CEN 5064 Software Design

Cloud Provisioning System(CPS)
Group #1

Design Document

Dionny Santiago
Edwin Malek
Xiaoyu Dong
March 31, 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 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.

In this deliverable, The focus switches on the design aspect of the software and/or component of the application. This is achieved through a series of UML diagram. A package diagram is used to describe the system in its entirety, and divide each subsystem visually. Followed by describing each subsystem with a component diagrams. Following, class diagrams were created , giving more details about the system without diving too deep into the attributes and methods. Another version of the class diagram is also produced, but this one includes the methods and attributes. And finally we refined the previously created the sequence diagrams for the same use cases that were chosen during the analysis document.

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

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: Pipe and Filter, and Model View Controller. 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.

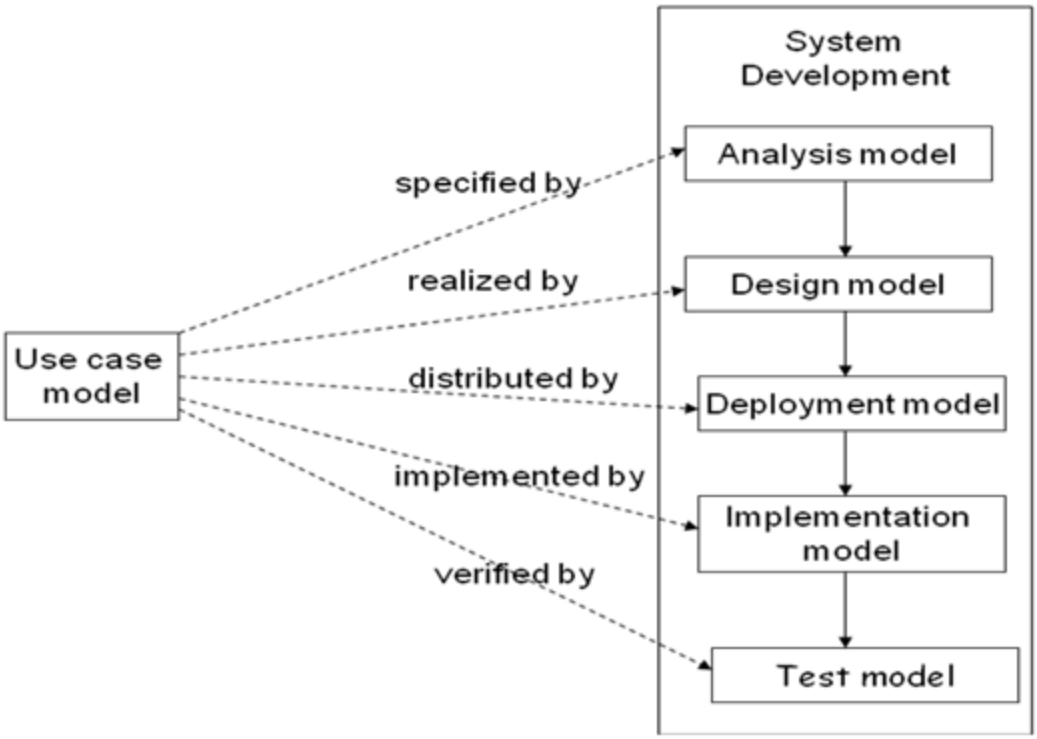


Figure 1-1 The Unified Software Development Process

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 executable 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. See Figure 1-2 for an illustration on MDSD.

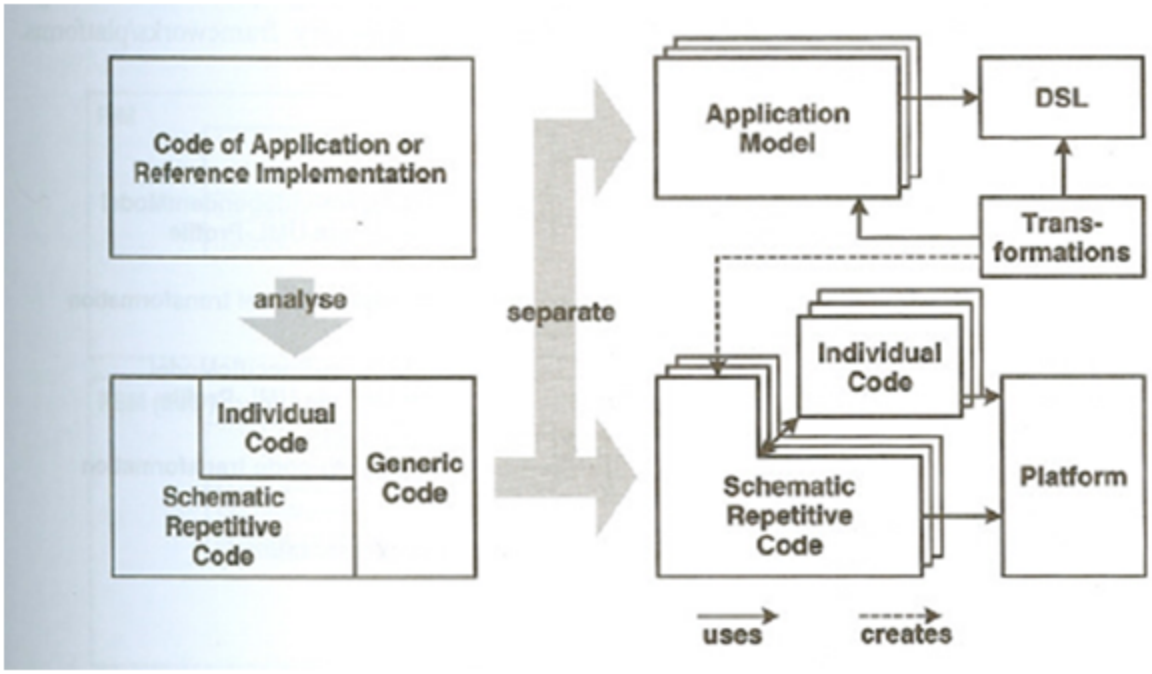


Figure 1-2 The Model Driven Software Development Process

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. The CProv application major features are to create a Cloud Provisioning Service model and be able to use this model and transform the model into CPS calls for a specific cloud service provider (Amazon EC2, Google CPS). Similar to some compilers, our EMF/GMF Model will be taken through a filter to be transformed into X-CProvML, and then From X-CProvML another filter will be applied to be transformed into CPS specific Calls. Each transformation intermediate steps will be written to a flat file, and saved on the drive. Thus, the primary architecture that will be used is the Pipe-and-Filter, where each filter package will contain and realize a transformation step. In our case, the application will only need 3 Filters before the result is sent to a sink. Within the first filter package, a secondary architecture will be applied: Model-View-Controller. The model in our case will be represented as a series of flat files which

are in themselves steps of the transformation. Each step does not necessarily need to be filtered twice. The user may only want/need to go through one transformation. The user created model(diagram) in the Modeling Environment part of the application will be saved as well as two XML files: one representing the layout of the diagram, and the other representing a tree representation of the diagram. The package diagram illustrated the above mentioned packages.

The Architectures were chosen according to the needs of the application. The Pipe and Filter architecture is adequate for software component(s) capable of transforming or changing the content of a file. In our case, the application will have three different transformation (G-CProvML to X-CProvML to CPS Calls). The Model-View-Controller architecture was chosen here to facilitate the changes to between the multiple views available to the user. The separation of the native model (diagram) and the code for the displaying of the appropriate information is separated.

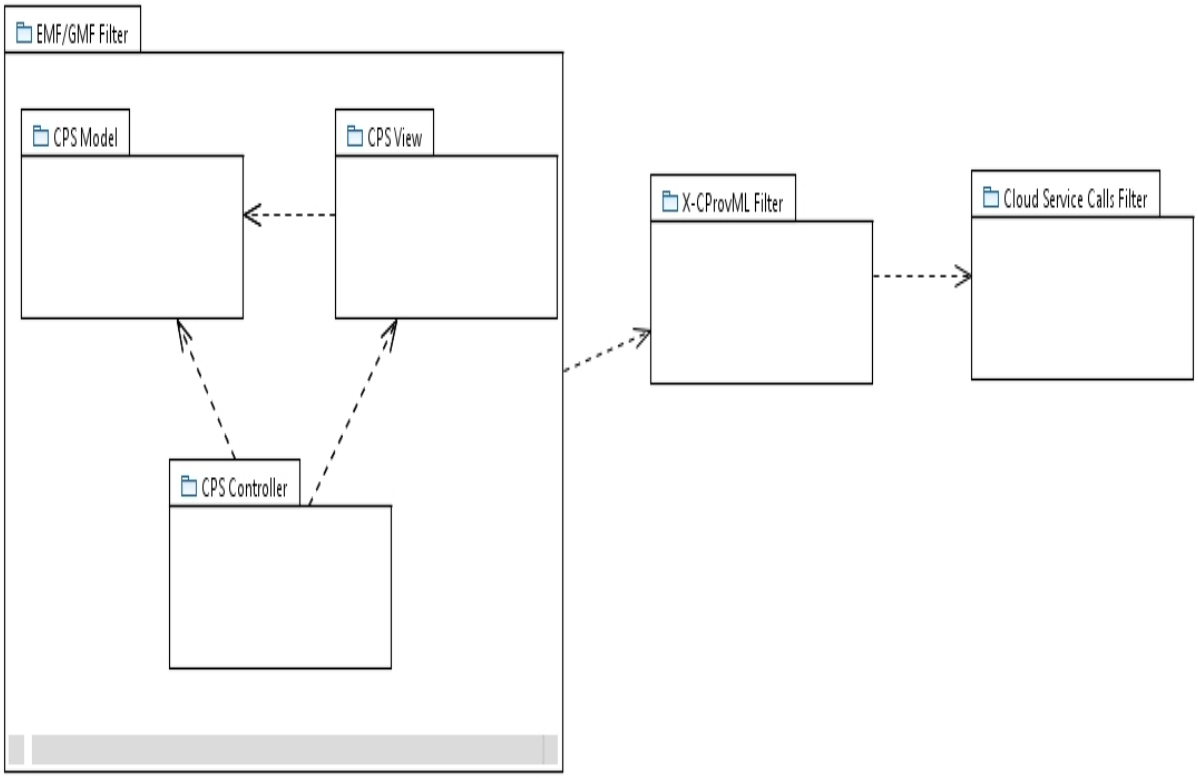


Figure 2-1 Architecture of the System

2.2 UML Profile for the Architecture

Our first UML profile is GCProvFilter. This profile defines a class extension stereotype named CP_MVC which organizes all of the Model-View-Controller stereotypes. Included, we have a CP_Model stereotype which defines a Name with an OCL constraint stating that the name for the model must be a nonempty string with a maximum length of 29. We also have a CP_View stereotype with a tagged value for the Type of the view. The GVE supports several different view types: a Canvas, an Outline, and a Tree. Each view type is defined within an enumeration named ViewType. Every view that uses the CP_View stereotype must have one of the supported view types. Finally, we have a CP_Controller stereotype to be used by all controller classes.

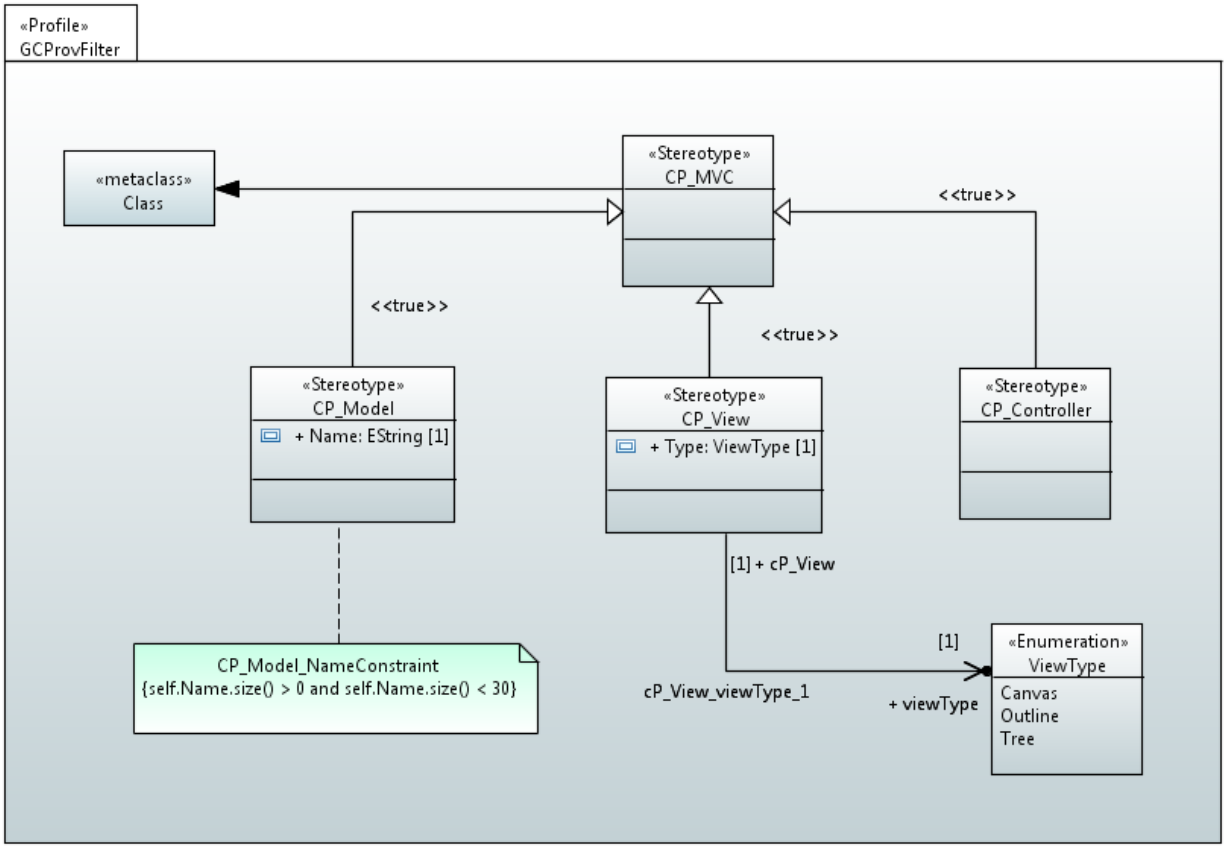


Figure 2-2 UML Profile GCProvFilter

Our second UML profile is PipeAndFilter. This profile defines a class extension stereotype named CP_PipeAndFilter which organizes all of the stereotypes relevant to the pipe and filter architecture. Included, we have several stereotypes: CP_Source which represents the source of a pipe and filter pipeline, CP_Pipe which represents a pipe, CP_Filter which represents a filter, CP_Output which represents the output of a filter, and CP_Sink which represents the final output of the pipeline. For our specific profile, each CP_Output has a filename and a file type. The file type is represented by an enumeration named FileType containing the values XML and Text. We

also define OCL constraints which enforce that the filename of a CP_Output must be nonempty and no greater than size 49.

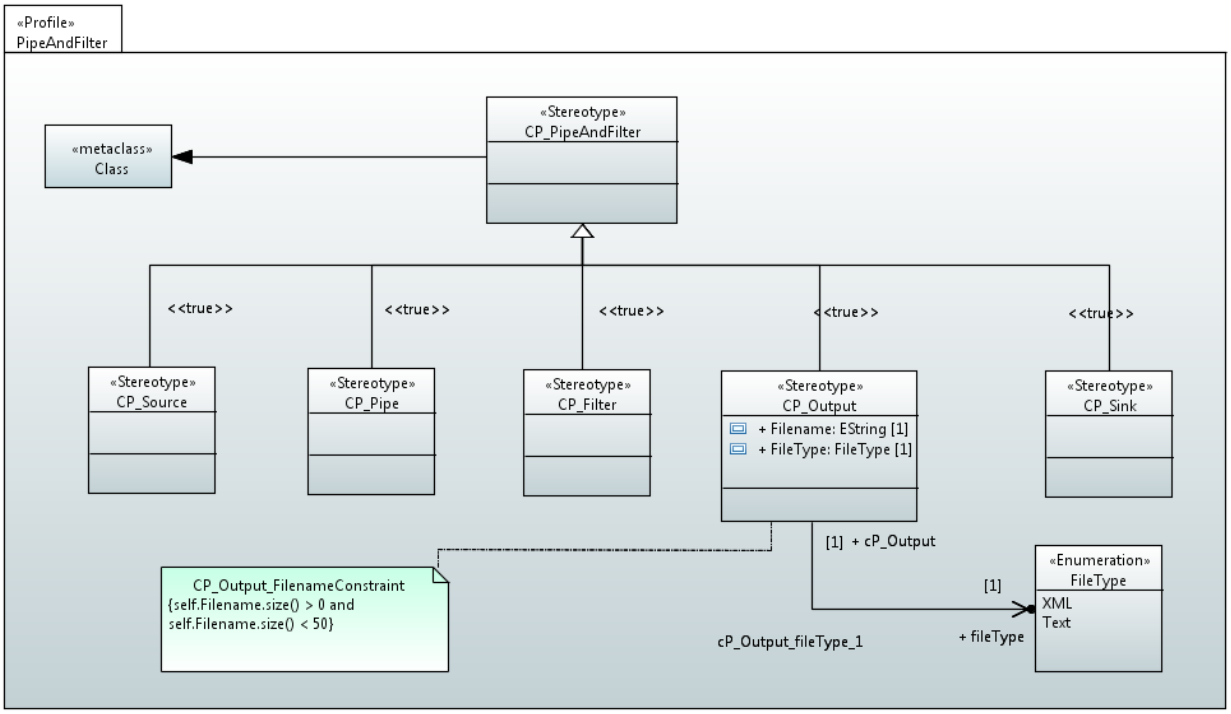


Figure 2-3 UML Profile PipeAndFilter

2.3 Generative Architecture

As shown in the figure below, this section focuses on the various architectural concepts and the various generator outputs to represent each concept. Finally, each generated concept is tied to a specific platform.

The first architectural concept is the Presentation layer. Using our generative architecture, this is all generated for us into a Graphical Model that uses both the Java Platform and the Eclipse EMF/GMF Platform. The idea is that while a major subset of our graphical model is generated, there are pieces we would like to customize, and for that we will rely on the standard Java Platform.

Next we have the CPS Metamodel which is represented by an Ecore Model which generates code using the Eclipse EMF/GMF Platform.

The CProv Model is represented by a generated graphical model (G-CProvML) as well as a layout model (L-CProvML), and is realized by the Eclipse EMF/GMF platform as well.

The execution model is represented by the X-CProvML model and is realized by several cloud provisioning libraries for Amazon and Google.

Finally, the execution engine is realized by the Cloud API Caller which uses the cloud provisioning platforms previously mentioned.

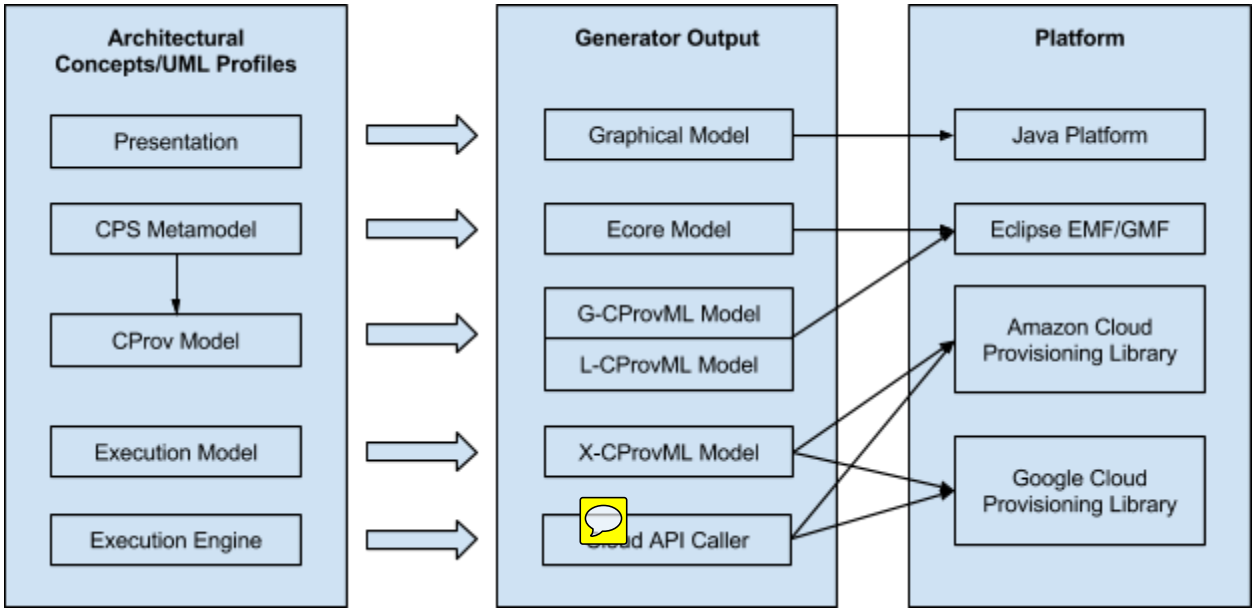


Figure 2-4 Generative Architecture Overview

2.4 SubSystem Decomposition

First we will start by analyzing the three major overarching subsystems:

EMF/GMF Filter Subsystem: Represents the GVE including all user interactions with the GVE, and transformation from the CProv model in memory on the GVE to respective EMF/GMF XML files. Consider the following sub-packages within this subsystem:

> Model Subsystem: Represents objects in memory representing the model being constructed by the user of the CPS system.
>
> View Subsystem: In charge of updating the views which the user will be seeing according to the changes made in the Model subsystem. As well as displaying appropriate icons, and pallette elements (GMF generated)
>
> Controller Subsystem: Responsible for coordinating the interactions between the views and the model. Directly responsible for processing all user actions.

