AUTOMATIC CODE GENERATION FOR RECURRING CODE PATTERNS IN WEB BASED APPLICATIONS AND INCREASING EFFICIENCY OF DATA ACCESS CODE

J Senthil[1], S Arumugam[2], S Margret Anouncia[3] Abhinav Kapoor[4]

[1] SCSE,VIT University, Vellore-14, India

[2] Nandha College of Technology, Erode-52, India

[3] SCSE, VIT University, Vellore-14, India

[4] SCSE, VIT University, Vellore-14, India

ABSTRACT

Today, a lot of web applications and web sites are data driven. These web applications have all the static and dynamic data stored in relational databases. The aim of this thesis is to generate automatic code for data access located in relational databases in minimum time.

Keywords: Automatic code generation, Recurring code, Code generator

INTRODUCTION AND PREVIOUS WORK

The code generator made by Mr.Sergei Golitsinski which is the basis of this study was implemented in c# on the .Net platform. It generates SQL code for the database-level part of the code, and c# or VB.Net for the application level code.

There are two main approaches to code generation, often referred to as passive and active. The passive approach implies generating code only once (or re-generating it each time a modification is required). The active approach includes the option to automatically update previously generated and manually edited code. The code generator used is combination of both the approaches.
The application-level code is generated using the passive approach: the generator produces a set of classes, which contain the default data access methods.

The database-level code is generated using both approaches. The stored procedures follow the pattern similar to the application-level code: there are automatically generated procedures, which are re-generated each time the generator executes, and there are custom procedures, which are not affected by the generator. However, the tables and their structure are automatically updated. The code generator accepts as input a file with the description of the application and processes it in the following steps:

1. A *Parser* object is responsible for parsing the input and generating an parse tree. The parser is also responsible for validating the syntax and structural integrity of the schema in the input file. The objects constituting the application's abstract syntax contain detailed validation rules for each part of the application, such as checking that field lengths do not exceed their maximum values, that the data types are database-compatible, etc.

2. A *SchemaValidator* object is responsible for checking the application schema as a whole, which guards against duplicate class names, duplicate primary keys.

3. A *SchemaDatabaseLoader* object creates a *Database* object based on the schema file – which is an abstract model of the database part of the application.

4. An *SqlDatabaseLoader* connects to the application's database and does the same based on the schema retrieved from the database. The two abstract databases are compared by a *DatabaseComparer* object, which insures that the two schemas are compatible (for example, the data type of an existing field cannot be changed to an incompatible data type: a string cannot be converted to an integer, for that might result in loss of data). The D*atabaseComparer* object exposes several collections, including tables to create, tables to delete, tables to modify, constraints to create, etc., which are then accessed by objects responsible for generating the actual code.

5. A *DatabaseHelper* object takes the *DatabaseComparer* as input, generates all the database-level code, connects to the database and

updates it based on the data provided by the *DatabaseComparer*.

6. An *ApplicationLoader* object takes the parse tree as input and creates an abstract syntax tree, which is an abstract model of the application. This object is passed on to several objects, which generate the actual code. The data intensive web applications consist of three components:

1. Data Access Layer
2. Business Logic Layer
3. Presentation Layer

The application logic layer is unique for an application and does not contain recurring code patterns. Thus it is not suitable candidate for automatic code generation. Presentation layer is the user interface like desktop application. It does not require automatic code generation.

But the data access layer requires automatic code generation for producing recurring data access code which is used by different modules of the web application. Mr. Sergei Golitsinski had made a model in XML and code generator made by him generates data access code for Microsoft .NET/SQL server platform. The code generator produces at least 50% of data access code based on specifications provided in the data model but only 20-35% of data access code is used by the application.

OUR NEW APPROACH

Relational link is a pointer from one data unit to another. There is a lot of code which is unused with respect to application because code generator generates automatic recurrent code used by all applications in a website. Thus to increase the efficiency of data access code, modifications need to be made in data retrieval algorithms. Our proposal is to introduce the concept of "**Link Rank**" between the millions of link between data units in a huge database. It is based on probability distribution.

The steps involved are:

1. The data links frequently used by web application are given high ranks than the links between unused data units. For instance, if the database consist of data units A,B,C,D with B and D data links mostly used by the application.

LINKS	Number of accesses
A	2
B	6
C	1
D	9

Since, the probability of access of the data links B and D is very high as compared to other data links A and C. Hence the data links B and D are given high ranks as compared to links A and C. The link rank algorithm is implemented analogous to Google Page Rank algorithm, but it does concentrate only on inbound links and not outbound links.

2. The data links connected by higher ranks relational links are stored in separate servers with the previous servers acting as back-up store.
3. This will allow the code generator to produce data access code for frequently visited data units. Thus the efficiency of the code generated will increase manifold.
4. This will also lessen the burden of functioning on populated sites.

