City University London

MSc in Software Engineering

Project Report

2013-2014

Integrate OpenStack with EVEREST monitoring framework.

Duy Nguyen

Supervised by: Prof. George Spanoudakis

Date of Submission: 25th September 2014.

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirement and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: **Duy Nguyen**

Acknowledgement

I would like to thank Prof. George Spanoudakis for his time and assistance throughout the whole project. I would also like to thanks to my family for their endless support and patience.

Abstract

EVEREST is an event reasoning monitoring framework that monitors security and dependability properties based on Event Calculus language. The purpose of this MSc project is to investigate on how to integrate this monitoring framework with OpenStack cloud infrastructure. A detail design will be produced for the chosen integration mechanism and will be implemented for demonstration. This project will produce a Java program that capable of providing monitoring capabilities to OpenStack using EVEREST as its core monitoring engine. This report presents the development of the project following different stage of software development process.

Keywords: EVEREST, monitoring framework, OpenStack, cloud computing, Event Calculus.

Table of Contents

1. Introduction and Objective

1.1. Background

OpenStack (Openstack.org, n.d.), founded by Rack Space Hosting and NASA, is a growing open source project with the goal to provide a cloud computing platform for public and private clouds. Many large cloud providers such as HP, Mirantis, Cloudwatt and AURO have primary deployed it as an Infrastructure-as-a-Service (IaaS) solution in their clouds. As a result, Open Stack has become one of the most popular clouds computing platform solution and stands alongside with other such as Microsoft's Azure (Azure.microsoft.com, n.d.) and Apache's CloudFoundry (Cloudfoundry.org, n.d.).

Important features in Cloud environment such as Open Stack are the ability to know the availability of physical resources, their current status, to secure access to different type of resources and to ensure any service level agreements between cloud provider and their consumer is always satisfied. These security and dependability are key properties of cloud system. Any violation to these properties can lead to a drop in availability, reliability and safety which will lead to customer's dissatisfaction and as a result, the loss of profit and business value. However, it is not easy to manage such information in a reliable and scalable way, especially in a distributed environment with highly changing context as the cloud (Spanoudakis et al., 2009, p.214). The reason for this is as operational conditions changes, the security and dependability mechanism of a system may become ineffective and needed to be replaced or adapted to ensure desired behaviour and properties. Therefore, in order to enable the ability to react to dynamically changes, the runtime monitoring of security, availability and service level agreement properties is needed.

Runtime monitoring is a computer system analysis and execution approach that obtains the information from the running system and uses this information to perform analysis in order to detect behaviours that satisfy or violate a defined set of properties. Some of the properties can be deadlock freedom, order of behaviour, customer requirement or even performance requirement. Upon detecting any violation behaviour, the system will decide how to react to certain violation such as roll back execution units or report the logging of these problems to the client that require monitored result. The ability to detect violation and react to such an event make runtime monitoring very important since it enable the system to preserve its safety and dependability.

In 2009, Spanoudakis, Kloukinas and Mahbub of City University have developed a Runtime Monitoring Framework called EVEREST (EVEnt Rea-Soning Toolkit) which is deployed as a monitoring service to SERENITY Runtime Framework. EVEREST acts as a monitor that undertakes responsibility for checking conditions regarding the runtime operation of a system. By monitoring rules expressed in EC-Assertion, a XML presentation language based on Event Calculus, this Runtime Monitoring Framework can detect any violation of the monitoring rules and provides other capability such as *"deduce information about the state of the systems, detect potential violations of monitoring rules and perform diagnostic analysis in order to identify whether the events causing a violation are genuine or the result of a system fault or attack"* (Spanoudakis et al., 2009, p.215).

1.2. Objective

The overall aim was the integration of EVEREST with OpenStack to provide a comprehensive monitoring service for Infrastructure as a Service layer and in order to realize this overall aim, the objectives listed below should be achieved:

- **Objective 1:** To deploy and investigate how OpenStack works and how it could be integrated with EVEREST.
- **Objective 2:** To produce a comprehensive design of the chosen integration of OpenStack with EVEREST.
- **Objective 3:** To implement and demonstrate the chosen design using any programming language of choice.
- **Objective 4:** To perform testing on the integrated system to ensure the quality of the produced software.

In order to achieve these objectives, an integration program will be developed. This integation program is called OpenStackEverestIntegrationModule (OEIM). The program will use EVEREST as its core monitor engine, and receive monitoring rules from users, capture all the required information from Openstack and deliver the acquired rules and information to EVEREST for processing. This program will run as a separate process outside of Openstack and provide monitoring service for OpenStack Cloud Provider/Customer, who wants to monitor their cloud Infrastructure.

The objectives listed above will be tested by the following means to determine if it is success:

- **Objective 1:** this objective is considered to be success if this project can deploy OpenStack successfully. OpenStack's architecture is examined and researched in details and analysis on various methods to integrate EVEREST with OpenStack is shown and discussed.
- **Objective 2:** this project should come up with an adequate design class and sequence diagram of the chosen integration of OpenStack with EVEREST
- **Objective 3:** the produced software can prove that it can monitor OpenStack using EVEREST.
- **Objective 4:** the produced software is tested thoroughly and passes all the testes performed.

This project is considered to be successful if it can meet all the objectives specified above.

1.3. Reason for choice

This project is of my interest and helps me to improve my skill. Cloud computing is now one of the most required skills and technologies, by doing study and research on OpenStack cloud; I can significantly improve my knowledge and skill in this field. Runtime Monitoring framework like EVEREST is also catch my attention, by diving into the design and algorithm used by this

framework, I have gain more experience and knowledge about how to provide an intelligent, reliable and dynamic way to monitor a system.

From a more practical perspective, this project is a good choice, because my supervisor, Professor George Spanoudakis and his assistant Khaled Mahbub, are the creators of EVEREST. They have welcome my interest and provided me with a great deal of assistance and feedback.

1.4. Beneficiaries

The immediate beneficiary of this project would be the Open stack community as the project will make a direct contribution to OpenStack. Other cloud user and developer communities may benefit from the project in the future. As new monitor service provided, open source community would be benefit from a ready to use software which has been built. Cloud provider and administrator will have more choice when finding monitoring solution to their services.

For projects that have already used EVEREST as its monitoring framework such as CUMULUS, an EU project whose objective is to certify security properties of cloud services based on monitoring and testing evidence dynamically acquired during the operation of cloud services, can used our software framework and code to give it access to OpenStack cloud.

As a researcher, this project provides opportunity for me to learn about cloud computing technologies and enhance my designing and programming skill.

1.5. Scope

The scope of this project has been refined to be more specific compare to the original objective in the proposal. In the original objective, we intend to integrate Everest with all layers of OpenStack, from Infrastructure, to Platform and Software as a Service layer. However, due to the time constraints, the project has been refined to only provide monitor service for Infrastructure Layer of OpenStack.

EVEREST will be integrated with OpenStack in a loose way, instead of becoming a component of OpenStack bundle like Ceilometer or Heat. EVEREST will be the core monitor engine used by OEIM program. When cloud administrators want to provide monitor capabilities to their OpenStack cloud, they can run this program to monitor their cloud.

This project will not discuss how to deal with violation reported by OpenStack. This responsibility belongs to the owner of the monitored system. Upon detecting a violation, it will be up to the cloud administrator to decide what has to be done, i.e. report to the customer, or has automation code that deal with the violation.

This project is developed and tested on a Single Node deployment model of OpenStack (All-In-One). Futherwork and tested will need to be carried out in order to test the developed program on multi-nodes deployment model.

1.6. Intended method and work plan:

The development process followed to achieve the objective of this project is Modified Water Fall model, which is a sequential design process which includes Requirement, Design, Implementation, Verification and Maintenance. The Waterfall model is goods for products will

clearly understood requirement and work well with well understood technical tools, architecture and infrastructure which are true in our case (Munassar, Nabil and Govardhan, 2010, p.94-101).

Most of the work plan for this project is associated with each stage of the Modified Water Fall Process model:

- **Environment Setup:** Setup environment for developing and testing the project. The work includes setting up Openstack cloud as well as necessary IDE tools for writing the software.
- **Requirement and Analysis:** Perform requirement gathering for developing the project as well as literature review to gather knowledge about the current state of the field and problem. As well as finding appropriate approach to solving the objective and developing analysis class and sequence interaction diagram of the software.
- **Design:** Designing an architectural style for the system, and review the analysis class/sequence diagram to draw out the necessary subsystem and component for the software. The final result will be a detail design class and sequence interaction diagram of the software.
- **Implementation:** Actual implementation of the project. If there are any problem with the current design, revise the design of the system and make any necessary changes. The final result will be a complete working program.
- **Testing:** Writing unit test case, integration test as well as performance tests. Some monitor rules will be defined to be used in testing.
- **Maintenance:** Fixing bugs and refactoring the code if needed.
- **Report:** Based on document created at each stage, complete the final report.

1.7. Major Changes

One of the major changes is the change in goal of the project. The objective has been refined to be more specific compared to the original objective. Instead of providing monitoring service to all layer of Openstack, the new project objective is only providing monitoring service to the Infrastructure Layer. The reason for this change is because of the time constraint, for each layer of OpenStack, different mechanism and methods need to be researched and applied. Therefore, three months period is not enough for investigating and implementing the integration on all three layers of OpenStack. However, despite the changes, the final objectives of this project still maintain a significant complexity for the project.

Other changes, is the way how EVEREST is integrated with OpenStack. Instead of being a part of OpenStack bundle, EVEREST now becomes a part of OEIM program which provide a bridge between EVEREST and OpenStack. OEIM will capture all the required information from OpenStack and feed them to EVEREST.

1.8. Report Structures

This project report consists of six chapters:

- **Chapter 1:** this chapter set the scene of the project. It contains background information, objective of the project as well as success criteria which will be used to verify the outcome of the project.

- **Chapter 2:** this chapter deal with literature review conducted for this project. It involves studying the architecture of OpenStack, investigating any other monitoring service available, as well as finding out how to capture information from Openstack. It also includes studying EVEREST framework, finding out how to use Event Calculus to create EC-Assertion template for creating monitoring rules.

- **Chapter 3:** this chapter explain different methods apply at different stages of the project as well as their strength and limitation.

- **Chapter 4:** this chapter provides all the result at different stages, and verify them.

- **Chapter 5:** this chapter evaluate the result obtain and compare them with original objective, what has been met, what has gone wrong.

- **Chapter 6:** this chapter will discuss what has been achieved by the project, an overview of the project progress, learning outcomes and any further works.

2. Context

2.1. Overview

In the beginning of this chapter, we will discuss about the current state of OpenStack and EVEREST, what has been developed, and what will be developed in the future. Secondly, we will research about the architecture and different components of OpenStack to gain better understanding of the cloud environment; we will work with and find if there are any supports provided by any components in OpenStack that can help us to gather information about its underlying infrastructure. Since EVERST monitor rule based on EC-Assertion language, we will need to understand how this language work as well as its foundation the Event Calculus logical language. Thirdly, we will do researching on the EVEREST itself and diving deep into its coding implementation, by examining the research paper and code, we will gain in depth understanding about the tool that will be used. After that, we will take a look at some existing project that uses EVEREST as its monitoring engine, to understand how to use EVEREST as our core monitoring engine. Lastly, we will take a look at any existing monitoring solutions for OpenStack such as its buit-in monitoring module Ceilometer or RedHat's Nagios and compare them with EVEREST.

2.2. Current State

OpenStack first start as a join-in project between RackSpace and NASA to be an open-source cloud-software platform in July 2010(Wikipedia, 2014). From then it has been growing rapidly and quickly picked up by many companies and organization*."More than 200 companies joined the project, among which are AMD, Brocade Communications Systems, Canonical, Cisco, Dell, EMC, Ericsson, Groupe Bull, HP, IBM, Inktank, Intel, NEC, Rackspace Hosting, Red Hat, SUSE Linux, VMware, and Yahoo"* (OpenStack Training Guide, 2014, p.7). From the first official release, code-name Austin, which contain only 2 primary modules Nova and Swift, OpenStack

has reach its ninth official release by now. The latest release, which is called Icehouse, contains more than nine primary modules such as Nova, Horizon, Keystone, etc. Furthermore, a new release with code name Juno has already been under development, some interesting services will be incorporated in this new release such as Zaquar, a multi-tenant cloud messaging service for web developer which enable developers to send messages between different component of their Software-as-a-Service and mobile location (Wiki.openstack.org, n.d.), and Sahara, a data-processing service module that *"provide simple means to provisioning a Hadoop cluster on top of OpenStack"* (Wiki.openstack.org, n.d.).

As stated in the beginning of this chapter, there is a built-in monitoring module integrated as part of OpenStack bundle. Called by code name Ceilometer, this module provides the user with a set of function to collects metering data and provides alert for other components. However, there is some limitation of this module that we will discuss later in section 2.6.1.

EVEREST has been used as the monitoring framework for the runtime platform of the EU E7 project Serenity (Lorenzoli and Spanoudakis, 2009, p.312) and one of the low levels monitoring engine for SLA@SOI framework. We will take a deeper look at how EVEREST has been used in these two projects in section 2.5 to understand how to integrate Everest as part of the monitoring framework for a runtime system. By doing so, we can grab a better idea in how to design our integrated program with EVEREST.

2.3. OpenStack

OpenStack is an open-source cloud operating system that manages a large pool of resources such as computes, storages and networking. Recently, it also provides users with a dashboard which help users manage their cloud effectively through web interface. With Openstack, the users can have access to a shared collection of resources, they can gain access to more resources whenever there is a demand for expansion, or they can release unnecessary resources when there is reduction in usage. Hence, the system can scale up or down effectively depend on business demand.

There are 5 major features of OpenStack cloud (OpenStack Training Guide, 2014, p.2):

- **On-demand-self-service:** cloud users can manage their compute resource such as instances, storages or networks without the need to contact cloud service provider. The users can access their service through dashboard web interface or through command line tool to increase or decrease their resources easily. Even more, with the help of Heat module, cloud users can even configure their compute resources to scale up or down automatically based on usage condition.

- **Network Access:** users can control their resources over the network through the use of Restful Web Service, launching server instances, creating images, deleting volumes can be done by using OpenStack API.

- **Resource Pooling:** computing resources can be acquired or released easily by issuing simple command. OpenStack's components such as Nova, Neutron, etc. manage their own their resource pools and provide them to the user on request. Many different users can share the same resource pools. With this mechanism, computing resources can be used effectively, once they are not used, they will be returned to the pools or issued to

another users who are in need. As a result, there will not be any case, where one user is end up with too many resources while the others are in short of computing power.

- **Elasticity:** the system based on the cloud can scale up or down easily, computing resources can be released or acquired based on demand.

- **Metered or measured service:** the cloud can provide user with statistic and diagnosis on compute resources such as number of read requests per device, number of incoming bytes on VM network interface, etc. With this information, cloud users will have a better view of their resource usage; cloud provider can provide billing fee based on the users's usage. This feature is relevant to our project because it provide us with the ability to capture and monitor information from OpenStack.

With a cloud computing platform like OpenStack, developers with innovative ideas no longer need to concern about the underlying hardware infrastructure. Everything from software, platforms and infrastructure are sold as a service remotely through internet. Cloud computing providers provide their services according to three basic fundamental models:

- **Infrastructure as a Service (IaaS):** computers or virtual machines and other resources such as file-based storage, firewalls, load balancers, etc. are provided (Wikipedia, 2014).

- **Platform as a Service (PaaS):** computing platform such as operating systems, programming language execution environment and databases, etc. are provided

- **Software as a Service (SaaS):** applications are delivered as a services over the internet while the hardware and system software located in data centres (Armbrust et al., 2010, p.50)

2.3.1. OpenStack Architecture

The objective of our project is providing monitor capabilities to only IaaS layer. Therefore, we only focus on core components that form the IaaS layer of OpenStack in this section. There are currently 7 primary modules which work together to provide a complete IaaS. They can be integrated together using public application programming interface that provided by each service. Below is the list of different modules:

- **Dashboard (Horizon):** provides a Django based web application for user to manage their services (Wikipedia, 2014).

- **Compute (Nova):** is written in Python and is the main part of OpenStack, it is designed is to manage pooling of computer resources and work with many virtualization technologies such as KVM or Xen to provide virtual machine to cloud users (Wikipedia, 2014).

- **Network (Neutron):** provide virtual networking capabilities to Nova. With Neutron, users can develop their own networking models for their system (Wikipedia, 2014).

- **Identity (KeyStone):** provide authentication to all other services in the cloud. Cloud components such as Nova, Neutron will use Identity to authenticate users before performing their request. Identity supports authentication through username and

password credentials or token-based and Amazon Web Service style login (Wikipedia, 2014).

- **Object Storage (Swift):** provide user with a high availability, distributed consistent storage system. It take care of performing data replication across a storage cluster, thus ensuring data is always available to users. Cloud users can use it to store data efficiently and safely (Wikipedia, 2014).
- **Block Storage (Cinder):** "provides persistent block-level storage devices for use with OpenStack compute instances" (Wikipedia, 2014).
- **Image (Glance):** it can store disk and server images in Swift (Wikipedia, 2014).

The diagram below show the relationship between different services:

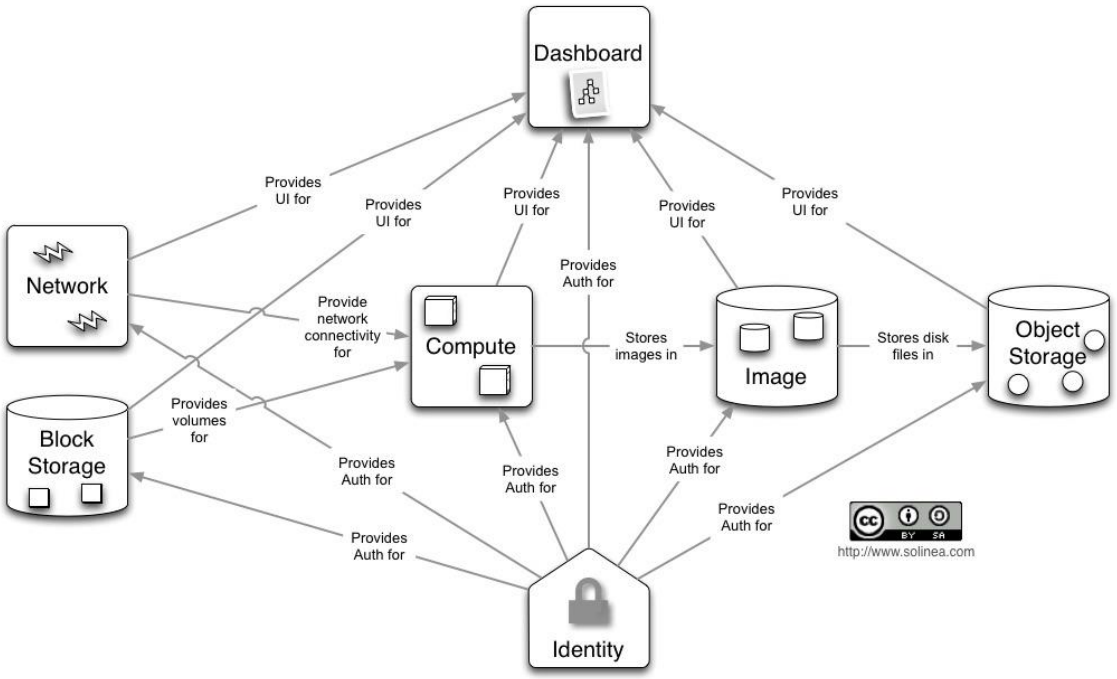


FIGURE 1.CONCEPTUAL DIAGRAM (OPENSTACK TRAINING GUIDE, 2014, P.23)

End User can use API provided by each module to use different services provides by the Infrastructure. For instance, in order to create new instance, user can create a HTTP/Rest POST request

/v2/{tenant_id}/servers

with the information about tenant_id, security_groups and ImageRef, etc. in the request body (Developer.openstack.org, 2014). Or information about each server can be queried by perform a simple HTTP GET request

/v2/{tenant_id}/servers/{server_id}

when receiving this request, Nova will check their server status and response with a JSon response to the caller with the information about the server status, their created time, their host id and many other relevant information (Developer.openstack.org, 2014). Below is a diagram showing the API offered by OpenStack:

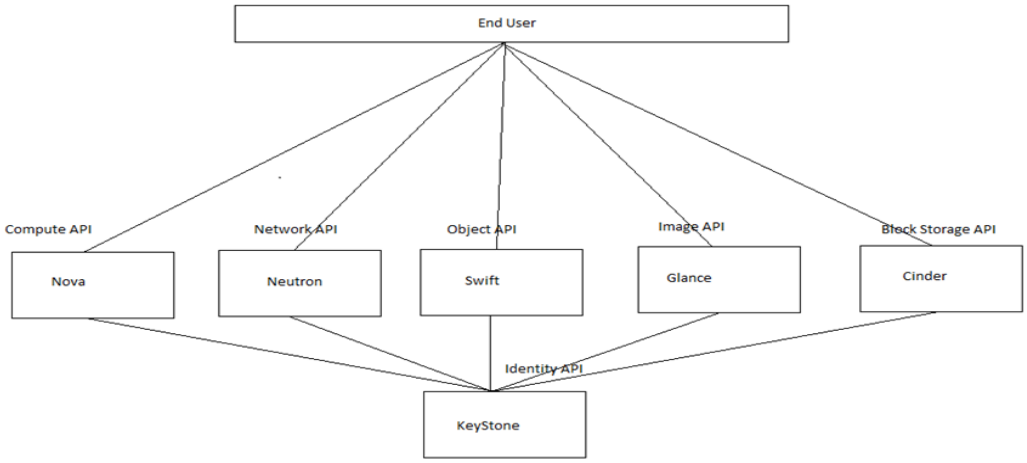


FIGURE 2.OPENSTACK API

These API inform us of a way to obtain information from OpenStack Infrastructure. Our intended software can make a call to these API to query for the required information to provide EVEREST with enough events to process for diagnosing violation.

2.3.2. OpenStack Ceilometer

The Ceilometer project is created to provide Meter and Monitored Service, so user can collect any information needed regarding OpenStack. There are some ways that Ceilometer uses to collect data from OpenStack (Docs.openstack.org, 2014):

- **Bus Listener:** Ceilometer listens to the Olso notification bus that collects notification from different service such as Nova, Cinder, etc. Those notifications then can be transform into metered and samples data.
- **Polling:** this is not a preferred way for collecting data; Ceilometer will poll OpenStack API by interval to obtain needed information.

The figure 3 below showing the way how data is collected by Ceilometer:

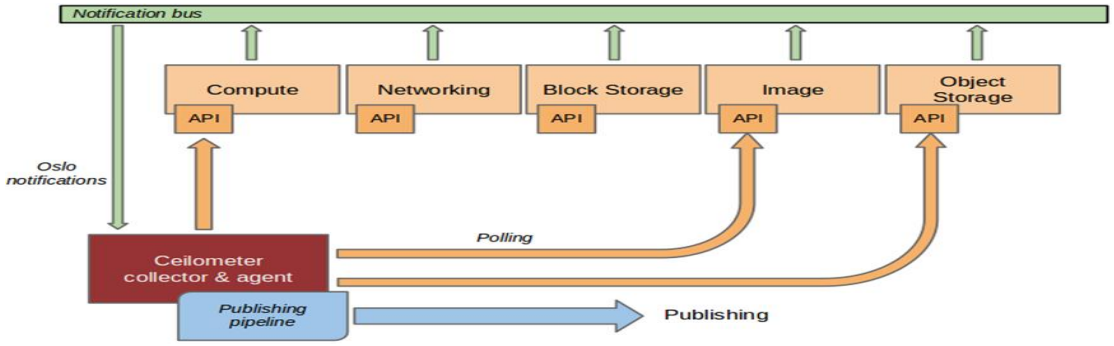


FIGURE 3.CEILOMETER DATA COLLECTION (DOCS.OPENSTACK.ORG, 2014)

As you can see, components such as Compute (Nova), Image (Cinder), Networking (Neutron) emit notification to the notification bus, and Ceilometer will listen to the message bus and capture those notifications for data collection.

This architecture made us think of an idea, instead of depends on Ceilometer for data collection, is there any way to listen directly to the notification bus and capture the event ourselves. This can give our software a big advantages of not depending on Ceilometer and it's API and more flexibility in term of data collection since Ceilometer only offer a limited set of operations for limited info. Furthermore, we can register to listen to only the component that the user wants to monitor thus filter out unnecessary information. In the next section 2.3.3, we will research about Notification Bus to know what information we can capture and how to register to the Notification Bus.

2.3.3. Notification Bus

AMQP is the messaging technology used by OpenStack cloud, there can be different AMQP brokers used such as RabbitMQ, ZeroMQ. Whenever, there are important event occurs, notification messages will be put to the message bus. Therefore, if we can register to the topics, exchanges published by the brokers, we can receive events from the notification bus. This can be done by obtain the host, port address that the message broker is running as well as names of topics and exchanges for each OpenStack components. These information can is really straight forward, because those are the configure detail that we will use when setting up our cloud.

However, not all OpenStack components publish their events to the notification bus, such as Swift. The table below showing the state of notification support in OpenStack (Sandywalsh.com, 2013):

Project	Oslo Adoption	CrUD	.start/.end	Usage Audit	.info & .error
Compute "Nova"	Yes	Yes	Yes	Yes	Yes
Object Storage "Swift"	No	No	No	No	No
Image Service "Glance" ***	No (uses custom notifier)	No	No	Partial	Yes (and .warn)
Identity "Keystone"	Yes	Yes (on User and Project objects)	No	No**	.info
Networking "Neutron"	Yes	Yes	Yes	Yes	Yes
Block Storage "Cinder"	Yes	Yes	No	Yes	Yes
Metering/ Monitoring "Ceilometer" *	Yes	No	No	No**	No
Orchestration "Heat" *	Yes	No	No	No	No
Database Service "Trove"	Yes	Yes	No	.exists	.info
Queue Service "Marconi" *	No	No	No	No	No

FIGURE 4.NOTIFICATION SUPPORT (SANDYWALSH.COM, 2013)

As you can see, notification support in OpenStack is strong, and we can capture information for important modules such as Nova, Neutron, Cinder, etc.

2.3.4. Critical Remarks

In conclusion, by doing literature review on OpenStack, we manage to figure out two way to obtain information from the Infrastructure Layer. The former is through the use of API provided by each component, and the latter is by registering to the notification bus on OpenStack. Finally, with the information capture, we can feed them to Everest for processing.

One of the problems working with OpenStack is lack of official documentation. It's hard to find detail documents about how to set up and deploy an OpenStack cloud as well as information about other less important services such as the notification bus's details. Another problem is the number of features and sub-projects developed by contributors from the community. Since it is an open-source project, it may attract bad contributions. Hence, the quality of many features provided by OpenStack community is not consistent.

2.4. EVEREST monitoring framework

2.4.1. Event Calculus logical language

In order for EVEREST to monitor particular properties, users need to let it know what they want to monitor and how to know if the properties are violated. This information is called rule and specified using an XML based language, called EC-Assertion. Since EC-Assertion is based on event calculus logical language, a basic understand about this language is need.

"The event calculus was introduced by Kowlski and Sergot as a logic programming formalism for representing events and their effects"(Shanahan, 1999, p.410). In the Event Calculus, we can specify what is the effect of a certain action or what action has happen at a specified time as well as the outcome of the combination of the action's effect and actions, i.e. what is true when the combination happen. As Shanahan explained, *"event calculus is a logical mechanism that infers what's true when given what happens when and what actions do"* (Shanahan, 1999, p.410). The figures below showing how Event Calculus functions:

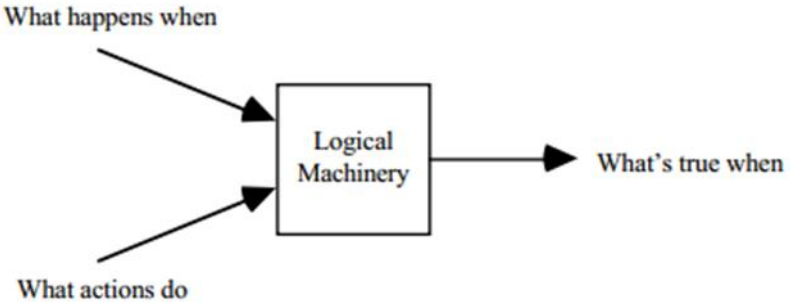


FIGURE 4.HOW THE EVENT CALCULUS FUNCTIONS (SHANAHAN, 1999, P.410)

The "What happens when" part is a list of actions happen at a specified time, and the "What actions do" specify the effect of each action. Finally, the "What's true when" part is the conclusion outcome of the combination of "What happens" and "What actions" parts. One simple example is given that "What action do": raining make road wet and "What happens

when": it's rain at 7:00 AM, we can come to a conclusion "What's true when": the road is wet at 7:00 AM.

With Event Calculus, a number of reasoning tasks can be conducted. These can be categorized into three main types of tasks (Shanahan, 1999, p.410):

- **Deductive:** in deductive task, knowledge about "What happens when" and "What actions do" is given; outcome of the actions are sought.
- **Abductive:** in abductive task, "What's true when" and "What actions do" is given, what action has taken place during the specified time need to be found out.
- **Inductive:** in inductive task, "What's true when" and "What happens when" is given, the effect of the actions is sought.

Based on these reasoning tasks, monitor rule for our system can be specified. For example, for a monitor rule that state "when perform reboot on an instance, the instance status will be Reboot". Whenever, there is an action reboot at t times ("What happens" , and the action change the status of the instance to reboot ("What action do"), deductive task can be performed to conclude that the instance status will be Reboot at t time, if the instance status is not reboot, then the rule is violated.

In the EVEREST, "What happen" and "What action" is specified in term of events and fluents An event is something happen at a specific time and fluents are state of the system which are affected by events. An event can initiate or terminate a fluent (Spanoudakis, n.d.). There are a set of predicates to describe what happens when or describe effect of action or initiate state of a system, etc.:

- **Happens (e,t,R(t1,t2):** this predicate used to represent the occurrence of an event e at some time t in between t1 and t2 (Spanousdakis, Kloukinas, Mahbub, 2009,p.219).
- **Initiates (e,f,t):** this predicate represents that fluent f starts to hold after event e occurs at time t (Spanousdakis, Kloukinas, Mahbub, 2009,p.219).
- **Terminates (e,f,t):** this predicate represents that fluent f stops to hold after event e occurs at time t (Spanousdakis, Kloukinas, Mahbub, 2009,p.219).
- **Initially (f):** this predicate signifies that fluent f holds at the start of a system (Spanousdakis, Kloukinas, Mahbub, 2009, p.219).
- **HoldAt(f,t):** this predicate signifies that fluent f start to hold at time t.

With the predicates above, Event Calculus rule for the reboot rules can be specified as follow:

Assumption:

Happens(reboot_event, t1, R (t1,t1) ^ HoldAt(status,t1) => Terminates(reboot_event, status, t1) ^ Initiates(reboot_event, status = Reboot, t1)

Rule:

Happens(reboot_event,t1,R(t1,t1)) ^ Happens(check_status,t2,R(t1,t2)) ^ Not Happens(active_event,t3,R(t1,t1 + t2)) => HoldAt(status,t2) ^ Relation(status == 'Reboot').

