CAL STATE FULLERTON

Department of Computer Science

This project has been satisfactorily demonstrated and is of suitable form.

This project report is acceptable in partial completion of the requirements for the Master of Science degree in Software Engineering.

The Adaptive Instructional Module Engine: A Proof of Concept for Feedback-Driven Interaction through Real-Time Assessment

Project Title

Brett Taylor Hice Lee

Student Name

Ning Chen, Ph.D.

Advisor's Name

Advisor's signature Date

Reviewer's Name

Reviewer's signature Date

The Adaptive Instructional Module Engine:

A Proof of Concept for Feedback-Driven Interaction through Real-Time Assessment

Brett Taylor Hice Lee

May 2012

Acknowledgements

I would like to extend my heartfelt gratitude and love for my wife, children and extended family for the support and encouragement necessary to complete this program. I am in great debt to these loved ones whom have been there to “pick up the slack” when I am occupied with schoolwork and research. They have been tolerant of my occasional moodiness and have been exceedingly understanding of the challenges in which I was engaged. I do hope to someday repay all of the hours spent away from them with interest. I would also like to extend many thanks to the outstanding Engineering and Computer Science faculty of California State University, Fullerton for developing this unique and enriching program, and for providing such high quality instruction and guidance. Dr. Ning Chen in particular has been very supportive of this ambitious project from the start, and his insight has proven invaluable.

Abstract

The Adaptive Instructional Module Engine (AIME) is a software project with the goal of eliminating much of the coding necessary for the development of adaptive instruction modules or web-based games. The project seeks to provide a JavaScript API, standardized data model and a simple web-based user interface for creating the elements necessary for tracking user progress for any number of “knowledge areas” or skill sets, at a specified level of mastery. In addition, administrative users will be able to observe the progress of users in each module. This project document details the vision and scope, software requirements specification, software architecture, implementation details, user-facing instruction and deployment guides, as well as information relating to the development approach, challenges, and lessons learned.

Keyword List

AIME:

References to *AIME server*, *AIME web application*, *system* or simply *AIME*, unless otherwise noted, refer to the *Adaptive Instructional Module Engine* web application; the platform being described, for generating, and reporting upon, adaptive interactions between external modules and users.

API:

Application Programming Interface: the necessary libraries, services and conventions used by web and game developers [1] when accessing AIME user data.

Availability:

Quality attribute; the continuous system functionality and the consequences of system failure (e.g., up-time and recovery). [2]

Conceptual Integrity:

Quality attribute; the "underlying theme or vision that unifies the design at all levels." [2]

Correctness and Completeness:

Quality attribute; the ability of the architecture "to allow for all of the system's requirements and runtime resources constraints to be met." [2]

Cost and Benefit:

Quality attribute; the level of benefit, or the quality of product, which is achieved for a given budget. [2]

External Module:

An external module is being defined as the business and user interface component for an instructional module, which interfaces with AIME through a JavaScript API. The requirements for the external modules are not specifically defined herein, except that it must use the provided API methods for accessing data, implement the required interfaces, and reside in a location that is accessible by AIME.

Framework (or, web application framework):

A foundation to aid developers in the implementation of (web) applications, providing common core functionality. [3]

Grails:

Refers to the rapid application development framework for the Groovy programming language. [4]

Groovy:

Refers to the dynamic programming language, similar to Java, which runs on the Java Virtual Machine (JVM) technology stack. [4]

Knowledge Area:

A knowledge area is being defined as a unit of required achievement in the instructional model. This may be as granular or general as the developer requires.

Instructional Module:

An instructional module is being defined as a single entity created within AIME which provides sequencing instructions and user-progress data persistence services, having a one-to-one relationship with an external module, where the business logic for runtime is executed.

Modifiability:

Quality attribute; the cost or difficulty, related to change in a system. [2]

Performance:

Quality attribute; the timing of user-facing interruptions/delays, and resources necessary for maintaining a high level of functionality. [2]

SCORM:

References to “SCORM” or “SCORM 2004” refer to the Sharable Content Object Reference Model version 2004. [6, 7]

Security:

Quality attribute; the system's ability to prevent unauthorized access. [2]

Targeted Market:

Quality attribute; refers to whom and how the product will be made available. [2]

Testability:

Quality attribute; the difficulty in exposing software or usability faults. [2]

Time to Market:

Quality attribute; the ability for development time to be condensed in response to pressure from external sources. [2]

Usability:

Quality attribute; the ease for a user to complete a task. [2]

Table of Contents

1 Introduction

1.1 Purpose

This document details the proof of concept for the Adaptive Instructional Module Engine (AIME); a development framework, which can provide interactivity and dynamic sequencing of external modules using real-time feedback. This proposed framework is specific to web-based applications, with a focus on instructional modules and JavaScript based games that have a requirement for adaptive game-play.

1.1.1 Feedback-Driven Interaction through Real-Time Assessment

The mechanism providing the adaptive behavior for AIME relies on user feedback and real-time assessment of user progress. This assessment occurs in JavaScript modules that interface with end users, and is passed along to the AIME server for persistence. The results of the assessment are used to evaluate the user's progress in a particular area and determine the content that is available as a result of the re-evaluation.

1.2 Objectives

The objectives for this project shall be met in part by the artifacts delivered in this document and in part by the source code and deployable web application contained on the included CD-ROM media.

Artifacts delivered in this document include:

- Vision and Scope Documentation
- Software Requirements Specification
- Software Architecture Documentation
- User Documentation and Deployment Instructions

Artifacts delivered in the attached CD-ROM media include:

- Original source code

- A distributable web application objectives, or WAR file
- An example instructional module for demonstration purposes
- Consolidated project summary and presentation files in the form of this written document, an audio file and slide deck

1.3 Basic Requirements

1.3.1 Produce the Web Application

Details of the software requirements for the web application have been documented in the Software Requirements Specification. Because of the nature of the product development for this project, in that the developer is also managing the project, no structured requirements change process is being implemented. Any potential changes to software specification requirements that occur within development are evaluated based on the scope of the initial release and added, removed or amended as necessary to complete the project within stated deadlines with satisfactory completion of the stated deliverable artifacts.

1.3.2 Produce Demonstrable Instructional Module

The example instructional module shall be a basic JavaScript based application or dynamic test harness, which provides some insight into the adaptive capabilities of the AIME web application. This module shall be bundled with AIME and seeded in the production database.

1.3.3 Produce Documentation

The documentation artifacts include those that both inform the development process as well as those that provide the users of the software product with guidance on the use, configuration and deployment of AIME.

2 System Development Process Methodology

2.1 Initial Planning

Initial planning has consisted of drafting a Vision and Scope document, Software Requirements Specification and a Software Architecture Description, all of which cover the high-level concepts of the initial release of the Adaptive Instructional Module Engine. As part of this initial planning, market research has been performed and a need for a web application development platform implementing SCORM 2004 or as-of-yet released adaptive instructional standards, such as CMI 5 has been established. [13] The content of the aforementioned documents have been consolidated and expanded upon to make up the majority of this document.

2.1.1 Design

High-level design considerations are described in the Architectural Design section of this document.

2.1.2 Development

Development of the software has been performed according to a modified rapid application development or iterative development methodology. This approach started with requirements planning, which have been addressed in the Software Requirements Specification, and implements requirements at various stages of development.

Upon completion of key milestones, feedback is elicited from outside individuals, including the project advisor, peers and colleagues. Each cycle of development acknowledges the time necessary to address feedback.

3 Vision and Scope

3.1 Business Requirements

3.1.1 Background, Business Opportunity, and Customer/Market Needs

The Adaptive Instructional Module Engine (AIME) is a project to create a software solution to online instruction, and other domains wherein a need for adaptive response from the software based upon measurement of a users progress or achievement exists.

The rationale behind AIME is the individualized implementation of instructional design theory known as the ADDIE model. The ADDIE instructional design model is made up of five components: analysis, design, development, implementation and evaluation. [11] This implementation of the ADDIE process is more or less a feedback loop, where users are first evaluated based on needs, the content is designed and developed based on those needs, and then the implemented content is delivered and user response is evaluated. From there, the loop begins again until mastery of all knowledge areas. In instructional design this is a much more "macro" process, spanning the entire duration of instruction, [12] whereas in the proposed implementation, feedback and reevaluation is done in real-time.

Opportunities exist for an open-source, lightweight product, such as the one being proposed, for the purpose of quick development and deployment of JavaScript-based applications, games and instruction modules which rely on adaptive interaction between the user and content. [13] Allowing for easy development of sequencing based upon user-defined objectives and metrics reduces development time significantly, as well as the amount of code that needs to be written.

