COMP420 Interim Report: Automated Database Wrapper Class Generation

Prepared for
Dr. Mark Apperley
Dr. Sally Jo Cunningham
Department of Computer Science
University of Waikato

Prepared by
Jason Kelly
COMP420-06C: Report of an Investigation
Department of Computer Science
University of Waikato

Friday, 2 June 2006

Summary

The following interim progress report on the Automated Database Wrapper Class Generation software development tool provides a summary of the work completed to date, and outlines the remaining tasks required to complete this software development tool project. This document focuses on the technical requirements of the project and only briefly covers programming details. The project implementation consists of three phases: firstly, research, design and refinement of original concepts; secondly, implementation of design; lastly, user acceptance testing and feedback.

Table of Contents

Background

In the forth quarter of 2005, all students intending to enrol in the honours stream for the Bachelor of Computing and Mathematical Science (BCMS) degree at the University of Waikato were instructed to select and propose a project for the 2006 honours paper Report of an Investigation. Lecturers from the Department of Computer Science composed a list of possible honours paper projects related to their research areas, from which students could select a project. Students were also able to propose their own projects provided a member of the Department of Computer Science was willing to supervise the project.

The proposal I developed for the honours paper project stemmed from the lack of availability of an easy-to-use, open source software development tool to automate the generation of database wrapper classes based on a textual representation of the database structure. Dr. Sally Jo Cunningham has kindly agreed to supervise my honours project.

Project Description

This project is intended to provide Java software developers with an easy-to-use development tool for automatically generating database wrapper classes based on the Structured Query Language (SQL) script file that defines the database schema. Database wrapper classes mirror the database tables, and are used to retrieve data from and write modified data to the database (Lauesen, 1998).

By wrapping database access in a set of object classes we are able to treat the database as objects and not have to concern ourselves with the database-specific access details hidden in the objects. This methodology provides a couple of benefits: firstly, it provides a layer of abstraction between the database system and the application logic, which allows software developers to separate the application logic from the database access code, which in turn makes maintenance of the code base easier because it more modular and the code is grouped logically by the functions it performs; secondly, it allows the underlying database technology to be changed while insulating the application logic from those changes.

The goal of this project is to build a Java-based open source software development tool that is able to:

- *Parse and interpret SQL schema definition files*: many Database Management Systems (DBMS) often stray from the SQL 1999 standard and implement their own proprietary data types and SQL constructs. The parsing engine should correctly handled and interpret these custom elements.

- *Generate and write to disk wrapper classes*: Java classes should be generated based on the tool's interpretation of the SQL schema definition file. The classes should utilise the JDBC Application Programming Interface (API), and be compatible with JDBC Type 4 drivers[1]. All wrapper classes generated should be encapsulated in a user-specified package.

The life of this development tool is planned to extend beyond that of the honours paper project, and as such, suitable hosting should be found for the project. As it is planned to be an open-source application, after an appropriate Open Source Initiative (OSI) approved licence[2] has been selected for the project, then hosting can be organised with hosting service provider such as SourceForge[3].

Research and Design

There were two possible routes to take on designing how the wrapper class generation tool would be built. The first route was to build a stand-alone tool which could be run by developers as and when they needed to build the wrapper classes. The benefit of having a stand-alone tool of this nature is that it is independent of the developers' chosen development environment as its only requirement would be the Java Run Time Engine (JRE). A disadvantage it that the tool would be another application that the developers would have to run.

The second route, which was the one I chose for this project, was to integrate the tool as a plug-in into a Java-based Integrated Development Environment (IDE). I

[1] See Appendix B for JDBC driver type definitions.
[2] http://www.opensource.org/licenses/
[3] http://sourceforge.net/

chose to integrate the database wrapper class generation tool with the Eclipse IDE[4] for three reasons:

- It has a large user base and a very active online community.
- It is an open-source development environment written in Java, and is written using a modular, plug-in orientated framework, which has public APIs that allow for the extension, reuse and modification of all functionality already available in the IDE.
- I have used it for Java application development and am more familiar with it than with other Java IDEs.

The benefits of integrating into an IDE as a plug-in are that, firstly, the IDE manages controls the development environment; secondly, the IDE is able to provide a notion of project-context which helps ensure code is generated in the correct location; thirdly, the plug-in tool can use the built-in functionality of the IDE to generate the wrapper class source files; and lastly, the extra functionality that the tool offers developers in now available to them from within the IDE.

