functions provided by the previous requirement, the testing process was very similar.

### 6.1.3 Execution of Processes

In order to test processes each different operation needed to be tested for a variety of different applications and documents, again comparing the functionality and display of the website to that of the Java application. Several applications where created each with three or four documents. Associated with each of these applications were several processes, each of which contained many steps in order to test every operation.

## 6.2 Metric Tests

### 6.2.1 Page Load Times

Two values of load time were measured in order to test the performance. The page generation time for a graphical document and a process were tested by measuring the page load time on a local web server. As the pages would be loaded from a local server there would be no delay for sending data over the internet or a network, and therefore measure generation time only. Secondly the page load time was measured on a remote server in order to measure the total time taken to download the website and view a page. It was important to measure both values as a page may load very quickly on a local server however, if it has many large files which need to be transferred, loading the site from a remote server may take some time.

The command line program *wget* was used to download the files as this basic application would show the time to receive the file only and not the time to display the page. The page requisites argument *(–p)* was used so that all additional files would also be downloaded, fully simulating a page access from a web browser. A script, a small program, was created to repeatedly run the command and calculate an average time. The script used and an example output are shown below.

```
#!/bin/bash
execLimit=5
sum=0
function avgLoadTime() {
   site=$1
   for i in $(eval echo {0..$execLimit})
   do
      folder="wget.tmp."$i
      output=`(time -p wget -q -p -O $folder $site) 2>&1 | grep real | cut -d " " -f
2`
      sum=`echo "scale=2; ${sum}+${output}" | bc`
   done
   avg=`echo "scale=4; ${sum}/${execLimit}" | bc`
   echo "Average time for $site: $avg"
   rm -rf wget.tmp.* 2>&1
}
avgLoadTime "http://localhost/phpTest/viewProcess.php?processName=Transform"
avgLoadTime "http://localhost/phpTest/viewProcess.php.html"
avgLoadTime
"http://80.176.138.28/TeachingToolsProject/viewProcess.php?processName=Transform"
avgLoadTime http://80.176.138.28/TeachingToolsProject/viewProcess.php.html
```

***Figure 15 - Script used to calculate page load times***

```
Average time for http://localhost/phpTest: .7520
Average time for http://80.176.138.28/TeachingToolsProject: 1.8780
```

***Figure 16 - Example output from page load script***

It can be seen from the example output above that, although the page load time for a remote server (the second output) is slightly greater than the local server, the overall load time is not unreasonable for a small website. This small generation time allows for more complex pages,

if necessary for future requirements, to be created without having a negative impact on the site usability.

Once the overall load time of the site was calculated the different between static page load time and dynamic page load time was calculated, using the same script as used before. These static pages were created by copying the output from the dynamic pages and creating a temporary static page. For example the page viewProcess.html was created by copying the output from the page viewProcess.php?processName=Transform. This was done to ensure the same page was being loaded so the two values could be compared. The results obtained are shown in Figure 17.

```
Average time for http://localhost/phpTest/viewProcess.php?processName=Transform: .8920
Average time for http://localhost/phpTest/viewProcess.php.html: 1.5280
Average time for
http://80.176.138.28/TeachingToolsProject/viewProcess.php?processName=Transform:
3.5820
Average time for http://80.176.138.28/TeachingToolsProject/viewProcess.php.html:
5.2640
```

***Figure 17 – Difference in page load times for static and dynamic pages***

It was expected that the HTML pages would be loaded slightly faster than the dynamic PHP pages, however, the results from this test were not as expected. After investigating the cause of this difference the only apparent explanation was that the headers (the additional data 'telling' the browser what is being sent and how it is to be received) which are generated by the web server were slightly larger for the HTML pages compared to those for the PHP pages. However this was an unlikely cause as these headers form a very small part of the data sent.

In order to fully understand the reason for this difference the test was carried out a second time using a web browser diagnostic tool to monitor a 'real' browser's behaviour. The tool used was the Firefox extension FireBug[1], which can graphically display how pages are being downloaded and the time taken. This output can be seen below.

[1] FireBug is an extension for Firefox which 'allows the debugging, editing, and monitoring of any website's CSS, HTML, DOM, and JavaScript, and provides other Web development tools' (http://en.wikipedia.org/wiki/Firebug_%28web_development%29). It is available for download from https://addons.mozilla.org/en-US/firefox/addon/1843 (accessed 24 March 2010).

