5-2011

A Comparative Study of the Systematic Mapping of Object-Oriented Models to Code Development Frameworks

Martin Hristov Georgiev
The College at Brockport, mgeorgiev@acm.org

A Comparative Study of the Systematic Mapping of Object-Oriented Models to Code Development Frameworks

A Senior Honors Thesis

Presented in Partial Fulfillment of the Requirements
for Graduation in the College Honors Program

By

Martin Hristov Georgiev

Computer Science & Mathematics Major

The College at Brockport, State University of New York
May 2011

Thesis Director: Dr. Sandeep Mitra, Associate Professor

Department of Computer Science

Acknowledgements

The author expresses his appreciation to Dr. Sandeep Mitra from the Computer Science Department at The College at Brockport, State University of New York for taking the time and providing a close guidance throughout the analysis, design and development stages of the case study as well as for revising this paper for consistency. Further, the author thanks Dr. Mitra for introducing him to the software engineering field, in addition to providing him with invaluable academic and career advisement throughout the years.

The author also thanks Phyllis Griswold, Claire VanDerBerghe and Rick Kincaid from the Career Services Department at the college for working closely with us throughout the development of the Career Advisor Contacts Database System, constantly revising development progress and providing us with timely feedback. Further, the author thanks Dr. Thambrahalli Rao from the Computer Science Department at the college for clarifying important software design patterns needed in the implementation phase of the case study. Special thanks are also extended to PhD candidate Tsvetomira Radeva and Dr. Srikanth Sastry from MIT, Momchil Kyurkchiev from Google Inc., and the Open Source community for their generous help during the implementation stage of the case study.

Finally, the author thanks Steven Lewis, Gian Carlo Cervone and the Web Services Department at the college both for their assistance in revising the database schema of the application and for taking the responsibility to maintain and extend the application in order to accommodate future needs of the college and its community.

Table of Contents

List of Abbreviations.....4

List of Figures, Tables and Code Samples.....5

Abstract.....6

1: Introduction.....7

2: Case Study – Career Advisor Contacts Database System.....9

2.1: Case Study – Business Requirements.....9

2.2: Case Study – Analyses.....10

2.3: Case Study – Design.....11

2.4: Case Study – Implementation.....12

3: Framework Comparison.....13

3.1: Framework Comparison – Back-End.....13

3.1.1: CakePHP.....13

3.1.2: Google Web Toolkit.....15

3.1.3: "Plain-old Java" on the desktop – via RPC.....18

3.2: Framework Comparison – Middle-Tier.....18

3.2.1: CakePHP.....18

3.2.2: Google Web Toolkit.....23

3.2.3: "Plain-old Java" on the desktop – via RPC.....25

3.3: Framework Comparison – Front-End.....26

3.3.1: CakePHP.....26

3.3.2: Google Web Toolkit.....28

3.3.3: "Plain-old Java" on the desktop – via RPC.....29

4: Status of Implementation.....31

5: Conclusion.....32

6: References.....33

List of Abbreviations

AJAX – Asynchronous JavaScript with XML

CLU – Common Library Utility

CSS – Cascading Style Sheets

GRASP – General Responsibility Assignment Software Patterns

GUI – Graphical User Interface

GWT – Google Web Toolkit

HTTP – Hypertext Transfer Protocol

MDA – Model-Driven Architecture

MVC – Model-View-Controller

PIM – Platform Independent Model

PSM – Platform Specific Model

RPC – Remote Procedure Call

SEEM – Software Engineering Effectiveness Model

UML – Unified Modeling Language

List of Figures, Tables and Code Snippets

Figure 1: The Standish Group: Chaos Summary 2009.....7
Figure 2: Use case workflow: Record a Registration Request – Alumnus.……..10
Figure 3: Analyses Level Sequence Diagram: Record a Registration Request – Alumnus.……..11
Figure 4: Design Level Sequence Diagram: Record a Registration Request – Alumnus.……..12
Code Snippet 1: CakePHP Database Configuration Tool.……14
Table 1: CakePHP – Back-End Database Analysis: Summary.……..15
Code Snippet 2: GWT Database Configuration – MySQL.……..16
Table 2: GWT – Back-End Database Analysis: Summary.……..17
Code Snippet 3: CakePHP – Internal Object Organization.……..19
Figure 5: Record a Registration Request – Alumnus.……..22
Table 3: CakePHP – Middle-Tier Analysis: Summary.……...23
Figure 6: GWT - Object Organization.……...24
Table 4: GWT – Middle-Tier Analysis: Summary.……...25
Figure 7: MVC Architecture.……..27
Table 5: CakePHP – Front-End Analysis: Summary.……..28
Table 6: GWT – Front-End Analysis: Summary.……...29
Table 7: “Plain-old Java” on the desktop - via RPC – Front-End Analysis: Summary.……...30

Abstract

Despite recent advances in Software Engineering, the ‘software crisis’ persists. Researchers have explored the concept of Model-Driven Architectures (MDA) to obtain a high-level view of a software application in a technology-independent manner. These constitute what are known as Platform-Independent Models (PIMs)). Thereafter, a PIM’s features are systematically mapped to an implementation environment features (creating a Platform-Specific Model (PSM)). Most MDA techniques focus on the structural aspects of the system under construction. Little attention is paid to mapping behavior models to implementation technologies. This paper presents results from an approach that focused on modeling the behaviors of single-threaded, GUI-on-database systems using UML Sequence and State diagrams, and then systematically mapping these to desktop-based and web-based implementations. Well-known GRASP principles, especially the ‘Information Expert’ and the ‘Front Controller’ concepts, are applied to obtain a high-quality behavioral model. Thereafter, precise mapping techniques from model to implementation environment constructs are devised. Our results demonstrate that the mapping from the same model to different environments is strongly influenced by the features of the environment itself. We present our results in the context of a case study for a real-world customer, implemented using the following platforms: “Plain Old Java” on the desktop, Google Web Toolkit (GWT) and a Model-View-Controller (MVC) framework using PHP - CakePHP. We present several important results. We have learned that a model ‘Front Controller’ object that serves as the primary interface to the user, and is independent of the back-end database tables it interfaces with, can be easily mapped to appropriate constructs in Java-based desktop and GWT implementations. However, CakePHP’s naming conventions make this mapping indirect, and may thus break the ‘traceability’ from model to implementation. Our various results describe a means to measure the extensibility and maintainability of such implemented systems.

Keywords: Behavior Mapping, Frameworks, Case Study, Google Web Toolkit, CakePHP, Java, PHP

1: Introduction

Designing easily traceable, maintainable and extensible software applications is the software engineers' number one priority. This has led to the development of a number of software design methodologies, such as the Waterfall model, the Rational Unified Process (RUP), the various Agile methodologies and the locally-used methodologies, such as the Software Engineering Effectiveness Model (SEEM), among others [1][2][3][4]. Furthermore, several architectural approaches towards creating models have been developed over the past few years (e.g. GRASP). The application of such principles to the modeling process standardizes the nature of the model and enhances its quality. The quality of the model facilitates the mapping of such models to implementation code [1][5]. Despite these developments over the past half a century, recent research on the success rate of software projects shows that the software crisis continues to exist [6]:

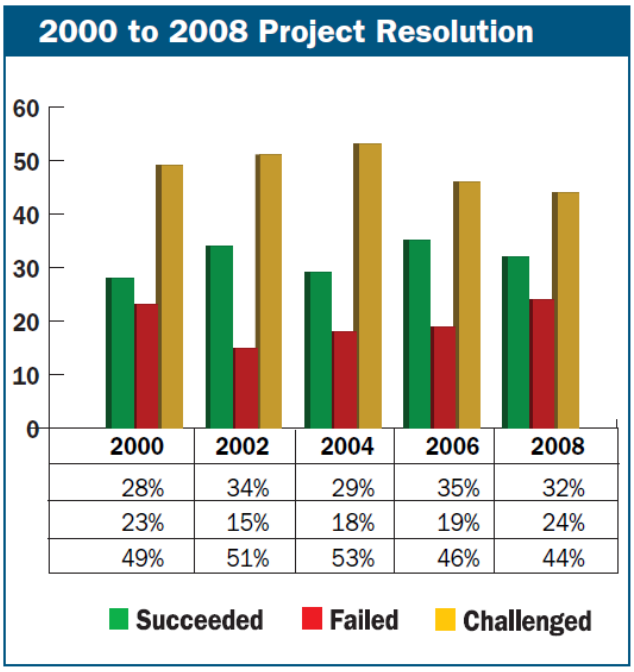

Figure 1: Credit: The Standish Group: Chaos Summary 2009

Keeping the recommendations made by the Standish Group in their latest report in mind, we evaluated several software design methodologies. We identified the locally-developed Software Engineering Effectiveness Model (SEEM) as highly appropriate, mainly due to its focus on behavior modeling [1]. Many of the other model-driven architectural approaches we analyzed indicated that they primarily focus on structure, with behavior being a secondary consideration.