X-CProv-XML Filter Package: Responsible for the transformation from the G-CProv-ML to the X-CProv-ML XML files.

CPS service Calls Filter Package: This subsystem is responsible for creating the appropriate calls depending on the Cloud Provisioning Service provider that was picked by the user. We call the output of this transformation the CPS calls.

The following component diagram represent parts of the subsystem dependent another: CPSDiagram component depends on the L-Cprov-XML component (layout) and the G-CProv-XML (tree representation), and each communicate through their respective interfaces.

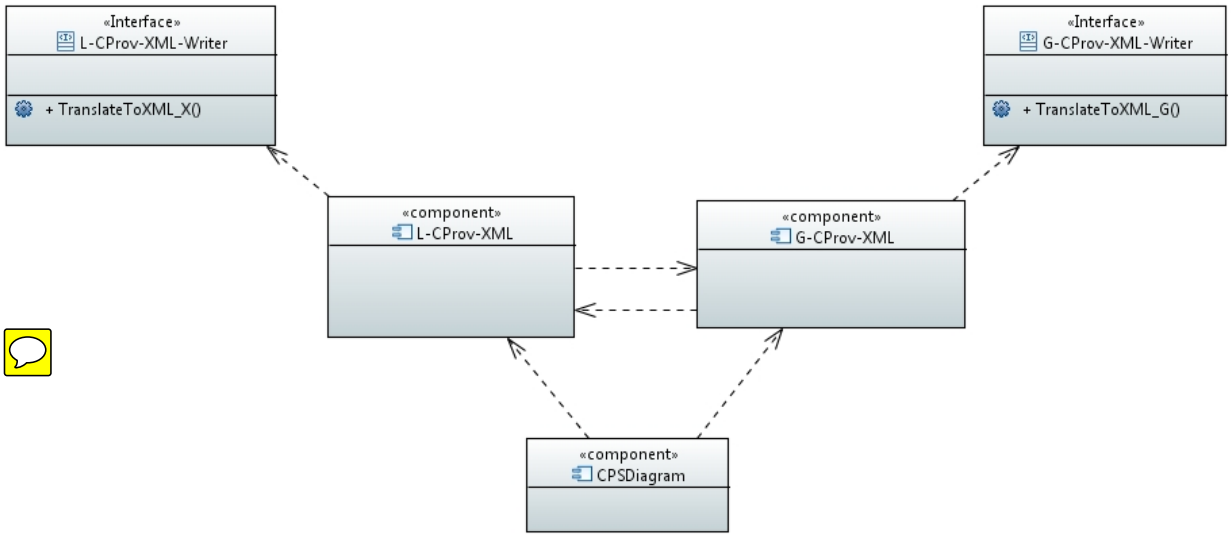


Figure 2-5 Generative Architecture Overview

The following component diagram represent the a simple filter called here (Translator), which gets the source G-CProv-XML and sends to digest of the filter to X-CProv-XML-out interface.

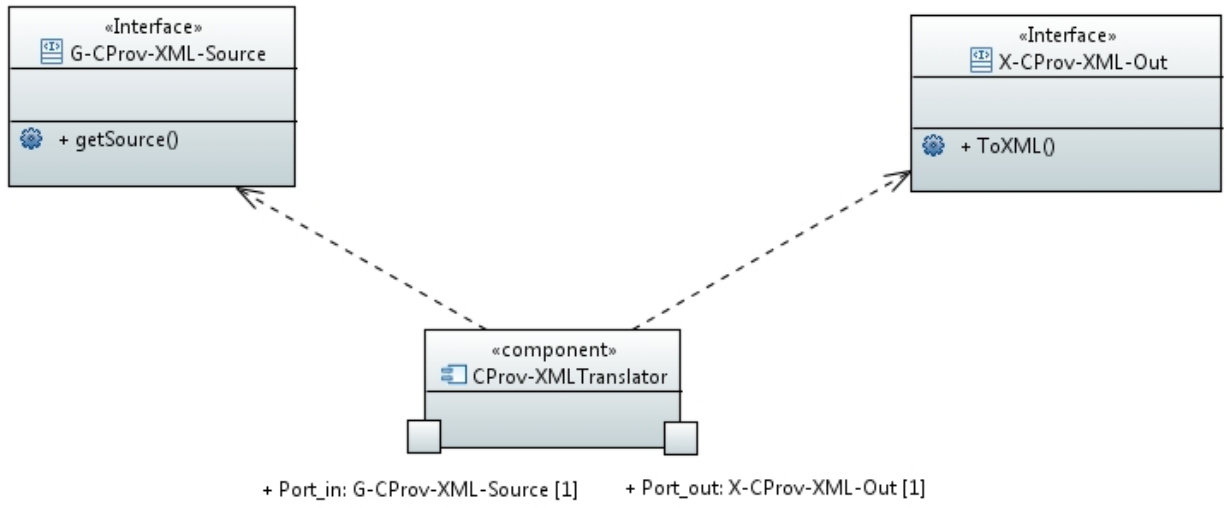


Figure 2-6 Generative Architecture Overview

The following component diagram represents a filter called here translator, which gets the XML file from the X-CProv-XML interface and invokes the implemented method TranslateToCPSCalls through the CPS_Call_out interface.

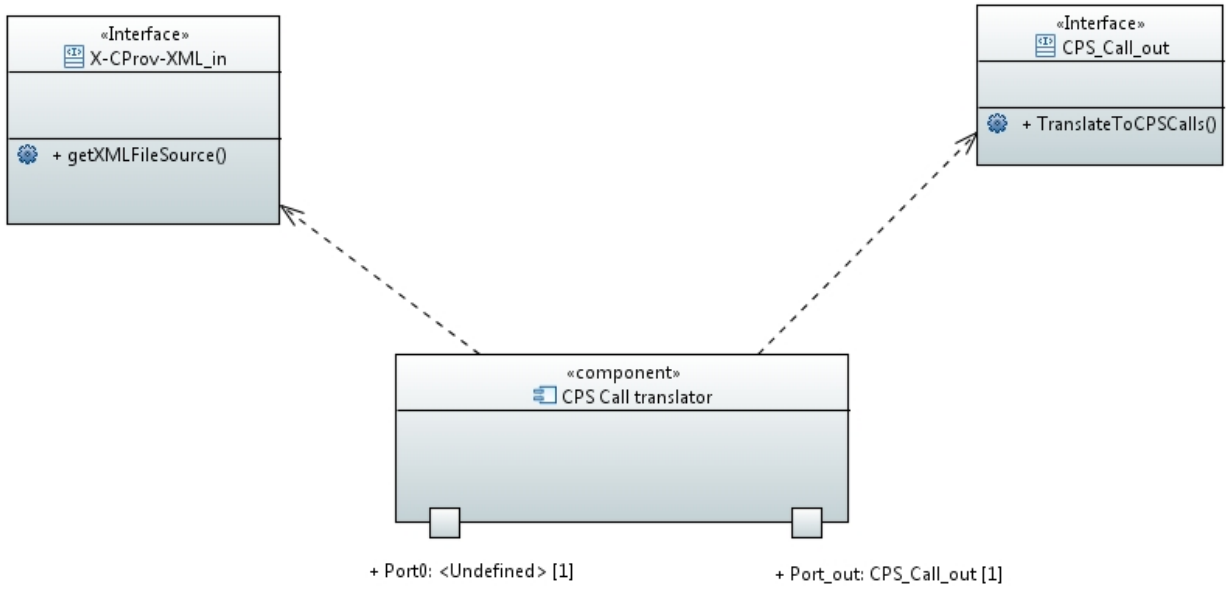


Figure 2-7 Generative Architecture Overview

3 Object Design

This chapter will start with the minimal Class Diagram, showing the use of the Profiles described in the above section, along with the design patterns used throughout the Class Diagram, followed by the refined Sequence Diagrams from the Analysis Model, and the State machine diagrams for the main control object in the system/subsystems. This will be followed by the Detailed Class Diagram involving all the rightful Methods, Attributes, and Constraints. Finally a detailed explanation of the Constraints chosen for the main controller in the main subsystem will follow.

3.1 Overview

Starting from the bottom part of the class diagram, The profile is used in the modeling classes represented here as children of the abstract class ModelElement. The model will be represented through 5 classes, each representing a type of icon to add to the canvas. The classes ModelElement, Storage, Network, SecurityGroup, Instance, and Environment belong to a class called Model which also serves as the source data for all the transformations to occur. It is from the generated XML file(s) of the Model that Filters will change. The Model is depended on by the ModelEditor class in charge of different commands and the views available to the user. The ModelEditor class uses the Singleton pattern to ensure only a single instance is used by the system. The ModelEditor communicates with the model via several different views using a ViewStrategy object, which uses the Strategy Pattern (left hand side) to render a view of the model in different ways. The ModelEditor also makes use of the Command Pattern (right hand side) to organize possible system interactions into commands that are undoable and redoable.

The ViewStrategy realizes the Strategy Pattern using the TreeView, CanvasView, and OutlineView all representing the 3 available views to the user, and the commands are realized with the classes CreateConnectionCommand, ExitCommand, TransformCommand, SaveCommand, AddNodeCommand, all implemented to provide the possibility for the user to come back to different object states through undos and redos. The Execution Environment is realized through the Filters classes: GCProvMLFilter, XCProvMLFilter, and CPSCallFilter. Each of the filter classes transform from the Model generated XML to another type of XML file, and outputs the result of the transformations to their respective physical manifestations through the use of annex classes (XMLWriter, TextWriter).

When generating the calls to a cloud provisioning system, another strategy pattern is used to create different CPS cloud calls according the particular syntax of the different CPS providers (AmazonEC2, Google Cloud). The following shows the Minimal Class Diagram explained above. All calls to the filter subsystems are piped through a Façade class using the Façade pattern called PipelineFaçade to drastically reduce the coupling between subsystems, in this case the different filter subsystems.

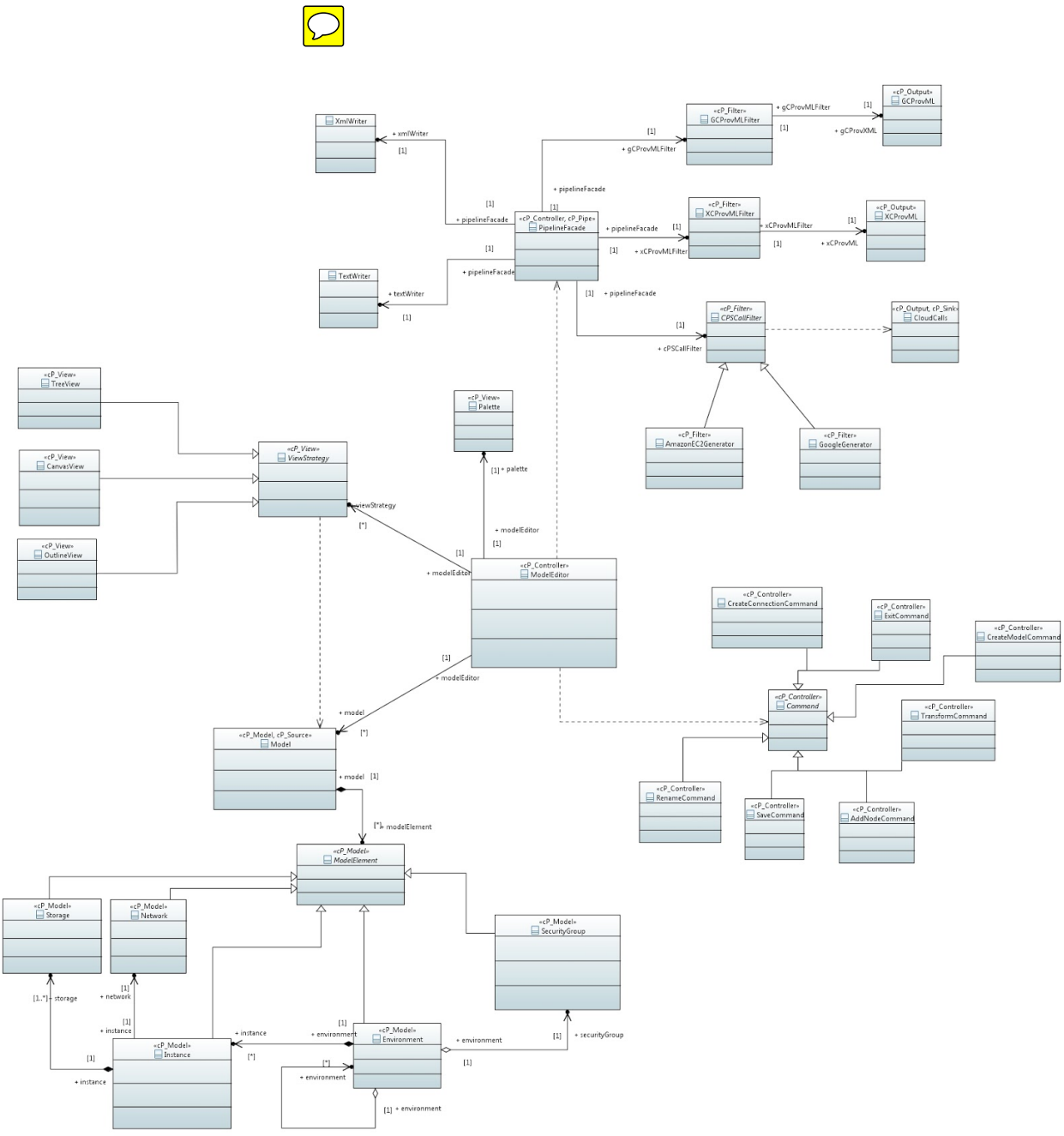


Figure 3-1 UML Minimal Class Diagram

We will break down the minimal class diagram in sections for a better understanding of each section:

Model and Source classes section (Model-View-Controller Subsystem):

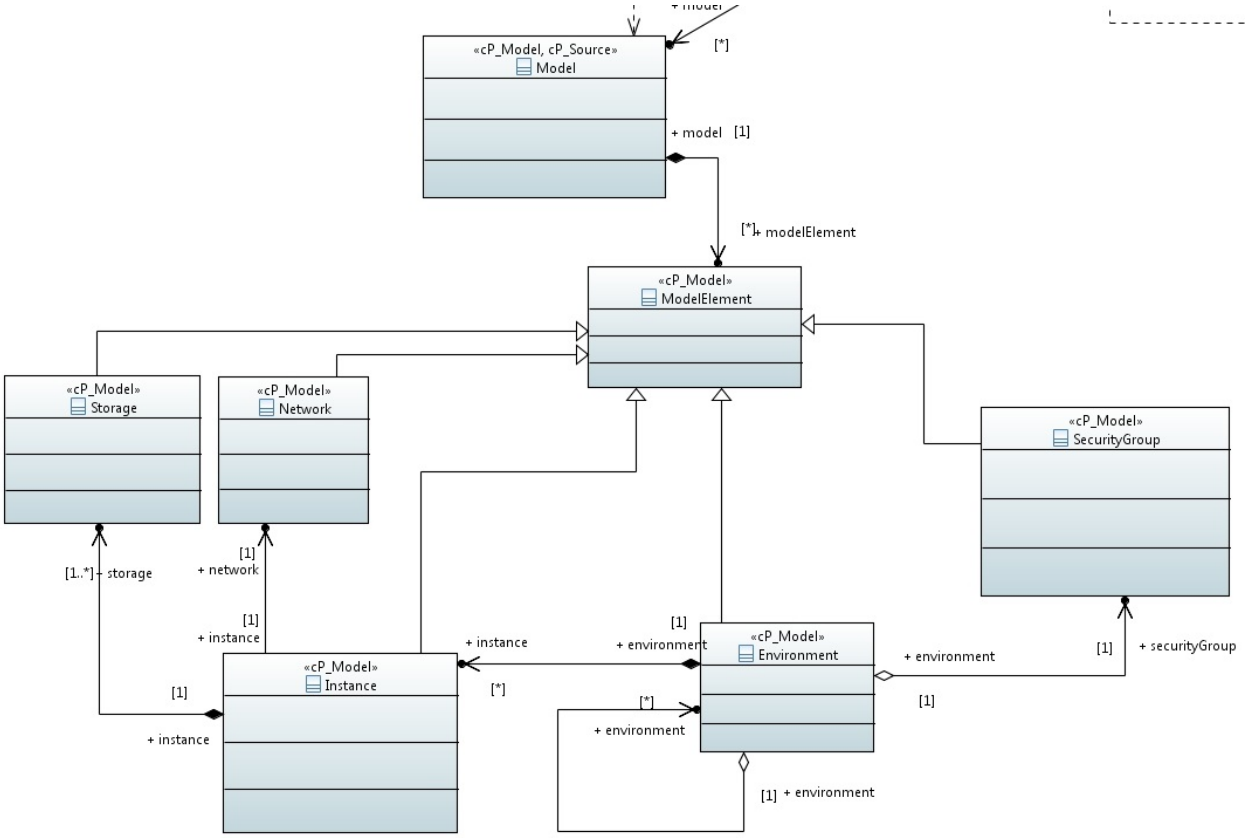


Figure 3-2 UML Minimal Class Diagram (Model)

View Classes sections (Model-View-Controller Subsystem): Strategy Pattern applied to output different views.

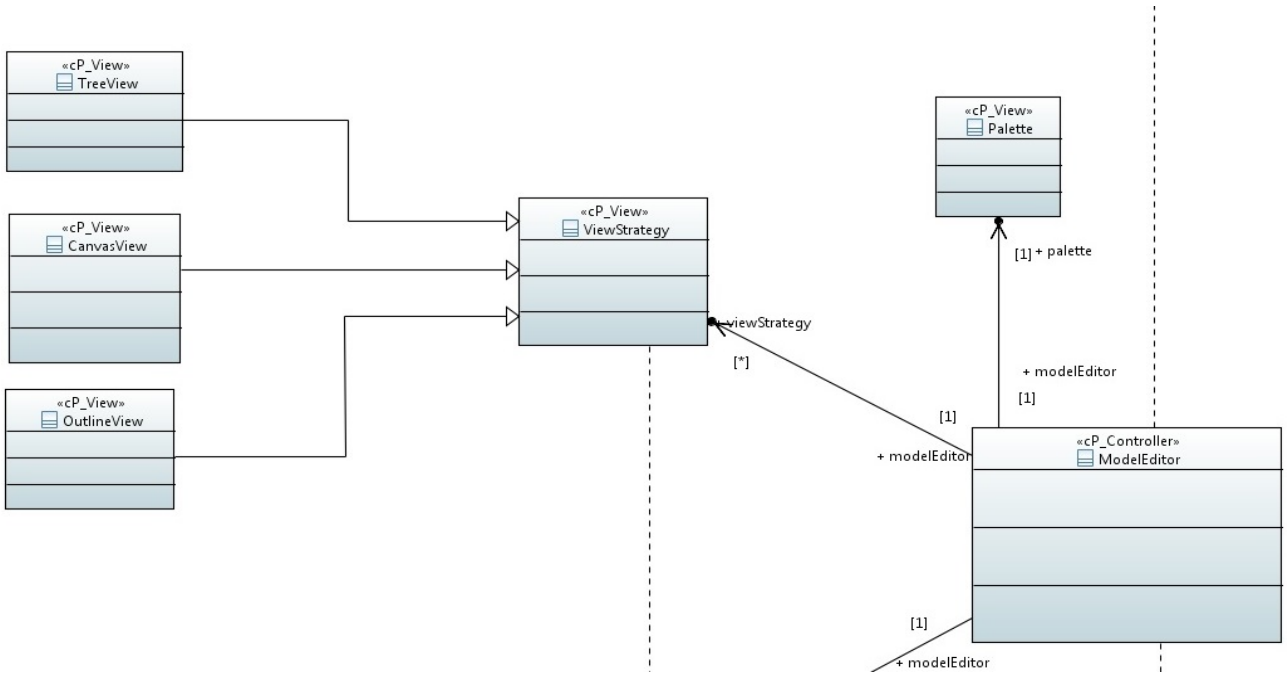


Figure 3-3 UML Minimal Class Diagram (View)

Controller classes section (Model-View-Controller Subsystem): Command pattern used here to achieve different logical commands for user. The singleton pattern is used here on the ModelEditor class.