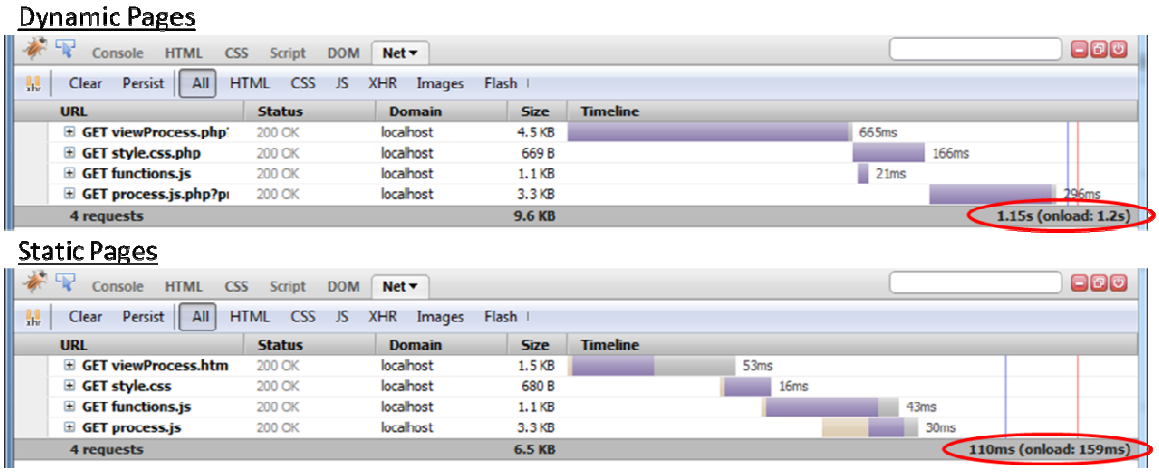


***Figure 18 –Difference in page load times using FireBug***

After reviewing the results from using FireBug it became apparent that in reality a web browser, such as Firefox, downloads parts of a website simultaneously where the command line tool created downloaded the parts of the site synchronously. This difference may explain the different in results. Another factor affecting the performance of the PHP pages is the server used was optimised for executing and delivering PHP pages. This may explain the possible speed comparison between PHP and HTML pages.

### 6.2.2 JavaScript Execution

JavaScript was tested on several clients to ensure that it was both compatible with the client and browser, and to ensure the performance was acceptable. This test was done by calculating the current time at the start of an operation, and recalculating the time after the operation had finished. The difference of these two values could be used to calculate the total running time for the operation. This value was calculated for stepping forward through a process calculating the time for each step, the total time to make all changes for that step. The same process was used to calculate the time for stepping backwards through a process. The test was repeated on several different clients; a high performance laptop, a medium performance virtual machine, and a low performance virtual machine. ‘Virtual machines’ are computers that are created on another physical computer, they act completely independently of the ‘host’ computer and for all intents and purposes are separate computers. Virtual machines were used so the performance of the machine could be modified in order to provide different specifications of client performance.

| | Time to Step Forward (ms) | | | Time to Step Backward (ms) | | |
|---|---|---|---|---|---|---|
| | *Step 1* | *Step 2* | *Step 3* | *Step 1* | *Step 2* | *Step 3* |
| High Performance Windows 7 Laptop | 9 | 5 | 4 | 2 | 2 | 2 |
| Mid Performance XP Virtual Machine | 14 | 5 | 6 | 5 | 3 | 3 |
| Low Performance Linux Virtual Machine | 36 | 7 | 7 | 3 | 5 | 5 |

***Table 1 – JavaScript performance comparison***

The table generated from the results obtained is shown above. The results show that there is a slight increase in the time taken for the operations as the performance of the client decreases. This time, however, in each circumstance is very small and greatly within any performance requirements for a website.

## 6.3 Evaluation

Within each test performed the website passed without any significant issues. Although the times taken to load from a remote server were in the region of 3 or 4 seconds, a high performance website such as Google (www.google.co.uk) took between 1.5 and 3 seconds using the same method of testing. This suggests the load times for the website are very reasonable and this allows room for additions to be made without impacting on performance. The JavaScript for processes also performed very well without taking longer than half a second for any operation. The time taken to perform an operation, such as step forward, may increase if many changes are made within each step. As the time taken for a small to medium sized operation is very low there is the flexibility for the time taken to increase slightly.

