VRIJE UNIVERSITEIT

Advanced database integration in Ximpel interactive video

J.Heymans

22-Jun-11

Master thesis
Computer Science, Software Engineering
Faculty of Sciences
Vrije Universiteit van Amsterdam

Supervisor: prof. dr. Anton Eliëns

Contents

Abstract

Ximpel is an interactive video platform which provides users with the option to influence the course of the video. Ximpel also provides a scoring system so it can also be used as a visual game platform. My thesis will focus on expanding the possibilities of Ximpel by researching if it is possible for Ximpel to save data, which then can be accessible from outside the Ximpel platform. During this research I will also implement the discovered possibilities and document this process.

“What we have to learn to do, we learn by doing”
- Aristotle -

1. Introduction

The broader aim of this thesis besides the one described in the problem description, is ultimately to encourage the readers to think about some of the different methods of application that are achievable with Ximpel, but have not yet been fully explored or looked at. I end with a number of conclusions and recommendations for the future group of users.

1.1. Problem Description

A Ximpel video can provide users with a set of interactive possibilities which include:

- The answering of questions. A correct answer can optionally be worth a number of points and at the end of the video, the total score can be displayed.
- Choices in the path that the video will take. A traditional video is strictly linear, but in a Ximpel video, the user can have a choice in what part of the video he watches.

In its current form Ximpel does not have the ability to save data, so the choices a user makes, or the answers a user has given to questions is lost once the video ends. The lack of this ability prevents Ximpel from being used for any purpose where the recording of user decisions and interactive data is necessary.

For my Master's thesis I will research if it is possible for Ximpel to save data which can then be accessible even after the video has ended, and how this can be achieved. The preferred method for saving data would be to write the information into a database. I have formulated the following main questions which I will attempt to answer at the end of my project:

- Can database functionality be added to Ximpel?
- What is the benefit of adding database functionality to Ximpel?
- What database is best suited and why?
- How can the project be organized?

A secondary goal of my Master's project is to provide functionality that enables Ximpel to query a database for certain variables by using a custom media type designed for this. Besides providing a specific example of how this can be done, I also need to provide a generic solution which can be implemented in future versions of Ximpel. This solution must make it possible for mini-games and other flash content to make use of the database feature Ximpel provides without modifying the Ximpel application.

Besides answering the questions above, this thesis is meant to serve as a manual for further extending database functionalities in Ximpel.

1.2. Constraints

During the project I have started to develop a custom Content Management System which makes it easier to manage the information that Ximpel will write to the database. This will not be covered in my thesis because it falls beyond the scope of adding database integration to the Ximpel platform. When the CMS is finalized it will be included in the source code of the entire project, but as it will simply be a couple of PHP functions which perform SELECT queries on the database it will not be explained further in the thesis.

1.3. Definitions and Acronyms

User: The person that is playing the Ximpel video and interacting with it.

Video provider/Admin: The person that presents the Ximpel video to user and has determined which information will be logged in the database.

PHP: A general-purpose scripting language originally designed for web development to produce dynamic web pages. For this purpose, PHP code is embedded into the HTML source document and interpreted by a web server with a PHP processor module, which generates the web page document

QOC diagram - QOC means Question (A question that deals with a concern), Options (The different options I have in approaching the given concern) and Criteria (The Criteria that I have to take in to consideration when finding an answer to the question).

2. The Ximpel platform

2.1. Background

Ximpel stands for the eXtensible Interactive Media Player for Entertainment and Learning.
It was designed and developed by:

- Winoe Bhikharie, MSc (developer)
- Hugo Huurdeman, MSc (developer/designer)
- Marek van de Watering, MSc (designer)
- Anton Eliëns, prof. dr. (supervisor)

Ximpel was originally developed for the Clima Futura climate game and has also been used in education for creating short viral videos which allow for further interactive explorations.
When watching a Ximpel interactive video the viewer can be presented with quiz questions which have a true/false answer and branch questions. Branch questions require the user to make a choice by clicking on one of the overlays currently displayed on the screen. These overlays can be seen as possible answers to the branch questions. Clicking on one of these overlays leads to the playback of videos that are relevant for the chosen answer. It is also possible to add multiple-choice questions, which have the same functionality as the true/false questions.

Ximpel's goal is to provide an open multimedia platform which can be used for both entertainment and education. The features that Ximpel offers are:

- Customizable, clickable overlays and visuals.
  These can be used to access different branches of the storyline and can be linked to both external and internal information sources.
- Customizable questions.
- A scoring mechanism to give a certain weight to the answers given.

The variables, such as the clips to be shown, the branches, overlays, questions and score points are all modifiable through a collection of 2 XML configuration files *[1][2]*. The Ximpel player then reads the information stored in these files.
The 2 XML configuration files are:

- an playlist.xml
- ximpelConfig.xml

2.2. Flex

Ximpel was developed by using the open source Adobe Flex framework. It was originally released in March 2004 by Macromedia which was purchased by Adobe in 2005.
The Flex SDK comes with a set of user interface components including buttons, list boxes, trees, data grids, several text controls, and various layout containers. Charts and graphs are available as an add-on. Other features like web services, drag and drop, modal dialogs, animation effects, application states, form validation, and other interactions round out the application framework. Interactivity in the Flex framework is achieved through the use of Actionscript.

For Ximpel to communicate with a database, modifications will have to be made to the source code. While researching in which ways Actionscript can communicate with a database I discovered that it actually cannot do this.

With the current release of the Flex SDK, the Actionscript can only communicate with a SQLite database if the entire project is designed as an Adobe Air application *[8]*.
An Adobe Air application is a standalone desktop application which cannot be deployed on the internet.

While searching for a way to communicate with the database in another way I discovered that Actionscript does offer a HTTP service which can perform both HTTP POST and HTTP GET requests. This would be sufficient to get information out of Ximpel by posting the data to a URL. It became clear that a middle-tier would be necessary which can handle the communication with Ximpel and the database.

Because this middle-tier will be accessed by a HTTP POST or GET request, the most logical implementation would be a web service that can communicate with a database. This web service will probably be implemented using a server side scripting language such as PHP, ASP or JSP.

2.3. The benefit of adding database functionality

Knowing that it will probably be possible to add database functionality to Ximpel leads us to question what the benefits can be. Because there is no literature available which covers the benefits of adding database functionality to an interactive video platform, I researched some literature on the function of databases in web 2.0. According to Web 2.0 Architectures [6]:

Databases are typically used to persist data in a centralized repository designed in compliance with a relational model. Other types include hierarchal databases and native XML databases. Each type of database is tasked with handling centralized persistence of data that may be retrieved later for a variety of purposes. Databases vary in size and complexity based on the amount and nature of the data stored.

The obvious benefit of adding a database to Ximpel is that it can be used to store data in a centralized repository. Because Ximpel currently does not save data in any form, it is impossible to determine if a database will be the best way to store data. However, since databases are widely used as data storage resources of websites and business applications it is relatively safe to assume that a database will work well with Ximpel as well.

The benefit of saving information from Ximpel in a database can best be explained by the following example:

Consider an educational video targeted at young children. In the video the children must identify the word spoken by a character onscreen. They identify the word by clicking on the correct overlay which represents the word. This video can help the children to enhance their vocabulary faster and in a fun way.
A simple benefit of adding database functionality in this example would be that teachers can now keep a record of how well each child is doing. They can see how many words are identified correctly and can track the improvement of the children over an amount of time.

Besides the example above, database functionality will provide Ximpel with more possibilities such as:

- Enabling Ximpel for possible commercial use
- Statistical Analysis on the stored data
- Generating a replay of a certain users playthru

3. Project setup

3.1. Software development

There are several different approaches to software development. Some take a more structured, engineering-based approach to developing solutions, whereas others may take a more incremental approach, where software evolves as it is developed piece-by-piece. Most methodologies share some combination of the following stages of software development:

- Market research
- Gathering requirements for the proposed business solution
- Analyzing the problem
- Devising a plan or design for the software-based solution
- Implementation (coding) of the software
- Testing the software
- Deployment
- Maintenance and bug fixing

The combination of these stages is often referred to as the software development lifecycle, or SDLC. Different approaches to software development may carry out these stages in different orders, or devote more or less time to different stages. The level of detail of the documentation produced at each stage of software development may also vary.

Several specific software development models exist to streamline the development process. Some of these are:

- Waterfall model
- Spiral model
- Iterative and incremental development
- Agile development
- Code and fix

Each one has its pros and cons, and it is usually up to the development team to adopt the most appropriate model for the project.

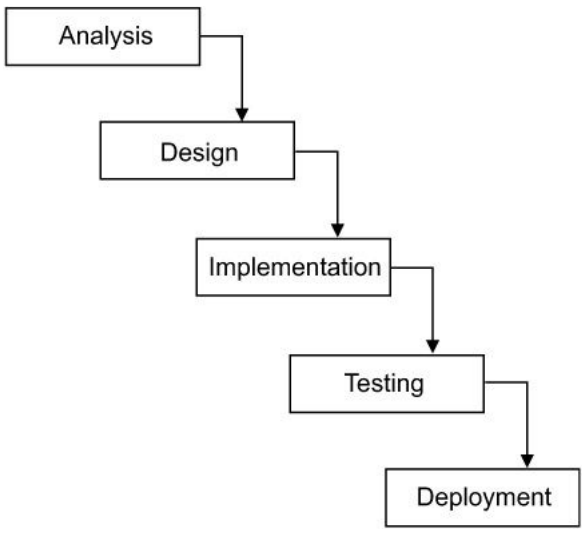
3.2. Waterfall model

Although I am not developing completely new software, I think it is best to approach the project as a software development project. This will provide the structure that is required to successfully complete the project and provide a guide for documenting the relevant phases of the project.
For my project I have chosen to follow the waterfall model.

The waterfall model can be considered the classic approach to software development.
In a pure waterfall model phases are not revisited once they have finished. Before moving on to a next phase a review may occur which allows for the possibility of adding changes to a phase.

Although the waterfall model discourages revisiting and revising any prior phase once it's completed I will not enforce this during the project. My master's project is a relatively small project with me being the only developer, so the revising of previous phases will not have as much impact as it would have in larger projects.

The phases of the waterfall model can be seen in the following model.



My project does not require all of the phases in the waterfall model. The phases which are necessary for my project are:

- The Analysis phase.
  In this phase the requirements of the project are determined.
  This phase is covered in Chapter 4

- The design phase.
  The design decisions are made in this phase. These decisions are based on the requirements of the project. This phase is covered in Chapter 5.

- The implementation phase.
  This is the actual writing of code. In my project this can be regarded as the modifications and additions which are made to the Ximpel source code. This phase is covered in Chapter 6.

The remaining 2 phases will not be formally performed or documented, but testing will be done throughout the project.