Therefore, we hypothesized that a new approach to system development is needed – namely, one that seeks to ensure the traceability of models to code, and thus enhance the maintainability and extensibility of code. With such an approach, future object-oriented software systems may help to address the ongoing software crisis [6].

In this paper, we use a “real world” project as a case study to validate our research ideas – i.e. a project carried out for a real customer who wishes the developed system deployed for use on a day-to-day basis. The project involved the creation of a Career Advisor Contacts Database System for The College at Brockport, State University of New York. The goal of our research is to estimate the efficacy of the mapping of object oriented behavioral models to code development frameworks. We begin the system design by creating a Platform Independent Model (PIM) using UML sequence diagrams. Since care is taken to ensure that the model is independent of any specific implementation environment, the creation of such PIM allows us to achieve a layer of abstraction above the underlying technology. Then, working from the PIM, we create the platform specific model (PSM), which is tightly associated with the underlying development framework. Unlike other MDA approaches, which focus on the structure of the system, our primary focus is on behavior. We wish to model the end-to-end flow of data – from the user to the back-end (database) and back. We make extensive use of two well-known GRASP principles – The Information Expert and the Front Controller. We aim to achieve very low coupling and high cohesion, allowing us to leverage the system's extensibility and maintainability. Lastly, we base the conclusions of our comparative analysis on three separate implementations of the same software system: Java on the desktop - utilizing the Java Swing library and the Remote Procedure Call (RPC) to exchange data between the client side and the server side, Java on the web – using Google Web Toolkit (GWT), and PHP on the web – using CakePHP.

Evaluating the various implementation environments used in our case study, we realized that we can identify the three major tiers of a typical client-server system: the back-end (database), the middle-tier (server side), and the front-end (client side).

At the back-end, we are mostly concerned with: the ease of connecting the application’s framework to the underlying database, the use of naming conventions to facilitate code

development and subsequent code maintenance, and the management of the movement of data to and from the database – namely, retrieving data from the database, persisting new data to the database and updating already existing data into the database.

In the middle-tier, our primary focus is on code reuse, and on lowering the coupling and increasing the cohesion between the implementation constructs. This not only facilitates robust system development, but also enables us to ensure easy traceability, maintainability and extensibility of the software system.

At the front-end, we address the problem of decoupling the GUIs used by the application from the underlying database schema. Further, we discuss potential application dependencies on technologies, such as AJAX/JavaScript, and how they could hinder the application's availability.

2: Case Study - Career Advisor Contacts Database System

In order to present our research ideas better, we focus on our case study extensively in the remainder of this paper. We outline the design and development of a real world object-oriented software system on three distinct platforms. The main objectives of our system under consideration – i.e. the Career Advisor Contacts Database System for the Career Services Department at The College at Brockport are: first, allow alumni and friends (non-alumni) of the college to register with the system and volunteer to be mentors; second, allow current students at the college to query the database and retrieve the contact details of prospective mentors; third, enable the Career Services Department at the college to act as a “man in the middle” and facilitate the communication between current college students and their prospective mentors.

2.1: Case Study – Business Requirements

The main functionality required from the system can be outlined as the following set of disjoint workflows:

- Record/Approve a registration request from an alumnus/a non-alumnus
- Register/Approve an update request from an alumnus/a non-alumnus
- Add a new alumnus administratively
- Update/Delete an alumnus/a non-alumnus administratively
- Department/System administrator searches alumni records
- Current student searches alumni/non-alumni records

2.2: Case Study – Analyses

A detailed description of the 'Record a Registration Request – Alumnus' workflow is presented in the use case workflow shown in Figure 2:

Use Case Name:	**1. Record a Registration request from a new alumnus**
Description:	
A Secretary records a registration request from a new alumnus	
Preconditions:	
1. Alumnus has a valid e-mail address. 2. Alumnus has information about ALL Brockport degrees he/she obtained (or worked on), including semester of graduation (last semester attended)	
Workflow:	
1. Alumnus approaches Secretary with a request to register. 2. Secretary provides Alumnus with the registration form. 3. Alumnus provides the following information on the registration form: a. Full name (prefix, first, middle, last, suffix) b. Mailing address c. E-mail address d. Phone number(s) (provide 0 to 2 phone numbers) e. Designation of the primary mode of contact (by default, this will be e-mail) (*NOTE: Primary mode of contact cannot be empty – e.g., a non-provided phone number*) f. Privacy designations for each of the following: e-mail address, mailing address, phone numbers (*NOTE: Primary mode of contact cannot be private*) g. For ALL degrees obtained (worked on) at Brockport: Major, Degree (Bachelor's/Master's), last semester attended (*NOTE: There must be at least one of these*) h. (Optional) For additional degrees obtained elsewhere: Major, Degree (Bachelor's/Master's/Ph.D.), last semester attended, Name of institution i. (Optional) For all jobs worked: Name of employer, Field of employment (chosen from a set of standard designations used by Career Services), Start date, end date (*NOTE: Start date must be earlier than end date*) j. (Optional) Willingness to help Brockport - chosen by selecting from a set of designated alumni "help categories" k. (Optional) Advice to Brockport (up to 500 characters) l. (Optional) Brockport-provided ID (e.g., Banner ID) m. Unique user name for later use (e-mail address suggested) n. Password for identification (provide twice) 4. Secretary verifies that all the obligatory pieces of information are provided on the form, the constraints shown in italics above are met, the user name is unique, and that both the password entries are the same. If a Brockport-provided ID is given, Secretary also verifies that this is unique. 5. Secretary files the form in the "Registration request" ledger(s). 6. Secretary informs the alumnus that they will hear (via e-mail) from the Administrator if their request is approved.	
Results:	
A new "Registration Request" is filed in the appropriate ledger(s)	
Alternates:	
Alumnus' registration request is rejected for one or more of the following reasons: obligatory information not provided, constraints provided in italics above are not met, user id is not unique, password verification technique fails, Brockport-provided ID is not unique.	
Entities Involved:	
Alumnus, Secretary, Alumnus registration data, Registration request ledger(s)	

Figure 2: Use Case Workflow: Record a Registration Request – Alumnus

In the next step, we used UML to create a platform independent model, consisting of a set of analysis sequence diagrams for the workflows presented above [7][8]. At this step, we aim for higher level of precision in the model and seek to eliminate any remaining ambiguities resulting from the use of natural language in the use case workflow descriptions. We should emphasize that we create a technology-independent model in a manner that enables it to serve as the foundation of developing a technology-dependent model in the next step. The analysis sequence diagram for the 'Record a Registration Request – Alumnus' is presented in Figure 3:

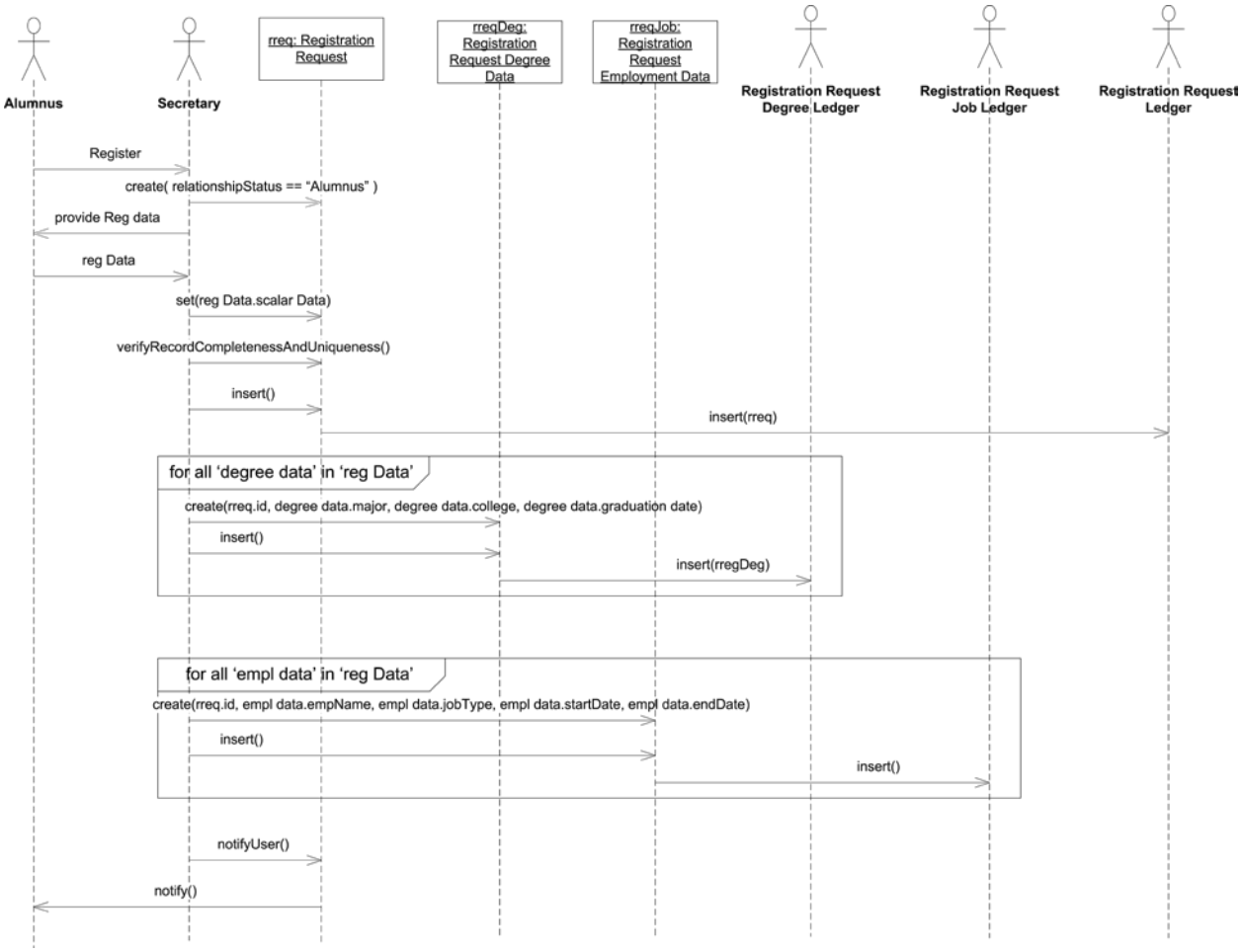

Figure 3: Analyses Level Sequence Diagram: Record a Registration Request – Alumnus

2.3: Case Study – Design

Our first step in this phase is to create design level sequence diagrams for all of the analysis level sequence diagrams. During this phase, we consider the MVC architecture, which is incorporated into many implementation frameworks [11][12]. Design level sequence diagrams

are created by enhancing the analysis level diagrams with framework-mandated MVC entities, such as views and controllers. These diagrams are still platform independent to a large extent. They only include entities which are specific to a particular architecture (i.e. MVC). MVC is a generic architecture applicable to various different frameworks. Nevertheless, using MVC enables the modeler to show the end-to-end flow of data – from the human user using the view at the front-end to the back-end database and vice versa. The design level sequence diagram for the 'Record a Registration Request- Alumnus' is presented in Figure 4:

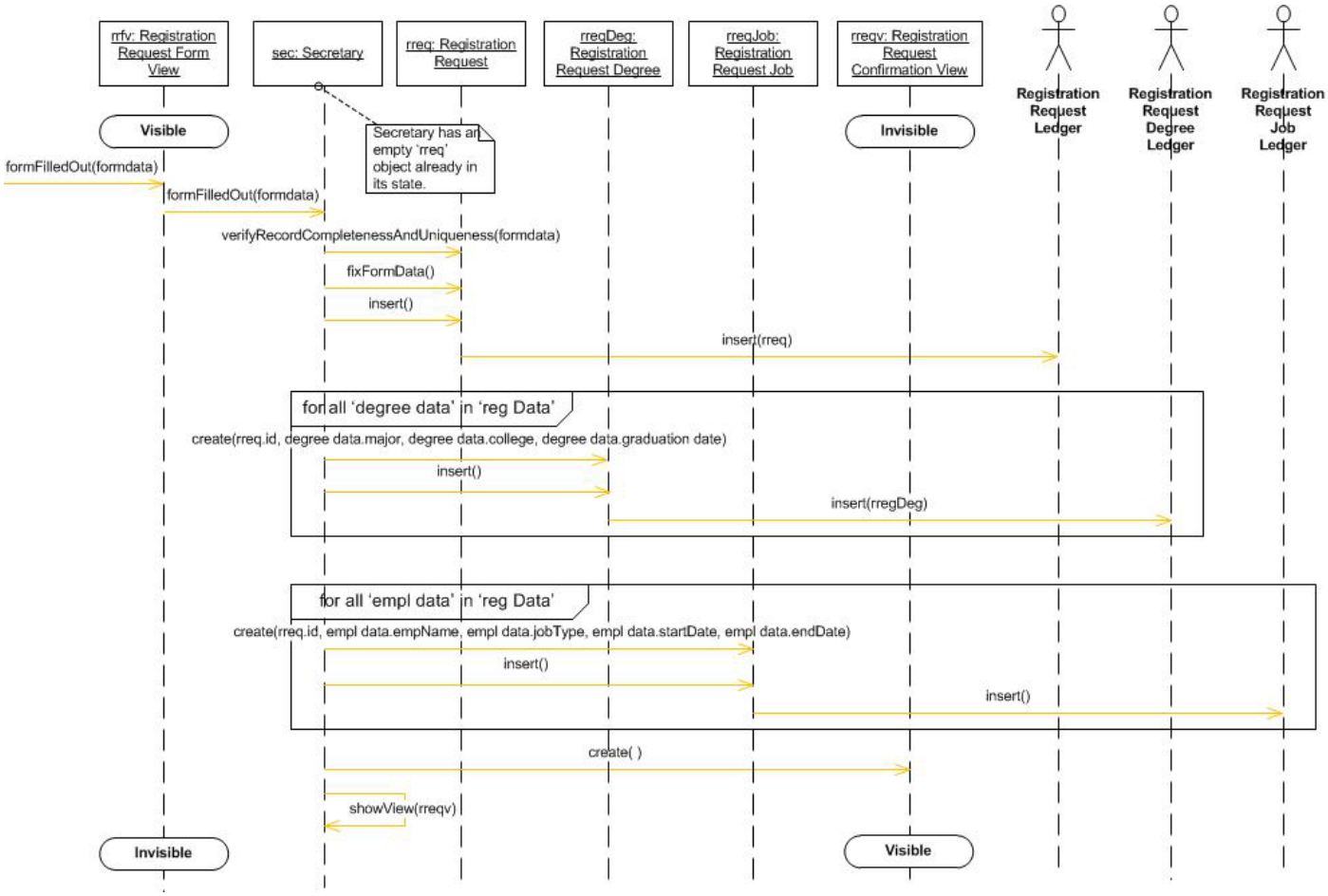

Figure 4: Design Level Sequence Diagram: Record a Registration Request-Alumnus

2.4: Case Study – Implementation

Having built the full and complete PIM, we move onto creating the PSM. At this stage we aim at creating three distinct implementations of our case study and compare their advantages and disadvantages. Our main objective here is to systematically map PIM to PSM using direct

mapping and data encapsulation in order to achieve code separation based on behavior, thus reducing coupling and enhancing cohesion [10]. This will not only give us a solid base for our comparison study but also help us devise usability guidance for each individual framework [5]. We chose these frameworks – “Plain-old Java” on the desktop – Java Swing over RPC, Google Web Toolkit and CakePHP – because they claim to natively support the object-oriented paradigm and the MVC architecture [11][12]. Further, all three implementation environments are supported by three very well established organizations: Oracle, Google and MIT. Therefore, they are likely to stay available and supported in the next several years. Last but not least important, all three environments are free, which makes them highly preferable by all organizations across the industry.