# 7.0 Outcome

The following section provides an overview of the solution to the problem defined in the Requirements section. It also identifies any possible improvements to the website and future developments.

## 7.1 The Implemented Solution

### 7.1.1 Viewing Documents

Figure 19 illustrates the similarity between the viewing of documents using the Java application and the website generated from the application. The representation of textual documents is shown in the top two windows contained in the figure, the left window is the display from the Java application, and the right window is from the website. This illustrates that the representation of textual documents in the website is identical to that of the Java application.

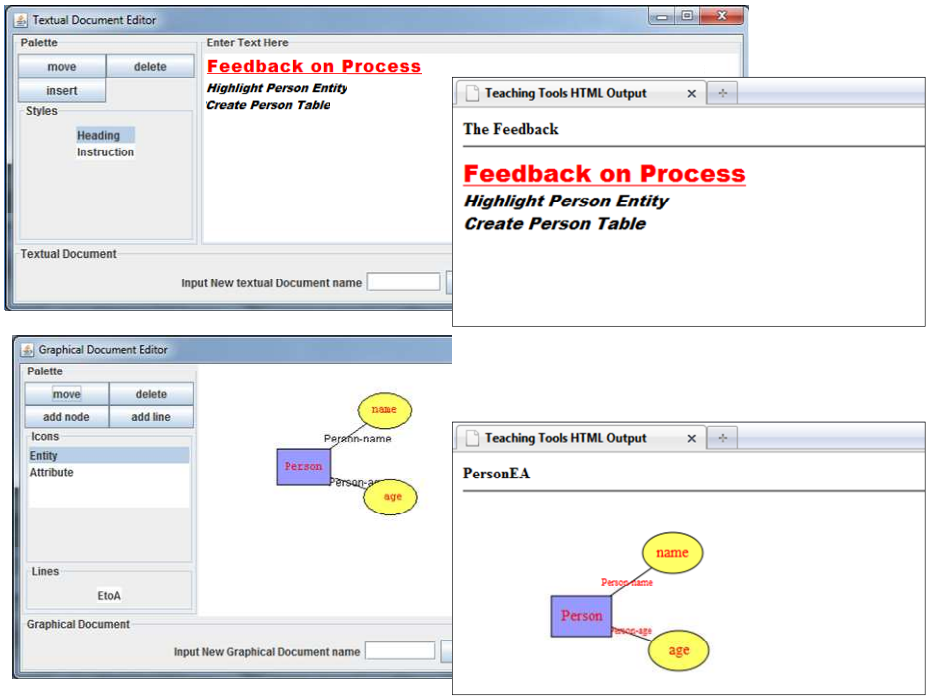


***Figure 19 – Comparison between displaying documents within Java Application and generated website***

Similarly, the representation of graphical documents is shown in the bottom two windows contained in the figure, again the left window is the display from the Java application and the right window is from the website. As with textual documents, this illustrates that the representation of documents is identical between the two displays.

### 7.1.2 Viewing Applications

The figure below contains the two views of applications, the left window from the Java application and the right from the website. It can be seen that the two views are identical, the created module and the generated website meet the defined requirements. The labels for the two arcs within the Java application are slightly different. This is due to errors within the Java application as it is still under development.