Learning Management Systems (LMS) provide support for content delivery and management of content as well as many other instruction-centric features, and there are plenty of LMS options available on the market. While these systems tend to work well in their domain, they also tend to be “pedagogy neutral,” which in effect means that they are not designed around support of a specific model for instruction. AIME is not pedagogy neutral, and puts emphasis on the need for adaptive delivery of content individualized to the learner, and in this approach is also open to the application of this model to other domains. A potential alternative domain could be online gaming: wherein a platform focusing on adaptive interaction between user and content, driven by real-time feedback could provide for more fulfilling game play.

With regard to existing products, none found in the course of market research fills this particular need. In the domain of instructional learning, LMS products provide some features that AIME seeks to implement, however the focus of these solutions is only on instructional modules. In addition, LMS products usually support the launching of instructional modules and sequence management as but one of an array of features designed around managing the online learning process. For users that do not need all of the many features common in an LMS, or users in want of a basic platform for building JavaScript-based applications that are not part of an instructional module, AIME will be ideal.

The intention is for AIME to be released to the open source community, as a solution to the common problem of adaptive sequencing in JavaScript-based applications; as such it will not generate revenue directly. Benefits to be gained by the successful implementation of AIME will be in reduced development time for interactive JavaScript based applications, games and instructional modules, which can be monetized using advertisements, pay-for-play or in-app purchases. The benefit of releasing the product open source will be realized in increased stability as the result of contributions from the open source community and decreased development costs in maintaining the product.

3.1.2 Business Objectives and Success Criteria

BO-1: Attract open-source developers to improve and maintain framework.

BO-2: Develop a framework upon which monetized products can be built.

BO-3: Decrease development time necessary for creating new adaptive learning modules or games.

BO-4: Use AIME to develop a library of monetized applications and learning modules for revenue generation.

SC-1: Open-source release of first iteration within six months.

SC-2: At least two monetized applications within twelve months of project start.

3.1.3 Business Risks

RI-1: The product fails to attract the interest of qualified developers in the open source community.

RI-2: The initial product offering does not fulfill the needs of developers.

3.1.3.1 Consequences and Mitigation of Risks

In the case that the product fails to attract the attention of the open source community, the costs of maintaining the product fall primarily on the developing organization. Mitigation tactics may include more aggressive marketing and recruitment in the open source community to attract attention. Development of attractive applications making use of AIME and the publishing of statistics relating to the benefits of using AIME may also generate buzz.

In the case that the initial product offering does not fulfill the needs of developers, potential revenue from monetized applications using the platform may be deferred until additional development on the product. Mitigation tactics during requirements elicitation and design would include profiling of the target audience (web application, game and content developers) and previewing functional prototypes for feedback form developers. This tactic may also serve to mitigate the first scenario by generating additional buzz in the community.

3.2 Vision of the Solution

3.2.1 Vision Statement

AIME will accelerate the development of adaptive games and learning modules, encouraging innovation by reducing the amount of code necessary to develop an adaptive module, creating a solid base upon which to build web applications, and also by supporting rapid application development methodologies.

3.2.2 Major Features

FE-1: Create, edit and delete instructional modules.

FE-2: Create, edit and delete knowledge areas of an instructional module.

FE-3: Launch and provide sequencing for a JavaScript-based web application or module.

FE-4: Track and report on user progress for modules.

3.2.3 Assumptions and Dependencies

AS-1: Progress in applications can be quantified (assigned a value) which can be persisted for a given knowledge area assessment within a module.

DE-1: Games and modules must be able to launch via JavaScript.

DE-2: Games and modules must be able to make calls to JavaScript functions.

3.3 Scope and Limitations

3.3.1 Scope of Initial Release

FE-1: Creation of CRUD pages, schema and object models to support management of instruction modules.

FE-2: Creation of CRUD pages, schema and object models to support management of instruction module knowledge areas.

FE-3: Creation of JavaScript libraries necessary to launch and provide sequencing for modules.

FE-4: Creation of administrative portal and setup of user roles to support future tracking and reporting of user progress.

3.3.2 Scope of Subsequent Releases

FE-1, FE-2:

Enhanced validation of entry and wizard-based creation of modules and all other related components.

FE-3: Support for packaging of modules and direct import into engine. Support for future standards.

FE-4: Development of reports and dashboards to monitor users progress, customizable to specific user roles.

FE-5: An alternative modeling option, using XML to define an instructional module, it's knowledge areas, their assessment thresholds and sequencing details, which can be imported into AIME when a module is created. User progress information would be all that necessitates persistence in such a scheme.

3.3.3 Limitations and Exclusions

LI-1: No development environment is provided for the business logic and presentation components for modules.

LI-2: Data model and persistence support for instructional modules is limited to objects required for sequencing and tracking user progress.

3.4 Business Context

3.4.1 Operating Environment

3.4.1.1 Portal Access

AIME administrative portals will be accessible to a limited percentage of users, in a development environment via web access.

The following considerations relating to operating environment should be made regarding requirements elicitation:

- Web application runtime environment
- Performance is not a high priority
- All data is persisted in a database (locally or remote)

3.4.1.2 AIME Module Access

User access to AIME modules being launched is more generally available, but the no user interface is provided with the exception of an API for the launching of the AIME module. The following considerations relating to operating environment should be made regarding requirements elicitation:

- API calls need to be fast for module runtime
- Module data that is not specific to sequencing instructions is not managed by AIME
- Module runtime not relating to sequencing instructions is not dependent on AIME

4 Requirements Specification

A note on requirements changes: As this development effort is a proof of concept with a single developer and primary stakeholder, there is no requirements change process in effect. Changes to requirements and scope can take place at the discretion of the developer, as the project evolves. In a traditional development environment, this would not be recommended; details relating to the change management process would normally be detailed in this section of the document.

4.1 Overall Description

4.1.1 Product Perspective

The Adaptive Instructional Module Engine is a new system, which is to be used as a framework for sequencing and progress tracking for web applications to be developed separately. AIME is expected to reduce development time and the amount of code necessary to deploy a web application relying on adaptive sequencing.

4.1.2 User Classes and Characteristics

Administrator

Can manage access to the system by users. Can access reports on end user information and track end user progress.

Developer

Can create, edit and delete instructional modules and their supporting elements.

External Module

Though not an actual user, the external module is being documented as a user class because external module elements will need to be launched, and access AIME via a JavaScript API.

End User

The end user does not interact directly with AIME. End users interact with external modules, which interface with AIME via an API. End user progress data relating to AIME functionality is persisted in the database. End users are directed to authenticate when they attempt to launch a module.

4.1.3 Operating Environment

OE-1: AIME shall operate on a JVM technology stack, running on the current production version of Apache Tomcat 7 application server.

OE-2: AIME administrative and development portals shall be accessible via the current and last major production release (upon initial release of AIME) of Mozilla Firefox and Apple Safari web browsers.

OE-3: AIME shall be deployable by an organization on any platform that can run Apache Tomcat 7 application server.

4.1.4 Design and Implementation Constraints

CO-1: The AIME web application shall be written in the Groovy programming language, and idiomatic Java where appropriate, using the latest production version of the Grails development framework. [4]

CO-2: AIME will use the latest production version of the open source H2 file system database. [5]

CO-3: AIME will implement the SCORM 2004 [6] JavaScript API to interface with external modules as a template, but does not need to be fully compliant.

CO-4: Any generated HTML should conform to the latest HTML 5 specification, with graceful degradation for HTML 4-only browsers.

4.1.5 User Documentation

UD-1: AIME Server Deployment guide, documenting instructions on how to deploy the AIME web application.

UD-2: Developer user guide, providing an outline of how to create and configure an instructional module.

4.1.6 Assumptions and Dependencies

DE-1: Implementation of SCORM 2004 JavaScript API depends on details documented in the SCORM 2004 4th Edition Specification. [6]

AS-1: References to “administrators”, “administrator users” or “admin users” refer to users with the role of “Administrator”.

AS-2: References to “developers” or “developer user” refer to users with the role of “Developer”.

AS-3: References to “end user,” “learner” or “external module user” refer to users with the role of “EndUser”.

4.2 System Features

4.2.1 Create, Read, Update or Delete an Instructional Module

4.2.1.1 Description

A developer logs into the development portal and creates, modifies or deletes an instructional module, which is used to track end user progress and provide sequencing to the external module. This includes the creation, modification or deletion of supporting elements for a module such as a knowledge area or sequence objects.

4.2.1.2 Stimulus/Response Sequences

Stimulus: Developer requests to create a new instructional module.

Response: The system prompts the developer for the name of the module. Once created, the developer is given the ability to create knowledge areas for the module.

Stimulus: Developer selects a module and requests to create a new knowledge area for the specific module.

Response: The system prompts the developer for the name and assessment threshold of the knowledge areas. Once the knowledge areas are created, the developer has the ability to configure sequencing for the knowledge areas.