4. Project Requirements

4.1. Requirements analysis

Before starting a software development project, it is important to document the need and conditions which the finished product will have to meet. The process that encompasses this task is called Requirements analysis. In most software development projects there will be several stakeholders involved who might each have their own set of requirements for the system which is being produced. The documenting of these requirements can be achieved by performing the following tasks:

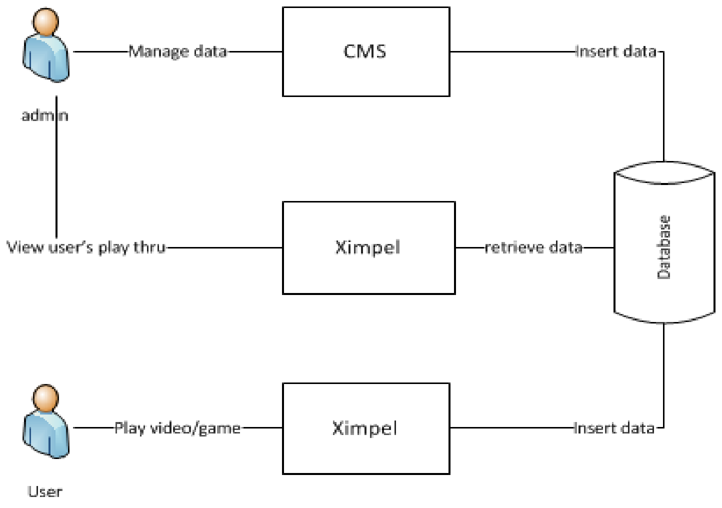
- Requirements elicitation
- Requirements analysis and negotiation
- Requirements specification
- Requirements validation
- Requirements management

After performing these tasks a list of requirements will have been formed and which needs to be managed, because it is possible for requirements to change during the project, however this is usually undesirable. For my master's project the only stakeholder who needed to be elicited was my master's project supervisor. He made known the requirements that were important to him and I combined these with some requirements of my own which I thought would be useful to the end user. The resulting functional and non-functional requirements are described in the chapters 4.2 and 4.3.

4.2. Functional requirements

A functional requirement defines a function of the software system. A function is described as a set of inputs, the behavior, and the outputs. Functional requirements are supported by non-functional requirements (sometimes also known as quality requirements) which impose certain constraints on the design or implementation of the system.

The functional requirements are first shown in the following generic diagram after which they are described in more detail.



Requirement	Requirement Definition
insert_branch_question_text	The Ximpel application inserts the text of the branch question which was presented to the user into the database.
insert_user_given_answer	The Ximpel application inserts the selected answer from the user into the database. This can then be true/false answer, or the chosen answer of a multiple choice question.
insert_question_correct_answer	The Ximpel application inserts the answer which is regarded as the correct answer into the database. This only applies to the true/false and multiple choice questions.
insert_score	At the end of the Ximpel presentation the accumulated score is inserted into the database.
insert_branch_question_choice	The Ximpel application inserts the name of the overlay the user clicks on. This click on an overlay mostly signifies a branching choice.
insert_current_video	The Ximpel application inserts the currently playing video into the database.
insert_userID	The Ximpel application inserts a generated userID in to the database.
generate_replay.xml	The CMS of the system can generate a so called replay.xml of a certain user's playthru.
get_random_video	The Ximpel application can select a random video by using the custom media type.
get_ video	The Ximpel application can select a video which complies with certain criteria by using the custom media type.

4.3. Non-functional requirements

A non-functional requirement is a requirement that specifies certain criteria that are used to judge the operation of a system instead of focusing on specific behaviors *[5]*. A non-functional requirement describes how the system is supposed to be in contrast to a functional requirement which describes what a system is supposed to do. Non-functional requirements are often also referred to as qualities of a system. Non-functional requirements can be divided into 2 main categories:

1. Execution qualities
   these include such qualities as security and usability (which are visible at run time)

2. Evolution qualities
   these include such qualities as testability, maintainability, extensibility and scalability. (which are part of the static structure of the system)

4.3.1. Usability

Usability can be regarded as "the ease of use" of the system. This depends on the targeted user of the system. In my master's project the targeted user for whom usability is important is primarily the administrator. The administrator determines which information is written to or can be extracted from the database. Also the system must be easy to setup and deploy as a whole.

4.3.2. Portability

Portability means that the system should be easy to move from one environment to another. In the case of my master's project the system must be able to function on different platforms such as windows and Linux.

4.3.3. Extensibility

Extensibility of the system means that the system can be easily expanded upon. In the case of my master's project, the modified code must be clear and understandable so it can be used in future modifications or extensions of the system.

4.3.4. Security

The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them.

4.3.5. Modifiability

Modifiability means that the resulting system can be easily modified and the information about modification is easy to find.

5. Project Design

5.1. Design

In this chapter I will describe the various design choices that were made during the project. I will list each choice in a table which contains the considered options, the arguments for each option and the resulting choice. After each "decision table" I will also have a small diagram which visually displays the choice made and shows how it relates to the non-functional requirements.

5.2. Design decisions

Concern#1		***Which database to use?***
Ranking criteria		*Criteria#1: Usability.* *Criteria #2: Portability.* *Criteria #3: Extensibility.*
Options	***Identifier: Name***	*Concern#1-Option# 1 : MySQL*
	Description	*The MySQL database that is used for storing information requires a server to function.*
	Status	*This option is rejected*
	Relationship(s)	*Concern#2*
	Evaluation	*Criteria#1: MySQL is easy to use and has a widespread user base, with lots of information available.* *Criteria#2: MySQL is not very portable because it requires a server to function. This also means that when moving, the database an export and import of the tables will be necessary.* *Criteria#3: Because MySQL is widely used, it will be easy to understand and extend the system that is using this database.*
	Rationale of decision	*This option is rejected because portability is a major requirement for the system.*
	Identifier: Name	*Concern#1-Option#2: SQLite*
	Description	*The SQLite database can be contained in a single file and does not require a server to function.*
	Status	*This option is* ***accepted***

	Relationship(s)	-
	Evaluation	*Criteria#1: SQLite is very lightweight and easy to use, because it does not require a running database server.* *Criteria#2: With SQLite the entire database is contained within a single file which makes it very portable. This database can be easily moved between locations, and a novice user does not have to setup anything in regard to the database for it to function properly.* *Criteria#3: Because SQLite is widely used and supported, it will be easy finding information and will be make it easy to extend as system that uses SQLite.*
	Rationale of decision	*This option is accepted because it satisfies all of the relevant criteria.*

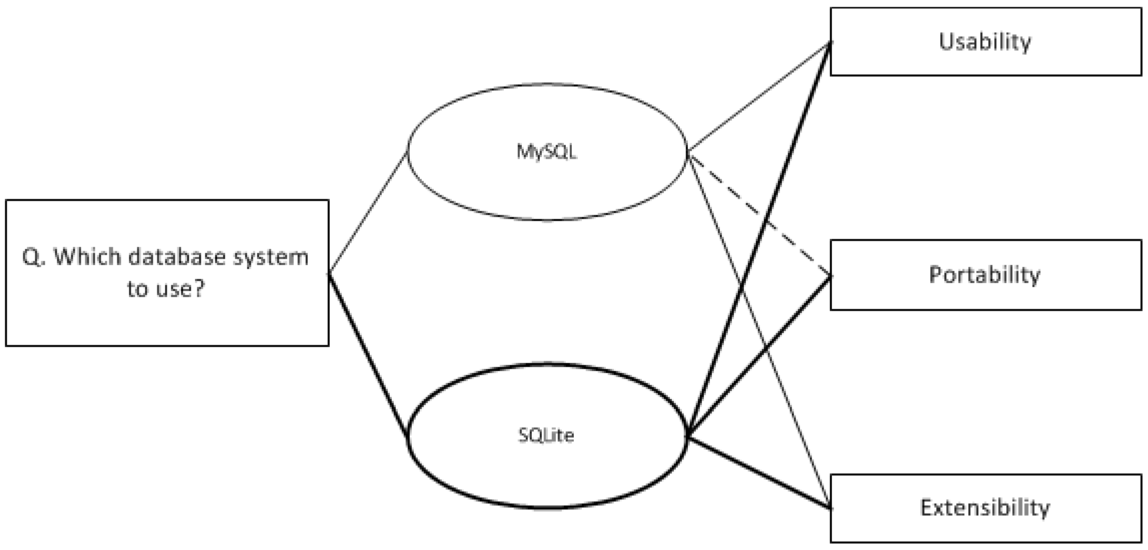


Diagram 1: Which database to use?

Concern#2		***Which language to use for the CMS?***
Ranking criteria		*Criteria #1: Extensibility.* *Criteria #2: Modifiability.* *Criteria #2: Portability.*
Options	***Identifier: Name***	*Concern#2-Option# 1 : PHP*
	Description	*Developing the Content Management System in PHP to manage the information inserted into the database. PHP is a server side embedded programming language which is widely used and supported. PHP is free to use and most webhosting providers offer support for it.*
	Status	*This option is* ***accepted***
	Relationship(s)	*Concern#1 & Concern#3*
	Evaluation	*Criteria #1: By using PHP it will be easy to extend the system, because PHP has a low learning curve and support information is widely available.* *Criteria #2: Because PHP is widely supported it will be easy to modify the code.* *Criteria #3: PHP is provided by most of the webhosting providers and it is very easy to setup your own PHP (apache) server locally, which makes PHP very portable.*
	Rationale of decision	*This option is accepted because it provides a high degree of extensibility and modifiability while still offering portability.*
	Identifier: Name	*Concern#2-Option#2: ASP/JSP*
	Description	*ASP is a programming language created by Microsoft for creating dynamic websites similar to those created with PHP. JSP is a Java technology that allows software developers to create dynamically generated web pages with HTML, XML, or other document types, in response to a Web client request.*
	Status	*This option is rejected*
	Relationship(s)	-
	Evaluation	*Criteria #1: The use of these programming languages will provide a steeper learning curve and will make it harder to extend the resulting system, especially for a novice user.*

		Criteria #2: Modifying the code will also be harder with these languages because of the higher learning curve as well. *Criteria #3: Both ASP and JSP require specific servers to function; these are not as widely offered by webhosting providers as PHP is.*
	Rationale of decision	*This option is rejected because it does not satisfy our ranking criteria.*

