\pdfoutput=1

\documentclass{l4proj}

%\graphicspath{{images}}

%

% put any packages here

%

\begin{document}

\title{How to Produce a Level 4 Project Report}

\author{Patrick Prosser}

\date{October 18, 2012}

\maketitle

\begin{abstract}

We show how to produce a level 4 project report using latex and pdflatex using the

style file l4proj.cls

\end{abstract}

\educationalconsent

%

%NOTE: if you include the educationalconsent (above) and your project is graded an A then

% it may be entered in the CS Hall of Fame

%

\tableofcontents

%==============================================================================

\chapter{Introduction}

\pagenumbering{arabic}

Teaching programming is inherently difficult. The way it is taught in schools in the UK is by explaining the basic concepts related to a specific topic, presenting simple examples to illustrate how these concepts can be applied and posing a more complex problem for pupils to solve. However, in the initial stages of becoming programmers, beginners often lack a good enough understanding of the domain from just the simple examples to be able to solve the problem~\cite{cooper2008, renkl2005}. This can lead to pupils struggling to find a solution, rather than gaining a better understanding of the problem-solving process.

A good way of teaching somebody an intellectual activity is by showing them the process of thinking involved in carrying it out. This is a form of apprenticeship known in the literature as "cognitive apprenticeship"~\cite{cognitive\_apprenticeship}. In any apprenticeship model, the learner needs to see many examples of the activity to be learned in order to develop the experience necessary to attempt a new, related, activity. Unfortunately, due to limited number of hours dedicated to each individual subject in schools, teachers are somewhat restricted to using only the traditional methods of teaching. Time simply would not allow them personally to show their pupils many examples of what cognitive steps they should undertake in order to solve a problem.

Step-by-step guidance of the process of solving particular problems can help beginners gain a better understanding of the problem-solving process generally. Books provide such guidance in the form of worked examples. These examples have proven to be effective ~\cite{sweller1985, tarmizi1988, ward1990, zhu1987}. It has also been found that they reduce the cognitive load when acquiring a skill~\cite{sweller98}. However, such books may not necessarily accommodate the needs of a particular teacher. Furthermore, finding a close enough example for a particular topic may become a time-consuming and discouraging activity for a teacher.

Having this in mind, a Glasgow University PhD student, Yulun Song, has developed a Java standalone application called \textit{Interactive Worked Examples (IWE)}~\cite{song-thesis}. It aims to address the issues mentioned above as well as to evaluate to what extent such an application will prove effective in lowering the learning curve for students in Computing Science. It consists of two interfaces: one for students and one for creators of worked examples (who are typically teachers as well). The author interface enables the creation of examples to accommodate a teacher’s specific needs. The student interface provides users with a selection of examples to work on.

The application has proven to be effective at enhancing the teaching of Computing Science in university~\cite{song-thesis}. Since the research questions around IWE were to explore the extent to which it can fit in the teaching process in a university context and whether it would be a potentially successful learning technique, the prototype does not aim at large scale deployment. A sensible next step is to put the system into use in schools, where support for computing education is urgently needed~\cite{royalsoc2012}. However, many issues in deploying IWE arise because of it being in the form of a Java standalone application. In schools in the UK there tends to be a blanket policy about the systems provision on any subject. In order to install a program on a school machine, a request to the service provider responsible for the particular school needs to be made. The service provider will then need to analyse the risk that installing a new program will pose to the whole system and submit a further request to a local authority responsible for the particular school. This overhead would be enough to prevent most teachers from considering adoption of the software, both from a time and cost standpoint.

This issue of software provisioning in schools gives the major motivation for this project to recreate IWE as a web-based application in order to start effectively presenting worked examples in a larger context. This will avoid the complicated and time-consuming process of installing IWE in schools. Furthermore, schools will be able to receive the latest updates of the application and its worked examples with no effort. Ultimately, a web-based system could share worked examples developed nationally and even internationally. The web-based version of IWE is called \textit{Worked Examples Viewer (WEAVE)}.

In addition to being a more easily deployable version of IWE, WEAVE takes a step further to move from author-student to author-student-teacher target user groups. This brings in interesting new aspects. Teachers will be able to see personalised information about how their pupils interact with the examples. Authors, on the other hand, will receive information about the general usage of these examples, rather than personalised one.

Another benefit of WEAVE being web-based is that the worked examples in the system will not be limited to the ones created by one teacher or a group of teachers only. Instead, examples created by any teacher will immediately be available to everyone. This would contribute to a collaborative way of developing such examples and would give the chance for pupils to undertake further learning if they desired so. Furthermore, teachers would be able to benefit from their colleagues’ expertise as well as get ideas and adjust them to their specific needs with less effort than creating new examples from scratch. Ideally, such a system can be revolutionary in improving the teaching practices in schools, help teachers understand the difficulties of their pupils and enable them to help each other to become better in teaching Computing Science.

The rest of this dissertation describes more background for the context of the project, the requirements for, as well as the design and the implementation of WEAVE together with the testing methods that were used to ensure that the application works as intended. An evaluation chapter follows making conclusions about how easy to use and to integrate successfully in everyday teaching practice WEAVE is. The final chapter is dedicated to the conclusion and the future developments for the system which will be addressed shortly.

\chapter{Background}\label{background}

\section{Worked Examples}

\subsection{Definition of a Worked Example}

Clark defines a worked example as “a step-by-step demonstration of how to perform a task or how to solve a problem"~\cite{clark2006}. Another definition for worked examples is given by Atkinson as “instructional devices that provide an expert's problem solution for a learner to study.”~\cite{atkinson2000}. An effective worked example consists of a problem description, steps towards the solution and instructions at each step representing an expert’s process of thinking~\cite{renkl2005}. Of key importance is the step-by-step guidance for reaching the solution. It encourages the learner to form their own explanation for the undertaken step~\cite{renkl2004} as well as think about what might follow next before they proceed. In essence, worked examples help novices to build an understanding of a concept so that in later stages they will be able to effectively apply this understanding to solve other problems related to this concept.

\subsection{Worked Examples and Learning}

The common assumption that the best learning is by practicing solving problems is not necessarily true for learning Computing Science. Renkl~\cite{renkl2005} argues that without being exposed to worked examples first, novices have a very restricted knowledge on the domain to be able to effectively reach a solution. Solving problems involves a lot of working memory resources. However, the memory capacity of beginners should be used for building new knowledge instead. Clark argues that solving practice problems leads to using too much memory capacity thus not leaving enough of it for learning new knowledge~\cite{clark2007}.

Studying worked examples “is one of the earliest and probably the best known cognitive load reducing techniques”~\cite{paas2003}. It has proven to be effective in learning how to solve problems~\cite{van1997}. While worked examples reduce the cognitive load, they also provide a better understanding of the concepts under consideration. This builds up the necessary expertise required to solve a particular type of problem effectively.

\subsection{Worked Examples in a School Context}

The traditional methods of teaching Computing Science in schools across the UK do not include the best proven method to learn a cognitive skill described above. Often in schools, Computing Science concepts are introduced by explaining what the concept is, followed by a simple example. Then pupils are presented with a problem to solve themselves. The jump to problem solving is too quick and the importance of worked examples has not influenced the teaching methods. Keeping in mind that teachers are often limited time- and money-wise, a possible reason for this is because there is no easy means of finding and adapting existing worked examples to the specific needs of a teacher.

As part of his research project, the former Glasgow University PhD student Yulun Song developed software to facilitate the creation and viewing of worked examples~\cite{song-thesis}. The thesis statement for the research outlines the basic aims for the project. The system developed is such that it:

\begin{itemize}

\item{``delivers usable, best practice interactive worked examples to students in a computing science context;''}

\item{``enables teachers to create such interactive worked examples without bespoke programming, and to evolve them on the basis of feedback from the students.''}

\end{itemize}

Song was particularly interested in Computing Science problems due to their transformation-based nature. They involve the analysis and the transformation of one representation of the problem, such as text definition or a diagram, into another representation, i.e. the solution. An example described in the research thesis is building a database system from a specific set of requirements expressed in the form of a problem description in human language. The text describing the problem needs to be transformed into a graphical representation of the same problem - an ER diagram, which is then translated into a machine language such as SQL. Judgement and decision-making play a huge role in solving such a problem. However, these only come with experience and in order to gain such experience Reed and Bolstad~\cite{reed1991} claim that one example- which is the typical case in schools- might be insufficient. In his research, Song argues that a system that enables the user to view multiple worked examples would prove efficient in such a context. He therefore developed a tool for the provision of worked examples in Computing Science so that the user is exposed to more than one of those examples.