Stimulus: Developer selects an instructional module and chooses to modify or delete the record.

Response: The system processes the update or delete.

Stimulus: Developer selects a knowledge area and chooses to modify or delete the record.

Response: The system processes the update or delete.

4.2.1.3 Functional Requirements

Create.Module:

AIME shall permit a developer user, when logged into the Development portal to create a new instructional module.

Create.Module.KnowledgeArea:

AIME shall permit a developer user, when logged into the Development portal to create new knowledge areas for an existing instructional module.

Read.Module:

AIME shall permit a developer user, when logged into the Development portal to inspect elements of an existing instructional module.

Read.Module.KnowledgeArea:

AIME shall permit a developer user, when logged into the Development portal to inspect elements of an existing knowledge area.

Update.Module:

AIME shall permit a developer user, when logged into the Development portal to update values for elements of an existing instructional module.

Update.Module.KnowledgeArea:

AIME shall permit a developer user, when logged into the Development portal to update values for elements of an existing knowledge area.

Delete.Module:

AIME shall permit a developer user, when logged into the Development portal to delete an existing instructional module. The developer shall be prompted with messaging to cascade delete child records or remove dependencies manually.

Delete.Module.KnowledgeArea:

AIME shall permit a developer user, when logged into the Development portal to delete an existing knowledge area. The developer shall be prompted with messaging to cascade delete child records or remove dependencies manually.

List.Module:

AIME shall permit a developer user, when logged into the Development portal to list all instructional modules.

List.Module.KnowledgeArea:

AIME shall permit a developer user, when logged into the Development portal to list all knowledge areas relating to the selected instructional module.

4.2.2 Launch and Sequence External Modules

4.2.2.1 Description

An external module is referenced by an instructional module within AIME, and needs to be able to launch and retrieve sequencing information relating to the progress of the end user. This launching and sequencing will be managed through a JavaScript API based on the API wrapper functions detailed in the SCORM 2004 standard. [6]

4.2.2.2 Stimulus/Response Sequences

Stimulus:	End user requests launch of external module.
Response:	End user is prompted for authentication information and external module initializes with AIME via API adapter. Sequencing objects in JavaScript functions are initialized.
Stimulus:	End user requests to terminate external module.
Response:	External module terminates with AIME via API adapter.
Stimulus:	External module requests information from AIME via API adapter.
Response:	AIME provides response via API adapter through AJAX call to AIME web application. Value is returned from AIME *propertyMap* object.
Stimulus:	External module requests to set a value with AIME via API adapter.
Response:	AIME sets the value via API adapter through AJAX call to AIME web application.
Stimulus:	External module requests to commit changes via API adapter.
Response:	AIME commits the values in memory to the database. Sequencing JavaScript objects are updated, if necessary.

Stimulus:	External module requests last error code.
Response:	AIME provides response via JavaScript function to the last cached error code.
Stimulus:	External module requests last error string.
Response:	AIME provides response via JavaScript function to the error string associated with the last cached error code.
Stimulus:	External module requests diagnostic information.
Response:	AIME provides response via JavaScript function to the cached diagnostic information.
Stimulus:	External module reads sequencing.
Response:	AIME provides response via JavaScript function to evaluate the sequencing of the Knowledge Areas.

4.2.2.3 Functional Requirements

ExternalModule.Initialize:

When an end user requests an external module, AIME shall prompt the user to authenticate. AIME will then generate the sequencing XML for the external module based upon the user's progress, and launch the external module.

The external module will call the Initialize() JavaScript function in the global namespace of the module launching browser window.

ExternalModule.GetValue:

If initialized, when the external module calls the GetValue(parameter) JavaScript function, AIME shall return the SCORM data model element's value, if not from memory, from the database.

If not initialized, an error is logged.

ExternalModule.SetValue:

If initialized, when the external module calls the SetValue(parameter_1, parameter_2) JavaScript function, AIME shall set the value, parameter_2, to the SCORM data model element, parameter_1, to be saved in memory for the current session.

If not initialized, an error is logged.

ExternalModule.Commit:

If initialized, when the external module calls the Commit() JavaScript function, any values saved in memory since Initialize() or the last Commit() call shall be committed to the database.

If not initialized, an error is logged.

ExternalModule.GetLastError:

When the external module calls the GetLastError() JavaScript function, the last error code is returned.

ExternalModule.GetErrorString:

When the external module calls the GetLastString() JavaScript function, the text of the last error is returned.

ExternalModule.GetDiagnostic:

When the external module calls the GetDiagnostic(parameter) JavaScript function, details relating to the parameter key shall be returned not to exceed 255 characters. If the parameter key is unknown, an empty character string shall be returned.

4.2.3 Administrative and Development Web Portals

4.2.3.1 Description

Developer and Administrative users need to be able to log into AIME to perform actions relative to their user roles.

4.2.3.2 Stimulus/Response Sequences

Stimulus:	Developer and Administrative users log into AIME web portal.
Response:	Users are authenticated and presented with options based upon their user roles.
Stimulus:	Administrative user requests to create a new user account.
Response:	Administrative user is prompted for new user information.
Stimulus:	Administrative user requests to assign user roles to an existing user.
Response:	Existing user is granted access to areas relative to the user role settings.
Stimulus:	Administrative user requests to update a user's information.
Response:	Existing user is updated with the requested changes.
Stimulus:	Administrative user requests to delete a user account.
Response:	User account is deleted.
Stimulus:	Administrative user requests to review end user progress.
Response:	Administrative user is presented with listing of end user progress relating to instructional module.

4.2.3.3 Functional Requirements

Portal.Login.Admin:

When administrative users attempt to login to the web portal, AIME shall prompt for a user name and password. Upon authentication, users are logged in.

Portal.Login.Developer:

When developer users attempt to login to the web portal, AIME shall prompt for a user name and password. Upon authentication, users are logged in.

Portal.Login.EndUser:

When end users attempt to launch an external module, AIME shall prompt for a user name and password. Upon authentication, the requested module will launch.

Portal.Admin.CreateUser:

AIME shall permit an administrator user to create a new user record. The administrator user shall be prompted for user and role information at the time of creation.

Portal.Admin.ReadUser:

AIME shall permit an administrator user inspect the properties of a user record.

Portal.Admin.ReadUser.ViewReports:

AIME shall permit an administrator user to view reports relating to an end user's progress, listed by external module, for each user of with the role of "EndUser".

Portal.Admin.UpdateUser:

AIME shall permit an administrator user to update the properties and role of a user record.

Portal.Admin.DeleteUser:

AIME shall permit an administrator user to cascade delete a user record, and all related records.

Portal.Admin.ListUsers:

AIME shall permit an administrator user to list all users and their role.

4.3 External Interface Requirements

4.3.1 User Interfaces

UI-1: All user interfaces with AIME shall be presented with HTML (HTML 5 compliant specification) rendered by Groovy Server Pages (GSP) templates, or static HTML.

UI-2: Page style will use CSS 3 compliant style sheets.

4.3.2 Hardware Interfaces

No specific hardware interfaces identified.

4.3.3 Software Interfaces

SI-1: AIME shall run as a web application within the latest production version 7.x release of the Apache Tomcat application server. [14]

SI-2: AIME shall use the latest production version 2 release of the H2 database for data persistence. [5]

SI-3: JavaScript components shall make use of the latest production version of JQuery JavaScript library. [15]

SI-4: AIME server-side development shall be written in the Grails 2.0 development framework. [4]

4.3.4 Communications Interfaces

CI-1: AIME shall make use of the HTTP communications protocol for data transfer between the client and server.

4.4 Other Nonfunctional Requirements

4.4.1 Performance Requirements

No performance requirements identified at this time.

4.4.2 Safety Requirements

No safety requirements identified at this time.

4.4.3 Security Requirements

SE-1: Only administrator users may read or update user information.

SE-2: Only administrator users may view reports on end user progress.

4.4.4 Software Quality Attributes

No specific quality attribute requirements for initial release; however, quality attributes are taken into consideration in the Architectural Design section.

4.5 Other Requirements

In the initial release of AIME, no other requirements are being considered.

5 Architectural Design

5.1 Architectural Overview

5.1.1 Environment

The Adaptive Instructional Module Engine is being developed to operate within the Java Virtual Machine (JVM) platform, specifically as a web application to be run within the Apache Tomcat runtime environment.

5.1.2 Standards and Technologies

The development framework AIME will employ is the Grails framework, which supports the Java and Groovy programming languages. In order to leverage the benefits of a dynamic programming language and the rapid application development strategy which Grails supports, the majority of the business logic will be written in the Groovy programming language.

5.1.3 Schema

The Grails framework provides powerful code generation for tasks that tend to be very redundant, including dynamic object relational mapping (ORM) and schema generation. With this in mind, any diagrams implying something akin to a data model are strictly conceptual in nature and may not actually reflect the implemented database schema.