There are two parts in this monitor rules, the assumption and the rule. In the assumption, the effect of the action taken is specified, when the reboot_event action takes place, the state of the system will be 'Reboot'. In the rule section, what happen when and what's true are specified. In this case, when there are action reboot_event and action check_status action and there are no action active_event taken between t1 and t2, then the system status will be checked, if the status is 'Reboot', the rules is satisfied, if not the rule is violated.

In conclusion, using Event Calculus, many complex rules can be set based on the sequence of events taken. It is especially useful since events from OpenStack can be captured, and provided to EVEREST for reasoning.

2.4.2. EC-Assertion XML based language

EC-Assertion is an XML language based on Event Calculus; it is developed to be used as a language for specifying monitor rule/formula for EVEREST. The representation of Event Calculus's predicate is written in XML format, a machine-friendly format. In this report, some of the example of EC-Assertion will be provided for understanding and testing.

In EC-Assertion, an event can be invocation or response of system operation or messages exchanges between different components or notification events in our case. The following event structure is used to represent event in EC-Assertion (Spanousdakis, Kloukinas, Mahbub, 2009, p.220):

event (_id, _sender, _receiver, _status, _sig, _source)

In this structure, *_id* is the unique identifier of the event, *_sender* is the component that performs the invocation/response, *_receiver* is the component that receives the invocation/response. While the *_status* is the status of an event, REQ if it is an invocation or RES if it is a response), *_sig* represent the signature of the invocation/response which include operation name and it's argument or result. Lastly *_source* is the component that captures the event (Spanousdakis, Kloukinas, Mahbub, 2009, p.220). Since EC-Assertion has specified structure for the event, we will need to convert our captured event from OpenStack into this format also.

In the EC-Assertion, the rule has two main part head and body. If the body is evaluated to true then the head must also be evaluate to true, or else the rule is violated (Spanousdakis, Kloukinas, Mahbub, 2009, p.220). In our example above *Happens(reboot_event,t1,R(t1,t1)) ^ Happens(check_status,t2,R(t1,t2)) ^ Not Happens(active_event,t3,R(t1,t1 + t2))* is the body part, and *HoldAt(status,t2) ^ Relation(status == 'Reboot')* is the head part. In EVEREST, when the body is satisfies it will check the head of the rule to determine if the rule is satisfied or not.

In conclusion, knowledge of EC-Assertion is necessary if we want to monitor using EVEREST. The xml schema of EC-Assertion is provided along with this report. In addition to this, any events captured from OpenStack need to be converted to EC-Assertion format for compatible with EVEREST.

2.4.3. EVEREST

EVEREST is a monitoring framework which offer interfaces for submitting rules to it or for checking, receiving events from the monitored applications and querying monitoring result. It consists of three main components: a monitor manager, a monitor and an event collector (Spanousdakis, Kloukinas, Mahbub, 2009, p.217).

The monitor manager main responsibilities are receiving monitoring rules and providing API for obtaining result. The event collector takes charge of receiving events from the monitored application and passing them to the monitor manager. Lastly, the monitor will check if the received events violate any of the rules given to it (Spanousdakis, Kloukinas, Mahbub, 2009, p.217).

In our case, we need to implement the event capturer to capture event from OpenStack and convert them to EC-Assertion event and feed to EVEREST.

When receiving monitor rules, EVEREST will convert those the rules into template objects and maintain them to work with events sent to it at runtime. Whenever there is an event e occur, EVEREST will find the templates that have predicates that can be unified with the event e and update them. When there is an event that can be unified with a template t, it means that an event that can affect rule t has happen and rule t has to be checked. Once all the predicate in the body of the template has been unified and is true, EVEREST will proceed with processing the head of the template, if at least one predicate in the head is false then the rule represented by the template is violated (Spanousdakis, Kloukinas, Mahbub, 2009, p.230).

1. SLA@SOI

SLA@SOI framework provides libraries and tools, utilities to implement Service Level Agreement (SLA) aware service. This framework needs to have its SLA guarantee terms monitored and reported to the SLA@SOI framework.

This framework use EVEREST as one of its low level monitoring engine. However, since the SLA guarantee terms are specifies in a language that is different from the one that the engines use to express operational monitoring specification. A wrapper for the monitoring engine needs to be implemented. This wrapper will translate the SLA expressed language into the language that is supported (EC-Assertion) by the monitor. The implementation of the EVEREST wrapper in SLA@SOI framework is EVEREST RCG (Khaled, Geogre and Theocharis, 2011, p.80)

The SLA event is also translated to the LogEvent object, which represent the event format of EC-Assertion before submitting to EVEREST for reasoning.

2. SERENITY

SERENITY framework enabling the abilities to configure, deploy and select dynamically the components that realise the safety and dependability mechanism. EVEREST is used as a monitoring engine for monitor the runtime operation of those components.

In this framework, the SERENITY Runtime Framework (SRF) will submit monitoring rules to the EVEREST by interface exposed by the monitor manager. An Event Capturer outside of EVEREST will intercept events during the operation of monitored application and send them to the SRF, which later forward them to EVEREST's event receiver. The Event Capturer is part of the monitored application.

2.4.4. Critical Remark

In conclusion, in order to develop a program that monitor OpenStack and using EVEREST as its core monitoring framework, the following components need to be implemented: an EventCapturer to capture event from OpenStack and an EventConverter to convert OpenStack

event to EVEREST compatible event. The event captured by EventCapturer will be converting to EVEREST event using EventConverter before processing.

Although EVEREST is a good monitoring framework that takes advantages the strength of Event Calculus language, it requires developers to understand and master Event Calculus and EC-Assertion to use it effectively. Adept knowledge of Event Calculus is required in order to write complex monitor rules for EVEREST.

2.5. Existing Monitoring Solutions in OpenStack

In this section, two existing monitoring solution for OpenStack is examined and compared with EVEREST.

2.5.1. Ceilometer

Ceilometer (Wiki.openstack.org, 2014) is an integrated OpenStack project, its capabilities is collecting data and providing alerts for Nova, Neutron and Cinder. The data collected is mainly used for billing; the metering service can be accessed through the Telemetry API. Although, Ceilometer provides users with sufficient information about each service, in term of monitoring it is not flexible enough. For instance, in order to monitor a specific property of the system, user will have to develop a script to specify monitoring logic to check for violation. Furthermore, each time a new monitor property is added, the script need to be modified to add more monitor logic in the code, thus make the monitor module hard to scale. On the other hand, with EVEREST, users only need to specify their monitoring rule and EVEREST will automatically provision the additional rule without any modifying in code. When users want to add a new rule to the monitor module, they simply add the rule to the EC-Assertion rule file, and EVEREST will automatically monitor the rule. As a result, it is very easy to add new rule to monitor new properties and make the monitor module scale easier.

On the other hand, although EVEREST is used as our monitor framework, Ceilometer's Telemetry API can still be useful. For instance, Ceilometer's API can be inovked to obtain data from OpenStack if the data is not available from the notification bus. Thus give us more choice about how to obtain information

2.5.2. Nagios

"Nagios is a powerful monitoring system that enables organizations to identify and resolve IT Infrastructure problem before they affect critical business process" (Nagios.org, 2014). It can be configured to generate alert if there are any issues detected. Nagios has been widely known as a monitoring solution for OpenStack. In fact, in the Redhat Enterprise Linux OpenStack Platform guide, Nagios is used as a guide for Monitoring section.

By default, Nagios comes with a set of basic local statistic such as swap usage, number of current user, etc. In order to add a new OpenStack service to the monitored services, users have to create a command script file to specify what and how they want to monitor, and then the command script file will need to be registered in the command list. Finally, users can define the host they want to monitor using the command script. There are also many built in commands in Nagios that can be used for monitoring services.

The command script can be written in any language from C to Shell, Perl and Python thus monitoring logic can be developed easily. In addition to this, the command script is reusable. In opposite, EVEREST requires users to fully understand Event Calculus and EC-Assertion language. However, with Event Calculus, monitoring rule can be expressed easier when the rule related to the time constraint between events. Furthermore, in Event Calculus, actions and their effects as well as the expected outcome can be described more naturally and shorter. Since we can make use of OpenStack notification bus, it is better to listen to event for information instead of execute command to get information as Nagios. Another reason for choosing EVEREST is to provide community with more choice regarding monitoring solutions.

2.6. Concluding Remarks

The literature review section has enabled the development of a better understanding about architecture of OpenStack and EVEREST as well as the Event Calculus and the EC-Assertion language that is used by EVEREST to specify monitoring rules. In addition it has helped us identify 2 approaches that are available for obtaining information from OpenStack, i.e., first is by listening to notification bus and second by polling OpenStack API to request information. As a result, we know that there are at least 2 components that need to be developed in order to use EVEREST: the EventCapturer and the Poller (API call component). Consequently, the development effort required for the integration of EVEREST with OpenStack focused on these two components.

3. Methods

3.1. Overview

This chapter outlines the development process that was used to implement the project, the main phases of it, and the methods and tools used in each of these phases.

3.2. Development Process

After carefully investigate different type of software process model, Modified Waterfall Process model is chosen as the process model for the project.

Modified Waterfall Process is a modified version of its popular predecessor the Waterfall model. The Waterfall model is goods for products will clearly understood requirement and work well with well understood technical tools, architecture and infrastructure (Munassar, Nabil and Govardhan, 2010, p.94-101). However, it is really hard in practice because it is nearly impossible for any big project to finish a phase of software products development perfectly before moving to next phases. Once the project finishes the requirement phase, they have to move next to the design phase and cannot return back to the requirement phase to make any changes in requirement. If there are any problems found during a stage of a process and require changes in any previous phase, the whole development process model has to be start from beginning again. As a result, this process model is nearly impossible in real practise, since customer changes occur frequently, and changes in each stage of the process is also happen frequently. However, since our project objective is clear, the technical tools and project architecture is quiet straight forward; we can benefit from the simplicity and well defined stage's objective of this type of process

model. In order to follow this process model, we had to find any other process models that inherit the advantages of this process as well as overcome the weakness of this process model.

One of successors of Waterfall Process model, Modified Waterfall Process model has been introduced to address these weaknesses. It allows designer and developer to bounce back to previous phase to make any necessary modification if needed. The diagram below illustrates the flow of Modified Waterfall Process model:

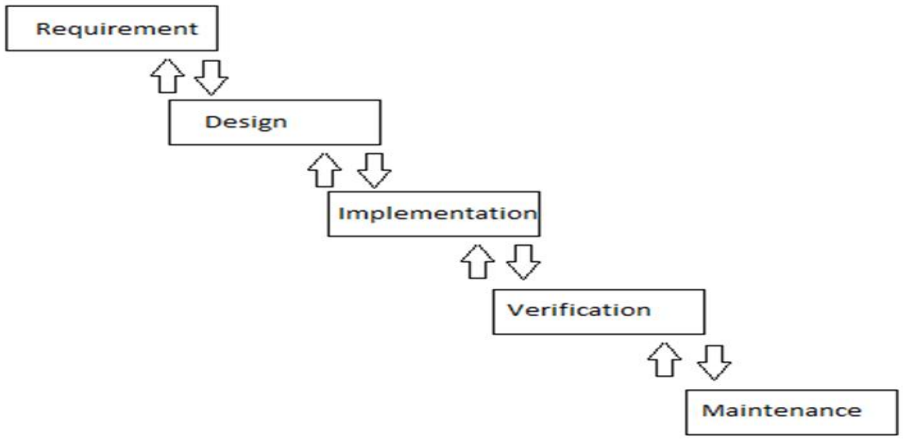


FIGURE 5.MODIFIED WATERFALL PROCESS

As you can see, if there are any problem occur during a stage of the process and require change in requirement, developer, designer can return back to the previous stage to modify the requirement. With Modified Waterfall Process model, we can benefit from the simplicity and clear stage's objective of Waterfall Process as well as the flexibility of returning to previous stage.

There are 5 stages in Modified Waterfall Process that we used in our project:

- **Requirement:** in this stage, project requirement was established, this includes functional and non-functional as well as performance requirement. A set of different use cases might be produced in this stage.

- **Design:** in this stage, we specified a design of the system which would meet the requirement. The design defines all the major components in the system and the way how they interact with each other. The result of this stage were detail class and sequence design diagram.

- **Implementation:** in implementation stage, we decided which language to use to implement the design as well as other third party library and tool. This stage involved actual coding and the result was complete functional and testable software.

- **Verification:** this stage involved performing test, to check if the software satisfies the objective's requirement. Testing can be unit test, integration test and performance test.

- **Maintenance:** this is the final stage. If there were any faults or bugs, the software would be modified to correct the faults. Other additional modifications could be performance improvement.

In conclusion, the reason for choosing Modified Waterfall Process model is because the objective of the project is clear and well understood, there is actually no involvement with customer and the project is short (3 months).

3.3. Environment Setup Phase

Although this stage is not included in the process model, it is very important to our project. Since we have to work with OpenStack, it is significantly important to set up OpenStack environment for working as well as testing. In this stage, we evaluated different tools and deployment model for OpenStack and set up the environment.

3.3.1. Hardware Resource

The only computing hardware resource that we had for this project is a K55V Asus Laptop with the following specification:

- CPU: Intel Core I5 – 3230M, 2.6 GHz.
- Memory: 8 GB RAM.
- HDD: 500 GB.
- Network: 1 built in 1GB network interface card.

Since our resource is limited, we had to find a suitable deployment model for OpenStack, such as Single Node Deployment (All-In-One) model

3.3.2. Operation System

OpenStack can be deployed on the following operating system: Ubuntu, Redhat, CentOS and Fedora. In our project, we decided to use Ubuntu 12.04 LTS as our operating system to deploy OpenStack. The reason is simply because OpenStack support this Ubuntu release and I have experience working with Ubuntu before.

3.3.3. Deployment Model

There are 2 models that you can deploy OpenStack: multi-node deployment and single-node deployment.

Multi-node:

Multi-node deployment has different OpenStack services run on different node (computer), a typical deployment of multi-node installation require at least:

- **A Controller Node:** a controller node typically host control services such as: KeyStone, Glance and Nova API, conductor and scheduler. It also hosts other supporting services such as message queue and database.
- **A Network Node:** a network node typically host network services such as Neutron or Nova-Network and responsible for managing virtual networking between instances as well as linking virtual machines to external network.

- **One or many Compute Node:** a compute node will run the virtual machine instances and networking agent if we use Neutron for networking.

Single-node:

A single-node deployment hosts all OpenStack services such as KeyStone, Glance, Nova, and Neutron in only one node.

Obviously, multi-node deployment is a better choice because different services reside on different nodes thus provides better performance and availability. However, since our hardware resources were limited, only one single computer, we could only use the single-node deployment model. One advantages is that it is simpler to setup and since the project was still in experiment it was sufficient to test the software in Openstack single-node environment.

3.3.4. Devstack

Devstack is a tool that assist user in setting up OpenStack development environment. *"Devstack's mission is to provide and maintain tools used for the installation of the central OpenStack services from source (git repository master) suitable for development and operational testing"* (Wiki.openstack.org, 2014). With the help of devstack, developers don't need to spend a lot of time to configure and deploy manually each OpenStack service. As a result, we don't have to spend a lot of time in setting up environment and can focus on developing our software. Therefore, Devstack was used for setting up development environment in our project.

In order to use Devstack, we need to have system with a fresh install of Linux which is Ubuntu 12.04 LTS in our case. There is simple configuration step before running Devstack which involve specifying our network range, fixed network size and password for different services in local.conf file. Below is an example of the configuration (Devstack.org, 2014):

[[local|localrc]]

FLOATING_RANGE=192.168.1.224/27

FIXED_RANGE=10.11.12.0/24

FIXED_NETWORK_SIZE=256

FLAT_INTERFACE=eth0

ADMIN_PASSWORD=<Admin_Password>

MYSQL_PASSWORD=<MySql_Password>

RABBIT_PASSWORD=<Rabbit_Password>

SERVICE_PASSWORD=<Service_Password>

In our OpenStack environment, we used MySQL as our database and RabbitMQ as AMQP message broker. Other configuration option can be added to further configure the OpenStack cloud.

With Devstack, we can speed up our development time and don't have to spend a lot of time setting up environment. However, Devstack is only appropriate for setting up development and testing environment. It is by no mean a choice for production deployments.

3.3.5. Notification Bus

In order to capture event from OpenStack, we configured our notification bus so services like Nova, Cinder can publish their notification messages to the bus. The following is an example on how to configure Nova service to issue notification:

```
[[post-config|$NOVA_CONF]]

[DEFAULT]

use_syslog = True

notification_driver=nova.openstack.common.notifier.rpc_notifier

nova.openstack.common.notifier.rabbit_notifier

notification_topics=notifications,monitor

notify_on_state_change=vm_and_task_state

notify_on_any_change=True

instance_usage_audit=True

instance_usage_audit_period=hour
```

2.1.1. Verification Method

The following way was used to verify our OpenStack installation:

- Once DevStack finish installing OpenStack, it will give us a link to access our Dashboard,a web interface to manage our service. If we can access the web interface and check all our services are presented then we can be sure that our installation is successful.

After verifying OpenStack installation, we need to make sure the notification bus is working as expected. A simple program was created to register to the notification bus and listen to events. When we create an instance if the program can capture the event then our notification bus is setting up correctly.

2.2. Requirement & Analysis Phase

In this stage we came up with requirement from the system based on the objectives. After that, some analysis was done to gain more understanding about the requirement as well as give an initial shape to the structure of the software. The result of the analysis was analysis class diagram and analysis object interaction diagram; these diagrams were used as input to our design and implementation decision. The analysis diagram result will be presented in Result section of the report.

During our analysis and design phase, Unified Modeling Language (UML) was used as our modelling language. UML is a very powerful modelling language which provides system designer with a way to visualize a system's architecture in a diagram. We can use UML to model the whole system without concern about the implemented language. The tool that we used to develop UML model is Microsoft Visio 2013.

2.2.1. Requirement Gathering

In this stage, the objectives of the project are divided into smaller tasks and listed as functional and non-functional requirement for each task. The result is a table that contain functional and non-functional requirement for the software. This table will be presented in the Result section of this report.

By examining the way how events and information can be captured and obtained from OpenStack, important functionalities and components were identified as in literature stage. In order for our software to work with OpenStack correctly using EVEREST, requirements for each component were specified as well as non-functional requirements to ensure software quality. Requirement gathering in this stage is mostly based on the methods that we have identified and the overall objectives of the project.

2.2.2. Analysis Class Diagram

The analysis class diagram describes the logical structure of the system based on abstract descriptions of objects and their relations. Class diagram at this stage only help us to identify what components should present in our system and the operations that is necessary to realise the functional requirement as well as the relationship between them.

The produced analysis class diagram would be provided as logical, conceptual model of the system for our design decision in Design stage, where we came up with a complete picture about all the components in the system, their properties and behaviour as well as the relationship between them.

The analysis class diagram for our project at this stage concentrate on the following issues:

- Identified the main components, entities and model them as class as well as distinguish their roles in the system (control, entity or boundary).
- Identified the relationship between the classes created.
- Specified the operation that each class must provide in order to fulfil the requirements

2.2.3. Analysis Sequence Diagram

While analysis class diagram helps us to identify the logical structure of the system, analysis sequence diagram outlines the behaviour of the system in term of object interaction to deliver the functionality for the system. It specify in general abstract terms the interaction between objects defined in analysis class which is sufficient for realising the functionality describes in the requirement. The produced analysis sequence diagram would be the input to inform design decision between object interactions in Design phase.

The analysis sequence diagram for our project at this stage only focus on the following issues:

- Identified in abstract term the interaction between objects.
- Identified any changes need in analysis class diagram based on the process of specifying interaction between objects.

2.3. Design Phase

In this section we came up with design class and sequence diagram based on the requirement and analysis diagram produced in analysis phase. These diagrams were also modelled using UML modelling language. The diagram result will be presented in Result section of this report.

2.3.1. Design Class Diagram

The objective of design class diagram is to provide details specification of classes that implement the system as well as the relationship between them. This diagram also specifies the operation and full details of their signature for each class.

The design of classes typically informed by the analysis classes produced in analysis phase. The design classes can be produced by refining of analysis classes. By examining analysis classes, a full specification of attributes and their visibility (private, public) and full details of class operations such as parameters, return type, exception and visibility will be identified. New classes can also be introduced to realize interface or to bundle implementation of system functionalities or follow a design pattern. In this stage, practices involves in implementation of the system such as programming language, available third party libraries, underlying DBMS are considered.

The design class diagrams are very important in implementation phase. Because by designing the system before hand, we can have an overall look of the future system structure and we can define any underlying problem lies in the intended design before doing actual coding. Additionally, the design class diagram act as a skeleton for our actual coding class which make implementation much faster.

The activities involved in design class diagram include the following:

- Specified design class, their full specification of attributes and operations.
- Identified relationship between design classes; include association, generalisation and association.

2.3.2. Design Sequence Diagram

The aim of design sequence diagram is to provide in details the interactions between objects that will implement different functional requirements of the system in a timely manner.

The design sequence diagrams designed at this stage were based on analysis sequence diagram and design class diagram. Design sequence diagrams were created by refining analysis sequence diagram and adding additional implementation details.

Implementation phase benefit a lot from the use of design sequence diagram. With design sequence diagram, developer can understand how system's object interact with each other, how different objects interact together to perform a specific algorithm and business logic flow is reflected clearly on design sequence diagram. By designing sequence diagram, we could predict

any underlying problem in the current design and change the design without doing code modification which is time consuming. Therefore, it is important that we came up with complete sequence diagram before proceeding to implementation.

The activities involved in design class diagram include the following:

- Specified full details implementation of interactions between object.

2.4. Implementation Phase

In the implementation section, we are going to examine different tools and libraries that used in our project.

2.4.1. Java Language

Java (Oracle.com, 2014) is an object-oriented programming language developed with the aim of "Write Once, Run Everywhere". It is designed to be strong type, concurrent and general purpose, Java's classes is compiled into an intermediate byte-code format. When a program run the compiled byte-code will be interpreted by Java Virtual Machine to execute machine instructions.

The main advantages of Java are its platform-independent characteristic, automatic allocation and garbage collection as well as strong multi-threading support. However, it is slower than compiled language like C/C++ and it is not suitable for scripting.

Java was used as our implementation language for the project. One of the reasons for this decision is because EVEREST is written in Java, therefore in order to use it effectively, the new program will be written in Java. The second reason is because Java is backed by a large choice of open source library such as Hibernate for database access, or Spring MVC for developing web application. The last reason is simply because of my proficiency with Java. Eclipse was used as our Integrated Development Environment (IDE).

2.4.2. Spring Framework

Spring Framework (Projects.spring.io, 2014) is an open-source application framework that provides developers with support for small scale application to developing enterprise, web application. Its core feature is the Inversion of Control container; the container is responsible for managing bean object life cycle from creation, attaching them to other object to destroy the bean object. The container can be configured using XML or Annotation configuration.

There are many features in Spring Framework such as Dependency Injection, Aspect Oriented Programming and web MVC framework, etc. Spring Core, Spring Context and Spring Bean library were used by our project, mainly because it's powerful dependency injection capabilities. Thus will improve the loosely couple nature between components which make it easier for testing and maintenance. In order to use spring framework, we used spring xml configuration to configure our spring setting.

2.4.3. Log4j

Logging is a very important component of software development, it help developers to debug problem effectively thus make maintenance much faster and easier. Any information of runtime system that developers want to logged will be print out to file. However, a poor written logging

component can slow down the system performance significantly; therefore we need to identify a fast and reliable logging library for our project.

Log4j (Logging.apache.org, 2014) is a fast, reliable and extensible open source logging framework that written in Java. By using external configuration file, log4j is highly configurable; the logging configuration such as logging directory, layout, etc. can be changed easily without having to modify the application binary code.

As a result, log4j was chosen as the logging component in our software. We used log4j version 1.2.16 in our project. A log4j property file was defined in order to use log4j in our project.

2.4.4. OpenStack4j

OpenStack4j (Unruh, 2014) is an open source OpenStack client that support provisioning and control of an OpenStack system. It has support for many OpenStack components such as Nova, KeyStone, Cinder, etc. By hiding all the complex work of building HTTP Request header and body to the OpenStack service as well as maintain connection session with OpenStack, the library only exposes simple Java API method for other system to use. Therefore, save us a lot of time from developing component for building HTTP Request and session management.

During researching for third party library to use with OpenStack, we also discovered jcloud (Jclouds.apache.org, 2014), Apache's open source multi-cloud framework. Although jcloud provides independent of cloud platform which can be portable between different clouds, it doesn't fully support all OpenStack service especially Ceilometer. That's the reason why we chose to use OpenStack4j instead of jcloud.

2.4.5. Maven

Maven (Porter and Zyl, 2014) is a software project management tool used primarily for Java project. Maven help developer to specify how software is built and its dependencies using based on the concept project object model (POM). The way how software is built and its dependencies on other modules are specified by an XML file with .pom extension.

Instead of downloading and managing the module dependencies yourself, Maven will automatically download Java dependency library and Maven plugin from a public central repository. All the download modules and libraries will be store locally on the host.

With Maven, we can perform automation process on build; test and release for our software and it also resolve and manage dependencies for our project. Furthermore, new developer who is not familiar with the project can instantly know how to build, test and release the project based on the configuration files. Therefore, we decided to use Maven as our software management tool in this project.

2.4.6. RabbitMQ Client

Since our notification bus's broker is RabbitMQ, we need to find a way to communicate to the bus in order to register to exchange/topic and listen to event from message queue.

After researching, we found RabbitMQ Java client library which allows Java code to communicate with RabbitMQ server. Through the use of RabbitMQ Java client, we don't need to write a specific module to handle communication with RabbitMQ server which can be a very

hard and time consuming task. Therefore, RabbitMQ Java client library was chosen to be used for building RabbitMQ client in our project.

2.5. Testing Phase

With the completion of Implementation phase, the next phase in Waterfall model is Verifying stage. In this stage, we carried out a number of testing to ensure the quality of the product. There are 4 level of test: Unit Testing, Integration Testing and Performance Testing. Each testing level focused on a specific objective. The overall goal of the testing phase was to verify the quality of the system and check if the software satisfies the requirement as well as discover any bugs and problems in the software

2.5.1. Unit Testing

The aim of unit testing is to verify the functions written is worked as expected. Unit test alone cannot verify the functionality of entire software; it is only used to check if each component in the software is working correctly.

In our unit testing, we used JUnit (Junit.org, 2014) to perform unit test. JUnit is a testing framework for the Java programming language. In order to perform unit test, each important components in our software had their associate test class that verify their functionality. The software is considered to pass Unit Testing if all the testing classes are passed.

2.5.2. Integration Testing

In integration testing, individual component was combined and tested together. In this testing, groups of components that work together to realise a specific functional requirement will be tested. It is to ensure that there are no defects in the integration between software's components.

We also used JUnit to perform integration test. For each functional that was realised by the integration of different components, we produced a test case to ensure there are nothing wrong happen in the integration. The software is considered to pass Integration Testing if all the integration testes are passed.

2.5.3. System Testing

The aim of system testing is to ensure the software as a whole meets its requirement. In this testing, we had a list of test cases to check if the software satisfies the requirement, i.e. for a specific monitor rule, the software can detect violation when the monitored properties is violated. The software is passed if all the test cases are passed

2.5.4. Performance Testing

This is additional testing that was used to ensure the non-functional requirement of the software. In this test, we determined how our system will perform under a particular workload, i.e. how long it take for the software to finish a particular tasks. The software is passed if all the performance testes are passed

2.6. Maintenance Phase

In maintenance stage, activities such as faults correction, performance improvement or software' source code quality improvement were performed.

2.6.1. Refactoring

In refactoring, section of code with bad design was identified and improved. This made our code become better overtime and avoid any future problem that we could encounter when extending the software.

2.6.2. Bug Fixings

In this step, we corrected any faults found in the testing stage. After correcting the faults, the software would be tested again to ensure the fix is effective.

3. Results

3.1. Overview

This chapter presents the results obtained from each software development stages as well as a sample output produced from the new software that we have implemented. The results will be presented in a sub section corresponding with the development phase. The first part of this chapter concerned with the requirement specification and the analysis class and sequence diagrams obtained from the analysis phase of the development process. The next section presents the design class and sequence diagrams result in the Design phase. Finally, the implementation result with sample outputs and screenshot as well as testing result will be described in the last two sections.

3.2. Requirement and Analysis Result

3.2.1. Requirement

Based on the objective of this project, the overall objective has been divided into small functional and non-functional requirement as listed in the table below:

ID	Requirement	Type	Pass Criteria
F-1	The software must capture events from OpenStack's notification bus.	Functional	Events from OpenStack's notification bus can be captured successfully by the software. This can be tested by perform action with OpenStack' resources such as creating instance. Then the software's log can be analysed to check if the captured message has been printed in the log.
F-2	The software must convert events from OpenStack's notification bus to EVEREST's compatibility	Functional	Events from OpenStack's notification bus can be converted to EVEREST's events successfully. This can be tested by perform action

	events.		with OpenStack's resources such as creating instance. The converted events will be printed out in the log. If the converted events are found in the log, the software has satisfied this requirement.
NF-1	The software must convert events from OpenStack's notification bus to EVEREST's compatibility event correctly.	Non-Functional	Events from OpenStack's notification bus can be converted to EVEREST's events correctly. This can be tested by writing a small unit test to compare the converted result with the expected result.
F-3	The software must obtain information from OpenStack by calling OpenStack API.	Functional	The software can call OpenStack API to query for information. This can be tested by checking the software log. When querying for information, the call will be printed in the log.
F-4	The software must receive formula written in EC-Assertion and feed it to EVEREST.	Functional	The rule is successfully feed to EVEREST. This can be tested by checking the database to see if the rule has been recorded.
F-5	The software must detect rule violation successfully.	Functional	Rule violation is detected successfully. This can be test by the following steps: • Input a rule the software. • Create events that cause the rule to be violated. • Check in the database to see if the violated rule has been recorded.
NF-2	The software can detect rule violation correctly.	Non-Functional	Rule violation is detected correctly. This can be tested by the following steps: • Input a rule the software. • Create events that cause the rule to be violated. • Check in the database to see if the rule has been recorded as violated rule. • Create events that satisfy the rule. • Check in the database to see

			if the rule has been recorded as satisfies.
NF-3	The software should process 2000 events in 5 minutes.	Non-Functional	The time taken for the software to process 2000 events is less than 5 minutes. This can be tested by writing a small unit test that compares the start time of the processing and the end time of the processing. If the (end time – start time) < 5 minutes then the test is passed. Dummy events will be generated for the test case.

From the original objectives, requirements have been divided into functional and non-functional requirements. In order for EVEREST to monitor security, dependability or customer agreement properties from OpenStack, the new software need to be able to capture or query for information from OpenStack. Therefore, requirement **F-1** and **F-3** has been specified in order to ensure our software can do the required functionality. Requirement **F-2** is needed because event received from OpenStack's notification bus is in different format with event that EVEREST expected. As a result, our software have to convert the notification bus's events into internal EVEREST event. Since, users have to specify monitor rules for EVEREST to perform monitoring activities, there should be a way for our software to capture user's rules and feed them to OpenStack. This functionality has been specified by requirement **F-4**. Lastly, requirement **F-5** specify the main responsibility of our software, the software should be able to use EVEREST to diagnosing event and detecting rule violation. Come with each requirement is the Pass Criteria identified for each requirement as well as instruction on how to validate them.

