CEN 5064 Software Design

**Cloud Provisioning System(CPS)**

**Group #1**

**Dionny Santiago**

**Edwin Malek**

**Xiaoyu Dong**

**February 17, 2015**

**Professor: Dr. Peter Clarke**

# Abstract

This document goes over the specifications for the Cloud Provisioning Graphical Editor System that team 1 is building, using of the EMF/GMF eclipse platform. This document defines the necessary functionalities that the future system will provide to our users. Through the use of UML diagrams, we will attempt to define our system and its responsibilities to our users (feature diagrams, class diagrams, sequence diagrams, activity diagrams, object diagrams, use case diagrams, etc…). We will also use a verbal description of each of our use cases along with specifically chosen scenarios for the most important use cases. Our system will be divided into a modeling environment and an Execution environment, both of which could in theory function independent of each other.

Cloud provisioning is still a fairly new technology, with great potential, but the systems today lack the visual representation in order for the less tech-savvy users to be able to see a complex system and tailor the service to their needs.

Contents

[Abstract 2](#_Toc414140678)

[1         Introduction 5](#_Toc414140679)

[1.1       Purpose of the System 5](#_Toc414140680)

[1.2    Functional and Nonfunctional Requirements 5](#_Toc414140681)

[**Functional Requirements** 5](#_Toc414140682)

[**Non-Functional Requirements** 7](#_Toc414140683)

[1.3    Design Methodology 7](#_Toc414140684)

[1.4       Definitions, acronyms, and abbreviations 9](#_Toc414140685)

[1.5 Overview of document 10](#_Toc414140686)

[The rest of the document contains more detailed information about the various architectural and design decisions made throughout development process of the CProv systems. The document is laid out in three more chapters and an appendix chapter. 10](#_Toc414140687)

[Chapter 2 covers our proposed system software architecture. We begin by presenting the major subsystems and overviewing each of the subsystems. In addition, we present our architectural pattern choices and discuss our reasoning process for selecting such patterns. An UML profile for our architecture is presented, along with a description of our generative architecture. Finally, we conclude this chapter with details about subsystem decomposition. 10](#_Toc414140688)

[Chapter 3 covers our object design. We begin by presenting a minimal class diagram for the subsystems to be implemented. Furthermore, we present refined sequence diagrams from the analysis model and a state machine for the main control object in each of the major subsystems. We conclude the chapter by presenting a detailed class design, including the meta-model used for the CProv systems, and OCL constraints for the static model. 10](#_Toc414140689)

[Chapter 4 provides a glossary of terms used in this document. 10](#_Toc414140690)

[Chapter 5 provides various appendices to further detail our design choices and log the various meetings and tasks assigned to each team member throughout the course of developing this Design Model. 10](#_Toc414140691)

[2         Proposed System Software Architecture 11](#_Toc414140692)

[This chapter covers our proposed system software architecture. We begin by presenting the major subsystems and overviewing each of the subsystems. In addition, we present our architectural pattern choices and discuss our reasoning process for selecting such patterns. An UML profile for our architecture is presented, along with a description of our generative architecture. Finally, we conclude this chapter with details about subsystem decomposition. 11](#_Toc414140693)

[2.1       Overview 11](#_Toc414140694)

# 1         Introduction

## 1.1       Purpose of the System

In this section we will discuss the purpose of our system, along with the scopes of the system, as well as a list of definitions, acronyms, and abbreviations used throughout the document.

## 1.2    Functional and Nonfunctional Requirements

The purpose of our system will be to allow a user to create a visual representation of today’s cloud provisioning services offers by Amazon, Google and others. While still retaining the possibilities to edit and configure his/her cloud environment and most important deploy the visually created system using the Amazon or google Cloud provisioning APIs

**Functional Requirements**

1. The system shall allow cloud administrators to create CPS models based on the CPROV language.

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):**CPROVME001 Create New Model

2. The system shall allow cloud administrators to translate the existing model into a CPROV-XML representation capable of deployment through the CPS appropriate API calls.

**Use cases:** All use cases related to General Execution Environment.

**Use Case ID(s):**CPROVEE004 Transformation to Amazon EC2 Call(s)

3. The system shall allow cloud administrators to print their existing model to XML representation.

**Use cases:** All use cases related to General Execution Environment.

**Use Case ID(s):**CPROVEE005 Print to XML

4. The system shall allow cloud administrators to validate their CPS models against a set of rules based on current CPS systems such as the Amazon EC2.