\subsection{Problems with Existing Methods for Delivering Worked Examples}

One can argue that worked examples can be found in many books and lectures so at first it may seem questionable what value would software bring to the existing provision. Song's thesis~\cite{song-thesis}, however, raises some strong arguments to be taken into consideration, as shown below.

\begin{itemize}

\item The worked examples in books or lectures are not interactive enough. The readers of books or the attendees of a lecture are presented with some examples, but often the process of thinking why a particular action is undertaken or is a better option for reaching a solution remains unexplained. One can argue that the university context has some grounds for interactivity or discussions. Yet many students may not exploit this due to shyness or simply because they might not know what questions to ask. Even if some interaction happens, this is not recorded or captured as part of the teaching process so the students cannot go back and review it.

\item The worked examples present may not fit well enough to the teaching needs. Books aim to target a large portion of potential readers so they need to be general enough to fit every reader's needs. However, this means that one particular reader may need to adjust their studying or teaching around this general example. What would be more beneficial - and Song aims to address - is to adjust the worked examples depending on the teaching or learning needed.

\item Worked examples in books provide little or no feedback on how they were used to the author or to teachers who benefit from using such examples in their teaching. For example, the only available information for a book would be the number of copies sold. This would not provide any insight on the value the examples brought to the reader. What is desired is information about how a particular worked example was used, were there any problematic areas and how the readers benefited from it. Such information would allow the authors to improve their future work at constructing worked examples. In addition, this information could be beneficial to teachers or lecturers who could use it for assessing what parts of the example were problematic and adapting their teaching accordingly.

\end{itemize}

\subsection{How Does a Computer-Based Application Solve the Problems with the Traditional Methods of Delivering Worked Examples?}

The piece of software proposed and developed as part of Song's work aims to address all the issues mentioned above. The student becomes actively involved with the material since revealing the steps required to reach a solution is under their control. The entire problem solving process can be fully captured and the students can easily go and review parts causing confusion. Complete explanation of every step is provided, enabling the student to follow the process of thinking of an expert. Revealing the solution step by step encourages thinking about the next logical step and guides the student towards the correct direction of thinking before they get confused. In addition, usage data can easily be captured to give feedback on how these worked examples were used. Data intended to be collected includes time spent at each step and answers to any questions present in the examples. Such information can be beneficial to two groups of people:

\begin{itemize}

\item Authors of worked examples. They could benefit from the knowledge of the time the majority of the students spend at each step. If this time exceeds dramatically the expected time for this step, this can be an indication that the step is unclear and brings confusion. Knowing this, authors could update the example by including a better explanation or by breaking this step into more than one steps and examine the effect this has. This way authors will learn how to build their examples and this will also bring benefits to the reader in terms of provision of improved worked examples.

\item Teachers. The knowledge that their students visit a step multiple times or spend too long before proceeding would indicate to the teacher that their students do not understand the material for this step well enough and they might need to revisit it in class.

\end{itemize}

\section{Interactive Worked Examples Tool}

The following subsections provide more information about the IWE tool.

\subsection{Intended Flow of Interaction}

There are two well-distinguished groups of users of the IWE tool – authors of examples, and students. Each group is serviced by a separate interface of the application. The flow of interaction of these groups with the system, as presented in Song's thesis, is shown on Figure~\ref{iwe\_flow\_of\_interaction}. The original figure can be found in Song's thesis as Figure 2.6 (p.57). The following bracketed numbers correspond to the associated numbers in the Figure.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics{images/IWE\_flow\_of\_interaction.png}

%\vspace{-30mm}

\caption{IWE Flow of Interaction.}

\label{iwe\_flow\_of\_interaction}

\end{figure}

\begin{itemize}

\item{Authors can create (1) and modify (5) interactive worked examples through the author interface. This interface also presents them with any student feedback (4) on these examples.}

\item{The student interface serves as a worked examples viewer (2) where students are given the opportunity to ask questions and write comments (3).}

\item{The interaction between teachers and students is direct rather than through the system.}

\end{itemize}

\subsection{Structure of IWE}

IWE stores the worked examples in XML files. There are three types of files that are of particular interest for this Level 4 project- \texttt{Documents.xml}, \texttt{Applications.xml}, and \texttt{Processes.xml}. The structure of those files is graphically represented on Figure \ref{xml\_files}. A description of each file can be found below.

%\vspace{-7mm}

\begin{figure}[t!]

\centering

\includegraphics[width=\textwidth]{images/xml\_files\_structure.jpg}

%\vspace{-30mm}

\caption{Structure of the XML files storing the worked examples created in IWE.}

\label{xml\_files}

\end{figure}

\begin{itemize}

\item The \texttt{Documents.xml} file stores the collection of documents created by an author. A document is one of the representations involved in a particular worked example- perhaps it is the problem specification, or an intermediate solution, or the solution. It is split into fragments which are small logically separated portions of the document. The reason for splitting the document into fragments is so that the document can be revealed gradually, to show the step-by-step problem solving process. Individual fragments can also be highlighted to be brought to the viewer’s attention.

\item The \texttt{Applications.xml} file stores layout information about worked examples, bringing together the particular documents involved in the worked example. The way the documents are laid out visually, in panels, is recorded.

\item The \texttt{Processes.xml} file defines the steps for the worked examples. For each step there are a number of changes and an explanation of those changes. There are two types of changes. The first type specifies which fragment of a document is involved in this change. These fragments can be shown, hidden, highlighted or unhighlighted depending on the effect the author is aiming to achieve. The second type of changes corresponds to a question and possibly a set of options the user can select from in an attempt to answer it.

\end{itemize}

There is one more type of XML file which is not shown on Figure \ref{xml\_files}. It contains information about different styles that can be used for the worked examples- there is a similar, although simpler, version of the style mechanism found, for example, in word processors or CSS style sheets. There is no need for this file to be discussed in any detail. However, the reader needs to know that documents have styles associated with them depending on the type of document, enabling different fragments within a document to be shown in different typographical styles.

\subsection{Main Features of the IWE Student Interface}\label{iwe\_student\_interface}

The student interface aims to provide an effective worked examples viewer. A screenshot of the final version of Song's prototype can be seen on Figure \ref{iwe\_student\_interface}. The most important characteristics are labeled with numbers and are detailed below.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics{images/iwe\_student\_interface.jpg}

%\vspace{-30mm}

\caption{A screenshot of the student interface of IWE.}

\label{iwe\_student\_interface}

\end{figure}

\begin{enumerate}

\item An area for showing the worked examples installed on the system and enabling the user to choose an example to work on.

\item Panels showing different documents for a particular worked example.

\item An area for controlling transitions between steps.

\item An explanation area where the expert's process of thinking involved on the current step is shown.

\item Highlighting of the newly appeared text at a particular step for drawing the user’s attention to the new content relevant for the current step.

\item Highlighting of fragments of interest for a particular step.

\end{enumerate}

As the student uses the controls in area (3) to move through the worked example, the contents of the documents panels and the explanation area change to reveal the developing solution and the thinking process behind it.

Other features of the IWE student interface, which are not shown on the screenshot, are the ability of the tool to ask the user questions and to record data such as time spent at each step and answers to questions.

\subsection{Relation of IWE to this Level 4 Project}

The evaluation on IWE conducted as part of Song's research clearly shows the benefits of the tool. It has proven to be well-accepted and valuable as a technique to enhance a student’s learning experience. It also achieved its goal to enable teachers to more easily and quickly develop worked examples to fit their needs. The aim of this prototype, however, doesn't cover deployment of the software in educational institutes but rather it has demonstrated that it would bring benefit to both students and teachers.