3.2.2. Overall Architecture

Based on the literature review about OpenStack and EVEREST before, there are 2 main components that will need to be implemented: the EventCapturer and the Poller. The EventCapturer listens for event from notification bus and Poller polls for information from OpenStack by calling OpenStack API. The diagram below illustrates the overall architecture of our program:

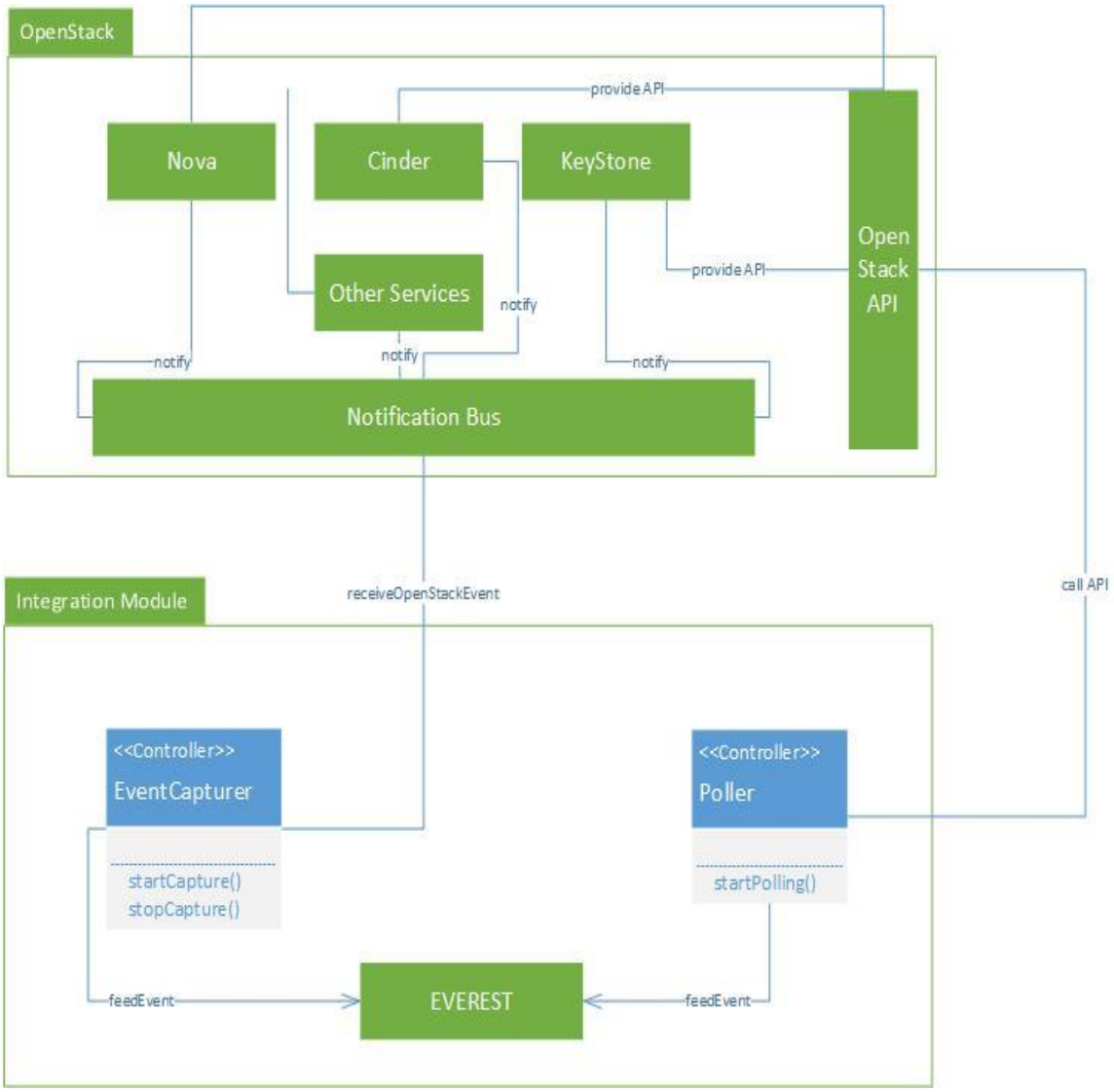


FIGURE 6.OVERALL ARCHITECTURE

3.2.3. Analysis Class Result

Since there are many different methods in OpenStack API for querying different information, an interface PollerTask is created to represent different query tasks. And for each querying job, an implemented of PollerTask needs to be created. The Poller class manages the execution of all tasks. A MainController class is used to manage the creation and usage the two Poller and EventCapturer classes.

Although all notification message from OpenStack have the same format. Their payload is different; each type of notification message contains different information in their payload body. As a result, it is hard to convert all messages directly to EVEREST's event format. The solution to this problem is having the EventConverter converts events received from OpenStack and transforms them to an intermediate unified format event objects called OpenStackEvent. After that, a new EverestTranslator class translates the OpenStackEvent to EVEREST's events. This way all different format messages are converted to same format event objects first. Then the intermediate event objects can be translated to EVEREST's events easily.

In order to feed events and rules to EVEREST, An EverestGateway class acts as a gateway to EVEREST, it handles the initial configuration as well as creation of EVEREST's entry class NewDataAnalyzerEC. The table below showing all the major components that we have defined:

Class Name	Description
EventCapturer	Listen to event from notification bus.
EventConverter	Convert notification event to intermediate OpenStackEvent.
EverestTranslator	Translate OpenStackEvent to EVEREST's LogEvent.
OpenStackEvent	The intermediate event class.
LogEvent	Belong to EVEREST framework and represent EVEREST's event.
Poller	Manage the execution of PollerTask
PollerTask	Call to OpenStack API for obtaining information.
MainController	Manage EventCapturer and Poller instances.
EverestGateway	Act as the gateway to EVEREST.Manage EVEREST configuration and creation.

The diagram below is the analysis class diagram of our system:

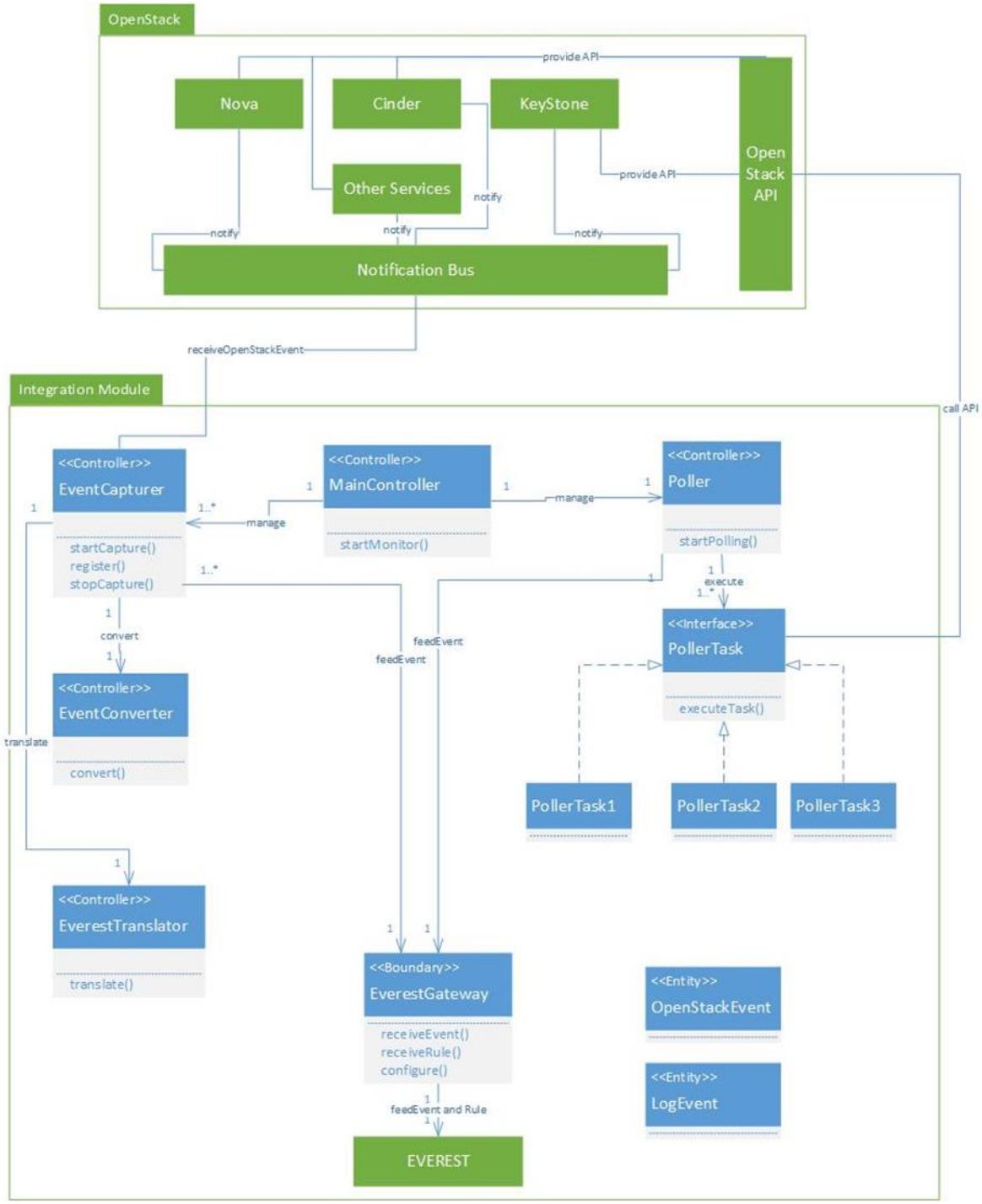


FIGURE 7 ANALYSIS CLASS DIAGRAM

The diagram in Figure.7 also include OpenStack's model in order to illustrate the interaction between the new software with OpenStack's Infrastructure service clearly. As you can see, the EventCapturer class listens to notification bus to receive event from OpenStack and PollerTask class calls OpenStack API to query for information. Both EventCapturer and Poller submit their events directly to EverestGateway. The MainController's responsibility is to manage Poller and EventCapturer instance. Since, there can be more than 1 notification bus in OpenStack cloud, i.e. 2 notification message queues, each installed in different nodes; we decide that the relationship between EventCapturer and MainController should be Many to One.

Once receiving notification from OpenStack, the EventCapturer converts the message to OpenStackEvent object and then use EverestTranslator to translate the intermediate event object to LogEvent.

3.2.4. Analysis Sequence Result

There are 5 functional requirements that have been identified in section 4.2.1 above. Each of the functional requirements is illustrated by an analysis sequence diagram to describe integrations between components to realize the requirements. The 5 functional requirements are listed below:

- **F-4:** The software must received formula written in EC-Assertion and feed it to EVEREST.
- **F-1:** The software must capture event from OpenStack's notification bus.
- **F-2:** The software must convert event from OpenStack's notification bus to EVEREST's compatibility event.
- **F-3:** The software must obtain information from OpenStack by calling OpenStack API.
- **F-5:** The software must detect rule violation successfully.

The following sequence diagrams illustrate the interaction between components in abstract term to realize different functional requirement. The first analysis sequence diagram specifies how different system's components interact together to realize requirement **F-4**.

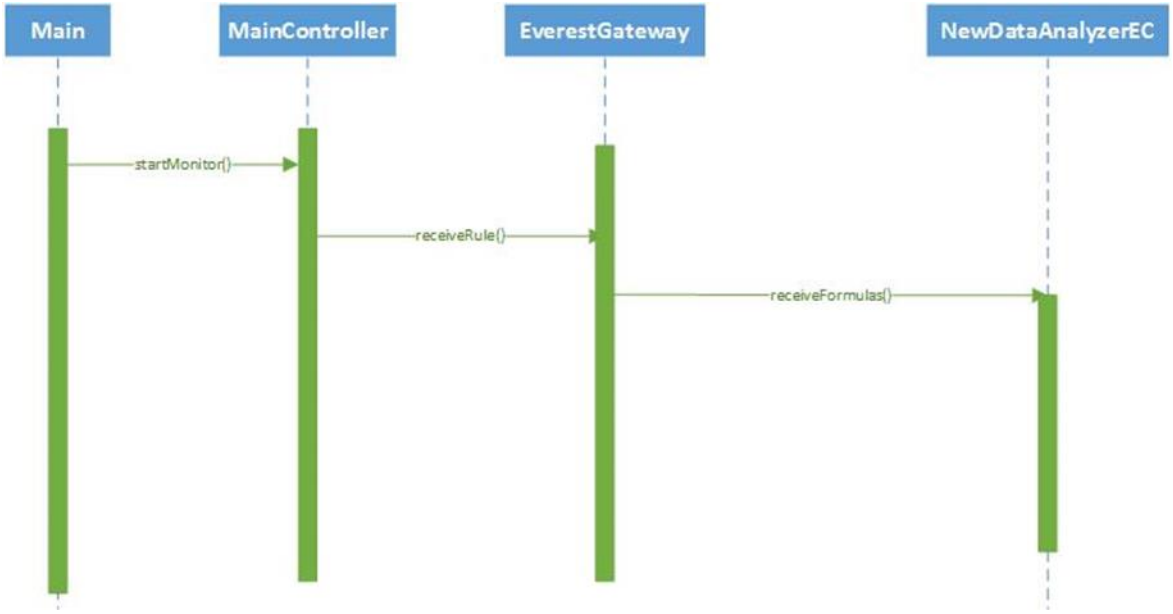


FIGURE 8ANALYSIS SEQUENCE DIAGRAM FOR SUBMITTING RULE.

The Main component in this case is the entry point of our program, it is the main class that contain public static void main (String[] args) method. When our software runs, this is the first class that will be loaded and interpreted by JVM to act as a starting point for our software. Once main method is called, the MainController instance is created and it's startMonitor() method is called. However, how can users input their monitor rule to our software? Basically, when installing our software, the user will have to specify a local directory that will store all the

software' configured file. The monitor rule specified by user will be put in an xml file written in EC-Assertion language, and the xml file will be put in the configured directory. When startMonitor() method is called, the MainController instance loads the xml rule file in the configured directory and passes receiveRule() message to EverestGateway instance with the rule file path as it's parameter. Finally, EverestGateway instance feeds the rule to EVEREST' class instance NewDataAnalyzerEC for processing and record in database.

The second analysis diagram show how our software's components interact to realize requirement **F-1**, **F-2** and **F-5**. These requirements relate to each other, each requirement is connected to each other in order to realize the main functionality of our software: detecting rule violation. As a result, we decide to combine them together in this analysis diagram:

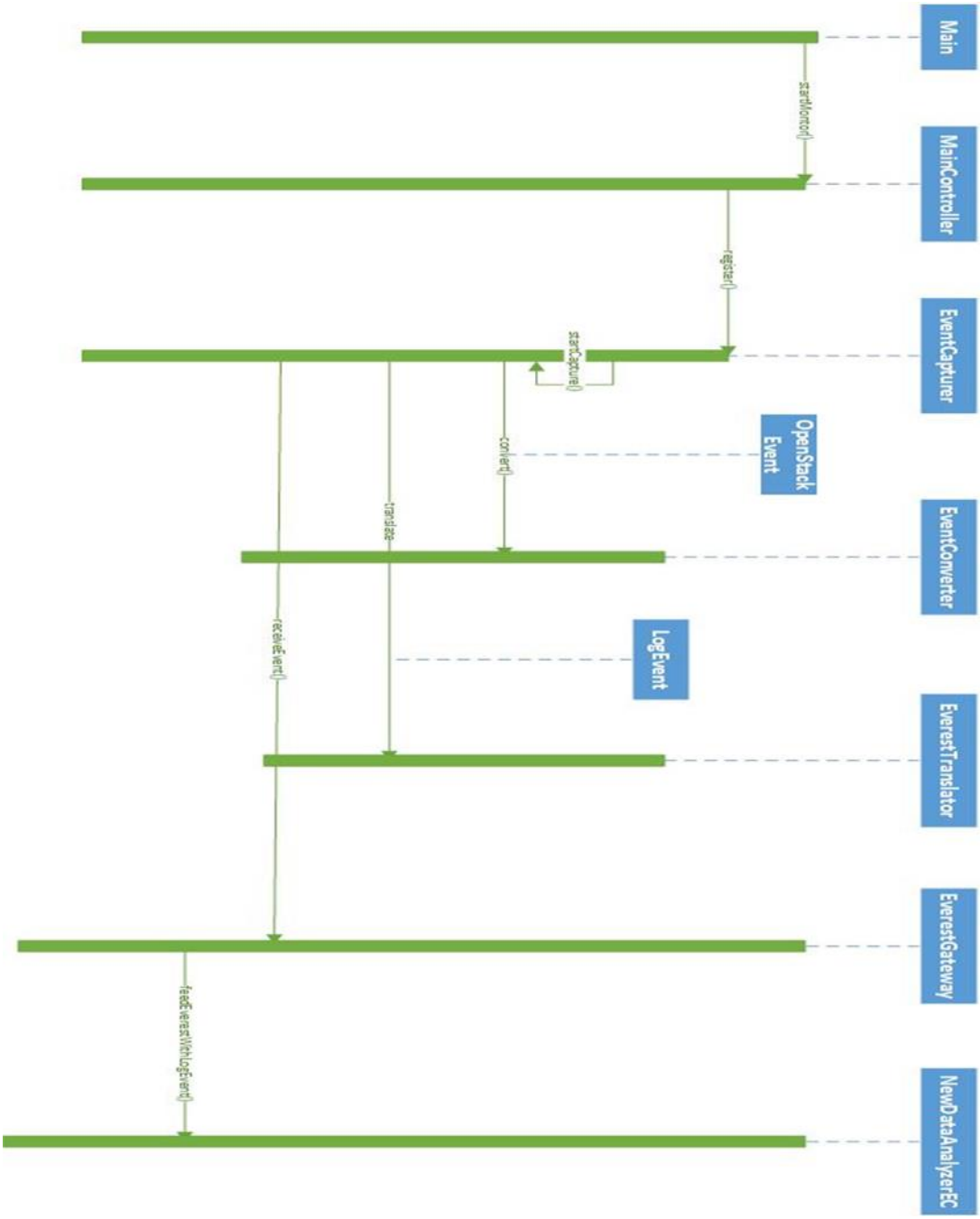


FIGURE 9ANALYSIS SEQUENCE DIAGRAM – EVENTCAPTURER

In the analysis diagram above, when our program is started, the main method will invoke MainController's startMonitor method. This method then call EventCapturer's register() method to register a listener to OpenStack's notification bus's topics. After that EventCapturer instance will start listen on the registered topic to receive incoming notification events. Upon receiving event message from OpenStack, EventCapturer convert the message to LogEvent object using EventConverter and EverestTranslator instance and submit the converted event to EverestGateway instance. Finally, the gateway pass the event to NewDataAnalyzerEC instance for event processing, when an events that cause rule violation are detected, the violated rule will be recorded in the database.

The third analysis diagram show how our software's components interact to realize requirement **F-3**, **F-5**. The reason for this combination is the same as the second analysis diagram.

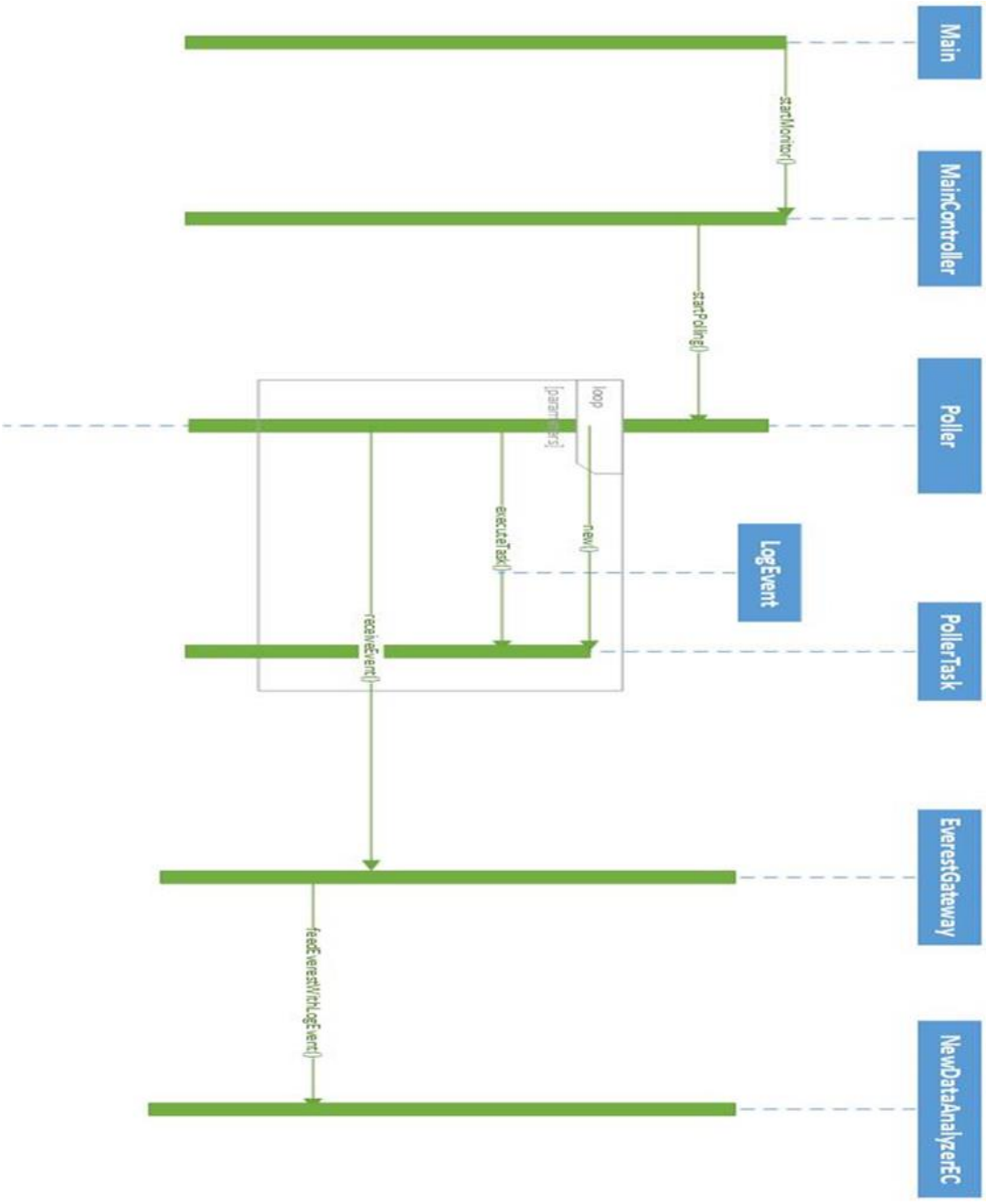


FIGURE 10ANALYSIS SEQUENCE DIAGRAM – POLLER

The interaction flow for functional requirement **F-3** is somewhat similar to the second analysis sequence diagram in Figure 9. The main different is the way to obtain information, in Figure 10, information is obtained by using Poller instance. When Poller instance is started, it will loop through all the PollerTask instances that it stores and execute their method. The implementation for executeTask() method for each implement class of PollerTask is different depend on what information they want to obtain. After querying for information such as server instance info, volume info, etc. the PollerTask instance will automatically convert the info directly into LogEvent object. The reason for not using EventConverter to convert to an OpenStackEvent object is because the information querying from OpenStack API is totally different in their format and content to each other, therefore, we cannot find a way to unify them together into an intermediate object like how we do with event captured from notification bus. As a result, the implement classes of PollerTask have to manually extract the wanted info from the queried information and convert them to LogEvent objects. Finally, the Poller will submit the event result from PollerTask to EverestGateway and have them processed. Any events that violate the rule will be detected, and EVEREST will record the violated rule in the database.

In conclusion, the requirement and analysis phase activities helped us to come up with a set of functional and non-functional requirement for our software as well as an abstract design of system's components. The requirements were the objective that our design tried to solve. And the result of analysis diagram gave us an overall of idea of which components should be created and the relationships between them as well as the initial design our software's architecture. The analysis diagram and requirement were used as input material to inform design decision in Design phase.

3.3. Design Result

In this section, we will show diagrams produced in Design phase and give detail explanation about each components and their relationship with each other as well as how they interact with each other.

3.3.1. Design Class Result

Based on the analysis class diagram, we had come up with a complete class design for our software as below:

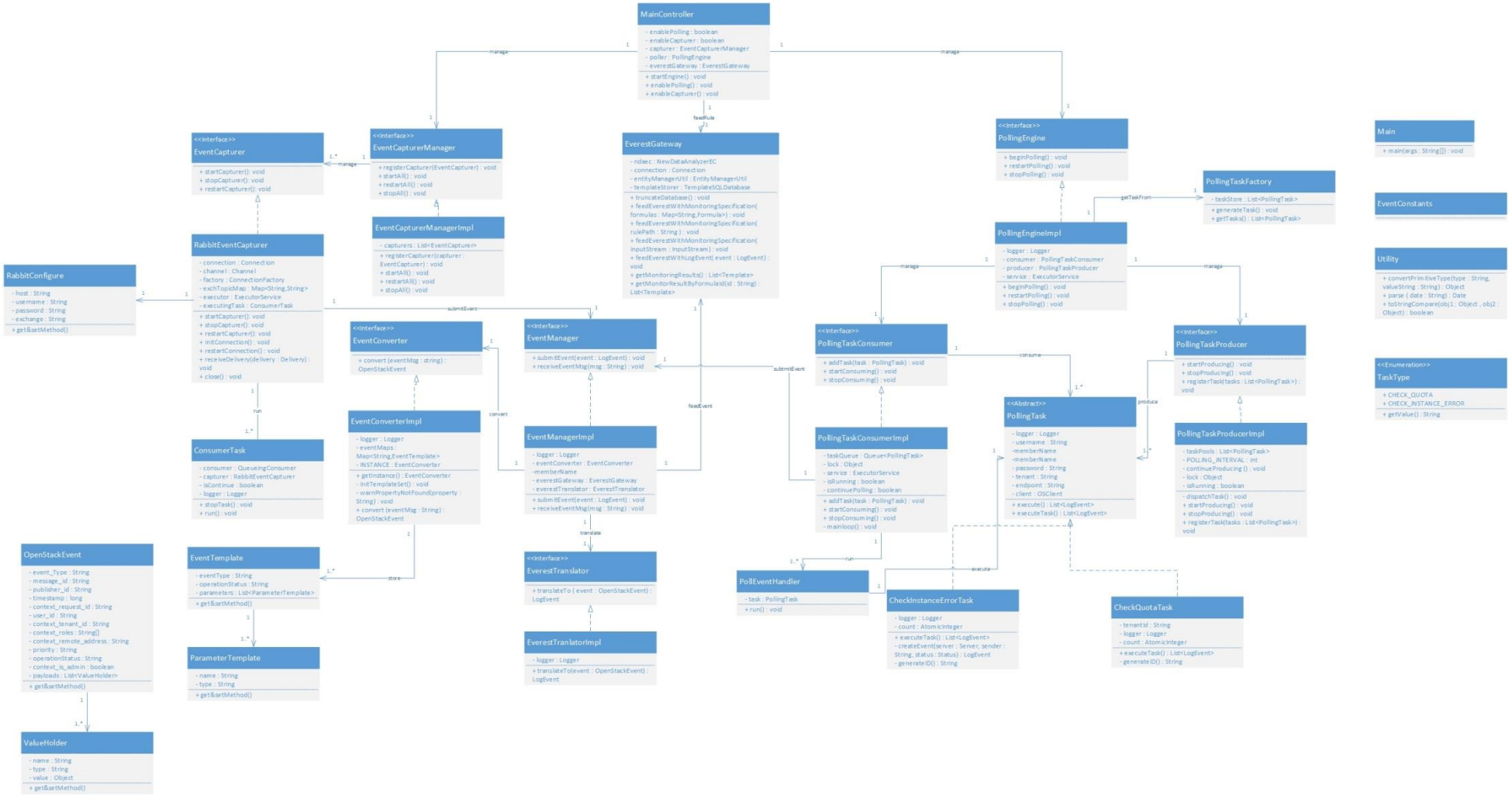


FIGURE 11DESIGN CLASS DIAGRAM

Although, the design class diagram is much more complicated than the analysis design class, the idea behind the software's design is still the same as the analysis design. The design still consists of 3 main modules:

- **The EventCapturer module**: this module contains 4 important classes: EventCapturerManager, EventCapture, EventConverter and EverestTranslator.
- **The Poller module:** this module consists of 5 important classes: PollingEngine, PollingTaskConsumer, PollingTaskProducer, PollingTask and PollingTaskFactory.
- **The Gateway module**: this module contains 2 important classes: EventManager and EverestGateway.

1. The Event Capturer Module

The diagram below is part of Figure.11 Design Class Diagram, which make up the EventCapturer module:

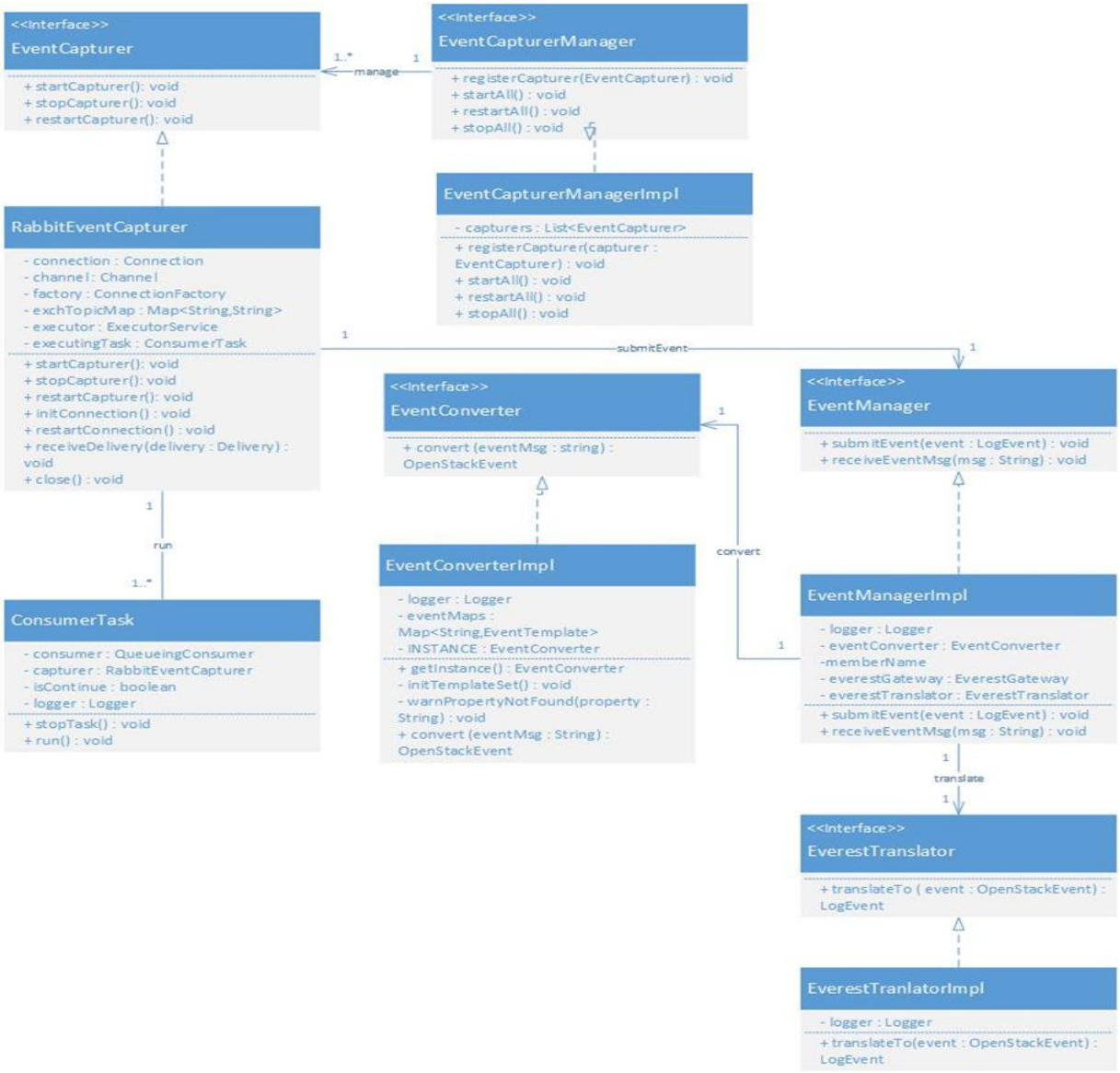


FIGURE 12DESIGN CLASS - EVENTCAPTURER PART

In this design, the responsibility of managing EventCapturer instances is resides in EventCapturerManager class; the EventCapturerManagerImpl is the implementation of EventCapturerManager interface.