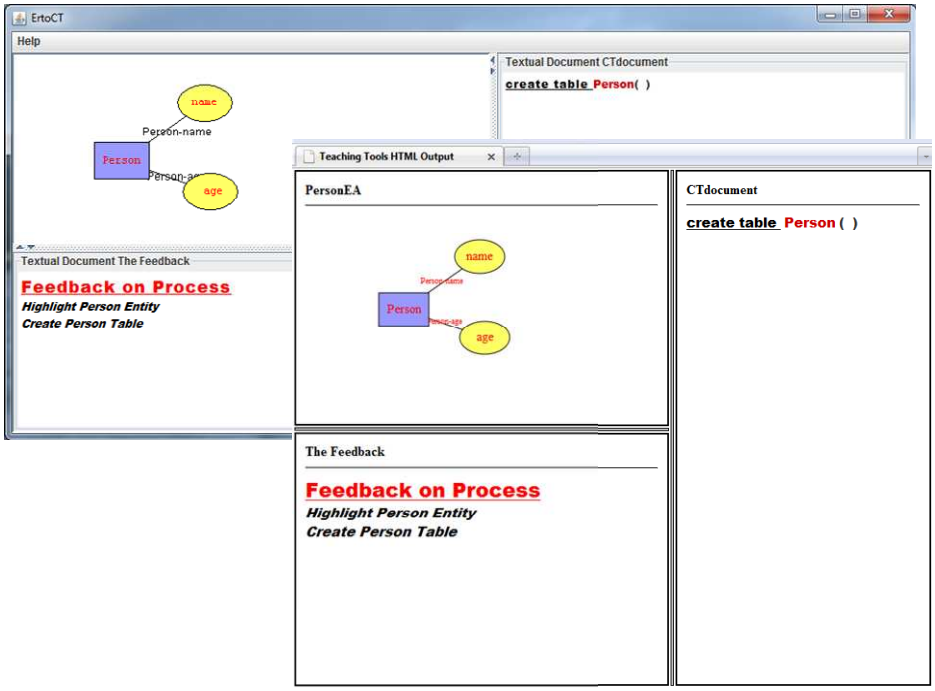


***Figure 20 - Comparison between displaying application within Java Application and generated website***

### 7.1.3 Executing Processes

Two representations of the execution of process are shown in Figure 21, as with the two previous examples the left window is from the Java application and the right window from the website. This shows that processes can be viewed and executed using the website in the same way that they can from the existing application.

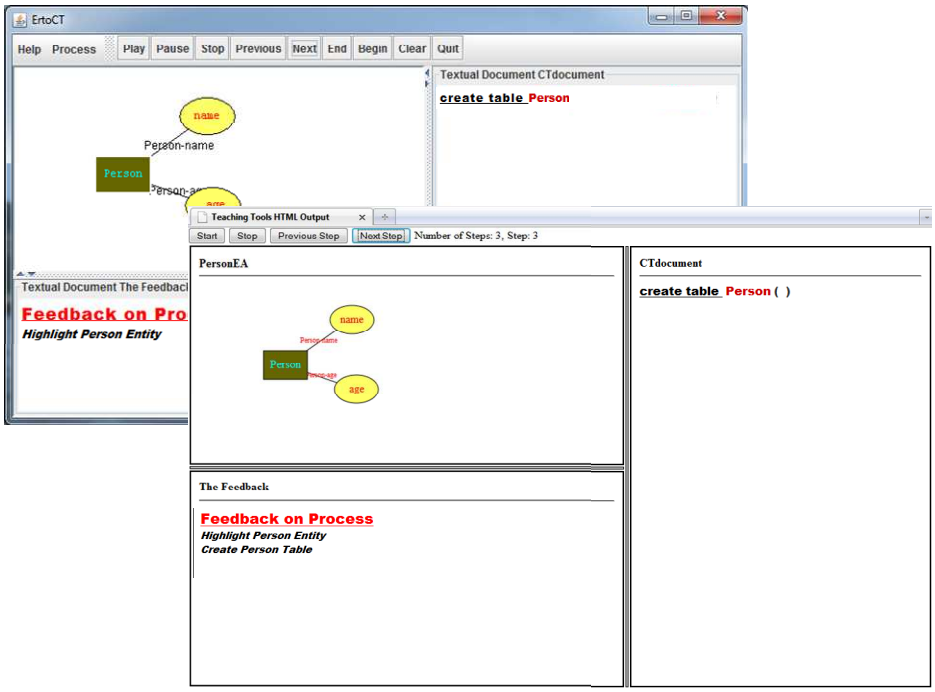


***Figure 21 - Comparison between execution of processes within Java Application and generated website***

## 7.2 Areas for Improvement or Extension

Although the implemented module fully meets the requirement set out at the start of the design phase (Requirements section, page 18) there are several areas for improvement and several opportunities for extension. These improvements can be implemented in a later release and area outlined below.

### 7.2.1 Improving Performance of CSS Files

Within the current solution although the CSS files are dynamic and their output is based on the contents of the data files, the styling for every document within those data files is generated. The result of this is the time to output this styling for a very large data file, in excess of 100 documents, could take a large amount of time.

This excess time could be minimised by modifying the dynamic CSS page to take an argument containing a document, or list of documents. This argument could then be used to only output the styling for that document. This is a very simple modification that, for sites with large data files, should considerably improve performance.