The java code to rank the links in java platform is given below. This is modified version of the java code generated for google page ranking by Nima Goodarzi.

```
import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import Jama.Matrix;

public class Ranking {

private final double DAMPING_FACTOR =0.85;

private List params = new ArrayList();

public static void main(String[] args) {
```

```
        Ranking ranking = new Ranking();

        System.out.print(ranking.rank("C"));

}

        /* * * Solve the equation of ax=b, which : a
is the generated matrix based on *  * the parameter
constants. x is the link ranks matrix. b is a n*1
matrix*  * which all the values are equal to the
damping factor. *  */

        public double rank(String linkId) {

                generateParamList(linkId);

Matrix a = new Matrix(generateMatrix());

double[][] arrB = new double[params.size()][1];

for (int i = 0; i < params.size(); i++) {

        arrB[i][0] = 1 - DAMPING_FACTOR;

}

Matrix b = new Matrix(arrB);

// Solve the equation and get the link ranks

Matrix x = a.solve(b);

int ind = 0;

int cnt = 0;

for (Iterator it = params.iterator(); it.hasNext();) {

String curlink = (String) it.next();

                if (curPage.equals(pageId))

ind = cnt;

cnt++;

}

return x.getArray()[ind][0];

}

/* This method generates the matrix of the linear
equations. The generated matrix is a n*n matrix
where n is number of the related pages. */

private double[][] generateMatrix() {

double[][] arr = new
double[params.size()][params.size()];

for (int i = 0; i < params.size(); i++) {

for (int j = 0; j < params.size(); j++) {

arr[i][j] = getMultiFactor((String) params.get(i),

(String) params.get(j));

}}

                return arr;}

/* This method returns the constant of the given
variable in the linear equation.*/

private double getMultiFactor(String sourceId, String
linkId) {

if (sourceId.equals(linkId))

return 1;

else {

String[] inc = getInboundLinks(sourceId);

for (int i = 0; i < inc.length; i++) {

if (inc[i].equals(linkId)) {

return -1;

}}

        /* This method returns list of the related
pages. This list is also the parameters in the linear
equation*/

private void generateParamList(String pageId) {

                // Add the starting page.

                if (!params.contains(pageId))

params.add(pageId);

                // Get list of the inbound pages

                String[] inc =
getInboundLinks(pageId);

                // Add the inbound links to the
params list and do same for inbound

                // links
```

```
for (int i = 0; i < inc.length; i++) {

if (!params.contains(inc[i]))

generateParamList(inc[i]);

                    }
          }

          / * Return list of the inbound links to a given
page.*/

private String[] getInboundLinks(String pageId) {

// This simulates a simple page collection

Map map = new HashMap();

map.put("A", new String[] { "C" });

map.put("B", new String[] { "A" });

map.put("C", new String[] { "A", "B" });

return (String[]) map.get(pageId);

}
```

CONCLUSION AND FUTURE WORK

The implementation of this system requires initial high cost of installing extra servers for backup of the least accessed data. But, the increase in the cost of new servers will be less significant as compared to higher efficiency in working of web sites and web applications. It will also improve the user interface.

REFERENCES:

Bochicchio, M., & Fiore, N. (2004). *WARP: Web application rapid prototyping.* Proceedings of the 2004 ACM Symposium on Applied Computing. (pp. 1670-1676). Nicosia, Cyprus.

Ceri, S., Fratenali P., & Bongio, A. (2000). Web Modeling Language (WebML): a modeling language for designing web sites. *Computer Networks: The International Journal of Computer and Telecommunications Networking. 33(1),*137-157.

Ceri, S., Fratenali P., & Matera, M. (2002). Conceptual modeling of data intensive web applications. *IEEE Internet Computing. 6(4),* 20-30. Chen, P. (1976). The entity-relationship model - toward a unified view of data.*ACM Transactions on Database Systems. 1(1),* 9-36.

Cleaveland, C. (n.d.). *Program Generators with XML and Java*. Retrieved April 14, 2006 http://www.craigc.com/pg/chap1.htmlCodd, E. (1970). A relational model of data for large shared data banks. *Communications of the ACM. 13(6),* 377-387.

Fratenali, P., & Paolini, P. (2000). Model-driven development of web applications: the Autoweb system. *ACM Transactions on Information Systems. 18(4),* 323-382. Glass, R. (1996). Some thoughts on automatic code generation. *ACM SIGMIS Database. 27(2),* 16-18.

Hunt, A., & Thomas, D. (2000). *The pragmatic programmer.* New York, NY: Addison-Wesley. Jacob, M., Schwarz, H., Kaiser, F., & Mitschang, B. (2006a). *Modeling and generating application logic for data-intensive web applications.* Proceedings of the Sixth International Conference on Web Engineering. (pp. 77-84). Palo Alto, CA, USA.

Jacob, M., Schwarz, H., Kaiser, F., & Mitschang, B. (2006b). *Towards an operation model for generated web applications.* Workshop Proceedings of the Sixth International Conference on Web Engineering. Palo Alto, CA, USA.

Java BluePrints (n.d.). *Model-View-Controller design pattern*. Retrieved April 12, 2006 at http://java.sun.com/blueprints/patterns/MVCdetailed.

Jensen T., Tolstrup T., & Hansen, M. (2004). *Generating web-based systems from specifications*. Proceedings of the 2004 ACM Symposium on Applied Computing. (pp. 1647-1653).

The Anatomy of a Large-Scale Hypertextual Web Search Engine, Sergey Brin and Lawrence Page {sergey, page}@cs.stanford.edu Computer Science Department, Stanford University, Stanford, CA 94305

Page Rank algorithm implemented by Developed by Nima Goodarzi

* Website: http://www.javadev.org

First Author: J.Senthil, obtained bachelors degree in Computer Engineering from Kongu Engineering College and Masters degree from Illinois Institute of Technology, Chicago, USA in Computer Science with Honors. Research interest is in Software Automation and in Pervasive computing. Currently working in VIT, as Assistant Professor Senior.

Second Author: Dr.S.Arumugam, CEO of Nandha Educational Institution.

Third Author: Dr.S.Margret Anouncia, Director, SCSE, VIT University, Vellore.

Forth Author: Abhinav Kapoor,SCSE,VIT University, Vellore.