Technology analysis and redesign of the Instructional Module Development (IMOD™) System

Saishruti Shetty
Department of Engineering
Arizona State University
sjshetty@asu.edu

Abstract—IMOD™ is a tool that will assist instructors in the field of science, technology, engineering and mathematics (STEM) in the design of their courses. Based on the principles of Outcome Based Education (OBE), this tool will lead to better course planning and an improved learning experience for the users of the tool and their students respectively. This report presents an analysis of the existing IMOD™ prototype, a review of technologies chosen to remodel the IMOD™ system, and the subsequent implementation of the new IMOD prototype.

Index terms—Web application, Server side programming, Instructional module.

I. INTRODUCTION

Most science, technology, engineering and mathematics (STEM) educators have little training on developing a course based on factors such as objectives of the course, the time frame in which the objectives are to be met and the content to be covered in the course. There are tools in the market that enable instructors to build courses based on some of these factors, for example, scheduling classes. However, there is no tool that takes into consideration the complexity of educating students effectively based on a set of objectives and given a set of resources. The IMOD™ system aims to solve this complex problem and provide instructors an easy-to-use interface that will allow them to design their courses.

The IMOD™ system is an open-source web-based course design software that:

• Guides individual or collaborating users, step-by-step, through an outcome-based education process as they define learning objectives, select content to be covered, develop an instruction and assessment plan, and define the learning environment and context for their course(s)

• Contains a repository of current best pedagogical and assessment practices, and based on selections the user makes when defining the learning objectives of the course, the system will present options for assessment and instruction that align with the type/level of student learning desired

• Generates documentation of course designs. In the same manner that an architect's blueprint articulates the plans for a structure, the IMOD course design documentation will present an unequivocal statement as to what to expect when the course is delivered

• Provides just-in-time help to the user. The system will provide explanations to the user on how to perform course design tasks efficiently and accurately. When the user explores a given functionality, related explanations will be made available

• Provides feedback to the user on the fidelity of the course design. This will be assessed in terms of the cohesiveness of the alignment of the course design components (i.e., content, assessment, and pedagogy) around the defined course objectives.

II. THE PROBLEM STATEMENT

For the scope of this project, the following objectives were to be met:

• Study the current implementation and identify the technologies to be used in the revision and future development of the IMOD™ web application

• Refine the database schema design

• Install and configure the server side and client side technologies of IMOD™

• Develop the login, registration and index pages for the IMOD™ system

III. CHALLENGES IN THE PREVIOUS IMOD™ SYSTEM

The following challenges were prevalent in the previous version of the IMOD™ system:

• The existing database used inconsistent naming conventions and certain relationships had to be redesigned to better match the requirements of the project.

• PHP is not compatible with Apache Jena™. Apache Jena™ is a Java framework for building Semantic Web applications. Jena provides a collection of tools and Java libraries to help one develop semantic web and linked-data applications. The IMOD™ system is going to be a semantic web application in the future and...
using technologies compatible with Apache Jena was a major requirement of this project.

- ExtJs has a steep learning curve and can be difficult to debug.
- There was no version control and work was done on the same project simultaneously by different developers. This posed a big issue - integrating modules.
- The developers used their own database versions and made changes to the tables on a need-basis. This resulted in inconsistent relationships and naming conventions.

IV. ANALYSIS OF TECHNOLOGIES

The new implementation of the IMOD system had to overcome the disadvantages of the previous system. The purpose of analyzing various technologies was to ensure rapid development with the latest technologies in the field of software development and use open source technologies wherever feasible. Towards this end, an analysis of web application frameworks, version control systems, server side technologies and client side technologies was performed. Appendix A shows all of the technologies considered and the analysis that informed the final selection. The sections below provide a detailed description of the selected technologies.

A. Groovy on Grails

Groovy on Grails is an open source, full stack, web application framework for the Java Virtual Machine. It takes advantage of the Groovy programming language and convention over configuration to provide a productive and stream-lined development experience.