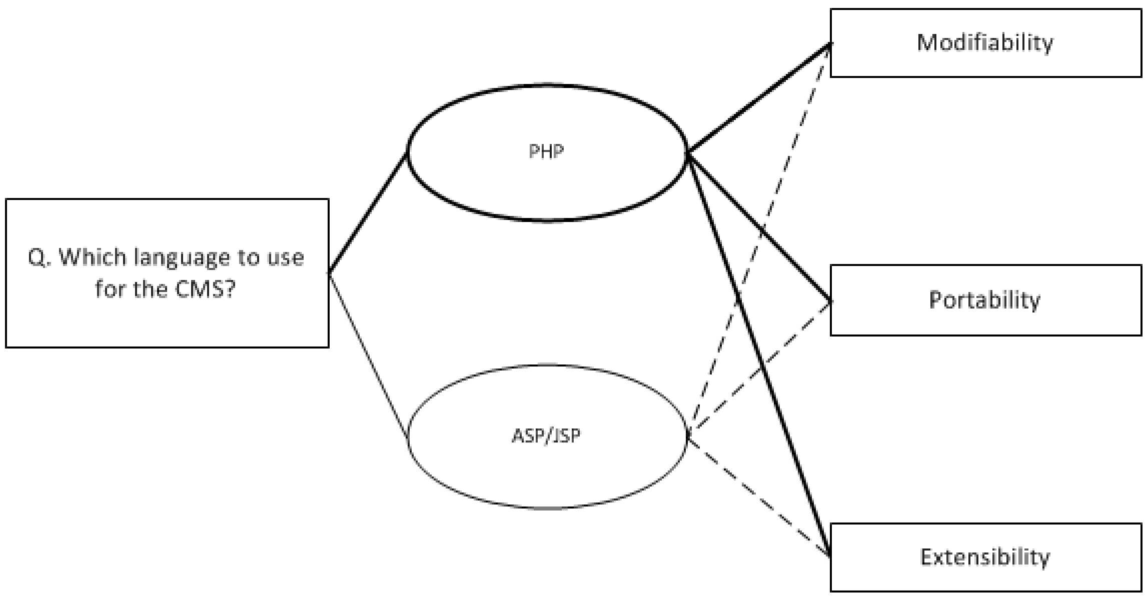


Diagram 2: Which language to use for the CMS?

Concern#3		***How to communicate with the database?***
Ranking criteria		*Criteria #1: Usability.* *Criteria #2: Security.* *Criteria #3: Portability.* *Criteria #4: Extensibility*
Options	***Identifier: Name***	*Concern#3-Option# 1 : Thru a PHP gateway*
	Description	*Communication with the database will be done by first posting the variables to a PHP file which will in turn post the variables in to the relevant tables in the database.*
	Status	*This option is **accepted***
	Relationship(s)	*Concern#1*

	Evaluation	*Criteria #1: PHP is a widely known and supported language which is easy to use, and supporting information will be easy to find.* *Criteria #2: Using PHP to insert data into the database security must be considered. Care must be taken to prevent malicious use such as SQL injection. Also, the file should only be called by the ximpel application itself. Security can be a weak point of this approach.* *Criteria #3: PHP is offered by many web hosting providers and it is very easy to setup at home, so the system remains very portable.* *Criteria #4: By having the database insertion code in a PHP file it will be easier to extend the system down the line.*
	Rationale of decision	*This option is accepted because it satisfies 3 out of 4 of our non-functional requirements.*
	Identifier: Name	*Concern#3-Option#2: Direct connection with SQLite database*
	Description	*Having the Ximpel application itself handle the retrieving/posting of information into and from the database. This option would prevent the Ximpel application from being able to be deployed on the web as it would become more of a desktop application.*
	Status	*This option is rejected*
	Relationship(s)	*-*
	Evaluation	*Criteria #1: The system will remain easy to use with this option.* *Criteria #2: Security will be better with this option, because the database access code will be contained within the Ximpel application itself.* *Criteria #3: The system will be less portable in the sense that it will lose the possibility to be deployed on the web with this option.* *Criteria #4: This option will not satisfy the extensibility requirement, because the source code of XImpel will need to be changed whenever the database structure is altered.*
	Rationale of decision	*This option is rejected because it does not satisfy our extensibility and more importantly it does not satisfy the portability requirement.*

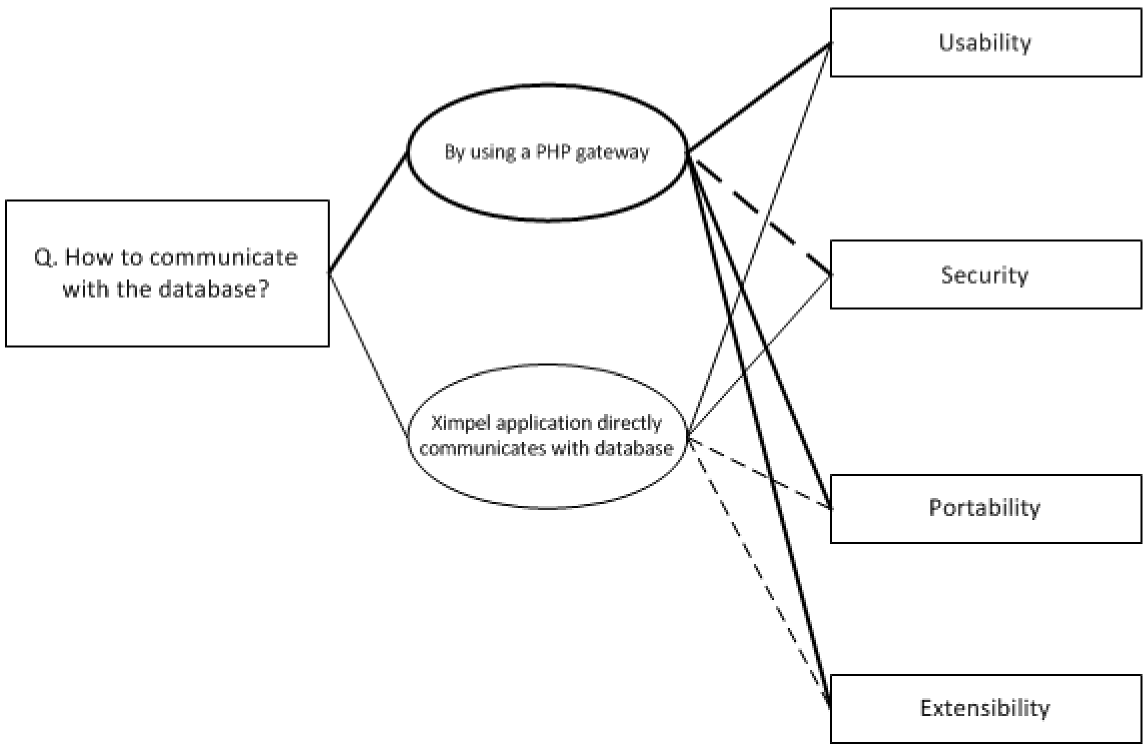

Q. How to communicate with the database?
By using a PHP gateway
Ximpel application directly communicates with database
Usability
Security
Portability
Extensibility

Diagram 3: How to communicate with the database?

5.3. Related concerns

Some of the concerns described in the previous section are related to each other. In this section an overview will be given of the relationships between these concerns.

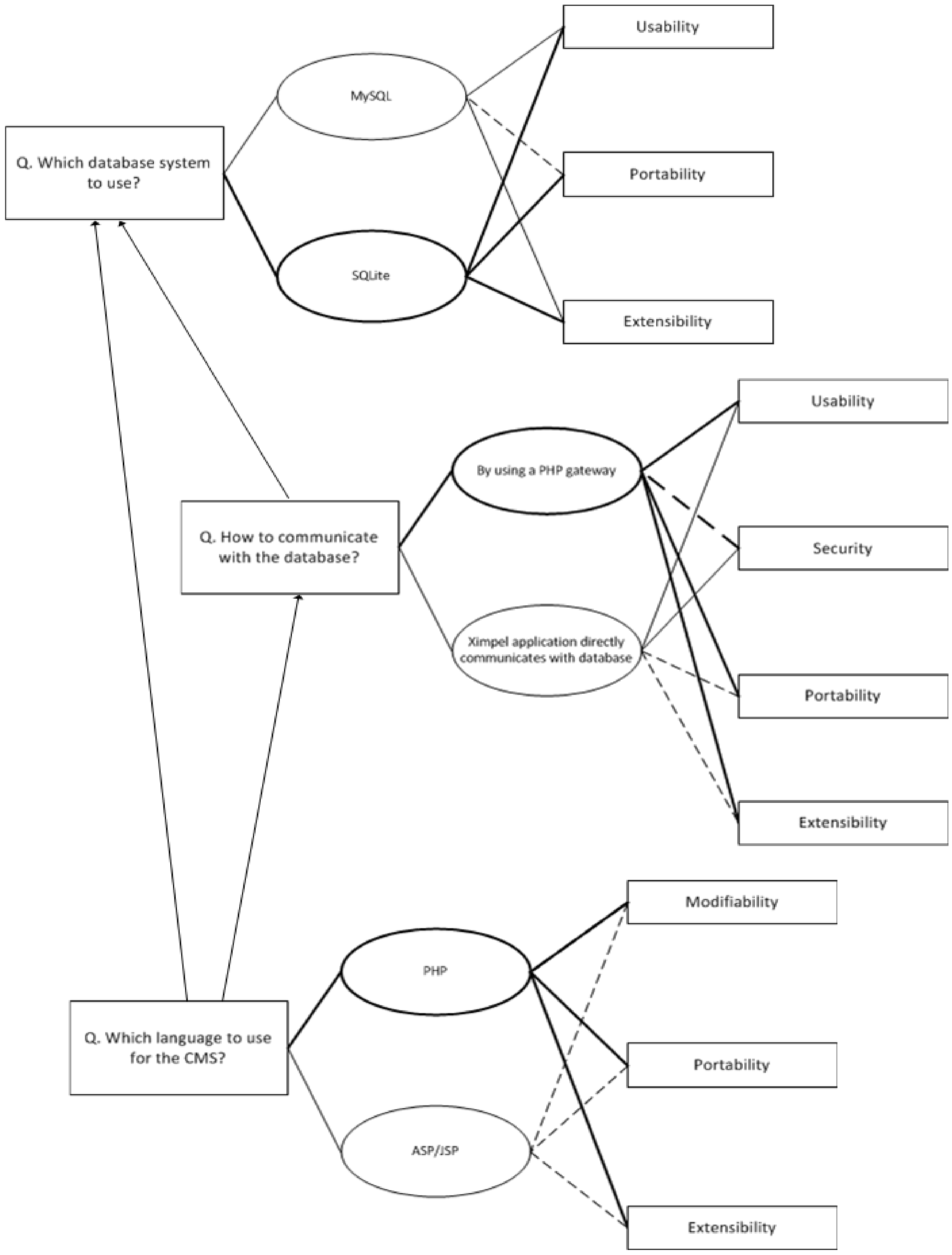

Usability
MySQL
Q. Which database system to use?
Portability
SQLite
Extensibility
Usability
By using a PHP gateway
Q. How to communicate with the database?
Security
Ximpel application directly communicates with database
Portability
Extensibility
Modifiability
PHP
Q. Which language to use for the CMS?
Portability
ASP/JSP
Extensibility

Diagram 4: Relationships between the concerns.