The disadvantage of using the plug-in model is that developers who do not use the IDE that the tool integrates with will not be able to use the tool unless they change to the supported IDE.

The second step in the design process was to determine what the major components were, and what their sub-components were. The following list gives an overview of what the components were:

- Eclipse IDE UI Integration
    - Menu bar item
    - Toolbar button
    - Popup menu item
    - Wizard
- SQL Interpreter
    - SQL file parser
    - Database structure model builder
- Source Code Generator

[4] http://www.eclipse.org/

The following three sub-sections describe each of the components listed above.

Eclipse IDE UI Integration

To ensure that users are able to easily and quickly access the wrapper class generator tool, it should have an easy-to-see and use representation in the user interface of the IDE. To achieve this, there should be three different user interface extensions: an entry on the menu bar, a toolbar button, and an entry on a popup menu. The menu bar item and toolbar item should be available at all times that the plug-in is active. The entry on the popup menu should only be available when the plug-in is active and when a SQL script file is highlighted.

When any of these three user interface extension points are activated, the user should be presented with a wizard, which will be used to gather the information the wrapper class generator requires to perform its task. The wizard should be divided into two pages: the first page should obtain the location of the SQL script file that is to be parsed and the location of the package into which to place the generated source code; the second page should present the user with a list of all the tables for which wrapper classes will be generated, allowing the user to select and deselect tables. Deselected tables will not have wrapper classes generated for them.

SQL Interpreter

The SQL Interpreter has two main sub-components: the SQL parser and the database structure model builder.

The SQL parser will parse the SQL script file that the user has selected on the first page of the wizard. The parser will be responsible for extracting the table structure information from the SQL script file. This information will include the table names, the column names, the column data types, the column length restrictions, the primary and foreign key information, and some data integrity check information. These extracted details will then be passed to the database structure model builder, which will then build an in-memory representation of the database structure.

These two steps will be preformed between pages one and two of the wizard. The information held in the in-memory database structure model can then be used to

allow the user to select which tables they want wrapper classes generated for. The tables that the user selected for wrapper class generation would be flagged for generation in the model. If, for any reason, the SQL parser determines that there are errors in the SQL script file, then the user can be alerted to that fact on the second page of the wizard.

Source Code Generator

The source code generator is responsible for using the Eclipse IDE code generation API to write the database wrapper classes to the package that the user specified. The code generator iterates through the in-memory database structure model generating the wrapper classes for the tables that have been marked for wrapper class generation. The code generator will also infer and generate some simple query methods based on primary and foreign key constraint information obtained from the script file.

The source code generated by the code generator will be complete and syntactically correct, and should be usable immediately for simple database access. The generated source code will also have full documentation and the documentation should be usable immediately as API documentation for the wrapper classes.

Work Completed

With regard to hosting the project on SourceForge, I have registered an account with them and have obtained documentation that specifies how an application for the hosting of an open-source project is processed, and what supporting project information is required for the application process.

The integration into the Eclipse IDE user interface is mostly completed. The menu bar item, toolbar button, and popup menu item contributions to the IDE graphical user interface are completed and functioning correctly. Activating any of the three will start the wizard for the plug-in. The interface contributions are specified using an XML file which describes what the contributions are, how they should function, and which action classes should be run when the contributions are selected.

The first page of the wizard is completed. The first page provides text entry fields that allow the user to specify the source SQL script file and the destination

package. The user also has the option to turn on an overwrite flag which will instruct the code generator to replace any classes existing in the specified package which have the same names as any of the generated classes. Both text entry fields have "Browse" buttons, which allow the user to browse the project work-space graphically using a tree structure, and allows them to select an SQL file and package for each text field respectively.

Screenshots of the above completed graphical user interface contributions are included in Appendix A.

I have also completed planning what overall structure the generated wrapper classes should have. The tool will generate a generic table wrapper which will contain functions applicable to all tables, such as general database query methods and result set transversal. Individual table wrapper classes will then extend the functionality provided by the generic wrapper class, and implement any table-specific functionality, such as running table-specific queries against the database table. The table wrapper class API will encapsulate table instance-data (one row of data in the table) in an object, thereby allowing information for an instance of an entity to be treated as a single object. The data-instance object will also provide the data integrity enforcement. See Appendix C for a diagram representing this structure.

