

Specification and Automatic Code Generation of the Data Layer for Data-Intensive Web-Based Applications

Master of Science in Computer Science
Thesis Defense by Sergei Golitsinski
May 2, 2008

What is this thesis about?

Overall purpose:

Propose new approach to developing data-intensive web-based applications

Hypothesis:

It is possible to build a code generator which will significantly improve development of these apps by generating at least 50% of the data access code based on a specification of the application's data model

Testing the hypothesis:

- * design data definition language
- * develop rules for deriving required data access
- * implement code generator
- * apply approach to real-world apps and measure results

Data-intensive and web-based applications: systems, which require comprehensive data access functionality for providing web-based access to data stored in a data repository, such as a database

Today's agenda

I will discuss:

- Code generation, why it is useful and how it works
- Data definition language designed for this project
- How to derive data access methods from a data model
- Implementing a code generator
- Generating code for real applications and measuring results
- Major findings and lessons learned

I will not discuss:

- Architecture of a data-intensive web-based application
(very large topic – no time to discuss / available in thesis online)

The Hypothesis

Motivation

- Multiple recurring patterns in application development > lots of repetitive work.
- Primary motivation: search for a way to simplify development

Current research

- Most approaches: the developer is required to specify all data access functionality
- The only alternative: automatically generating the very basic operations

My big idea

- Specifying the data model of the application is enough for automatically generating most of the required data access functionality.

Hypothesis

It is possible to build a code generator which will significantly improve development of data-intensive web-based applications by generating at least 50% of the data access code based on a specification of the application's data model.

Why does code generation improve development?

Generating repetitive code is still repetitive code!

BUT: does not lead to any of the problems caused by code duplication: any edits are made to the specification – the code itself is never manually altered.

Benefits of Code Generation:

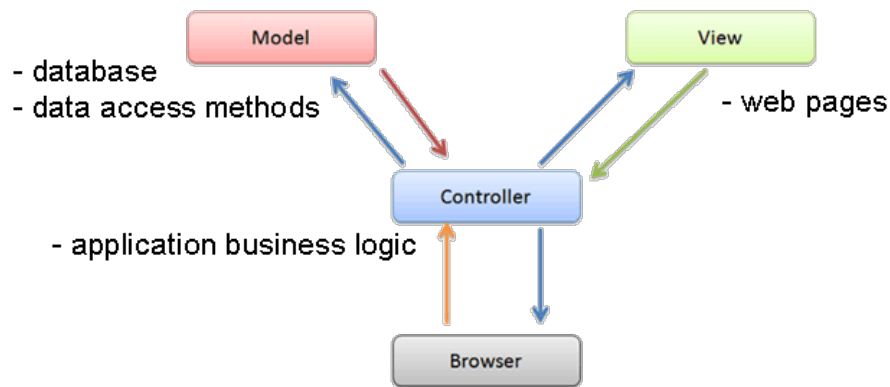
- Writing a specification is much faster than writing all the code
- Less manual refactoring: less errors
- Specifications: easier to read, write, edit, debug, and understand
- Separation of concerns
- Generate docs, tests, diagrams, etc...
- Consistency of modifications
- Correctness of generated code
- Build models and focus on areas which cannot be generated by a machine

What are we generating?

Where to start?

- Describe to the machine what exactly it must generate
- Describing is more complicated than just writing the code
- Makes sense only if we had to write the same code multiple times
- First step: **identify recurring code patterns**

Decomposing the Application: Model-View-Controller



Model = data layer

View = presentation layer

Controller = business logic layer

Business Layer

unique for each app



Presentation Layer

contains recurring patterns



Data Layer

very similar across applications



The role of modeling

Code generator: a program that translates a domain specific language or specification into application source code

Code generation:

- modeling the features to be generated
- translating the model into code

Other systems model (more or less):

- structure of underlying data: **data layer**
- data access operations: **data layer**
- navigation or hyperlink structure: **presentation layer**
- web pages: **presentation layer**

Modeling web pages:

- too much detail, the only solution – simplification of requirements

Modeling navigation:

- web pages and website navigation menu are two different systems
- website's structure becomes static

CONCLUSION: Model the data layer

How to model the data layer?