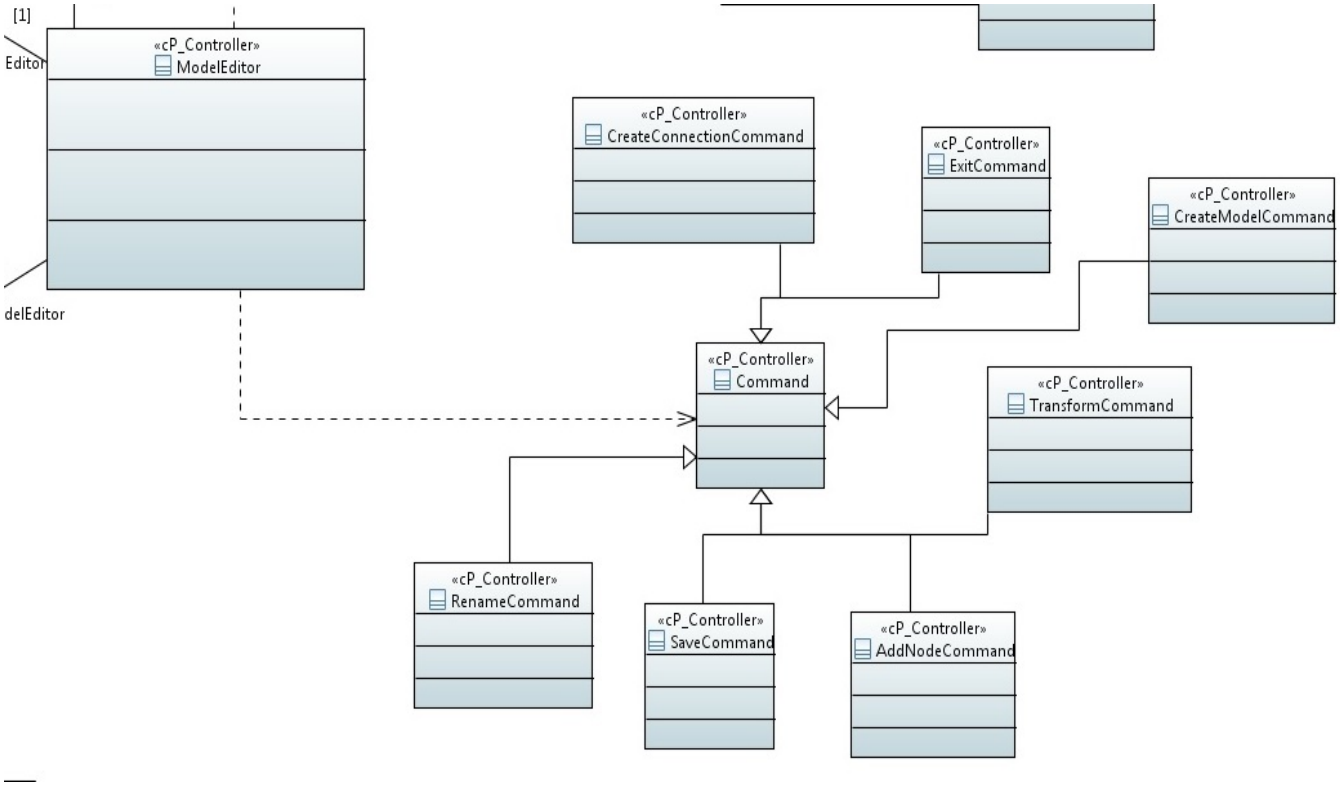


Figure 3-4 UML Minimal Class Diagram (Controller)

Filter, Output, and Sink classes section (Pipe And Filter Subsystem) : All used for transformations, a Facade pattern is used here with the PipelineFacade class.

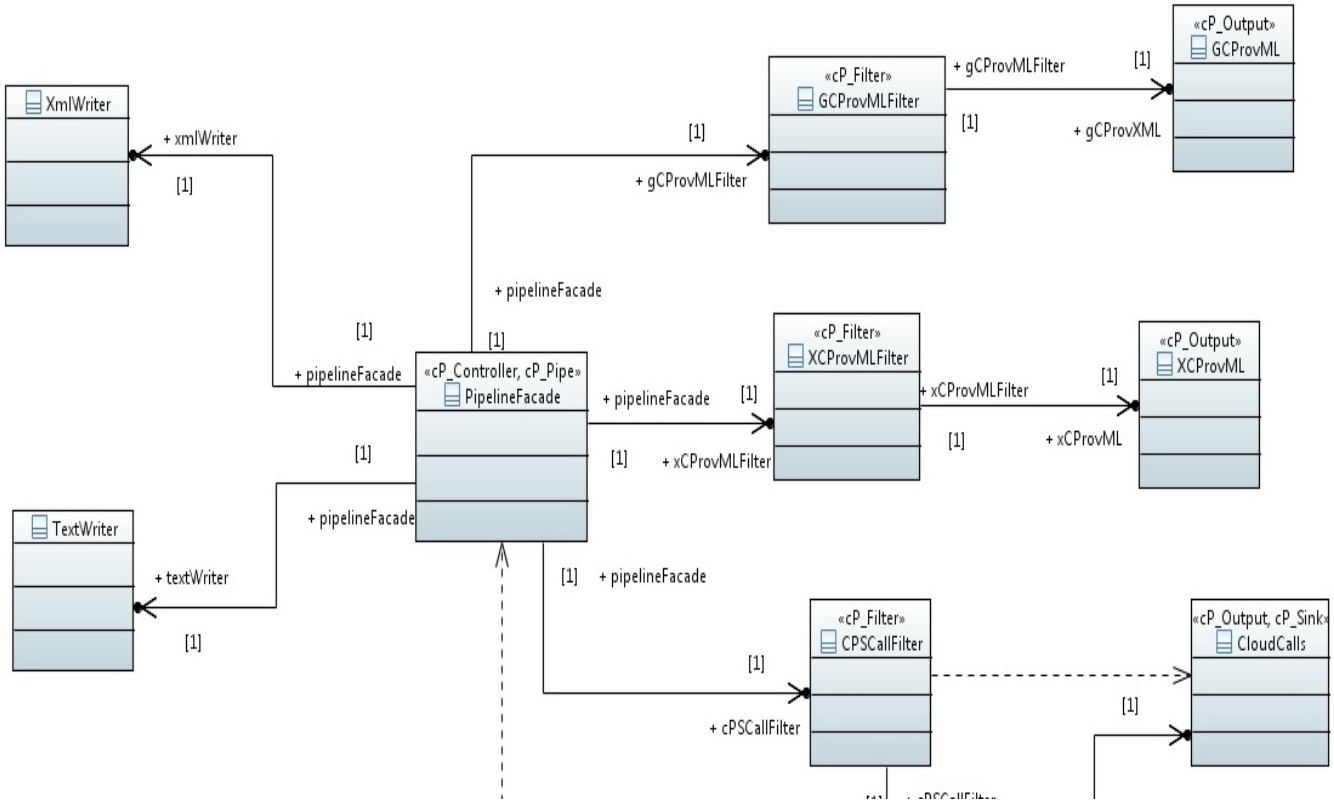


Figure 3-5 UML Minimal Class Diagram (Filter, Pipe)

Filter, output, and sink classes section (Pipe And Filter Subsystem): Another Strategy pattern is used here:

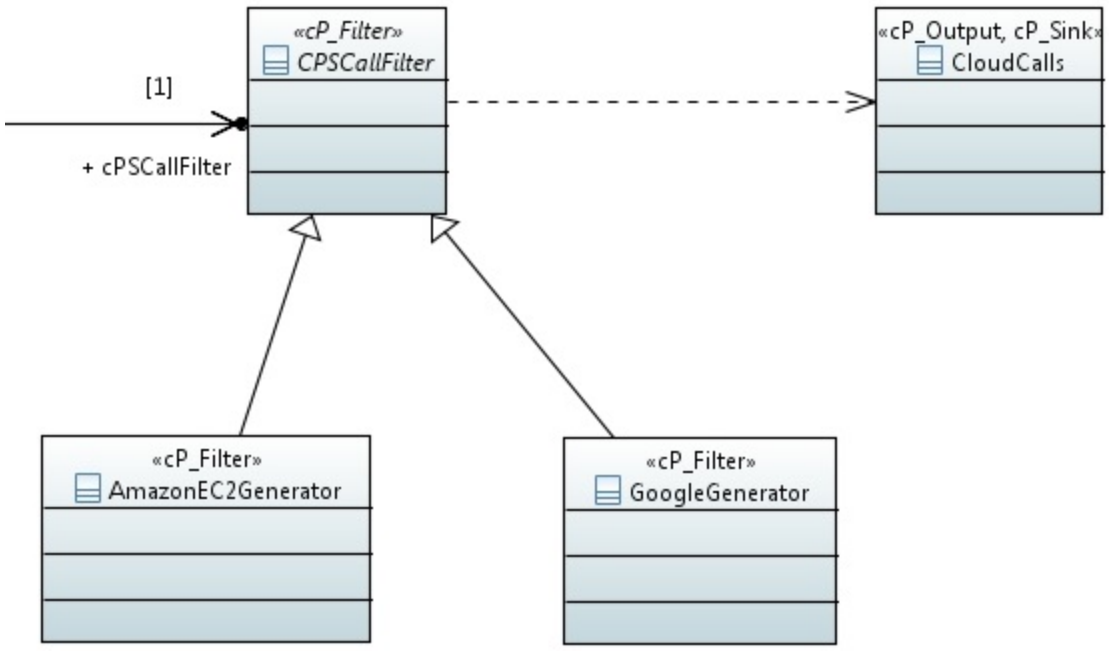


Figure 3-6 UML Minimal Class Diagram (Filter-Strategy Pattern)

3.2 Object Interaction

3.2.1 Sequence Diagrams

Use case CPROVME001 - CreateNewModel gives rise to the following sequence diagram:

The User (cloud administrator) unknowingly uses the main controller (ModelEditor singleton object) to start this sequence, the createNewModel method is invoked. A createModelCommand object is created (construct method) and then followed by the execute method in charge of creating the model and storing the state of creation in case of undo commands possibly at a later time. A Model object is then initialized and the palette object populates itself with all the element of the CPS. Lastly, the viewStrategy (generalized here since all the views should be rendered as well) object calls the render method on itself.

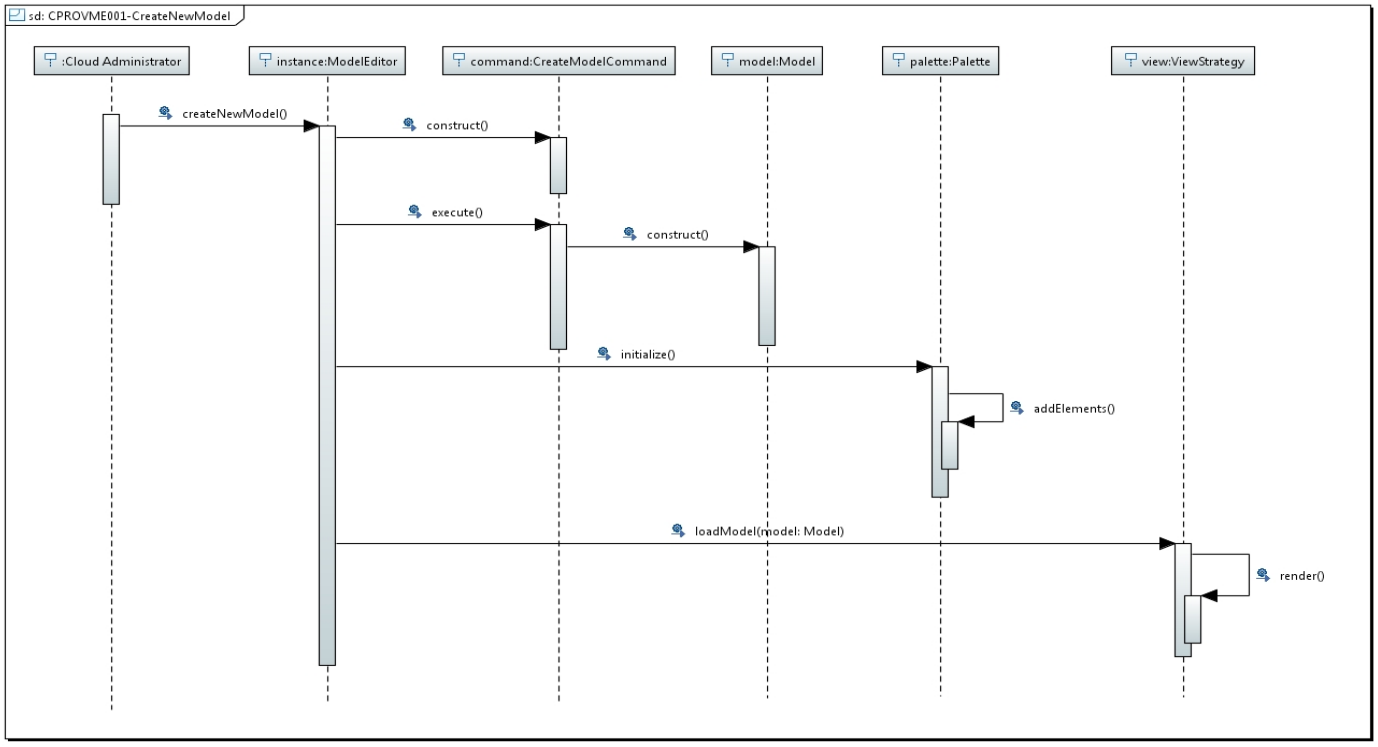


Figure 3-7 UML Sequence Diagram CreateNewModel

Use case CPROVME005 - AddInstanceNode gives rise to the following sequence diagram:

The cloud administrator selects an instance node on the palette, which internally invokes the selectElement method of the main controller (ModelEditor singleton object), which passes the reference of the node clicked to the palette object and passes the argument to the selectModelElement method of the palette object. The next action involves the user to click on the canvas in the desired location on the canvas. The main controller method addElement is invoked which must construct a command object (createNodeCommand in this case) in order to support any possibility of undos at a later time. The initialized command object executes which in turn creates an Instance object, then adds to the model object and finally the view updates itself with the previously mentioned render method. The system then requires our user to input a name, the main controller starts the sequence, and since this action can also be undone, a corresponding command object has to be created (RenameCommand object), the name property of the instance object is then set and the view is once again rendered.

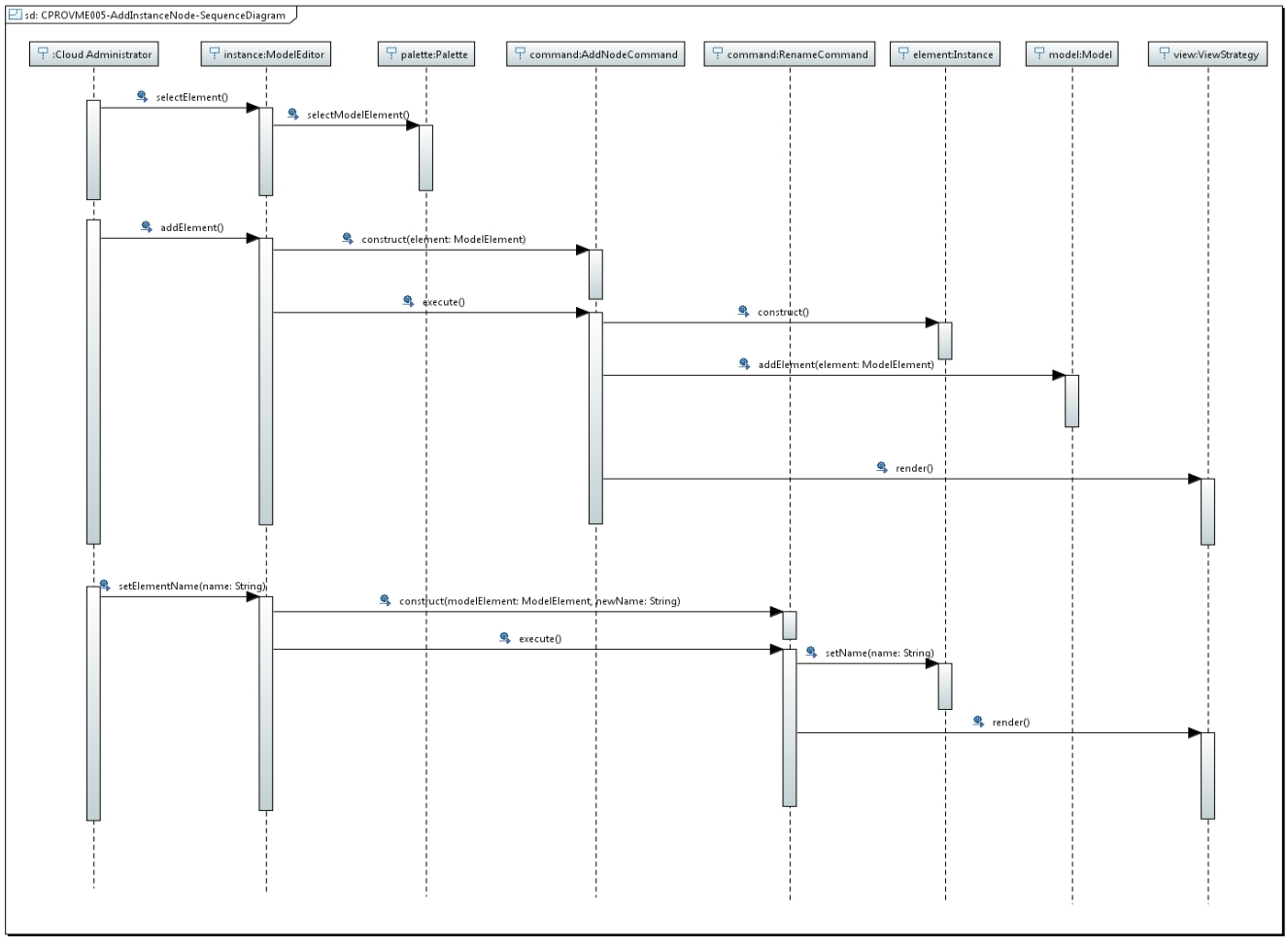


Figure 3-8 UML Sequence Diagram AddInstanceNode

Use case CPROVME014 - AddConnection gives rise to the following sequence diagram:

The cloud administrator is required to select a connection type in the palette, and select source node and then the destination node. In this case the user has already clicked on the connection icon in the palette. The main controller sets the source element with the connection (here setSourceElement method), then the controller sets the destination (here setDestinationElement method). Since this user defined command can be undone at a later time, a command object is created (in this case a CreateConnectionCommand) which is initialized by passing the source and destination elements to be executed with their respective methods. The model is then updated by calling the addConnection method and the canvas view is rendered.