5.2 Stakeholder Concerns

5.2.1 Stakeholder Groups

- Administrator users
- Developer users
- End users
- Hosting Services/Systems Engineers/Support

- Sales and Marketing/Management
- Software Engineering/Testing staff

5.2.2 System Quality Attributes

5.2.2.1 Availability

Primary Stakeholders

- Developer users
- End users
- Administrator users
- Hosting Services/Systems Engineers/Support

Response Measure

Availability can be measured in the amount of time the system is completely or partially non-functional.

Assessment

The Grails framework provides the same exception handling capabilities as Java except that checked exceptions do not need to be caught. [8]

5.2.2.2 Modifiability

Primary Stakeholders

- Software Engineering/Developers
- Sales and Marketing/Management
- Hosting Services/Systems Engineers/Support

Response Measure

Modifiability can be measured by the effort necessary to make changes to the software system for maintenance and introduction of new functionality.

Assessment

The Grails framework provides tools that automatically generate much of the code necessary to support a new feature or change to functionality. [8]

5.2.2.3 Performance

Primary Stakeholders

- End users
- Hosting Services/Systems Engineers/Support
- Sales and Marketing/Management

Response Measure

Performance response is measured in the length of delays to accomplishing a user-initiated task.

Assessment

Performance tuning of the Apache Tomcat container and the JVM is well documented. [9]

5.2.2.4 Security

Primary Stakeholders

- End users
- Software Engineering/Developers
- Hosting Services/Systems Engineers/Support

Response Measure

Security is measured according to the level of effort necessary to circumvent protections to gain unauthorized access to the system or sensitive data.

Assessment

Numerous security plug-ins and modules are available for use within the Grails framework. [8]

5.2.2.5 Testability

Primary Stakeholders

- Software Engineering/Testing staff

Response Measure

Testability is measured according to the level of effort necessary to find defects in the software system.

Assessment

The JVM and Apache Tomcat container provide the ability to attach debug agents and debug remotely. [10] In addition, Grails has built in unit testing capabilities extending the de facto standard in unit testing, JUnit. [8]

5.2.2.6 Usability

Primary Stakeholders

- Developer users
- End users
- Administrator users
- Sales and Marketing/Management

Response Measure

Usability is measured by how effectively (and efficiently) a user is able to complete a task.

Assessment

User interface design should follow best practices of usability. Compartmentalized UI components (GSP and Groovy actions) that are commonly used should be created according to modern web standards, utilizing cascading style sheets (CSS) to provide a consistent user experience. [8]

5.2.3 Crosscutting Concerns

Significant design effort should be place on the following crosscutting concerns, which are not limited to a single component within the web application stack.

- Security
- Logging
- Fault detection

- Reporting

Note: Mention of these concerns does not guarantee any sort of requirement satisfaction for the initial release of AIME as a proof of concept.

5.2.4 Separation of Concerns

The Grails framework forces upon the application developer the application of the Model-View-Controller design pattern. [17] This pattern fits within the Separation of Concerns software development paradigm, which has become common in web application development. [16]

5.3 Architectural Views

5.3.1 Logical View

5.3.1.1 Primary Presentation

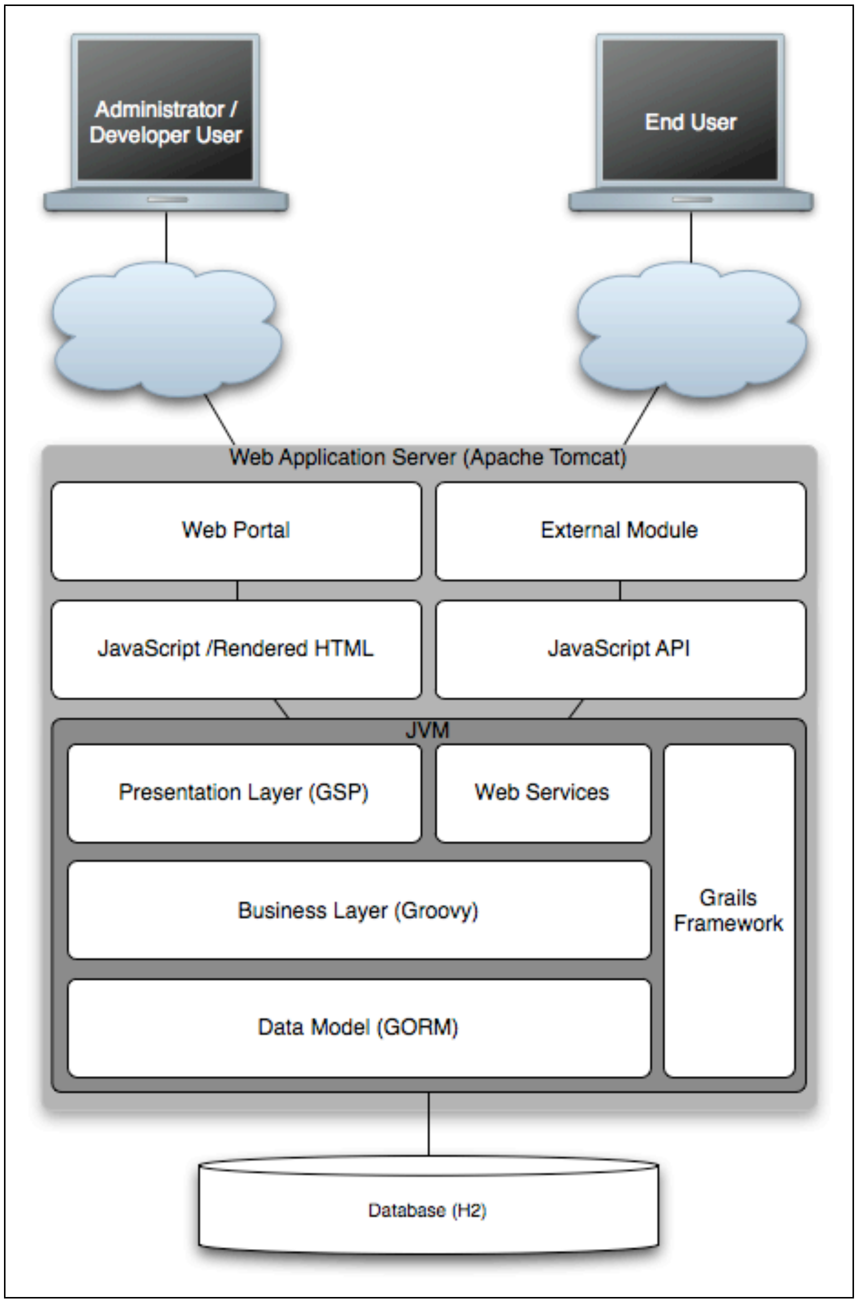

Figure 1: Logical view of system architecture

5.3.1.2 Element Catalog

Administrator / Developer User:

Represents users with the role of Administrator or Developer, accessing AIME via a web browser.

End User:

Represents users the role of End user, accessing AIME by launching an external module via a web browser.

Web Application Server:

Represents the runtime environment for the web application server, in this case Apache Tomcat.

Web Portal:

Represents the authenticated web site which serves as an access point for users with the role of Administrator and Developer.

External Module:

Represents the external implementation of the instructional module.

JavaScript / Rendered HTML:

Represents the technology layer, which expresses the web portal.

JavaScript API:

Represents the technology layer made up of the necessary JavaScript libraries, which support external modules.

JVM:

Represents the Java Virtual Machine, the platform upon which AIME is build.

Grails Framework:

Represents the development framework upon which AIME is build within the JVM.

Presentation Layer:

Represents the technology layer responsible for generating the user interface elements.

Web Service:

Represents the technology layer responsible for providing responses to client-side requests (primarily through asynchronous JavaScript calls.)

Business Layer:

Represents the technology layer where most of the application logic is performed. In AIME this will consist mostly of compiled Java and Groovy code.

Data Model:

Represents the technology layer, which models database elements into objects to be manipulated in the business layer. Also provides the mapping by which data can be persisted back to the database.

Database:

Represents the technology layer, where data elements are saved to a persisted, in this case to a H2 database.