5.4. Database Structure

5.4.1. Storing data

Before structuring the database, I first had to determine exactly which information would have to be stored in it by the Ximpel application. The most relevant information that needs to be stored will be the questions and answers that the user had come across and the total score accumulated by that user. The information that is stored in the database must be detailed enough to be able to construct a "replay" of the specific user playthru. Keeping this in mind, more information than just the question and scoring was necessary.

After experimenting with a few scenarios I brought the necessary variables down to the following:

Questions

- *A unique user id*
- *The name of the video where the question was displayed*
- *The type of question (branch or normal question)*
- *The text of the question*
- *The answer given by the user*
- *The correct answer*

Score

- *A unique user id*
- *The total score*

Video

- *A unique user id*
- *The name of the video currently playing*
- *The ordering of the currently playing video (knowing if it the 1st, 2nd , etc)*

As you can see in the list above a unique user id is present in every table that contains information inserted by the Ximpel application. The unique user id is necessary to be able to bind the different variables to a specific user playthru. Also the name of the currently playing video is necessary because in the replay we need to know which question belongs to which video. The ordering of the video is also needed, because we need to know in which order to display the videos when running a replay.

The required information mentioned above leads to the following database structure:

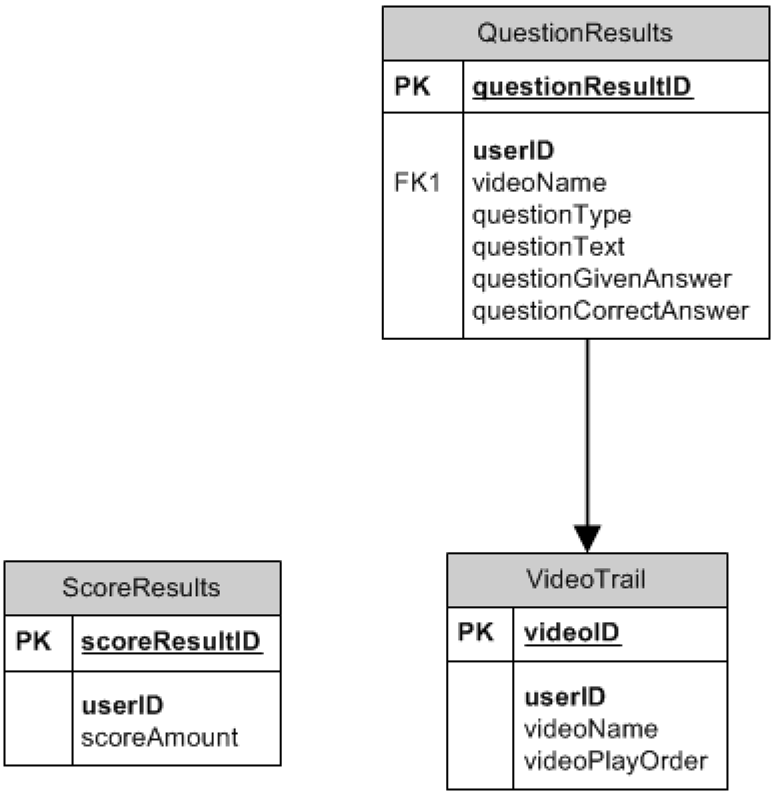


Diagram 5: Database structure of storable data by the ximpel application

5.4.2. Retrieving data

Besides storing data in the database, the Ximpel application will also be able to retrieve data from the database. This will be done by using a custom media type. The information that can be retrieved will mainly be videos. The tables from which the Ximpel application can retrieve its data, were already created during my Bachelor's project, but I will list the database structure here as a reference.
These tables are part of the playlist builder and can currently be used to generate the playlist that Ximpel requires. But as the database can contain many rows in the video table it is possible to query the database for a video file that matches certain criteria. For this the relevant table would need to be expanded with the extra information. In the case of the video table these could be:

- Date
- Topic
- Duration

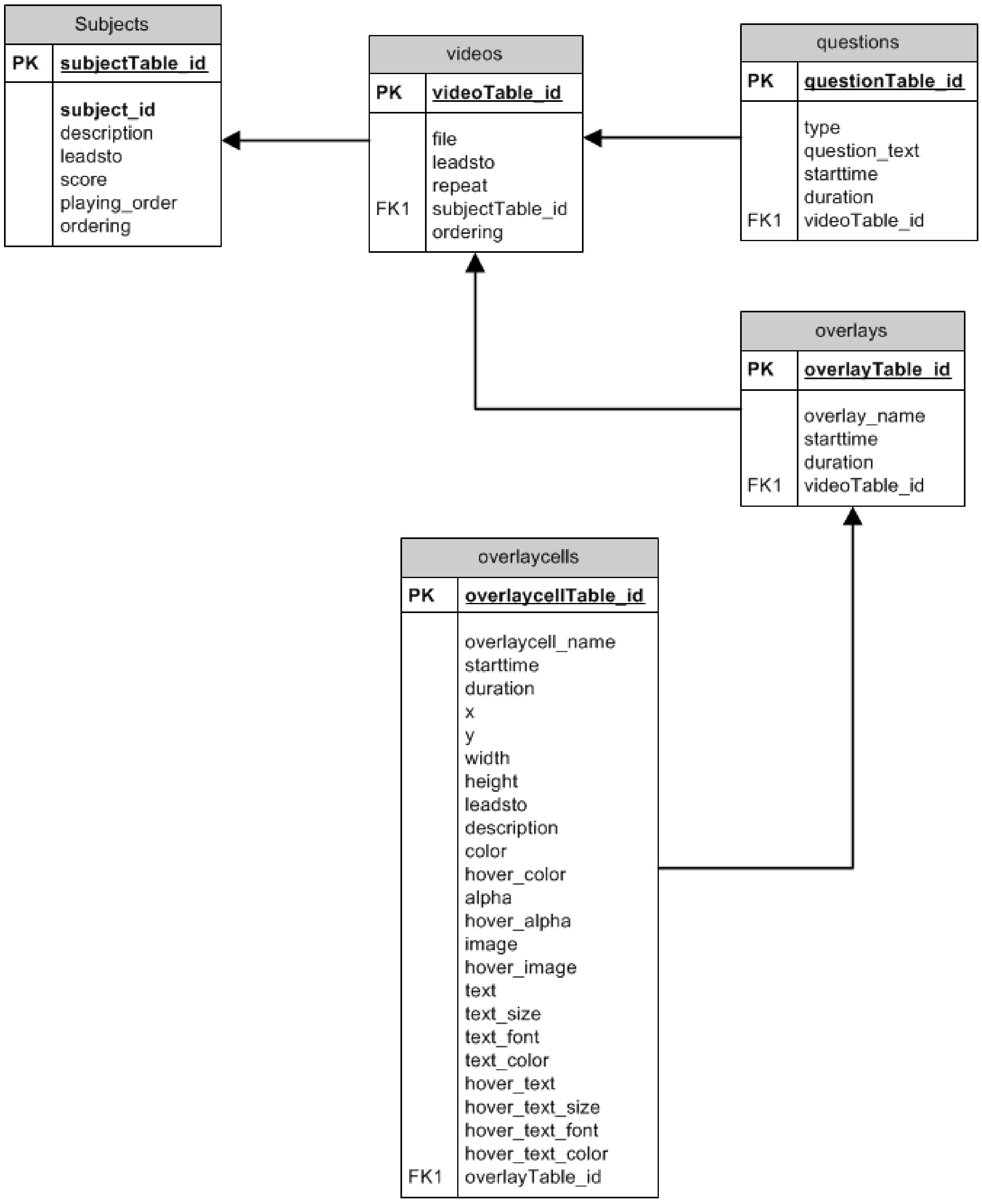


Diagram 6: Database structure of the playlist builder

6. Implementation & Integration with Ximpel

The source files of Ximpel have the .mxml extension. MXML is an XML-based user interface markup language, first introduced by Macromedia in March 2004. It can be compiled into SWF files by using the Adobe Flash Builder IDE (formerly Adobe Flex Builder) or the free Flex SDK. Modifying these files primarily required knowledge of Actionscript.

6.1. Modifications to Ximpel source code

Because I wanted to keep all the original functions of Ximpel intact, I decided from the start to not alter any existing code, but to only make additions. By doing so I tried to keep the robustness of the original Ximpel code and I also made it easier to remove or alter my additions if necessary in the future. An added advantage of this approach is that my additions can also be more easily ported to future versions of Ximpel.

The additions to the code are still very “raw” and there is room for improvement, but they definitely work as expected. The 2 source files which I modified were:

- ximpelApp.mxml
- XimpelPlayer.mxml

6.1.1. Added variables

The variables I added to the Ximpel source code are listed below together with the rationale of why I needed them.

- ***#1: DEFAULT_DATABASE_QUESTION_LOGGING***
  Source file: ***XimpelPlayer.mxml***
  Type: String.

  This is a static variable which contains the value *Questions*: it is used to determine if question logging is enabled.

- ***#2: DEFAULT_DATABASE_Score_LOGGING***
  Source file: ***XimpelPlayer.mxml***
  Type: String.

  This is a static variable which contains the value *Score*: it is used to determine if score logging is enabled.

- ***#3: DEFAULT_DATABASE_Question_AND_SCORE_LOGGING***
  Source file: ***XimpelPlayer.mxml***
  Type: String.

  This is a static variable which contains the value *QuestionsAndScore*: it is used to determine if question and score logging is enabled.

- ***#4: _database***
  Source file: ***XimpelPlayer.mxml***
  Type: ***String***.
  Added to function: **configServiceResultHandler().**

  This is a variable which is populated at runtime with the value defined in the XimpelConfig.xml by the video provider. This value will be compared against the static variables #*1*, #*2* and #*3* to determine which type of database logging is required.

- ***#5: db_questionText***
  Source file: ***XimpelPlayer.mxml***
  Type: ***String***.
  Added to function: **displayExtraQuestion().**

  This variable will be populated with the text of the question currently on screen. It will be written to the database by function #3 whenever the user clicks an answer, thereby calling the ***evaluateAnswer*** function.

- ***#6: db_userGivenAnswer***
  Source file: ***XimpelPlayer.mxml***
  Type: ***String***.
  Added to function: **evaluateAnswer ().**

  This variable will be populated with the answer given by the user to the question currently on screen. It will be written to the database by function #3 whenever the user clicks an answer, thereby calling the ***evaluateAnswer*** function.

- ***#7: db_questionAnswer***
  Source file: ***XimpelPlayer.mxml***
  Type: ***String***.
  Added to function: **evaluateAnswer ().**

  This variable will be populated with the correct answer to the question presented to the user. The ***evaluateAnswer*** function will call function#3 which will write the information into the database.