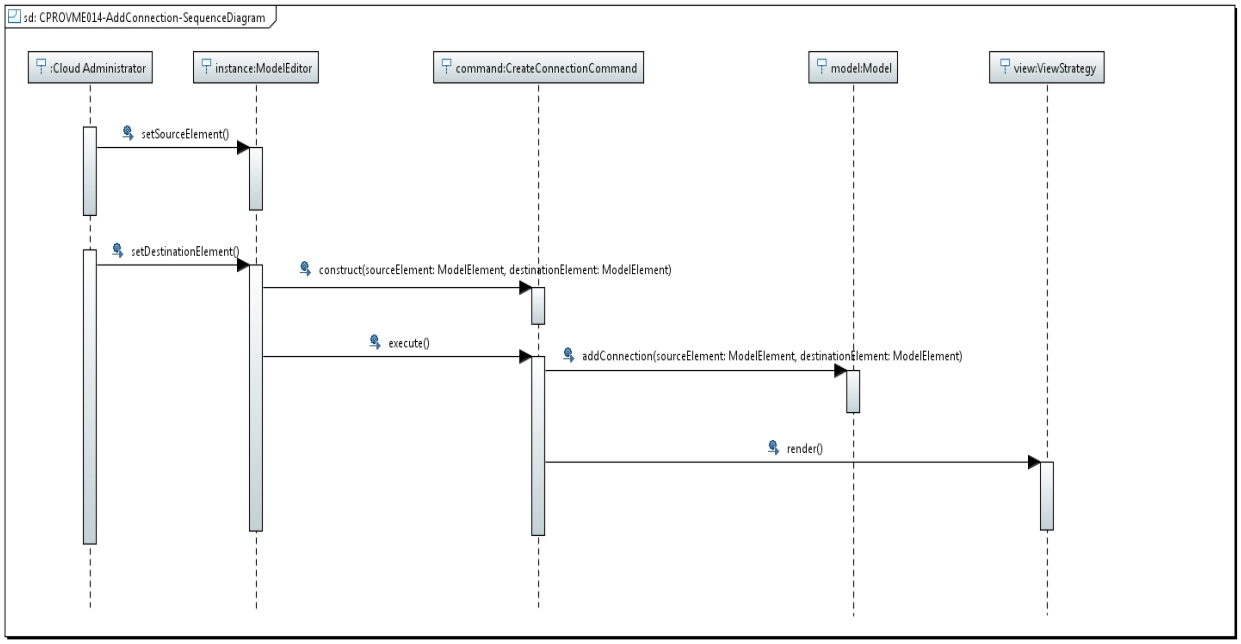


Figure 3-9 UML Sequence Diagram AddConnection

Use case CPROVEE003 - ValidateAgainstCloudProvisioningSchema - gives rise to the following sequence diagram:

The cloud administrator has the option to validate a model at any time. The validation process starts with a click input from the user. The main controller (ModelEditor) invokes a validate method which in turn calls the same method of the Model class which actually does the validation process. once the validation is done, the view is then rendered to reflect validation confirmation or error generation.

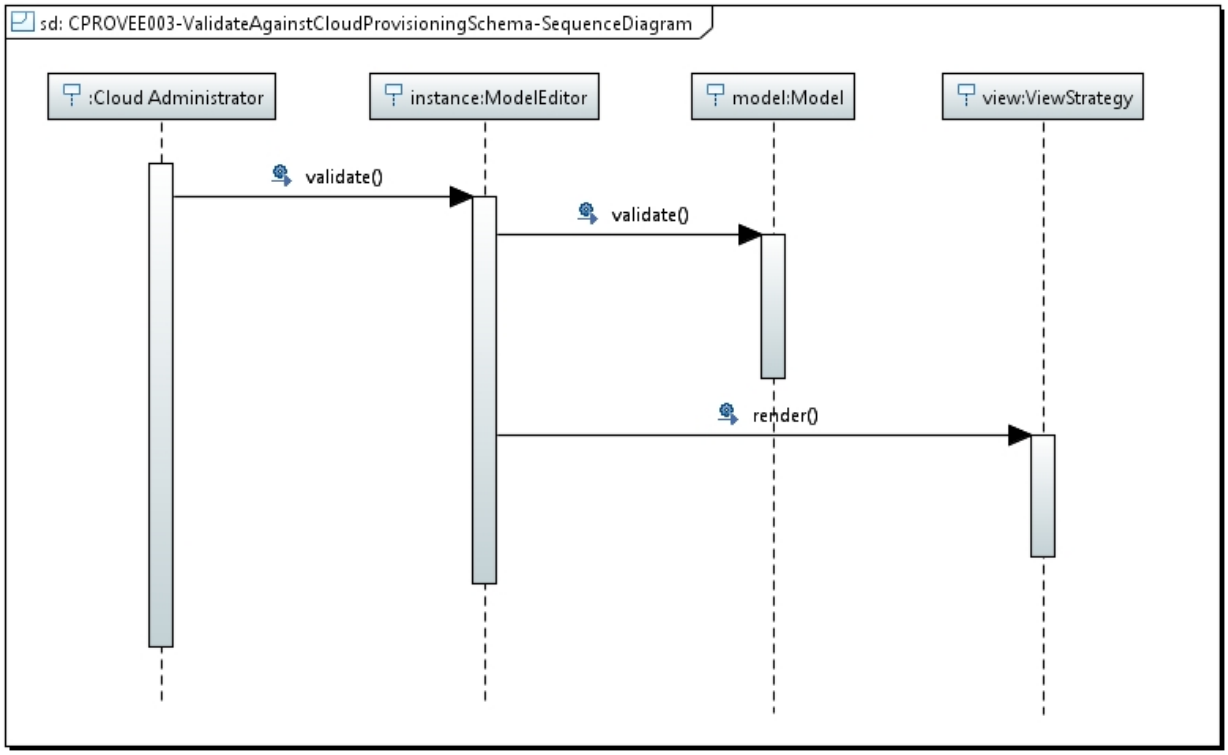


Figure 3-10 UML Sequence Diagram ValidateAgainstCloudProvisioningSchema

Use case CPROVEE003 - TransformToCProvXML - gives rise to the following sequence diagram: due to the limited size of the page, we will break down the following sequence diagrams in three sections:

The cloud administrator wishes to transform from Model representation straight to the X-CProvML, in this case the main controller (ModelEditor Class) passes the file to the pipeline facade class, which has a method that goes through both transformations in one call. The process takes place by first transforming to G-CProvML by feeding it the model, then creates a G-CProvML object, which is then fed through the X-CProvML filter through a method call outputting to a X-CProvML object, An XML representation in memory is then fetched to be returned to the pipelineFacade class to be finally turned into a persistent file (in our case by calling the persistXMLfile method on the XMLWriter class).

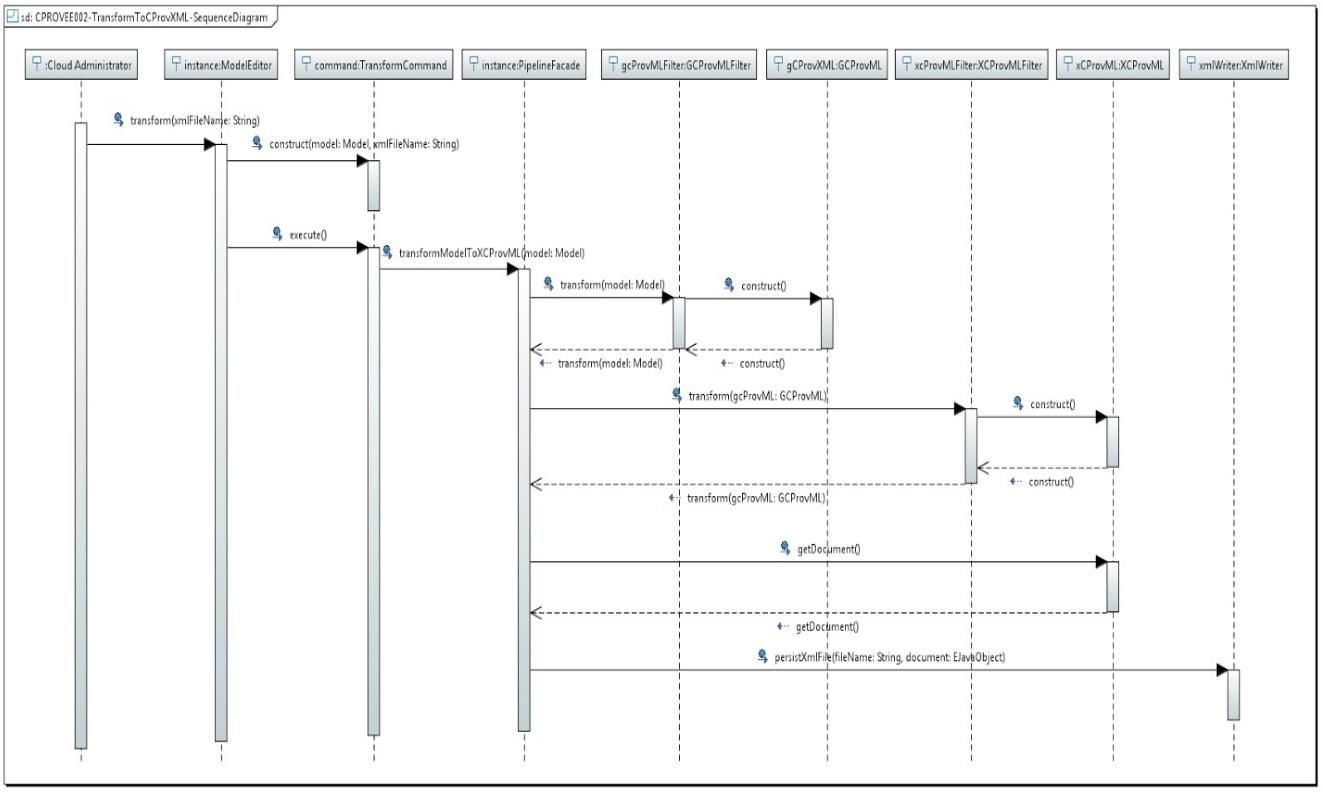


Figure 3-11 UML Sequence Diagram TransformToCProvXML

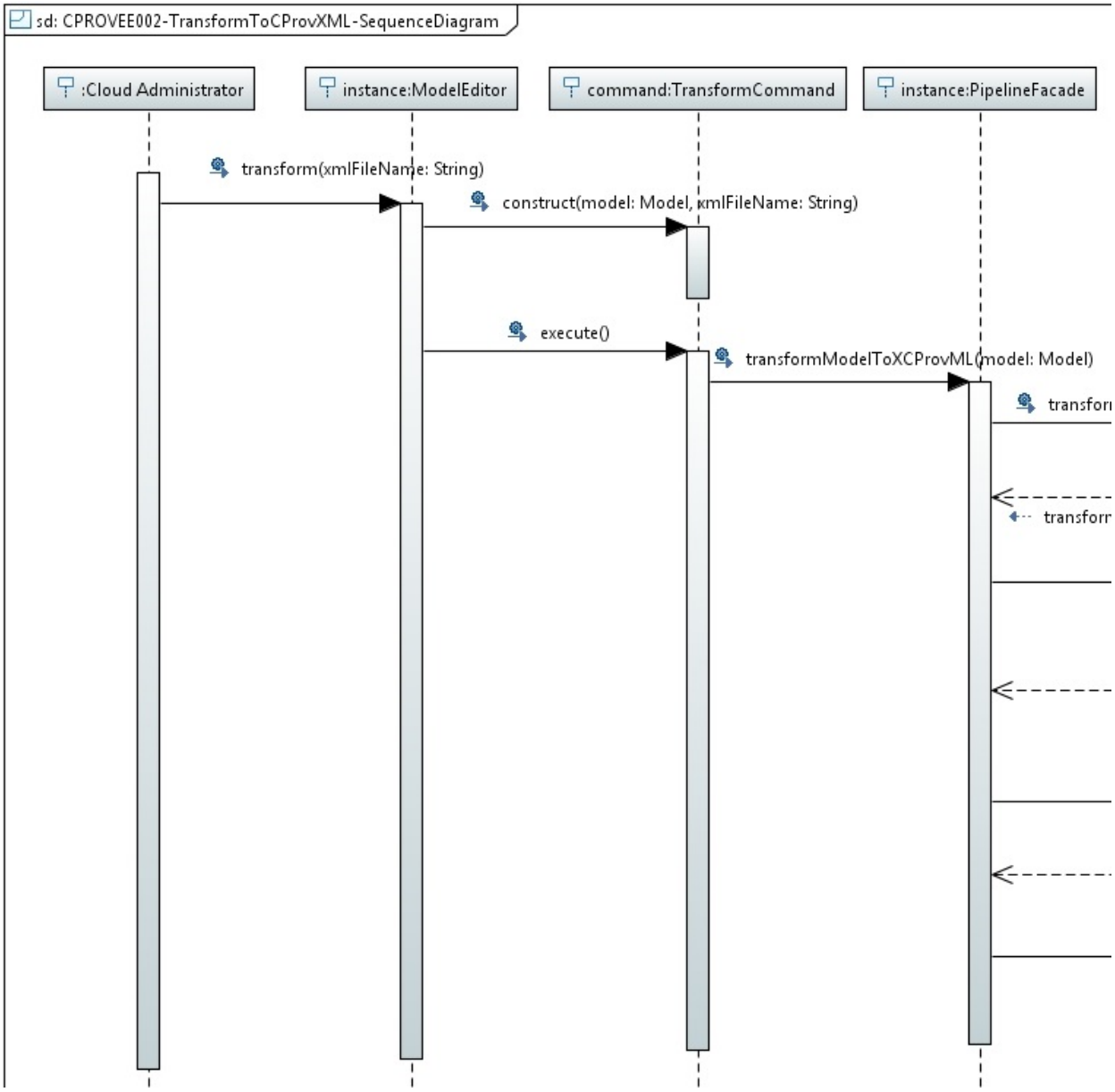


Figure 3-12 Zoomed in UML Sequence Diagram TransformToCProvXML (Left Side)

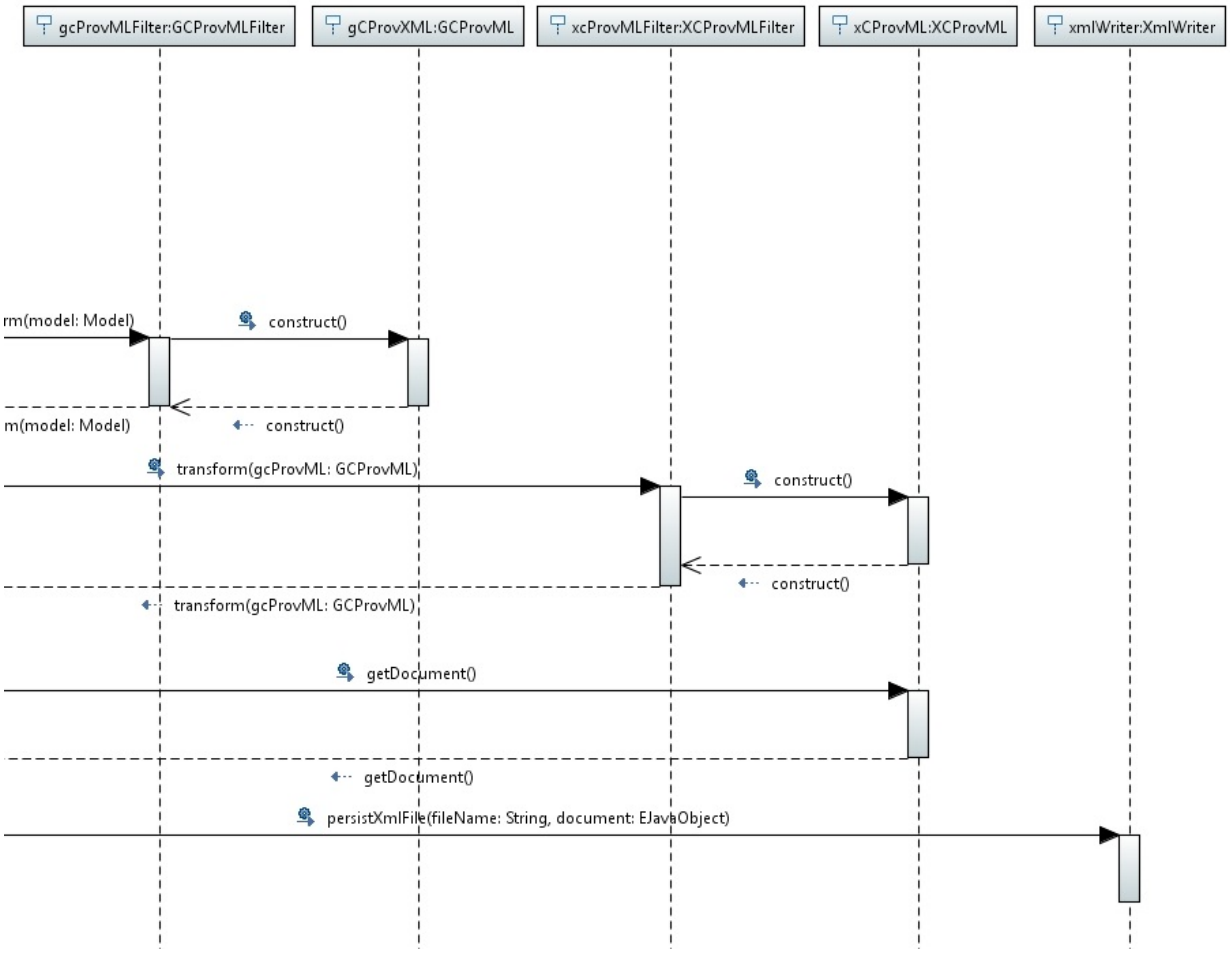


Figure 3-13 Zoomed in UML Sequence Diagram TransformToCProvXML (Right Side)

Use case CPROVEE001 - SubmitRequestForInstanceDeployment- gives rise to the following sequence diagram:

The cloud administrator wishes to transform a previously loaded or created model to Cloud calls. In order to accomplish that, the subsystem will have to create invoke a transform method from the main controller (ModelEditor Class) and delegate the transformation down the line by first creating a command object to support the possibility of undo at a later point in time. Once the command object is created, the model is fed through the pipelineFacade object to begin its first concrete transformation, and continues through a series of transformation (Model to G-CProvML to X-CProvML to CPSCalls). Each transformation is realized through a GCProv filter, then the XCProvML Filter and Finally CPSCalls filter, after each filter acts on its original input, an object with the content of the digest is created respectively. Finally through the use of an annex class (TextWriter), the output object of the CPSCallFilter class is written and saved to a flat file.