The motivation for this Level 4 project is to make use of the findings in Song's research and take a step forward in deploying the software in schools across the UK. The focus of the project will be reconstructing the student interface of IWE into a web-based form, as well as providing a new interface for teachers to separate them as a distinct group of users. The author interface of IWE is beyond the scope of this project and no understanding of any of its aspects is needed by the reader to follow this dissertation.

For the rest of the dissertation, the reader’s attention will be drawn to the transition process of the Java standalone application IWE into a deployable online version called \textit{Worked Examples Viewer (WEAVE)}.

\chapter{Requirements}\label{requirements}

This chapter provides a detailed description of the functional and non-functional requirements for WEAVE.

\section{Preface}

As described in Chapter~\ref{background}, this Level 4 project builds upon an existing system for facilitating the use of worked examples in educational context. The evaluation of IWE clearly shows that such software would be a valuable asset contributing to the learning process of students. Due to the overly complicated procedure required to deploy IWE in schools while it is in the form of a Java standalone application, the need to turn it into a more easily deployable online version arose. Interviews with highly motivated teachers, who are part of PLAN C project\footnote{\texttt{http://www.cas.scot/plan-c/}. Last accessed March 24\textsuperscript{th}, 2015.}, have identified the need for one more interface to be used in schools. In order to improve their teaching practice and to be able to provide high quality feedback to their pupils, these teachers would benefit from knowing how pupils in their classes use these worked examples. Information that would be valuable for them includes:

\begin{itemize}

\item identification of which pupils interacted with which examples;

\item aggregated information on answers selected for multiple choice questions and the pupils that selected each answer;

\item answers to any open-ended questions;

\item information about the average time spent at each step of an example as per the whole class (or an individual); and

\item summary data of the total time spent at an example and the last step reached by each pupil in the class.

\end{itemize}

This project aims to achieve four goals.

\begin{enumerate}

\item[G1.] Build a web-based viewing system that is interoperable with the author interface of IWE, i.e. ensure that worked examples created using the old system can be viewed in the new system.

\item[G2.] Provide an interface for teachers that will help them gain more information on how the worked examples are used by their own pupils.

\item[G3.] Replicate as closely as possible the student interface of IWE.

\item[G4.] Ensure that worked example authors can view usage data in an anonymous manner, such that individual pupils, classes or schools are not identifiable.

\end{enumerate}

\section{Classification of Requirements}

The requirements are classified according to the \textit{MoSCoW}~\cite{jonasson2012} classification method. The categories considered are:

\begin{itemize}

\item \textbf{must-have}- requirements that are crucial for the achievement of the goal of this project and must be implemented.

\item \textbf{should-have}- requirements that are considered to be important but not necessarily crucial for achieving the goal of this project and should be implemented.

\item \textbf{could have}- requirements that have been identified as features that would add further value to the prototype but are thought of as stand-out ones rather than ones contributing to the correct functioning of the prototype and may not be implemented due to constraints.

\end{itemize}

The \textbf{would-like} category coming from the \textbf{W} in \textit{MoSCoW} is not part of the classification methods used for this project due to the fact that all the requirements fit comfortably in the other categories.

\section{Functional Requirements}

\subsection{Interoperability with the Existing Author Interface}

The existing system uses XML files to store the worked examples. The structure of these files is shown on Figure \ref{xml\_files} in Chapter~\ref{background}. The web-based system will need to read in worked examples from these data files. Furthermore, feedback from pupils and teachers will inevitably lead to changes being required in some of the worked examples. The existing authoring tool supports editing of worked examples, and it is expected that it will still be used to make such changes. WEAVE will need to be able to support these changes. Due to the fact that the update model of IWE is destructive- no versioning of the examples is supported- and that WEAVE does not provide means for modifying examples, the update model will need to follow the one of the old system.

The prototype:

\begin{itemize}

\item \textbf{must} be able to parse an XML file containing the fragmented problem specifications of the worked examples and their solutions.

\item \textbf{must} be able to parse an XML document containing information about individual steps of the worked examples (e.g. which fragments of a document must be shown/hidden/highlighted/unhighlighted, the explanation associated with a step and a question if one was provided).

\item \textbf{must} be able to parse an XML document containing information about the layout of worked examples (e.g. number of panels for the example, their order and problem solutions associated with each panel).

\item \textbf{must} be able to parse an XML document containing information about the styling associated with each example (e.g. font style, font size, etc.).

\item \textbf{must} be able to support easy addition of new worked examples created using the old authoring tool.

\item \textbf{must} be able to incorporate new versions of worked examples already installed in the web-based system.

\end{itemize}

\subsection{Teacher Interface Requirements}\label{privacy}

A major part of the contribution of WEAVE is to enable teachers to receive information about how their pupils worked with examples, while authors of such examples and Computing Science researchers must receive such data in an anonymised way. The desired effect is for teachers to be able to see usage data for their classes as well as individuals in these classes. However, protecting the privacy of both teachers and pupils is a major issue. The authors of worked examples will be able to see any usage data for the examples they created. If this data is informative enough for them to identify the person standing behind this data, this would be highly unethical and would violate privacy.

In this section, the requirements for the teacher interface are outlined. The next chapter will describe how the privacy issues mentioned above are resolved by the system and will discuss in further detail these requirements.

The teacher:

\begin{itemize}

\item \textbf{must} be able to choose whether to track their pupils' progress or not.

\item \textbf{must} be able to view how the students in a particular class interacted with the examples.

\item \textbf{must} be able to view how a particular student interacted with the examples.

\end{itemize}

\subsection{Replication of the IWE Student Interface}

One of the goals for this Level 4 project (G3) is to replicate the student interface of IWE as closely as possible. The reasoning behind the requirements for the student interface, as well as the positive conclusion from their evaluation, are described in detail in Song's thesis. These were found acceptable for the purpose of this project. In short, the research literature describes the qualities of successful worked examples~\cite{renkl2005}, and these qualities were included in Song's design. Their implementation in his software system was shown to be suitable for use by students in the extensive evaluation carried out.

The prototype:

\begin{itemize}

\item \textbf{must} enable the pupil to select a worked example from a list of existing examples.

\item \textbf{must} support multiple panels for the different parts of the problem solution.

\item \textbf{must} contain a dedicated area for the explanation.

\item \textbf{must} support showing/hiding/highlighting/unhighlighting of fragments.

\item \textbf{must} support the option to ask pupils questions.

\item \textbf{must} enable the pupil to go back and forwards through steps.

\item \textbf{must} highlight the newly introduced fragments at each step.

\item \textbf{should} record time spent at a step.

\item \textbf{should} record answers to questions.

\item \textbf{should} enable the pupil to reset the example there are working on.

\item \textbf{should} provide a means for drawing the pupil's attention to the newly introduced fragments.

\end{itemize}

\subsection{Additional Features Needed for the Student Interface}

This section describes the requirements for satisfying goals G2 and G4 – to support identifiable usage information for the teacher, and anonymous usage information for worked examples authors and for Computing Science education researchers. It is important that each teacher is able to link usage data to their groups/pupils, while authors of such examples must not be able to identify by any means what the group or who the pupil is due to the privacy issues discussed previously.

The prototype:

\begin{itemize}

\item \textbf{must} be able to record usage data for an example in such a way that the pupil's teacher will be able to know who the data belongs to.

\item \textbf{must} be able to record the usage data for an example in such a way that the true identity of the pupil associated with this data is not revealed when this information is sent to authors of worked examples and Computing Science education researchers.

\item \textbf{must} allow the pupil to use the system without any identifying information.

\end{itemize}

\section{Non-Functional Requirements}

The non-functional requirements for WEAVE are guided mostly by the web-based nature of the system and by the context it is intended for. In order for pupils to be able to study the worked examples effectively, and also due to the small workstation screen sizes found in schools, the area showing the worked example should be maximised. Furthermore, due to the step-by-step nature of worked examples, some steps may put more emphasis on the explanation while others might be more intensive in the problem specification areas so the system must be able to deal with such situations accordingly. In addition, features which make the interactions with the examples more convenient and which would minimise effort, such as shortcuts and appropriate fitting of the whole system on the screen, are highly desirable.

The web-based nature of WEAVE poses a possible problem when uploading modifications to existing examples because pupils might be working on these examples at the same time. Consistency must be ensured in such cases, meaning that the pupil should be able to see either the old or the new version of the example, rather than a mixture of both.