3: Framework Comparison

Since all of the frameworks are based on the MVC architecture, the three distinct tiers of the system: the back-end – database, the middle-tier – server side, and the front-end – client side – are present in all of our implementations. Our comparison study, and the conclusions we derive, are based on our careful analysis of these three packages.

3.1: Framework Comparison – Back-End

At the back-end, we compare both the flexibility and the amount of support each individual framework provides. Some of the database technologies we consider are: MySQL, PostgreSQL, Oracle and DB2. As stated above, we analyze the ease of connecting the code components of each framework to the database associated with our application. We also investigate the use of naming conventions to facilitate code development and subsequent code maintenance. Lastly, we address the data management problem – i.e. the means of retrieving data from the database, persisting new data to the database and updating already existing data into the database.

3.1.1: CakePHP

CakePHP provides native support for a wide variety of database technologies, such as MySQL, MySQLi, SQLite, PostgreSQL, DB2, Oracle, and Firebird, among others [11]. Further, it allows the developer to create custom database drivers and in this way extend the set of

supported database engines. Thus, from a database technology perspective CakePHP provides great flexibility at minimum cost.

Due to the built-in database configuration tool, connecting the CakePHP framework to the back-end (database) of the software application is greatly facilitated. In fact, configuring the database simply requires the location of the database server, the username and the associated password for accessing the database [11]. Code Snippet 1 shows such a development configuration for a MySQL database:

```
class DATABASE_CONFIG
 {
        var $default = array(
                                'driver' => 'mysql',
                                'persistent' => 'true',
                                'host' => 'csdb.brockport.edu',
                                'login' => 'mgeorgiev',
                                'password' => '*ezer0K',
                                'database' => 'careerservicescontacts',
                                'prefix' => ''
        );
}
```

###################### CODE SNIPPET 1: CakePHP Database Configuration Tool ######################

Once the configuration settings are provided, the user can trigger the database configuration tool by going to the index page of the application and check if it can connect to the database. On this page the user is notified in real time for any potential problems. Therefore, all data access problems can be resolved at configuration time, rather than at development time.

Application development in CakePHP is further facilitated via the `bake` built-in tool [11]. However, in order to be able to take advantage of the automatic code generation, we need to follow all framework conventions across all layers of the application. For instance, table names should be plural and in the C-style format, such as `registration_requests`, rather than in the camel case format – `registrationRequest` or `registrationRequests`. Therefore, CakePHP ensures rapid application development at the expense of strictly defined conventions (mainly in the set of

classes that are created, and the names used for naming these classes, their attributes and methods) over configuration.

Data management in CakePHP is also automatic when `bake` is used to generate the application's skeleton [11]. In fact, `bake` generates all functions associated with saving, editing and viewing the application's data. Further, `bake` generates most of the code required for validating the application's data [11]. Therefore, retrieving data from the database, updating already existing data in the database, and persisting new data to the database is natively supported by the framework - again at the expense of strictly defined conventions over configuration. It is important to note here that none of the `bake` functionality is available if the CakePHP conventions are not strictly followed.

Based on the analysis of the back-end (database) support in CakePHP, we can summarize our conclusions in Table 1:

Support of database technologies	MySQL, SQLite, PostgreSQL, DB2, etc.
Automatic Code Generation at the Model Layer	Yes
Automated Data Management	Yes
Ease of Connecting the Framework to the Database	Very Easy – Using the Built-in DB Tool
Strictly Defined Conventions	Yes – C- Style names
Data Validation	Automatically Provided Via `bake`

Table 1: CakePHP – Back-End Database Analysis: Summary

3.1.2: Google Web Toolkit (GWT)

In contrast to CakePHP, Google Web Toolkit (GWT) does not provide native support for any database technology [12]. However, GWT allows the developer to manually configure the database connection via the JDBC connector, Hibernate, JPA or some other framework that supports persistence of data. Therefore, MySQL, Oracle, PostgreSQL and DB2, among others, can all be used to store application data. Unfortunately, the flexibility allowed by GWT in this context comes at the expense of a certain amount of code complexity, which in turn may prolong the application development time.

Regardless of the database technology being used, GWT does not have a simple built-in database configuration tool, as CakePHP does [11][12]. Setting the database access requires multiple changes in several files, as described below. Additional libraries may also be required in

the build path, depending on the type of database connection. Hence, it becomes the developer's responsibility to verify that all required components are available and that the application can connect to the database successfully.

One of the simplest databases to connect to using GWT is MySQL. Code Snippet 2 shows the minimal configuration necessary to achieve this connection. Further, it depicts the behavior the remote service (i.e. the server side component) needs to implement in order to obtain a connection to the back-end (database):

```
public class MySQLConnection extends RemoteServiceServlet implements
DBConnection {
        private Connection conn = null;
        private String status;
        private String url  =
                  "jdbc:mysql://csdb.brockport.edu/careerservicescontacts";
        private String user = "mgeorgiev";
        private String pass = "*ezer0K";

        public MySQLConnection(){
                try{
                        Class.forName("com.mysql.jdbc.Driver").newInstance();
                        conn = DriverManager.getConnection(url, user, pass);
                }
                catch (Exception e){
                        //Abort
                }

        }
}
```

###################### CODE SNIPPET 2: GWT Database Configuration – MySQL ######################

In addition to the Java code in code snippet 2, the developer should also make required changes in the web.xml file under the /war/WEB-INF folder in order to make the GWT framework 'aware' of the MySQLConnection servlet. This includes mapping the URLs the user will use to interface to it. Furthermore, the remote service servlet needs to use other JDBC constructs – like Statement – to interface to the database. In most cases SQL code needs to be written explicitly in order to retrieve/insert/update/delete data (unlike CakePHP, which provides built-in functions to do this job) [11][12]. Results from queries come back to the application using the JDBC ResultSet data structure, which has a complex API to master and use. In contrast, in CakePHP the result comes back in an associative array, which is the standard data structure used by all other parts of the application [11]. Using a tool, such as Hibernate or JPA, in GWT can help to reduce this complexity somewhat. We do not discuss these details here due to space

considerations. However, the user has to carefully decide upon using one of these tools and manually tailor the rest of the code to use the same tool properly. In CakePHP, the one standard way of interfacing to the database allows for automatic code generation.

Code generation at the application model level is not supported by GWT [12]. This "inconvenience" provides us with flexibility – in GWT we are not constrained by any naming convention for the database tables and table fields. Camel-cased names are recommended, per Java conventions, but not enforced. Although this flexibility comes at the expense of increased development time, this architecture facilitates the ability to re-target our application to other legacy systems (components that interface to the back-end database and encapsulate behavior we can reuse) or other persistence frameworks. We are clearly aware of the parts of the GWT application that need to be changed to use these other systems/frameworks.

Data manipulation is not managed automatically in GWT, as opposed to CakePHP [11][12]. All data validation requirements and enforcing them are the developer's responsibility. Failure to implement data validation procedures could create security issues, as improper data may corrupt the database and/or be a security risk [9][12]. One can summarize the GWT approach to the management of the back-end by stating that in the context of relational databases the developer is required to provide the object to relational model mapping mechanism. While this mapping is influenced in part by the APIs provided by the tool used (JDBC, Hibernate, etc.), the developer must ensure that the mapping fits the needs of the application. Retrieving data from the database may also require serializing objects, depending on the database technology and the intermediate database connector being used. Hibernate is such an example.