The main features of this framework are as follows:

- Rapid: The Grails framework is one the fastest ways to get a web application up and running.
- Robust, proven technologies: Grails is based on Spring and Java – technologies that are tried and tested by millions.
- Interoperability with Semantic Web technologies: One of the major requirements of this project was having the ability to use the Apache Jena API in the future. Groovy runs on JVM, hence, it is compatible with Apache Jena API which is written in Java.
- Simplicity: Groovy has a clean and simple syntax, and is easy to learn and understand.
- Easily defined relationships: In Grails, it is easy to define relationships between classes using Grails Object Relational Mapping (GORM). These relationships could be one-to-one, one-to-many or many-to-many, and may be unidirectional or bidirectional.

- Scaffolding: Relationships specified in the domain classes can be used to create, read, update and delete database entries.
- Model, View, Controller architecture: Once domain classes are generated, controllers and views can be generated using grails commands.
- Plugins are available for most things any web application would need, for example, the SpringSecurity plugins for login.
- Support is available on all modern browsers automatically.
- Groovy on Grails has a very active development community, with ample documentation and other resources.

B. Groovy/Grails Tool Suite, the development IDE

The Groovy/Grails Tool Suite™ (GGTS) provides the best Eclipse-powered development environment for building Groovy and Grails applications. GGTS provides support for the latest versions of Groovy and Grails, and comes on top of the latest Eclipse releases.

- Included with GGTS is the developer edition of vFabric tc Server, the drop-in replacement for Apache Tomcat.
- GGTS is freely available for development and internal business operations use with no time limits.

C. PostgreSQL, the database management system

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

Following are some features of PostGreSQL:

- It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows.
- Its closest competitor, MySQL, is now owned by Oracle, making its future uncertain.

D. Git

Git is a distributed revision control and source code management (SCM) system with an emphasis on speed.

The advantages of using Git for source code management are as follows:

- Git is decentralized. The local copy is a repository with ability to commit to it and get all benefits of source control on the local system. This is useful when there might be connectivity issues. Local commits can be pushed to remote repositories any time the connectivity is reestablished. This makes it better than centralized version control systems like SVN.
- Git is well suited for open source projects – a project can be forked and changes can be made in the forked
project, one can then ask the original project maintainer to pull code from one’s fork.

- Github is free and allows developers to connect and review code.
- Git is now supported on Windows as well – initially it was only supported on *nix platforms

V. SYSTEM ARCHITECTURE

Grails uses Spring MVC as the underlying web application framework. Model–view–controller (MVC) is a software architecture pattern which separates the representation of information from the user's interaction with it. The Grails architecture can be described as follows:

- The foundation of Grails is the Java Virtual Machine (JVM).
- There is a separation between the Java language and the JVM. This is important in Grails because in the next level up from the JVM, both the Java and Groovy languages are used.
- The final layer of the architecture is the application layer. This layer follows the Model-View-Controller (MVC) pattern.
- Grails uses Spring MVC as the underlying web application framework.
- A controller handles requests and creates or prepares the response. A controller can generate the response directly or delegate to a view.
- A controller can have multiple public action methods, each of which maps to a URI.
- A model is a Map that the view uses when rendering information on the web page. The keys within that Map correspond to variable names accessible by the view.
- The architecture diagram has been shown below.


VII. IMPLEMENTATION

This section gives a step-by-step account of how the current IMOD™ project was implemented. The required downloads and installation steps are listed, followed by the configuration of the database and the grails application. This is followed by the login system, the domain classes and the relationships between them, the controllers and the views. Code snippets have been included with explanations accompanying them.

A. Initial setup

The following steps were followed for the initial setup of the Postgres database and:
- Windows 7 (64 bit) was used during the development process.
- Downloaded and installed the Java Development Kit (JDK) version 1.7.
- Downloaded Grails 2.2.1 from http://www.grails.org/download.
- Created a subdirectory C:\grails and unzip’ed the contents of the Grails download into this subdirectory.
- Set up environment variables so that the command line is aware of both Java and Grails:
  - Variable: JAVA_HOME
    Value: C:\Program Files\Java\jdk1.7.0
  - Variable: GRAILS_HOME
    Value: C:\grails\grails-2.2.0
- Clicked on “OK”. The environment variable settings were completed at this point.
- More detailed instructions on installing all of the above are available here—“http://www.grailsexample.net/installing-a-grails-development-environment-on-windows”.
- Downloaded and installed Git for Windows from “http://msysgit.github.io/”.
- Downloaded and installed Postgres for Windows from “http://www.postgresql.org/download/windows/”.

