Moodle Course Content Access Control System

Software Architecture Document

Version 1.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Architectural Representation 5

2.1 Logical view 5

2.2 Process view 5

2.3 Implementation view 5

2.4 Physical view 5

2.5 Use Case view 6

3. Architectural Goals and Constraints 6

4. Use-Case View 6

4.1 Use-Case Realizations 6

5. Logical View 7

5.1 Overview 7

5.2 Architecturally Significant Design Packages 7

6. Process View 7

7. Deployment View 7

8. Implementation View 7

8.1 Overview 7

8.2 Layers 7

9. Data View (optional) 7

10. Size and Performance 8

11. Quality 8

Software Architecture Document

# Introduction

[The introduction of the **Software Architecture Document** provides an overview of the entire **Software Architecture Document**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **Software Architecture Document**.]

## Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

## Scope

[A brief description of what the Software Architecture Document applies to; what is affected or influenced by this document.]

This Software Architecture Document establishes the top-level design and overall architecture for the Course Content Access Control System (CCAC) of Moodle. The documents explain the design details from different views – where a view is a description of the entire sub-system from a single perspective.

## Definitions, Acronyms, and Abbreviations

[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the **Software Architecture Document**.  This information may be provided by reference to the project’s Glossary.]

CCAC: Moodle Course Content Access Control System

RUP: Rational Unified Process

UML: Unified Modeling Language

SAD: Software Architecture Document

Resource: With respect to the Moodle, a resource is a any entity added by the course conductor to the course content. This can be a Lecture note, video, assignment, quiz, choice…etc.

## References

[This subsection provides a complete list of all documents referenced elsewhere in the **Software Architecture Document**. Identify each document by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

## Overview

[This subsection describes what the rest of the **Software Architecture Document** contains and explains how the **Software Architecture Document** is organized.]

The rest of the Software architecture document discuss each view in the “4+1” view model. This model contains the Logical View, Physical View, Process View, Implementation View and the Scenario. This document also discuss about the performance issues and the quality of the extension that has a major impact on the architecture.

# Architectural Representation

[This section describes what software architecture is for the current system, and how it is represented. Of the **Use-Case**, **Logical**, **Process**, **Deployment**, and **Implementation Views**, it enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]

This document presents the architecture as a series of views; use case view, logical view, process view and deployment view. There is no separate implementation view described in this document. These are views on an underlying Unified Modeling Language (UML) model developed using Visual Paradigm.

In order to depict the software as accurately as possible, the structure of this document is based on the “4+1” model view of architecture [KRU41].

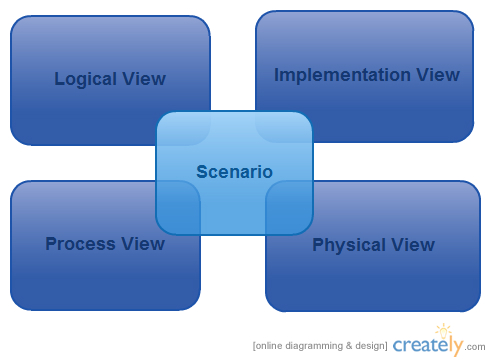


Figure 1-"4+1" model view

The “4+1” View Model allows various stakeholders to find what they need in the software architecture.

## Logical view

**Audience**: Designers.

**Area**: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations.

This is represented using block and line diagrams.

**Related Artifacts**: Design model

## Process view

**Audience**: Integrators.

**Area**: Non-functional requirements: describes the design's concurrency and synchronization aspects.

This discusses about the processes and how they communicate with each other.

**Related Artifacts**: UML activity diagrams

## Implementation view

**Audience**: Programmers.

**Area**: Software components: describes the layers and subsystems of the application. It is concerned with the management of the software.

**Related Artifacts**: Implementation model, components

## Physical view

**Audience**: Deployment managers.

**Area**: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects.

**Related Artifacts**: UML deployment diagram.

## Use Case view

**Audience**: all the stakeholders of the system, including the end-users.

**Area**: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system.

**Related Artifacts** : Use-Case Model, Use-Case documents

# Architectural Goals and Constraints

[This section describes the software requirements and objectives that have some significant impact on the architecture; for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.]

## Constraints

This is a project which develops a sub system for the existing Moodle system. So, in order to keep the integrity with the existing system, this component should not violate the existing architecture of the system. Hence, this is based on the existing architecture of the Moodle.

## Goals

* Usability  
  The required training time for a normal user to be productive should be about 10 minutes and for the power user about 5 minutes.
* Reliability   
  There cannot be any chance to show private content to the public users.
* Performance   
  The system should not make a user noticeable delay in normal function without CCAC due to CCAC activities.

# Use-Case View

[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]

Even though there are many use cases related to the functionality of the Moodle, only some use cases are related to the CCAC. Those are listed below.

1. Installation/ uninstallation CCAC
2. Add resource
3. Change resource access level
4. View course
5. Get resource

## Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality.]