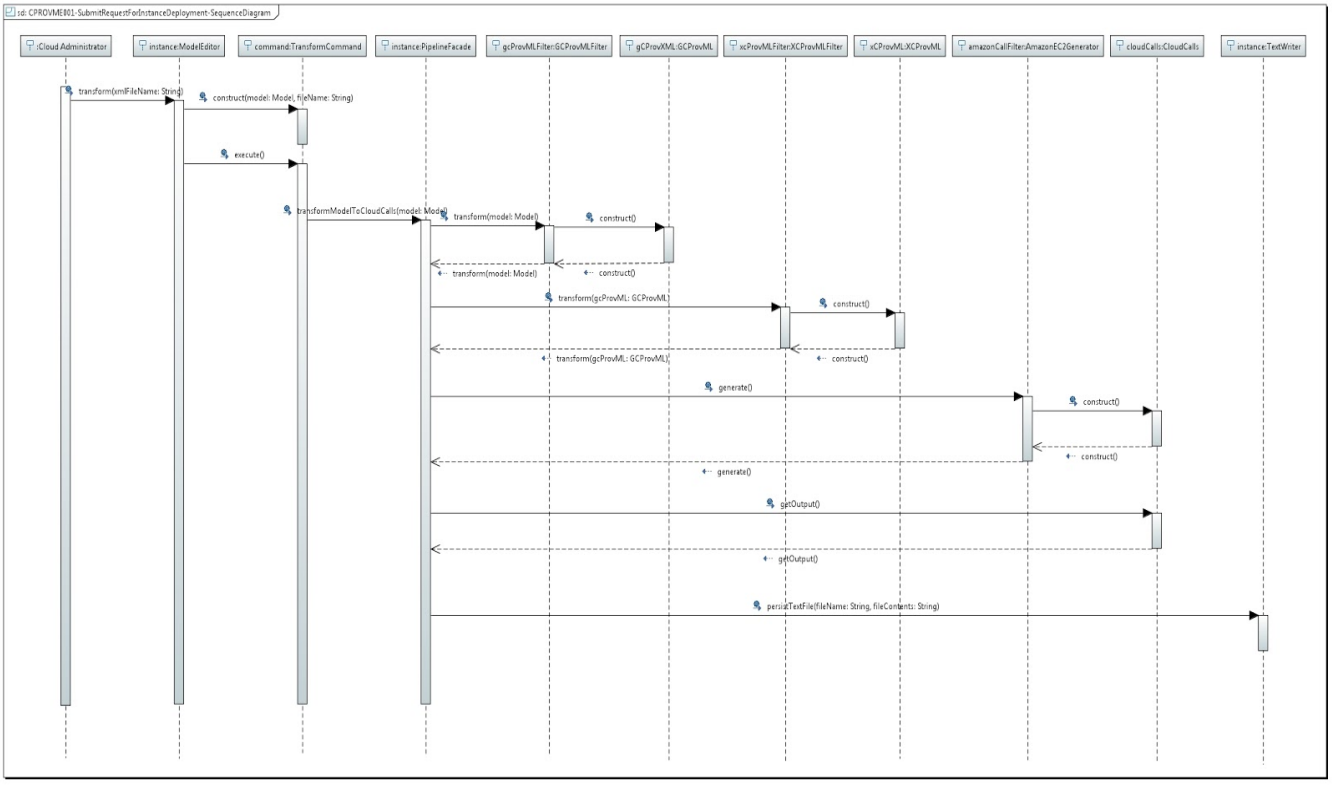


Figure 3-14 UML Sequence Diagram SubmitRequestForInstanceDeployment

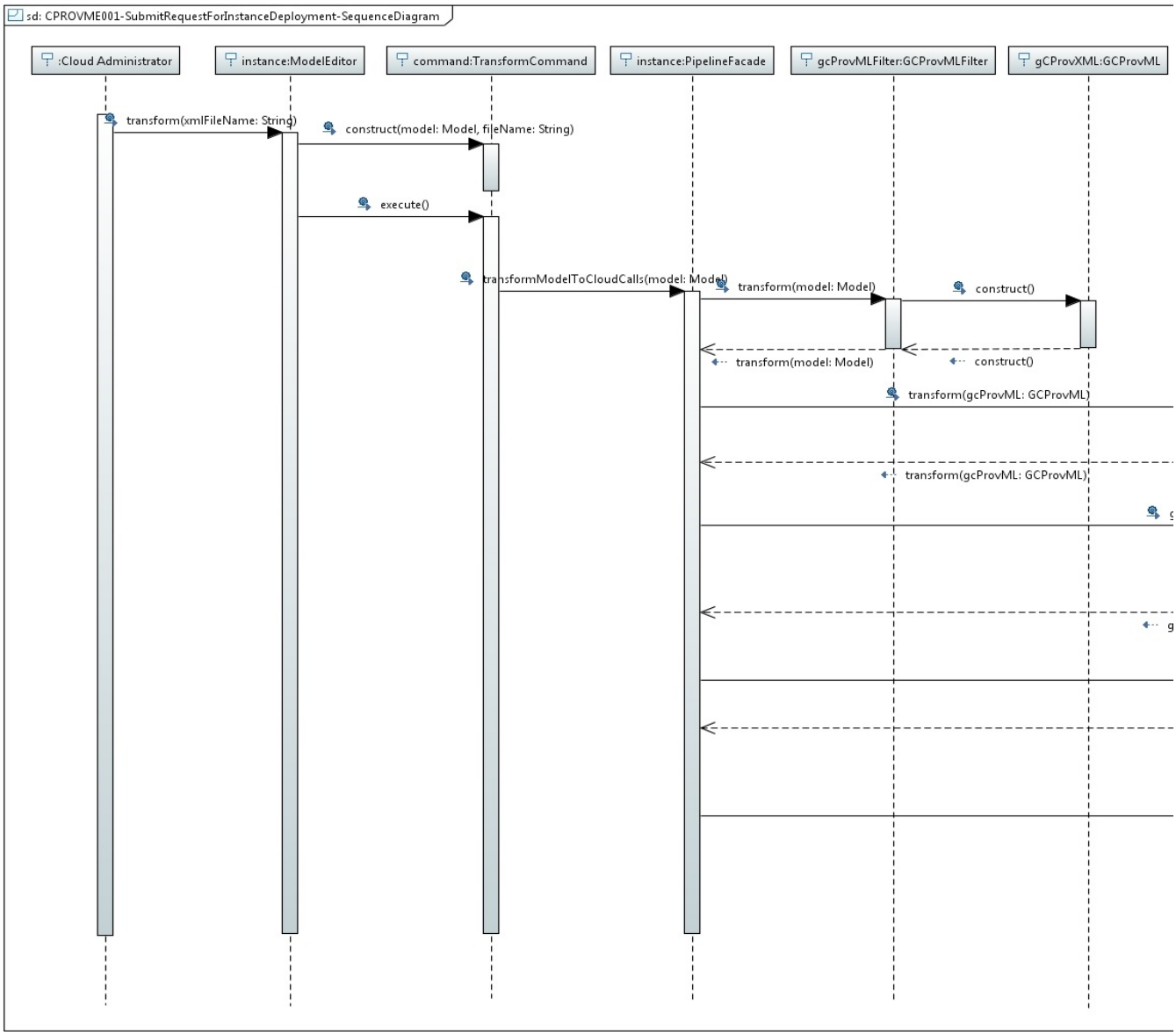


Figure 3-15: Zoomed in UML Sequence Diagram SubmitRequestForInstanceDeployment (Left Side)

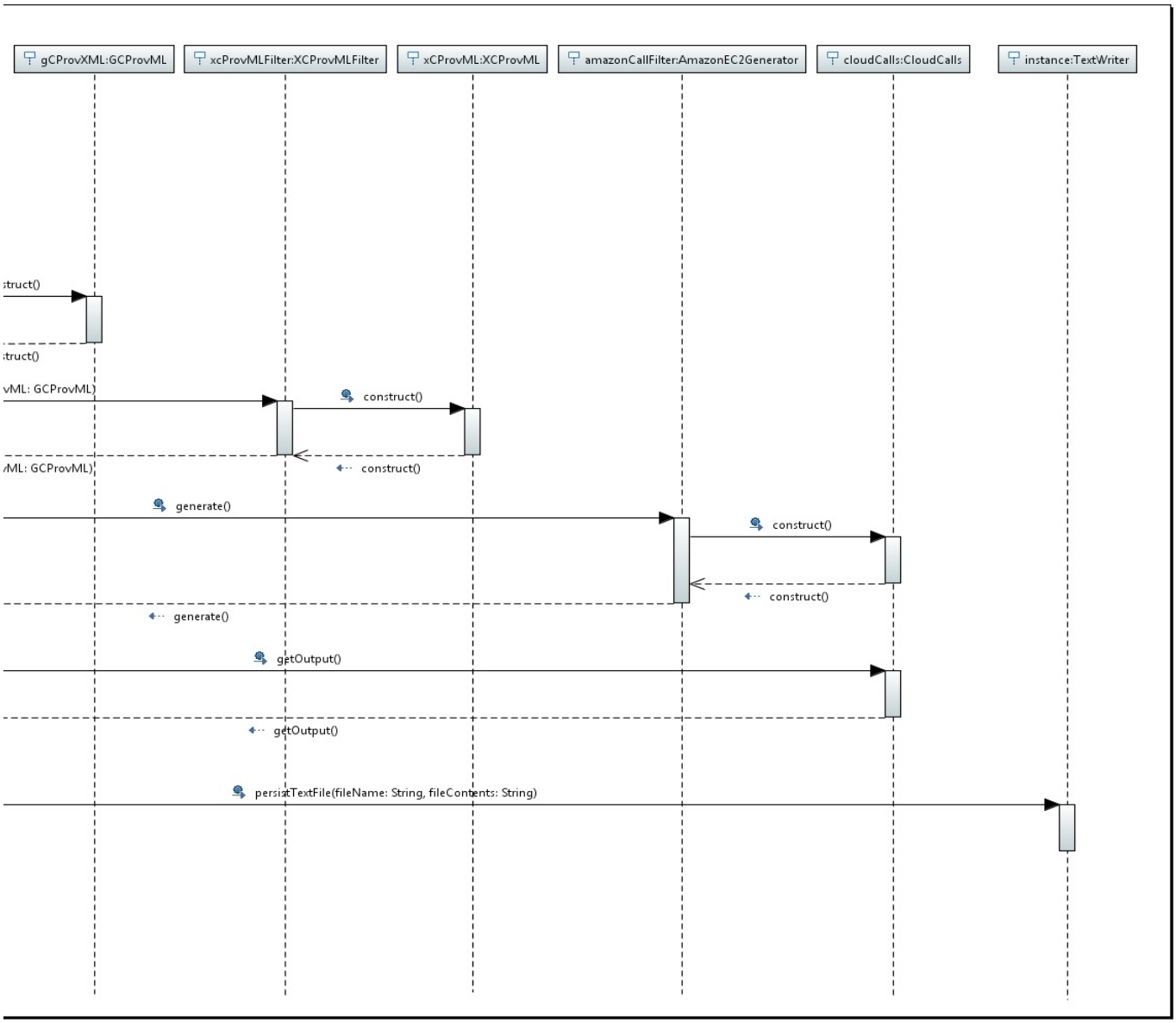


Figure 3-16: Zoomed in UML Sequence Diagram SubmitRequestForInstanceDeployment (Right Side)

3.2.2 State Machines Diagrams

The purpose of the EMF/GMF allows the user to switch from the model opened state to the transformation state from Model to G-CProvML. In order to be able to go ahead with any transformation the model must first validate successfully without generating any errors. In the event that the validation fails, then the state is returned to the current model to be edited. If the transformation is successful, the generated G-CProvML is then output to a file.

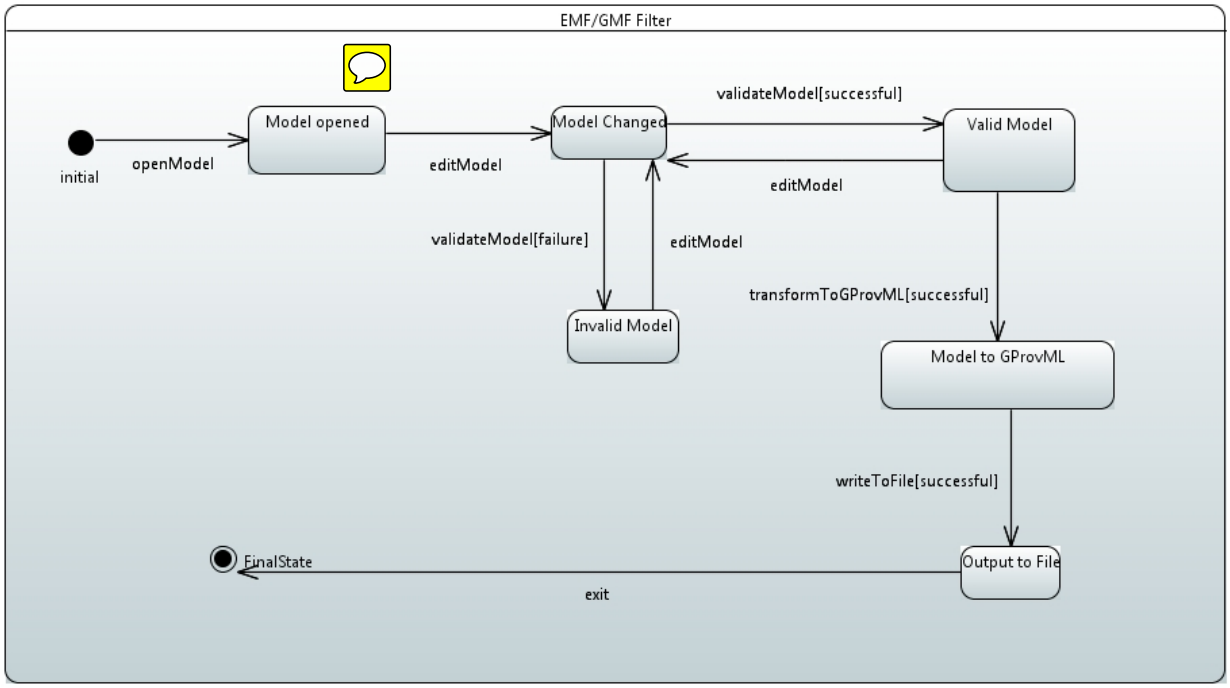


Figure 3-17 UML State Machine Diagram EMF/GMF Filter

The purpose of this filter shines once the G-CProvML file has been generated and it needs to be transformed to X-CProvML, if the transformation is successful, then the X-CProvML is outputted to a file and proceeds to the final state. In this case if the transformation is not successful, then it is redirected to an "Error" state representing that the input file is probably corrupted. if that is the case, it is redirect to the final state.

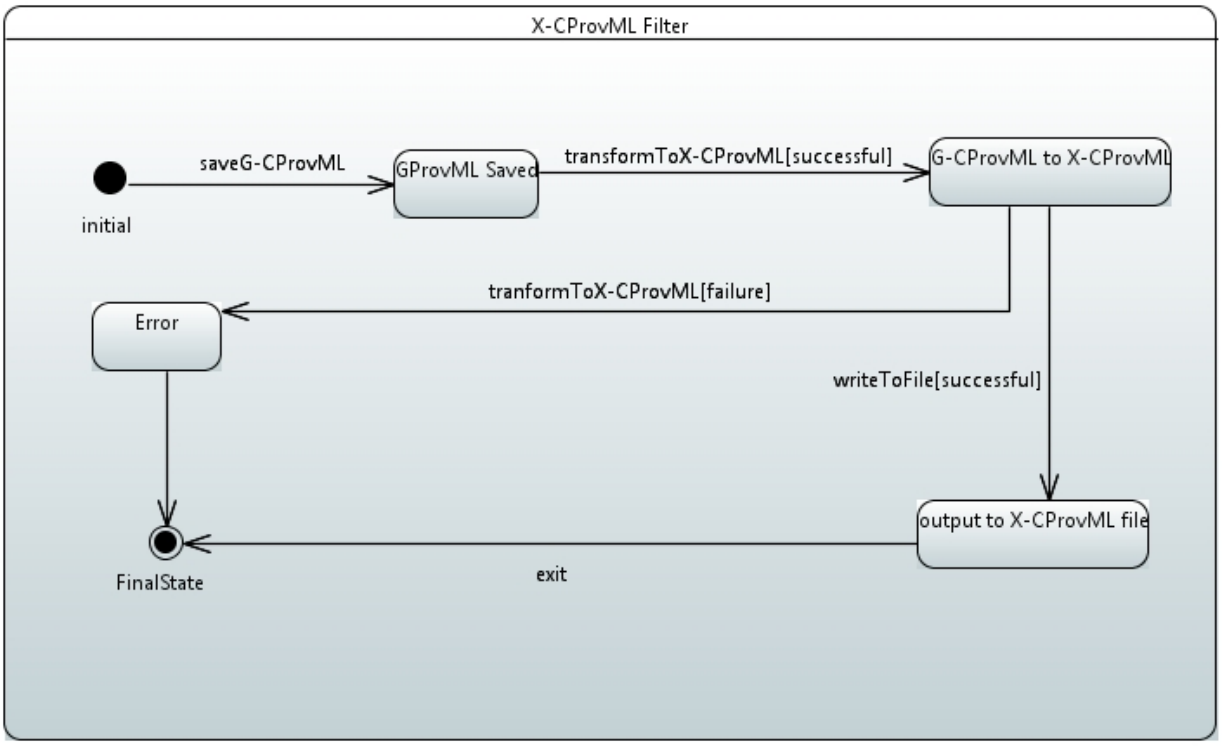


Figure 3-18 UML State Machine Diagram X-CProvML Filter

The purpose of this subsystem is to transform the previously generated X-CProvML to its final CPS appropriate calls. Once the X-CProvML is saved then it is transformed to the Cloud Calls. The user must then choose what type of calls to output (CPS provider here). In the case that the EC2 calls are chosen the appropriate file is generated, if the google Calls are selected the google CPS calls are generated and output to a file. In the event of a transformation failure, an error state will be produced and finally redirected to the final state. This type of failure can probably only happen if the input file is not of the right type, or corrupted.

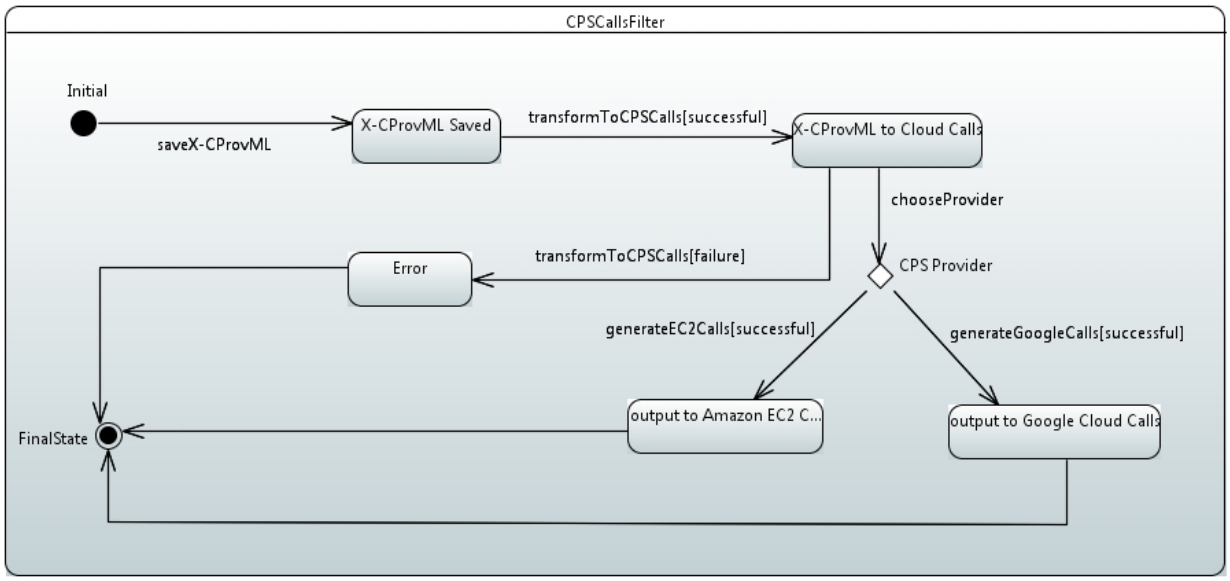


Figure 3-19 UML State Machine Diagram CPSCallsFilter

3.3 Detailed Class Design

The detailed Class Diagram is the entire CPS system, it is divided into four quadrants with our main controller class in the middle of the diagram (ModelEditor). The ModelEditor class has the role of being the intermediate class between the user requests and the views. Its methods reflects all the possible changes doable to the model, as well as refreshing the views once they have been changed. The Modeling Environment (ME subsystem) are the classes located in the bottom left of the diagram. All the classes representing the types of node and connections are displayed, Model (object representing the entire CPS diagram), ModelElement (node class) abstract class, Storage, Network, security group, Environment, and instance all represent the possible nodes that a user can add to a model.

Moving on to the view subsystem (still part of the Modeling Environment) located on the left of the main controller class. A strategy design pattern is used to help render the appropriate view for the user (the view is being rendered by the EMF/GMF platform here). The viewStrategy class has 3 children classes each designated to render a specific view (CanvasView, OutlineView, TreeView).

The controller subsystem (Modeling Environment) is located on the right side of main controller. It is mainly designed as a Command pattern to support each command that the user would need to execute and possibly undo at a later point in time. There is a respective command class for each of the model classes (plus the Execution Environment classes). Everytime a change to the model applied a corresponding command object is created in order to keep track of the changes. AddNodeCommand class for ModelElement class, CreateConnectionCommand for AddConnection class, and RenameCommand class for the setName method of the ModelElement class just to name a few.

The last subsystem (Part of the Execution Environment) is located on the top left and right of the main controller class. The EE utilizes a Facade/singleton pattern class called PipelineFacade class as a pipe for all transformation requests. Through the use of two annex classes (XMLWriter and TextWriter, XML factory builder class responsible for the heavylifting), The facade class passes the Model object to be transformed to the first stage of transformation to GCProvML XML type files. GCProvMLfilter class outputs to the GCProvML class, The GCProvML output object is fed to the XCProvMLFilter class and outputs to the XCProvML class, The XCProvML object is fed in turn to the CPSCAllFilter class, and Finally uses another stragety pattern to CloudCallGenerator class to be finalized into the appropriate CPS calls for the specified Cloud Provisioning provider (Amazon EC2, Google Cloud, etc…).