- ***#8: db_branchQuestionText***
  Source file: ***XimpelPlayer.mxml***
  Type: ***String***.
  Added to function: **checkForBranchQuestion ().**

  This variable contains the value of the branch question currently on screen. It will be written to the database by function #1. When the user clicks on an overlay, the function ***handleClick*** is called which in turn calls function #1.

- ***#9: db_branchQuestionAnswer***
  Source file: ***XimpelPlayer.mxml***
  Type: ***String***.
  Added to function: **handleMouseOver ().**

  This variable contains the chosen answer to the branch question currently on screen. It will be populated when the user mouses over an overlay representing a possible answer to the branch question. It will be written to the database by function #1. When the user clicks on an overlay, the function ***handleClick*** is called which in turn calls function #1.

- ***#10: db_currentVideo***
  Source file: ***XimpelPlayer.mxml***
  Type: ***String***.
  Added to function: **handlePlayMedia().**

  This variable contains the name of the currently playing video. This variable cannot be influenced by the user. It is always written to the database by function #2 to keep track of the different videos that the user has seen.

- ***#11: db_videoPlayOrder***
  Source file: ***XimpelPlayer.mxml***
  Type: Number.
  Added to function: **handlePlayMedia().**

  This variable contains the number of the currently playing video. This variable cannot be influenced by the user. It is always written to the database by function #2 to keep track of the order in which the different videos where displayed to the user.

- ***#12: db_tempVideoName***
  Source file: ***XimpelPlayer.mxml***
  Type: String.
  Added to function: **handlePlayMedia().**
  This variable is used to prevent the system from adding a video to the database when the current video is looping. A video will only be added to the database when it differs from the previously played video.

- ***The total score***
  Source file: x***impelApp.mxml***
  Type: String.
  Function: **goToEvaluationScreen().**
  There was no need to create a new variable or function for this information. The existing gameScoreLabel.text variable was used.

- ***db_userID***
  Source file: **XimpelPlayer.mxml**
  Type: String.

  This variable is populated at runtime with a unique string. A user id has to be unique and is the most important variable because it links all the information of 1 playthru together. Without it, it would be impossible to reconstruct a user's playthru. It is inserted to the database by functions 1, 2 and 3 together with their individual variables.

6.1.2. Added functions

Besides the variables described in the previous section, it was also necessary to add some functions to the Ximpel source code. I will list the added functions below.

- ***function#1: insertBranchQuestion()***
  Source file: x***impelApp.mxml***

  This function is called when the user clicks on an overlay thus indicating that a choice has been made. When a choice is made, the information in the variables 8 & 9 is final, so they can be written to the database. This function contains a hardcoded variable to determine the type of information being posted. The value of this variable is: "***branch"***.

- ***function#2: insertVideoTrail()***
  Source file: x***impelApp.mxml***

  This function is called when a video file starts playing. It then compares variables 10 & 12 with each other. If they are not the same, the name of currently playing video is written to the database. If they are the same, it means that the current video is looping and does not need to be written to the database again. This function contains a hardcoded variable to determine the type of information being posted. The value of this variable is: "***videotrail"***.

- ***function#3: insertQuestion()***
  Source file: x***impelApp.mxml***

  This function is called when the user clicks on an answer to a question. Variables 5, 6, and 7 are then written to the database. This function contains a hardcoded variable to determine the type of information being posted. The value of this variable is: “***question”***.

6.2. Custom media type

For retrieving data from the database a custom media type can be used. The advantage of using a custom media type is that you do not have to modify much of the Ximpel source code itself. The custom media type can be created like any other. As an example I will describe how to create a custom media type that can retrieve a video result from the database. There are many ways that this can be achieved, so the solution that I present is by no means to be considered the best or only way.

In the example below the video provider can define query criteria in the form of an attribute of the custom media type. The custom media type will pass the defined criteria on to the PHP gateway file which will use the criteria to query the database and return the results. What this criteria is and how many criteria are allowed can be customized by the video provider by making changes to the PHP gateway file and to the custom media type. For my example I will assume that 1 criteria is used and that it returns a string containing a video filename.

As we are only querying the database for a video file name that matches certain criteria, we can reuse the StandardVideo media type almost completely to handle the rest of the video playing configuration and settings.

1. First we copy the contents of the StandardVideo media type and paste them into our new media type which we will call **DatabaseVideo**. So the filename will be DatabaseVideo.mxml.

2. We then have to modify the typeName() function so it returns a different type. We will call our type ***dbvideo***. This is the name of the tag in the playlist.xml which will be used to identify our custom media type to Ximpel.

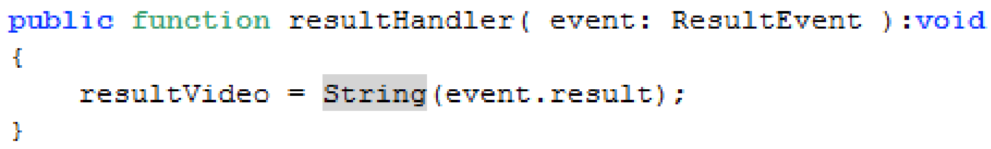
3. Next we have to implement a HTTPService that will allow us to do a call to the PHP gateway which will in turn query the database and return our results. The code for this is:

```
<mx:HTTPService id="getData"
                url="<URL_TO_PHP_GATEWAY>"
                method="POST"
                resultFormat="text"
                result="resultHandler(event)"
                fault='{Alert.show("Cannot reach database insert php")};'
/>
```

Code fragment 1: HTTPService for performing post to PHP gateway

The <URL_TO_PHP_GATEWAY> must be replaced with the valid URL that leads to the PHP gateway file that can retrieve data from the database. The PHP file will contain the actual code that performs the query on the database.

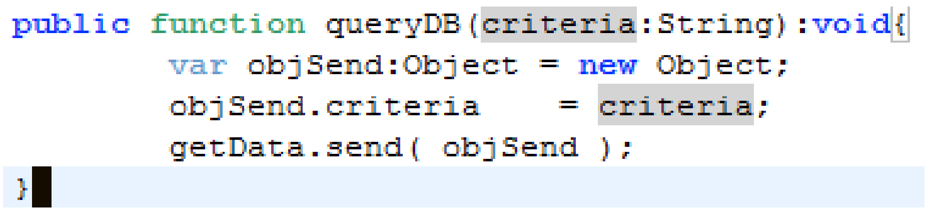
4. We also have to implement a result handler which will handle the results that the PHP gateway will provide us. For this example we assume that the PHP gateway will return a string containing a file name of a video. So we define a global variable called resultVideo and implement the result handler as follows:



```
public function resultHandler( event: ResultEvent ):void
{
     resultVideo = String(event.result);
}
```

Code fragment 2: Resulthandler which will populate global variable with result from query

5. Now we have to create a small function that will perform the call to the PHP gateway using the previously created HTTPService.

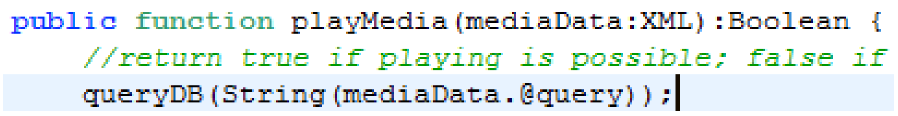


```
public function queryDB(criteria:String):void{
        var objSend:Object = new Object;
        objSend.criteria    = criteria;
        getData.send( objSend );
}
```

Code fragment 3: Function that perform post of criteria variable to PHP gateway file

6. This queryDB function can be called from the ***playMedia()*** function and has to be passed the criteria retrieved from the attribute of custom media type's tag. In this example we have named the attribute "query", which would make the tag in the playlist.xml appear as: **<dbvideo query ="<criteriaString " />**

   The implementation of the function is as follows:



```
public function playMedia(mediaData:XML):Boolean {
     //return true if playing is possible; false if
     queryDB(String(mediaData.@query));
```

Code fragment 4: Call to queryDB function

7. Now our variable resultVideo contains the result of the query performed on the database. This result is a file name of a video. All that is left to do, is to pass this variable to the ***var videoURL:String*** in the ***playMedia ()*** function and the video should play as if it was normally retrieved from the playlist.xml.

As stated before, the example above assumes an error free approach to retrieving results from the database. If for instance the database would return multiple results or no results at all, logic would have to be added to the custom media type and/or the PHP gateway file to handle these scenarios appropriately.

6.3. Generic database interaction design

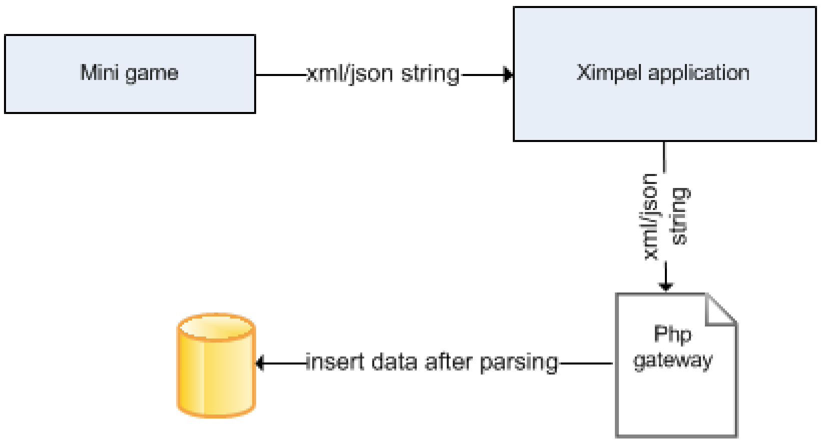
The current database integration of Ximpel is pretty specific about which information is written to the database. Because Ximpel can include other flash media, such as mini games or map applications a generic way of passing information to the database could become necessary in the future. As part of my master's thesis I had to think of a possible generic database integration which would make it possible to write data to the database with as little modifications to the Ximpel source code itself, making it easier for novice users.

I would recommend adding a generic string variable to the ximpelApp.mxml source file, and define a function similar to the function mentioned in section 4.1.2.which would pass this string variable on to a PHP gateway file.

A mini game or other flash application could then place all the variables that it wants to insert into the database in 1 string. For instance by creating JSON or XML string containing all the relevant variables it wants to write to the database. To get this string to Ximpel an event and dispatch can be defined from the mini game. The event contains the string field and can pass this string along from the mini game to Ximpel. The ximpelApp listen for the Event and define a handler that can do the eventual call to a PHP gateway file. The PHP gateway file can then parse the JSON or XML string and process the variables as needed.

The Event and handler can be setup beforehand by developers, so the eventual creator/provider of the mini game would only need to call the events and not need to implement much in the Ximpel source code.

The diagram below illustrates the main idea of passing data in a generic way from a mini game/flash application which runs inside Ximpel to the database.



7. Usage scenarios

Usage of the system can be seen from 2 perspectives. The first is the perspective of the user, which is the person who is actually viewing and interacting with the Ximpel video. The second is the perspective of the video provider/admin which is the person who has defined the playlist and configuration of the Ximpel video and who has determined which data is written to the database. The video provider/admin is also the one that determines what happens with the data after it has been inserted into the database. A portion could be presented to the user in the form of a high scores list for example.

