Propel Designer

Final Report for CS39440 Major Project

Author: Steven James Meyer (svm9@aber.ac.uk)
Supervisor: Prof. Reyer Zwiggelaar (rrz@aber.ac.uk)

April 11, 2013
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a MEng degree in
Software Engineering (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature .....

Date .....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature .....

Date .....

Acknowledgements

I am grateful to...
I'd like to thank...

Abstract

Databases can be usefully modelled as Entity Relationship Diagrams (ERDs). Such diagrams use visuals to show information about database schemata such as relationships and cardinalities, entities and attributes in a way which is easy for a human to recognise. Colouring and emphasis can further assist in using ERDs as a useful tool for sharing and communicating a database schema.

Propel is an open-source Object-Relational Mapping (ORM) for PHP. It creates objects to allow for programmatic database manipulation and for database abstraction. It creates these objects and the database implementation from schema files. These schemata define the tables, columns, keys, indices and relations and their attributes within a database.

Schema files are coded in XML. While this makes them understandable so far as the data is marked-up in a human-readable manner, it makes sharing and collaborating difficult. The linear structure makes it difficult to see the structure and relationships. This, in turn, makes it difficult to ensure that the schema is modelling the domain and following business logic correctly. They are also not very friendly for the less technically-inclined.

It can be seen, then, that it would be useful to have a tool which can be used to manipulate Propel schemata by means of direct XML manipulation and by ERDs.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

Background

Data modelling and database design are vast topic areas accompanied by many volumes of published theory and literature. This project focusses on a fragment of the topic; conceptual design/ schemata. Specifically, it is concerned with entity relationship (ER) grammar and mainly with ER diagrams.

The ER modelling grammar for conceptual modelling serves two major purposes. Firstly, it can be used as a communication device used by an analyst to interact with an end-user. Secondly, it can be used as a design tool at the highest level of abstraction to communicate a deeper understanding to the database designer [Umanath & Scamell (2007)]. Id est the database structure and semantics can be described in a manner which is not specific to any implementation.

Where object relational mapping (ORM) is used, such diagrams can also be useful devices for describing the relationships between objects to the application programmers. This is particularly useful where business logic is applied at the application level; developers (who may not have designed the database, themselves) must have a way to understand the logic represented by the database design and to understand those parts of the business logic which must be applied at the application level. This is particularly relevant when using an ORM such as Propel, where the level of database abstraction afforded to the application programmers comes at the price of losing the ability to describe logic at the database level. Database-level logic, in this case, is represented by stored procedures, triggers et cetera.

For the ORM Propel, database schemata are described as XML files. These descriptions are a mixture of database instruction such as entities (tables), attributes (columns) and simple referential actions for deletion and updating of rows, and of PHP instructions for the generated PHP code.

These files make for more difficult traversal for application programmers and database engineering alike, as references must be searched for among the (possibly) many entity descriptions and indices are separately described from the attributes they reference. It is also far too tempting for application programmers to make changes to the schemata when requirements change. Maintaining any ER diagrams in such situations is a maintenance issue, and checking that the resultant schemata still correctly and properly represent the business logic becomes very difficult.

There exists many database modelling tools which support the creation and editing of ERDs. Many of these tools are useful tools for inspecting existing databases, providing support for many different back-ends. However, they are primarily aimed at SQL interaction with these databases, rather than at maintaining ER diagrams.

There are products which will natively support ER diagrams and ORM frameworks, but these

products are not free software. Existing software, as it stands, usually has one of these weaknesses:

- They do not support ORM. They support different databases, but cannot be made to work with an ORM schema.
- They do not support ERDs or relational modelling schema (a form of physical data modelling).
- They are not free software. They are either not *gratis* (without cost) or not *libre* (without restrictions) or both.

Objectives

The most basic aim of this project is to create a free tool to create ER/RM diagrams from Propel schema XML and vice versa. It should provide the expected features of existing modelling tools, but be specifically aimed at Propel schemata. The normalised information-preserving grammar of Propel is more closely related to relational modelling grammar, so RM modelling grammar would be preferable to ER modelling grammar.

The core goals of the project are:

- To produce an schema diagram consisting of (*i*) tables, (*ii*) relationships (*iii*) attributes (and their types) and (*iv*) cardinality notations using a Propel schema as input.
- To make such a diagram interactive/editable with a database modelling tool.
- To produce a Propel schema XML using a schema diagram as input, id est the reverse of goal 1.
- To be gratis to use.

The project had these stretch goals which were available if the core goals were to be achieved in good time:

- Further integration with Propel to allow for additional logic such as validators and behaviours.
- Include inheritance. This is part of the Propel schema and introduces aspects of enhanced ER modelling.
- To be free software, id est libre.