**Use cases:** All use cases related to General Execution Environment.

**Use Case ID(s):**CPROVME017 Validate Model,

5. The system shall allow cloud administrators to modify their models, specifically the properties of the nodes present on the diagram, before any deployment to CPS is necessary.

**Use cases:** All use cases related to Modeling Environment.

**Use Case ID(s):** CPROVME008 Edit Node properties.

6. The system shall allow for a cloud administrator to import an CPROV XML file, and transform it to a CPS model (diagram and tree)

**Use cases:** All use cases related to General Execution Environment.

**Use Case ID(s):** CPROVEE006 XML to Model.

7. The system allow for the cloud administrator to add an Environment Node to the previously created model

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):** CPROVME003 Add Environment Node

8. The system allow for the cloud administrator to add an Instance Node to the previously created model

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):** CPROVME005 Add Instance Node

9. The system allow for the cloud administrator to add an security group Node to the previously created model.

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):** CPROVME004 Add security group Node

10. The system allow for the cloud administrator to add a storage Node to the previously created model.

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):** CPROVME006 add Storage Node

11. The system allow for the cloud administrator to add a Network Node to the previously created model.

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):** CPROVME007 Add Network Node

12. The system allow for the cloud administrator to delete a Node on the previously created model.

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):** CPROVME013 Delete Node

13. The system allow for the cloud administrator to add a connection between two nodes on the previously created model.

**Use cases:** All use cases related to General Modeling Environment.

**Use Case ID(s):** CPROVME014 Add Connection

14. The system allow for the cloud administrator to load an existing model previously created.

**Use cases:** All use cases related to General Execution Environment.

**Use Case ID(s):** CPROVME016 Load Existing Model.

15. The system allow for the cloud administrator to Save an existing model.

**Use cases:** All use cases related to General Execution Environment.

**Use Case ID(s):** CPROVME018 Save Model.

CPROVME001 Create New Model, CPROVME002 Add Model Element, CPROVME003 Add Environment Node, CPROVME004 Add security group Node, CPROVME005 Add Instance Node, CPROVME006 add Storage Node, CPROVME007 Add Network Node, CPROVME008 Edit Node properties, CPROVME009 Edit Instance Properties, CPROVME010 Edit security Group Properties, CPROVME011 Edit Storage Properties, CPROVME012 Edit Network Properties, CPROVME013 Delete Node, CPROVME014 Add Connection, CPROVME015 Delete Connection, CPROVME016 Load Existing Model, CPROVME017 Validate Model, CPROVME018 Save Model.

**Non-Functional Requirements**

1. Usability:

a)      No previous Training Time

c)      On average the user should be able to submit a deployment request in under 1 minute.

2. Reliability:

a)      Mean time to Failure – 5% failures for every 1000 attempts to add an instance.

b)      Availability – The system shall be available as long as the user’s machine is running.

3. Performance:

a)      Deployment of the current workspace diagram should complete in under one minute.

b)      System handles only one request for deployment at a time.

4. Supportability:

a)       The execution environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

5. Implementation:

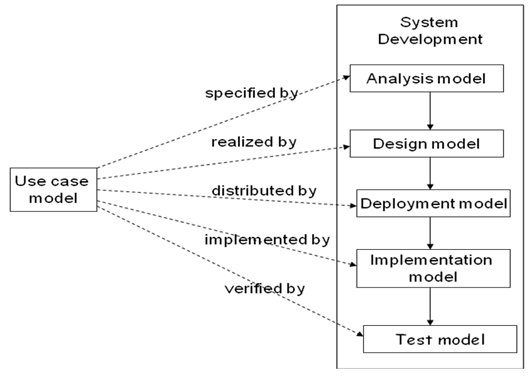
a)       The software will be implemented via a Eclipse plugin that the user can install and run.

## 1.3    Design Methodology

The software development model that has been used in the project is Unified Software Development Process (USDP). It is a component-based, use case driven, architecture centered, iterative and incremental developmental process that uses the Unified Modeling Language (UML) to represent models of the software system to be developed. The USDP model is use-case driven and is built on top of traceable relationships. See Figure 1-1 for an illustration on USDP.