Since pupils may have not worked with such a system before, they may benefit from a brief guide on how to use WEAVE in an optimal way.

These considerations form the following requirements:

\begin{itemize}

\item The prototype \textbf{must} be easy to use.

\item The worked examples \textbf{must} fit the entire screen.

\item The size of the area showing the worked examples \textbf{must} not exceed the size of the screen.

\item A modification to a worked example \textbf{must} not affect pupils doing the same example.

\item The student interface \textbf{should} include a tutorial on how to use the system.

\item The teacher interface \textbf{should} provide information on how to use each feature.

\item The panels showing the problem content \textbf{should} be resizable.

\item The explanation area \textbf{should} be resizable.

\item Shortcuts for easier transition between steps \textbf{could} be added.

\end{itemize}

The following chapter will describe the design decisions which were constructed based on these requirements.

\chapter{Design}

This chapter outlines the design decisions for this Level 4 project. As it was previously described, WEAVE is based on the earlier Java standalone application IWE. This affects to a great extent these decisions. Following the well-established software engineering principle of reusability, guidance for the design of WEAVE was to reuse any good aspects of IWE's design while improving its weaknesses.

\section{Storage of Data}

The scope of this Level 4 project does not include the creation of a web-based author interface. This means that the design of WEAVE must ensure an easy and efficient way for importing and storing the existing worked examples. In addition, WEAVE must support uploading updates to existing examples. As described in Chapter~\ref{background} , the examples are stored in the form of XML files. To remind yourself of the structure of these files, please refer to Figure~\ref{xml\_files}.

During the exploration process of IWE, a weakness of storing the information about the examples in the XML files used by IWE was identified. Since these files are easy to access and modify by the authors of worked examples, and this is typically much easier than modifying the worked examples using the author interface, one may be tempted to make changes to the examples manually. However, references to some objects may be present in more than one of these XML files. For example, in the \texttt{Documents.xml} file, fragments are stored as individual elements identified by a fragment id. When the steps for the example are defined in the \texttt{Processes.xml} file, each fragment is referred to with its id and also, for an unexplained reason, the fragment text is included. A problem with storing information about the examples in files is that if an element is modified in one of the files, consistency about this element must be ensured across all copies, or else the system fails. It is not trivial having to find the same feature across multiple files and in the end consistency and validity are not guaranteed. This appears to be a case of poor software engineering, as there is no apparent reason for the fragment text to be copied between XML files, and not just the fragment id.

In the new system the worked examples are stored in a database where the relationships between different objects are expressed via foreign keys. So, for example, the same fragment is not copied but referred to by its foreign key. In addition to ensuring consistency, using a database adds a level of reliability that the data is valid because internal integrity checks are made before saving an object to the database. Further advantage is that the Django administrator interface allows an easy means of exploring and modifying the examples in a consistent and safe manner.

\section{The Data Model of Worked Examples}

The way the worked examples are structured in the XML files generated by the author interface of IWE is shown in detail on Figure~\ref{xml\_files}. The bullet points below can be used as a quick reminder. The structure of the XML files determines to a great extent how the worked examples are stored in the database. Figure~\ref{WEERDiagram} shows the UML Entity-Relationship diagram for the worked examples in WEAVE. The numbers associated with a particular relationship between the entities correspond to the numbers in the bullet points below to illustrate the similarities between the two methods of representation of the worked examples for the different systems.

\begin{itemize}

\item Each worked example consists of one or more panels (1), which show the text of a document for the worked example (2).

\item The text of each document of a worked example is split into different fragments (3) (portions of the text for the example).

\item Processes define the steps for each example (X). There are many steps for each worked example (4).

\item Each step of an example consists of a set of changes (5) and an explanation (6).

\item A change can involve either showing/hiding/highlighting/unhighlighting a fragment (7) of a document (8) or asking a question (9) (open-ended or multiple choice (10))

\end{itemize}

Please note the bullet point that has X instead of a number. The \texttt{process} element in the XML files was found unnecessary when storing the worked examples in the database and it would have led to redundancy. Instead, each worked example (referred to as an \texttt{application} in both Figure~\ref{xml\_files} and Figure~\ref{WEERDiagram}) is associated with a set of steps. The order of these steps is determined by the \texttt{order} attribute of the Step entity. When a worked example is loaded, all the steps for this example are retrieved and are shown in the order defined by their \texttt{order} attribute.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/WorkedExamplesERDiagram.jpg}

\vspace{-30mm}

\caption{A UML Entity-Relationship diagram for the worked examples.}

\label{WEERDiagram}

\end{figure}

Please note that this is a simplified version of the objects storing the worked examples for illustration purposes. As with the XML files, there are objects for the different type and style of documents and fragments. Inserting them in the UML Entity-Relational diagram on Figure~\ref{WEERDiagram} would have made it unclear and difficult to read. The type and style of documents and fragments are defined using foreign key relationships with the respective objects. On Figure~\ref{WEERDiagram}. these foreign key relationships are expressed by adding \textit{(FK)} next to the name of the object for the particular type or style involved in this relationship.

\section{Design for the Translation of the Worked Examples from XML Elements into Database Objects}\label{command\_line}

The database is populated with the elements stored in the XML files produced by the author interface of IWE via a command line interface. A population script- \texttt{population\textunderscore script.py} file- is run on the command line. It takes one optional parameter to specify the path to the XML files. If this parameter in not provided, the script will look for the files in a default directory for storing the examples. More details on the implementation of this population script are provided in Section~\ref{population}. Instructions on how to run the script and where the example directory is are provided in a \texttt{readme.txt} file in the project.

\section{Authentication and Privacy}\label{authentication}

A significant discussion point was how to authenticate teachers and their students due to the privacy and ethical issues discussed in Section~\ref{privacy}.

As a reminder, the main issue around privacy/ethics is to avoid storing information in the system that could identify individual schools, teachers or pupils. If this can be achieved in the design, then data gathering for worked example authors and system designers, essential to improve both worked examples and the system itself, can be carried out with no requirement for informed consent for data usage by the pupils or teachers.

The requirement here, then, is that names for teachers, classes and pupils do not contain identifying information.

One possibility was for students to create their own accounts and give their usernames to the teacher, so that he/she could monitor their progress. However, this approach could potentially result in various complications.

\begin{itemize}

\item Pupils would need to be explicitly instructed that their accounts should not reveal their true identity, since this information would be stored in the system database, and visible to systems administrators.

\item Pupils would not have any benefit of having their own accounts as only the teacher will be the one who would use their usernames for something meaningful, i.e. to check their progress.

\item This approach solves identification at an individual level but each pupil needs to belong to a group as well, to assist the teacher in monitoring the progress of a particular class, so this becomes an additional management overhead for the teacher.

\end{itemize}

A second option was considered, in which the teacher would create an account for each of their classes. For this account, they would need to create usernames for their pupils. Keeping in mind that a teacher would often have more than one class and that each class consists of twenty to thirty pupils, the following problems arise:

\begin{itemize}

\item The teacher would need to create a lot of accounts and this could be a non trivial and time consuming task.

\item The teacher would need to ensure that they will be able to match each of their pupils to their id since they must not use any names due to the privacy issues mentioned above.

\item Account names should not be guessable by other pupils, to avoid any one of them pulling a prank on another pupil by using their username. Hence the teacher can't use an easy sequence of anonymous usernames such as p1, p2, p3 etc. On the other hand, long, anonymous usernames should be avoided since these are tedious to store and transmit and are likely to lead to errors in use.

\end{itemize}

To get around the privacy issues and the problems with the options described above, a third approach based on the general idea of the second one was adopted. The main points of this approach are:

\begin{itemize}

\item Teachers create their own accounts, with a username that does not identify them or their school.

\item Once logged in, they will be able to create groups, one for each of their classes, with a name that does not identify the school.

\item On creation of the group, the teacher needs to specify the number of the pupils in this class. WEAVE will then generate random ids for these pupils. An id consists of two random letters followed by a single digit. While being short enough to be easily remembered, such an id ensures that the privacy of pupils is protected due to its random nature. Furthermore, the number of possible combinations for all pupil ids is large enough so that it will be highly unlikely that pupils will be able to “guess” one of their classmate’s pupil id and work with the examples on their behalf.