Based on the analysis of the back-end database support in GWT, we can summarize our conclusions in Table 2:

Support of database technologies	MySQL, PostgreSQL, Oracle, DB2, etc.
Automatic Code Generation at the Model Layer	No
Automated Data Management	No
Ease of Connecting the Framework to the Database	Difficult – All configuration is manual
Strictly Defined Conventions	No; camel case is recommended
Data Validation	Manual; Only if enforced by the developer; Could create security issues/corrupt the database

Table 2: GWT – Back-End Database Analysis: Summary

3.1.3: “Plain old Java” on the desktop – via RPC

Using “Plain-old Java” on the desktop as the implementation environment for our case study, and RPC to connect to the underlying database, we encountered issues similar to those we came across when using GWT. Therefore, the conclusions we state in Table 2 above are valid in this case as well. Our conclusions are further reinforced by the results obtained in [13].

3.2: Framework Comparison – Middle-Tier

At the middle-tier we focus on achieving three major results: code reuse via common library utilities (CLU), low cohesion and high coupling. All of these are needed in order to ensure rapid and robust system development. By creating common library utilities (CLU)s, we eliminate the repetition of code in the application. This not only decreases the amount of time needed to test the application, but also facilitates the extensibility and maintainability of the application. Further, by decreasing the coupling and increasing the cohesion between the different components of the system, we aim to achieve direct traceability from model to code and vice versa.

3.2.1: CakePHP

In CakePHP “object-orientedness” has a somewhat different meaning from what is generally understand to be the features of an object-oriented implementation environment [11]. Although there are different types of classes, they are used largely as internal components of the framework rather than as providers of an application programming interface in the application layer. As a result, object mapping is indirect and data encapsulation is not applicable. Further, objects do not have the same organization as they do in the object-oriented “world” we know from languages like Java [11]. Objects in CakePHP are really containers of some functions/methods. They do not encapsulate data – most of the data they need to work with is passed to and from them via one or more layers of associative arrays. This creates security issues, as all of the code (object representation) is tightly coupled to the underlying database (i.e. array indices must match the column names used in database tables, etc.). Code Snippet 3 shows the internal organization of the ‘User’ object in the CakePHP implementation environment:

```
Array
(
    [User] => Array
        (
            [id] => 4db4bd5b-5118-4cfa-a294-72978915a220
            [bbid] => 800123456
            [prefix] =>
            [first_name] => Martin
            [middle_name] => H.
            [last_name] => Georgiev
            [suffix] =>
            [username] => mgeorgiev@acm.org
            [password] => 1234567
            [group_id] => 3
            [created] => 2011-04-24 20:16:27
            [modified] => 2011-04-24 20:16:27
        )
    [Group] => Array
        (
            [id] => 3
            [name] => Students
        )
    [Address] => Array
        (
            [0] => Array
                (
                    [id] => 30
                    [user_id] => 4db4bd5b-5118-4cfa-a294-72978915a220
                    [address] =>
                    [city] =>
                    [state_id] =>
                    [zip_code] =>
                    [country] =>
                    [primary_phone] => 305-349-3438
                    [alternative_phone] => 718-690-1050
                    [preferred_contact_method_id] => 2
                    [created] => 2011-04-24 20:16:27
                    [modified] => 2011-04-24 20:16:27
                    [PreferredContactMethod] => Array
                        (
                            [id] => 2
                            [name] => Email
                        )
                )
        )
    [CareerContact] => Array
        (
            [0] => Array
                (
                    [id] => 28
                    [user_id] => 4db4bd5b-5118-4cfa-a294-72978915a220
                    [is_alumnus] => 1
                    [advice_to_students] => Look for internships
                    [advice_to_career_services] => Assist students in finding
internships
                    [administrator_notes] =>
                    [career_contact_status_id] => 3
```

```
                    [mail_privacy] =>
                    [email_privacy] => 0
                    [primary_phone_privacy] => 0
                    [alternative_phone_privacy] => 1
                    [created] => 2011-04-24 20:16:27
                    [modified] => 2011-04-24 20:19:15
                    [CareerContactStatus] => Array
                        (
                            [id] => 3
                            [name] => Submitted
                        )
                )
        )
    [RequestDegree] => Array
        (
            [0] => Array
                (
                    [id] => 65
                    [user_id] => 4db4bd5b-5118-4cfa-a294-72978915a220
                    [major] => Computer Science
                    [institution] => The College at Brockport
                    [graduation_year] => 2011
                    [last_semester_id] => 1
                    [degree_level_id] => 2
                    [is_private] => 0
                    [created] => 2011-04-24 20:16:43
                    [modified] => 2011-04-24 20:16:43
                    [LastSemester] => Array
                        (
                            [id] => 1
                            [name] => Spring
                        )
                    [DegreeLevel] => Array
                        (
                            [id] => 2
                            [name] => Bachelor's
                        )
                )
    [RequestJob] => Array
        (
            [0] => Array
                (
                    [id] => 48
                    [user_id] => 4db4bd5b-5118-4cfa-a294-72978915a220
                    [employer] => Excellus Blue Cross Blue Shield
                    [title] => Intern
                    [description] => Analyzed the corporate wireless network
for security holes
                    [start_date] => 2008
                    [end_date] => 2008
                    [is_private] => 0
                    [created] => 2011-04-24 20:18:16
                    [modified] => 2011-04-24 20:18:16
                )
    [UserDegree] => Array
        (
        )
```

```
    [UserJob] => Array
        (
        )

    [AssistCategory] => Array
        (
            [0] => Array
                (
                    [id] => 2
                    [name] => Talk to students about my employer
                    [AssistCategoriesUser] => Array
                        (
                            [id] => 108
                            [user_id] => 4db4bd5b-5118-4cfa-a294-72978915a220
                            [assist_category_id] => 2
                        )

                )

            [1] => Array
                (
                    [id] => 5
                    [name] => Talk to students about job search
                    [AssistCategoriesUser] => Array
                        (
                            [id] => 109
                            [user_id] => 4db4bd5b-5118-4cfa-a294-72978915a220
                            [assist_category_id] => 5
                        )

                )

        )

)
 ########## Code Snippet 3: CakePHP – Internal Object Organization ##########
```

Behavior mapping is also very different in CakePHP, as opposed to that we know from more conventional object-oriented languages (e.g. Java/C#). Methods, as they are known in the object-oriented world, are in fact PHP functions accessed in CakePHP via the class name (much in the same way as static methods in Java) [11]. Moreover, function names are expected to correspond to the names chosen for the views. Controllers are tied to the respective Models via the CakePHP naming scheme. As a result, the default scheme is that both a Controller and the associated Model are tightly coupled to an underlying database table. Although deviations from this pattern are possible, they are not encouraged by CakePHP's conventions [11]. All of this creates obstacles to enhancing the maintainability and extensibility of the application. However, by favoring convention over configuration CakePHP provides automatic code generation of basic

functionality, such as add/edit/view and delete, at the middle-tier level via the `bake` built-in tool.

Data Management is also scattered throughout the application in CakePHP. There is no 'Front Controller' concept [11]. We see multiple points of data access, as each user interaction often goes to a different application controller. While typically this application controller will be associated with a default Model which talks to a database table with the same name, some configuration can be done to make each one of the application controllers talk to multiple tables. This is not recommended by CakePHP, and is certainly not the default behavior. If used, it also curbs the ability of the `bake` built-in tool to generate code correctly. However, this non-standard configuration is needed in many situations – including our project.