Current Work

After examining a number of OSI licenses, the list of potential licences that could be chosen to publish this project under has been narrowed to four. These are the Eclipse Public License[5] (EPL), the Common Public License version 1.0[6] (CPL1.0), the GNU Public License[7] (GPL), and the GNU Lesser General Public License[8] (LGPL).

In the wizard I am refining the filters for the project-space browsing dialogs to filter out resources that need not be displayed, such as Java source code files when searching for a SQL script file. I am also working on the second page of the wizard, positioning the correct components and testing that they work correctly.

[5] http://www.opensource.org/licenses/eclipse-1.0.php
[6] http://www.opensource.org/licenses/cpl1.0.php
[7] http://www.opensource.org/licenses/gpl-license.php
[8] http://www.opensource.org/licenses/lgpl-license.php

I have begun planning how the in-memory database structure model will be implemented, and how the SQL parser will populate it with information. The model will structured as an n-ary tree with the first level of nodes containing the table names, the second level of nodes containing the column names, and the third level of nodes containing column attributes. The column attributes would be stored in an array structure, and would include details such as the data types of the columns, any data integrity constraints, and whether or not the columns are members of primary or foreign keys. See Appendix C for a graphical representation of the proposed model structure.

Future Work

Future work on this project within the scope of the honours paper is organised based on completion order. Future work beyond the planned scope for the honours project is included at the end of this section.

On completion of the current tasks, I plan to implement the SQL parser and the in-memory database structure model concurrently. Once this is complete I will integrate these components into the wizard process between the two wizard pages. Once this integration is complete I will build the code generation algorithms to generate, using the API functionality provided in the Eclipse IDE, the Java source code for the required wrapper classes.

Following the completion of the SQL parser I will begin testing the parsing component and ensure that it is building the correct database structure models for database schema SQL scripts written for different proprietary DBMS. After the completion of the code generator I will test that the plug-in tool is in-fact writing syntactically correct code that will perform its intended function correctly.

When I am confident that the plug-in is stable enough for general user testing and evaluation, I will organise developer testing and feedback for the plug-in.

Beyond Honours

Future development after the completion of the honours paper may include some of the following.

Users should be able to annotate the SQL script file with SQL comment lines that specify customised SQL queries and method names for the plug-in tool to generate in addition to the standard queries. A use case for this feature would be where the user has a need to retrieve a set of rows from the database based on a column that is not a part of a primary or foreign key, such as customers' surnames.

On the second page of the wizard users should be shown a list of the table columns and their related method names, and be given the ability to modify the function names as they see fit. A use case for this feature could be where the underlying technology has been changed and a default wrapper class generation would render method names that are different to the previously available API. This would allow users to modify the generated wrapper API at generation time.

Conclusion

At the completion of this honours project, the Automated Database Wrapper Class Generator plug-in, for the Eclipse IDE, should be able to extract database structure information from a SQL script file containing the database schema. It should then be able to use that structure information to generate a set of wrapper classes which, without modification, can be used to access the database.

References

Lauesen, S. (1998). Real-life object-orientated systems. In *IEEE Software* (1998, March/April), 15, 2, 76 – 83.

Appendix A: UI Integration

The following screenshots are taken from the work currently done regarding the integration of the plug-in into the Eclipse IDE user interface.

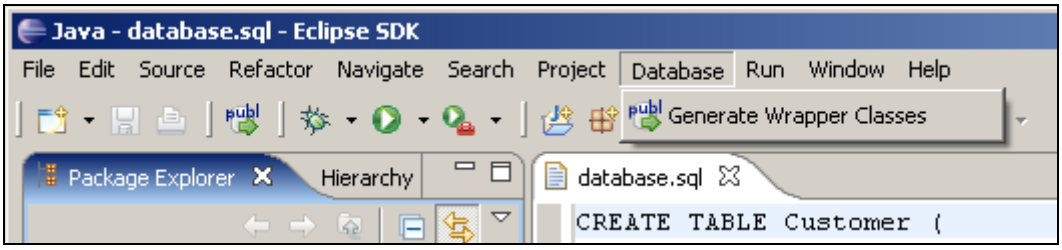

Fig. 1 Screenshot showing menu bar item and toolbar button.