B. Database connection

The following steps were followed to establish the database connection between the Grails application and a Postgres database:
- Created a postgres database called “imodv6”.

![Fig 1: Grails Architecture](source: http://www.igm.univ-mlv.fr/~dr/XPOSE2009/Groovy_and_Grails/grails_architecture.php)
• Created a grails application called “imodv6”.
• Downloaded and copied postgresql-9.2-1002.jdbc4.jar under lib/ directory of the grails application.
• Made following changes to DataSource.groovy:

```groovy
dataSource {
    pooled = true
    driverClassName = "org.postgresql.Driver"
    username = "grails"
    password = "grails"
}
hibernate {
    cache.use_second_level_cache = true
    cache.use_query_cache = true
    cache.region.factory_class = 'net.sf.ehcache.hibernate.EhCacheRegionFactory'
}
// environment specific settings
environments {
    development {
        dataSource {
            dbCreate = "update" // one of 'create', 'create-drop', 'update', 'validate', '
            url = "jdbc:postgresql:imodv6"
        }
    }
    test {
        dataSource {
            dbCreate = "update"
            url = "jdbc:postgresql:imodv6test"
        }
    }
    production {
        dataSource {
            dbCreate = "update"
            url = "jdbc:postgresql:imodv6prod"
            pooled = true
            properties {
                maxActive = -1
            }
        }
    }
}
validationQuery="SELECT 1"
}
```

• Changed the username and password according to the imodv6 database user permission levels. Changed the URL in DataSource.groovy to reflect the database name.
• In BuildConfig.groovy, added this line in dependencies: runtime ‘postgresql:postgresql-9.2-1002.jdbc4’
• Executed the command “grails run-app” for the current grails application in the Groovy Grails Tool Suite. This resulted in a successful connection between the Grails application and the Postgres database.

C. Login system

The following steps were followed in order to configure the login system:
• In conf/BuildConfig.groovy, added the following under “plugins”:

```groovy
compile ":spring-security-core:1.2.7.3"
```
and executed “run-app” command for the grails application.
• In conf/BuildConfig.groovy, added the following under “plugins”:

```groovy
runtime ":jquery:1.8.3"
compile ":mail:1.0.1"
compile ":jquery-ui:1.8.24"
compile ":famfamfam:1.0.1"
```
and executed “run-app” command for the grails application.
• Note: It is required to execute run-app after each step, otherwise the dependencies will not be installed correctly.
• In conf/Config.groovy, added the following under “plugins”:

```groovy
grails {
    mail {
        host = "smtp.gmail.com"
        port = 465
        username = "imod.grails@gmail.com"
        password = "B0bblegum"
        props = ["mail.smtp.auth":"true",
```

```groovy
minEvictableIdleTimeMillis=1800000
timeBetweenEvictionRunsMillis=1800000
numTestsPerEvictionRun=3
testOnBorrow=true
testWhileIdle=true
testOnReturn=true
```
"mail.smtp.socketFactory.port": "465",
"mail.smtp.socketFactory.class": "javax.net.ssl.SSLSocketFactory",
"mail.smtp.socketFactory.fallback": "false"]
}
}
and executed “run-app” command for the grails application. This is the e-mail ID created for the current application. When new users register, they will receive and e-mail from this email ID. Note: Exclude the conf/Config.groovy file from the public git repository since the email ID and password will be known to everyone otherwise.

- In conf/BuildConfig.groovy, added the following under “plugins”:

```groovy
compile "spring-security-ui:0.2"
```

and executed “run-app” command for the grails application.

- Executed the following script which comes with the Spring Security plugin: s2-quickstart imodv6 ImodUserRole. This command has the following syntax: s2-quickstart <package> <user> <role> <requestmap>

- The above steps result in successful implementation of login and registration for the grails application.

D. Domain classes

- Before creating the domain classes, the database schema design was decided upon and prototyped with multiple revisions.

- Domain classes are core to any business application. They hold state about business processes and also implement behavior. They are linked together through relationships that can be one-to-one, one-to-many, or many-to-many. These relationships may be unidirectional, or bidirectional such as between an owner class and an owned class.