### 7.2.2 'Minifying' CSS and JavaScript Files

A common way of improving website performance is to 'minify' files. Minifying files involves removing any unnecessary whitespace, tab characters, new lines and spaces. Whitespace in large files can make up a large proportion of the file and removing this

whitespace has the effect of making file sizes smaller. Smaller file sizes allow files to be sent over the internet faster than larger files, thus improving performance.

### 7.2.3 Cross Compatibility

An issue with the current solution is the cross compatibility between web browsers. Currently there are minor display issues using Firefox within the Linux operating system. In addition to this, Internet Explorer cannot display the SVG documents. With the introduction of HTML5 this cross browser compatibility should be made easier to implement and, given more time for this project, cross browser capability could easily be improved.

### 7.2.4 Website Styling & Structure

Website styling and structure is another simple addition which, which not changing the functionality of the website, will improve the usability of the website. This usability would be improved by creating a hierarchical structure of pages within the website. Adding styles to pages would also make the website more aesthetically pleasing to use.

### 7.2.5 Ability to Output Zipped Files

This improvement would be a convenience function rather than a key new feature. This improvement would allow users of the module to create an archive of the website files to allow easier transfer to a web server.

## 7.3 Project Success

During the initial stages of the project the requirements for the new module were defined, and these requirements summarised in a single statement of user need. As the implemented module, and the generated website, meets the requirements specified during these initial stages, the outcome can be deemed a success. This document demonstrates this successful outcome and the possibilities for extension.

# 8.0 Bibliography

BBC News. (2010, March 11). *Computer games for the classroom?* Retrieved March 19, 2010, from http://news.bbc.co.uk/1/hi/school_report/8562916.stm

BBC News. (2009, December 2009). *Web links Devon schools to Iraq*. Retrieved March 19, 2010, from http://news.bbc.co.uk/1/hi/england/devon/8400115.stm

Blackboard Inc. (2010). Retrieved Mar 2010, 1, from Blackboard: http://www.blackboard.com/

Bonwell, C. C., & Eison, J. A. (1991). *Active Learning: Creating Excitement in the Classroom.* ERIC Clearinghouse on Higher Education; George Washington Univ., Washington, DC.

Bouras, C. P. (2001). e-Learning through Distributed Virtual Environments. *Journal of Network and Computer Applications , 24*, 175-199.

Bouras, C., Fotakis, D., Kapoulas, V., Koubek, A., Mayer, H., & Rehatschek, H. (1999). Virtual European School - VES. *IEEE Multimedia Systems 1999, Special Session on European Projects.* Florence, Italy.

British Learning Council. (2010). *What is Connecting Classrooms?* Retrieved March 2010, 19, from http://www.britishcouncil.org/learning-connecting-classrooms-what.htm

Cooper, R., Zhao, L., & Wang, C. (2007). A Model Driven Architecture and Toolset for Building Immersive Software Engineering Teaching Tools. *European Conference on Electronic Learning, Copenhagen, October 2007.*

Gavin McArdle, T. M. (2004). A Web-Based Multimedia Virtual Reality Environment for E-Learning. *EUROGRAPHICS* .

IBM. (2009). *IBM Rational Unified Process (RUP)*. Retrieved Mar 2, 2010, from http://www-01.ibm.com/software/awdtools/rup/

Intel. (2010). Retrieved Feb 27, 2010, from Free teaching tools and resources for Teachers - Intel® Education: http://www.intel.com/education/tools/index.htm

Joye, P.-A. (2007, November 28). *LibGD*. Retrieved March 20, 2010, from http://www.libgd.org/Main_Page

Melanie Misanchuk, T. A. (n.d.). *Building community in an online learning environment: communication, cooperation and collaboration*. Retrieved Mar 2010, 1, from http://frank.mtsu.edu/~itconf/proceed01/19.pdf

Moodle Trust. (2010). Retrieved Mar 2010, 1, from Moodle: http://moodle.org/

Open Source Initiative. (2010). Retrieved March 21, 2010, from Open Source Initiative: http://www.opensource.org/

Oracle Corporation. (2004, Aug 11). Retrieved Nov 2009, 21, from Java™ 2 Platform Standard Edition 5.0 API Specification: http://java.sun.com/j2se/1.5.0/docs/api/

PHP Group. (2010, Feb 26). Retrieved Dec 17, 2009, from PHP Manual: http://www.php.net/manual/en/

Richard Cooper, C. W. (2006). IMPLEMENTING IMMERSIVE LEARNING ENVIRONMENTS FOR TEACHING SOFTWARE ENGINEERING. *7th Annual Conference for Information and Computer Sciences* (pp. 218-224). Higher Education Academy.