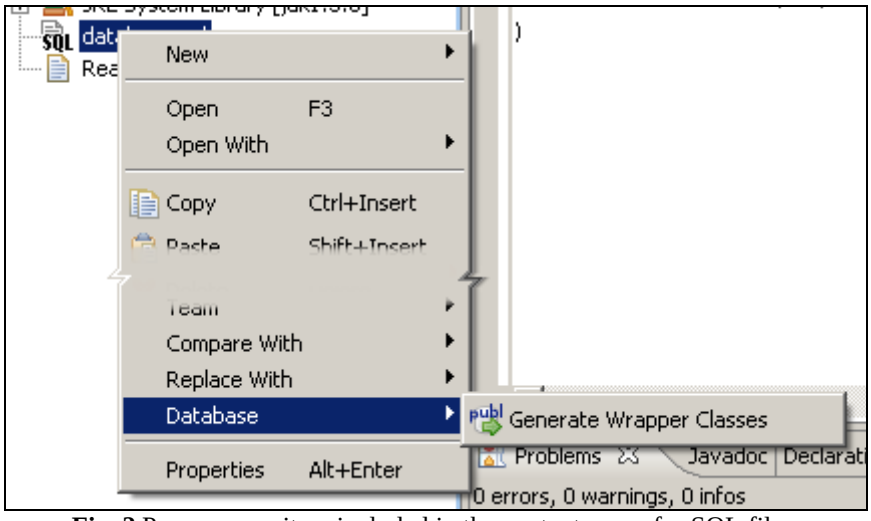

Fig. 2 Popup menu item included in the context menu for SQL files.

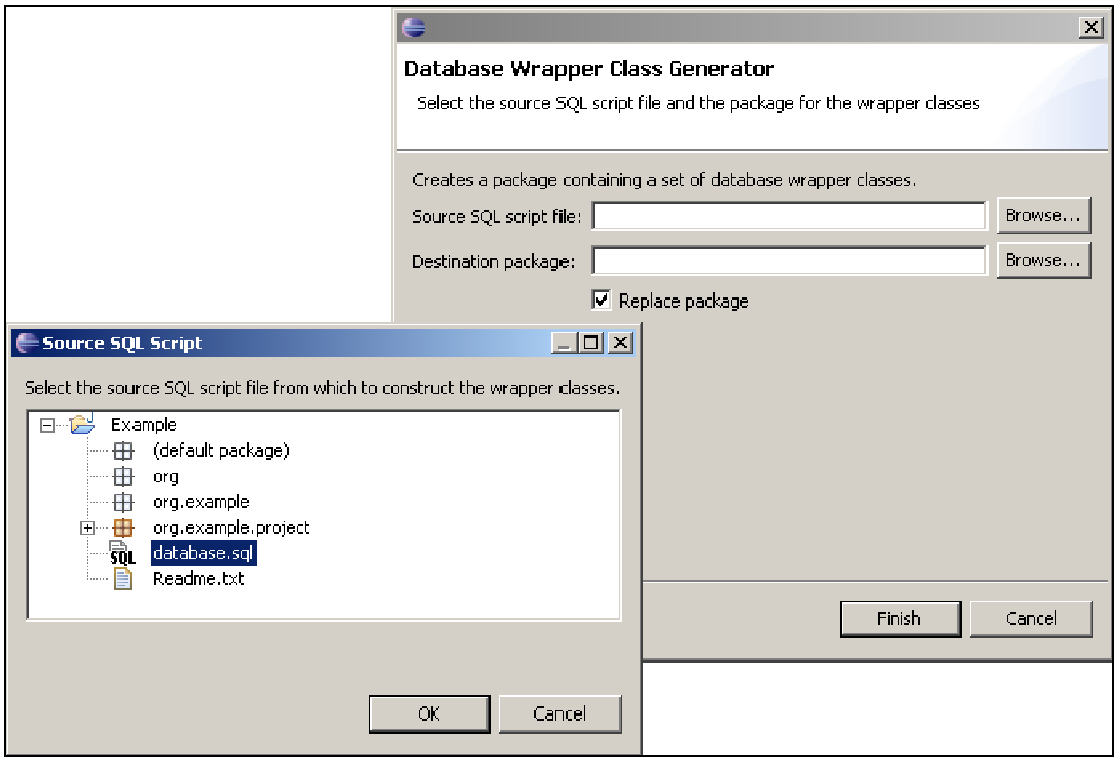

Fig. 3 Wizard showing the "Browse" dialog for finding the SQL script file.