Class EventCapturerManagerImpl : attributes	
private List<EventCapturer> capturers	This list variable store the list of managed EventCapturer instances
Class EventCapturerManagerImpl : method signatures	
public void registerCapturer(EventCapturer capturer)	This method registers a new EventCapturer instance to this manager. The new registered object will be managed by this manager.
public void startAll()	This method will start all the managed EventCapturers instances. The capturers will begin to listen for event from notification bus.
public void stopAll()	This method will stop all the managed EventCapturers instances. The capturers will stop listening for event from notification bus.
public void restartAll()	This method will restart all the managed EventCapturers instances. The capturers will restart the whole listening process again. This method can be used in case if there is any exception happen and we want the capturers to resume listening for event again.

The EventCapturer class is now an interface. So there can be different implementation of capturers for different message queue brokers such as RabbitMQ implementation, ZeroMQ implementation. In our software rightnow, there is only implementation for RabbitMQ implementation. However, further extension of other message brokers can be implemented easily by implement EventCapturer interface and the system can manage it naturally. The RabbitEventCapturer class is our RabbitMQ implementation in the software. In order to register to RabbitMQ message queues, certain configuration properties are required such as: RabbitMQ host address, exchange and topic name, username and password. This information is configured in rabbit.properties file and the RabbitConfigure takes those properties and stores in its attribute. After that, RabbitConfigure instance is passed as parameter when calling RabbitEventCapturer constructor. Finally, RabbitEventCapturer uses this information to register to RabbitMQ's topics. The table below show details about RabbitEventCapturer class:

Class RabbitEventCapturer : attributes	
private Connection connection	Is com.rabbitmq.client.Connection class, this object create connection to a rabbit broker. Created by using ConnectionFactory.

private Channel channel	Is com.rabbitmq.client.Channel class, act as an interface to AMQ channel.
private ConnectionFactory factory	Is com.rabbitmq.client.ConnectionFactory class, is a factory class that produce connection to a broker.
private Map<String, String> exchtopicMap	This is a Map contain key-value pair. The key is the name of exchanges in the notification bus and the value is the topic name of the queue that we want to receive notification.
private ExecutorService executor	We use this ExecutorService in order to create different listening thread for each topic.
private List<ConsumerTask> executingTasks	The list of ConsumerTask instances. Each ConsumerTask instance is associated with a topic thread. In each thread, the ConsumerTask will listen for notification message from a topic queue and deliver them back to RabbitEventCapturer.
Class RabbitEventCapturer : method signatures	
public void startCapturer()	This method starts the capturer by creating ConsumerTask for each exchange/topic and run them in separate threads.
public void stopCapturer()	This method stops all the ConsumerTask threads.
public void restartCapturer()	This method stops all the ConsumerTask threads and restarts the whole process.
public void initConnection()	This method establishes connection to RabbitMQ broker.
protected void restartConnection()	This method restarts connection to RabbitMQ broker.
protected void receiveDelivery(Delivery delivery)	This method receives the notification message received in ConsumerTask and delivers them to other component.
public void close()	Stop all the ConsumingTask and release all resources such as closing all connections.

The ConsumerTask class is created to consume messages from different topic. The ConsumerTask is introduced to enable concurrent processing of messages from many topics. If we follow the old design in analysis phase, the capturer will be stuck on listening messages from one topic, only after receive event from that topic, it then can proceed to listen to event from other topics. The old design will create blocking and bottle neck in this class when listening for

events. The new design is aim at solving this problem. The table below shows the details information of this class.

Class ConsumerTask : attributes	
private QueueingConsumer consumer	Is com.rabbitmq.client.QueueingConsumer class, this class receive message and notification from queue by subscription.
private RabbitEventCapturer capturer	The EventCapturer associated with this ConsumerTask object.
private volatile boolean isContinue	A flag for checking if we want to continue receiving message from queue.
Class ConsumerTask : method signatures	
public void run()	This method starts a loop to receive incoming message from the message queue.

The EventConverterImpl is the implementation of interface EventConverter. This class's responsibility is to convert OpenStack's event to an intermediate OpenStackEvent object. We will explain how our software convert event message from OpenStack to OpenStackEvent object using EventTemplate in detail in section **4.4.2**. The table below show details of EventConverterImpl class.

Class EventConverterImpl : attributes	
private Map<String, EventTemplate> eventMaps	The key-value Map of the EventTemplate object and its event type. A detail of how EventConverterImpl use EvenTemplate to convert event will be explained briefly in section 4.3.1
Class EventConverterImpl: method signatures	
private void initTemplateSet()	Initiate the event template map.
public OpenstackEvent convert(String eventMsg)	Convert event messages from OpenStack to OpenStackEvent object.
private void warnPropertyNotFound(String property)	This private method used for logging purpose.

The EverestTranslatorImpl is the implementation of interface EverestTranslator. This class is responsible for translating OpenStackEvent object to LogEvent object. The table below show details of EverestTranslatorImpl class:

Class EverestTranslatorImpl: attributes
Class EverestTranslatorImpl: method signatures

public LogEvent translateTo(OpenstackEvent event)	Translate OpenStackEvent object to LogEvent object.

The EventConverter and EverestTranslator class are used by EventManager class. This class receives event messages captured by EventCapturer and uses EventConverter and EverestTranslator to convert it to LogEvent object. Finally, it submits LogEvent object to EverestGateway. Detail of EventManager will be explained in The Gateway Module section.

2. The Poller Module

The diagram below is part of Figure.11 Design Class Diagram, which make up the Poller module:

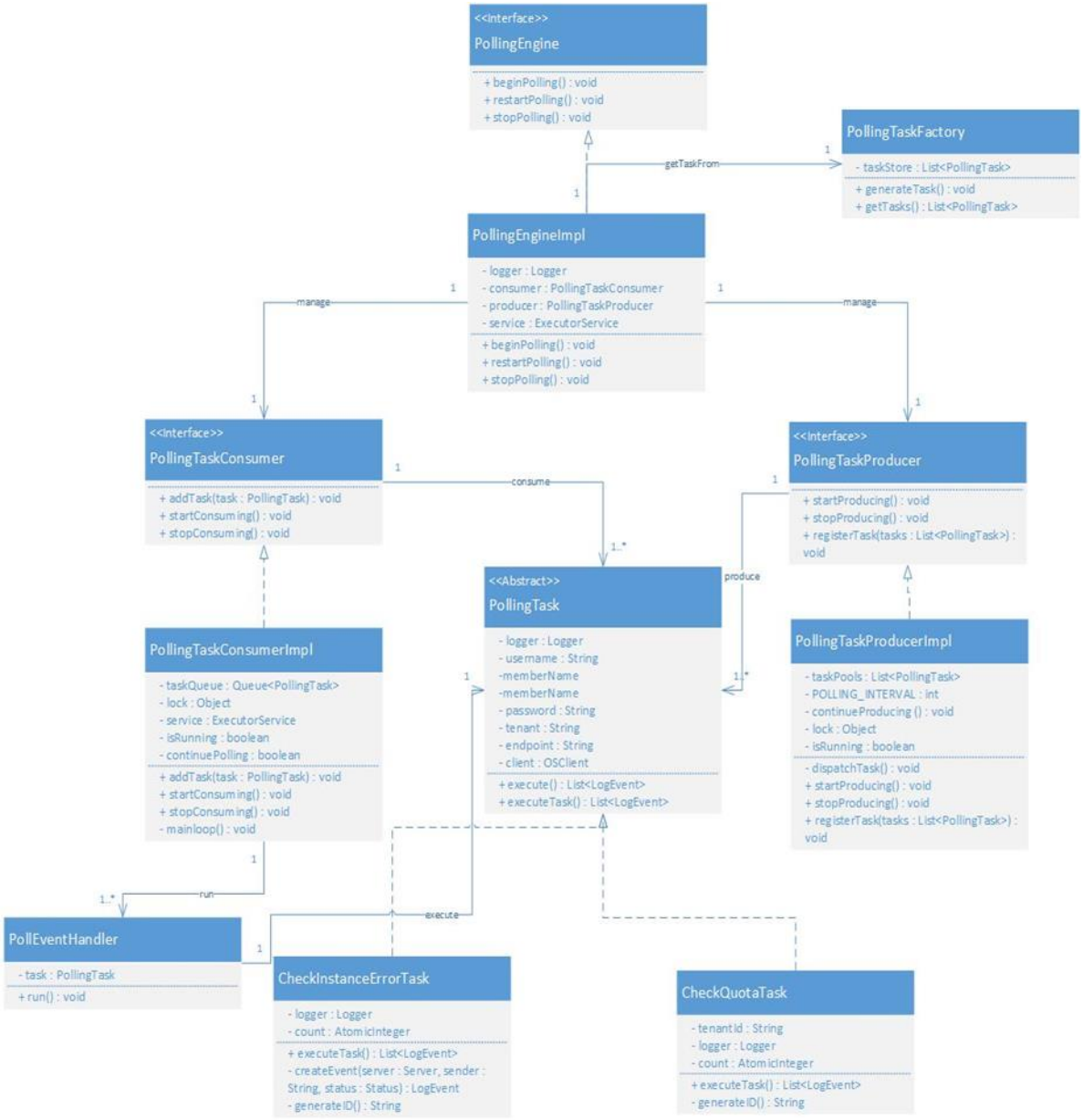


FIGURE 13DESIGN CLASS DIAGRAM - POLLER PART

The PollingEngine class in this diagram is the starting point of this Poller module. This class responsibility is to start/stop the whole polling engine. The polling engine consist of 2 main components the task producer and task consumer. The task producer produces PollingTask for querying OpenStack information, and the task consumer, takes tasks produced by the producer and execute them. This design follow Producer-Consumer pattern using wait() and notify() method. The table below show details of PollingEngineImpl, an implementation of PollingEngine interface.

Class PollingEngineImpl: attributes	
private PollingTaskConsumer consumer	The task consumer

private PollingTaskProducer producer	The task producer
private ExecutorService service	We use this ExecutorService in order to create separate thread of execution for task producer and task consumer.
Class PollingEngineImpl: method signatures	
public void beginPolling()	Starting the polling engine. This method will create 2 different threads for task producer and task consumer.
public void restartPolling()	Restart the polling engine.
public void stopPolling()	Stoping the polling engine. This method will stop task producer and consumer thread.

The PollingTaskProducerImpl is the implementation of PollingTaskProducer interface. Its main responsibility is to manage a pool of reused task. Instead of creating new task for each interval, it reuses a previous task that performs the same function. For each interval, the producer pushes all the tasks in the task pool to its consumer for task execution. The table below shows details of PollingTaskProducerImpl.

Class PollingTaskProducerImpl: attributes	
private List<PollingTask> taskPools	A list contain all the tasks for reusing
private final int POLLING_INTERVAL	The poll interval. After polling for information, it will wait for a period of time before continuing polling.
private boolean continueProducing	A flag that check if stopProducing() is called.If set to true, the producer will stop produce tasks to consumer.
Class PollingTaskProducerImpl: method signatures	
public void registerTasks(List<PollingTask> tasks)	Used for registering task instance to the producer's task pool.
private void dispatchTasks()	Dispatch the task instance to PollingTaskConsumer for execution.
public void startProducing()	Start a loop that dispatch task to consumer by interval.
public void stopProducing()	Stop dispatch task to consumer.

The PollingTaskConsumerImpl is the implementation of PollingTaskConsumer. Its main responsibility is to take task produce by PollingTaskProducer and execute them. Each PollingTask will be executed in a separate thread to improve concurrency and performance.

Class PollingTaskConsumerImpl: attributes	
private Queue<PollingTask> taskQueue	A queue contains all tasks produced by PollingTaskProducer.
private ExecutorService executors	We use this ExecutorService in order to create separate thread of execution for each PollingTask.
private volatile boolean continuePolling	A flag that check if stopConsuming () is called.If set to true, the consumer will stop execute new tasks.
Class PollingTaskConsumerImpl: method signatures	
public void startConsuming()	Start a loop to take tasks from taskQueue and execute them in separate threads.
public void stopConsuming()	Stop executing new tasks. Set continuePolling to false.
public synchronized void addTask(PollingTask task)	Used by PollingTaskProducer to add task to taskQueue for execution.

The PollingTaskFactory responsibility is to create all type of PollingTask instance and put them to PollingTaskProducer's taskPool for reusing. Any new class that implement PollingTask, added to the system will have to register itself in this factory for later execution.

Class PollingTaskFactory: attributes	
private List<PollingTask> taskStore	A list contains all the registered task instances.
Class PollingTaskFactory: method signatures	
public void generateTask()	Create all the task instances that will be used for polling.
public List<PollingTask> getTasks()	Get all the registered task instances.

The PollingTask is an abstract class. If we want to create a new task that other OpenStack's API to query for information, we have to create a new class which extend PollingTask and implement querying logic in there. We have implemented 2 tasks CheckInstanceErrorTask and

CheckQuotaTask class to be subclass of PollingTask. These 2 classes are used for testing and demonstration purpose.

Class PollingTask: attributes	
protected String username	The username of OpenStack's user credential
private String password	The username of OpenStack's user credential
protected String tenant	The tenant id of the OpenStack's user
private String endpoint	The identity endpoint to connecto
protected OSClient client	Is org.openstack4j.api.OSClient class.The OpenStack client will be used for calling OpenStack API.
Class PollingTask: method signatures	
public List<LogEvent> execute()	Called by consumer to execute the task. However actual implementation logic is reside in subclass's executeTask() method. This method simply create an OSClient and call the executeTask() method that overridden by subclasses.
protected abstract List<LogEvent> executeTask()	This method contains the actual implementation logic. It will query for OpenStack's information and extract the necessary data to create LogEvent objects.

3. The Gateway Module

The diagram below is part of Figure.11 Design Class Diagram, which make up the Gateway module:

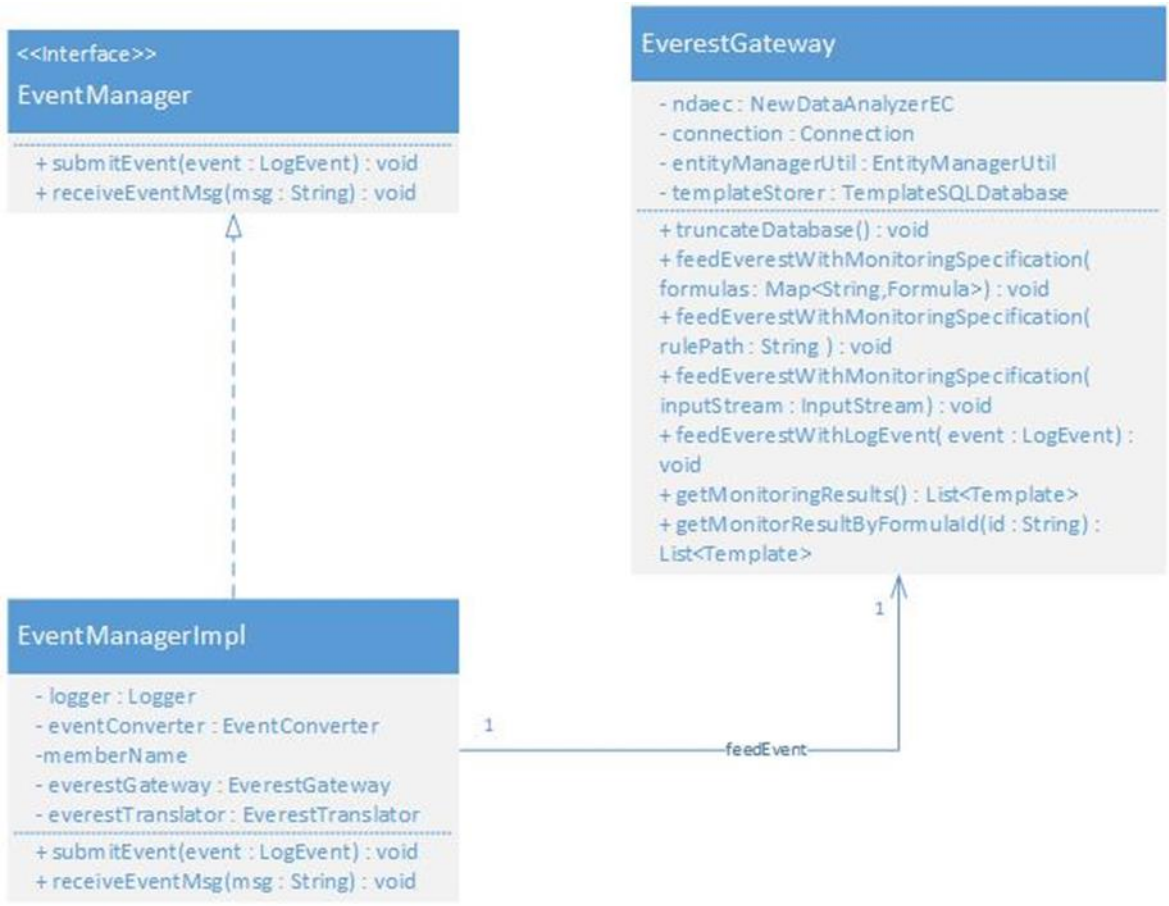


FIGURE 14DESIGN CLASS DIAGRAM - GATEWAY MODULE

The EventManager class is responsible for handling event message logic. It receives raw event messages from EventCapturer and uses EventConverter and EverestTranslator to convert them to LogEvent objects and finally submits the converted event to EverestGateway. It also receives LogEvent objects created by PollingTask and submit them to EverestGateway. The EventManagerImpl is the implementation class of EventManager interface.

Class EventManagerImpl: attributes	
private EventConverter eventConverter	Instance of EventConverter.
private EverestGateway everestGateway	Instance of EverestGateway.
private EverestTranslator everestTranslator	Instance of EverestTranslator.
Class EventManagerImpl: method signatures	
public void submitEvent(LogEvent event)	Submit LogEvent object to EverestGateway.

public void receiveEventMsg(String eventMsg)	Receive OpenStack's event message from EventCapturer and convert it to LogEvent object using EventConverter and EverestTranslator.

Class EverestGateway: attributes	
private NewDataAnalyzerEC ndaec	Instance of uk.ac.city.soi.everest.NewDataAnalyzerEC.
private TemplateMySQLDatabase templateStorer	Instance of uk.ac.city.soi.everest.database.template.TemplateMySQLDatabase.
Class EverestGateway: method signatures	
public void feedEverestWithMonitoringSpecifications(LinkedHashMap<String, Formula> slaFormulas)	Submit rule that has been converted to Formula objects to EVEREST.
public void feedEverestWithMonitoringSpecifications(String filePath)	Take filePath as input and load the rule xml file based on directory specified by filePath. The rule will then be converted to Formula objects and submit to EVEREST.
public void feedEverestWithMonitoringSpecifications(InputStream inStream)	Take inputStream as it input and parse the input stream to Formula objects and submit to EVEREST
public synchronized boolean feedEverestWithLogEvent(LogEvent event)	Submit LogEvent to EVEREST for processing.
public ArrayList<Template> getMonitoringResults()	Get the monitor result from EVEREST's database
public ArrayList<Template> getMonitorResultByFormulaId(String formulaId)	Get the monitor result by rule Id from EVEREST's database.

3.3.2. Design Sequence Result

Based on sequence diagram produced in analysis stage, we have come up with a set of design sequence diagram each illustrate a specific scenario.

The first sequence diagram illustrate requirement **F-4**: The software must receive formula written in EC-Assertion and feed it to EVEREST.

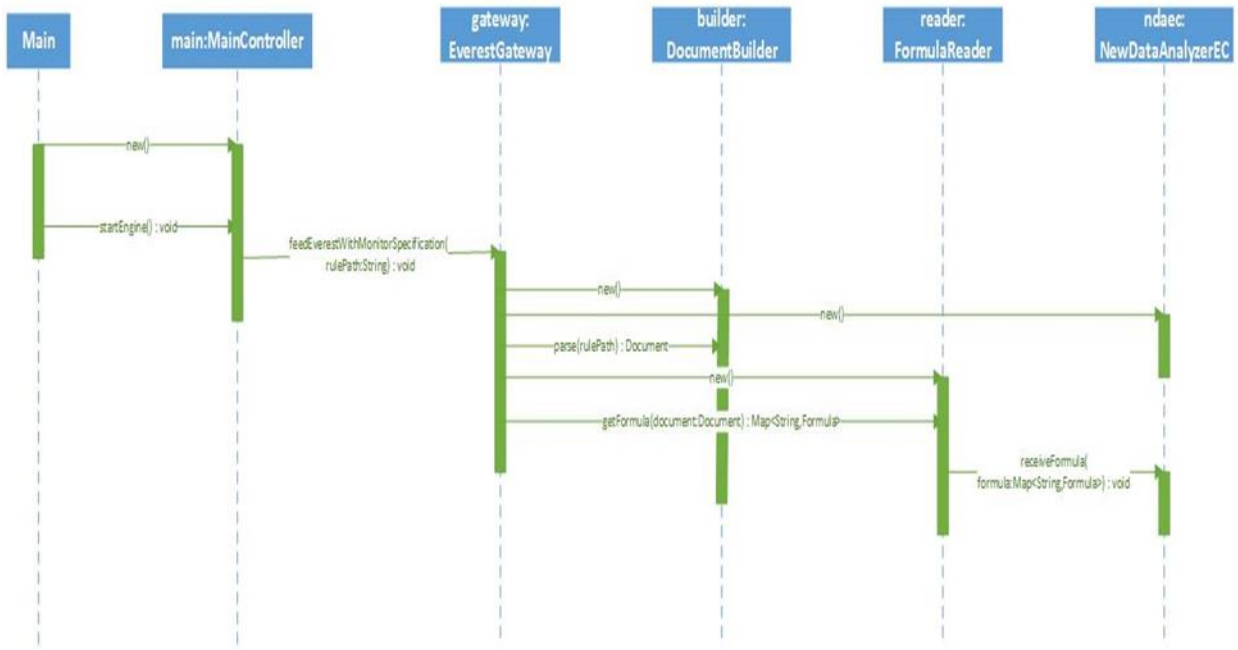


FIGURE 15.DESIGN SEQUENCE - REQUIREMENT F-4

The integrations in Figure 15, is perform in the following step:

1- When the program start, MainController object will be created and the main method will invoke its startEngine() method.

2- When startEngine() method is called, the MainController will pass the directory of rule xml file to EverestGateway object.

3- The EverestGateway object will then parse the rule xml file to Formula objects using FormulaReader and pass those objects to NewDataAnalyzerEC. Further rule processing will be performed by EVEREST and is not in this diagram scope. Finally, the Formulas will be recorded in database as Template.

The next sequence diagram will show how EventCapturer objects are set up and running:

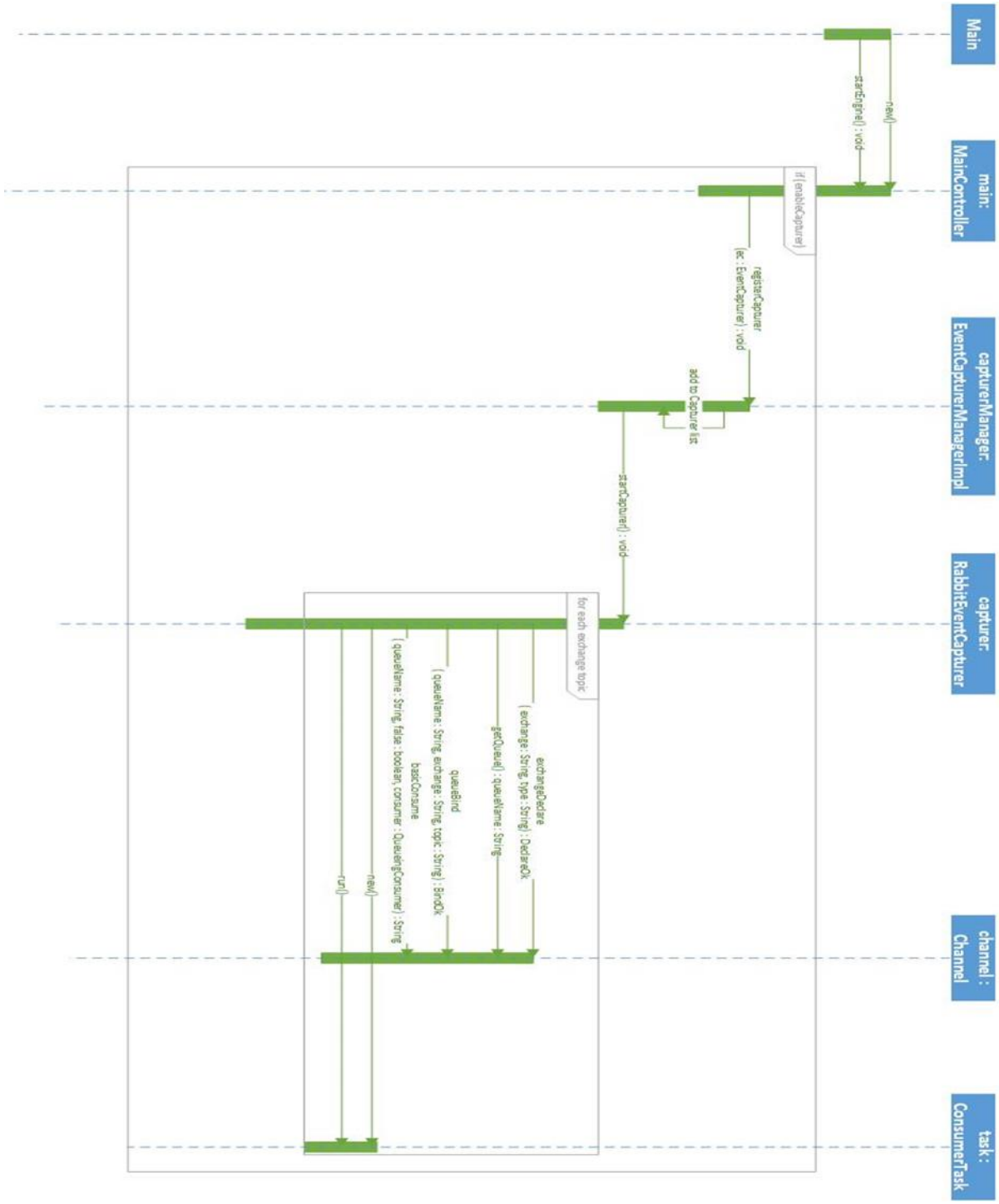


FIGURE 16DESIGN SEQUENCE DIAGRAM - SETUP CAPTURER

The integration in Figure.16 is performed in the following steps:

1- When the program start, MainController object will be created and the main method will invoke its startEngine() method.

2- The MainController then register the EventCapturer object to EventCapturerManager.

3- When EventCapturerManager adds the EventCapturer object to its managed EventCapturer list, it also call startCapturer() method of the EventCapturer object.

4- The startCapturer() method will register the capturer with all the specified topic in notification bus. For each topic, a ConsumerTask will be created to start listening on its own thread.

The following sequence diagram illustrates how PollingEngine is setup and running:

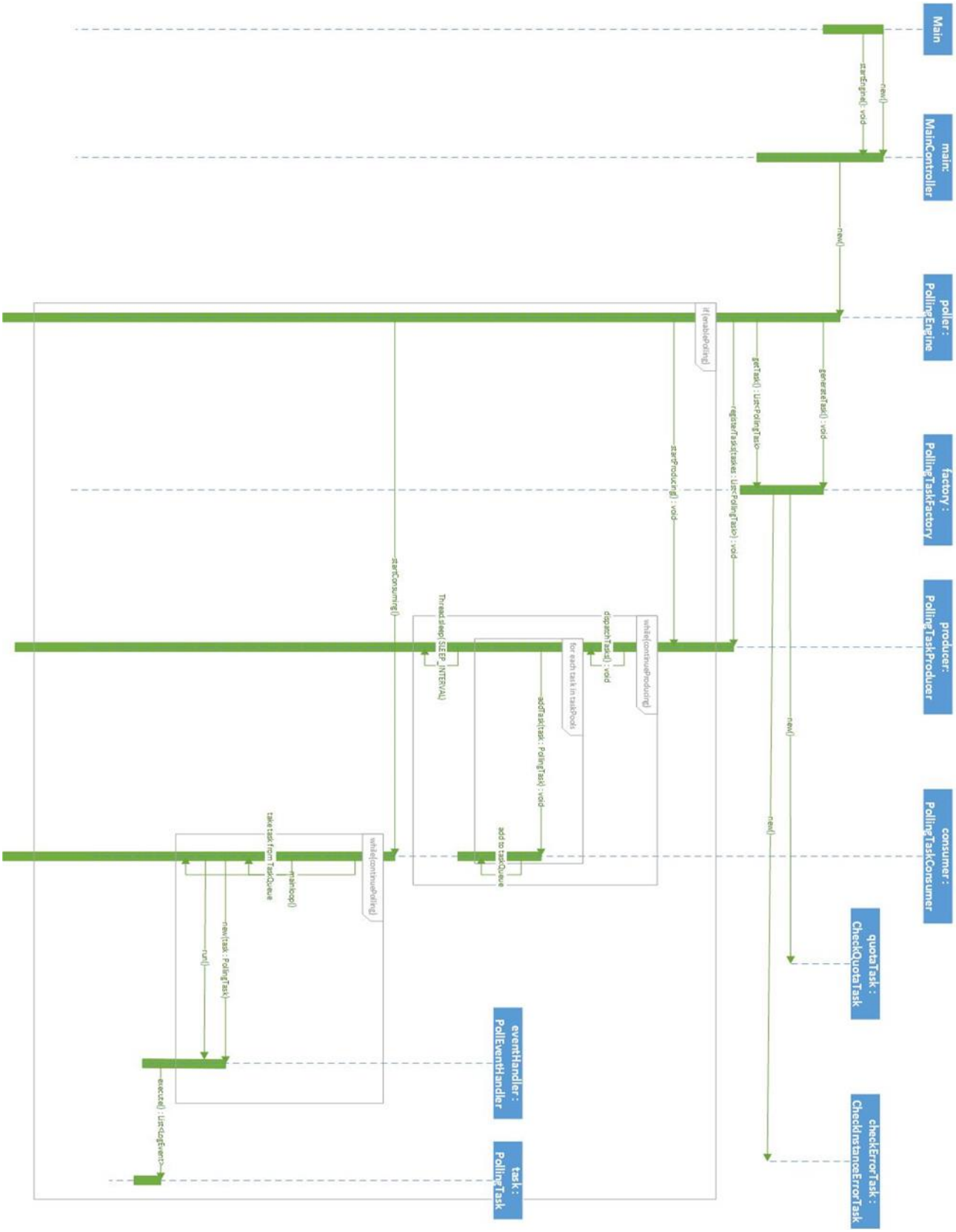


FIGURE 17DESIGN SEQUENCE DIAGRAM - POLLINGENGINE SETUP

The interactions in Figure 17 are performed in the following steps:

1- When the program start, MainController object will be created and the main method will invoke its startEngine() method.