5.3.1.3 Context Diagram

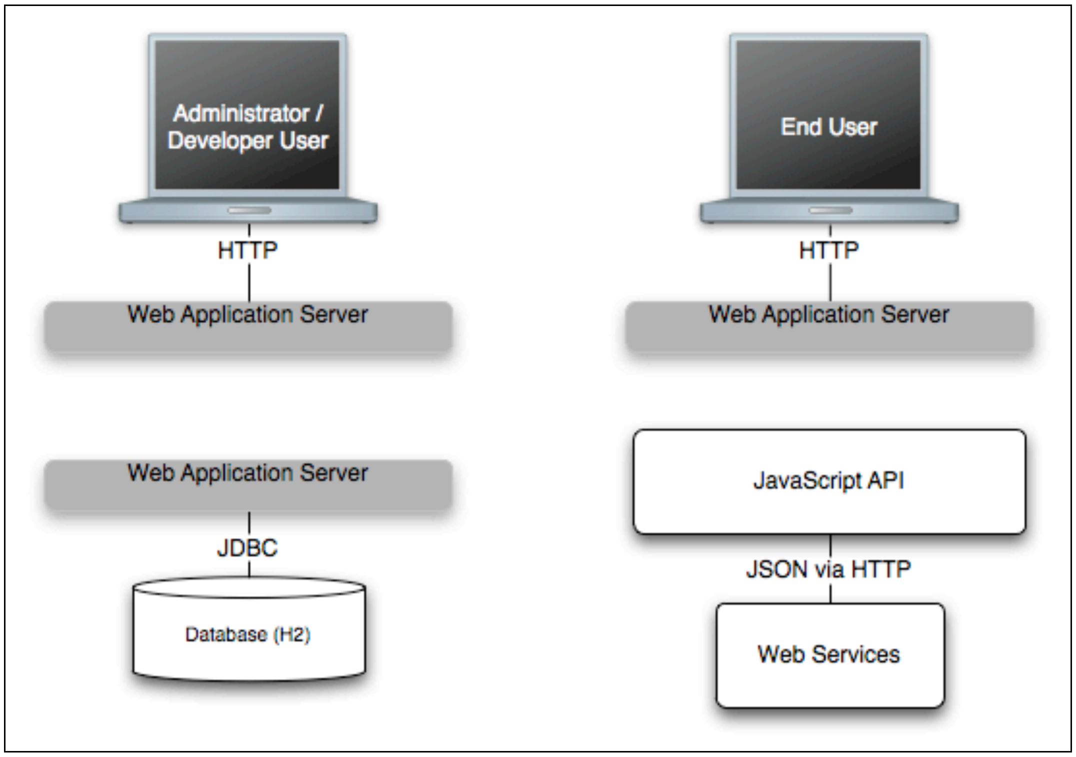

Figure 2: Context diagram for system architecture

5.3.2 Development View

5.3.2.1 Primary Presentation

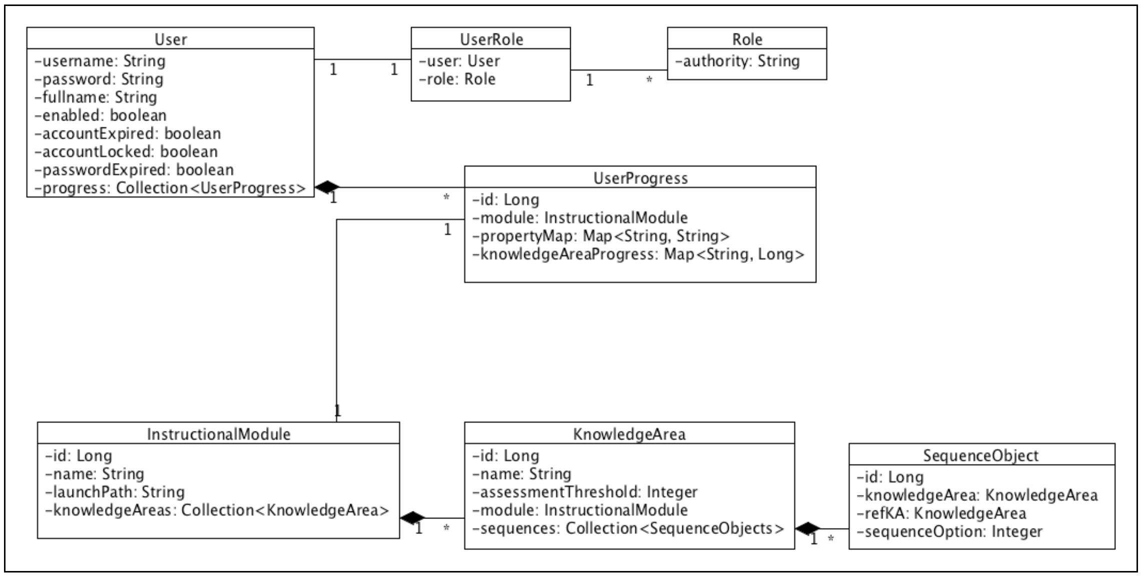

Figure 3: Class diagram depicting major domain classes

5.3.2.2 Element Catalog

User:

Represents the User super class.

UserRole:

Links the User and Role model objects.

Role:

Can be that of an Adminisrator, Developer or End User

InstructionalModule:

Represents the InstructionalModule class, which has a one-to-many relationship with KnowledgeArea entities.

KnowledgeArea:

Represents the KnowledgeArea class, which has a one-to-many relationship with SequenceObject entities.

SequenceObject:

Maintains dependency information with other KnowledgeArea entities.

UserProgress:

Tracks userProgress in multiple knowledgeAreas and links a User with an InstructionalModule.

5.3.2.3 Context

The high-level class diagrams illustrated in Figure 3 provides a simplified view of the user and instructional module domains. The view is not complete by any means, but reflects some of the major entities to be implemented and their most important functions.