\end{itemize}

Talking to a lead teacher, Mr Peter Donaldson, who is part of the PLAN C project\footnote{\texttt{http://www.cas.scot/plan-c/}. Last accessed March 24\textsuperscript{th}, 2015.}, a potential inconvenience of this approach was identified. Using WEAVE for longer periods than one academic year could lead to a significant increase of the groups. In addition to the growth of the number of groups, some teachers might prefer to use the same name for their classes across years. To resolve these issues, a further classification of groups by the academic year the group belongs to was adopted.

Mr. Donaldson was generally happy with the idea that teachers select the number of pupils for each group at the creation of this group. However, he pointed out that it is possible for a pupil to arrive in a class at a later stage than the beginning of the academic year. Using the selected approach would have required that the teacher creates a new group just to add one pupil only. This could be very problematic, because data about the same pupils would be spread across two different groups and most of the pupils will be given two pupil ids which could become really confusing for both teachers and pupils. To avoid these problems, the option for teachers to update a group was added to the design decisions for the authentication part. In addition, groups can also be deleted in case of creation of unneeded groups.

\section{The Data Model for the Users of WEAVE}

There are four classes in the database which represent the special relationship between pupils and their teachers and classes. These are shown in the UML Entity-Relationship Diagram shown on Figure~\ref{UsersERDiagram}.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/UsersERDiagram.jpg}

\vspace{-30mm}

\caption{A UML Entity-Relationship diagram for the users of WEAVE.}

\label{UsersERDiagram}

\end{figure}

Each teacher can have many groups. A group is associated with a particular academic year. This would allow the presence of many groups with the same name for a particular teacher, as long as they are in different academic years. There can be many groups for each academic year. Each student belongs to one group. There are many students for a group.

\section{Presentation of Data to the Teacher}\label{graphs}

The main goal of the teacher interface is to present to the teacher data associated with a particular group or pupil. Three different types of data are recorded from the student interface:

\begin{itemize}

\item Time at each step.

\item The direction of the transition to each step i.e. is the pupil going backwards or forwards to a step.

\item Answers to questions.

\end{itemize}

Careful consideration was needed to reach a solution that would visualise this data in a way which would enable the teacher to comprehend it easily and encourage further analysis. In addition, the ability for the teacher to be able to view data both at a class and at an individual level further influenced the design decisions.

In a discussion with Mr. Donaldson- as a teacher who would use such data- it has been decided that presenting the data in the form of graphs would be beneficial to teachers as they would be able to identify patterns and any exceptional events for a particular worked example. These graphs should reveal information about the performance of the whole class as well as of individual pupils at each step of the example keeping in mind that some steps have questions. For this purpose, five different types of graphs were decided upon:

\begin{itemize}

\item Average Time. This graph would show the average performance of the pupils in the whole class on a particular example. Different steps will be represented on the x-axis of the graph by their step number. The average time spent on each step will be shown by the y-axis. Since the x-axis consists of the step number, this is not very informative to the teacher because they would need to look at the actual example to remind themselves about the context of the step. To avoid the need for that, hovering over the point representing the step will show the beginning of the explanation. Clicking on that point will open a window showing the whole text of the explanation, the average time spent on that step and how many pupils made a backwards transition to the step. As mentioned above, some steps contain a question, rather than an explanation. Such steps will be identified by a question mark in front of the step number on the axis label. Instead of showing an explanation on mouse hovering, a message encouraging teachers to click in order to see pupils' answers is shown. Clicking on the point for that step will show the question, and a bar chart with all the possible answers and how many and which pupils selected each option.

\item Student Time. This graph is conceptually the same as the Average Time graph. Instead of showing information about the whole class, however, it shows the total amount of time spent at each step by a selected pupil.

\item Student Answers. This column chart shows the options for a selected question and the number of pupils who chose each option. Hovering with the mouse over each bar shows the list of pupils who selected the answer represented by this column.

\item Class Steps. This graph shows information about the time spent at a chosen step of an example by the pupils of a class. It is in the form of a column chart where each column represents a pupil's attempt. This means that there might be more than one column for each pupil if they have attempted the selected step more than once- each column revealing information about the time spent by that pupil at a particular attempt of the step.

\item Class Summary. This is a table showing summary information about the total time spent by each pupil at a particular example, how many times they returned to previous steps and the last step they reached. This would show the teacher how much effort did the pupil put in each example, how many problematic or unclear steps they encountered as well as whether they completed the example or which step they gave up on. This is ideal as a quick overview of a class's progress, highlighting problematic pupils.

\end{itemize}

\section{Architecture}

The architecture of WEAVE consists of three distinct tiers as visualised on Figure~\ref{architecture}.

\begin{itemize}

\item Presentation tier. This is the top level of the overall architecture also known as the client side web interface. It defines the appearance of the website by rendering HTML and CSS and provides means for users to interact with the application. The clients are in the form of web browsers. On every interaction, they send requests to the server in the form of HTTP GET or POST requests.

\item Django Middleware. This tier consists of two distinct components.

\begin{itemize}

\item The first component of this tier serves as a communication point between the client and the database. In this tier the requests from the client are parsed and translated into \textit{ORM (Object Relational Model)} requests- a language understandable by the database. These requests are passed forward to get or store the information in the request from/in the database. After the backend generates an ORM response, the middleware is responsible for translating it into an HTTP response and passing it back to the presentation tier.

\item The second component is the connection point between IWE and WEAVE. This is where the translation of the XML elements takes place, so that the examples can be stored in the form of ORM objects. This translation is done with an XML parser. The relationships between these elements are established and the database is populated with the objects defined by the parser.

\end{itemize}

\item Data layer. This tier represents the database in which all the information used or generated by WEAVE is stored in the form of database objects (typically known as rows in a relational database). On GET requests the backend tier responds with an object meeting the criteria specified in the ORM request. On POST requests, the database creates a new object with the features specified in the request and stores it in the database.

\end{itemize}

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/architecture.jpg}

\vspace{-30mm}

\caption{The N-tier architecture of WEAVE.}

\label{architecture}

\end{figure}

\section{User Interface}

As already mentioned, WEAVE is intended for three distinct groups of users, each with their different needs. However, for the purpose of this Level 4 project, only two of these groups will influence the user interface. Addressing the needs of authors is beyond the scope of this project. This is an intermediate step between moving from using entirely IWE to using entirely WEAVE. For this purpose, no author graphical user interface is provided. To create new examples, authors need to use the author interface of IWE. These examples are then added to WEAVE via a command line interface as pointed out in Section~\ref{command\_line}. This is why the section on the user interface is split into two subsections only, which describe the user interfaces for pupils and for teachers.

\subsection{Student User Interface}

A core purpose of this Level 4 project is to translate the student part of IWE into a more easily deployable online version. Careful consideration about the layout of the student interface of IWE is evident. The evaluation of IWE proved that the current interface is well accepted by students. A screenshot of this interface is presented on Figure~\ref{iwe\_student\_interface}. Generally, it has been decided to take advantage of Dr. Song’s findings and to reuse a very similar interface.

\subsubsection{Home Page}

The need for some additional features of this interface arises due to the fact that teachers need to be able to identify their students in order to monitor their interaction with the examples. To ensure that students are using the system in the intended way and to encourage them to use the details provided by their teacher, the examples will be hidden to them until they log in – either by entering some authentication information or by identifying themselves as anonymous users.

Once the student has logged in, the area prompting the user for details is exchanged for the list of worked examples to encourage them to choose one to work on. Due to the fact that the system is required to accommodate examples created by many teachers across the UK, a filtering by the name of the worked examples functionality has been provided.