2- The created PollingEngine will generate a list of PollingTask using PollingTaskFactory (in our case are CheckQuotaTask and CheckInstanceErrorTask).

3- The list of PollingTask objects are then registered to PollingTaskProducer.

4- When startProducing() is invoked on PollingTaskProducer instance. The PollingTaskProducer will dispatch all registered tasks to PollingTaskConsumer repeatedly by interval.

5- When startConsuming() is invoked on PollingTaskConsumer instance. The instance will start a loop and take PollingTask object produced by PollingTaskProducer and wrap it in a PollEventHandler object.

6- The PollEventHandler is a implementation of Runnable interface. It will be executed by ExecutorService. When PollEventHandler object is running, it will execute the PollingTask object wrapped in it.

The requirements **F-1**, **F-2** and **F-5** are realized by the following sequence diagram:

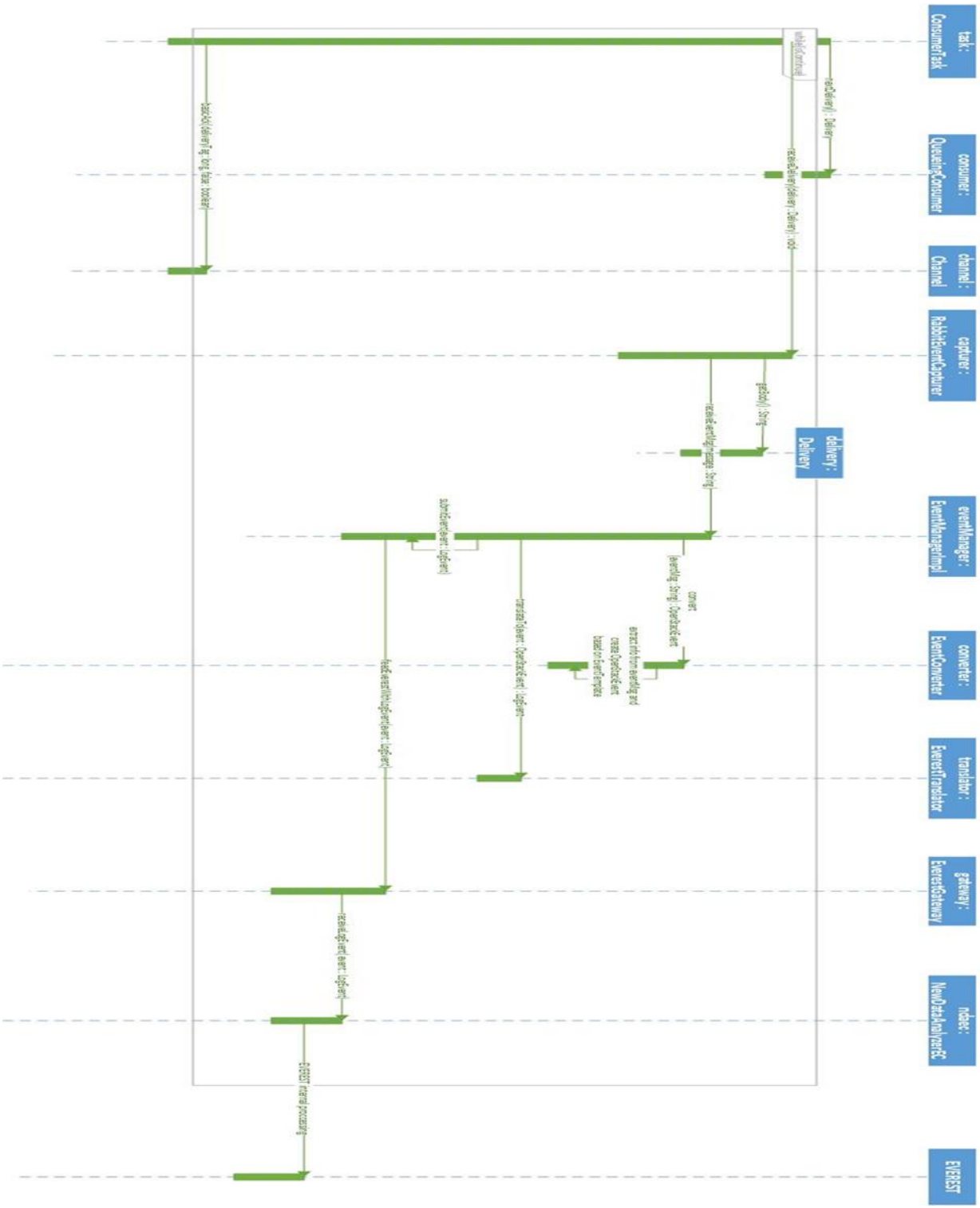


FIGURE 18DESIGN SEQUENCE DIAGRAM - REQUIREMENT F-1,F-2,F-5

The following steps are executed in figure 18:

1- When ConsumerTask object receives a message from QueuingConsumer, it will send the received message to RabbitEventCapturer.
2- The RabbitEventCapturer will then send the message to EventManagerImpl object.

3- The EventManagerImpl will convert the raw message to OpenStackEvent object by using EventConverter object and then translate the object to LogEvent object using EverestTranslator.
4- After that, EventManagerImpl will feed the LogEvent object to EverestGateway.
5- Finally, the EverestGateway will submit the LogEvent to NewDataAnalyzerEC by invoking receiveLogEvent(LogEvent) method.
6- The NewDataAnalyzerEC will process the event.

Lastly, requirement **F-3** and **F-5** are realized the below sequence diagram:

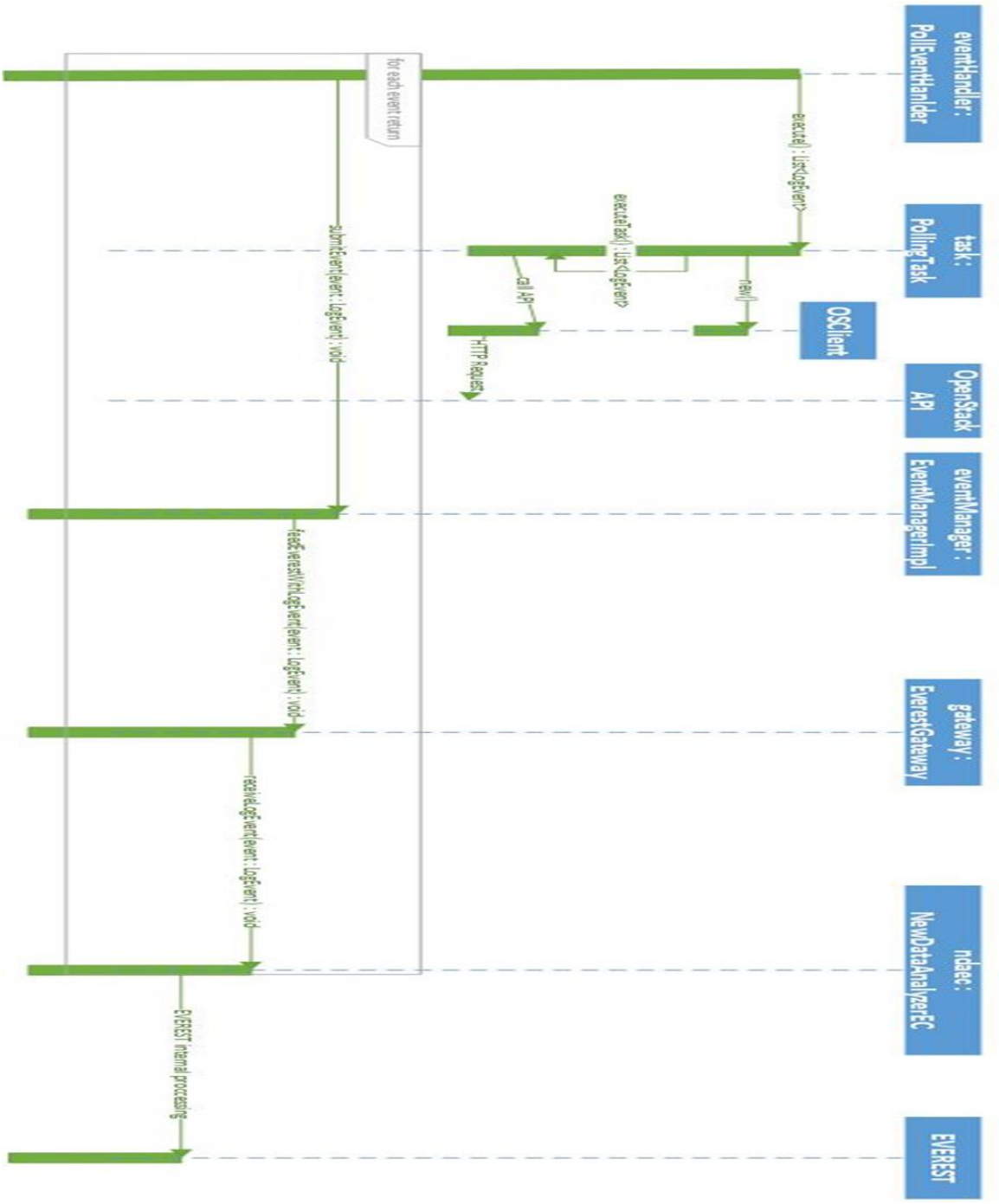


FIGURE 19.DESIGN SEQUENCE DIAGRAM - REQUIREMENT F-3, F-5

The following steps are performed in figure 19:

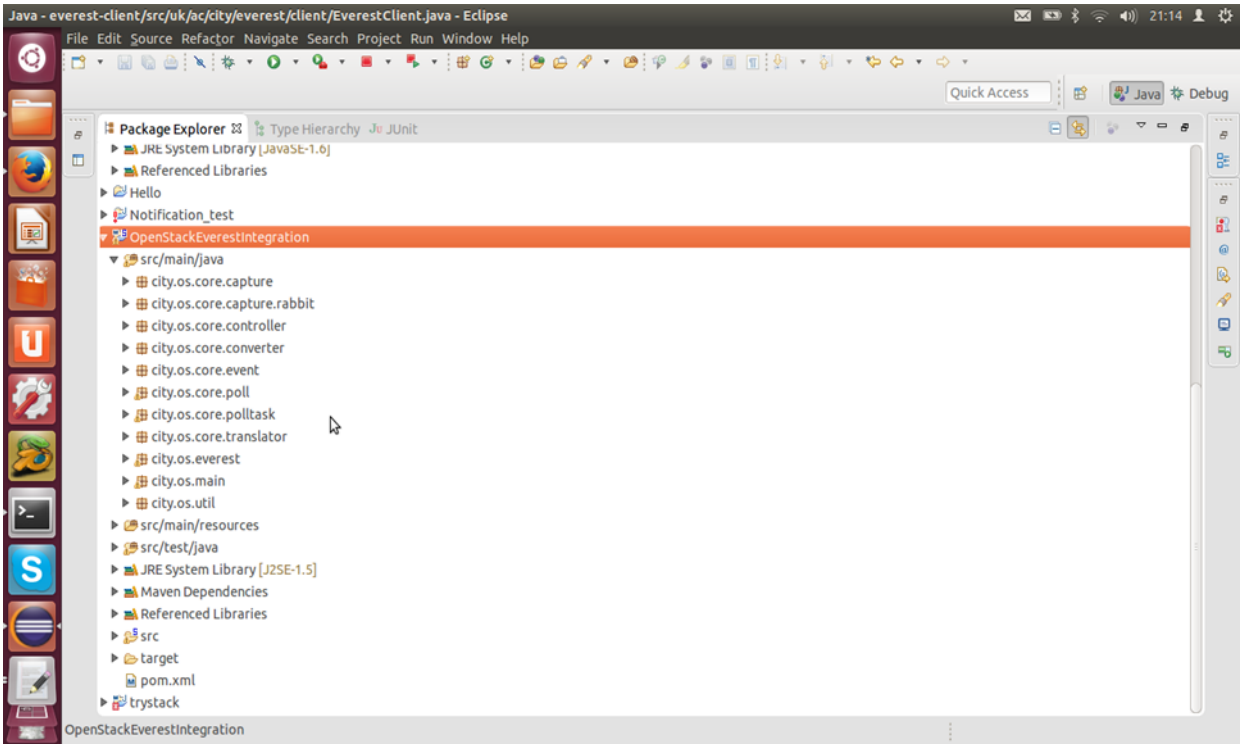
1- When PollingTask is executed, it will create an OSClient object to perform connection to OpenStack API and call the API methods.
2- After receving OpenStack information, PollingTask will extract all the necessary information and create corresponding LogEvent objects.
3- The LogEvent objects will then be send to EventManagerImpl by calling submitEvent(LogEvent) method.
4- EventManagerImpl object then submit the LogEvent object to EverestGateway.
5- Finally, the EverestGateway will submit the LogEvent to NewDataAnalyzerEC by invoking receiveLogEvent(LogEvent) method.
6- The NewDataAnalyzerEC will process the event.

3.4. Implementation Result

In this section, we will present the output produced by the software and a small section that explain about the algorithm that we use to convert the event.

3.4.1. Result Output

Based on the design class and sequence diagram, our software has been implemented. The packages are created corresponding to the components that we have identified in Design phase:



In order to verify if our software actually work, we have to check that our software is capable of listen for notification event as well as query for information from OpenStack in order to feed information to EVEREST. Therefore, we need to test 2 ways of integrations with OpenStack: capture event and query for info.

1. **<u>Using Event Capturer Result</u>**

For the first way, we have performed several steps to check for the result of our software. First, we have created a monitor rule for experimenting purpose. The monitor rule is the time taken for creating a compute instance in OpenStack is less than 4000ms, the rule is written in Event Calculus as follow:

Happen(compute.instance.create.start(instance_id,user_id), t1, R (t1,t1) ^ Happen(compute.instance.create.end(instance_id,user_id), t2, R(t1,t2) => Relation((t2 – t1) < 4000)

And written in xml file using EC-Assertion language as below:

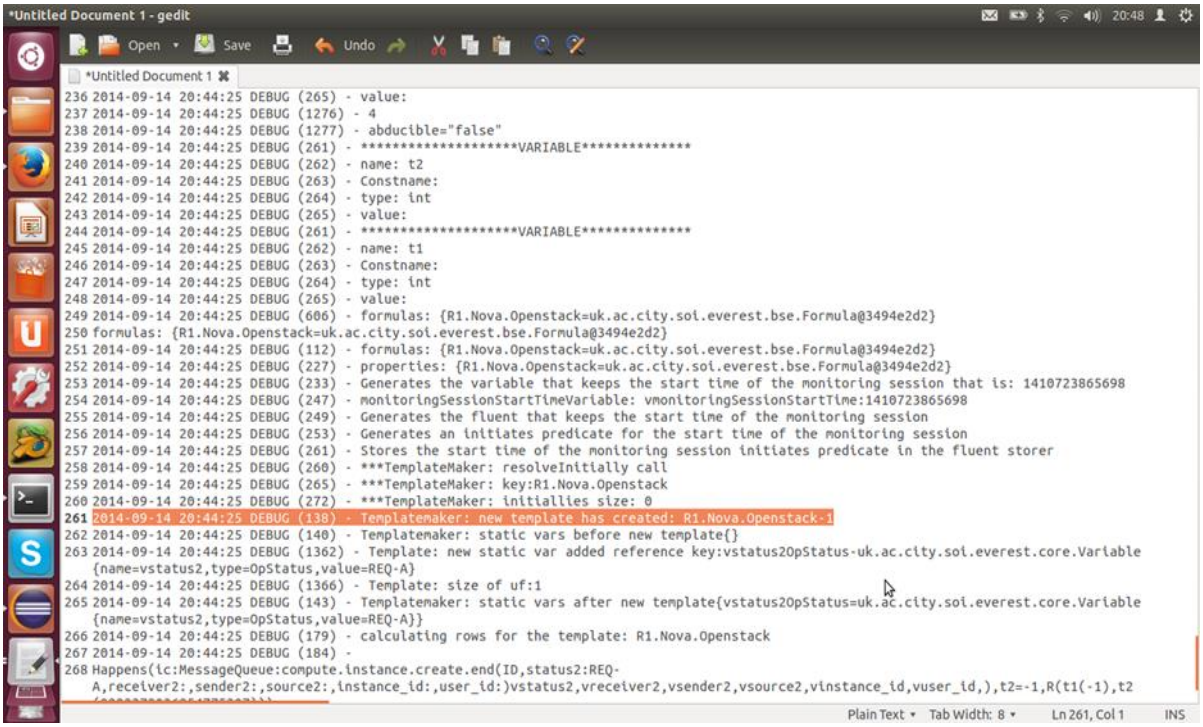
```
<formula xmlns="" diagnosisRequired="false" forChecking="true"
formulaId="R1.Nova.Openstack" threatDetectionRequired="false"
type="Future_Formula">
    <quantification>
        <quantifier>forall</quantifier>
        <timeVariable>
            <varName>t1</varName>
            <varType>TimeVariable</varType>
        </timeVariable>
    </quantification>
    <quantification>
        <quantifier>existential</quantifier>
         <timeVariable>
             <varName>t2</varName>
             <varType>TimeVariable</varType>
         </timeVariable>
    </quantification>
    <body>
        <predicate abducible="false" negated="false" recordable="true"
unconstrained="true">
            <happens>
                <ic_term>

<operationName>compute.instance.create.start</operationName>
                    <partnerName>MessageQueue</partnerName>
                    <variable forMatching="true" persistent="false">
                        <varName>status1</varName>
                        <varType>OpStatus</varType>
                        <value>REQ-B</value>
                    </variable>
                    <variable forMatching="true" persistent="false">
                        <varName>sender1</varName>
                        <varType>string</varType>
                        <value>
                        </value>
                    </variable>
                    <variable forMatching="true" persistent="false">
                        <varName>receiver1</varName>
                        <varType>string</varType>
                        <value>
                        </value>
                    </variable>
                    <variable forMatching="true" persistent="false">
                        <varName>source1</varName>
                        <varType>string</varType>
                        <value>
                        </value>
                    </variable>
```

```
                    <variable forMatching="true" persistent="false">
                        <varName>instance_id</varName>
                        <varType>string</varType>
                        <value>
                        </value>
                    </variable>
                    <variable forMatching="true" persistent="false">
                        <varName>user_id</varName>
                        <varType>string</varType>
                        <value>
                        </value>
                    </variable>
                </ic_term>
                <timeVar>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                </timeVar>
                <fromTime>
                    <time>
                        <varName>t1</varName>
                        <varType>TimeVariable</varType>
                    </time>
                </fromTime>
                <toTime>
                    <time>
                        <varName>t1</varName>
                        <varType>TimeVariable</varType>
                    </time>
                </toTime>
            </happens>
        </predicate>
        <operator>and</operator>
        <predicate abducible="false" negated="false" recordable="true"
unconstrained="false">
            <happens>
                <ic_term>
                    <operationName>compute.instance.create.end</operationName>
                    <partnerName>MessageQueue</partnerName>
                    <variable forMatching="true" persistent="true">
                        <varName>status2</varName>
                        <varType>OpStatus</varType>
                        <value>REQ-A</value>
                    </variable>
                    <variable forMatching="true" persistent="false">
                        <varName>receiver2</varName>
                        <varType>string</varType>
                        <value>
                        </value>
                    </variable>
                    <variable forMatching="true" persistent="false">
                        <varName>sender2</varName>
                        <varType>string</varType>
                        <value>
                        </value>
                    </variable>
                    <variable forMatching="true" persistent="false">
                        <varName>source2</varName>
                        <varType>string</varType>
                        <value>
```

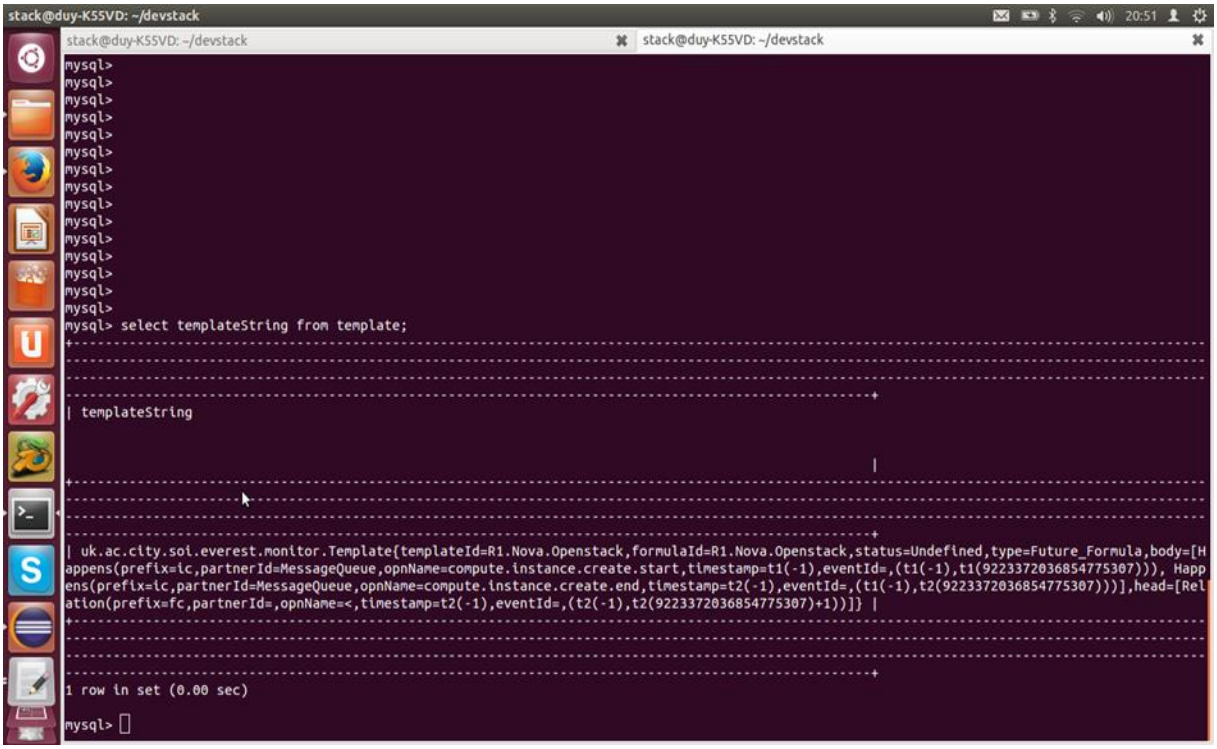
```
                    </value>
                </variable>
                <variable forMatching="true" persistent="false">
                    <varName>instance_id</varName>
                    <varType>string</varType>
                    <value>
                    </value>
                </variable>
                <variable forMatching="true" persistent="false">
                    <varName>user_id</varName>
                    <varType>string</varType>
                    <value>
                    </value>
                </variable>

            </ic_term>
            <timeVar>
                <varName>t2</varName>
                <varType>TimeVariable</varType>
            </timeVar>
            <fromTime>
                <time>
                    <varName>t1</varName>
                    <varType>TimeVariable</varType>
                </time>
            </fromTime>
            <toTime>
                <time>
                    <varName>t2</varName>
                    <varType>TimeVariable</varType>
                </time>
            </toTime>
        </happens>
    </predicate>
</body>
<head>
    <relationalPredicate>
        <lessThan>
            <operand1>
                <operationCall>
                    <name>sub</name>
                    <partner>self</partner>
                    <variable persistent="false" forMatching="false">
                        <varName>t2</varName>
                        <varType>int</varType>
                    </variable>
                    <variable persistent="false" forMatching="false">
                        <varName>t1</varName>
                        <varType>int</varType>
                    </variable>
                </operationCall>
            </operand1>
            <operand2>
                <constant>
                    <name>time</name>
                    <value>4000</value>
                </constant>
            </operand2>
        </lessThan>
```

```
                <timeVar>
                <varName>t2</varName>
                <varType>TimeVariable</varType>
                </timeVar>
        </relationalPredicate>
    </head>
</formula>
```

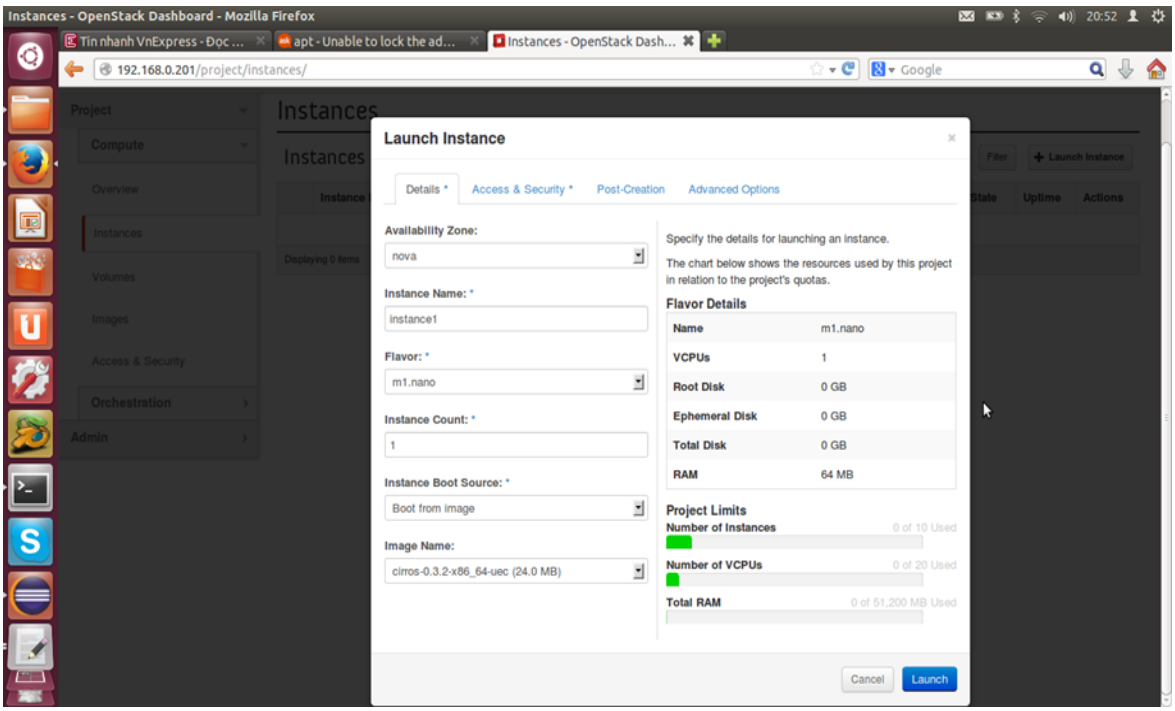
After finish defining the rule, we have submitted the rule to the software by specifying the xml file's path and the software has fed the rule to EVEREST successfully. The logs file below showing the rule feeding result:



As you can see, the log showed that a new template has been created for our rule R1.Nova.OpenStack. A quick check to the database to make sure the rule is really saved.



By looking at the database, we have made sure that a new rule with id R1.Nova.Openstack has been created and ready for monitoring activity. In the next step, we performed creating instance action to check if our software can detect if the time taken for this action is satisfied our new rule. First, a new instance is created using OpenStack's dashboard.

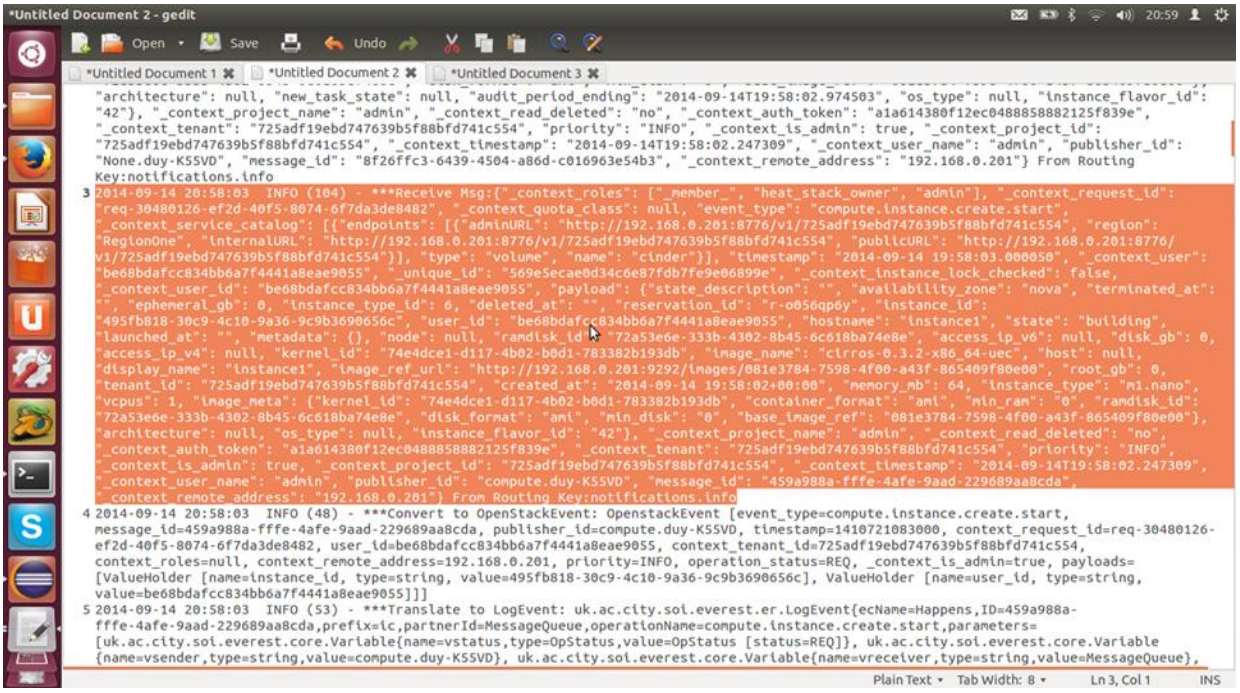


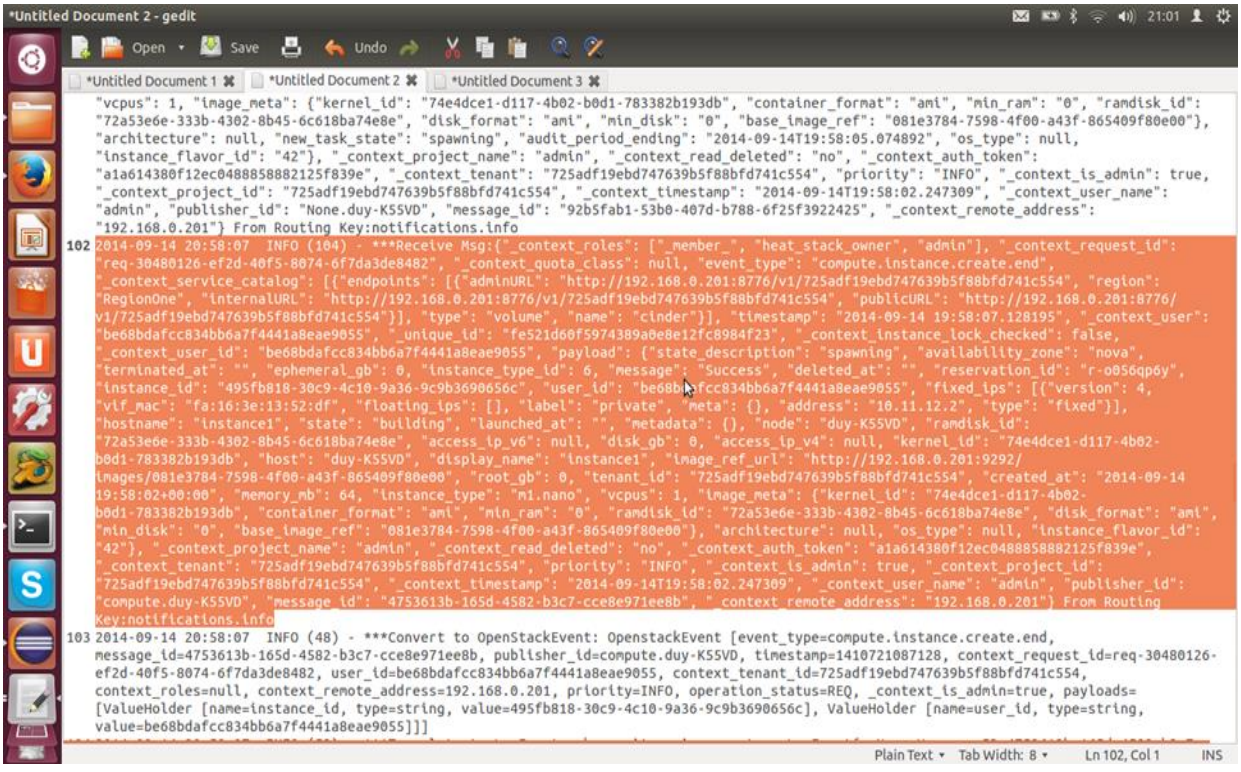
A new instance has been created by OpenStack, we now check if our software works as expected. When creating an instance, OpenStack will emit 2 important notification messages:

- *Comput.instance.create.start* message: this message notifies the start of creating instance operation and includes its start timestamp in the message.