- Once the database schema prototype was finalized, Grails Object Relational Mapping, commonly known as GORM, came into picture. Under the hood, GORM uses Hibernate 3, a very popular and flexible open source ORM solution.

- Domain classes are created using the create-domain-class command, or in the Groovy Grails Tool Suite, by right-clicking on the “domains” subfolder and selecting “New” -> “Domain class”.

- Selected the package in which the domain class should be created and Grails generates the domain class.

- Added the fields to the domain class, add constraints that apply to the domain class fields, and add relationships between the domain classes.

- Executed “run-app” for the Grails application. This results in the tables corresponding to the domain classes to be generated in the “imodv6” database. This can be verified in pgAdmin.

- In case the structure of a domain class needs to be changed after the database table for the domain class is created, the table will need to be deleted/dropped from the database before executing the run-app command again.

E. Controllers

- Controllers are generated for a domain class using the “generate-controller <domain class name>” command.

- Controllers for a domain class like Imod required beforeInterceptor() to ensure that the user is logged in:

```groovy
def beforeInterceptor = {
    if (!springSecurityService.isLoggedIn()) {
        redirect(controller: 'login', action: 'auth')
    }
    return false
}
currentUser = springSecurityService.currentUser
```

- The user-specific data can then be displayed using the following code in list():

```groovy
def displayList = Imod.executeQuery("select distinct i from Imod i where i.owner="+currentUser)
```

F. Views

- Views are generated for a domain class using the “generate-views <domain class name>” command.

- This command typically produces the following files under the views/<domain class name> subfolder: _form.gsp, create.gsp, edit.gsp, list.gsp, show.gsp.

- The file _form.gsp contains all the fields that are displayed in the create.gsp, edit.gsp, list.gsp and show.gsp pages. One can modify the fields and their order by modifying _form.gsp.
• If an external JavaScript library like jQuery UI tabs needs to be used, the best way to follow is:
  a. Copied the file (say, 'jquery-ui-1.10.2.custom.min.js') in the web-app/js/ subfolder, so that the file web-app/js/jquery-ui-1.10.2.custom.min.js exists.
  b. In conf/ApplicationResources.groovy added the jqueryui module:

```groovy
modules = {
  application {
    resource url: 'js/application.js'
  }
  jqueryui {
    resource url: 'js/jquery-ui-1.10.2.custom.min.js'
  }
}
```

c. In the groovy server page, added the following:

```groovy
<r:require module="jqueryui" />
```

G. User interface:

The user interface was designed such that it resembles the IMOD™ brand library as closely as possible. The jQuery UI library has been used for tab, accordion and dialog box features of the website. The jQuery UI theme used is “flick”. Appendix A has screenshots of the web application – the index page, the login page, the i-mod list page and the overview page of the i-mod.

VIII. TESTING

Grails provides many ways to making testing easier from low level unit testing to high level functional tests. Whenever a controller or view is generated in Grails, the accompanying unit tests are automatically generated under the test/unit directory in the Grails applications. These tests are executed by running the 'test-app' grails command. Further user interface testing was performed by using Chrome’s developer tools and console.

IX. FUTURE SCOPE

In the future development of the application, all the modules would have additional security for user access. Users would be able to share and co-own i-mods. Scheduling of classes would be implemented in the user interface. Help/tutorial options would be embedded in the user interface to guide the user through the process of creating an i-mod.

ACKNOWLEDGMENT

I would like to thank Dr. Srividya Bansal and Dr. Odesma Dalrymple for providing their valuable guidance and advice throughout the project work and giving me the opportunity to work on this project where I got a chance to learn and apply new technologies. I would also like to extend my thanks to John Lewis and Parker Neff for their guidance with the technology stack used for this project.

APPENDIX A

a. Database schema diagram (page 7)
b. Screenshot of the index page of the new IMOD™ system (page 8)
c. Screenshot of the login page of the new IMOD™ system (page 8)
d. Screenshot of the list of i-mods that belong to a user (page 8)
e. Screenshot of the i-mod overview page (page 8)
f. Analysis of technologies – attached excel sheet
Fig 2. Database Schema and relationships