In order to be able to exploit the worked examples viewer in an optimal way and to familiarise students with it, a tutorial appears on the main page. The idea for having a tutorial was borrowed from IWE. However, the way the tutorial was constructed there was identified as potentially ineffective at communicating all the information the pupil needs to know before working on examples due to the fact that it contains a lot of text which may discourage some of the pupils to read it. Furthermore, even if they read the tutorial, they may not understand what is referred to in the text because they may have not seen the worked examples viewer and its features in advance. A different approach was chosen for the tutorial of WEAVE. It is split into different steps describing an individual feature using minimal text and a screenshot of the feature. Screenshots of the tutorials in the two systems are shown on Figure ~\ref{IWEvsWEAVEStudent}. for comparison.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/IWE\_vs\_WEAVE.jpg}

\vspace{-30mm}

\caption{Screenshots from the student interface of IWE and WEAVE.}

\label{IWEvsWEAVEStudent}

\end{figure}

\subsubsection{Page for Viewing an Example}

The page for viewing a worked example is very similar to the one used for the IWE student interface. Figure~\ref{IWEvsWEAVETutorial} shows screenshots of the student interface for each of the systems to illustrate the changes to the interface discussed below. Due to constraints imposed by the size of the screens in schools, the design of WEAVE needed to be adjusted accordingly. The area for selecting an example (referred to as \textbf{Element 1}) is placed on the navigation bar with all the examples appearing in a drop down menu on request. This saves a significant portion of the screen which can be used for the problem specification instead. Another space consuming element is the bar showing the current step (\textbf{Element 3 }in the IWE student interface). In WEAVE this information is shown as a part of the explanation (\textbf{Element 4}) instead. The elements surrounded with boxes with the same colour and having the same number have an equivalent functionality across the two systems. The reader can remind themselves of the purpose of each area by referring back to Section~\ref{iwe\_student\_interface} in Chapter~\ref{background}.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/IWE\_vs\_WEAVE\_tutorial.jpg}

\vspace{-30mm}

\caption{Screenshots from the tutorial of IWE (left) and WEAVE (right).}

\label{IWEvsWEAVETutorial}

\end{figure}

\subsection{Teacher Interface}

\subsubsection{Non-logged in Teachers}

The purpose of the teacher interface requires the teacher to be logged in. Therefore, on the first visit of the page the teachers are presented with registration and log in areas only.

\subsubsection{Logged in Teachers}

The discussion in the design decisions on the authentication to WEAVE (Section~\ref{authentication}) and the way usage data is presented to the teachers (Section~\ref{graphs}) identifies the main features of the interface for the logged in teacher. Options for all the activities a teacher can undertake via the teacher interface are present on their home page to avoid the need for transitions between different pages and to simplify navigation of the website. The metaphor here is of a control dashboard. The wireframe for the interface can be seen on Figure~\ref{teacher\_main\_wireframe}.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/teacher\_interface\_home\_wireframe.jpg}

\vspace{-30mm}

\caption{A wireframe for the home page of the teacher interface (when the teacher is logged in).}

\label{teacher\_main\_wireframe}

\end{figure}

The main page is split into three areas.

\begin{itemize}

\item Area for registering, updating and deleting a group. These three options are provided in the same area on the screen. When the teacher selects the desired option, the elements for this area change accordingly. For example, when the user wants to create a group, they need to enter the group name and the number of students for that group. On update or deletion of a group, on the other hand, they select the group name from a dropdown list. The list of existing groups is shown to remind which group names are unavailable to this teacher. The textbox for entering the number of students accepts integer input only for error prevention purposes. On the submission of the request to create/update/delete a group, a message confirming the status of the action is show.

\item View Group area enabling the teacher to select a group for which to view the student ids. Again, for simplicity and error prevention, the teacher selects the group via a dropdown list rather than typing its name.

\item View Progress icon which navigates the teacher to the statistics page.

\end{itemize}

\subsubsection{View Group Page}

Teachers are able to see the randomly generated pupil ids for a class in the \texttt{View Group} page. Because managing login details with school pupils is known to the project supervisor to be a problematic issue, this page was designed in close consultation with teachers to ensure the simplest way for them to distribute pupil ids. Furthermore, it is designed to ensure that the anonymity of data gathered by the system is complete.

The pupil ids are provided in a table with empty columns for the pupil name and two identical columns with the pupil ids. There are two expected modes of use:

\begin{enumerate}

\item Teachers print this group sheet and fill in the names of their pupils by hand. This avoids any potential problems with storing identification information in the system. Teachers can then cut one of the columns for the pupil ids, further cutting the column into the individual pupil ids and hand them privately to each pupil. The teacher keeps a paper record of which pupil has which id.

\item The system is designed to enable teachers to copy and paste the pupil id column into a spreadsheet containing the pupil names. This can be printed as before, or projected for pupils to read off.

\end{enumerate}

\subsubsection{View Progress Page}

Most of the design decisions for the different types of graphs are explained in Section~\ref{graphs} above. The wireframe for this page is shown on Figure~\ref{teacher\_progress\_wireframe}. The teacher needs to select the particular group and the type of data they are interested in. If there is no data for that selection or the selection is invalid, an appropriate message appears on the screen. Otherwise, a graph is shown. This graph is downloadable to enable saving the data for progress at different points in time and could be used for comparison by the teacher.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/teacher\_interface\_progress\_wireframe.jpg}

\vspace{-30mm}

\caption{A wireframe for the page showing the progress of students.}

\label{teacher\_progress\_wireframe}

\end{figure}

The following chapter will describe how the design decisions for WEAVE were implemented.

\chapter{Implementation}

This chapter outlines the most important and interesting aspect of the implementation of WEAVE.

\subsection{Technology Choices}

This section describes the considerations taken into account during the process of selecting the technologies for this Level 4 project. It then follows with the list of technologies chosen based on these considerations and the constraints of this project.

\subsubsection{Considerations taken into account}

Due to the size and the nature of this project, the technologies for it were selected carefully based on the following criteria:

\begin{itemize}

\item Scalability. WEAVE is intended to be used in around 400 schools in Scotland by thousands of pupils- and if successful, then internationally. This is why the system being scalable is one of the most important considerations taken into account.

\item Ease of development. This criterion is important due to the constraints of the Level 4 project. It was preferred that the selected technologies allow quick and easy implementation and testing.

\item Maximum coverage of the requirements. The selected technologies should allow the satisfaction of the requirements in an efficient way. They should also allow flexibility for any possible changes.

\item Abstraction of other components. Separating different components is a well-recognised software engineering practice and would improve the maintainability of the code.

\item Sufficient documentation. Learning and improving skills in the chosen technologies is one important benefit of this project so the presence of clear and detailed documentation is desired.

\item Prior experience. Having good knowledge in the technologies used will result in a smaller learning curve and maximum effort could be concentrated in development, testing and improvement of the product.

\end{itemize}

\subsection{Choices}

\subsubsection{The Backend}

The choice for a web-application development framework was of crucial importance for this level 4 project. To facilitate the most suitable choice of framework, the Questions-Options-Criteria design model was used (Maclean, A.; Young, RM.; Bellotti, VME.; Moran, T. (1996), "Questions, Options, and Criteria: Elements of Design Space Analysis", in Moran, T.; Carroll, J., Design Rationale Concepts, Techniques, and Use, Lawrence Erlbaum Associates, pp. 53-106) . Figure 5.1. shows the question, the different options and the criteria considered for this project. Based on the maximum satisfaction of the criteria in Figure 5.1., the Django web framework , written in Python, was selected for implementing the backend component. Firstly Django will allow scalability due to the fact that it supports many database engines and switching between them only requires changing the settings of the project. This project will use the default database engine- SQLite because there will be no need for a more powerful one for the time being. However, as WEAVE evolves and becomes more widely deployed, it will be trivial to change to a more powerful engine. Secondly, Django allows rapid application development (RAD). It has support for many libraries which ensures that all the requirements for this project can be met. Django imposes separation of concerns which is discussed in more detail later in this chapter. Both Python and Django are very well-documented and developing web based application using them has turned into an enjoyable activity thanks to the award-nominated beginner’s guide to web development with Python and Django “How to Tango with Django”( http://www.tangowithdjango.com/) available online. Furthermore, I have a sufficient prior experience coming from studying Python in the first year at the University of Glasgow and the Django framework in the Distributed Information Management 3. Finally, Django is open-source so there was no need to pay for the software used to develop this project.

\subsubsection{The Web Interface}

There are two groups of technologies for the web interface- for the server and the client side.