Most common approach: the **entity-relationship model** (ER): using sets and relations, model objects of the real world and their inter-relationships

However:

- no fine-grain control over database
- yet another level of abstraction - additional implementation complexity

Conclusion: use database logical model

Define data operations

```
<content>
  <object name="User">
    <attribute name="Name" type="string"/>
  </object>
</content>
<operations>
  <object name="User">
    <add/><modify/><delete/>
    <sort attribute="Name" direction="ascending">
      <filter compare="equal" attributes="Name"/>
    </sort>
  </object>
</operations>
```

1. Unnecessary to specify the obvious
2. Repetitive patterns in retrieval

Derive data operations from model

For each data object:

1. adding, modifying, reading and deleting a record,
2. reading a collection of records based on some criteria – with a record representing an entity or a relationship.

HAS NOT BEEN DONE

Data access requirements

Add, modify, display, delete a single record - *trivial*
Display multiple records – *not so trivial*

Sorting

Records must be sortable by all fields
displayed in a list

Filtering

The size of the displayed collection
may be (or should be) reduced by
entering search criteria

Paging

View collection “one page at a time.”
Becomes absolutely necessary with
large collections

Search options:

iMIS ID

School

City

State

Joined: - (MM/DD/YYYY - MM/DD/YYYY; leave blank to include all dates)

Results 1-51 of 285 Rows to display: Page 1 of 6

iMIS ID	School	City	50	Modified	Modified by	
1200422	York College of Pennsylvania	York	100	01/14/2004	CHRISG	details
1200199	Xavier University	Cincinnati	150	05/02/2002	BRENT	details
1200198	Wright State University	Dayton	200	11/28/2001	BRENT	details
1531133	Westminster College	New Wilmington	250	12/10/2003	LINDA	details
1200192	Western Kentucky University	Bowling Green	300	11/14/2001	BRENT	details
1200190	Western Illinois University	Macomb	350	12/05/2001	BRENT	details
1200191	Western Carolina University	Cullowhee	400	03/28/2003	BRENT	details
1200189	West Virginia University	Morgantown	450	09/18/2003	BRENTA	details
1200188	West Virginia State University	Institute	500	11/21/2005	BRENT	details

Account

Department

Group

Last name

Results 1-50 of 357 sorted by Full Name Page 1 of 8

Selected	Full Name
<input type="checkbox"/>	Abraham, Fred
<input type="checkbox"/>	Ackerson, Debra
<input type="checkbox"/>	Adams, Becky
<input type="checkbox"/>	Agbese, Pita
<input type="checkbox"/>	Alicia, Auditors
<input type="checkbox"/>	Allen, Ben
<input type="checkbox"/>	Alper, Sandra
<input type="checkbox"/>	Andersen, Kim

Data model specification

```

Application (1)
  Namespace (1 or more)
  Name (1)
  Class (1 or more)
    Name (1)
    Type (1): record, link, readonly, final
    Table (1)
      Name (1)
      External (0 or 1): true, false
      Field (1 or more)
        Name (1)
        SqlDataType (1)
        Identity (1): true, false
        PrimaryKey (1): true, false
        ForeignKey (0 or more)
          RefTable (1)
          RefField (1)
        Unique (0 or 1): true, false
        Encrypted (0 or 1): true, false
        Display (0 or 1)
        ExcludeFromTable (0 or 1): true, false
        IncludeWithParentTable (0 or 1): true, false
        IncludeInList (0 or 1): true, false
        DefaultSort (0 or 1): true, false
        ReadonlyType (0 or 1): created, modified, timestamp
    AdditionalField
      Name (1)
      SqlDataType (1)
      Sql (1)
      SortExpression (1)
      Display (0 or 1)
      ExcludeFromTable (0 or 1): true, false
      IncludeWithParentTable (0 or 1): true, false
      IncludeInList (0 or 1): true, false
      DefaultSort (0 or 1): true, false
    AdditionalSproc (0 or more)
      Name (1)
      Param (0 or more)
      -- ...