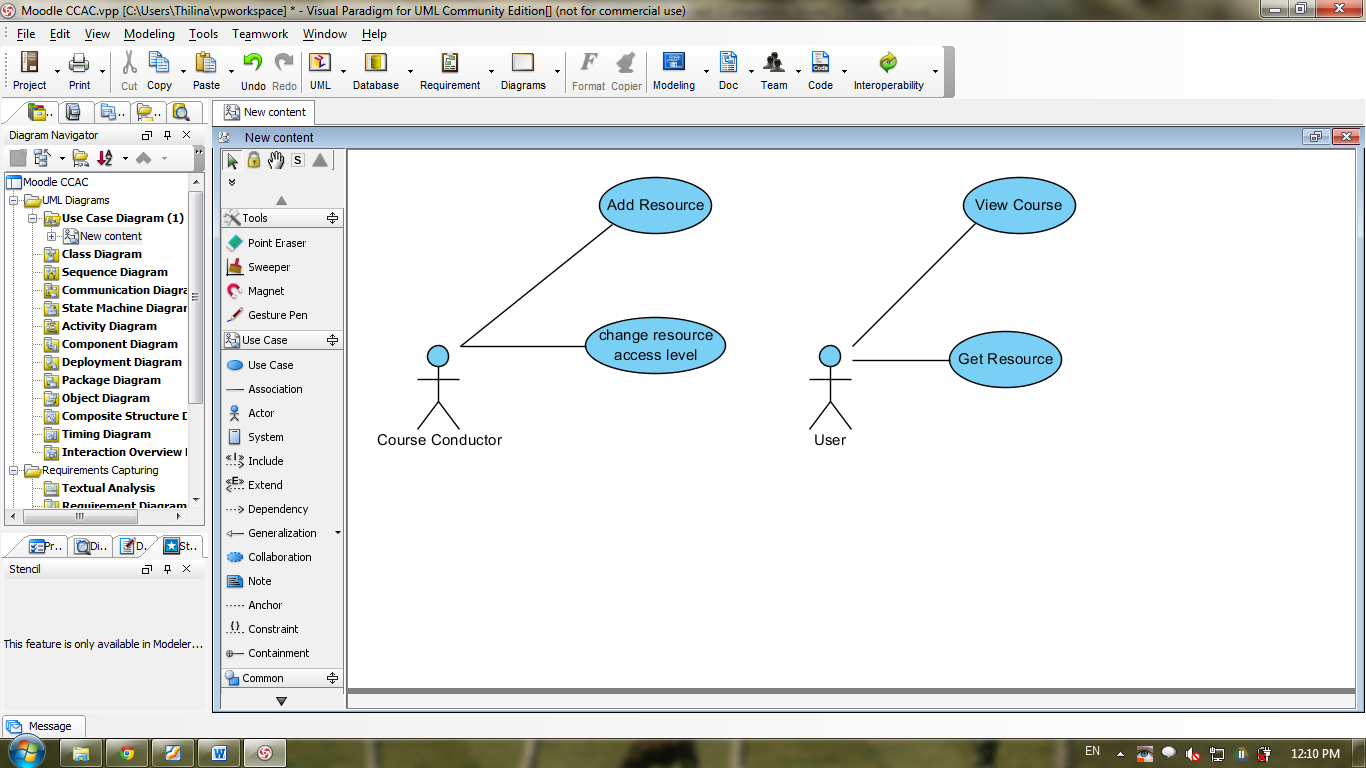


Figure 4.1

# Logical View

[This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.]

This is a description of the logical view of the architecture. This describes the most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers.

Also describes the most important use-case realizations, for example, the dynamic aspects of the architecture. Class diagrams may be included to illustrate the relationships between architecturally significant classes, subsystems, packages and layers.

## Overview

[This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.]

As this is a sub component development project for Moodle, overall architecture of the Moodle is important for the project.

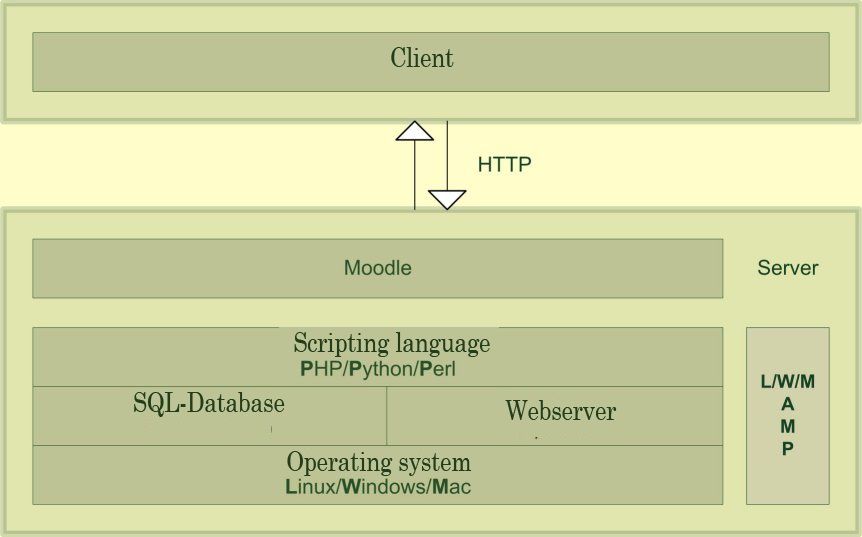


Figure 5.1

## Architecturally Significant Design Packages

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

# Process View

[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]

# Deployment View

[This section describes one or more physical network (hardware) configurations on which the software is deployed and run. It is a view of the Deployment Model. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs) that execute the software and their interconnections (bus, LAN, point-to-point, and so on.) Also include a mapping of the processes of the **Process View** onto the physical nodes.]

# Implementation View

[This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.]

## Overview

[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers. ]

## Layers

[For each layer, include a subsection with its name, an enumeration of the subsystems located in the layer, and a component diagram.]

# Data View (optional)

[A description of the persistent data storage perspective of the system. This section is optional if there is little or no persistent data, or the translation between the Design Model and the Data Model is trivial.]

# Size and Performance

[A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.]

# Quality

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]