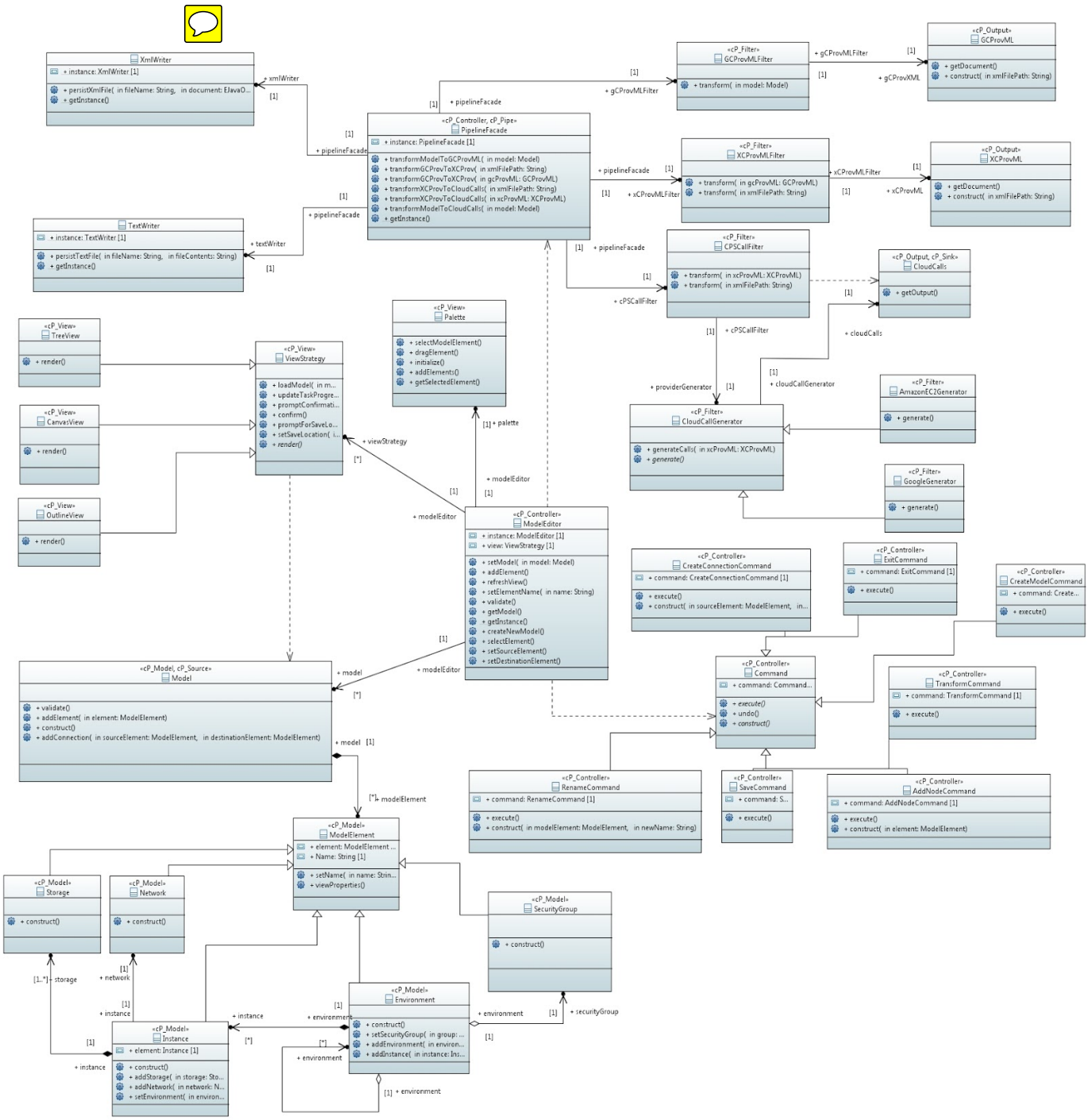


Figure 3-20 UML Detailed Class Diagram

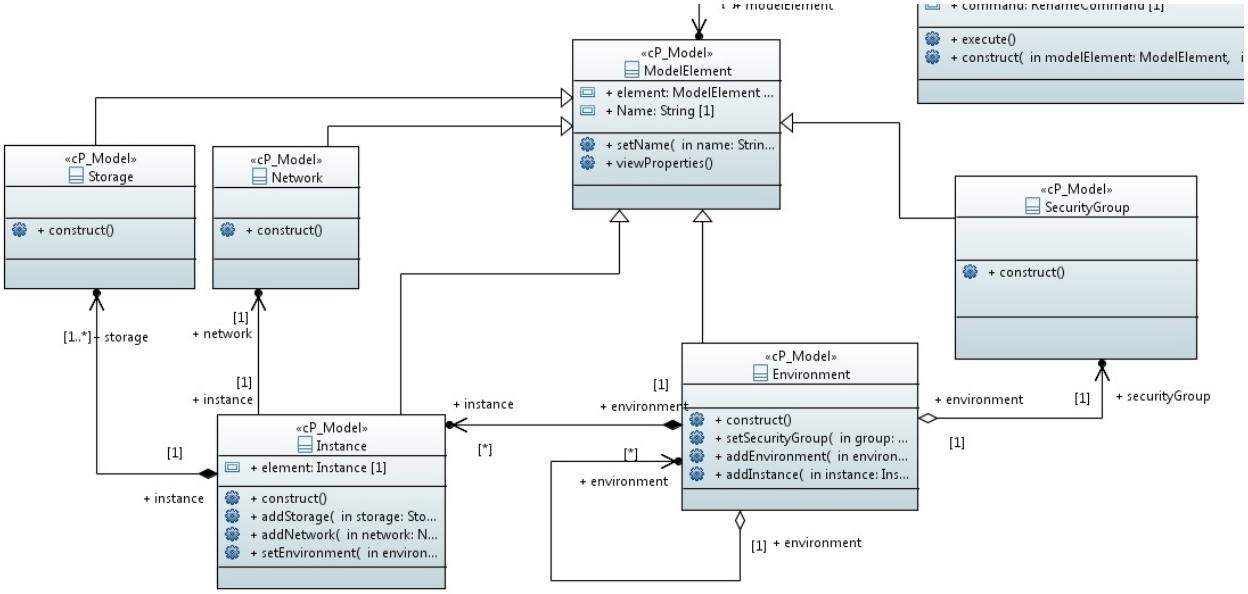


Figure 3-21 Zoomed in UML Detailed Class Diagram (Model)

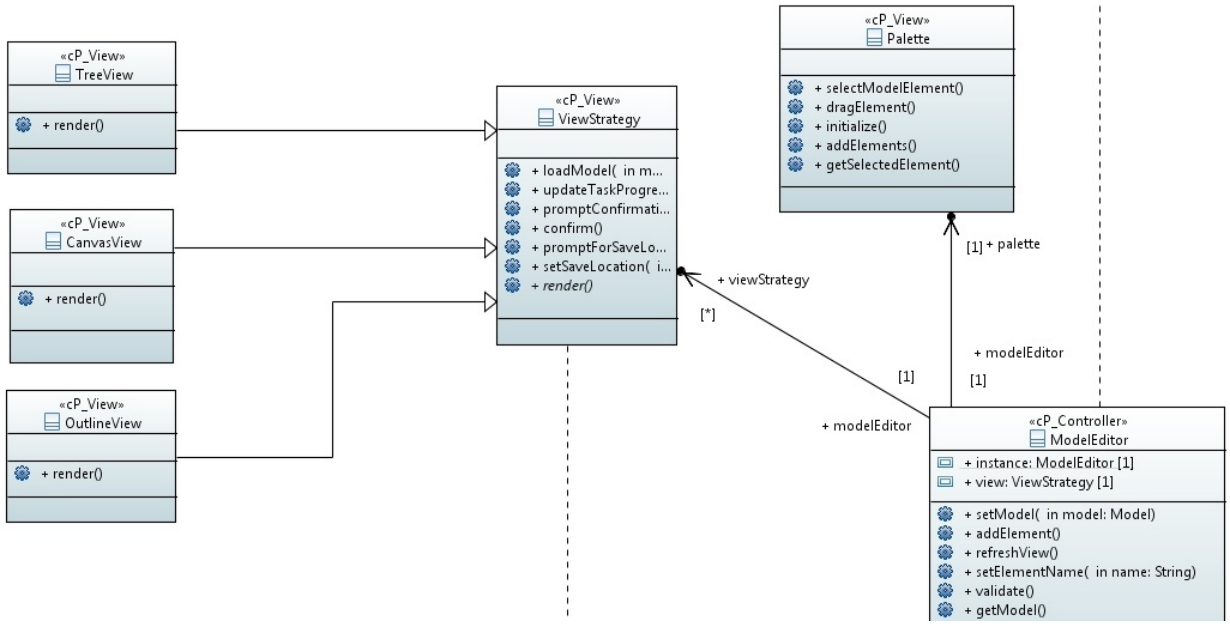


Figure 3-22 Zoomed in UML Detailed Class Diagram (View)

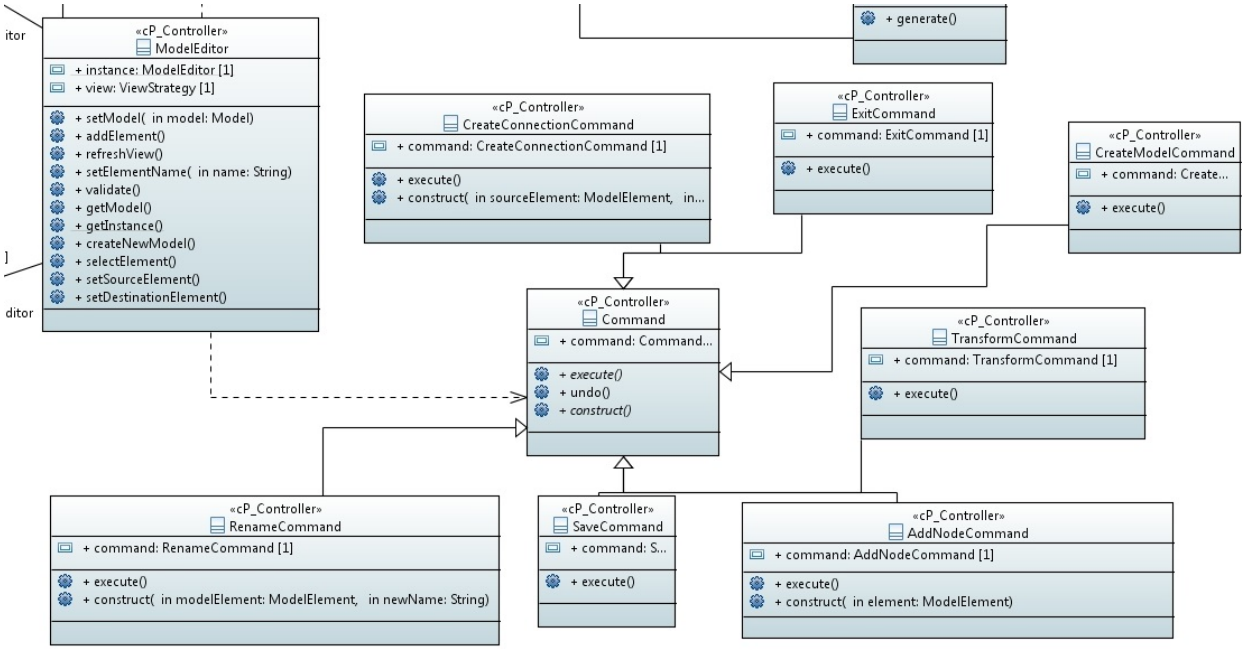


Figure 3-23 Zoomed in UML Detailed Class Diagram (Controller)

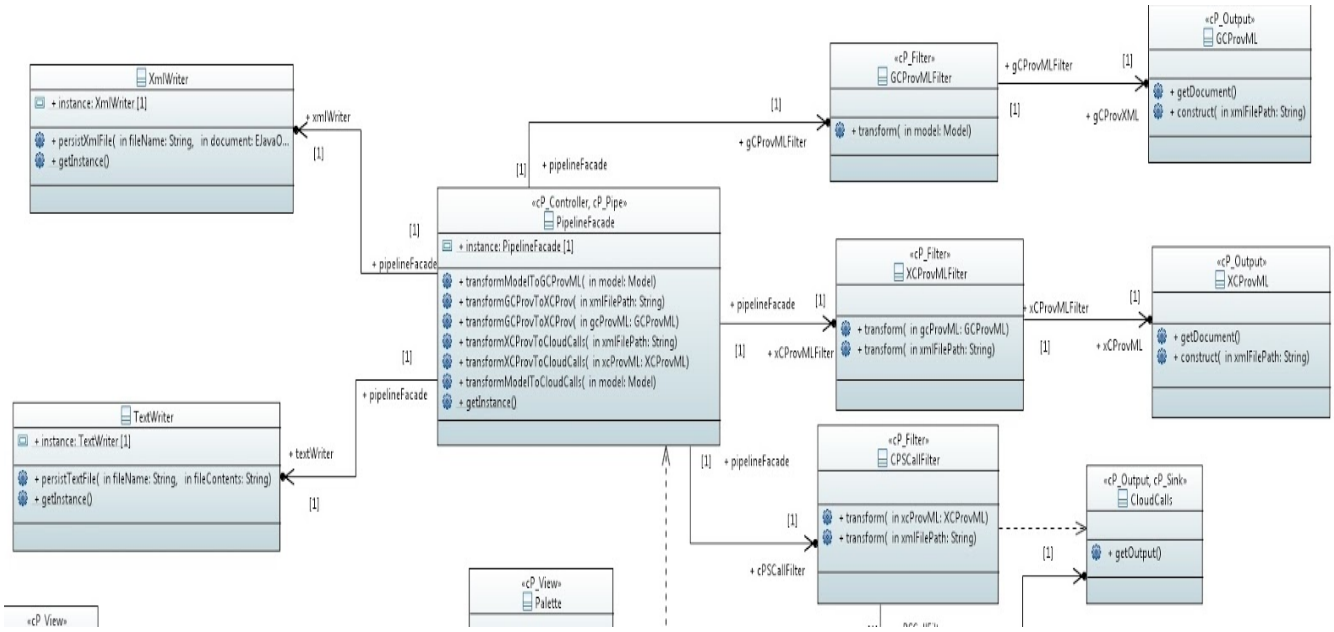


Figure 3-24 Zoomed in UML Detailed Class Diagram (Filters)

The following diagram represents the modeling environment that the user will interact with in order to create a CPS diagram on the canvas. The NodesDiagram class serves as a container for all the subsequent nodes to reside in. The Node class represents the parent class of the possible

nodes. The modeling environment will have five types of Nodes (Storage, Instance, Environment, SecurityGroup, and Network). The storage node will have a storageType referenced through an enumeration type called StorageType. The Instance Nodes will have a referenced CPU and OS type referenced by an enumeration type called CPU type and OSType respectively. Each instance nodes can have many Storage nodes and at most one Network. Only 5 instance nodes can belong to a environment node, but environment node can be comprised of other environment nodes. Each environment node can have at most one SecurityGroup Node. Nodes in general can be linked with connections, each node can serve either as source of the connection or destination of the connection. Nodes can have many incoming and outgoing connections. Attribute CPU and Size of Instance Node and Storage Nodes have constraints placed on their value: CPU cannot be no less then 3Ghz in clock speed, and the Size of the storage can be no less then 3GB, and no more than 2000GB, Ram size can be no less than 1GB and no more than 16GB of memory.

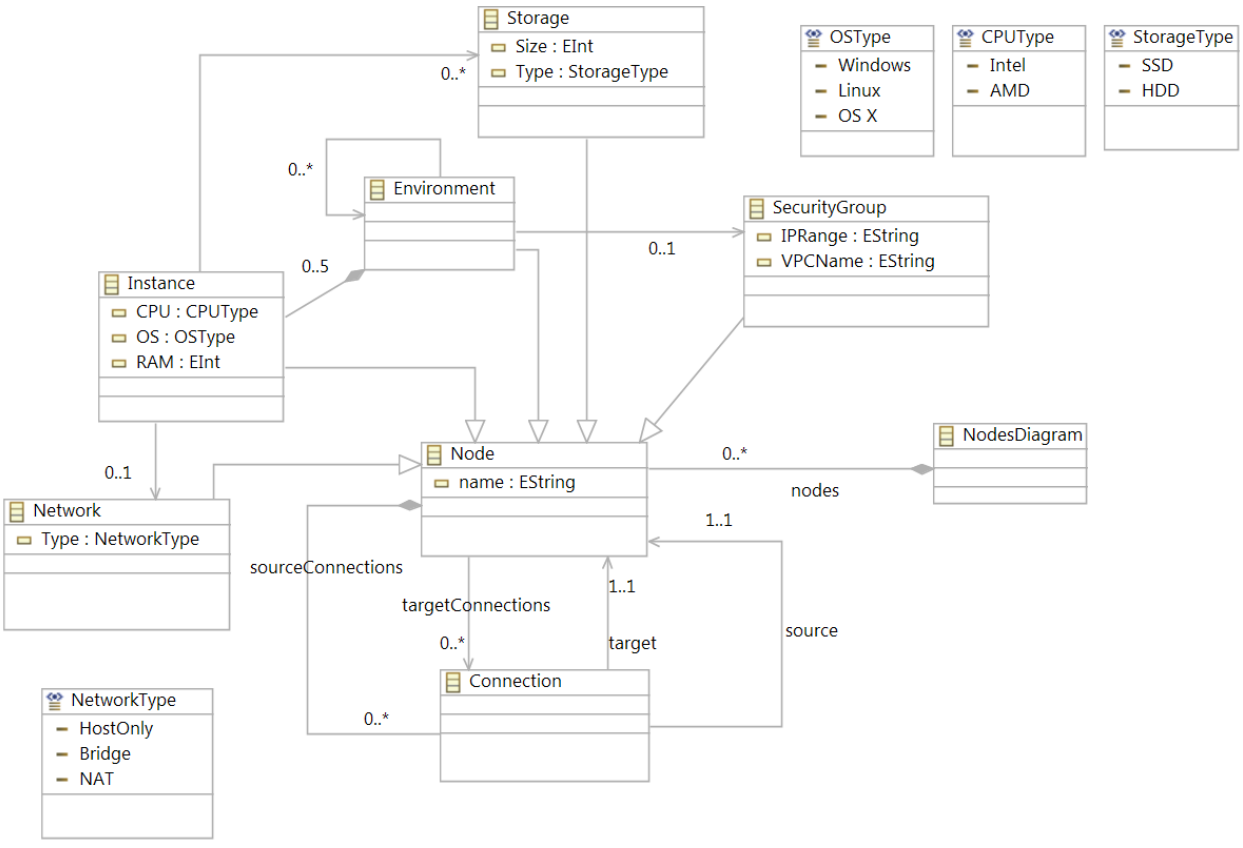


Figure 3-25 Class Diagram (Ecore) to Generate Modeling Environment using EMF/GMF

3.4 OCL Constraints

In this section we will present OCL constraints for the static model, or the main controller of our main subsystem. Our main controller is the ModelEditor class which is within our main subsystem, the EMF/GMF Filter subsystem.

```
context ModelEditor
inv: self.instance <> null

context ModelEditor
inv: self.view <> null

context ModelEditor::GetModel()
pre: self.model <> null

context ModelEditor::transform(fileName: String)
pre: fileName.size() > 0 and fileName.size() < 64

context ModelEditor::setElementName(name: String)
pre: name.size() > 0 and name.size() < 64


context ModelEditor::setSourceElement()
post: self.sourceElement <> null

context ModelEditor::setDestinationElement()
post: self.destinationElement <> null

context ModelEditor::selectElement()
post: self.selectedElement <> null

context ModelEditor::addElement(element: ModelElement)
post: self.elements.size() = self.elements@pre.size() + 1
```

4 Glossary

Class Diagram – A model representing the different classes within a software system and how they interact with each other.

Component – A physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.

Model – an abstract representation of a system that enables us to answer questions about the system.

Post-condition – A predicate that must be true after an operation is invoked.

Precondition – A predicate that must be true before an operation is invoked.

Sequence Diagram – A model representing the different objects and/or subsystems of a software project and how they relate to each other during different operations for a given use case.

State Machine Diagram - A model representing the different states possible for a software subsystem.

Unified Modeling Language (UML) – A standard set of notations for representing models.

Use Case – A general sequence of events that defines all possible actions between one or many actors and the system for a given piece of functionality.