```

```

<class>
  <name>User</name>
  <type>record</type>
</class>
<class>
  <name>Permission</name>
  <type>readonly</type>
</class>
<class>
  <name>UserPermission</name>
  <type>link</type>
  <table>
    <field>
      <name>UserId</name>
      <primaryKey>true</primaryKey>
      <sqldatatype>int</sqldatatype>
      <foreignkey>
        <reftable>User</reftable>
        <reffield>Id</reffield>
      </foreignkey>
    </field>
    <field>
      <name>PermissionId</name>
      <primaryKey>true</primaryKey>
      <sqldatatype>int</sqldatatype>
      <foreignkey>
        <reftable>Permission</reftable>
        <reffield>Id</reffield>
      </foreignkey>
    </field>
  </table>
</class>

```

etc...

Defining data access methods

Problems:

1. attributes which are generated or updated automatically
2. weak entities
3. different sets of fields for collections of records

Solutions:

1. “read-only” field types are treated in a special way
2. “delete children” parameter in delete method
3. special field attributes: ExcludeFromTable, IncludeWithParent, etc...

Defining the set of methods:

1. Decompose into 5 types of data access:
 - Instance-related for data objects (retrieve, update)
 - Non-instance-related for data objects (getRecords, delete)
 - Non-instance-related for data objects for each one-to-many relationship
 - Non-instance-related for data objects for each many-to-many relationship
 - Non-instance-related for data object links for each many-to-many relationship
2. Generate specific methods for each type

List of Generated Data Access Methods

Instance-related data object functionality

- get record
- update record

Non-instance-related data object functionality

- create new record
- delete record
- get list
- get records
- get records with paging
- get records with paging and a filtering criteria

Non-instance-related data object functionality for each one-to-many relationship:

- get records by relationship
- get records by relationship with paging
- get records by relationship with paging and a filtering criteria

Non-instance-related data object functionality for each many-to-many relationship:

- get records by link
- get records by link with paging
- get records by link with paging and a filtering criteria
- get links
- get links with paging
- get links with paging and a filtering criteria

Non-instance-related functionality for each many-to-many relationship:

- create link
- create all links by first data object
- create all links by second data object
- delete link
- delete links by first data object
- delete links by second data object

Implementing the code generator

Approaches to code generation:

- Passive: generates code only once (or re-generates each time)
- Active: updates previously generated and manually edited code

My code generator implementation:

Application-level: passive. For manual edits, create classes extending generated classes

Database-level: combination of both

The code generation process:

Accepts a file with the description of the application and -

1. A Parser parses input and generates a parse tree. Validates the syntax and structural integrity of the schema in the input file
2. A SchemaValidator checks the schema as a whole, guarding against duplicate class names, duplicate primary keys, maintaining correct references in foreign key descriptors, etc.
3. A set of objects load the current database schema, compare it with the new schema and update the database
4. An ApplicationLoader object takes the parse tree as input and creates an abstract syntax tree, which is passed on to objects, generating the code

- Implemented in c# on the .Net platform. Generates SQL, and c# or VB.Net

The “real world” applications

1. Witness Identification

Used in criminology for eyewitness identification. A user (a witness) is presented with a sequence of head shots of suspects, selected from a set of several hundred thousand images

Main challenge: manipulation of a very large set of data

Eyewitness Identification System		
Main Menu		
Lineup Administration	Photo Database	System Settings
Administer lineup	Photos	Users
View lineup results	Race types	Roles & permissions
	Hair colors	Lineup text
Cases & Lineups	Weight ranges	
Cases	Age ranges	
Suspect profiles		
Lineups		