5.3.3 Process View

5.3.3.1 Primary Presentation

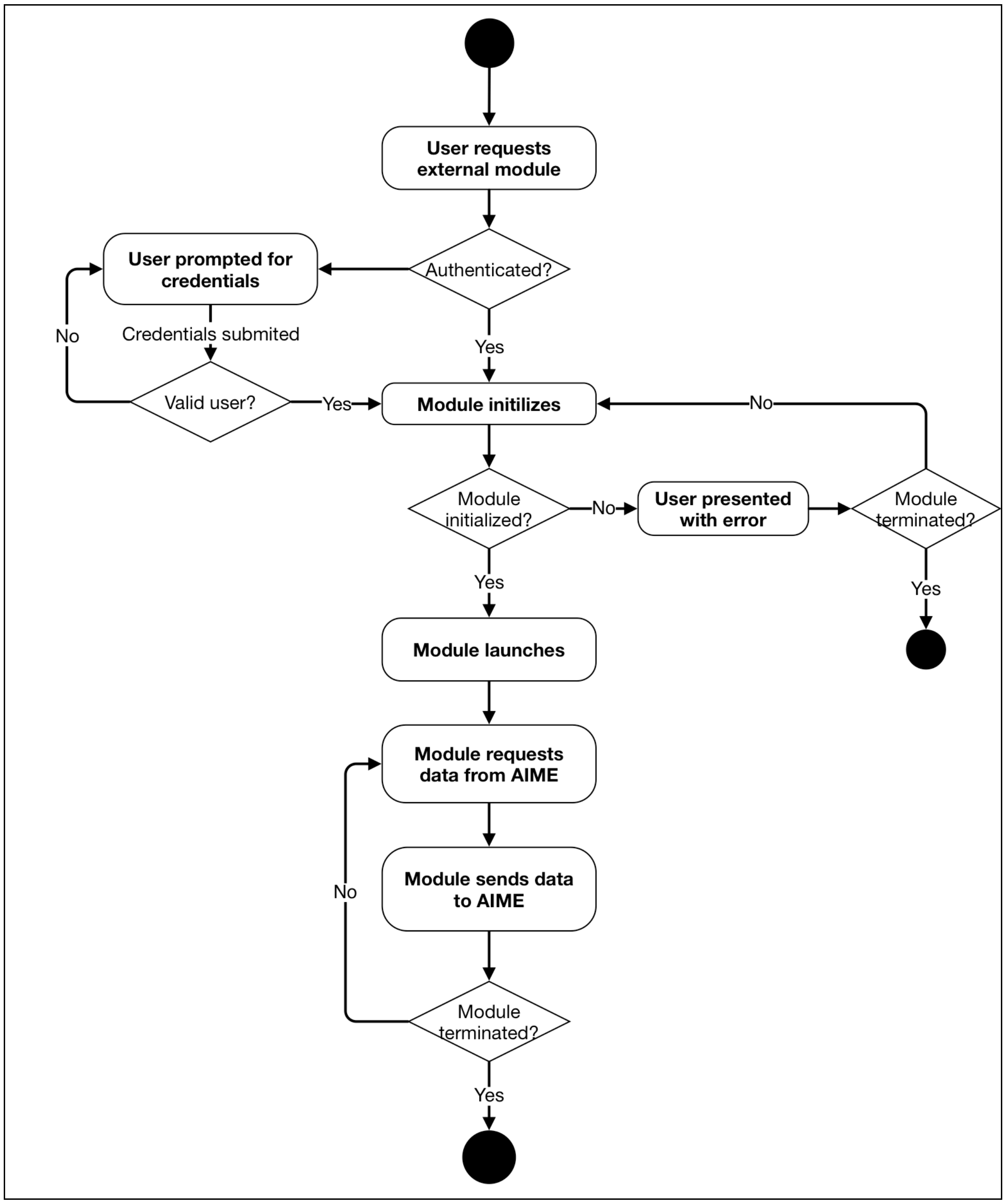

Figure 4: Process diagram for launching an external module

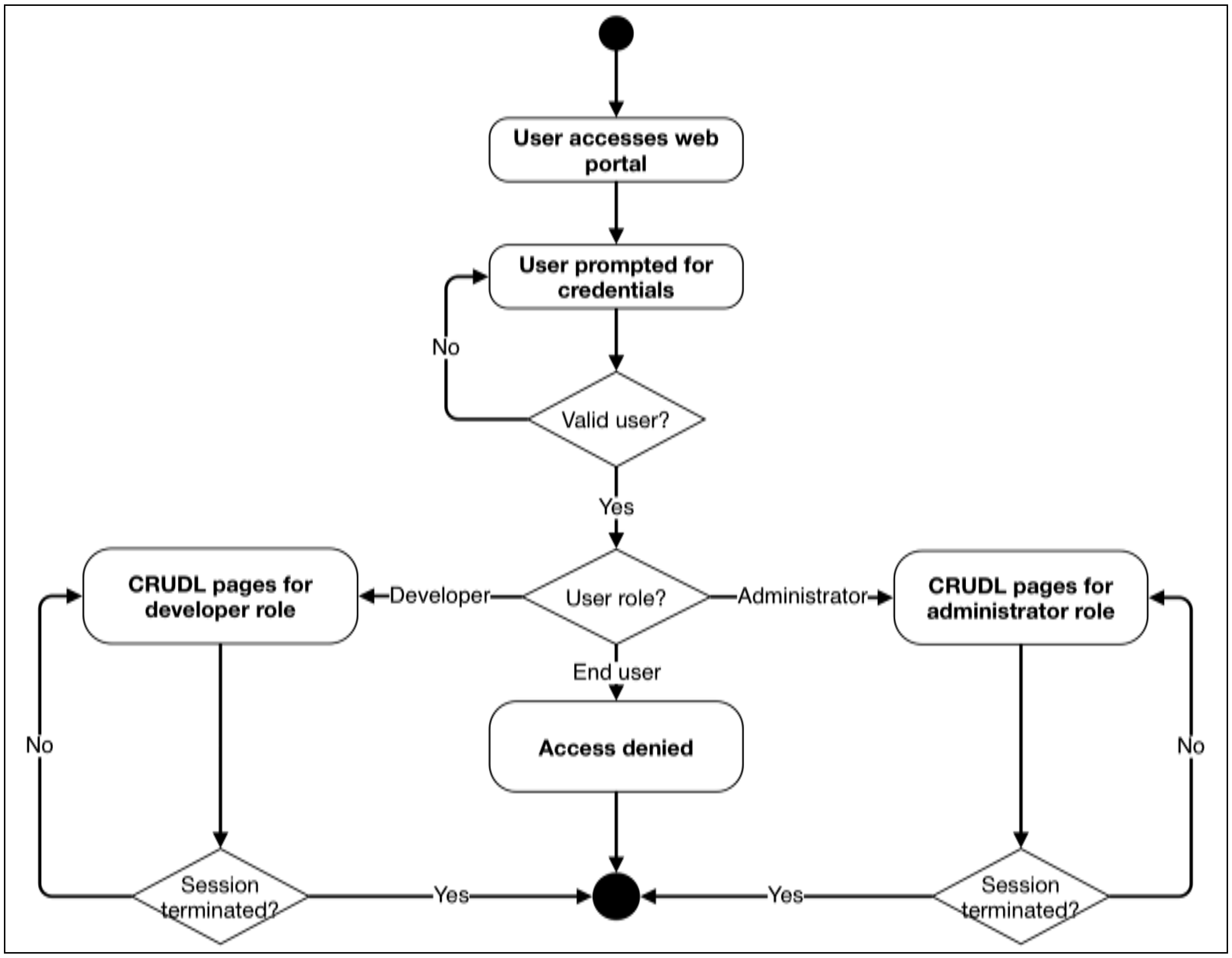

Figure 5: Process diagram for administrator and developer portal access

5.3.3.2 Context

Figure 4:

Describes the processes involved for an end user launching an external module.

Figure 5:

Describes the processes involved for a user accessing the web portal.

6 Implementation

6.1 Integrated Development Environment

The AIME project was developed within the SpringSource Tool Suite, an Eclipse-based integrated development environment (IDE). [18] This IDE was chosen for a number of reasons, chiefly the developer's familiarity with Eclipse-driven IDEs and the tight integration with the Grails framework and native support of the Groovy programming language. In terms of the development of AIME, virtually no work needed to be done outside of the IDE. The IDE was able to interface with Grails for all tasks, including the deployment of the web application to the installed Apache Tomcat server.

The installed version of SpringSource Tool Suite is 2.8.1.
The installed version of Apache Tomcat is 7.0.25.

The development environment runs Mac OS X 10.6.8. Testing environments will include Windows XP and Linux distributions running the same version of Tomcat.

6.1.1 Grails Installation and Setup

Installation of Grails and Groovy consisted of expanding a ZIP archive file for each onto the workstation hard drive and adding references to the home folders and binaries to the following environmental variables (in `~/.bash_profile` in this case):

```
export GROOVY_HOME=/groovy
export GRAILS_HOME=/grails
export PATH=$GRAILS_HOME/bin:$PATH
export PATH=$GROOVY_HOME/bin:$PATH
```

The installed version of Groovy is 1.8.6. The installed version of Grails is 2.0.0.

6.2 SCORM Implementation

An early inspiration for the implementation of AIME came in part from the design of the Sharable Content Object Reference Model (SCORM). SCORM itself is a collection of standards, not a standard in itself, which was compiled by the Advanced Distributive Learning (ADL) Initiative of the US Department of Defense. [21] The objective of SCORM is to provide a set of standards for developers of instructional content.

Since development started on AIME, it became apparent that SCORM is a bit robust for the needs of AIME. This is not to say that future iterations of AIME will not implement SCORM, but for now, attaching SCORM to the project entails a significant commitment to compliance, which was not possible within the scope of this project. Elements of SCORM that were implemented include the JavaScript API wrapper functions, because the set of functions seemed to fit well with the exchange model that AIME uses (see Section 3.2.2). The SCORM data model has not been implemented, however the notion of storing a generic set of key/value pairs to preserve the state of module runtime made sense for AIME and therefore, this too was implemented.

6.3 Application Elements

6.3.1 Domain Classes

In Grails, domain classes are Groovy classes that describe the model objects used by the underlying Hibernate Object Relational Mapping (implemented by Grails as *GORM*) to map persisted data to objects. AIME has the following domain classes:

com.btlee_tech.aime.module

- *InstructionalModule.groovy*
- *KnowledgeArea.groovy*
- *SequenceObject.groovy*

com.btlee_tech.aime.security

- *Role.groovy*
- *User.groovy*

- *UserRole.groovy*

com.btlee_tech.aime.userdata

- *UserProgress.groovy*

The domain classes in the *com.btlee_tech.aime.module* package represent the classes, which make up the structure of the instructional module as modeled previously in **Figure 3**.

The domain classes in the *com.btlee_tech.aime.security* package are initially laid down by the Spring Security Core plug-in, but have been extended to support AIME functionality.

The UserProgress domain class in the *com.btlee_tech.aime.userdata* package aggregates data related to user progress in the module. These classes essentially join InstructionalModule and User persistence objects.

6.3.2 Controllers

The controller classes do the bulk of the business logic. Grails can generate some of these, which can be useful for CRUDL views; however, some modifications need to be made for controllers that are not simply handling data that needs to be persisted. AIME has the following controller classes:

com.btlee_tech.aime.api

- *ModuleRestController.groovy*
- *UserProgressRestController.groovy*

com.btlee_tech.aime.module

- *InstructionalModuleController.groovy*
- *KnowledgeAreaController.groovy*

com.btlee_tech.aime.security

- *LoginController.groovy*
- *LogoutController.groovy*

- *UserController.groovy*

com.btlee_tech.aime.userdata

- *UserProgressController.groovy*

Controller classes in the *com.btlee_tech.aime.api* package are related to REST services, which are explained in more detail later.

Controller classes in the *com.btlee_tech.aime.module* package provide support to for CRUDL pages generated by the views.

Controller classes in the *com.btlee_tech.aime.security* package support the Spring Security Core plug-in and CRUDL pages for User persistence.