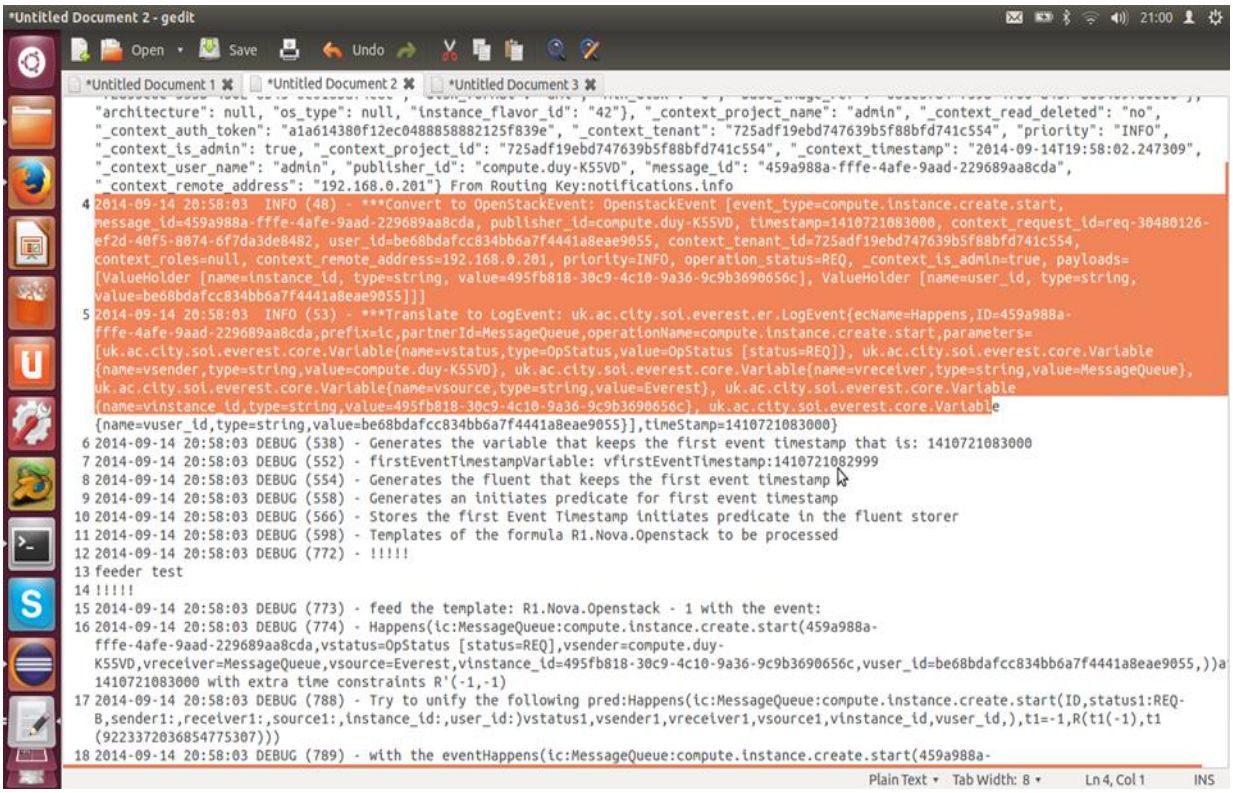
- *Comput.instance.create.end* message: this message notifies the end of creating instance operation and includes its end timestamp in the message.

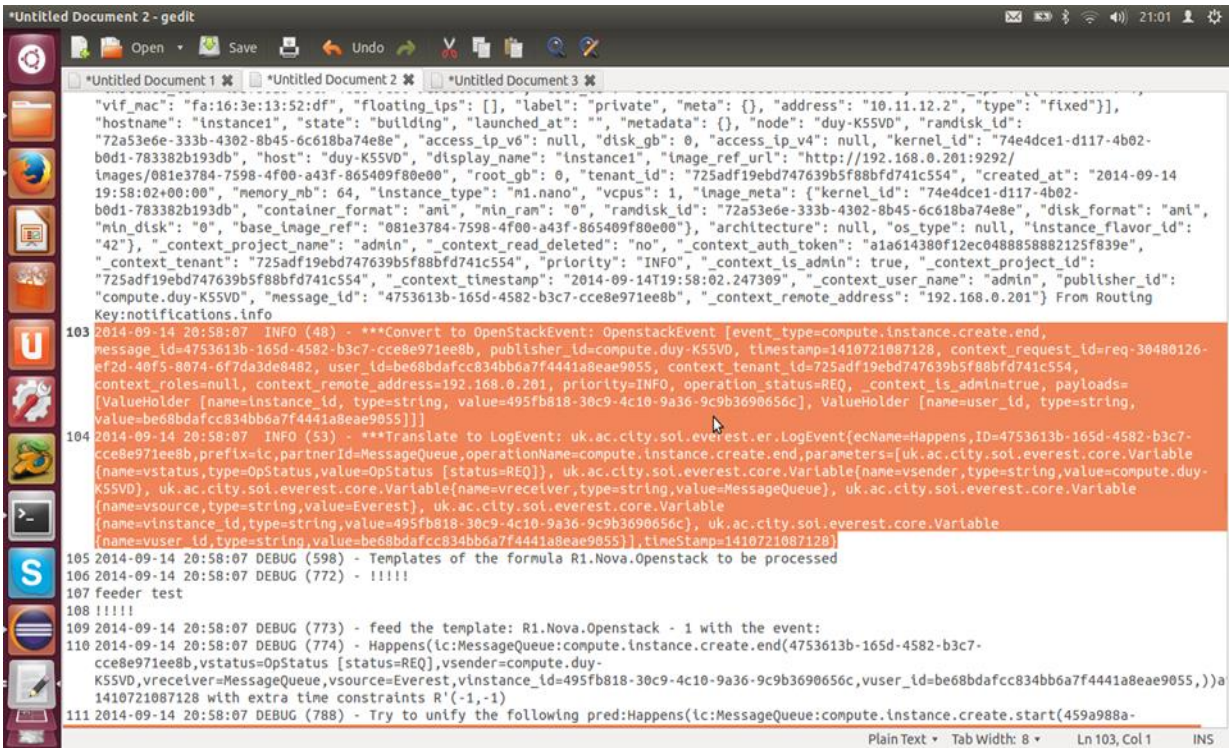
Our software should be able to catch those two messages and convert them to LogEvent object. The following screenshots shows the actual result:





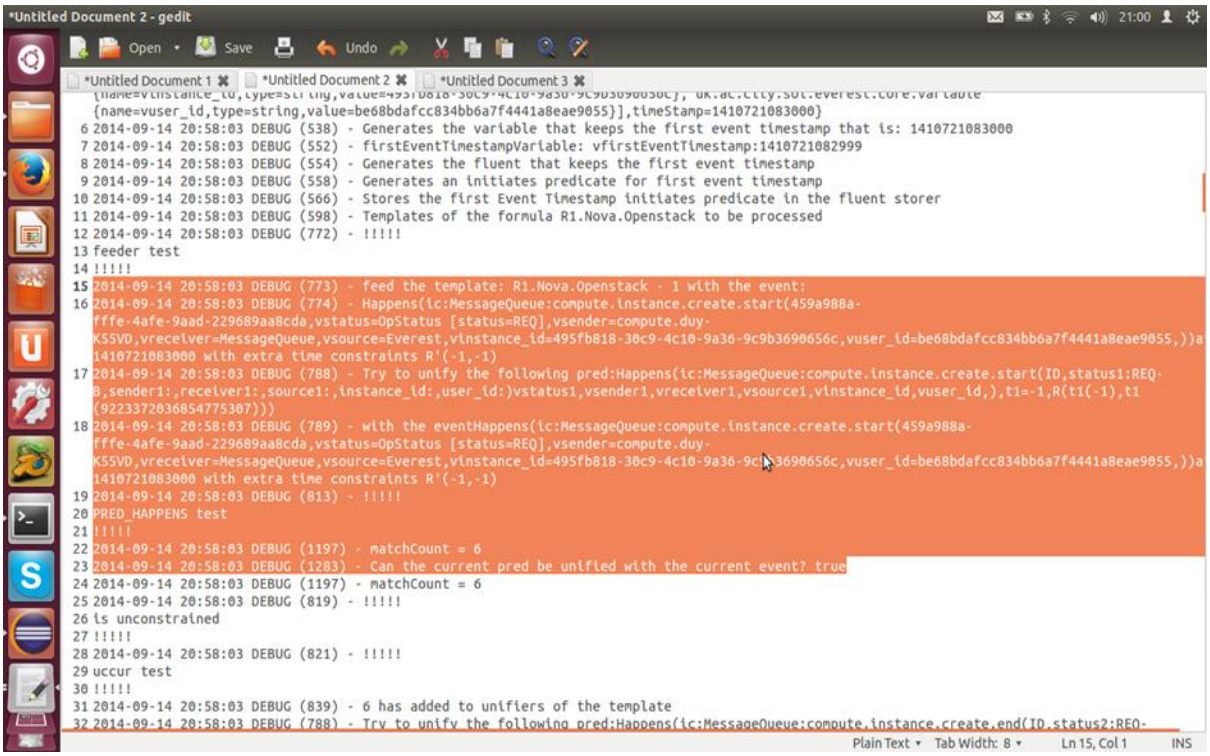
From the result above, we can confirm that our software able to capture message from OpenStack using RabbitEventCapturer. However, it should be able to convert the raw messages to LogEvent objects. By examining the log, we can check to see if the software works as expected. The screenshot below shows the actual result:



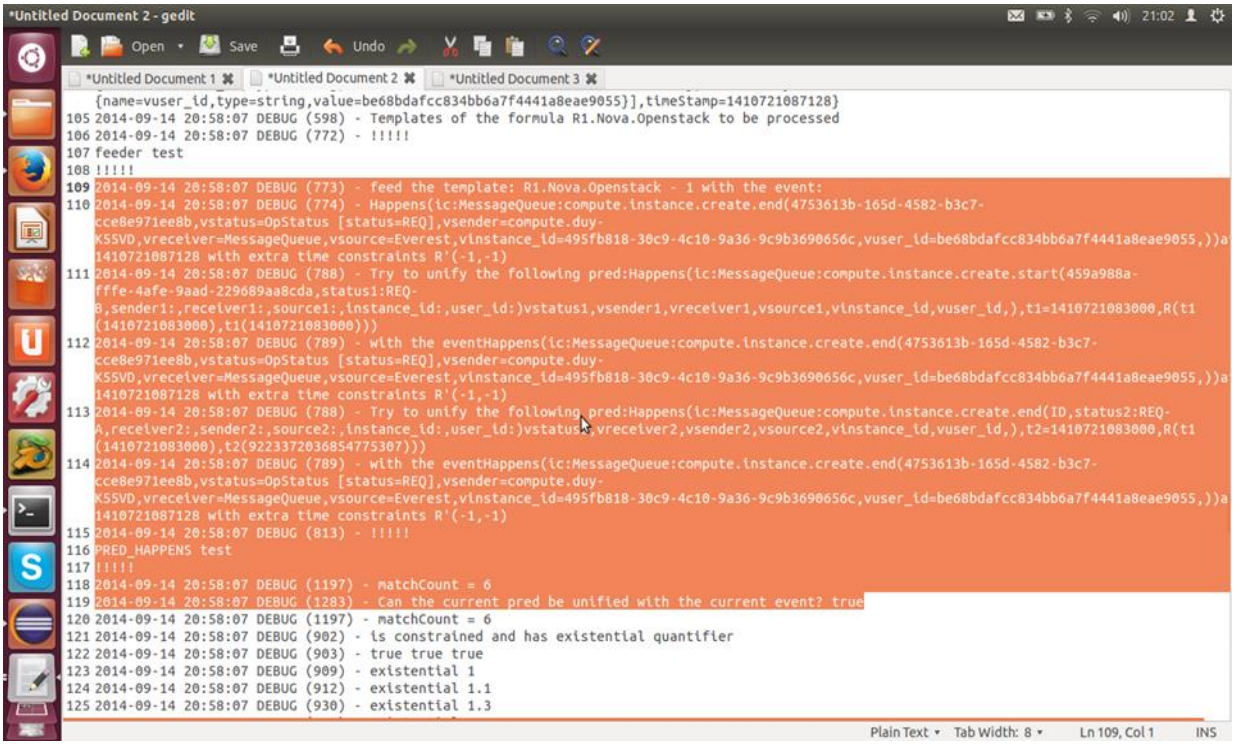


The *compute.instance.create.start* and *compute.instance.create.end* messages from OpenStack have been converted to LogEvent objects successfully. After receiving event, EVEREST will try

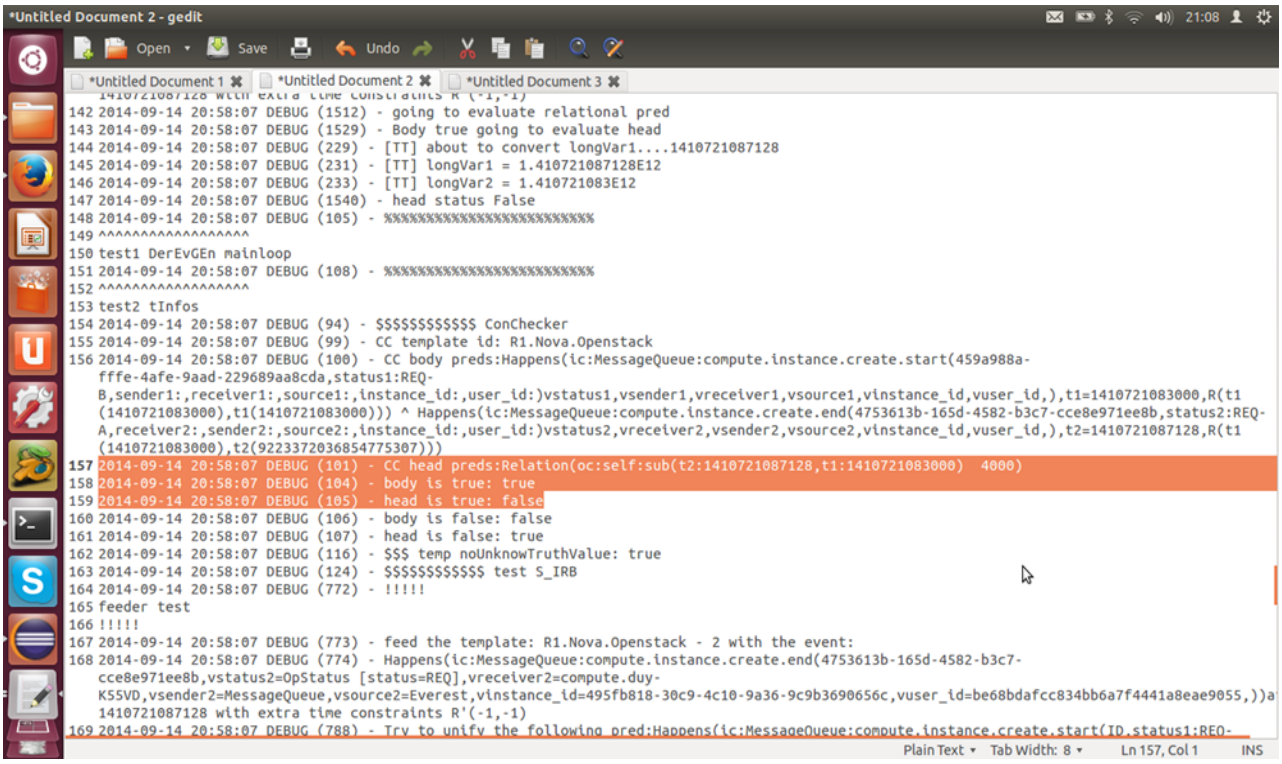
to unify the event with the set of rule it has to check the rule associate with this event. The following screenshot shows the unified result for *compute.instance.create.start* event:



According to the screenshot above, the unification between event *compute.instance.create.start* and rule template R1.Nova.OpenStack's *Happen(compute.instance.create.start(…))* predicate is true. Therefore, rule template R1.Nova.OpenStack is associated with *compute.instance.create.start* event. EVEREST now waits for *compute.instance.create.end* event message to determine if the monitor rule is violated. The below screenshot shows the unification result of *compute.instance.create.end* event:



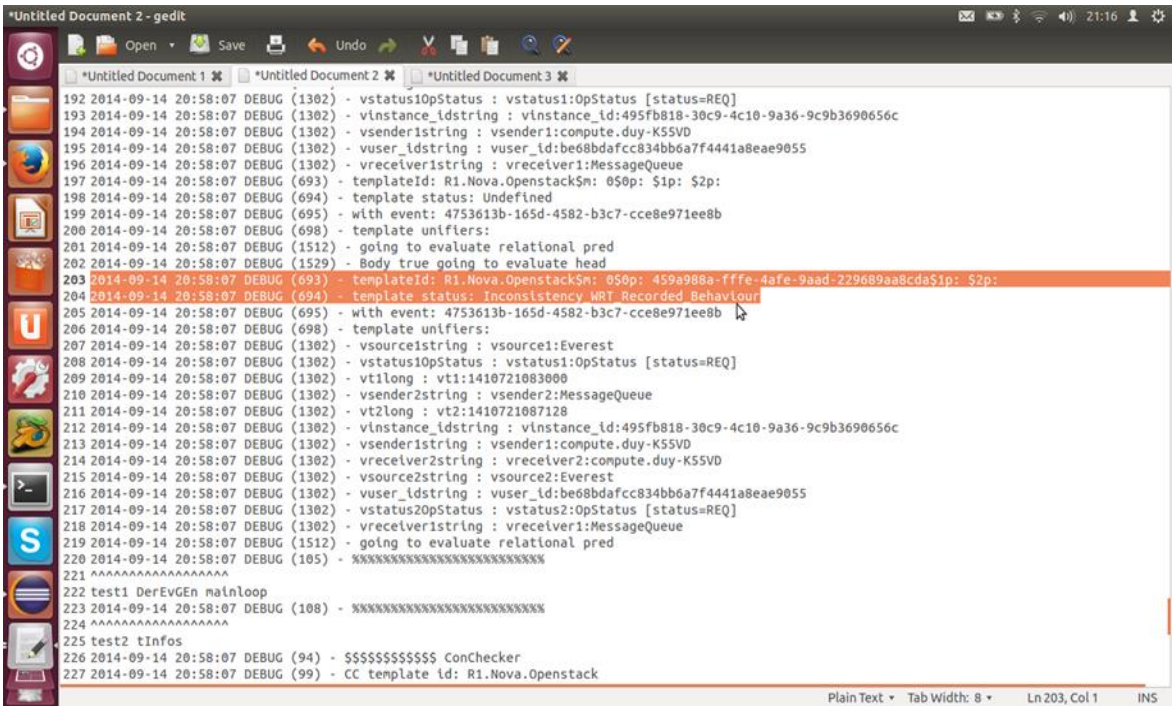
The unification between event *compute.instance.create.end* with rule R1.Nova.Openstack's *Happen(compute.instance.create.end(...))* predicate is true. The next step is determining if the monitor properties is violated. EVEREST will check the head part of the rule to determine if it's true.



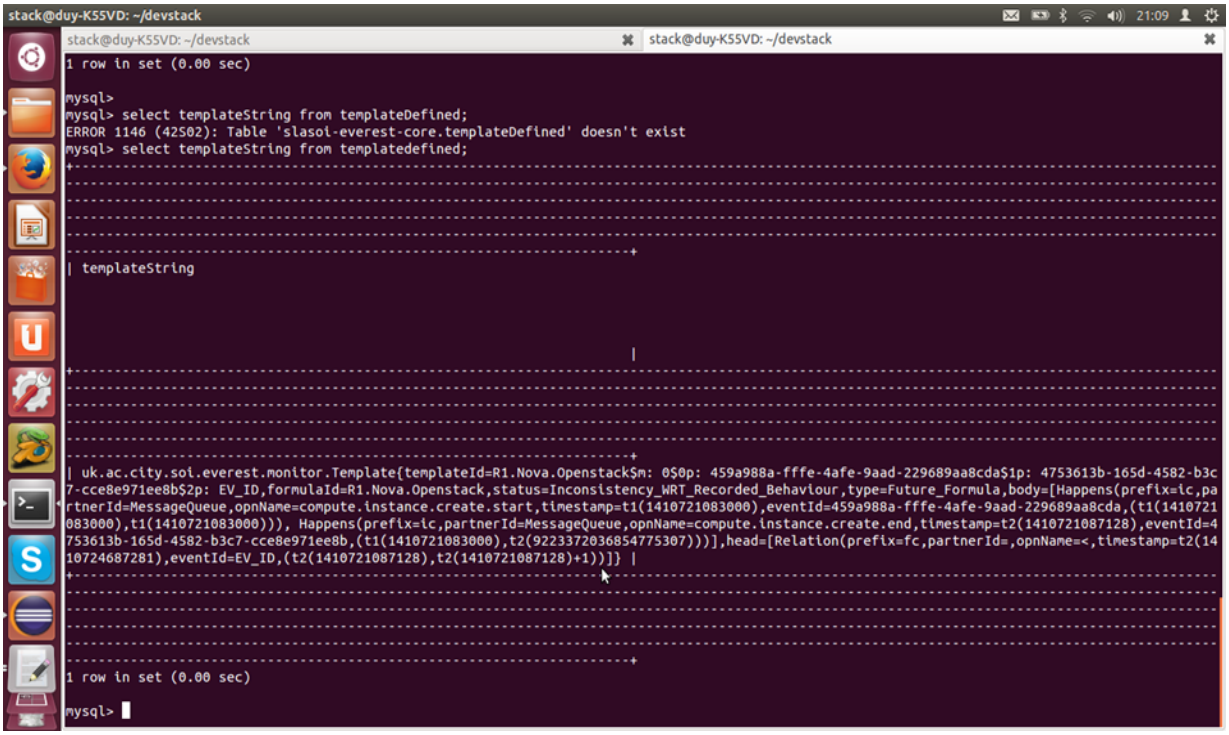
The timestamp t1 of *compute.instance.create.start* event is 1410721083000 and timestamp t2 of *compute.instance.create.end* event is 1410721087128 and the time different between them is:

(1410721087128 – 1410721083000) = 4128ms

Which is bigger than 4000ms, therefore the result of head predicate *Relation((t2 – t1) < 4000)* is false. This mean the rule has been violated. The screenshot below show the result determined by our software.



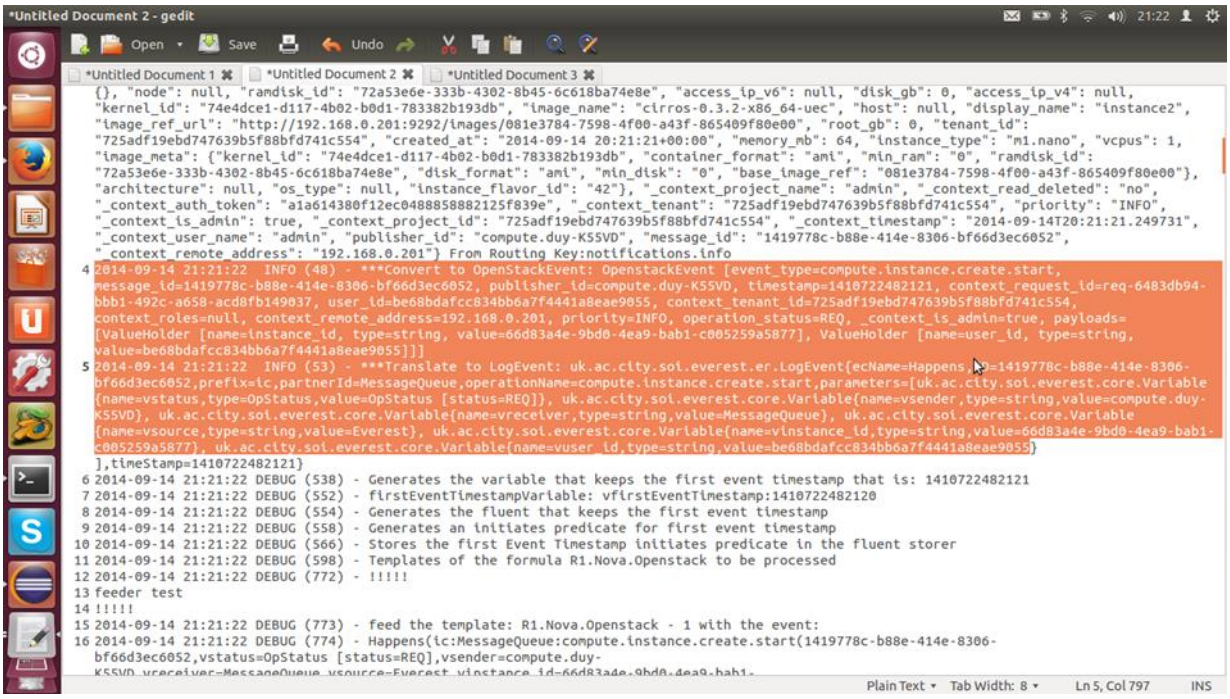
The below screenshot shows template result recorded in MySQL database.



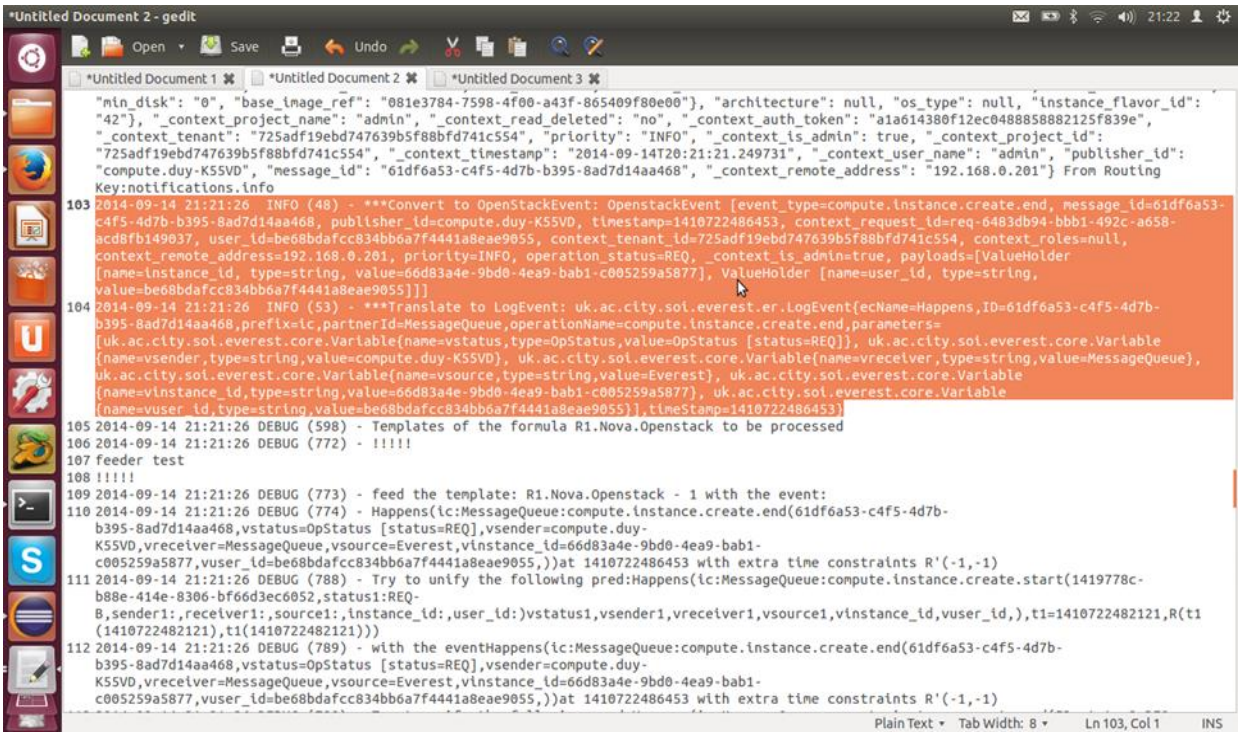
The rule template has been recorded in database as violated with the event id of events that caused the violation. Now we checked how our software reacts when events that satisfies

monitor rule condition occurs. First, we modified the rule condition to be less strict. The rule is time taken for creating compute instance in OpenStack is less than 10.000ms instead of 4000ms.