The “real world” applications

2. Account Reporting

Provides university's constituents with access to various university accounts

Main challenge: uses multiple databases and requires elaborate data access functionality to generate complex data reports

The screenshot shows the 'System Administration' page for the 'UNI Foundation Statement of Account'. The 'Accounts' section is active, displaying a search form and a table of accounts. The search form includes fields for 'Admin Unit', 'Number', and 'Description', and a dropdown for 'Account Types' with options 'Alumni Association', 'Foundation', and 'All'. Below the search form are buttons for 'View Info', 'Set Users', and 'View Users'. The table below shows a list of accounts with columns for 'Full Account Number', 'Account Description', 'Administrative Unit', 'Account Status', 'Modified', and 'Modified By'.

Full Account Number	Account Description	Administrative Unit	Account Status	Modified	Modified By
F21.0270.00.00	Teaching Careers In The Elem. Grades Quasi Endowed Fund	222	active	03/20/2008	Sharon Hannasch
F21.0276.00.00	SOM Benefit Concert Endowed Scholarship	236	active	03/05/2007	Ann Delphin
F21.0277.00.00	Kappa Delta Pi Quasi Endowed Scholarship	221	active	03/19/2008	Sharon Hannasch
F21.0278.00.00	Tom Pettit Scholarship	232	active	11/21/2005	Susan Sayer
F21.0279.00.00	Martha Ellen Tye Endowed Art Scholarship	231	active	09/27/2006	Molly Wilson
F21.0281.00.00	Katherine S Humphrey Memorial Scholarship	214	active	08/14/2007	Sharon Hannasch
F21.0288.00.00	Stanley & Norma Reeves Scholarship Quasi-Endowment	222	active	03/17/2008	Sharon Hannasch
F21.0289.00.00	David Kennedy Memorial Scholarship	236	inactive	03/23/2007	Sharon Hannasch

The screenshot shows the 'System Administration' page for the 'UNI Foundation Statement of Account'. The 'Roles' section is active, displaying a table of roles and a modal window for setting role permissions. The table below shows a list of roles with columns for 'Description' and 'Rank'.

Description	Rank
Deans	0
Idle	0
Off Campus Users	0

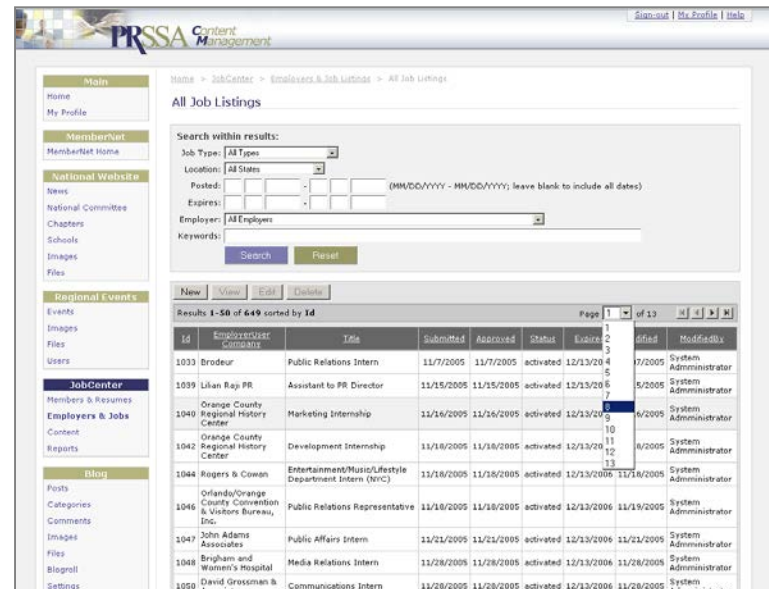
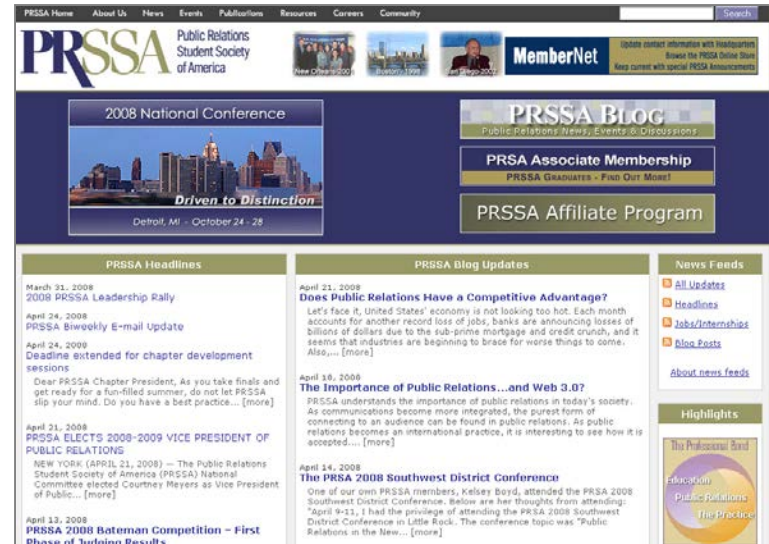
The modal window 'Set role permissions for Off Campus Users' is open, showing options to select a category and email permissions. The 'Email' category is selected, and the 'Email: view archive' and 'Email: send & delete' permissions are checked. The 'Save' button is visible.