Controller classes in the *com.btlee_tech.aime.userdata* package support CRUDL pages for end users.

6.3.3 Views

Views in Grails render HTML using the Groovy implementation of Java Server Pages templates called Groovy Server Pages (GSP). Grails allows for automatic generation of GSP templates for controller classes that also have a corresponding domain class. Some of the GSP templates AIME has are generated automatically, but have been modified significantly. In practice, each controller method that is exposed through the user interface as HTML should have its own GSP template, though not all controller methods need to have a GSP template. Those that return non-HTML HTTP responses will not need a GSP template.

One special case is */userProgress/launchModule.gsp*, a custom-build template, which renders a launch, page for external modules. This template populates the rendered HTML with hidden content tags containing constant values used by the JavaScript on the page to inform the external module through the API.

6.3.4 REST Services

Representational State Transfer, or REST, constitutes a set of principles describing the use of web services using HTTP methods and URL addresses in such a way that each request is stateless in nature. [19] AIME implements RESTful services for the purpose of providing data relating to user progress in a module via two HTTP methods: GET and POST.

The RESTful web services that AIME exposes do not allow for creation or deletion of records, so use of only GET and POST methods is sufficient to meet the needs of the users. A GET request to */aime/api/userProgress/launchModule/[id].json* will return a JSON response of the data needed to inform the JavaScript function making the call. POST requests to the same URL with a JSON payload will be used to update the UserProgress properties, and will return back the updated object, again in JSON format. The sequencing mechanism uses the same calls to the same rest service. The objects returned in JSON are indexed as *userProgress* and *kaInfo*. The former contains module information and user progress related data, while the latter contains details relating to Knowledge Areas and sequencing.

6.3.5 JavaScript Libraries

The most important JavaScript library developed is modulelauncher.js. This library is loaded by the aforementioned *launchModule* GSP template. It pulls constant values from the page, which are stored in hidden content elements, into JavaScript variables upon load. It also contains functions that perform asynchronous GET and POST operations to the server, caching values into variables which are accessed by API wrapper functions that implement those described in the SCORM 2004 Run Time Environment documentation. [7]

6.4 Lessons Learned

In the course of development of the AIME proof of concept, much has been learned. While the overall design has produced a functional prototype, there have been mistakes along the way, and this document would be remiss to exclude those mistakes from the final report. In the spirit of intellectual integrity, an honest attempt at a retrospective analysis has been compiled below for each area in which challenges arose.

6.4.1 Sequencing Components

The largest challenge to implementation of AIME came in the sequencing components. In the original design, there were assumptions made that the SCORM standard would be utilized and the sequencing elements for the latest revision of SCORM could be easily plugged into the implementation. Once it became apparent that SCORM did not exactly meet the needs of AIME, and in fact was overkill in some aspects, the sequencing element was left without a detailed design. Implementation was started on the sequencing element before the design was fully nailed down, so to speak.

In future revisions, a new design will need to be implemented for sequencing to make it function as a type of constraint on the runtime of the module. In addition, more of the sequencing heavy lifting will need to be pulled into the application side, out of the JavaScript on the client side, to allow for a more optimized sorting algorithm for concurrent groups of knowledge areas. Since the sequencing components are well encapsulated and not exposed to the content developers, these changes should not impact the public API for module developers.

6.4.2 User Interface

Though the upfront work on the user interface produced a decent presentation, it became apparent that since this framework is being developed for power users and developers, an alternative approach using XML might be preferable.

Future revisions will feature an option to import an XML file, whose format will be dictated by a standard Document Type Definition (DTD) file. This would allow a user to detail the components of a module and its knowledge areas and sequencing quickly and in a way that is portable and can be versioned. It is possible that the XML file could just be placed in the module folder, and as long as all components are in their proper place no access to the AIME UI would be necessary to load a module. A script might even be devised to allow the creation of stubbed JavaScript modules based on the definitions provided in the XML.

Other improvements for consideration include a fully headless option that would make full use of RESTful services for communication, opening the possibility for rich client access, including but not limited to, mobile applications and external systems integration. AIME could even become a plugin for use in development on non-Java technology stacks.

6.4.3 Security Concerns

The Spring security plug-in provided by Grails made it very easy to lock down certain controllers, but due to time constraints an in-depth audit of security features was not possible. Better planning and allowances for testing in this area would benefit future revisions.

6.4.4 Logging Concerns

Debug and user-level logging were not areas that really given much emphasis in this proof of concept. Future revisions would need to make sure that this area is covered in more detail.

7 User Documentation

In the following sections, details relating to user documentation, specifically those relating to deployment, creation of instructional modules and accessing the example module are detailed. Future revisions of AIME, once it is hosted on an open source project website, will likely be provided in a continually updated wiki-style, self help website.

7.1 Deployment Instructions

To deploy the AIME web application, perform the following steps:

1. Download and install the Apache Tomcat version 7 application server (AIME's runtime environment) from: http://tomcat.apache.org/download-70.cgi
2. Locate the *aime.war* web application archive file in the */distribution* directory on the CD-ROM media and place it in *webapps* subdirectory of the Tomcat 7 install location
3. Launch the Tomcat 7 application server. If already started, be sure to restart the service.
4. Access AIME by pointing your web browser to http://localhost:8080/aime or, if working from a different workstation, http://[serverIPAddress]:8080/aime.

Default Access Accounts

Three accounts are seeded with AIME. Credentials for each default account are as follows:

Developer User

Username: `defDev`

Password: `dev123`

Admin User

Username: `defAdmin`

Password: `admin123`

End User

Username: `defUser`

Password: `user123`

7.2 Instructional Module Creation

In order to develop a module leveraging AIME's adaptive sequencing, the following sequence of events is suggested:

1. Login to AIME and select Instructional Modules
2. Create a new Instructional Module.
3. Once submitted, add Knowledge Areas and assessment thresholds. Assessment thresholds indicate the level of points at which a knowledge area is considered complete.
4. After Knowledge Areas have been created and the Instructional Module record has been updated, the sequencing for each Knowledge Area can be added. If this step is not performed, all knowledge areas are assumed to be concurrent. **Note:** Once sequencing has been added, your work within the AIME admin portal is complete.
5. Create the directory structure for your module and knowledge areas within the *modules* directory at the root-level *web-app* directory. The expected directory structure for JavaScript modules is detailed under *Conventions*, later in this section.
6. Create the required JavaScript module files as detailed below.

Details relating to each of these events, as well as the API are provided later in this section.

7.2.1 Conventions

7.2.1.1 Paths

- */aime/modules/[moduleName]*
    - The default location for all main module JavaScript content

- */aime/modules/[moduleName]/[knowledgeAreaName]*
    - The default location for all knowledgeArea content

7.2.1.2 Placement of Required JavaScript

Each module folder in modules must have a moduleMain.js file. This JavaScript module needs to have an *launch()* function to be called by the module launcher. The *launch()* function will need to initialize any values it needs, and call the *Initialize()* function in the JavaScript global namespace. The moduleMain.js file must implement public functions to handle presentation whereas the built in *sequenceManager.js* communicates with the available knowledge areas and determines sequence.

To get the next assessment, a call will need to be made to the *knowledgeArea.js* file in any of the KnowledgeArea subdirectories. Which knowledge area gets called depends on which knowledge areas are within sequencing scope. For example, two knowledge areas that are mutually concurrent will both be listed as available at a given time. Once one knowledge area is completed (i.e., the accumulated score is equal to or greater than the assessment threshold set for that knowledge area) the complete knowledge area is removed from the pool and any knowledge area that is listed as succeeding the completed knowledge area will be evaluated to be added to the pool of available knowledge areas for assessment. As long as it is not in conflict with any other uncompleted knowledge area, it will be added to the pool. Two knowledge areas will be in conflict if they are not mutually concurrent.

7.2.1.3 RequireJS

RequireJS provides the structure and encapsulation of JavaScript module files in AIME. RequireJS makes the some of the more dynamic features of AIME's JavaScript libraries possible. [20] The RequireJS core library is included on the moduleLauncher page, which initializes the variables and functions necessary to start a module. To find out more about RequireJS, including how to create and call a typical RequireJS module, visit http://requirejs.org/ or review the structure of the ExampleModule included within the *modules* subdirectory of the AIME root *web-app* directory.

7.2.1.4 Required JavaScript Objects

Communication between the module and knowledge areas is done by passing JavaScript objects, containing a payload of content and instruction on how evaluate the results of the user interaction. See the comment lines in the following examples for information relating to the object structure. The names of the object properties are as they should be in the following examples, but the values provided are only for illustration. For example, the double-quotes `""` indicate a text type of some sort, the zero (0) indicates an integer value, the square brackets `[]` indicate an array of values, and curly braces `{}` indicate an embedded JavaScript object.