7.1. A user's perspective

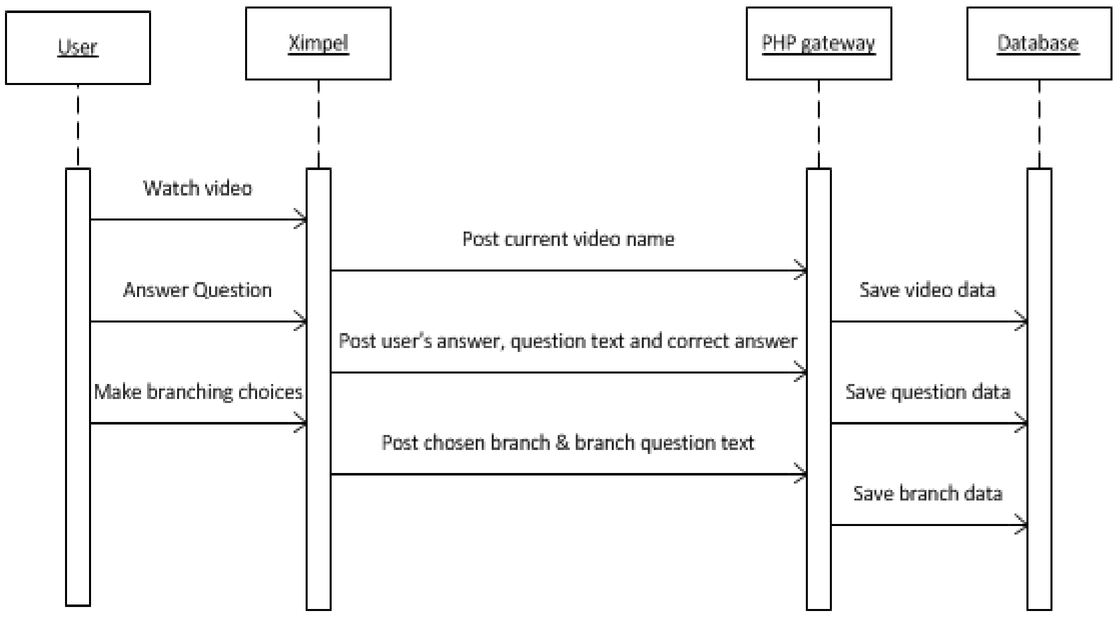
The user makes choices and possibly answers questions while playing the Ximpel video. The user does not have to be aware that his choices and answers are being stored in a database. I will provide 3 sequence diagrams that illustrate which information is written to the database from the user's perspective.

7.1.1. Interaction scenario 1: Logging of (branch) question data

In this scenario the user's answers to questions and choices made at branch questions are written to the database.

The information written to the database in this scenario is:

- User ID.
- Each video name.
- Question Text/ Branch question text.
- Answer the user has given/ branch choice chosen.

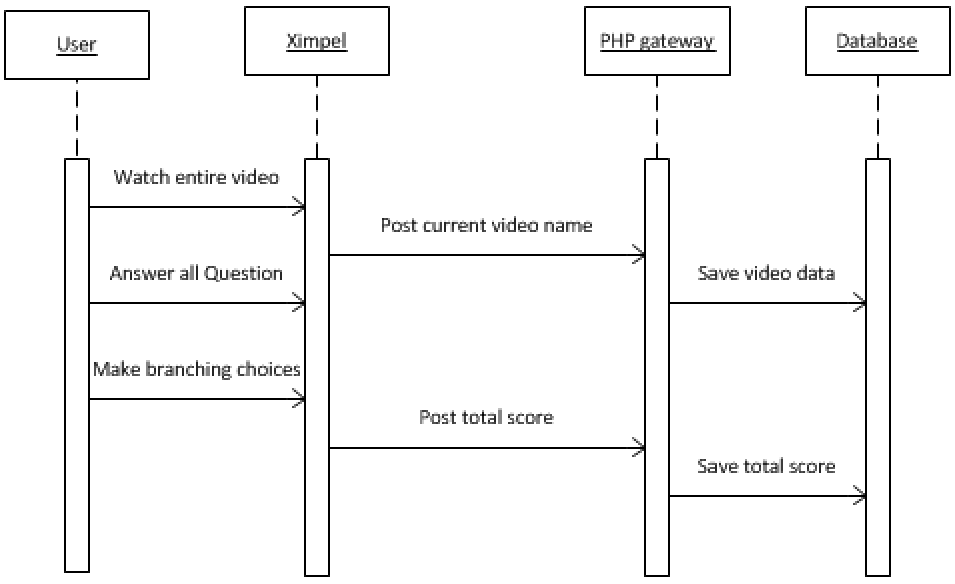


7.1.2. Interaction scenario 2: Logging of score data.

In this scenario only the total score is written to the database. The user watches the entire video and the accumulated score is saved in the database when the summary screen is present to the user.

The information written to the database in this scenario is:

- User ID.
- Each video name
- Total score

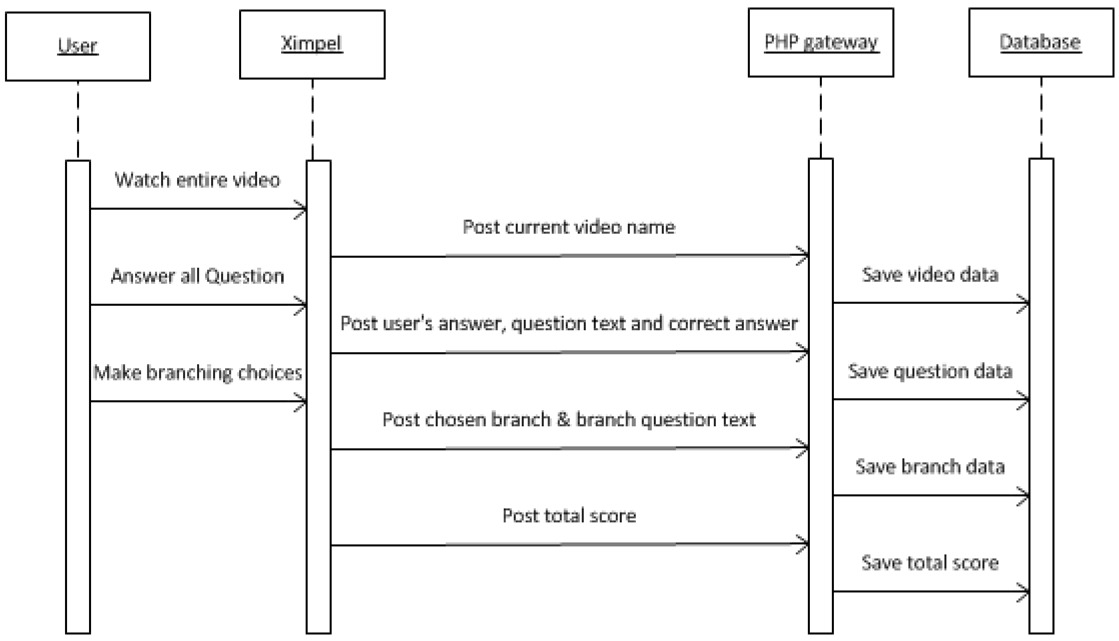


7.1.3. Interaction scenario 3: Logging of questions and score data.

In this scenario both total score and question answers or choices are written to the database.

The information written to the database in this scenario is:

- User ID.
- Each video name.
- Question Text/ Branch question text.
- Answer the user has given/ branch choice chosen.
- Total score



7.2. A video provider's perspective

The video provider's perspective focuses on the setup, configuration of the Ximpel and the management of the information that is written to the database.

7.2.1. Setup

Before the database features of Ximpel can be used, the video provider has to first setup the SQLite database and create the needed tables. This can be done easily by uploading all of the files in the folder named PHP. Then navigate to the following URL: ***http://<your_domain>/<path_to_folder_where_uploaded_files_are>/install.php***.

This will automatically create the database and the required tables. After this process is complete, a properly configured Ximpel will be able to communicate with the database.

7.2.2. Ximpel configuration

Ximpel writes to the database indirectly by performing a HTTP POST request with the information to a PHP gateway. The URL where Ximpel can reach this PHP file can be configured in the ximpeConfig.xml by adding the databaseUrl tag. An example of what this looks like is:

- <databaseUrl>www.example.com/inserter.php </databaseUrl>

A valid URL is necessary for Ximpel to communicate with the PHP gateway, replace www.example.com with your own valid domain. Also replace /inserter.php with the correct path that leads to your PHP gateway file (the default name of this file is inserter.php).

The video provider then determines which information is written to the database. This can be done by adding a database tag to the XimpelConfig.xml file. The tag can contain one of three text strings which each represent a logging option, these are:

- Questions
- Score
- QuestionsAndScore

The entire line of text which needs to be added to the XimpelConfig.xml will then be one of the following:

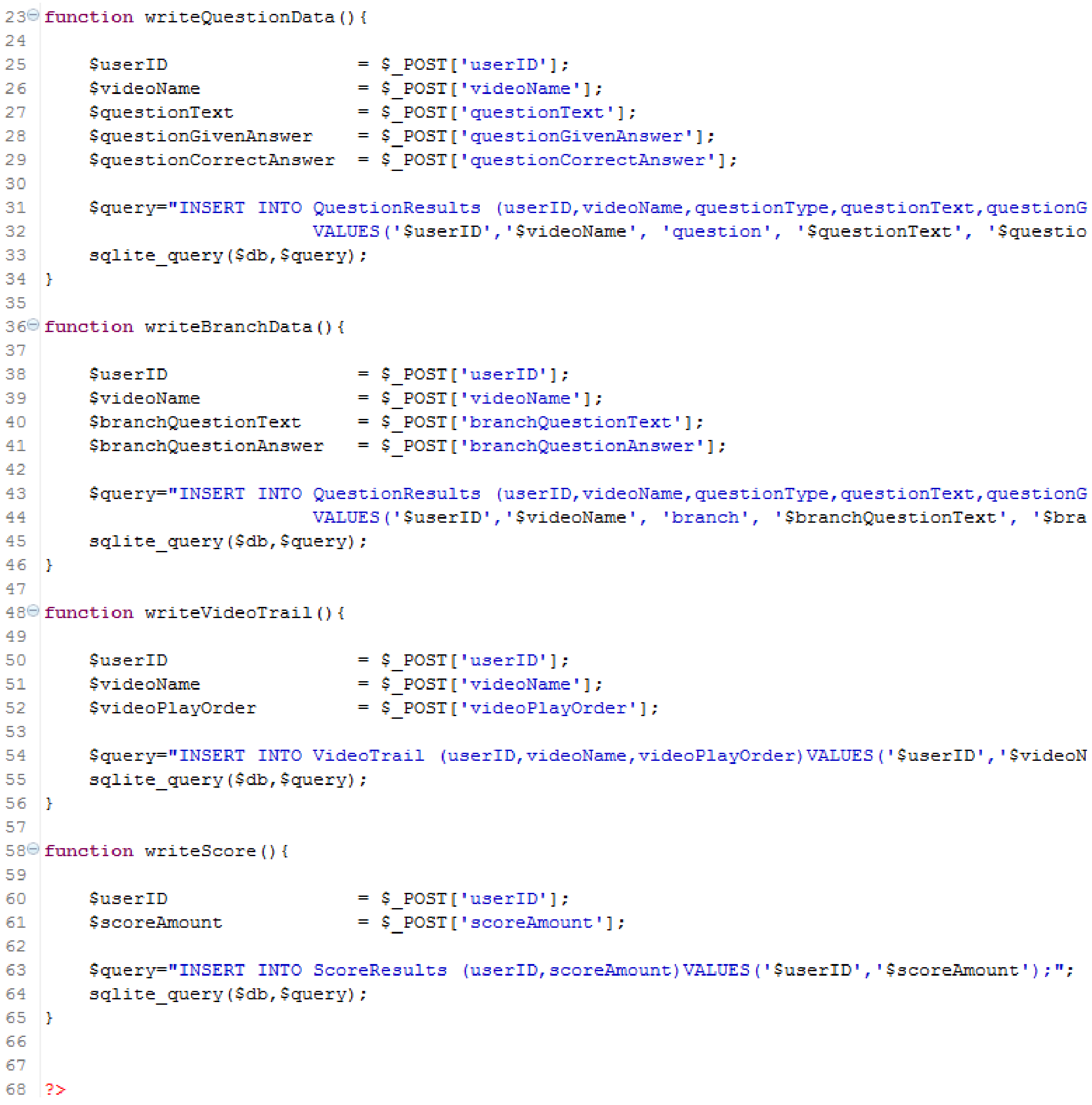
- <database>Questions</database>
- <database>Score</database>
- <database> QuestionsAndScore </database>

7.2.3. Writing to the database

After adding the required tag and the chosen logging option to the ximpelConfig.xml file, the video provider can choose to modify the PHP gateway file. This file is named ***inserter.php*** by default and contains a few functions which take care of writing the posted information to the database, these functions are:

- writeQuestionData()
- writeBranchData()
- writeVideoTrail()
- writeScore()

Each of these functions populates a number of local variables with the posted variables it receives and performs an INSERT query on the database. The video provider can modify these functions if he/she so chooses, but he/she has to keep in mind that some alterations might require a modification of the database structure as well.
The following is a code snippet of these functions and the full source code of this file can be found in the appendix.



```
function writeQuestionData(){

    $userID                 = $_POST['userID'];
    $videoName              = $_POST['videoName'];
    $questionText           = $_POST['questionText'];
    $questionGivenAnswer    = $_POST['questionGivenAnswer'];
    $questionCorrectAnswer  = $_POST['questionCorrectAnswer'];

    $query="INSERT INTO QuestionResults (userID,videoName,questionType,questionText,questionG
                        VALUES('$userID','$videoName', 'question', '$questionText', '$questio
    sqlite_query($db,$query);
}

function writeBranchData(){

    $userID                 = $_POST['userID'];
    $videoName              = $_POST['videoName'];
    $branchQuestionText     = $_POST['branchQuestionText'];
    $branchQuestionAnswer   = $_POST['branchQuestionAnswer'];

    $query="INSERT INTO QuestionResults (userID,videoName,questionType,questionText,questionG
                        VALUES('$userID','$videoName', 'branch', '$branchQuestionText', '$bra
    sqlite_query($db,$query);
}

function writeVideoTrail(){

    $userID                 = $_POST['userID'];
    $videoName              = $_POST['videoName'];
    $videoPlayOrder         = $_POST['videoPlayOrder'];

    $query="INSERT INTO VideoTrail (userID,videoName,videoPlayOrder)VALUES('$userID','$videoN
    sqlite_query($db,$query);
}

function writeScore(){

    $userID                 = $_POST['userID'];
    $scoreAmount            = $_POST['scoreAmount'];

    $query="INSERT INTO ScoreResults (userID,scoreAmount)VALUES('$userID','$scoreAmount');";
    sqlite_query($db,$query);
}


?>
```

7.2.4. Generating a replay

One of the most important requirements of what can be done with the data that is stored in the database is the ability to recreate a user's playthru. This can be done by navigating to the ***replayGenerator.php .***

The video provider is presented with a table of which each row contains a user id. Each user id represents an individual playthru. Next to each user id is a link titled "generate replay". When the video provider clicks on the "generate replay" link he is presented with a downloadable replay.xml file.This xml file is in the same format as a normal Ximpel playlist.xml and must be used in the same way.

The difference between the replay.xml and the playlist.xml is, that in the replay.xml there are no user interactive elements. When running Ximpel with the replay.xml as a playlist, all the videos that the original user has viewed will be displayed in the same order. If the original user was presented with questions (the correctness of the answer is not important)these will be displayed on a standard overlay together with the answer the user had given, as well as the correct answer to the question. The same is true for branch question information and at the end of the video the total score acquired by the original user will be displayed on an overlay as well.

The way the replay Generator compiles a replay is:

1. Using the unique user ID, a lookup is performed for all the videos that have the same unique user ID in the VideoResults table.

2. The results of video lookup are arranged in ascending order.

3. For each video found, a lookup is performed on the QuestionResults table where for each result an overlay will be created with the needed information.

4. All the results are placed in xml format in the same way a playlist.xml would be constructed.

The replay.xml will be playable in Ximpel regardless if Ximpel has database logging enabled.

8. Conclusions and recommendations

8.1. Conclusions

At the beginning of this thesis I formulated some questions which would have to be answered before and during the master's project. That it is possible to add database functionality to Ximpel has been proven and the benefits have been described in chapter 2. The fact that Ximpel cannot interact with the database directly turned out to be a positive aspect.

By having a middle-tier in the form a PHP gateway file, it makes it much easier for future developers to modify the database structure without having to modify the Ximpel source code. It also works very well with the proposed generic method for reading information from the database by using a custom media type.

By approaching the project as a traditional software development project was very educational. Even if I was the only developer I can see how important it is to have structure that can guide you during the project. This will of course be even more important when developing larger software products with multiple teams of developers.

Evaluating the project can best be done by determining if the resulting software satisfies the requirements established at the start. While the system provides all the functions required by the functional requirements, I would like to take a closer look at the non-functional requirements.

Usability

The resulting software is definitely easy to use and should not be harder to understand then Ximpel itself.

Portability

The resulting software provides the same degree of portability as the normal Ximpel system. The only difference is that the amount of files which has to be moved between environments has increased.

Extensibility & Modifiability

During the project I was able to reduce my modifications and additions to the Ximpel source code considerably. I have taken care to provide comments at all my modifications in the source itself and have described the changes in this thesis. I assume that this will provide a clear overview for anyone who wants to continue extending the software.

Security

Security is one of the points where improvements can definitely be added. The current software does not take security measures into account. The current software would not be ready for public release, but it can be used in a controlled environment and for educational purposes.

Overall I think the project has been successful; no project is ever a 100% success and there is always room for improvement. I think it was a worthwhile goal to integrate database functionality into Ximpel, because it has greatly increased the possibilities.

8.2. What is to follow?

It would be interesting to further develop the database functionalities of Ximpel. Currently the system can write information to the database and can read information from the database. But these functionalities do not influence each other. For example, currently Ximpel cannot write to the database and then wait for certain information to become available in the database before continuing.

If the database functionalities were further developed, Ximpel could perhaps be used for multiplayer games with the database acting as a shared resource indicating when one player has performed a certain task and the other player can then react to this.

Further developing the database functionalities would probably mean creating a completely new version of Ximpel as the modification would probably become very invasive. Because of this, I think it would be best to develop this version of Ximpel next to the original one and perhaps give the new version a different name.

I myself, having acquired the insight, understanding and interest through this thesis study; plan on working on Ximpel in my free time, as I think it remains a very interesting platform with lots of potential.

References

1. *Playlists in XIMPEL.* Retrieved April 2011, from http://ximpel.few.vu.nl/tutorials/Playlists_in_XIMPEL.html

2. *Configuration files in XIMPEL.* Retrieved April 20, 2011, from http://ximpel.few.vu.nl/tutorials/Configuration_files_in_XIMPEL.html

3. *Overlays in XIMPEL.* Retrieved June 22, 2009, http://ximpel.few.vu.nl/tutorials/Overlays_in_XIMPEL.html

4. Eliëns A., Huurdeman H., van de Watering M., Bhikharie S.V., XIMPEL Interactive Video -- between narrative(s) and game play, *Proc. GAME-ON 08*, Nov 17-19, Valencia, Spain