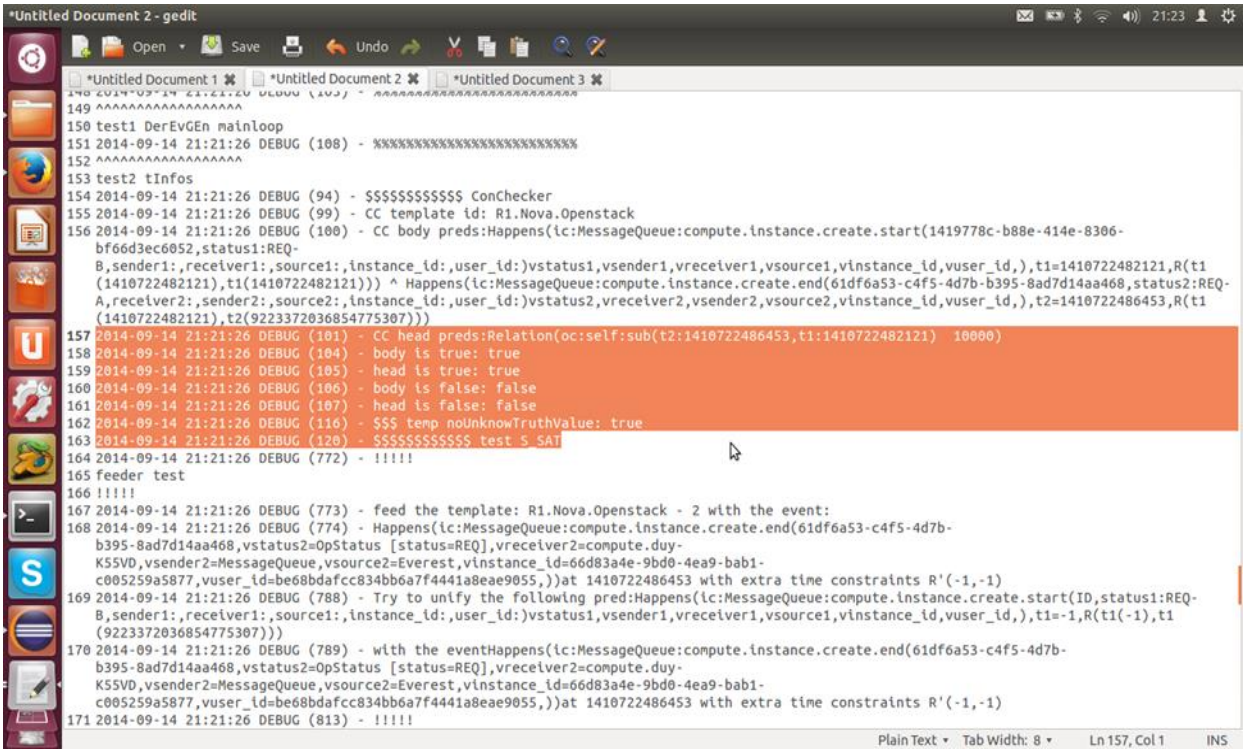
Now we performed creating instance action and check if the action satisfies the new rule. The screenshot below shows 2 new events captured by our software:



And



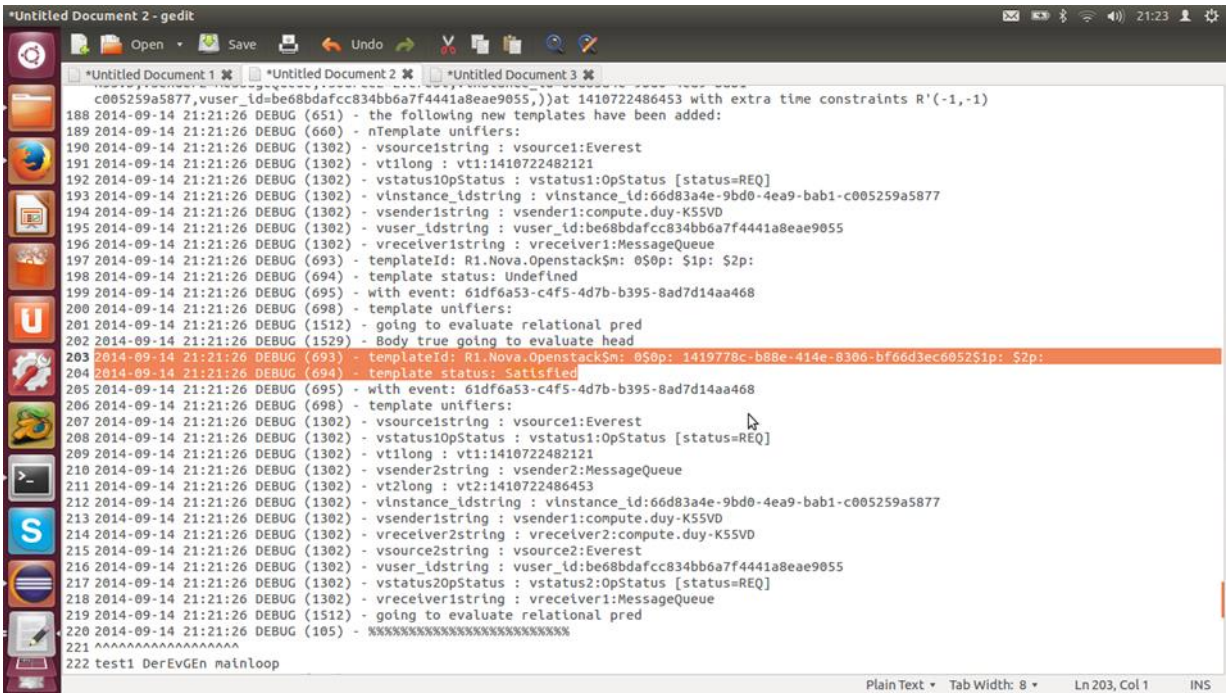
We now check to see if the action satisfies the new rule.



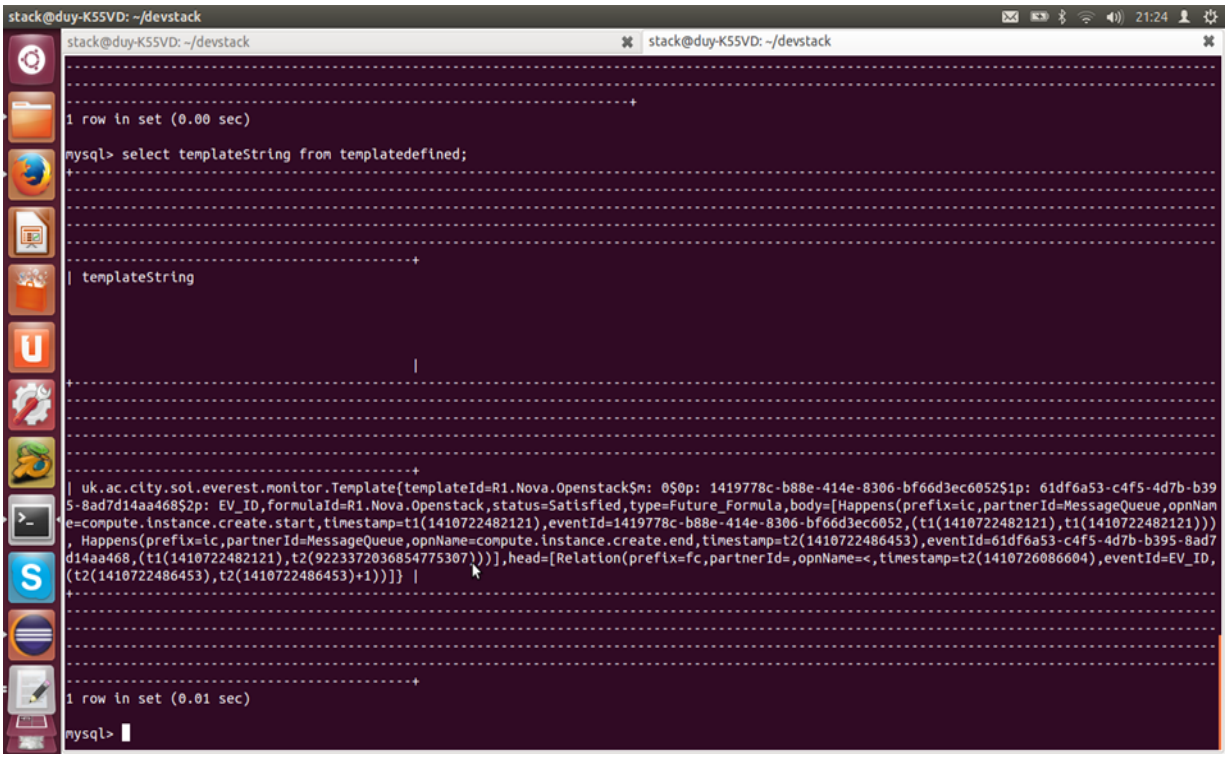
The timestamp t1 of *compute.instance.create.start* event is 1410722482121 and timestamp t2 of *compute.instance.create.end* event is 1410722486453 and the time different between them is:

(1410722486453 – 1410722482121) = 4332ms

Which is smaller than 10000ms, therefore the result of head predicate *Relation((t2 – t1) < 10000)* is true. This mean the rule has been satisfies. The screenshot below show the result determined by our software.



The screenshot above shows that EVEREST determines the rule is satisfied by these events. We will now check the template result recorded in database:

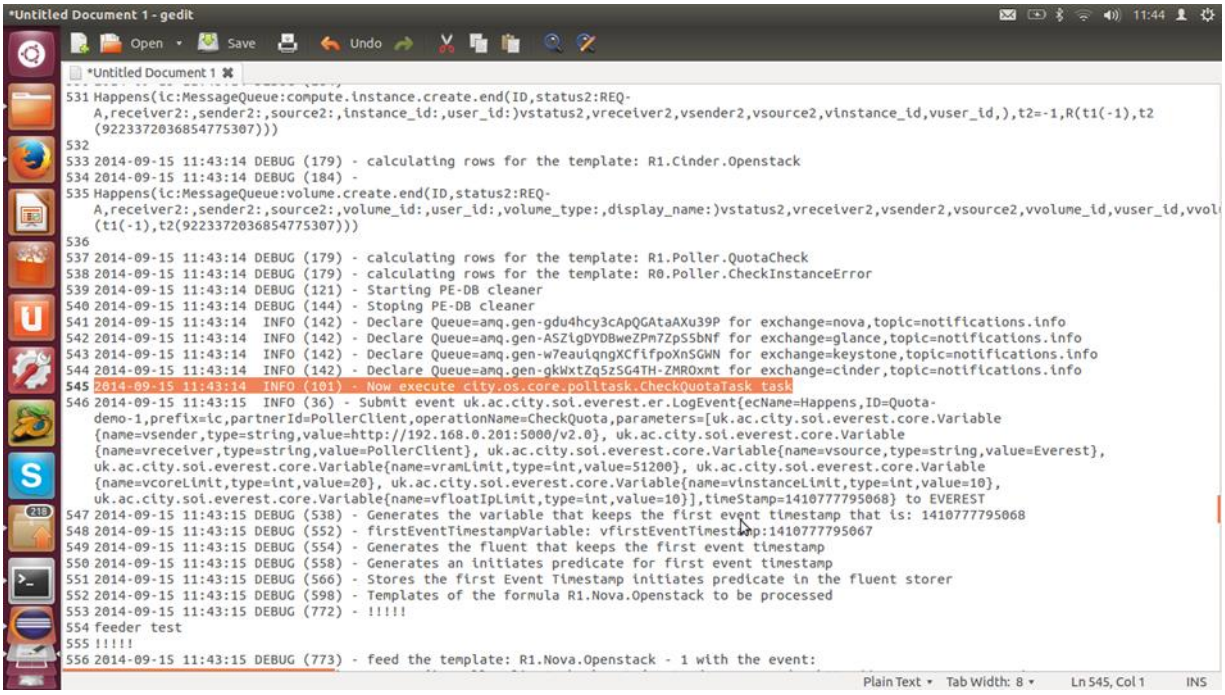


The rule template has been recorded in database as Satisfied with the id of events that satisfy the rule.

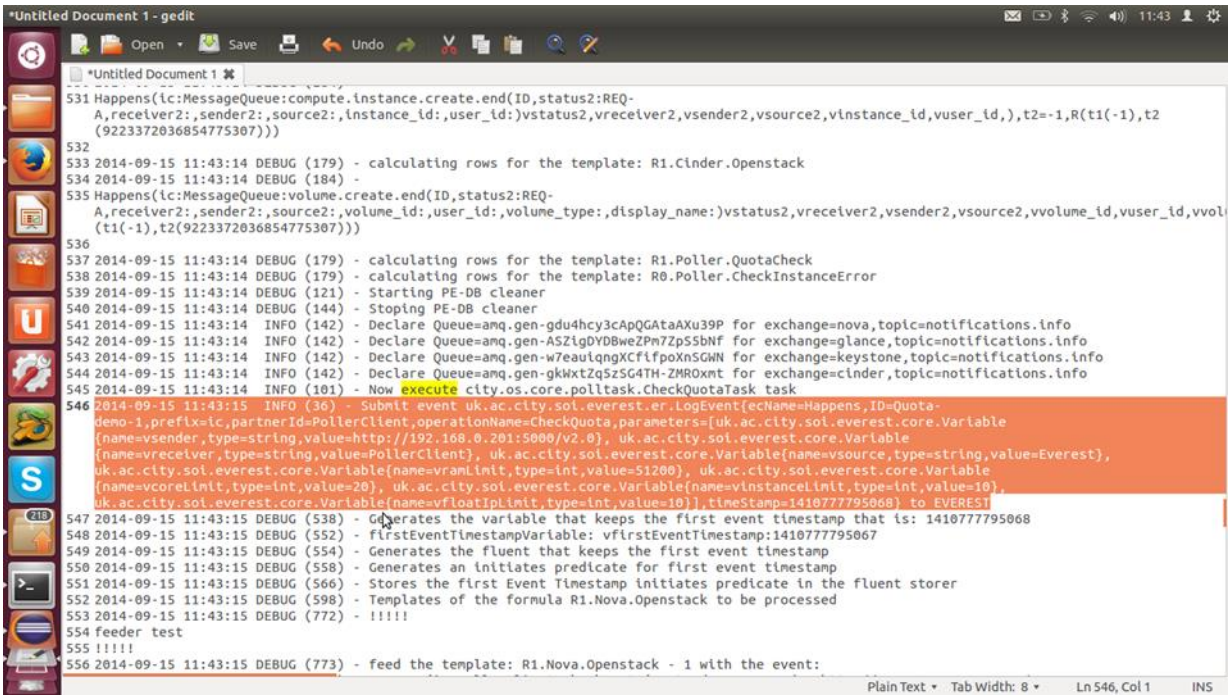
2. Using Poller Result

Next we checked how our software works using the Poller component to query information from OpenStack; we have developed a rule template for checking the outcome of our software. The rule is the quota for tenant demo is always 51200 Mb Ram, 20 CPUs, 10 instances and 10 float Ip Addresses.

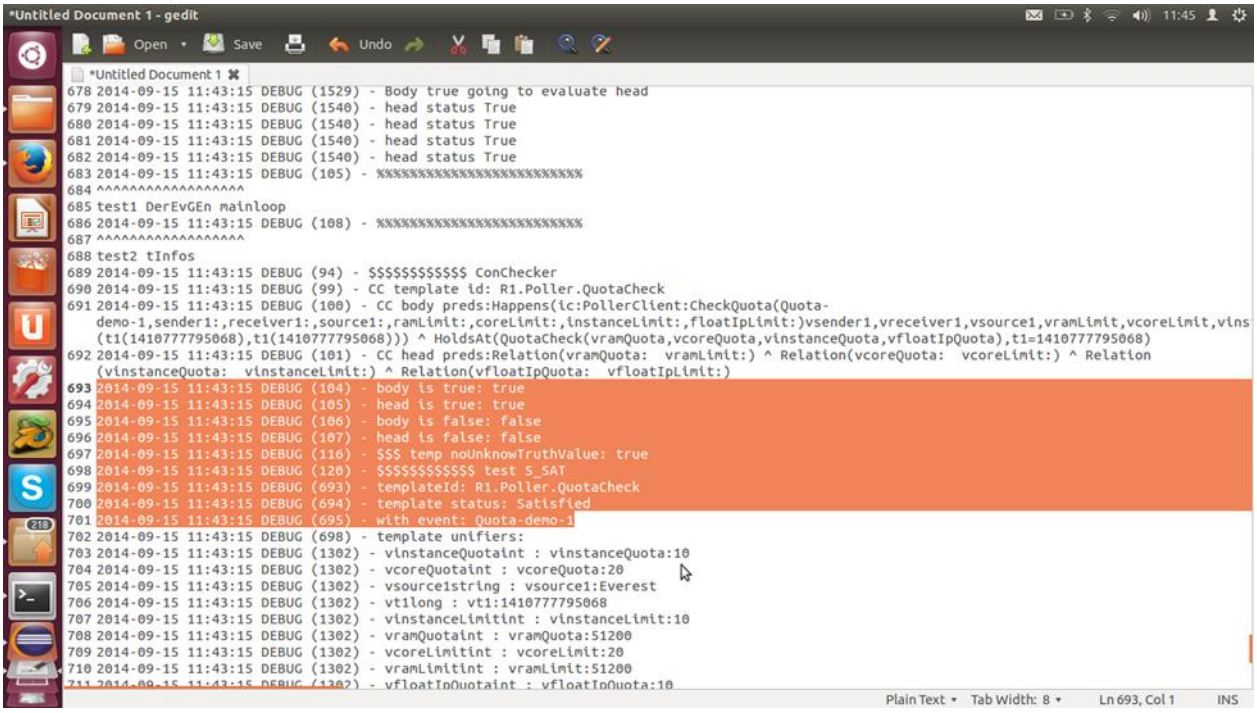
When starting our software, the CheckQuotaTask will be called periodically to obtain information about user demo's quota from OpenStack. The screenshot below showing CheckQuotaTask has been executed by our software.



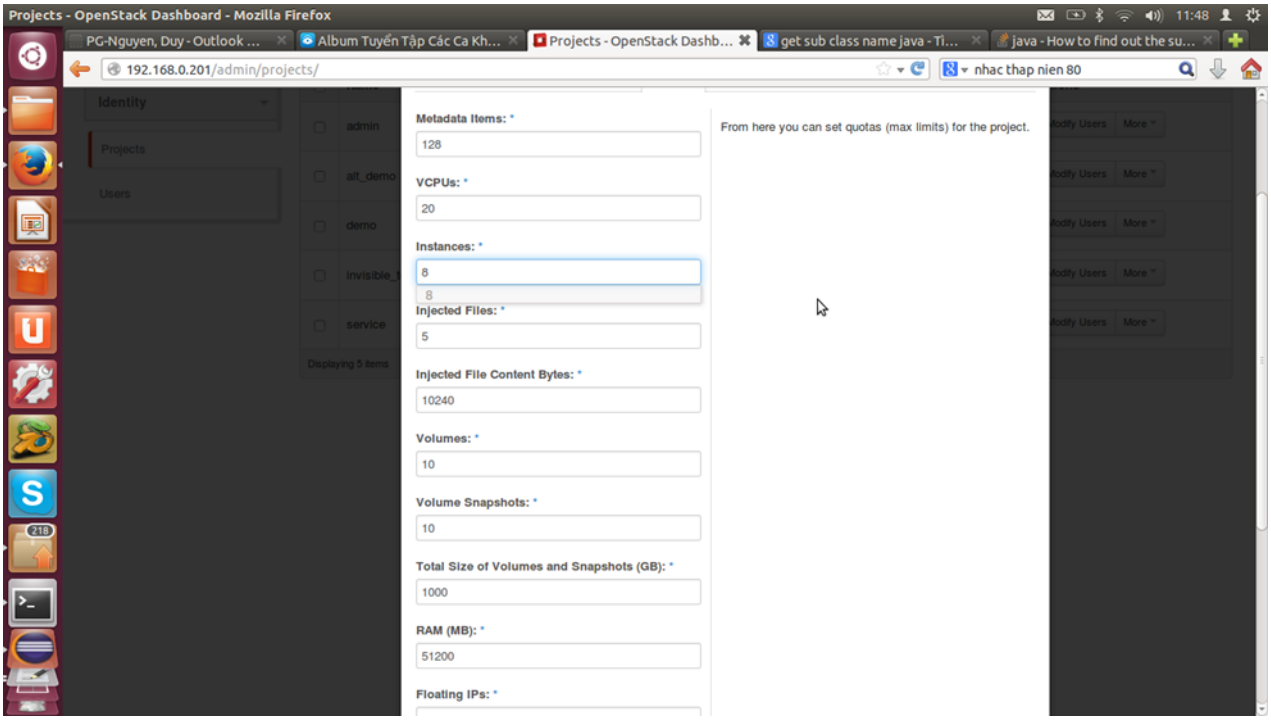
After receiving information from OpenStack, our software has extracted necessary datas and creates a LogEvent object and submit it to EVEREST. The below screenshot shows the event created by CheckQuotaTask.



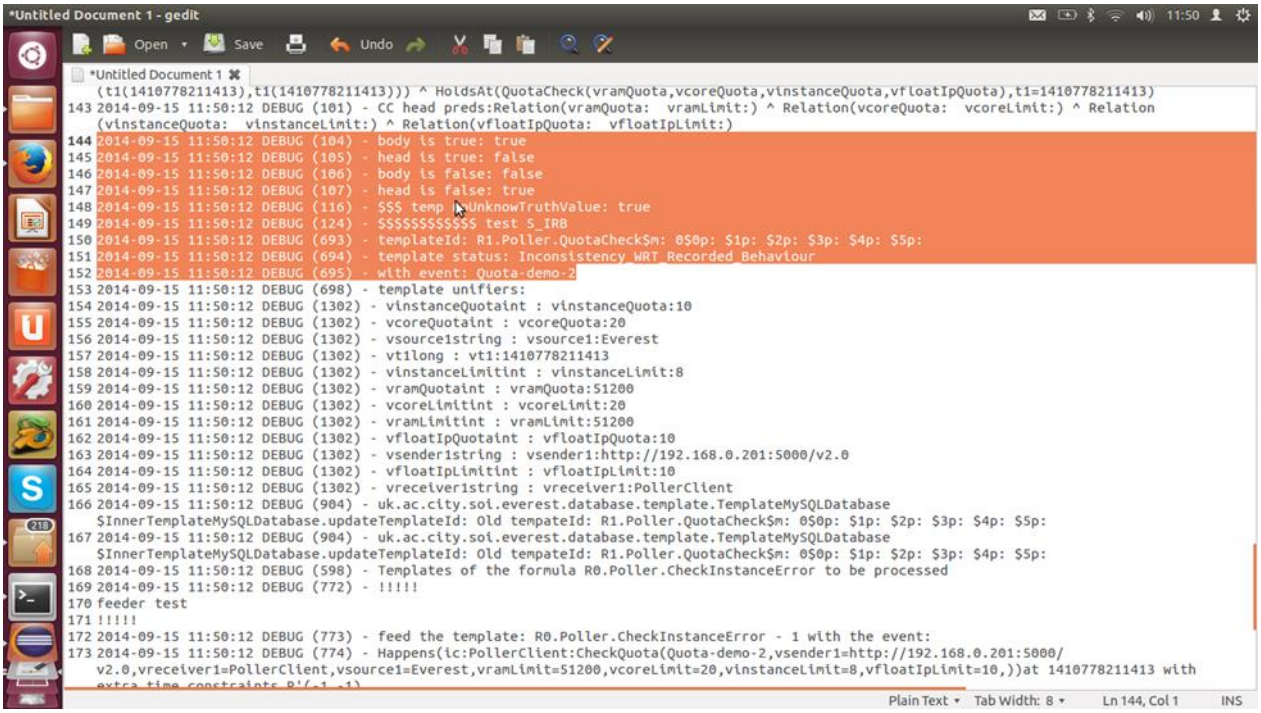
Since demo user was created with correct quota limit, the result for this event should be no violation, the rule has been satisfied. The screenshot below shows that our software has correctly determined the event as Satisfied.



Now, we changed the quota of user demo in order to causes a rule violation, the instance limit of user demo will be lowered to 8 instances only.



After 1 minutes (the polling interval), the CheckQuotaTask would be executed again, and the new event contain the user's quota would be submitted to EVEREST. The screenshot below shows the result of the second events.



Based on the screenshot above, our software has determined the result correctly. The new event has indeed caused a rule violation.

3.4.2. Event Converter Problem

First, we will discuss what problem we have faced when trying to convert OpenStack's event to EVEREST's event. After that, we will present the solution to this problem. This solution has been implemented in this software.

The Problem:

When EVEREST receives an event e, it will try to find a rule template r that can unified with this event. In order to decide if an event e can associate with a rule r, it will compare r's predicates's event signature with event e's signature. For example:

Predicate *Happen(**create_instance(instanceId, tenantId, host)**, t, R(t1,t2))* and *event(id, sender, receiver, status, **create_instance(instanceId,ipAddress),** source)* cannot be unified because the signature of the predicate's event is not the same as the event's signature.

Therefore, in order to unify the event from OpenStack with the monitor rule, the rule's predicates must have the same signature with OpenStack's event. Which mean for each monitor rule, the predicates that associate with an event must contain all the OpenStack's event data. Since each OpenStack event come with a lot of information such as this one below:


```
{"_context_roles": ["heat_stack_owner", "admin"], "_context_request_id": "req-
8e316916-0487-45ae-bcca-6e3c27fc5f65", "_context_quota_class": null,
"event_type": "compute.instance.update", "_context_service_catalog":
[{"endpoints": [{"adminURL":
"http://192.168.0.201:8776/v1/84c67e910ff14dfa96ce16cd6f49ae6a", "region":
"RegionOne", "internalURL":
"http://192.168.0.201:8776/v1/84c67e910ff14dfa96ce16cd6f49ae6a", "publicURL":
"http://192.168.0.201:8776/v1/84c67e910ff14dfa96ce16cd6f49ae6a"}], "type":
"volume", "name": "cinder"}], "timestamp": "2014-09-15 10:52:48.314443",
```

```
"_context_user": "37bf25ee50564aa5800b6cba94123385", "_unique_id":
"22b7eacf774c44daa95baeea56eca64f", "_context_instance_lock_checked": false,
"_context_user_id": "37bf25ee50564aa5800b6cba94123385", "payload":
{"state_description": "scheduling", "availability_zone": "nova",
"terminated_at": "", "ephemeral_gb": 0, "instance_type_id": 6, "bandwidth":
{}, "deleted_at": "", "reservation_id": "r-g1pxxcnm", "instance_id":
"795928bf-2f8c-4633-a6f9-ff04fafc8d63", "user_id":
"37bf25ee50564aa5800b6cba94123385", "hostname": "instance3", "state":
"building", "old_state": null, "old_task_state": null, "metadata": {}, "node":
null, "ramdisk_id": "de54a859-a5a2-4cf0-9af0-6bfb7b0a628e", "access_ip_v6":
null, "disk_gb": 0, "access_ip_v4": null, "kernel_id": "6599fd74-571e-4dac-
9fff-9041a8558ffe", "host": null, "display_name": "instance3",
"image_ref_url": "http://192.168.0.201:9292/images/4c6fa680-62e4-4b13-a7cf-
290e1566baa5", "audit_period_beginning": "2014-09-15T10:00:00.000000",
"root_gb": 0, "tenant_id": "84c67e910ff14dfa96ce16cd6f49ae6a", "created_at":
"2014-09-15 10:52:48.201649+00:00", "launched_at": "", "memory_mb": 64,
"instance_type": "m1.nano", "vcpus": 1, "image_meta": {"kernel_id": "6599fd74-
571e-4dac-9fff-9041a8558ffe", "container_format": "ami", "min_ram": "0",
"ramdisk_id": "de54a859-a5a2-4cf0-9af0-6bfb7b0a628e", "disk_format": "ami",
"min_disk": "0", "base_image_ref": "4c6fa680-62e4-4b13-a7cf-290e1566baa5"},
"architecture": null, "new_task_state": "scheduling", "audit_period_ending":
"2014-09-15T10:52:48.309519", "os_type": null, "instance_flavor_id": "42"},
"_context_project_name": "demo", "_context_read_deleted": "no",
"_context_auth_token": "648847549abcbb1dfca559ba6fd21b9f", "_context_tenant":
"84c67e910ff14dfa96ce16cd6f49ae6a", "priority": "INFO", "_context_is_admin":
true, "_context_project_id": "84c67e910ff14dfa96ce16cd6f49ae6a",
"_context_timestamp": "2014-09-15T10:52:47.744626", "_context_user_name":
"admin", "publisher_id": "api.duy-K55VD", "message_id": "09e0f288-37a3-4adf-
9068-f1c810de0516", "_context_remote_address": "192.168.0.201"}
```


The user will have to manually write all those info as the predicate's variables in the rule file. It's time consuming and ineffective, since for some monitor rules, user only need to monitor some properties such as to monitor instance status, the user only need to specify the status variable or to monitor the capacity of the instance, user only need to specify the ram, memory as predicate's variable. However, in this case, user has to specify all the information of the event as variable in order to unify the event with the rule.

In order to avoid this problem, we need to find a solution to let the user specify only the properties they are interested in the event instead of all the properties in the event.

The Solution:

We found a solution to this problem using an event template. Event template is an xml file that enables the user to specify what information they are interested in an event. For example we have event type e1(a,b,c,d,e,f,g,z,j,k, etc.) and rule r1 that want to unify with any event type e1 but it only interested in property b,d,f. We can create an event template that specify in event type e1 we only interest in b,d,f as below:

```
<template>
    <eventType>e1</eventType>
    <operationStatus>REQ</operationStatus>
    <parameters>
        <parameter>
            <name>b</name>
            <type>string</type>
        </parameter>
        <parameter>
            <name>d</name>
            <type>string</type>
        </parameter>
        <parameter>
            <name>f</name>
            <type>string</type>
        </parameter>
    </parameters>
</template>
```

When EventConverter encounter any event type e1, it will extract only b,d,f information and create an OpenStackEvent object contain those information. After that EverestTranslator will translate that object to LogEvent object and submit the event to EVEREST. As a result, EVEREST only think that it has receive event e1(b,d,f) instead of e1(a,b,c,d,e,f,g,z,j,k, etc.) and unify that event with rule r1: Happen(e1(b,d,f)) successfully.

With this solution, user doesn't need to write all the information contained in an event when specifying their rule.

3.4.3. Conclusion Remarks:

In conclusion, by following the design diagram, we successfully implemented software. By providing some output evidence, we can confirm that our software has worked. However, testing need to be conducted in order to detect any defect and ensure the quality of our software. The testing result determined if our software has met the project's requirement.

3.5. Testing Result

In order to verify outcome of our software, 4 rule templates has been defined. We fed these 4 rules to EVEREST and did various kind of testing to check if our software works correctly. The subsection below will show the result of our testing.

3.5.1. Unit Testing

In unit testing, I have developed a set of test classes to ensure the class's method work as expected. The following test classes have been written to ensure our classes working as expected:

- **EventConverterTest**: this class test the convert() method of EventConverter class. By creating a dummy event follow OpenStack's compute.instance.create.start event format, we can compare the result OpenStackEvent object produced by the method with an expected OpenStackEvent object that we have prepared. The result from this test is **Pass**.
- **EverestTranslatorTest**: this class test the translate() method of EverestTranslatorTest class. By creating a dummy OpenStackEvent object, we compare the result LogEvent

object produced by translate() method with an expected LogEvent object that we have prepared. The result from this test is **Pass**.

Since the logic for rule and event processing are resided in EVEREST and have been tested by EVEREST's developers, we will not write test method for them. However, unit tests for components in Polling module haven't been developed yet. The reason is because we only have limited time left. Further work for improving the unit test suite will be implemented in the future.