Appendix B: JDBC Driver Types

The JDBC technology drivers fit into one of four types:

- *Type 1*: Access is provided through the use of a JDBC-ODBC bridge. This means that the JDBC API calls functions in one or more ODBC drivers. In many case this type of driver requires that native ODBC driver and native database client software be installed on the machine using the JDBC-ODBC bridge.
- *Type 2*: Provides a native-API that the JDBC API can call on. Often the full JDBC API is not supported, and this type of driver requires that native driver code is installed on the machine.
- *Type 3*: This is a network protocol server middleware. A client can connect to the server middleware using the JDBC API, which is fully supported, and the server middleware translates the JDBC API calls into DBMS-specific network protocol API calls.
- *Type 4*: This is pure Java JDBC client that communicates with the DBMS using the proprietary protocol belonging to the DBMS. Type 4 drivers are usually only available from the DBMS manufacturers.

For more information about the different JDBC driver types, see the Sun Microsystems documentation regarding the different JDBC driver types found at: http://java.sun.com/products/jdbc/driverdesc.html

Appendix C: Structure Diagrams

The following is a graphical representation of the in-memory database structure model:

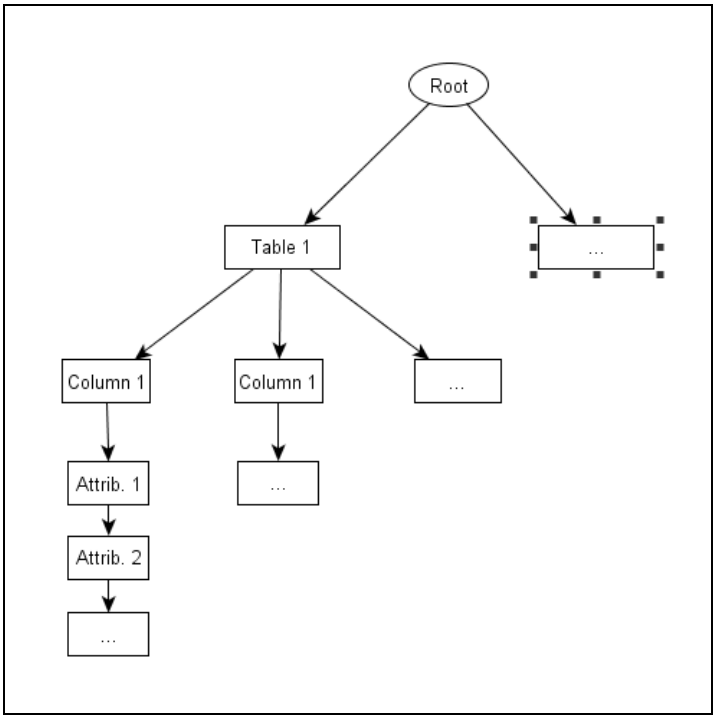

Fig. 4 In-memory database structure model.

Appendix D: Original Proposal

Tool to Automatically Generate Database Wrapper Classes

Prepared by:
Jason Kelly
Department of Computer Science
The University of Waikato

Prepared for:
Sally Jo Cunningham
Department of Computer Science
The University of Waikato

This proposal outlines the development of a software tool that would save developers time when writing database wrapper classes. When writing applications that rely on retrieving and maintaining data in a database system, developers usually write a set of reusable classes which contain all the database-specific code. Often, especially with large databases, these classes are repetitive and tedious to write. Often, because much of the code is fairly similar, developers may copy, paste and edit functions in the code, which greatly increases the chances of bugs in the code. This proposed tool would examine the Structured Query Language script file that was used to create the database, and based on the schema information, would then generate a set of wrapper classes that could be used to access the database tables. This tool would reduce the amount of developer time spent writing and debugging wrapper classes.

Introduction

In the development of information-centric applications, it is often common to embed Structured Query Language (SQL) statements within a series of reusable wrapper classes that the application logic can use for the retrieval and management of information stored in a database system. For large and complex database schemas, the wrapper classes can often be very tedious and repetitive to write. A solution to this, which this project proposes, would be to build the necessary wrapper classes automatically based on the database schema script file used to build the database.