The inadequate support of the 'Front Controller' concept in CakePHP is illustrated by Figure 5:

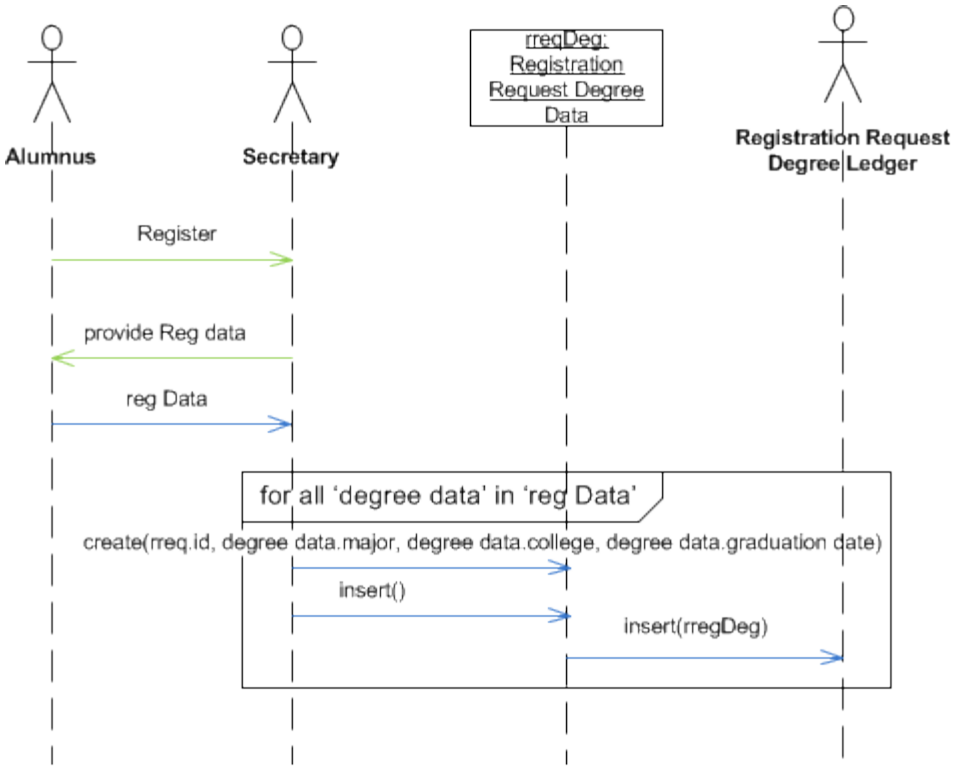

Figure 5: Record a Registration Request – Alumnus

In the context of CakePHP, the 'Secretary' – a role of the 'Front Controller – is split among multiple controllers. When a new user requests to register with the system, the 'Registration Request' controller assumes responsibility to manage the data. Thus, we may state that the

‘Secretary’ object above maps to the ‘Registration Request Controller’. However, this is not the only object the ‘Secretary’ maps to. Note that further requests to update the user’s data result in the ‘User’s’ controller to assume some of the responsibilities of the ‘Secretary’. In the mean while, when the system administrator goes to approve registration and/or update requests, the ‘Administrator’s’ controller assumes responsibilities designated to the ‘Secretary’ in our behavior model. This breaks the traceability from the PIM to the PSM, increases the complexity of the system unnecessarily, and makes further system extensibility and maintenance very difficult and time consuming. Thereafter, we can conclude that in CakePHP we observe increased coupling and decreased cohesion, which is the exact opposite of what we aim to achieve.

Based on the analysis of the middle-tier application layer in CakePHP, we can summarize our conclusions in Table 3:

Use of interfaces/abstract & base classes at the application layer	No; These are used at the internal framework layer
Data encapsulation	No; Data is held in multi-layer associative array
Decoupling the behaviors from the underlying database	No; Functions are tightly coupled to the database tables via the CakePHP naming conventions; Very low cohesion
Single point of access to the data/ Use of Front Controller	No; Multiple controllers can assume the responsibility of the Front Controller at different times
Automatic code generation at the middle tier level	Yes; Basic functionality provided only if all naming conventions are strictly followed
Testing Time	Longer than expected due to both the multiple points of data access and the lack of data encapsulation

Table 3: CakePHP – Middle-Tier Analysis: Summary

3.2.2: Google Web Toolkit (GWT)

As GWT builds on top of Java, it natively supports the set of object-oriented features we are familiar with from Java [12]. In fact, all of the code is written in Java with the exception of a couple of XML files (e.g. web.xml). Objects in the model map directly to objects in the implementation. Thus, we can use the basic object-oriented concepts of classes and data types to write extensible code. Data encapsulation is inherent in the use of the Java language and the

GWT framework [12]. Objects communicate via messages passed between them in a secure way. Therefore, the traceability from the PIM to the PSM is clear. This results in the development of extensible and maintainable applications that can better accommodate future change requests. Figure 6 shows the organization of the objects of our case study in the context of its GWT implementation. (It is worth noting here that such a class diagram is of little use in the CakePHP context, since the structural relationships shown here are not apparent in the PHP code – they appear only in the context of the use of the back-end database.):

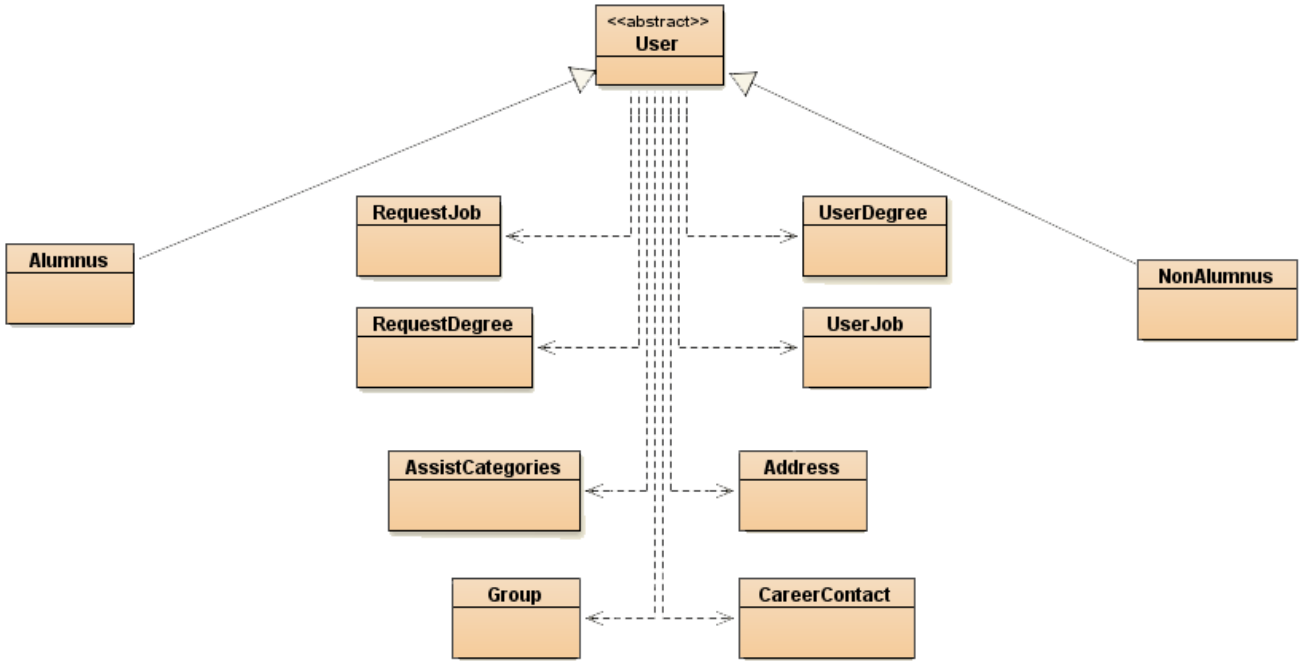

Figure 6: GWT - Object Organization

The behavior mapping is also easily apparent in GWT. The messages shown in the sequence diagrams correspond to method calls on the corresponding implementation-level objects. Also, as we can build CLUs, we can eliminate code duplication. Thus, we can speed up both the application development and the application testing. Further, since CLUs can be referenced from both internal and external application classes, code reuse is facilitated across all layers of the application. Hence, testing the application takes much less time than it does when using CakePHP. In GWT different behaviors map to different methods too [12]. Each method can provide data hiding natively by setting the respective method's accessibility to `private`. Method names can be representative of the behavior they encode. These names are for internal use only

and are not visible to the user of the system. Thus, we can ensure improved data security at minimal developmental cost. Unfortunately, this flexibility comes at the expense of the lack of automatic code generation at the middle-tier layer in a GWT application.

On the other hand, GWT excels over CakePHP in data management. Since we can use the GRASP principles, we can ensure single point of access to the back-end (database). Thus, the Front Controller – Secretary – in our model can be mapped directly to a remote GWT object. This object, in turn, can create/use other objects on the server corresponding to the Persistable objects in our behavior model. Therefore data encapsulation is fully applied.