The “real world” applications

3. PRSSA

Collection of web sites with a complex content management system, including regular web sites, a blog, a career web site and numerous administrative functionality

Main challenge: the amount of different features



Results

Scope of application
and amount of generated code

	Witness Identification	Account Reporting	PRSSA
Data objects	16	14	48
Data object links	4	8	11
Data object attributes	92	97	449
Generated database tables	20	22	59
Generated stored procedures / lines of code	251 / 8,372	315 / 8,899	577 / 26,072
Generated classes / lines of code	70 / 9,956	116 / 17,181	204 / 34,397
Total generated lines of code	18,328	26,080	60,469

Effectiveness

What part of the application's data access code
was generated

	Witness Identification	Account Reporting	PRSSA
Lines of code: (stored procedures + code in files)			
- total	18,410	30,710	63,228
- generated	18,328	26,080	60,469
- percentage of generated	99%	85%	96%
Stored procedures:			
- total	257	375	658
- generated	251	315	577
- percentage of generated	98%	84%	88%

Efficiency

What part of the generated data access
code was used in the application

	Witness Identification	Account Reporting	PRSSA
Stored procedures:			
- generated	251	315	577
- generated and used in application	88	63	179
- percentage of used	35%	20%	31%

Concern: 12,000 + 21,000 + 42,000 lines of generated code useless!

Conclusion: hypothesis supported in part:

- more than 50% of data access code was generated
- development was not improved as expected due to added complexity

Lesson learned / Further research

Observed patterns:

- Single-object methods: add, retrieve, modify, delete are always used
- Only half of data object link methods are used (based on one of the 2 objects)
- When a collection is retrieved with paging, retrieving it without paging - only as a minimized list

Possibilities for improvement:

- Better XML syntax: attributes vs. elements
- Using values for derived fields
- Data views to specify structure of collections
- Intermediate code representation
- Code templates

```
<class>
  <name>User</name>
  <type>record</type>
  <table>
    <field>
      <name>Id</name>
      <datatype>int</datatype>
      <identity>true</identity>
      <primaryKey>true</primaryKey>
    </field>
  </table>
</class>
```

```
<class name="User" type="record">
  <table>
    <field name="Id" datatype="Id" identity="true" primaryKey="true"/>
  </table>
</class>
```

Main Lesson Learned: Simplicity Versus Flexibility

- a) Flexible, yet complex system allows the specification of numerous criteria
- b) Rigid, yet simple system, has most of the options hard-coded

This experiment has proved that “keeping it simple” is a better approach

Questions, please?

Thesis and code available at lordofthewebs.com