Background

As software applications become more information-orientated, their reliance on information storage and retrieval systems grows. The demand for information storage and retrieval systems resulted in the development of database systems. While many database systems, both proprietary and open-source, have matured and become very powerful within the domain of managing information, the technology for interacting with the database systems programmatically is often complex to implement, such as with the *Open Database Connectivity* (ODBC) *Application Programming Interface* (API) (Grechanik *et al.*, 2002). Access is not standardised across the different database systems which offer different levels of ODBC compliance.

For these reasons it is better to offer the application developers a level of abstraction from the database access code by placing all database access code in a group of reusable classes in a centralised location. This, as Grechanik *et al.* (2002) also points out, helps to increase code maintainability.

Design

The proposed system will be written in the Java programming language and will thus be operating system (OS) independent to a degree – it will require a Java Runtime Engine (JRE) or other suitable Java Virtual Machine (JVM) to be installed on the computer it is running on. The proposed tool will be written using Java 5.0

language constructs, and will require a JRE of version 1.5 or greater to run. The source code generated by this tool will be compatible with a JRE of version 1.4 and greater.

The tool should run as an Eclipse IDE plug-in; however, if research shows that making it an Eclipse plug-in is too costly time-wise, then it will be developed to run as a stand-alone Java application.

It will accept, as its input, any valid SQL script file that contains SQL Data Definition Language (DDL) which creates a database and one or more tables in the database.

The system's output will include the Java source code, written to disk, for the wrapper classes which are generated from the SQL DDL script file given as the input. Several types of wrapper classes will be examined, and the one that is most intuitive and offers the best ease of maintenance will be used.

Evaluation

As this tool is intended to be used from within the Eclipse IDE, it will offer developers using Eclipse as their development tool an easy way to generate wrapper classes. This would, however, prevent other developers who do not use Eclipse from using this tool. As a stand alone application, this limitation would be avoided; however, Eclipse developers would not have the convenience of having this tool as a part of their IDE.

This tool will generate wrapper classes written in Java. Although the tool will be written to support the ability to write in different languages, for the purposes of this project only Java output will be supported.

Proposed Schedule

The table below outlines the proposed deadline dates for this project.

Task / Component	***Completion***
Research possibility of using Eclipse plug-in architecture	30/03/2006
Research SQL script parsers	2/04/2006
Implement SQL script parser	15/04/2006
Design plug-in architecture of code writer for future expansion	1/05/2006
Deliverable: Interim Report	2/06/2006
Build Java Code Writer	2/06/2006
Test and refine Java Code Writer	20/07/2006
Conference presentation draft	16/08/2006
Deliverable: Conference abstract	18/08/2006
Conference	1/09/2006
Deliverable: Final report	11/10/2006

Resources

The hardware and software resources required to develop this project are readily available. For the development of this project a computer with Java SDK versions 1.4 and 1.5 will be used. For writing and performing initial testing of the code I will use the freely available Eclipse Integrated Development Environment (IDE). I will test the generated Java source code against a number of freely available database servers, such as MySQL, PostgreSQL, and Microsoft SQL Server 2005 Express.

There already exists an open source tool which performs a similar function as would the tool this project aims to develop, although upon an initial examination it seems that this tool generates Java source code for use in enterprise web applications. Parts of the open source tool may prove to be useful in this project, so the open source tool will be examined further.

Two utilities I have developed previously will be useful in the development of this project. The first utility generated C# source code for database table wrapper classes. It is a simple utility, written in Java, which was used to write and comment

the getter and setter methods for wrapper classes of large tables (25 or more columns). The second utility, written in PHP, is a web-based tool for editing database table data. The utility dynamically builds the HTML form, and interacts with the database, based solely on an XML description of the database. If changes were made to the database, or a different database was used, all that would be required for the utility to work would be to update the XML database description.

Conclusion

This project, when completed, will provide developers with a time-saving tool that can be used to quickly generate a series of database wrapper classes based on the SQL DDL script file defining the database. The generated code will be usable immediately after generation, that is, it will retrieve data from and save data to the database without any modifications to the code. The code will also be fully documented and easy for the developers to customise should they require any code customisation.