From the knowledgeArea to moduleMain JavaScript module:

```
assessment = {
      assessmentId : 0,        // Integer value uniquely identifying this
                               // assessment out of all others in the
                               // knowledgeArea catalog.

      knowledgeArea : "",      // Name of knowledgeArea for which to return
                               // response.

      resultType : "",         // Possible options "multiple" or "single"

      content : "",            // HTML or XML to be rendered by moduleMain

      contentType : "",        // May be HTML, TEXT, XML or any other
                               // formatted messaging.  Presentation is
                               // handled by moduleMain at the implementation
                               // of the developer.

      possiblePoints : 0,      // Integer value to be assigned

      answerMap : {}           // JavaScript object mapping responses to
                               // integer values, provided in key : value
                               // pairs
}
```

The answerMap object is structured as follows:

```
answerMap : {
      correctAnswers : [""],   // Array of strings indicating potentially
                               // correct answer values.

      possibleAnswers : [""]   // Array of strings with formatted answers to be
                               // presented by moduleMain.
}
```

From the moduleMain to knowledgeArea JavaScript modules:

```
response = {
      assessmentId : 0,       // Integer value uniquely identifying this
                              // assessment out of all others in the
                              // requesting knowledgeArea catalog

      knowledgeArea : "",     // Name of knowledgeArea for which to
                              // return a response

      results : [],           // Array of responses to the assessment

      pointsAwarded : 0       // Integer value of all points awarded
}
```

7.2.2 JavaScript API

External modules in AIME are expected to follow the Model-View-Controller design pattern. This pattern is implemented in AIME such that *knowledgeArea.js* modules provide the model, the core *sequencemanager.js* module acts as a controller and *moduleMain.js* presents the information as the view. This allows developers to add, edit or remove knowledgeAreas without having to change controller or presentation code. See the following templates, with inline comments, for the interfaces to be implemented in the required JavaScript modules.

moduleMain.js:

```
define(['jquery'], function($j){
      //Private functions and variables go here//

      //Public functions and variables start//
      var mm = {
                  launch : function() {
                        // Sets up the the page or presentation target and
                        // calls Initialize() once done.
                  },

                  present : function (assessment) {
                        //Handles presentation of the assessment object
                  },

                  evaluate : function() {
                        // This function is called when the user's response
                        // is submitted. This function should construct the
                        // response from the submission, and call
                        // sequenceManager.submitResponse(response), then
                        // sequenceManager.getNextAssessment() should be
                        // called.
                  },

                  terminate : function() {
                        // Should wrap up any other presentation business
                        // and call Terminiate();
                  }
      };
      return mm;
```

knowledgeArea.js:

```
define(['jquery'], function($j){
      //Private functions and variables go here//

      //Public functions and variables start//
      var ka = {
                  getAssessment : function () {
                        // getAssessment should return an assessment object
                        // when called
                  },

                  respond : function (response) {
                        // You can do some internal tracking of responses
                        // here, or you can just save to the propertyMap
                        // from moduleMain
                  }
      };
      return ka;
      //Public functions and variables end//
```

For more information on the available on functions available to call from your modules, see *modules/ExampleModule* or the core JavaScript libraries *modulelauncher.js* and *sequencemanager.js* in the AIME *web-app* root directory.

7.2.3 Accessing the Example Module

All users types should be able to log into AIME and access the Example Module. The steps to do so are as follows:

1. Log into AIME
2. Select *User Progress / Registrations* from the Home page

   Please Note: Only users with the Admin role will be able to see all user progress records. If there is no UserProgress record, the user will need to be registered to access the module under *Manage Users* from the Home page.
3. Click the *Launch* hyperlink next to the UserProgress record of your choosing
4. Click *Start Module*

8 Conclusion

The trend in software development in the last decade has been toward writing less code. This can be observed in the popularity of the current generation of software development frameworks. Development frameworks generally focus on providing the developer the common tools necessary to perform the redundant, dull and potentially error prone tasks necessary to recognize their vision. [3] Going a step beyond the more general-purpose development frameworks, there is an opportunity for frameworks (and entire platforms) that speak to specific purposes to reduce new development effort even more so.

The Adaptive Instructional Module Engine (AIME) provides a proof of concept, which illustrates the possibility for reduced development effort for interactive, adaptive web applications, tutorial modules and games. This specific purpose is one that has tremendous market force to support it at the current time. Companies and individuals responsible for some of the most popular games on social media, mobile and web platforms, have a need to take their vision to market in short time in order to remain competitive. Specific-purpose platforms like AIME may help to shave off months of development-hours from a game project, allowing game developers to spend more time on realizing the vision. In education, there is often not much funding for large software development projects, however there is still a need to continually update instructional offerings. Providing content developers with the tools to make their instructional modules more adaptive to the learner without the need to code so much of the business logic would improve the offerings available for education and training.

AIME represents the first step toward more specific-purpose frameworks of this type and with some further refinement through ongoing development, has the potential to have a great impact for its target audience. As web frameworks come and go, so might AIME, if it is even fortunate enough to gain traction at all, but the concept of specific-purpose frameworks like what AIME intends to be, is not likely to be so transient. As developers are pushed to produce similar projects at an increasingly rapid pace, it seems that there is no alternative than development of a variety of specific-purpose frameworks to meet those similar needs. The author intends to exercise AIME in the real world with the development and release of a web-based adaptive game in the latter half of 2012. During the course of development for said game, updates will be made to the AIME server software to make it more robust and usable for general availability, all prior to an open source release.

9 References

[1] *API*, from Webopedia [2011]. Available: http://www.webopedia.com/TERM/A/API.html

[2] Len Bass, Paul Clements and Rick Kazman. *Software Architecture in Practice, Second Edition*. [2003] Addison-Wesley, pp 79-90.

[3] *Web application framework*, from DocForge [2011]. Available: http://docforge.com/wiki/Web_application_framework

[4] SpringSource, *Grails & Grails – Your Quest for Productivity Ends Here*. Available: http://www.springsource.com/developer/grails

[5] *H2 Database Engine.* Available: http://www.h2database.com

[6] Advanced Distributed Learning, *SCORM 2004 4th Edition*. Available: http://www.adlnet.gov/capabilities/scorm/scorm-2004-4th

[7] Advanced Distributed Learning, *SCORM® 2004 4th Edition Documentation Suite* Version 1.1., SCORM 2004 4th Edition Specification. Available: http://www.adlnet.gov/wp-content/uploads/2011/07/SCORM_2004_4ED_v1_1_Doc_Suite.zip

[8] Glen Smith, Peter Ledbrook, *Grails in Action* [2009], Manning Publications Co., pp 3-30, 36, 155-187, 194-195, 280-309.

[9] Daniel Mikusa, *Performance Tuning the JVM for Running Apache Tomcat* [2011]. Available: http://www.tomcatexpert.com/blog/2011/11/22/performance-tuning-jvm-running-tomcat

[10] Tomcat Wiki, *FAQ/Developing*. Available: http://wiki.apache.org/tomcat/FAQ/Developing

[11] *Definition: ADDIE Instructional Design Process*, from University Information Technology at Tufts University, Encyclopedia for Teaching with Technology [2007]. Available: https://wikis.uit.tufts.edu/confluence/display/UITKnowledgebase/ADDIE+Instructional+Design+Process

[12] Robert M. Gangne, Walter W. Wager, Katharine C. Golas and John M. Keller, *Principles of Instructional Design*, 5th Ed. [2004], Wadsworth Publishing.

[13] Kris Rockwell, *Implementing SCORM in Serious Games* [2011]. Available: http://www.slideshare.net/hybridlearning/implementing-scorm-in-serious-games-10105769

[14] The Apache Software Foundation, *Apache Tomcat* [2012]. Available: http://tomcat.apache.org/

[15] *jQuery: The Write Less, Do More, JavaScript Library.* Available: http://jquery.com/

[16] Derek Greer, *The Art of Separation of Concerns* [2008]. Available: http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/

[17] Scott Davis and Jason Rudolph, *Getting Started with Grails, Second Edition* [2010]. Available: http://www.infoq.com/minibooks/grails-getting-started

[18] SpringSource, *SpringSource Tool Suite - The Best Development Tool for Enterprise Java* [2011]. Available: http://www.springsource.com/developer/sts

[19] Alex Rodriguez, *RESTful Web services: The basics* [2008], IBM developerWorks. Available: http://www.ibm.com/developerworks/webservices/library/ws-restful/

[20] *RequireJS* [2011]. Available: http://requirejs.org/

[21] *SCORM Explained*. Available: http://scorm.com/scorm-explained/

10 Appendices

10.1 Index of Figures

10.2 Source Code

See attached CD-ROM media.