Based on the analysis of the middle-tier application layer in GWT, we can summarize our conclusions in Table 4:

Use of interfaces/abstract & base classes at the application layer	Yes
Data encapsulation	Yes; Supported natively
Decoupling the behaviors from the underlying database	Yes; Default behavior
Single point of access to the data/ Use of Front Controller	Yes; Full Support of the GRASP principles
Automatic code generation at the middle tier level	No
Testing Time	Shorter than that of a CakePHP implementation due to the use of CLUs

Table 4: GWT – Middle-Tier Analysis: Summary

3.2.3: “Plain old Java” on the desktop – via RPC

Similarly to GWT, plain Java supports the design and development of CLUs, in addition to the use of object-oriented concepts, constructs and data types. Thus, we can apply all of the design principles and paradigms, including the well-known GRASP principles we are familiar with from GWT. Therefore, we can decouple the middle-tier layer from the underlying database. In this way we can achieve high cohesion and low coupling ensuring the extensibility and maintainability of the software system. Unfortunately, similarly to GWT, this flexibility comes at the expense of the lack of automatic code generation at the middle-tier layer. Thus, we can draw the same conclusions here as the ones we made in the case of GWT in Table 4 above. Our observations are further confirmed by the findings described in [13].

3.3: Framework Comparison – Front-End

At the front-end (client side) of the application we aim to decouple the views from the underlying controllers and models. In other words, the implementation of the front-end should be completely independent of the implementation of the middle-tier and the back-end of the application. Again, this is in order to facilitate its extensibility and maintainability. Further, the front-end should preferably use fundamental technologies for displaying the data in the browsers, such as (X)HTML and CSS. This requirement is enforced by the desire to have the application accessible from multiple platforms. Therefore, technologies such as AJAX/JavaScript and JScript may or may not be available. In fact, some companies disable them on purpose, as they introduce security vulnerabilities [9].

3.3.1: CakePHP

In CakePHP all views are tightly coupled to functions in the corresponding Controller objects [11]. In turn, all controllers are tightly coupled to the Model objects via strictly defined CakePHP naming conventions. Further, at the back-end of the application, all models are associated with similarly named database tables [11]. This hierarchy of associations establishes considerable interdependencies between all layers of the application. Consequently, it may be said that this makes the code monolithic. In fact, since all of the application's code is tightly coupled to the underlying database tables via the naming conventions, simple design changes propagate from the back to the front of the application, affecting all intermediate layers. For instance, a change made to the name of a database table, or a field in a table, usually results in multiple changes across most of the files containing the code of the application. Also, the advantages usually gained from the features of the MVC architecture are not realized. As shown in Figure 7, the essential goal of this architecture is to achieve loose coupling among the model, the view, and the controller objects. However, the concept of naming conventions across these components greatly increases the coupling in CakePHP. It should also be noted that the router feature shown in Figure 7 is present in CakePHP [11]. However, it simply enables the developer to rewrite the URLs used to get to the main controller object. The *content* of the views still remains coupled to the *content* of the controllers – i.e. the names of the functions in the controllers are still hard coded and match the names of the views. Further, the field names used

as keys in the associative arrays must stay consistent across all views, controllers and models and must match the corresponding field names in the database tables [11].

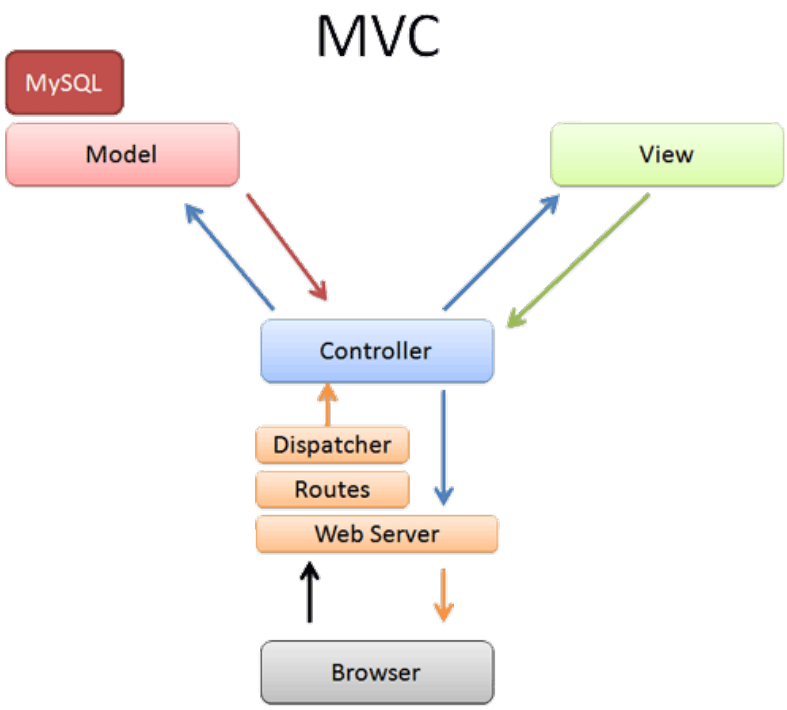

Figure 7: MVC Architecture

Coupling the user controls to the underlying database tables and database table fields via the strictly defined associative arrays holding the application's data creates security issues as well. In fact, a thorough analysis of the front-end – the GUIs of the application – can give an inside view of the database organization. Further, since data encapsulation is not available in CakePHP, ensuring the application's data integrity becomes a major undertaking. Possible solutions here are the use of UUIDs, Cookies, Session keys, etc [11]. However, we do not discuss these framework features here, due to space constraints.

On the opposite side, CakePHP supports automatic code generation at the front-end layer level via the `bake` built-in tool [11]. Unfortunately, this feature is useful only if all naming conventions are strictly followed across all layers of the application. Nevertheless, should the CakePHP naming conventions be followed, `bake` can generate all GUI forms responsible for adding, removing, editing and viewing the application's data [11]. Thus, by using `bake` we can speed up the core development process and allow more time for final product customizations and enhancements.

On the client-side CakePHP does not use any technology such as JavaScript or Jscript [11]. The application's front-end is exclusively in (X)HTML and CSS unless designed otherwise. Thus, regardless of the browser or the device the user uses to access the application, data is presented in an appropriate format.

Based on the analysis of the front-end application layer in CakePHP, we can summarize our conclusions in Table 5:

Front-End is tightly coupled to the Middle-Tier	Yes; The names of the views are the same as the names of the functions in the controllers
Front-End is tightly coupled to the Back-End	Yes, via the associative arrays holding the application's data
Front-End is dependent on JavaScript/Jscript/etc.	No; All GUIs are in (X)HTML/CSS unless otherwise configured
Automatic code generation at the Front-End layer	Yes, via the `bake` built-in tool

Table 5: CakePHP – Front-End Analysis: Summary

3.3.2: Google Web Toolkit (GWT)

In GWT, the client side views are written in Java, which are then translated to JavaScript at compile time [12]. All views interface to the 'Front Controller' via RPC. However, views are not coupled to the 'Front Controller' (Middle-Tier) via any naming convention. In fact, data encapsulation can fully be applied here and most of the implementation details can be hidden from the user. GUIs are not coupled to the models of the application either. Since data is being handled by objects not by associative arrays, as is the case in CakePHP, the front-end of the application is largely independent from the middle-tier and the back-end. Thus, internal changes in any of the tiers at the back of the application do not propagate forwards, as long as the names of the invoked methods are preserved. This can be achieved via the use of Java interfaces, which enable us to vary the actual implementation classes providing the middle-tier and back-end services. Therefore, the implementation in GWT meets the MVC architecture directives as shown in Figure 7 above, which makes the traceability from PIM to PSM clear. Further, it facilitates application's extensibility and maintainability.

GWT supports automatic code generation at the front-end application layer via the GWT's Designer tool [12]. Thus, we can speed up the application development and devote more time on customizing and enhancing the final product. It is important to note here that GWT's

Designer's functionality is different from the CakePHP's bake's functionality. Although both tools support automatic code generation, GWT generates the code based on the user controls the developer selects to add on the front-end of the application. Method calls to interface to the services in the middle-tier, and data management methods invoked on the back-end from the middle-tier, such as those that add, edit, delete and read the application's data, are the developer's responsibility i.e. these methods are not generated automatically. In contrast, CakePHP's `bake` tool generates all of the code (functions and controls) based on the database layout and the relationships between the database tables which are specified via the strictly defined naming conventions. Hence, GWT's Designer tool provides less functionality, but more flexibility than the CakePHP's `bake` tool.