\textbf{Server Side}

The Django web framework has a sufficient support for serving clients’ requests. This is achieved via direct communication with the database. Django allows clear separation between presentation and business logic due to the usage of the model-view-template software design pattern which guarantees better maintainability and readability of the code. This pattern is described in a greater detail in Section 5.2.1.

\textbf{Client Side}

An important role in the selection of technologies for the client side of the web interface plays the compatibility with the browsers which are expected to be used in schools, mainly Internet Explorer, Google Chrome and Mozilla Firefox. Typically, pupils in schools are constrained to use the browsers that are installed on the school machines and acceptable appearance and behaviour is crucial.

\textbf{HTML5}. This is the markup language chosen for the generation of the web pages. It is preferred to other markup languages due to the multiple benefits it has. Firstly, HTML5 provides an easy access to contents and elements which helps for design and debugging purposes. Secondly, it allows for writing of cleaner code where style and content are separated. Last but not least, HTML5 supports excellent cross-browser compatibility. HTML5 was preferred over XHTML(http://xhtml.com/), which was the other option under consideration, because HTML5 offers more flexibility.

\textbf{CSS}. Cascading Style Sheets (CSS) is used for defining the appearance of the web pages. It enforces separation of concerns between the HTML elements and their presentation. Furthermore, the use of CSS allows the control and flexibility over the appearance of different elements and results in a cleaner HTML code. An alternative considered while making this choice was Less (http://lesscss.org/). It adds features to the CSS language and makes it more maintainable. However, it would have involved some initial learning overhead and due to the time constraints of this project it was decided to focus the predominance of the time on other parts of the implementation instead.

JavaScript. Being free, open-source and supported by the majority of browsers, this scripting language is used to deal with the interactions of the client and the webpage. Alternatives for this choice of technology were not considered due to the simplicity and powerfulness of this language for its intended purpose.

\textbf{JQuery}. Having the same benefits as JavaScript, JQuery is an excellent solution for simplifying client-side scripting. It is used for defining the behaviour of different components on user interaction with the website. It allows easy manipulation of HTML elements and removes cross-browser issues. Furthermore, it has a very large community and it is easy to understand, as opposed to the alternatives considered, which include Kendo UI (http://www.telerik.com/kendo-ui) and Wijmo (http://wijmo.com/).

\textbf{Ajax}. Asynchronous JavaScript and XML (AJAX) is used for sending asynchronous requests to the server side avoiding the need for reloading the web page. This reduces the network overhead and the behaviour of the application “feels” closer to a desktop one. The alternative to using AJAX would have been to load pages synchronously. However, this would make the interaction cumbersome and slower.

\textbf{Bootstrap}. This is the most popular framework for developing responsive design of web applications. This is a combination of HTML, CSS, and Javascript code that is designed to facilitate building of user interface components. WEAVE is intended for school computers and the screen sizes may vary across schools. Furthermore, it is not guaranteed that students will use full screen size at all times of interaction. Due to the nature of WEAVE, students may need to open a different window with more information needed to enable them to solve a problem. These needs lead to the responsiveness of the application being crucial. An alternative considered was PureCSS(purecss.io) which provides a set of small, responsive CSS modules that would load faster. However, Bootstrap was preferred because of the larger community and the greater number of features it offers.

\textbf{Font Awesome}. This is an open source library used to simplify the user interface through the use of familiar icons for visualising possible means of interactions with the webpage.An example icon used across all the pages is the Home icon which users of websites are familiar with as a means to navigate to the home page. There are many open source libraries which could have been chosen for this purpose. These include Entypo (http://www.entypo.com/) and Typicons (typicons.com). Font Awesome was chosen on the grounds of providing better looking icons.

\textbf{HighCharts}. This free library is chosen because it allows the easy creation and control of interactive charts which are needed for the teacher interface of WEAVE. It has been chosen mainly because it provides many pre-built graphs, such as the line graphs and column charts needed for viewing the progress of pupils. Another benefit is that the library is compatible with older browsers like Internet Explorer 6. This is needed because in schools across the UK older browsers are used predominantly. A library which would bring more powerful features would be D3(http://d3js.org/). However, it was not chosen because it does not work well with older browsers.

\section{Connecting the Different Tiers}

As described in Section 4.4., the architecture of WEAVE involves three tiers- the presentation tier, the Django middleware and the Data tier. The Django middleware is split into two distinct sub-tiers- one to serve the communication between the client and the database, and one to deal with the imports of worked examples created by the author interface of the old system. For the purpose of this chapter, this architecture model will be split into two parts. The first part will represent the components in the purple area shown on Figure~\ref{architecture}. These components realise the communication between the client and the server. The second part will represent the components responsible for the transformation of the worked examples created by IWE into database objects. These components are coloured in green on the same figure.

\subsection{Client-Server Communication}

The different components responsible for the Client-Server communication are glued together via the Django variation of the \textit{Model-View-Controller (MVC)} design pattern-\textit{ Model-Template-View (MTV)}.

\subsubsection{Model}

The model represents the data stored in the system. There are two types of data objects- one for the worked examples and one for the usage data.

The objects storing the worked examples are strongly influenced by the structure of the XML files generated by the IWE author interface for holding these examples. Here, however, the relationships between objects are imposed by the use of foreign keys. For example, in the Documents.xml file, fragments are defined by their id and by the text of the fragment. Processes.xml file defines the steps for an example again by specifying both the id and the text of fragments. In this implementation, instead of the Document entity and the Step entity to be associated with a fragment via CharFields for the fragment id and the text of a fragment, fragments are defined as separate entities and documents and processes refer to them via foreign keys. This contributes to a much easier and reliable modification of worked examples due to the fact that if a change needs to be made to a fragment, this change will need to be made in one place only- the fragment object.

The objects storing the usage data for the worked examples are explained in detail in Section 5.3.

\subsubsection{Template}

Templates describe how the data is presented on screen. They are equivalent to the view in the well-known MVC design pattern. Each template is an HTML file defining the different elements to be rendered on screen. The style of these elements is defined via CSS. Depending on the user interaction with each of the interfaces, elements can be destroyed, hidden, created or modified accordingly using JQuery. Such calls are predominant in the teacher interface. For example, if the teacher wants to see the answers for a question in a particular example, a dropdown with the relevant questions appears upon selection of the worked example. If they wanted to see the total time a pupil worked on a particular example, the dropdown list of questions would be exchanged with a dropdown list of pupils belonging to the selected group.

The templates define alternative elements to be rendered depending on whether the user is logged in and if they are- based on their previous interactions with the system. There are different elements that can be rendered depending on previous interactions with the system. For example, the home page of the teacher interface shows register/log in sections when the teacher has not logged in, and if they have- areas to register/update/delete a group, view pupil list for a group and view statistics options. This is done via a template language condition:

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/template\_pseudocode.jpg}

\vspace{-30mm}

\label{template\_pseudocode}

\end{figure}

Similar template language is used to iterate over the list of examples which are passed to the context when the main page of the student interface is rendered. More details on how the list of students is passed are provided in the description of the HTTP requests below.

There are elements occurring in all templates of the interfaces. For example, the navigation bar on the top of each page. To improve maintainability and to avoid too much repetition of the same code over and over again, the reoccurring content is extracted into a base template from which all other templates inherit.

\subsubsection{View}

Each view plays the role of the Python callback function for a particular URL. Views are the equivalent of the Controller in the MVC design pattern. Different parameters may be passed via the request made by the client. There are two types of requests:

\begin{itemize}

\item • HTTP requests to render a page. An example of such request would be when the pupil clicks on a worked example to work on- the client will send a request for the page for this worked example. Each request has its context, containing information such as the client’s machine, etc. Many pages require some information to be passed upon rendering. For example, when a pupil authenticates themselves, the list of the existing worked examples must be rendered on screen. The view responsible for serving this URL knows that this page requires this list, so it will send an ORM request to the database for all examples. It will add these examples to the context dictionary and respond to the client’s request by rendering the template for the requested URL and passing the examples via the context dictionary.

\item AJAX requests. These requests do not render another page but are used to get information from/sent information to the database. There are two types of AJAX requests served by the views:

\begin{itemize}

\item GET requests. In these requests, the client asks for information from the database. Such requests are used mostly in the teacher interface when the teacher views the usage data of their pupils. The requested information is passed in the form of a dictionary and the responsible view gets the required variables using the keys of this dictionary.