Richard Cooper, J. M. *Software Systems to Support the Teaching of the Use of Relational Database Systems.*

Tarr, R. (2008). Retrieved Feb 27, 2010, from classtools.net: http://classtools.net/

W3 Schools. (2010). Retrieved Dec 2009, 02, from W3Schools Online Javascript Tutorials: http://www.w3schools.com/js/default.asp

Wang, C. (2005). A Framework and Toolset for the Development of Software Teaching Tools.

Wells, D. (2009). *Extreme Programming: A Gentle Inroduction*. Retrieved 02 Mar, 2010, from http://www.extremeprogramming.org/

World Wide Web Consortium. (2010). Retrieved Mar 2, 2010, from W3C Standards and Drafts: http://www.w3.org/TR/

# Appendix A – CD Containing Source Code and Supporting Documents

# Appendix B – Module User Manual

To generate a website output for a set of documents, start the Java application and load the required documents, applications and processes. Next click the *Web Output* menu button followed by the *Generate Web Output* menu item (Fig 1).

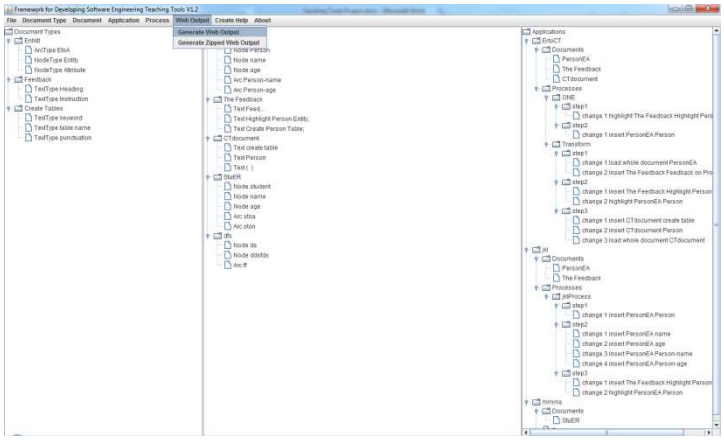

***Fig 1 – Application showing generate output button***

After clicking *Generate Web Output* the save dialog is shown (Fig 2). Select a folder to save the website to and click *Save*. A confirmation dialog will then be shown confirming the save has been successful (Fig 3). If an error message is shown the save has failed, ensure the folder can be written to and try again.

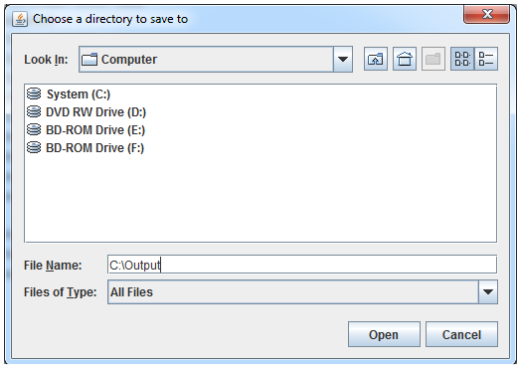


***Fig 2 – Save file dialog***

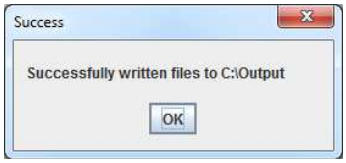


***Fig 3 – Confirmation dialog***

Once the website has been saved, copy the whole folder, in the example this is the folder *Output* contained within the *C:* directory, to the relevant directory on the web server. The website can then be accessed by going to the site *http://my.site.com/path/to/folder*.

# Appendix C – Integration Manual

All required files are contained within the WebOutputPackage. To make use of the module within the application, the WebOutputMenu must be added to the main user interface. To do this the package must be imported within the main frame and an instance of WebOutputMenu created, passing the main frame as an argument to the object. This menu object must then be added to a menu bar where the menu will be displayed.