On the other hand, GWT builds dependency on JavaScript at the front-end of the application [12]. Since all of the code on the front-end we write in Java gets translated to an equivalent JavaScript code at project compile time, the application will not be available if JavaScript is disabled on the client side. Further, the application may not work as expected for some mobile users, since not all mobile devices have full support of JavaScript. Thus, should we decide to use GWT for application development, we must be very well aware of these accessibility issues.

Based on the analysis of the front-end application layer in GWT, we can summarize our conclusions in Table 6:

Front-End is tightly coupled to the Middle-Tier	No; Data encapsulation can fully be applied
Front-End is tightly coupled to the Back-End	No; The front-end does not interact with the back-end at all; Data is passed in-between layers via objects; Data encapsulation can fully be applied
Front-End is dependent on JavaScript/Jscript/etc.	Yes; The front-end is dependent on JavaScript which may hinder the application's availability
Automatic code generation at the Front-End layer	Limited functionality via the GWT designer tool

Table 6: GWT – Front-End Analysis: Summary

3.3.3: "Plain old Java" on the desktop – via RPC

Similarly to GWT, plain Java allows the front-end to be independent from the back-end of the application. Data encapsulation can fully be applied in order to hide the implementation

details from the user. Data is captured from the user controls and sent back to the 'Front Controller' at the middle-tier via RPC. The front-end never interacts with the back-end directly. Thus, the MVC architecture directives shown on Figure 7 above are met. The traceability from PIM to PSM is clear, similarly to the implementation in GWT. Hence, the applications extensibility and maintainability is facilitated.

When using plain Java on the desktop, we can use tools, such as NetBeans, to automatically generate some of the code at the front-end layer of the application [14]. Again, similarly to GWT, NetBeans generates the user controls' code only i.e. methods to interface to the services provided by the middle-tier, and methods invoked from the middle-tier controlling the add, edit, remove, and view functionality of the application are the developer's responsibility. Nevertheless, automatic code generation alleviates the burden of using Java Swing to manually create the user controls.

Lastly, since plain Java on the desktop uses Java Swing to put up the GUIs of the application, it does not build any dependency on additional technologies, such as JavaScript, JScript, HTML, CSS, etc. Therefore, the application's availability is guaranteed, assuming there is an installed JVM on the client side. However, the client side of the application must be downloaded and installed on the client's system. Our observations are further confirmed by the findings described in [13].

Based on the analysis of the front-end application layer in plain Java on the desktop, we can summarize our conclusions in Table 7:

Front-End is tightly coupled to the Middle-Tier	No; Data encapsulation can fully be applied
Front-End is tightly coupled to the Back-End	No; The front-end does not interact with the back-end at all; Data is passed in-between layers via RPC; Data encapsulation can fully be applied
Front-End is dependent on JavaScript/Jscript/etc.	No; The Front-End uses Java Swing to display the GUIs of the application
Automatic code generation at the Front-End layer	Yes, via external tools, such as NetBeans

Table 7: plain Java on the desktop via RPC – Front-End Analysis: Summary

4: Status of the Implementation

Throughout the development of the Career Advisor Connection Database System, we worked very closely with the management team at the Career Services Department at The College at Brockport, State University of New York. As they are one of the major stakeholders of our case study, their feedback was very important to us. Consequently, we always carefully analyzed their recommendations and made appropriate changes to our system. Further, we stayed in contact with both alumni and current students at the college, as their input is essential for the success of the system. Similarly, we evaluated their recommendations and introduced appropriate changes to our application.

At the time of the submission of this thesis, we are making final changes to our system in order to align it with the standardization requirements of the Web Services Department at the college. In addition, we are porting the application to the college's website template, used across all college-wide applications. Upon completing these final two stages, we will hand the application over to the Web Services Department, which then takes the responsibility to host the application and its database on the production servers owned by the college. Further, Web Services will maintain and extend the application, as appropriate, in order to fulfill the future needs of the college and its community.

The analysis of plain Java on the desktop via RPC presented above and the conclusions derived were partly a result of a related work we did on a rental management system for broadcast equipment. The main system was created by the Computer Science Department for the Communications Department at the college. We used POJO on the desktop to implement an equipment reservation renewal feature. The analysis of the application was fully completed, and the feature fully implemented and deployed. The extended version of the system is currently in use by the Communications Department at the college.

5: Conclusion

Based on the analysis presented and the conclusions drawn in the preceding sections, we can summarize that both GWT and CakePHP have their strengths and weaknesses. On one hand, when using GWT, we can take full advantage of all strengths of Java. We can use the well-known GRASP principles to build easily traceable, maintainable and extensible applications. Unfortunately, GWT provides this flexibility at the expense of limited auto-configuration and auto-code-generation functionality. For instance, setting up the access to the database is one of the hurdles that needs to be overcome at the beginning of every project. However, since this is a one time procedure, we dismiss it as a disadvantage here. In contrast, the JavaScript dependency GWT builds cannot be easily dismissed. In fact, depending on the targeted audience and the importance of the GWT application, this dependency could be a major roadblock.

Overall, from a development perspective, we conclude that GWT is a good fit for large projects, as it ensures the traceability from PIM to PSM, critical for maintaining and extending software applications. However, the developer should be very well aware of the JavaScript dependency GWT creates before starting any actual application development. Further, we do not recommend GWT for developing small applications, due to the largely increased development time needed to compensate the lack of automatic code generation across all layers of the application.

On the other hand, CakePHP with its support of convention over configuration allows rapid application development and deployment. Unfortunately, as all application code is tightly coupled to the underlying database, the application is very difficult to maintain and extend. Code is monolithic largely due to strictly defined naming conventions resulting in the lack of CLUs. Further, the lack of support for some features, such as objects, as defined in the object-oriented paradigm, introduces indirect mapping from documentation to code and breaks the traceability of the application. Nevertheless, based on the analysis presented and the conclusions derived in the previous sections, we conclude that CakePHP is a good fit for small application development, in which complexity is not a major concern and achieving results in a short period of time is much more favorable than extending the application in the long run.

6: References

[1] T. Bullinger, S. Mitra and T. Rao, “Teaching software engineering using a traceability-based development methodology”, Journal of Computing Sciences in Colleges, vol. 20, no. 5, pp. 249-259, May 2005.

[2] P. Kruchten, *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley, 2003.

[3] S. Ambler, *The Object Primer: Agile Model-Driven Development with UML 2.0*, Cambridge, United Kingdom: Cambridge University Press, 2004.

[4] R. Gibbs, *Project management with the IBM Rational Unified Process: lessons from the trenches*. IBM: Prentice Hall PTR, 2007.

[5] T. Bullinger and S. Mitra, *The Reverse Detective: Pragmatic Software Requirements & Analysis*. Rochester, NY: WME Books, 2006.

[6] The Standish Group International, *Chaos Summary 2009*, The Standish Group, 2009.

[7] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Upper Saddle River, NJ: Prentice-Hall, 2000.

[8] M. Fontoura, W. Pree and B. Rumpe, *The UML profile for framework architectures*, 1st ed. Boston: Addison-Wesley.

[9] D. Flanagan, *JavaScript: The Definitive Guide*, 5th ed. Sebastopol, CA: O’Reilly Media, Inc., 2011.

[10] R. Mall, *Fundamentals of Software Engineering*, 2nd ed. India: PHI Learning Pvt. Ltd., 2004.

[11] D. Golding, *Beginning CakePHP: From Novice to Professional*, New York, NY: Apress, 2008.

[12] J. Dwyer, *Pro Web 2.0 Application Development with GWT*, New York, NY: Apress, 2008.

[13] M. Kyurkchiev, “Exploring the Model-View-Controller (MVC) Architecture for the Web Tier in Developing Internet-Based Applications,” B.S. thesis, Department of Computer Science, SUNY College at Brockport, Brockport, NY, USA, 2007.

[14] R. Rischpater, *Beginning Java ME Platform*, New York, NY: Apress, 2008.