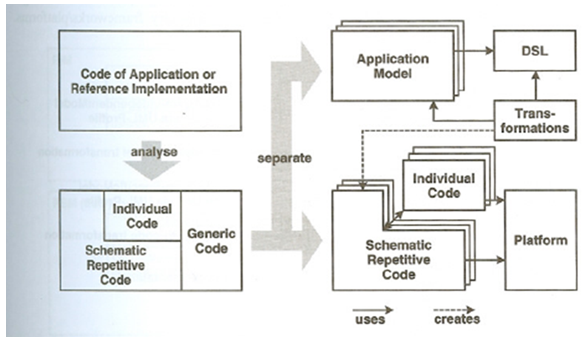
The iterative and incremental features help refine the final product as we get to know specific implementation platforms, namely Eclipse EMF/GMF and the Amazon EC2 library. The use case approach for gathering the systems requirements was also suitable to collect the functional requirements during the analysis phase of the project.

In addition, we eased the design of the system by using architectural and design patterns. The architectural patterns used are: Model View Controller, Pipe and Filter. We used the UML 2.0 notation for specifying the different artifacts of the system. The UML models used in the project are: uses case diagrams, class diagrams, sequence diagrams, activity diagrams, UML profiles.



**Figure 1-1 The Unified Software Development Process**

The USDP approach was complemented with the use of the Model Driven Software Development (MDSD) approach. Domain analysis, Meta-modeling, model-driven generation, template languages, domain-driven framework design, and the principles for agile software development form the backbone of this approach. MDSD models have the exact meaning of program code in the sense that the bulk of the final implementation can be generated from them. The main reason for using the MDSD approach was to increase the development speed. Since by automation run able code is generated from the formal models using one or more transformation steps. MDSD manages the complexity through abstraction since modeling languages enable programming or configuration on a more abstract level. Reusability was another big reason this approach was chosen, once all our architectures, transformations, and modeling languages have been defined they can be later used in a software product line for manufacturing diverse software systems. See Figure 1-2 for an illustration on MDSD.



**Figure 1-2 The Model Driven Software Development Process**

The total approach is the combination of the unified software development process and the model driven software development process. Unified Software Development Process uses the Model Driven Software Development which results in an increase in development speed, better software quality, reusability, interoperability and the portability.

## 1.4       Definitions, acronyms, and abbreviations

EE: Execution Environment

ME: Modeling Environment

XML:  Extensible Markup Language

XML: file format respecting a certain syntax

CPROV: Cloud Provision system and subsystem

CPROV-XML: XML syntax for Cloud provisioning.

CPU: Computer Processing Unit

RAM: Random Access Memory

VPN: virtual private network

VPC: virtual private cloud

VM: Virtual Machine.

CPS: Cloud Provisioning System.

NODE: Any icon from the palette

GUI: Graphical User Interface

## 1.5 Overview of document

The rest of the document contains more detailed information about the various architectural and design decisions made throughout development process of the CProv systems. The document is laid out in three more chapters and an appendix chapter.

Chapter 2 covers our proposed system software architecture. We begin by presenting the major subsystems and overviewing each of the subsystems. In addition, we present our architectural pattern choices and discuss our reasoning process for selecting such patterns. An UML profile for our architecture is presented, along with a description of our generative architecture. Finally, we conclude this chapter with details about subsystem decomposition.

Chapter 3 covers our object design. We begin by presenting a minimal class diagram for the subsystems to be implemented. Furthermore, we present refined sequence diagrams from the analysis model and a state machine for the main control object in each of the major subsystems. We conclude the chapter by presenting a detailed class design, including the meta-model used for the CProv systems, and OCL constraints for the static model.

Chapter 4 provides a glossary of terms used in this document.

Chapter 5 provides various appendices to further detail our design choices and log the various meetings and tasks assigned to each team member throughout the course of developing this Design Model.

# 2         Proposed System Software Architecture

This chapter covers our proposed system software architecture. We begin by presenting the major subsystems and overviewing each of the subsystems. In addition, we present our architectural pattern choices and discuss our reasoning process for selecting such patterns. An UML profile for our architecture is presented, along with a description of our generative architecture. Finally, we conclude this chapter with details about subsystem decomposition.

## 2.1       Overview

In this section we will provide a package diagram showing the major subsystems for our CProv design.