Chapter 2

Development Process

You need to describe briefly the life cycle model that you used. Do not force your project into the waterfall model if it is better described by prototyping or some other evolutionary model. You do not need to write about all of the different process models that you are aware of. Focus on the process model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

In most cases, the agreed objectives or requirements will be the result of a compromise between what would ideally have been produced and what was felt to be possible in the time available. A discussion of the process of arriving at the final list is usually appropriate.

You should briefly describe the design method you used and any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.

2.1 Introduction

Introduce the specific model that you chose to use.

2.2 Modifications

Did you have to modify the model to suit a one-person project. If so, what did you change and why?

Chapter 3

Design

You should concentrate on the more important aspects of the design. It is essential that an overview is presented before going into detail. As well as describing the design adopted it must also explain what other designs were considered and why they were rejected.

The design should describe what you expected to do, and might also explain areas that you had to revise after some investigation.

Typically, for an object-oriented design, the discussion will focus on the choice of objects and classes and the allocation of methods to classes. The use made of reusable components should be described and their source referenced. Particularly important decisions concerning data structures usually affect the architecture of a system and so should be described here.

How much material you include on detailed design and implementation will depend very much on the nature of the project. It should not be padded out. Think about the significant aspects of your system. For example, describe the design of the user interface if it is a critical aspect of your system, or provide detail about methods and data structures that are not trivial. Do not spend time on long lists of trivial items and repetitive descriptions. If in doubt about what is appropriate, speak to your supervisor.

3.1 Overall Architecture

3.2 Some detailed design

3.2.1 Even more detail

3.3 User Interface

3.4 Other relevant sections

Chapter 4

Implementation

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex, perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant? Your implementation might well be described in the same chapter as Problems (see below).

Chapter 5

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on 'real users'? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

5.1 Overall Approach to Testing

5.2 Automated Testing

5.2.1 Unit Tests

5.2.2 User Interface Testing

5.2.3 Stress Testing

5.2.4 Other types of testing

5.3 Integration Testing

5.4 User Testing

Chapter 6

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

The critical evaluation can sometimes be the weakest aspect of most project dissertations. We will discuss this in a future lecture and there are some additional points raised on the project website.

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

Appendix B

Code samples

2.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
  /*-------------------------------------------------*/
  /* Minimum Standard Random Number Generator        */
  /* Taken from Numerical recipies in C              */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity   */
  /* Always call with negative number to initialise  */
  /*-------------------------------------------------*/

  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
```

```
double temp;

if (*idum <=0)
{
  if (-(*idum) < 1)
  {
    *idum = 1;
  }else
  {
    *idum = -(*idum);
  }
  idum2=(*idum);
  for (j=NTAB+7;j>=0;j--)
  {
    k = (*idum)/IQ1;
    *idum = IA1 *(*idum-k*IQ1) - IR1*k;
    if (*idum < 0)
    {
      *idum += IM1;
    }
    if (j < NTAB)
    {
      iv[j] = *idum;
    }
  }
  iy = iv[0];
}
k = (*idum)/IQ1;
*idum = IA1*(*idum-k*IQ1) - IR1*k;
if (*idum < 0)
{
  *idum += IM1;
}
k = (idum2)/IQ2;
idum2 = IA2*(idum2-k*IQ2) - IR2*k;
if (idum2 < 0)
{
  idum2 += IM2;
}
j = iy/NDIV;
iy=iv[j] - idum2;
iv[j] = *idum;
if (iy < 1)
{
  iy += IMM1;
}
if ((temp=AM*iy) > RNMX)
{
```

```
    return RNMX;
  }else
  {
    return temp;
  }
}
```

Annotated Bibliography

Inc., AquaFold. 2012 (Nov.). *Aqua data studio features*. http://www.aquafold.com/aquadatastudio_features.html. Accessed 2012/11/19.

Sebesta, Robert W. 2007. *Programming the world wide web*. fourth, international edn. Addison Wesley.

Sherratt, Edel. 2011 (Nov.). *Entity relationship translation*. http://www.aber.ac.uk/~dcswww/Dept/Teaching/CourseNotes/current/CS27020/Lectures/ER-translation.pdf. Accessed 2012/11/18.

Stevens, Perdita, & Pooley, Rob. 2006. *Using uml: Software engineering with objects and components*. second edn. Addison-Wesley.

Team, Propel. 2012a (Aug.). *Basic relationships*. http://propelorm.org/documentation/04-relationships.html. Accessed 2012/11/19.

Team, Propel. 2012b (Sept.). *Database schema*. http://propelorm.org/documentation/04-relationships.html. Accessed 2012/11/19.

Umanath, Narayan S. & Scamell, Richard W.̇2007. *Data modeling and database design*. Thomson Course Technology.

van der Vlist, Eric. 2002 (Jan.). *Xsltunit*. http://xsltunit.org/0/2/. Accessed 2013/02/02.

w3schools. *Xslt elements reference*. http://www.w3schools.com/xsl/xsl_w3celementref.asp. Accessed 2013/02/02.