\item POST requests. These are requests to store some information in the database. Logging of usage data for the worked examples, such as the time spent on a particular step, is done via these requests. The response typically indicates whether the data was stored in the database successfully. More details how logging of data is implemented follows in Section 5.3.

\end{itemize}

\end{itemize}

\subsection{Translation of the XML Elements into Database Objects}\label{population}

The translation of the XML elements into database objects is done via a population script which uses an xml element tree parser. Each element in this tree is represented by its tag and a dictionary of its attributes. There are two possibilities for the attributes of the element. They can either be attributes of the database object for the respective element, or references to other database objects which will be used to create a foreign key relationship between these objects. This parser uses the depth first search algorithm to traverse the elements in the XML files. The pseudocode for translating the \texttt{Processes.xml} file (in IWE, each worked example is known as a 'process') into the respective database objects is shown below. Please note that in this pseudocode application stands for worked example. This is the terminology of IWE.

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/pseudocode.jpg}

\vspace{-30mm}

\label{population\_pseudocode}

\end{figure}

Please note that for simplicity reasons this preudocode describes the process for adding changes which involve showing/hiding/highlighting/unhighlighting of fragments and the presence of an explanation. It does not include the creation of changes which involve questions. However, these changes, as well as the rest of the XML files, are translated following the same logic.

From this pseudocode, it becomes clear that only relevant elements are stored in the database. For example, no objects were created for a process. Processes become redundant in this database organisation and they can be unambiguously represented via the Step objects instead.

Creating and updating objects are done via the Django get\textunderscore or\textunderscore create method and passing the primary key for this object. This call retrieves either a newly created object or an already existing object with the specified value for the primary key. When this object is retrieved from the database, its attributes are set to those coming from the attributes dictionary of the XML element. This ensures the correct creation or modification of the object and is the reason for the destructive update model of WEAVE.

\section{Logging of Usage Data}

Usage data is logged on every step transition. The data for each attempt of an example step is represented by a database object. There are two types of such objects to accommodate the two different types of steps- steps which involve showing/hiding/highlighting/unhighlighting of fragments of a document and showing an explanation, and steps which involve a question. The respective objects are UsageRecord and QuestionRecord.

On change of the step, an AJAX post request is done invoking the relevant method in the views depending on the type of the step. This POST request passes a dictionary with the necessary information for the data record:

\begin{itemize}

\item the example name, the step number, the time spent on this step and the direction for the transition for the step. This is the data sent for steps which involve changes of fragments and an explanation and is stored in a UsageRecord object.

\end{itemize}

Or:

\begin{itemize}

\item the example name, the step number, the time spent on this step, the answer for the question and an indication whether this was a multiple choice question. This is the information sent for steps which involve a question in a QuestionRecord object.

\end{itemize}

To handle this request, the relevant view extracts all the usage data using the relevant keys of the dictionary passed by the AJAX call. However, this dictionary does not pass any information about the user this data is coming from. This information is accessed via session variables instead. When the pupils specify their details, these are stored as session variables. For example, to store the pupil ID as a session variable when the pupil specifies their details, the view serving the AJAX call will include the line:

\texttt{request.session['pupil'] = pupil\textunderscore id}

When the view for handling logging of data is invoked, it checks the session variables for the teacher id, group id and pupil id .This is done via a simple call such as:

\texttt{pupil\textunderscore id=request.session.get(‘pupil’, None)}

None is the default value to return if no such session variable exists. After checking the session variables for the teacher id, group id and pupil id, the method adds the ones that were present to the data record and saves the record in the database.

\section{Resizing of the Worked Examples Panels and the Explanation Area}

A major implementation challenge for this Level 4 project was to satisfy the requirement to make the panels for the text of the worked examples, as well as the explanation panel, resizable. Difficulties arose due to two requirements for the system:

\begin{itemize}

\item The system must support worked examples which contain any number of panels for worked example text (not only one or two).

\item Both horizontal and vertical resizing were required.

\end{itemize}

The JQuery resizable plugin provides the option to specify that an element is resizable. When specifying this, it can also be specified that when the resizable element is resized, other selected elements are resized as well. This can be done by using the alsoResize option supported for the resizable elements. The code for making an element with id panel1 resizable and to specify that an element with id panel2 needs to be resized as well on resizing of panel1 would be as follows:

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/alsoResize.jpg}

\vspace{-30mm}

\label{alsoResize}

\end{figure}

However, this would mean that when making one panel wider, the other panel also becomes wider. By small addition to the plugin code, a similar option to resize the second element in the opposite direction could be added- alsoResizeReverse. To implement this option, it would have been required to change the sign in calculations done by the respective function. The above code would then look like this:

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/alsoResizeReverse.jpg}

\vspace{-30mm}

\label{alsoResizeReverse}

\end{figure}

The following subsections will describe the problems using this approach.

\subsection{Vertical Resizing of Worked Examples Panels}

This solution above could work for resizing the panels horizontally if there were two panels for the examples only. Using it for more than two panels would behave in an undesired and unexpected way due to too much alsoResizeReverse dependency between the panels.

Another JQuery plugin which was found to behave better in this situation is the colResizable plugin. It treats the area of the worked examples as an HTML table where the panels are different columns which can be resized. This plugin was used for the vertical resizing.

\subsection{Horizontal Resizing of Worked Examples Panel}

The resizable JQuery plugin was ideal for the horizontal resizing of the panels. However, it was incompatible with the colResizable plugin used for the vertical resizing of the panels. This is why, the height of the panels and the explanation are recalculated and set manually on each resize event.

This is the code:

%\vspace{-7mm}

\begin{figure}

\centering

\includegraphics[height=9.2cm,width=13.2cm]{images/horizontal\_resizing.jpg}

\vspace{-30mm}

\label{horizontal\_resizing}

\end{figure}

What this code does is when the user is finished with the resizing of the -explanation panel, is sets the height of the worked examples panels to be equal to the difference between the total height of the interface and the height of the explanation panel.

\section{Showing the Text of the Worked Examples in the Panels}

As described in Chapter 2 and Chapter 4, the documents containing the text of a worked example are split into many fragments. These documents are shown in separate panels on the viewer. These panels are in the form of HTML div elements. It is often the case that more than one fragment need to appear on one line in these div elements. This is why the fragments needed to be represented by inline elements rather than block elements in HTML. Inline elements allow multiple elements to be placed next to each other, while block elements are placed below each other. Span is the inline element chosen for representing each fragment. Please note that any inline element could have been chosen without observing any differences in the behaviour.

Due to the constraint that some fragments need to be inline with other fragments, a slightly undesired behaviour of the fragments arose when they do not fit in the width of the panel for a particular document. Words of these fragments can be split between two lines in unexpected parts. However, this is a limitation due to the use of HTML elements and their behaviour.

This chapter described the implementation of the major components of WEAVE. In the next chapter, the reader’s attention will be drawn to the evaluation of the system.

%%%%%%%%%%%%%%%%

% %

% APPENDICES %

% %

%%%%%%%%%%%%%%%%

\begin{appendices}

\chapter{Running the Programs}

An example of running from the command line is as follows:

\begin{verbatim}

> java MaxClique BBMC1 brock200\_1.clq 14400

\end{verbatim}

This will apply $BBMC$ with $style = 1$ to the first brock200 DIMACS instance allowing 14400 seconds of cpu time.

\chapter{Generating Random Graphs}

\label{sec:randomGraph}

We generate Erd\'{o}s-R\"{e}nyi random graphs $G(n,p)$ where $n$ is the number of vertices and

each edge is included in the graph with probability $p$ independent from every other edge. It produces

a random graph in DIMACS format with vertices numbered 1 to $n$ inclusive. It can be run from the command line as follows to produce

a clq file

\begin{verbatim}

> java RandomGraph 100 0.9 > 100-90-00.clq

\end{verbatim}

\end{appendices}

%%%%%%%%%%%%%%%%%%%%

% BIBLIOGRAPHY %

%%%%%%%%%%%%%%%%%%%%

\bibliographystyle{plain}

\bibliography{bib}

\end{document}