Model Driven Development Software – Software development paradigm that roots in software product line engineering, which is the discipline of designing and building families of applications.

Shape – An image that can be dragged into the canvas in the Visual Environment. It can be connected with other nodes using connections in order to build a communication model.

Connections – A line that connects one shape with another.

Participant – A special shape that represents a local or remote user of the communication system. One or more participants can establish a conversation using a communication model.

Network – A special shape that represents what the user is using to connect to the system.

Security Group – A special shape that represents what the user is using to connect to the system.

Storage – A special shape that represents a hard drive on an instance or outside an instance server.

Instance – A special shape that represents a server in the cloud.

Cloud – a term to define a group or single computer whose location is unimportant, but that the user can connect to and control.

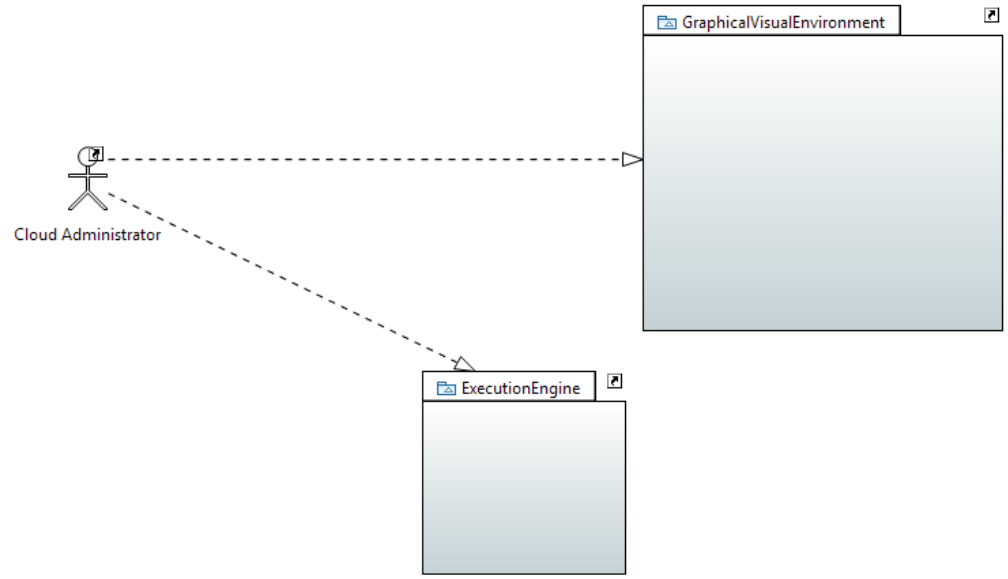
Palette – A location on the GUI where are the shapes are aggregated for the user to click and drag.

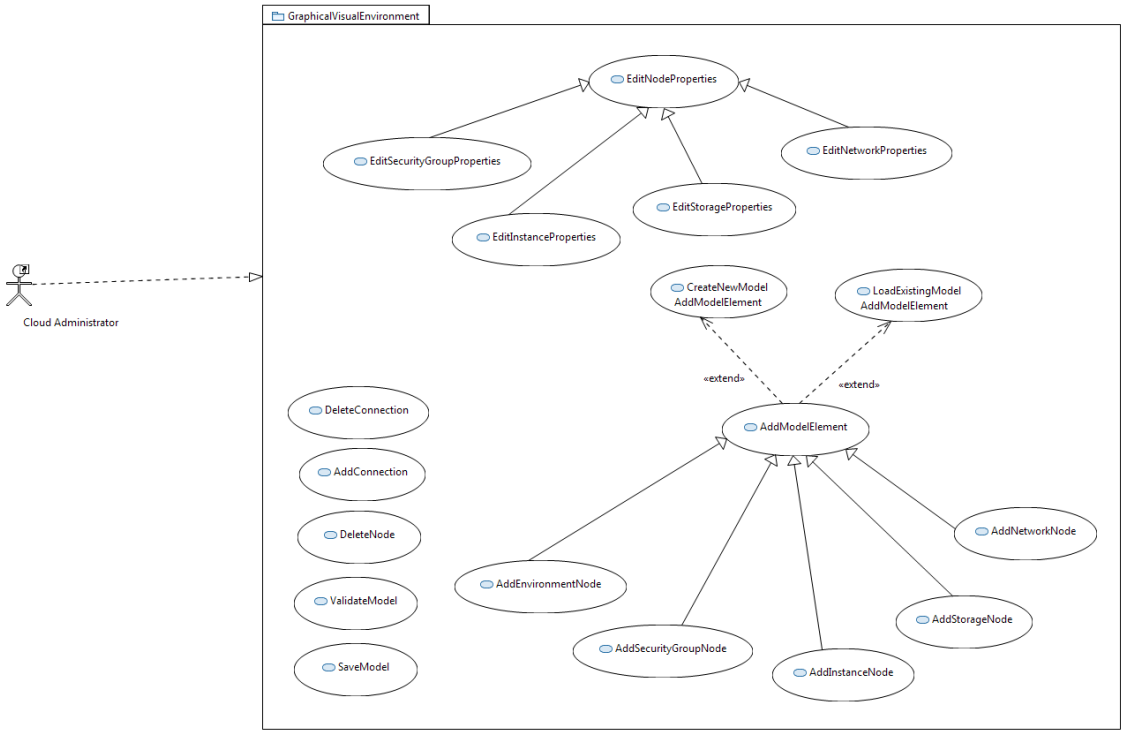
Workspace – The area where the user drags shapes and connections to build a communication model.

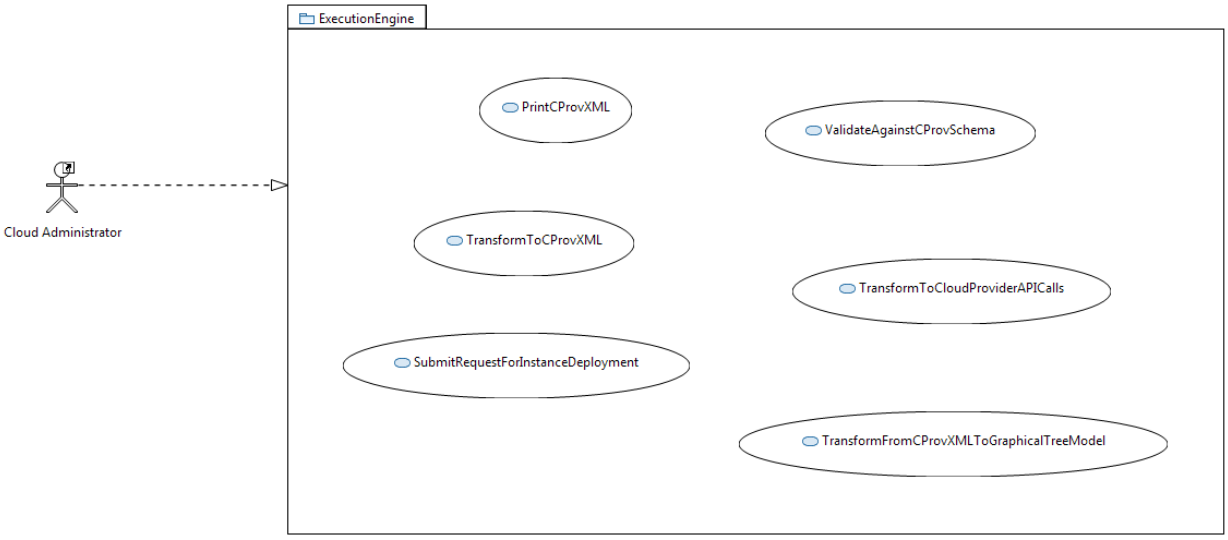
Constraints – A voluntary limitation imbedded into the design to obey syntax or semantic rule(s).

5 Appendix

5.1 Appendix A - Use case diagrams






ExecutionEngine
PrintCProvXML
ValidateAgainstCProvSchema
TransformToCProvXML
TransformToCloudProviderAPICalls
SubmitRequestForInstanceDeployment
TransformFromCProvXMLToGraphicalTreeModel
Cloud Administrator

5.2 Appendix B – Use cases

Execution Engine Use Cases

CPROV-EE Actors

- Cloud Administrator - A cloud provisioning administrator who is interested in modeling a particular deployment.

CPROV-EE Use Cases

Use Case ID: **CPROVEE001 Submit request for instance deployment**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visual environment must be running on the system.
2. A model must be active and must contain an instance of a server.

Description:

1. Use case begins when the user clicks on the deploy button on the toolbar.
2. The system shall provide the user a confirmation message with two options: Confirm/Cancel.
3. The user clicks on the Confirm button on the confirmation message.
4. The system translates the graphical representation of the server instance into a series of commands that utilize the Amazon EC2 API
5. The system executes the API calls to the Amazon EC2 service

Post-conditions:

1. A message appears stating that the deployment was successful.
2. The system returns focus to the graphical elements present on the workspace.

Alternative Courses of Action:

1. In step D.3 (Step 3 of Descriptions section), the user has the option to cancel the deployment of the selected diagram element present on the workspace.

Exceptions:

One or more element of the diagram cannot be translated. In this case an error message is returned to the user.

Related Use Cases:

CPROVME001 adding a Instance Node.

Decision Support

Frequency:

Each time the main user would like to deploy the diagram.

On average once per model.

Criticality:

High, it allows the user to establish the essence of the cloud deployment which provide virtual machines, and servers.

Risk:

Risk: medium risk, the system must communicate with a third party API and adhere to its contract. Since the API is always subject to change, our system must be maintained.

Constraints:

1. Usability:
    a) No previous Training Time
    b) 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..

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVEE002 Transform to CPROV XML**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. A model must be active and must contain an instance of a server.
3. The model must pass validation prior to transformation.

Description:

1. Use case begins when the user clicks on the "Transform to XML" button on the toolbar.
2. The system shall provide the user a confirmation message with two options: Confirm/Cancel.
3. The user clicks on the Confirm button on the confirmation message.
4. The system prompts the user for a Save location.
5. The user types in a Save location.
6. The user clicks OK.

Post-conditions:

1. A message appears stating that the transformation to XML was successful.
2. The system returns focus to the graphical elements present on the workspace.
3. The system saves the XML schema to a file in a location designated by the user.

Alternative Courses of Action:

1. In step D.3 (step 3 of Descriptions section), the user has the option to cancel the transformation to XML by clicking on the "Cancel" button.

Exceptions:
One or more element of the diagram cannot be translated. In this case an error message is returned to the user.

Related Use Cases:
CPROVME005 Add Instance Node

Decision Support

Frequency: Each time the main user would like to transform the diagram into XML schema.
On average once per model.

Criticality: High, It allows the user to use save many XML schemas for use on another system, or for memory saving.

Risk:

Medium risk, the system must communicate with a third party API and adhere to its contract. Since the API is always subject to change, our system must be maintained.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should be able to transform to XML 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) transformation to XML of the current workspace diagram should complete in under 2 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..

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVEE003 Validation against Cloud provisioning Schema**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. A model must be active and must contain at least an instance of a server.

Description:

1. Trigger: the use case begins when the user clicks right clicks on the workspace.
2. In the right click drop down menu, the user clicks on the “Validate”
3. The system start to validation process with a progress bar starting at 0 % and ends at 100% showing that the validation has been done.

Post-conditions:

1. A message appears stating that the validation process reported errors or no errors.
2. The system returns focus to the graphical elements present on the workspace.

Alternative Courses of Action:

1. In step D.2 (Step 2 of Descriptions section), the user has the option select validate from the menu bar under “Validation”.

Exceptions:

If one or more elements of the diagram does not respect our schema. In this case the error(s) are highlighted in the validation submenu located in the lower part of the screen.

Related Use Cases:

CPROVME005 Add Instance Node

Decision Support

Frequency: Each time the main user would like to validate the diagram.
On average once per changes applied to the model.

Criticality: High, It allows the user to verify is the diagram represents a sound model before the translation and deployment phases.

Risk: low risk, the system uses the EMF/GMF API to implement the validation.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should be able to validate a diagram in under 2 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) Validation 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..

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVEE004** Transformation to Amazon EC2 format

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. A model must be active and must contain at least an instance of a server.

Description:

1. Trigger: the use case begins when the user clicks the “Transform” menu button location on the top menu bar.
2. in the top menu bar list, the user selects “Transform to EC2 format”
3. the system starts the validation process if the model has never been validated.
4. The system starts transformation process in the background.

Post-conditions:

1. A message appears stating that the transformation process has finished.
2. The system returns focus to the graphical elements present on the workspace.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option select transform to EC2 format from the right click drop down menu.

Exceptions:

If the diagram does pass the validation process, Transformation to EC2 format cannot be done. In this case the error(s) are highlighted in the validation submenu located in the lower part of the screen.

Related Use Cases:

CPROVME005 Add Instance Node

CPROVEE003 Validation against Cloud provisioning Schema

Decision Support

Frequency: Each time the main user would like to transform the diagram to the EC2 format. On average once per changes applied to the model.

Criticality: High, It allows the user to Translate the diagram into the Amazon EC2 call which is the foundation of the plug in application.

Risk: High risk, Translating From a graphical language to EC2 format might cause ambiguity issues.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should be able to translate to EC2 format a diagram in under 2 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) Translation process of the current workspace diagram should complete in under 2 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..

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVEE005** Print to XML

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. A model must be active and must contain at least an instance of a server.

Description:

1. Trigger: the use case begins when the user clicks the “Print” menu button location on the top menu bar.
2. in the top menu bar list, the user selects “Print to XML”
3. the system starts the validation process if the model has never been validated.
4. The system starts transformation process in the background.
5. The system outputs the print to the console in eclipse environment.

Post-conditions:

1. The system returns focus to the graphical elements present on the workspace.

Alternative Courses of Action:
No Alternative Courses.

Exceptions:
If the diagram does pass the validation process, The print to XML cannot be done. In this case the error(s) are highlighted in the validation submenu located in the lower part of the screen.

Related Use Cases:
CPROVME005 Add Instance Node
CPROVEE003 Validation against Cloud provisioning Schema

Decision Support

Frequency: Each time the main user would like to print to XML format.
On average once per changes applied to the model.

Criticality: Medium, It allows the user to see the XML version of the model currently on the workspace. Furthermore, it allows possibility of transport to another system.

Risk: High risk, Translating From a graphical language to XML format might cause ambiguity issues.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should be able to print to XML format a diagram 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) Translation process of the current workspace diagram should complete in under 2 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..

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVEE006** XML to Model

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. A model must be active and must contain at least an instance of a server.

Description:

1. Trigger: the use case begins when the user clicks the “File” menu button location on the top menu bar.
2. in the top menu bar list, the user selects “load XML file”
3. A file picker windows opens.
4. the user selects the location of the XML file in the system.
5. The system starts transformation process in the background.
6. The system outputs diagram to the workspace.

Post-conditions:

1. The system returns focus to the newly created diagram on the workspace.

Alternative Courses of Action:

1. In D1 (step 1 of Description), the user can start the use case by right clicking the model in the tree view located on the left hand side interface.

Exceptions:

If the file loaded in the file picker is not in XML format the system shall reject the file, and send an error message to the user.

Related Use Cases:

CPROVME005 Add Instance Node

CPROVEE003 Validation against Cloud provisioning Schema

Decision Support

Frequency: Each time the main user would like to print to XML format.
On average once per changes applied to the model.

Criticality: Medium, It allows the user to create a diagram straight from an XML file. Furthermore, it allows possibility for the XML to come from another system or subsystem.

Risk: High risk, Translating from XML to a graphical language could produce errors in the positioning of the graphical elements in relation to the workspace grid.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should be able load a diagram from XML filein under 3 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) Translation process of the XML file to workspace diagram should complete in under 2 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..

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Modeling Environment Use cases:

CPROV-ME Actors

- Cloud Administrator - A cloud provisioning administrator who is interested in modeling a particular deployment.

CPROV-ME Use Cases

Use Case ID: **CPROVME001 Create New Model**
Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a File button located on the top left hand side of the workspace.

2. The user selects the “New CPROV Model” button.
3. The system opens a new diagram in the currently select Project folder.
4. The system displays a window for the user to enter a Name for the Model.
5. The user enters a name for the model.
6. The user clicks the “OK” button to confirm.
7. The system switches focus to the newly created model workspace.

Post-conditions:

1. The system creates a new Model along with its tree representation located on the left hand side of the workspace.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section) the user can choose to save a Model by selecting the model and pressing “CTRL-N”.

Decision Support

Frequency: Each and every time the main user would like to save a model.

Criticality: High, It allows the user to create a new model.

Risk: Low risk, the system EMF/GMF framework handles Model file creation.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 2 min to create a new model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to create a new model.
    b) Availability – The system shall be available as long as the user’s machine is running.
3. Performance:
    a) To create a new model in general should complete in under 2 minutes.
    b) System handles only one user at time.

4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME002 Add Model Element**
Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user click on a Model Element located in the palette.
2. The user clicks on the workspace in the desired location.
3. The system shall provide the user with a text box to enter the Model Element name.
4. The user types the name.
5. An icon representing the Model Element is then added to the workspace.

Post-conditions:

1. The Model Element is then created and focus is switched to the properties window.
2. The properties are now changed to their default value.

Alternative Courses of Action:

1. In step D.2 (Step 2 of Descriptions section), the user has the option to cancel the currently selected element on the palette.
2. In step D.4 the user can choose to not give the new Model Element a name. In this case, system will provide a default name.

Related Use Cases:
CPROVME003 Add Environment node

Decision Support

Frequency: Each and every time the main user would like to create a Model Element of a server. On average 5 Model Elements are added to a cloud deployment model.

Criticality: High, It allows the user to establish the essence of the cloud deployment which provide virtual machines, and servers.

Risk: Low risk, the system is using UML unambiguous modeling environment.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to add an Model Element to their model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to add an Model Element.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Model Element addition to the workspace should complete in under one second.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME003 Add Environment Node**
Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user click on the Environment Node icon located in the palette.
2. The user clicks on the workspace in the desired location.
3. The system shall provide the user with a text box to enter the Environment Node name.
4. The user types the name.
5. An icon representing the Environment Node is then added to the workspace.

Post-conditions:

1. The Environment Node is then created and focus is switched to the properties window.
2. The properties are now changed to their default value.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option to cancel the currently selected element on the palette.
2. In step D.4 the user can choose to not give the new Environment Node a name. In this case, system will provide a default name.

Related Use Cases:
CPROVME003 Add Environment node

Decision Support

Frequency: Each and every time the main user would like to create a Environment Node of a server. On average 5 Environment Nodes are added to a cloud deployment model.

Criticality: High, It allows the user to establish the essence of the cloud deployment which provide virtual machines, and servers.

Risk: Low risk, the system is using UML unambiguous modeling environment.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to add an Environment Node to their model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to add an Environment Node.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Environment Node addition to the workspace should complete in under one second.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME004 Add Security Group Node**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. At least 2 person/user icons must be present on the workspace for the SecurityGroup to be needed. (it is not mandatory)

Description:

1. Trigger: the use case begins when the user click on the SecurityGroup icon located in the palette.
2. The user clicks on the workspace in the desired location.
3. The system shall provide the user with a text box to enter the person's name.
4. The user types the name.
5. An icon representing the SecurityGroup is then added to the workspace.

Post-conditions:

1. The SecurityGroup is then created and focus is switched to its properties window.
2. The properties are now changed to their default value.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option to cancel the currently selected element on the palette.
2. In step D.4 the user can choose to not give the new SecurityGroup a name. In this case, system will provide a default name.

Decision Support

Frequency: Each and every time the main user would like to create a SecurityGroup of a server. On average 2 SecurityGroup are added to a cloud deployment model.

Criticality: High, It allows the user to control the access of potential system users once the model is deployed for execution.
Risk: Low risk, the system is using UML unambiguous modeling environment.

Constraints:

1. Usability:

    a) No previous Training Time

b) On average the user should take no more than 1 min to add an SecurityGroup to their model.

2. Reliability:

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

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

3. Performance:

a) the SecurityGroup addition to the workspace should complete in under two second.

b) System handles only one user at time.

4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME005 Add Instance Node**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user click on the instance icon located in the palette.
2. The user clicks on the workspace in the desired location.
3. The system shall provide the user with a text box to enter the instance name.
4. The user types the name.

5. An icon representing the instance is then added to the workspace.

Post-conditions:

1. The instance is then created and focus is switched to the properties window.
2. The properties are now changed to their default value.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option to cancel the currently selected element on the palette.
2. In step D.4 the user can choose to not give the new instance a name. In this case, system will provide a default name.