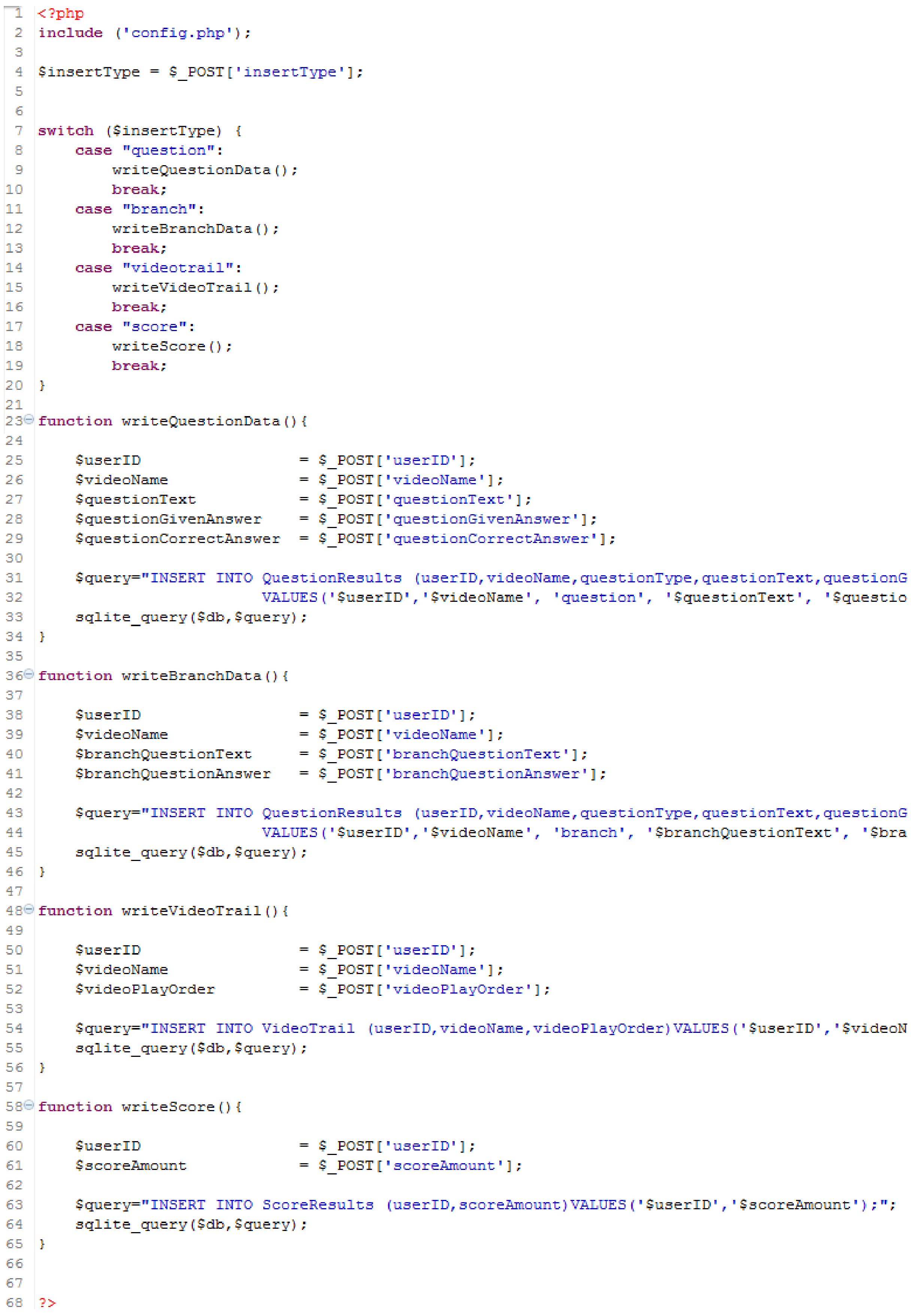
5. Hans van Vliet. (2007) *Software Engineering: Principles and Practice.* 2nd edn. John Wiley & Sons

6. James Governor, Dion Hinchcliffe & Duane Nickull. (2009) *Web 2.0 Architecture.* O'Reilly Media, Inc

7. Royce, W. (1970) Managing the development of large software systems. In *Proceedings of the IEEE Western Electronic show and convention. (WESCON), 25 – 28 Aug 1970, Los Angeles, USA,* pp. 1-9.

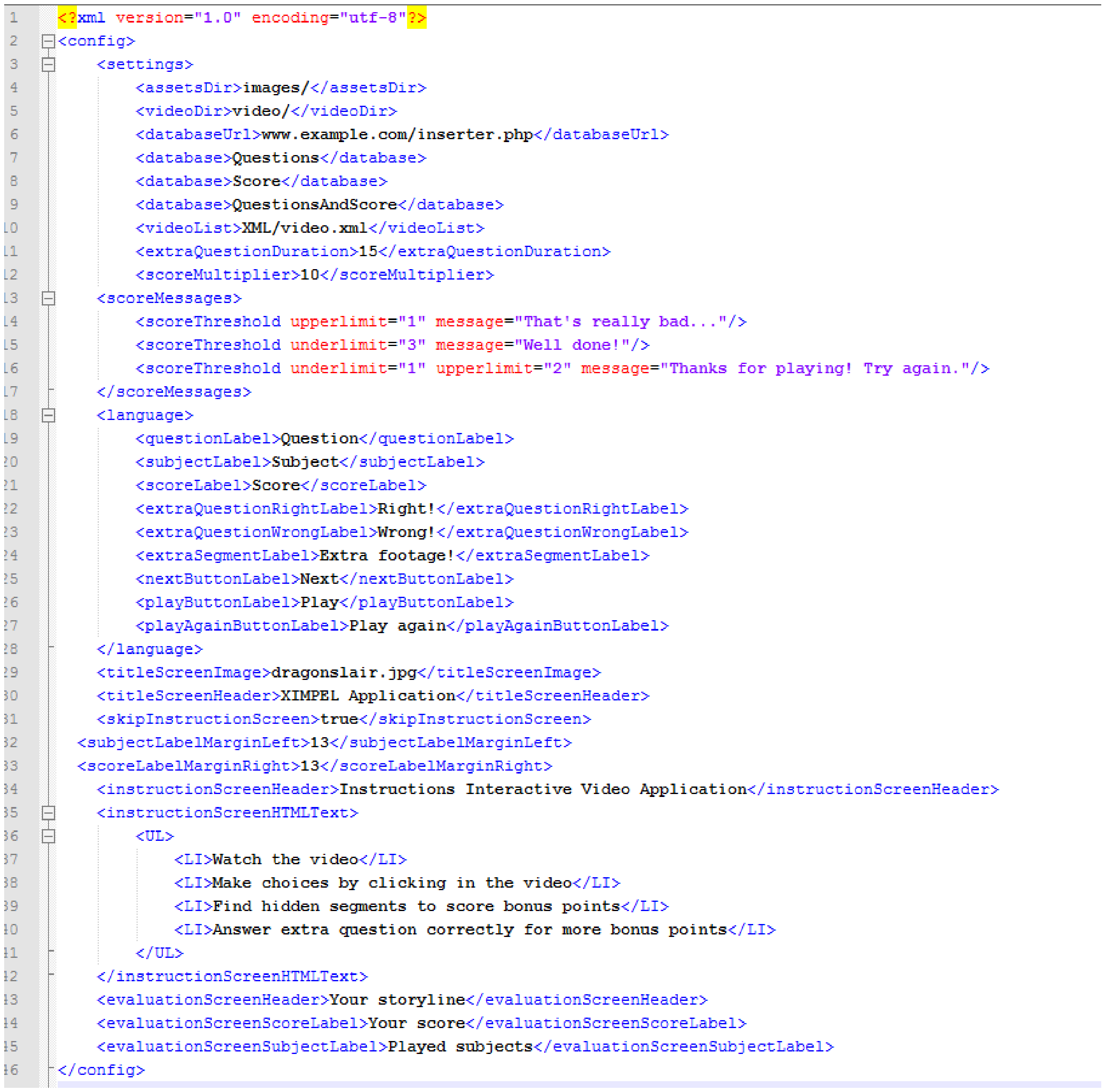
8. *Flex documentation.* Retrieved March 2011 from http://www.adobe.com/devnet/flex/documentation.html

Appendix: inserter.php



```php
<?php
include ('config.php');

$insertType = $_POST['insertType'];


switch ($insertType) {
    case "question":
        writeQuestionData();
        break;
    case "branch":
        writeBranchData();
        break;
    case "videotrail":
        writeVideoTrail();
        break;
    case "score":
        writeScore();
        break;
}

function writeQuestionData(){

    $userID                  = $_POST['userID'];
    $videoName               = $_POST['videoName'];
    $questionText            = $_POST['questionText'];
    $questionGivenAnswer     = $_POST['questionGivenAnswer'];
    $questionCorrectAnswer   = $_POST['questionCorrectAnswer'];

    $query="INSERT INTO QuestionResults (userID,videoName,questionType,questionText,questionG
                        VALUES('$userID','$videoName', 'question', '$questionText', '$questio
    sqlite_query($db,$query);
}

function writeBranchData(){

    $userID                  = $_POST['userID'];
    $videoName               = $_POST['videoName'];
    $branchQuestionText      = $_POST['branchQuestionText'];
    $branchQuestionAnswer    = $_POST['branchQuestionAnswer'];

    $query="INSERT INTO QuestionResults (userID,videoName,questionType,questionText,questionG
                        VALUES('$userID','$videoName', 'branch', '$branchQuestionText', '$bra
    sqlite_query($db,$query);
}

function writeVideoTrail(){

    $userID                  = $_POST['userID'];
    $videoName               = $_POST['videoName'];
    $videoPlayOrder          = $_POST['videoPlayOrder'];

    $query="INSERT INTO VideoTrail (userID,videoName,videoPlayOrder)VALUES('$userID','$videoN
    sqlite_query($db,$query);
}

function writeScore(){

    $userID                  = $_POST['userID'];
    $scoreAmount             = $_POST['scoreAmount'];

    $query="INSERT INTO ScoreResults (userID,scoreAmount)VALUES('$userID','$scoreAmount');";
    sqlite_query($db,$query);
}


?>
```

Appendix: example of ximpelConfig.xml with optional database tags



```xml
<?xml version="1.0" encoding="utf-8"?>
<config>
    <settings>
        <assetsDir>images/</assetsDir>
        <videoDir>video/</videoDir>
        <databaseUrl>www.example.com/inserter.php</databaseUrl>
        <database>Questions</database>
        <database>Score</database>
        <database>QuestionsAndScore</database>
        <videoList>XML/video.xml</videoList>
        <extraQuestionDuration>15</extraQuestionDuration>
        <scoreMultiplier>10</scoreMultiplier>
    <scoreMessages>
        <scoreThreshold upperlimit="1" message="That's really bad..."/>
        <scoreThreshold underlimit="3" message="Well done!"/>
        <scoreThreshold underlimit="1" upperlimit="2" message="Thanks for playing! Try again."/>
    </scoreMessages>
    <language>
        <questionLabel>Question</questionLabel>
        <subjectLabel>Subject</subjectLabel>
        <scoreLabel>Score</scoreLabel>
        <extraQuestionRightLabel>Right!</extraQuestionRightLabel>
        <extraQuestionWrongLabel>Wrong!</extraQuestionWrongLabel>
        <extraSegmentLabel>Extra footage!</extraSegmentLabel>
        <nextButtonLabel>Next</nextButtonLabel>
        <playButtonLabel>Play</playButtonLabel>
        <playAgainButtonLabel>Play again</playAgainButtonLabel>
    </language>
    <titleScreenImage>dragonslair.jpg</titleScreenImage>
    <titleScreenHeader>XIMPEL Application</titleScreenHeader>
    <skipInstructionScreen>true</skipInstructionScreen>
  <subjectLabelMarginLeft>13</subjectLabelMarginLeft>
  <scoreLabelMarginRight>13</scoreLabelMarginRight>
    <instructionScreenHeader>Instructions Interactive Video Application</instructionScreenHeader>
    <instructionScreenHTMLText>
        <UL>
            <LI>Watch the video</LI>
            <LI>Make choices by clicking in the video</LI>
            <LI>Find hidden segments to score bonus points</LI>
            <LI>Answer extra question correctly for more bonus points</LI>
        </UL>
    </instructionScreenHTMLText>
    <evaluationScreenHeader>Your storyline</evaluationScreenHeader>
    <evaluationScreenScoreLabel>Your score</evaluationScreenScoreLabel>
    <evaluationScreenSubjectLabel>Played subjects</evaluationScreenSubjectLabel>
</config>
```

* Note, that only one of the <database> tags must be used at a time.