Decision Support

Frequency: Each and every time the main user would like to create a instance of a server. On average 5 instances are added to a cloud deployment model.

Criticality: High, It allows the user to establish the essence of the cloud deployment which provide virtual machines, and servers.

Risk: Low risk, the system is using UML unambiguous modeling environment.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to add an instance to their model.
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) Instance addition to the workspace should complete in under one second.
    b) System handles only one user at time.

4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME006 Add Storage Node**
Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user click on the Storage Node icon located in the palette.
2. The user clicks on the workspace in the desired location.
3. The system shall provide the user with a text box to enter the Storage Node name.
4. The user types the name.
5. An icon representing the Storage Node is then added to the workspace.

Post-conditions:

1. The Storage Node is then created and focus is switched to the properties window.
2. The properties are now changed to their default value.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option to cancel the currently selected element on the palette.
2. In step D.4 the user can choose to not give the new Storage Node a name. In this case, system will provide a default name.

Related Use Cases:
CPROVME003 Add Environment Node

Decision Support

Frequency: Each and every time the main user would like to create a Storage Node of a server. On average 5 Storage Nodes are added to a cloud deployment model.

Criticality: High, It allows the user to establish the essence of the cloud deployment which provide virtual machines, and servers.

Risk: Low risk, the system is using UML unambiguous modeling environment.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to add an Storage Node to their model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to add an Storage Node.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Storage Node addition to the workspace should complete in under one second.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME007 Add Network Node**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user click on the Network Node icon located in the palette.
2. The user clicks on the workspace in the desired location.
3. The system shall provide the user with a text box to enter the Network Node name.
4. The user types the name.
5. An icon representing the Network Node is then added to the workspace.

Post-conditions:

1. The Network Node is then created and focus is switched to the properties window.
2. The properties are now changed to their default value.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option to cancel the currently selected element on the palette.
2. In step D.4 the user can choose to not give the new Network Node a name. In this case, system will provide a default name.

Related Use Cases:
CPROVME003 Add Environment Node

Decision Support

Frequency: Each and every time the main user would like to create a Network Node of a server. On average 5 Network Nodes are added to a cloud deployment model.

Criticality: High, It allows the user to establish the essence of the cloud deployment which provide virtual machines, and servers.

Risk: Low risk, the system is using UML unambiguous modeling environment.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to add an Network Node to their model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to add an Network Node.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Network Node addition to the workspace should complete in under one second.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME008 Edit Node Properties**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a Node icon located on the workspace.
2. The user clicks on the properties window.
3. The user selects the property to change by clicking on the property.
4. The user makes changes to the property selected.

Post-conditions:

1. The System applies the changes to the selected properties.
2. The properties are now changed to the chosen value(s).

Alternative Courses of Action:

1. In step D.3 (step 3 of Descriptions section) the user can choose to not to change any values in the properties.
2. In step D.4 (step 4 of Descriptions section) the user can choose undo the changes just made in the properties window by pressing CTRL-Z (undo)

Decision Support

Frequency: Each and every time the main user would like to edit the properties of a Node. Most Nodes have their properties edited prior to deployment.

Criticality: High, It allows the user to customize the settings of each of the Nodes present on the Model.

Risk: Low risk, the system is using UML unambiguous modeling environment. Editing the properties of a Node is merely using designed mutators for that object.

Constraints:

1. Usability:

a) No previous Training Time

b) On average the user should take no more than 1 min to edit the properties of a Node present on the Model

2. Reliability:

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

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

3. Performance:

a) Node editing in general should complete in under 2 minute.

b) System handles only one user at time.

4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME009 Edit Instance Properties**
Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on an Instance icon located on the workspace.
2. The user clicks on the properties window.
3. The user selects the property to change by clicking on the property.
4. The user makes changes to the property selected.

Post-conditions:

1. The System applies the changes to the selected properties.
2. The properties are now changed to the chosen value(s).

Alternative Courses of Action:

1. In step D.3 (Step 3 of Descriptions section) the user can choose to not to change any values in the properties.
2. In step D.4 (Step 4 of Descriptions section) the user can choose undo the changes just made in the properties window by pressing CTRL-Z (undo)

Decision Support

Frequency: Each and every time the main user would like to edit the properties of a Instance. Most Instances have their properties edited prior to deployment.

Criticality: High, It allows the user to customize the settings of each of the Instance Nodes present on the Model.

Risk: Low risk, the system is using UML unambiguous modeling environment. Editing the properties of a Instance is merely using designed mutators for that object.

Constraints:

1. Usability:

    a) No previous Training Time

    b) On average the user should take no more than 1 min to edit the properties of an Instance present on the Model

2. Reliability:

    a) Mean time to Failure – 5% failures for every 1000 attempts to edit the properties of an Instance.

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

3. Performance:

    a) Instance editing in general should complete in under 2 minute.

    b) System handles only one user at time.

4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME010 Edit Security Group Properties**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a Security Group icon located on the workspace.
2. The user clicks on the properties window.
3. The user selects the property to change by clicking on the property.
4. The user makes changes to the property selected.

Post-conditions:

1. The System applies the changes to the selected properties.
2. The properties are now changed to the chosen value(s).

Alternative Courses of Action:

1. In step D.3 (step 3 of Descriptions section) the user can choose to not to change any values in the properties.
2. In step D.4 (step 4 of Descriptions section) the user can choose undo the changes just made in the properties window by pressing CTRL-Z (undo)

Decision Support

Frequency: Each and every time the main user would like to edit the properties of a Security Group. Most Security Groups have their properties edited prior to deployment.

Criticality: High, It allows the user to customize the settings of each of the Security Group icons present on the Model.

Risk: Low risk, the system is using UML unambiguous modeling environment. Editing the properties of a Security Group is merely using designed mutators for that object.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to edit the properties of a Security Group present on the Model
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to edit the properties of an Security Group.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Security Group editing in general should complete in under 2 minute.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME011 Edit Storage Properties**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a Storage icon located on the workspace.
2. The user clicks on the properties window.
3. The user selects the property to change by clicking on the property.
4. The user makes changes to the property selected.

Post-conditions:

1. The System applies the changes to the selected properties.
2. The properties are now changed to the chosen value(s).

Alternative Courses of Action:

1. In step D.3 (step 3 of Descriptions section) the user can choose to not to change any values in the properties.
2. In step D.4 (step 4 of Descriptions section) the user can choose undo the changes just made in the properties window by pressing CTRL-Z (undo)

Decision Support

Frequency: Each and every time the main user would like to edit the properties of a Storage. Most Storages have their properties edited prior to deployment.

Criticality: High, It allows the user to customize the settings of each of the Storage icons present on the Model.

Risk: Low risk, the system is using UML unambiguous modeling environment. Editing the properties of a Storage is merely using designed mutators for that object.

Constraints:

1. Usability:

a) No previous Training Time

b) On average the user should take no more than 1 min to edit the properties of a Storage present on the Model

2. Reliability:

    a) Mean time to Failure – 5% failures for every 1000 attempts to edit the properties of an Storage.

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

3. Performance:

    a) Storage editing in general should complete in under 2 minute.

    b) System handles only one user at time.

4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME012 Edit Network Properties**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a Network icon located on the workspace.
2. The user clicks on the properties window.
3. The user selects the property to change by clicking on the property.
4. The user makes changes to the property selected.

Post-conditions:

1. The System applies the changes to the selected properties.

2. The properties are now changed to the chosen value(s).

Alternative Courses of Action:

1. In step D.3 (step 3 of Descriptions section) the user can choose to not to change any values in the properties.
2. In step D.4 (step 4 of Descriptions section) the user can choose undo the changes just made in the properties window by pressing CTRL-Z (undo)

Decision Support

Frequency: Each and every time the main user would like to edit the properties of a Network. Most Networks have their properties edited prior to deployment.

Criticality: High, It allows the user to customize the settings of each of the Network icons present on the Model.

Risk: Low risk, the system is using UML unambiguous modeling environment. Editing the properties of a Network is merely using designed mutators for that object.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to edit the properties of a Network present on the Model
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to edit the properties of an Network.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Network editing in general should complete in under 2 minute.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME013 Delete Node**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a Node icon located on the workspace.
2. The user presses the “delete” key.
3. The system deletes the selected Node.

Post-conditions:

1. The System applies the changes to the Model

Alternative Courses of Action:

1. In step D.2 (Step 2 of Descriptions section) the user can choose to delete by right-clicking the Node and selecting Delete from the drop down menu.
2. In step D.2 (Step 2 of Descriptions section) the user can choose to delete by pressing the “BackSpace” key.

Decision Support

Frequency: Each and every time the main user would like to edit the properties of a Network. Most Networks have their properties edited prior to deployment.

Criticality: High, It allows the user to remove the desired Node from the Model Diagram.

Risk: Low risk, the system is using UML unambiguous modeling environment. Deleting an Icon is supported by the EMF/GMF framework..

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to delete a Node present on the Model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to delete a Node from a Model.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Deleting a Node in general should complete in under 30 seconds.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME014 Add Connection**

Details:
Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. At least 2 icons must be present on the workspace before a connection can be added.

Description:

1. Trigger: the use case begins when the user click on the connection icon located in the palette.
2. The user clicks on the workspace on the desired source icon.
3. The user clicks on the workspace on the desired destination icon.
4. The system shall provide the user with a text box to enter the connection's name.
5. The user types the name.
6. An arrow representing the connection between the 2 icons is then added to the workspace.

Post-conditions:

1. The connection is then created and focus is switched to its properties window.
2. The properties are now changed to their default value.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option to cancel the currently selected element on the palette.
2. In step D.4 the user can choose to not give the new connection a name. In this case, system will provide a default name to the newly added connection

---

Decision Support

Frequency: Each and every time the main user would like to create a connection between two designated icons. On average between 5 and 20 connections are added to a cloud deployment model.

Criticality: High, It allows the user to determine the associations among different instances and users and/or sub-networks of the system.

Risk: Low risk, the system is using UML unambiguous modeling environment.

Constraints:

1. Usability:

a) No previous Training Time

b) On average the user should take no more than 1 min to add an connection to their model.

2. Reliability:

a) Mean time to Failure – 5% failures for every 1000 attempts to add an person/user

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

3. Performance:

a) the connection addition to the workspace should complete in under 1 minute

b) System handles only one user at time.

4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME015 Delete Connection**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a Connection icon located on the workspace.
2. The user presses the "delete" key.

3. The system deletes the selected Connection.

Post-conditions:

1. The System applies the changes to the Model.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section) the user can choose to delete by right-clicking the Connection and selecting Delete from the drop down menu.
2. In step D.2 (step 2 of Descriptions section) the user can choose to delete by pressing the "BackSpace" key.

Decision Support

Frequency: Each and every time the main user would like to edit the properties of a Network. Most Networks have their properties edited prior to deployment.

Criticality: High, It allows the user to remove the desired Connection from the Model Diagram.
Risk: Low risk, the system is using UML unambiguous modeling environment. Deleting an Icon is supported by the EMF/GMF framework..

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 1 min to delete a Connection present on the Model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to delete a Connection from a Model.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) Deleting a Connection in general should complete in under 30 seconds.
    b) System handles only one user at time.
4. Supportability:

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

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME016 Load Existing Model**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a File button located on the top left hand side of the workspace.
2. The user selects the "Load Existing Model" button.
3. A file picker window opens for the use to select a File.
4. Once selected the user clicks the "open" button located on the File Picker window.
5. The system loads the selected Model and displays the diagram in a new window in the current workspace.

Post-conditions:

1. The System loads the Model and displays it on the workspace.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section) the user can choose to load and existing model by clicking the open file button from toolbar menu located on the top of the workspace.
2. In step D.4 (step 4 of Descriptions section) the user can choose to cancel by pressing the "cancel" button.

Exceptions:

1. If the file type loaded in the File Picker is unrecognized by the system, a error message will display for the user, and the File Picker window will remain open.

Decision Support

Frequency: Each and every time the main user would like to load an existing file.

Criticality: High, It allows the user to load an existing model into the workspace. Without this function, transport from system to system would be rendered impractical.

Risk: Low risk, the system EMF/GMF framework handles file location, as well as File picker window.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 2 min to load an existing model in the current workspace.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to load an existing model.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) to load an existing model in general should complete in under 2 minutes.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME017 Validate Model**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.
2. A model must be active and must contain at least an instance of a server.

Description:

1. Trigger: the use case begins when the user clicks right clicks on the workspace.
2. In the right-click drop down menu, the user clicks on the "Validate"
3. The system start to validation process with a progress bar starting at 0 % and ends at 100% showing that the validation has been done.

Post-conditions:

1. A message appears stating that the validation process reported errors or no errors.
2. The system returns focus to the graphical elements present on the workspace.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section), the user has the option select validate from the menu bar under "Validation".

Exceptions:
If one or more elements of the diagram does not respect CProv modeling language, the error(s) are highlighted in the validation submenu located in the lower part of the screen.

Related Use Cases:
CPROVME005 Add Instance Node

Decision Support

Frequency: Each time the main user would like to validate the diagram.
On average once per changes applied to the model.

Criticality: High, It allows the user to verify is the diagram represents a sound model before the translation and deployment phases.

Risk: Low risk, the system uses the EMF/GMF API to implement the validation.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should be able to validate a diagram in under 2 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) Validation 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.

Modification History:

Owner: Edwin Malek, Dionny Santiago, Xiaoyu Dong
Initiation date: 02/01/2015
Date last modified: 02/01/2015

Use Case ID: **CPROVME018 Save Model**

Details:

Actor: Cloud Administrator (Main user)

Pre-conditions:

1. The graphical visualization environment must be running on the system.

Description:

1. Trigger: the use case begins when the user clicks on a File button located on the top left hand side of the workspace.
2. The user selects the "Save" button.
3. The system shows a brief progress bar at the bottom of the screen depicting the saving process.
4. The system displays a message to the user that the current model has been saved.

Post-conditions:

1. The System loads the Model and displays it on the workspace.

Alternative Courses of Action:

1. In step D.2 (step 2 of Descriptions section) the user can choose to save a Model by selecting the model and pressing "CTRL-S".

Decision Support

Frequency: Each and every time the main user would like to save a model.

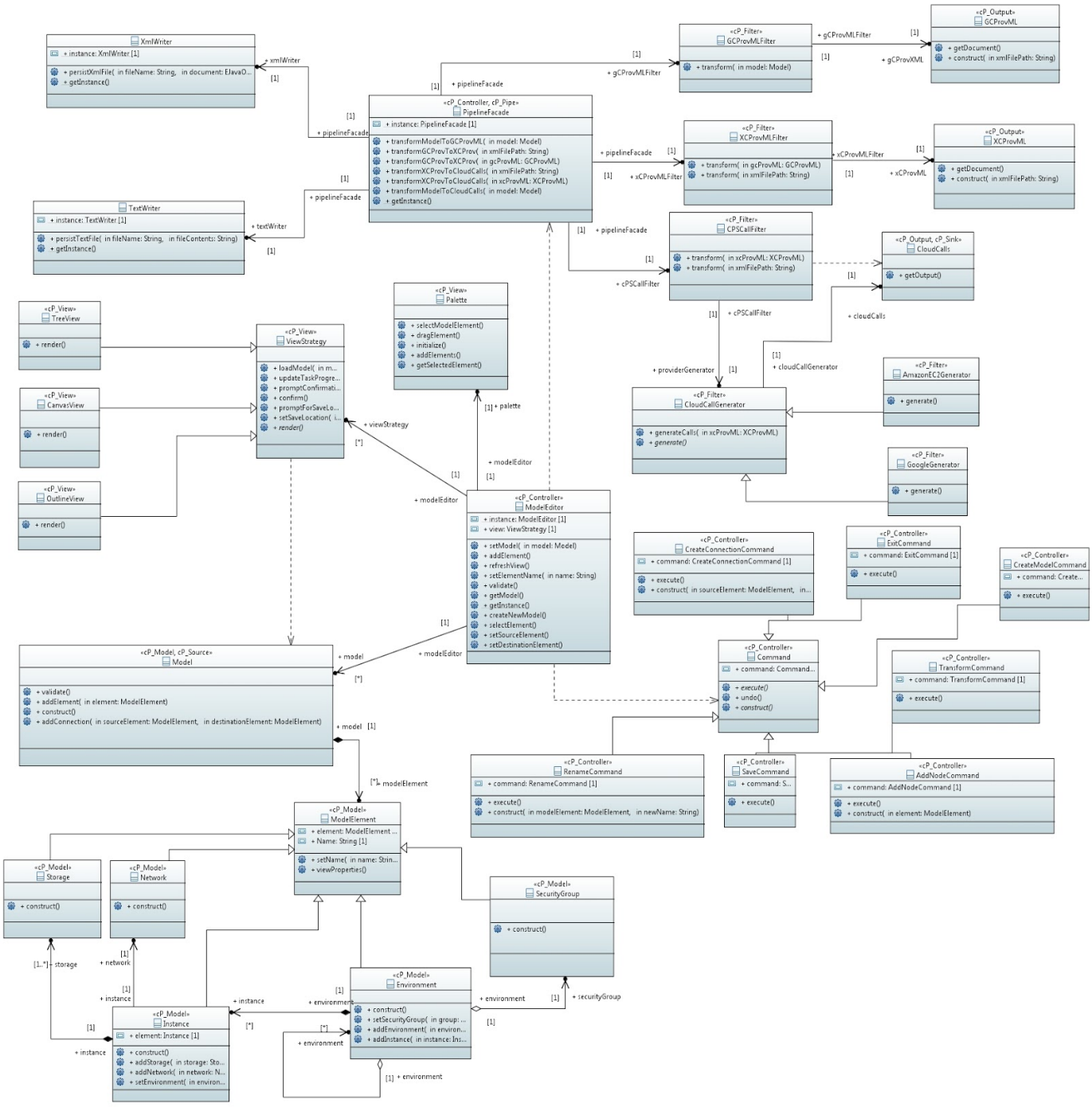
Criticality: High, It allows the user to write the current model to the memory of the system.

Risk: Low risk, the system EMF/GMF framework handles file saving.

Constraints:

1. Usability:
    a) No previous Training Time
    b) On average the user should take no more than 2 min to save a model.
2. Reliability:
    a) Mean time to Failure – 5% failures for every 1000 attempts to save a model.
    b) Availability – The system shall be available as long as the user's machine is running.
3. Performance:
    a) to save a model in general should complete in under 2 minutes.
    b) System handles only one user at time.
4. Supportability:
    a) The modeling environment should be handled by all versions of Eclipse, Windows, Mac OS, and Linux.

5.3 Appendix C - Detailed Class Diagram



5.4 Appendix D - Diary of meeting and tasks

Diary of Meetings

Meeting 1:

Place: Graduate lab

Date: 3/3/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Establishing all project architecture

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible architecture to adopt for the project

---

Meeting 2:

Place: Graduate lab

Date: 3/5/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Establishing all project architecture

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible architecture to adopt for the project

---

Meeting 3:

Place: Graduate lab

Date: 3/7/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Establishing all project architecture

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible architecture to adopt for the project

---

Meeting 4:

Place: Graduate lab

Date: 3/8/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Creation of the system's package diagram

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible architecture to adopt for the project

---

Meeting 5:

Place: Graduate lab

Date: 3/10/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Creation of the system component diagrams

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible ways to create our component diagrams for the project

---

Meeting 6:

Place: Graduate lab

Date: 3/12/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Creation of the system's minimal class diagram

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible classes to include in the diagram for the project

---

Meeting 7:

Place: Graduate lab

Date: 3/14/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Creation of the system's sequence diagrams, and state machine diagrams

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed the ordering of the calls on the sequence diagrams

---

Meeting 8:

Place: Graduate lab

Date: 3/15/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Creation of the system's Detail class diagram

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible classes to include in the diagram for the project

---

Meeting 9:

Place: Graduate lab

Date: 3/21/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Refinement previously created Detailed class diagram, addition of attributes and methods.

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

We discussed all the possible classes and methods to include in the diagram for the project

---

Meeting 10:

Place: Graduate lab

Date: 3/24/2015

Start: 7:40 PM

End: 9:00 PM

Facilitator: Edwin Malek

Attending: Dionny Santiago, Xiaoyu Dong

Minute Taker: Xiaoyu Dong

1. Objective

Design document editing and formatting.

2. Status

Everyone participated in coming up with ideas for the project.

3. Discussion

No discussions took place.