3.5.2. Integration Testing

The aim of integration test is to ensure our components work together to realize a functional requirement as expected. The integration testing is performed using both JUnit test classes and manual testing. For JUnit test classes, we have developed a testcase to check if the EventConverter and EverestTranslator can work together to convert OpenStack's event to LogEvent object. The test case is:

- **CapturerTest:** this class compare the LogEvent object created by the integration between EventConverter and EverestTranslator with an expected LogEvent object. The result from this test is **Pass** and satisfies non-functional requirement **NF-1**.

We have performed some manual testing to check if the integration between components can satisfies the functional requirement. Functional requirements **F-1,F-2,F-3,F-4,NF-2** and **F-5** has been tested using the Pass Criteria in the requirement tables and the results are pass. The Result Output demonstrated in section **4.4.1** showing a sample result of our testing.

3.5.3. System Testing

In system testing, the software has been left running for one day. We did various kind of action such as creating instances, creating volumes, change user quota limit and checked if the software can detect any rule violations. The result of this phase is passed. Our software can capture events, query for information and detect violations based on the information provided.

3.5.4. Performance Testing

Performance testing is done by developed a set of performance JUnit test classes. We created test classes to measure performance of EventConverter 's convert method, EverestTranslator 's translate method as well as the overall performance of our software. The table below showing the performance testing result.

Test Description	Output	Expected	Is Passed
Test performance of EventCapturer's convert() method. Test 3 times, each time converts 1 event.	1st : 1ms 2nd: 1ms 3rd : 1ms	Less than 200ms	Pass
Test performance of EverestTranslator's translate() method. Test 3 times, each time translates 1 OpenStackEvent object.	1st : 8ms 2nd: 11ms	Less than 200ms	Pass

	3rd: 11ms		
Test overall performance of converting OpenStack's event to LogEvent object, this involve convert() and translate() method. Test 3 times, each time translates 1 event.	1st: 17ms 2nd: 14ms 3rd: 9ms	Less than 500ms	Pass
Test the whole process of our software, from capturing event to determining the final monitor's result. In this test, we tested 3 times, each time with 2000 event from OpenStack.	1st: 832123ms 2nd: 869348ms 3rd: 837238ms	Less than 300000ms	Fail

Although most of the testes are passed, our final test to check for non-functional requirement **NF-3** is failed. Therefore, our software does not satisfy requirement **NF-3.**

3.5.5. Conclusion

In conclusion, after performing testing with our software, we can confirm that the software satisfy all requirements except for requirement NF-3. The software will need enhancement in the future to improve the overall performance.

4. Discussion

4.1. Results compared with objectives

In this section, we compare the results obtained in this project with the objectives set in Chapter 1- Introduction and Objectives.

Objective 1: To deploy and investigate how OpenStack works and how it could be integrated with EVEREST

Result: Achieved

In this project we have successfully examined various aspect of OpenStack. The overall architecture of OpenStack has been discussed in details and various components of OpenStack such as Nova, Cinder, Identity, etc. has been discovered and examined carefully about their responsibility as well as how they interact with each other. We also managed to deploy OpenStack successfully using Single Node Deployment model and can perform various operation on the cloud. Furthermore, 2 methods for integrate OpenStack with EVEREST is also investigated. The former is to integrate with OpenStack by capturing event from notification bus

and the latter is to integrate with OpenStack by querying for information using OpenStack API. Those 2 methods has been researched carefully and used in our integration design.

Objective 2: To produce a design of the chosen integration of OpenStack with EVEREST.

Result: Achieved

After examining 2 methods of integration, we have decided to use both of them for integrate OpenStack with EVEREST. A set of design class and sequence diagram to illustrate how we implement the system has been produced and explained carefully. The design include 2 point of integrations, once is the EventCapturer that capture event from notification bus and other is the PollingEngine that execute various PollingTask to query for OpenStack's information using OpenStack API.

Objective 3: To implement and demonstrate the chosen design using any programming language of choice.

Result: Achieved

In this project, we have use Java as our programming language to implement our integration design model. Based on the design class and sequence diagram produced, we have successfully implemented the software follow exactly our design model. The demonstration of the software has been shown in chapter 4 section 4.4.1 Implementation Result.

Objective 4: To perform testing on the integrated system to ensure the quality of the produced software.

Result: Partially Achieved

The software has been tested thoroughly using unit testing, integration testing and system testing as well as performance testing to check if it can meet the requirement. The result is passed for most of the test except for performance testing. However, since our objective is mainly about implement the chosen design and not specify any requirement for software performance, we can confirm that objective 4 has been partially met.

4.2. Results compared with theoretical and applied work

Although there are various tools and monitoring framework that are available for use with OpenStack, our software is the first monitor tool that uses the strength of Event Calculus in its monitor engine. By employing EVEREST as its core monitor framework, this software can receive rule written in Event Calculus based language and use the reasoning engine provided by EVEREST to check for monitor rule violation.

Since EVEREST has been used in various projects such as SERENITY, SLA@SOI and CUMULUS, we can be assured about the dependability of our monitoring framework. Developers who are familiar with EVERETS and Event Calculus can easily use this software to provide monitor service to their OpenStack cloud.

Furthermore, cloud user and cloud administrators can use our software to monitor their Service Level Agreement, which is a very important aspect of cloud based business. Users can write monitor rule to specify what service agreement they want to monitor such as user quota limit and

have our software check for violation. Cloud administrator can identify any problem happen in their cloud infrastructure by having their resources monitored.

The two point of integration design enables developers to easily extend this software. If there are any monitor properties that is not available through OpenStack's event, developers still can implement a new PollingTask class to query for those properties from OpenStack using the RestAPI.

However, this software's performance is not really a good choice for processing a large cloud infrastructure. The performance test above has shown that with a large number of events, the software failed to process in an acceptable response time.

In addition to that, the design of Poller module is really bad. It is not extensible and not convenient. Whenever, users want to monitor a new rule using Poller module, they have to create a new PollingTask to implement the task logic such as what API to call, how to convert the obtained information to LogEvent. Thus make the design hard to scale and repetitive.

4.3. Known Limitations

Since this is a first version of the software, there are still a number of limitations that need to be addressed in the future:

- The bad design of the Poller module. Therefore, it is not recommend using this module for rule integration, it is better to use EventCapturer module. Only if there is no event available for monitor properties, that we rely on the Poller module to provide event information.
- Currently, our software can only listen to 1 message queue although the design supports multiple message queues. This is a result of a bad decision making when implementing the design. Further work need to be done to correct this mistake.
- The UnitTest for the software is not adequate enough; further test cases need to be added to cover all the functionality of the software.
- The performance issues of our software. This is mostly because of the design of EVEREST. Because the class that implement the event processing logic is not thread safe, we have no other way but to synchronize the method that submit events to EVEREST. Although this solution help us to avoid concurrent problem, it has created a bottle neck in the synchronize method and we cannot use the strength of parallel processing to speed up the performance.

These known limitation will be improved by the future version of this software.

4.4. What I would change if repeat the project

If I were to repeat the project, the following list is what I want to change:

- Instead of using Single Node deployment model, we will use Multi Node model. The Multi Node model is more practical and is of more value when checking the output produced by the software. In addition to this, with Multi Node model, the software may

have encounter problem that is not happen in single machine environment, and we can discover those problems and correct them.

- Using Magic Draw to draw the UML model instead of Microsoft Visio 2013. Because my Magic Draw license has expired, I have to use Microsoft Visio 2013 to design my diagrams. Microsoft Visio 2013 tool come with limited support for UML diagram design and the shape layouts, colours don't have professional look.

4.5. Summary

In summary, all the objectives specified at the beginning of the project have been successful achieved except for objective 4 which is partially achieved. We have managed to come up with a design to integrate OpenStack with EVEREST and implement it. The result software work as expected and satisfy almost all the functional and non-functional requirement. However, in term of performance, further improvement will need to be done for the software to be used in real OpenStack's environment with hundreds of events per second. Furthermore, known limitation of the software also need to be taken care and changed in future release.

5. Evaluation, Reflections, and Conclusions

5.1. Project Evaluation

This section evaluates the works achieved during the project. The original choice of objectives, the methods used and project planning will be examined.

5.1.1. Things that went right

- We were able to set up OpenStack developing environment. This environment enables us to perform researching, developing and testing with OpenStack.
- OpenStack supports notification message, if there was no supporting for catching event from OpenStack, the work would be harder and integration choice would also be limited.
- There are many libraries available for working with OpenStack API. This makes us easier to implement the Poller module.

5.1.2. Things that went wrong

- EC-Assertions language is difficult to learn and master, it takes me a lot of time to be familiar with writing rule using EC-Assertion language.
- There are limited documents about EVEREST and how to use it.
- There are also limited papers and document about OpenStack, most of the information are available from Internet blogs and OpenStack's wikis.
- Since I'm new to cloud computing, it took me lot of time to learn and understand OpenStack's infrastructure

5.1.3. Objectives

The original objective of this project as in the research proposal is to integrate OpenStack with EVEREST and EVEREST will monitor all the service deployed on OpenStack include IaaS, PaaS, SaaS layer. However, since the time available for the project is limited, the objective has been scoped down to only provide monitor service at IaaS layer. The modified objectives have been specified in the beginning of this report at chapter 1 Introduction and Objectives.

In chapter 5, section 5.1, we have compared the results with the identified objectives. Based on the comparisons, almost all objectives have been concluded as Achieved except for objective 4, the result of this project has met all the objectives specified in the beginning. By integrating EVEREST with OpenStack, we have successfully provided monitor service to the Infrastructure layer of OpenStack.

5.1.4. Method and Planning

Different methods used in the project have been introduced in chapter 3 Methods. Following each stage of the project life cycle, different methods have been applied:

- **Environment Setup Phase:** Devstack has been used as a tool for setting up OpenStack's environment. Single Node deployment model is applied as model for our environment setup. Using these methods, we have managed to deploy a usable OpenStack's development environment for working and testing.

- **Requirement & Analysis Phase:** by using UML modelling language and analysis class and sequence diagram, we have developed an initial view of our software as well as identified main components and their relationship as well as functionality. The diagrams produced at this stage gave us a skeleton and ideas for developing details design model of our system.

- **Design Phase:** in this phase, we have used design class and sequence model to illustrate the details structure of the system as well as how each classes communicates and send messages to each other. This design class and sequence diagram produced in this stage is a blueprint for us to start implementing our software.

- **Implementing Phase:** various tools and framework has been used as this stage. We used Spring framework's container to promote Inversion Of Control design pattern in our system. Log4J is used as a logging framework, RabbitMQ Client is a AMQP client framework which is used for communicate with AMQP broker in our software. Maven is used as building and packaging tool in our project and OpenStack4J is our Java wrapper of OpenStack API. The tools and framework used in this stage help development easier and straightforward.

- **Testing Phase:** by using unit testing, integration testing, system testing and performance testing we have managed to inspect many aspect of our software as well as ensure the software's quality. Various bugs have been found in unit testing and integration testing stages.

- **Maintenance Phase:** we have refactored some class with bad code design as well as fixing some bugs found in testing.

The methods introduced in chapter 3 have been applied successfully in our project. The results shown chapter 4 has provided evidence about the outcome of our software.

The literature examined in chapter 2 gave us a lot of knowledge about OpenStack as well as provided hints and ideas about integration methods to be used.

The planning was made at the beginning of this project by allocating time to each activity in each stage in Modified Waterfall Process model. However, the time assigned for literature review and environment setup was underestimated. Because of the lack of knowledge in cloud computing, it took me a lot of time to understand OpenStack architecture and how it works. Also environment setup also took me a lot of time. At first, without using Devstack, I tried to manually set up OpenStack cloud service by service and still end up with a fail version of OpenStack. However, after using Devstack, the environment setup job is straight forward and easier. Compare to Environment Setup and Requirement& Analysis stage, the time taken for the rest of the stages is as expected. However due to time taken in set up and research OpenStack, we didn't have much time to set up enough test cases and develop enough unit tests for our software. Therefore, only important components are tested and verified using JUnit test classes.

5.2. Key Achievement

5.2.1. Project Contribution

The project achieved provides software that can be used for monitoring OpenStack's Infrastructure layer. It is also the first software that uses Event Calculus to provide monitoring capabilities. So if there are any parties that interested in using Event Calculus to monitor OpenStack cloud, this software is a choice for them.

CUMULUS is an EU project whose objective is to certify security properties of cloud services based on monitoring and testing evidence dynamically acquired during the operation of cloud services. And EVEREST is used in CUMULUS for monitoring. So this project can used our software framework and code to give it access to OpenStack cloud.

Another contribution is the integration design model. Based on our integration model, other developers can also develop their own monitoring system using the same integration mechanism that we have discovered.

5.2.2. Learning Outcomes

After working on this project for 3 months, my knowledge about cloud computing has improved significantly especially about OpenStack cloud computing platform. I also have learnt various deployment models that can be used to deploy our own cloud as well as how to develop a cloud aware application.

Event Calculus language is also one of the key achievement that I have learned with the knowledge of Event Calculus, I can used it on any future projects.

This project also help me to enhance my analysis and design skill significantly, I have gained more experience in how to design different components that follow best design practises as well as the ability to self-criticism on my own designs. Furthermore, my writing skill is also improved a lot from writing project report.

Aside from key achievement that I have gained from the project, a few lessons have been learned from the project management. Environment set up and literature review should never be

underestimated, more time should be invested in the initial planning. And extra time should be allocated to prepare for any unexpected problem that can occur.

5.3. Further work

Based on this project, a lot of idea for future project has been discovered. However before jumping to the idea of new project, some modification in the design of this software should be considered.

- The Poller component should be re-designed to avoid manually creating new task and register with the Poller. One idea is to use XML to describe information about the task, and code will be generated for that task based on the XML's information.
- Fix an implementation fault in our code that prevents our software to register for new message queue.
- Provided adequate unit test for all the components of our software.
- The performance of this software need to be improved, changed may be made to EVEREST to enable concurrent processing of events.

These works above is what we should do in the new version of this software. The list below is proposals for future projects:

- Support for other AMQP broker such as ZeroMQ. Currently our software only have support for RabbitMQ, additional support for other message broker should be implemented in the future projects.
- Provide monitoring capacities for other layer of OpenStack such as PaaS and SaaS. This way we can integrate EVEREST completely with OpenStack cloud.
- Integrate EVEREST with other cloud technology like Microsoft's Azure, Amazon's EC2.

5.4. General Conclusion

The main objectives of this 3 month project which are to investigate how to integrate OpenStack with EVEREST, to produce and implement a chosen integration model that can work with OpenStack. The objectives have been successfully accomplished with the outcome is a software that use EVEREST as its core monitoring engine. This software is the result of efforts putting in this project and is the demonstration of our integration mechanism. With a set of test cases that has been used to verify our software, we can confirm that this software work as expected and can be used as a tool to monitor OpenStack or a model for integration implementation. However, some works still need to done in future version of this software to provide users with a reliable and easy to use monitoring tools.

References

Openstack.org, (n.d). Home » OpenStack Open Source Cloud Computing Software. [online] Available at: http://www.openstack.org/ [Accessed 09 Sep. 2014].

Azure.microsoft.com, (2014). Azure: Microsoft's Cloud Platform. [online] Available at: https://azure.microsoft.com/en-us/ [Accessed 09 Sep. 2014].

Cloudfoundry.org, (2014). Welcome to the Cloud Foundry Community | Cloud Foundry Community. [online] Available at: http://cloudfoundry.org/index.html [Accessed 08 Sep. 2014].

Spanoudakis G, Kloukinas C. Mahbub K.: The SERENITY Runtime Monitoring Framework, In Security and Dependability for Ambient Intelligence, In Security and Dependability for Ambient Intelligence, Information Security Series, Springer, pp. 213-238.

Munassar, Nabil Mohammed Ali; Govardhan, A (2010), A Comparison Between Five Models Of Software Engineering. In International Journal of Computer Science Issues (IJCSI), Vol. 7 Issue 5, p94-101, Sep2010.

Wikipedia, (2014). OpenStack. [online] Available at: http://en.wikipedia.org/wiki/OpenStack [Accessed 09 Sep. 2014].

OpenStack Training Guide, 2014.

Wiki.openstack.org, (2014). Zaqar - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/Zaqar [Accessed 10 Sep. 2014].

Wiki.openstack.org, (2014). Sahara - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/Sahara [Accessed 10 Sep. 2014].

Davide Lorenzoli, George Spanoudakis (2009), Detection of Security and Dependability Threats: A Belief Based Reasoning Approach. In Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09, Third Conference, p.312-320, 18-23 June 2009.

Wikipedia, (2014). Cloud computing. [online] Available at: http://en.wikipedia.org/wiki/Cloud_computing [Accessed 11 Sep. 2014].

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy KonWinSki,Gunho Lee, David Patterson, Ariel Rabkin, ion Stoica, and Matei Zaharia (2010), A View of Cloud Computing.In Magazine Communication of the ACM, Volume 53 Issue 4, page 50, April 2010.

Developer.openstack.org, (2014). OpenStack API Documentation. [online] Available at: http://developer.openstack.org/api-ref-compute-v2.html [Accessed 12 Sep. 2014].

Docs.openstack.org, (2014). System Architecture — Ceilometer 2014.2.dev48.g85b859f documentation. [online] Available at: http://docs.openstack.org/developer/ceilometer/architecture.html [Accessed 12 Sep. 2014].

Sandywalsh.com, (2013). SandyWalsh.com: Notification Usage in OpenStack – Report Card. [online] Available at: http://www.sandywalsh.com/2013/09/notification-usage-in-openstack-report.html [Accessed 14 Sep. 2014].

Shanahan, M. (1999). The event calculus explained. Springer, pp.409--430.

Wiki.openstack.org, (2014). Ceilometer - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/Ceilometer [Accessed 15 Sep. 2014].

George Spanoudakis, The Language for Specifying Monitoring Rules in Serenity Pattern

Nagios.org, (2014). Nagios - Nagios Overview. [online] Available at: http://www.nagios.org/about/overview [Accessed 16 Sep. 2014].

Munassar, Nabil Mohammed Ali; Govardhan, A (2010), A Comparison Between Five Models Of Software Engineering. In International Journal of Computer Science Issues (IJCSI), Vol. 7 Issue 5, p94-101, Sep2010.

Devstack.org, (2014). Single Machine Guide - DevStack. [online] Available at: http://devstack.org/guides/single-machine.html [Accessed 17 Sep. 2014].

Wiki.openstack.org, (2014). DevStack - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/DevStack [Accessed 16 Sep. 2014].

Oracle.com, (2014). Oracle Technology Network for Java Developers | Oracle Technology Network | Oracle. [online] Available at: http://www.oracle.com/technetwork/java/index.html [Accessed 17 Sep. 2014].

Projects.spring.io, (2014). Spring Framework. [online] Available at: http://projects.spring.io/spring-framework/ [Accessed 17 Sep. 2014].

Logging.apache.org, (2014). Apache log4j 1.2 -. [online] Available at: http://logging.apache.org/log4j/1.2/ [Accessed 17 Sep. 2014].

Unruh, J. (2014). OpenStack4j - Fluent OpenStack client for Java. [online] Openstack4j.com. Available at: http://www.openstack4j.com/ [Accessed 18 Sep. 2014].

Jclouds.apache.org, (2014). Apache jclouds® :: Home. [online] Available at: https://jclouds.apache.org/ [Accessed 17 Sep. 2014].

Porter, B. and Zyl, J. (2014). Maven – Welcome to Apache Maven. [online] Maven.apache.org. Available at: http://maven.apache.org/ [Accessed 16 Sep. 2014].

Junit.org, (2014). JUnit - About. [online] Available at: http://junit.org/ [Accessed 19 Sep. 2014].

Appendix A: Project Proposal for MSc in Software Engineering

Name: Duy Nguyen.

Email Address: Duy.Nguyen.6@city.ac.uk

Contact Number: 07749 152160.

Project Title: Integrate OpenStack with EVEREST monitoring framework.

Supervisor: Prof. George Spanoudakis.

I. Introduction:

1. Background Information:

Security and dependability is one of the key properties of a system. Any violation to these properties can lead to a drop in availability, reliability, safety of a system or even total failure of the whole system. Therefore, these key properties is very important to every system especially those that involve critical activities such air traffic system, nuclear reactor controller. However, ensuring the security and dependability of complex system which operate in a distributed environment with highly changing context can prove to be very difficult (Geogre,Christos and Khaled,2009,p.214). The reason for this is because as operational conditions change, the security and dependability mechanism of a system may become ineffective and needed to be replaced or adapted to ensure the security and dependability properties. Therefore, in order to enable the ability to react to dynamically changes, the monitoring of the operation of the security and dependability mechanism is needed.

Runtime Monitoring is a computer system analysis and execution approach that obtains the information from the running system and uses this information to perform analysis in order to detect behaviours that satisfy or violate a defined set of properties. Some of the properties can be deadlock freedom, order of behaviour or even performance requirement. Upon detecting any violation behaviour, the system will decide how to react to certain violation such as roll back execution units or report the logging of these problems to the client that require monitored result. The ability to detect violation and react to such an event make runtime monitoring very important since it enable the system to preserve its safety and dependability.

In 2009, Geogre, Chritos and Khaled of City University have developed a Runtime Monitoring Framework called EVEREST (EVEnt Rea-Soning Toolkit) which is deployed as a service to SERENITY Runtime Framework. EVEREST acts as a monitor that undertakes responsibility for checking conditions regarding the runtime operation of the component that realize the safety and dependability mechanism of a system. This Runtime Monitoring Framework can detect any violation of the monitoring rules and provides other capability such as deduce information about the state of the systems, detect potential violations and perform diagnostic analysis (Geogre,Christos and Khaled,2009,p.215).

2. Problem Statement:

EVEREST has been used as core monitoring service for a number of systems. One of such system is SERENITY Runtime Framework where EVEREST main responsibility is monitor components that implements S&D Pattern. Another large system that uses EVEREST as its low level monitoring engine is SLA@SOI framework. SLA@SOI framework provides libraries and tools, utilities to implement Service Level Agreement (SLA) aware service. EVEREST is deployed as a general-purpose engine that monitors the behaviours and SLA term and agreement of distributed systems based on events captured from them during the system runtime operation (Khaled, Geogre and Theocharis, 2011, p.80).

After discussion with my supervisor Geogre Spanoudakis, we both think that EVEREST is a useful tool and could be exposed for used by other services that require their security and dependability mechanism to be monitored. As a result, the aim of this dissertation project is to integrate EVEREST into the cloud and provide Monitoring as a Service (MaaS) option to other services which deployed on the same cloud. OpenStack is used as our cloud computing platform in this dissertation project.

3. Aim & Objectives:

The aim of this project is to integrate Event Rea-Soning Toolkit (EVEREST) as a monitoring service on public and private clouds built by OpenStack operating system. This monitoring service will enable other cloud services (SaaS) deployed on the same cloud to subscribe itself to EVEREST and have it security and dependability mechanism monitored and reported by the monitoring service.

The following points summarize the objective of this project:

- To perform researching and investigating on cloud technologies and components provided by OpenStack.
- To install and run our own OpenStack cloud using different tools provided such as DevStack.
- To identify and specify the requirement needs to achieve our goals.
- To develop UML models to our integrated system in the future. With these UML models, we can gain a better understanding of our future system and have an overall look of our design decision.
- To implement our system based on the UML models defined during design phase.
- To conduct testing to verify if our system works as expected. In order to perform testing, we need to develop a test service to use our monitoring service.

4. Project Scope:

The scope of this project is limited to integrate EVEREST into OpenStack cloud only. The project only focuses on how to deploy EVEREST into OpenStack cloud and provide it as a Monitoring Service to other services on the cloud. Additional components will be implemented

in order to act as an intermediate gateway between EVEREST and external services. Apart from that, there will be no additional capabilities or feature implemented.

5. Beneficiaries:

The immediate beneficiaries of this project would be a direct contribution to the OpenStack community now and other cloud community in the future. As new monitor service provided, open source community would be benefit from a ready to use software which has been built to use cutting edge technology and research. Cloud developers will have more choice when finding monitoring solution to their services.

As a researcher, this project provides opportunity for me to learn about cloud computing technologies and enhance my designing and programming skill.

II. Critical Context:

This section aim to outline and provide in depth review of the literature consulted throughout the proposal. Literature analysis done in several sections.First is researching of Cloud Computing and its implementation OpenStack which give us a better understanding of the environment that we will deployed our monitoring system on. Second is researching on the EVEREST itself and diving deep into its coding implementation, by examining the research paper and code, we will gain in depth understanding about the tool that will be used. After that, we will take a look at some existing project that use EVEREST as its monitoring engine. Last but not least, we will investigate some software development methodologies that can be used as our development process.

1. Cloud Computing:

In computer networking, cloud computing refer to a communication network that involves a large number of computers that connected together. In science, cloud computing refer distributed computing where a program or application run on many connected computers at the same time (Wikipedia, Cloud Computing).

With cloud computing, developers with innovative ideas no longer need to concern about the underlying hardware infrastructure. Everything from software, platforms and infrastructure are sold as a service remotely through internet. Cloud computing providers provide their services according to three basic fundamental models:

- Infrastructure as a Service (IaaS) : computers or virtual machines and other resources such as file-based storage, firewalls, load balancers, etc. are provided (Wikipedia, Cloud Computing).
- Platform as a Service (PaaS): computing platform such as operating system, programming language execution environment and database, etc. are provided.
- Software as a Service (SaaS): applications are delivered as a services over the internet while the hardware and system software located in data centers (Michael, Armando, Rean, Randy, Andy, Gunho, David, Ariel, Ion and Matei,2010, p.50).

In our projects, EVEREST will be deployed as Monitoring as a Service (MaaS) to provide monitor service to other services deployed on the cloud.

2. OpenStack:

OpenStack is an open source project with the goal to provide an open source cloud computing platform for public and private cloud. Initially focusing on Infastructure as a Service (IaaS) , the project currently includes three main component (Ken, 2011,p.1):

- OpenStack Compute: this component provision and manage large networks of virtual machines and give the provider with the ability to offer on-demand computing resources. This component is called Nova.
- OpenStack Storage: consist of Object Storage (Swift) and Block Storage (Cinder). This component provides an API-accessible storage platform for static data such as email, photo storage or persistent block level storage devices for use with Compute instances (OpenStack Webpages).
- OpenStack Networking: this component is a system for managing networks and IP addresses. It is called Neutron.

3. EVEREST:

EVEREST is a monitoring framework which offering interfaces for submitting rules to it for checking, receiving events from the monitored applications and querying monitoring result. It consist of three main components: a monitor manager, a monitor and an event collector (Geogre,Christos and Khaled,2009,p.217).

The monitor manager main responsibilities are receiving monitoring rules and providing API for obtaining result. The event collector takes charge of receiving events from the monitored application and passing them to the monitor manager. Lastly, the monitor will check if the received events violate any of the rules given to it (Geogre,Christos and Khaled,2009,p.217).

After taking a look at the source code, the main flow of EVEREST is as follows:

- First, the monitored services need to submit a set of rules which are specified by using XML language, called EC-Assertion to the monitor manager. The rules need to be converted to uk.ac.city.soi.everest. Formula object before feeding to the monitor.
- Upon receiving the rules as Formula objects, the monitor component will translate these objects into uk.ac.city.soi.everest.monitor.Template objects using Template Maker and persist them into the database.
- During day-to-day operations, the monitored services will sent its activities as an event messages which are also specified by using XML language to the event collector. The events need to be converted to uk.ac.city.soi.everest.SRNTEvent TypeOfEvent objects before feeding to the monitor.
- When receiving the events, the monitor will translate these TypeOfEvent objects into uk.ac.city.soi.everest.er.LogEvent objects and add them to the EventQueue. The event will then be processed by the main loop running in the analyser. The

analyser will check the events against corresponding templates and update the violated templates in the database.

- There is getMonitoringResult() method return a collection of violated templates to the caller to find out the violated rules and their specific informations.

By understanding the implementation detail of EVEREST, we will know to integrate it to OpenStack properly.

4. Existing Project:

In order to understand how EVEREST is used as monitoring services, a research about existing project that used EVEREST is need to be conducted.

i. SLA@SOI:

SLA@SOI framework provides libraries and tools, utilities to implement Service Level Agreement (SLA) aware service. This framework needs to have its SLA guarantee terms monitored and reported to the SLA@SOI framework.

This framework use EVEREST as one of its low level monitoring engine. However, since the SLA guarantee terms are specifies in a language that is different from the one that the engines use to express operational monitoring specification. A wrapper for the monitoring engine needs to be implemented. This wrapper will translate the SLA expressed language into the language that is supported (EC-Assertion) by the monitor. The implementation of the EVEREST wrapper in SLA@SOI framework is EVEREST RCG (Khaled, Geogre and Theocharis, 2011, p.80)

ii. SERENITY:

SERENITY framework enabling the abilities to configure, deploy and select dynamically the components that realise the safety and dependability mechanism. EVEREST is used as a monitoring engine for monitor the runtime operation of those components.

In this framework, the SERENITY Runtime Framework (SRF) will submit monitoring rules to the EVEREST by interface exposed by the monitor manager. An Event Capturer outside of EVEREST will intercept events during the operation of monitored application and send them to the SRF, which later forward them to EVEREST's event receiver. The Event Capturer is part of the monitored application.

After examining these two above projects, we realize that a component needs to be implemented to act as a gateway to the EVEREST, just like EVEREST RCG and SRF. This component will act as a communication gateway which receive event and rule from monitored service and translate them to a format that EVEREST can understand as well as handle different communication channel from different monitored services.

III. Approaches:

This section discuss in detail the method used to achieve the desired result for the project. It describes the methods used for analysis, design, and implementation and testing.

1. Software Process Model:

After carefully investigate different type of software process model, I have decided to use **Modified Waterfall Process** model as the process model for the project.

The Waterfall model is goods for products will clearly understood requirement and work well with well understood technical tools, architecture and infrastructure which are true in our case (Munassar, Nabil and Govardhan, 2010, p.94-101) . However, it is really hard in practice because it is nearly impossible for any big project to finish a phase of software products development perfectly before moving to next phases. Therefore Modified Waterfall Process has been introduced to fix these problems. It allow designer and developer to bounce back to previous phase to make any necessary modification if needed.

2. Environment Setup:

Since we have to deploy EVEREST on to OpenStack cloud, we have to know how to install and configure OpenStack cloud in order to prepare a testing environment for the project.

According to the installation guide on OpenStack webpage, we need to use a Linux Distribution, in our case we will use **Ubuntu 12.04** as our Linux Operating System. The reason for this choice is simply because I have experience working with Ubuntu before. After that we need to download **DevStack** tool to perform installation and configuration.

3. Requirement Gathering:

In the Requirement Gathering phase, we will use **Usecases and Usecases Specification**. As I have researched, "Use cases are the best choice for requirements capture when the system is dominated by functional requirement" (Arlow and Neustadt, 2007, p.91), it is clearly that Usecases and Usecases Specification is suitable for our project since most of the project's requirement is functional requirement. The Usecases will be drawn using Magic Draw which I have used during Software System Design module.

4. System Design:

During system design phase, **Unified Modeling Language (UML)** will be used as our modelling language. UML is a very powerful modelling language which provides system designer with a way to visualize a system's architecture in a diagram. We can use UML to model the whole system without concern about the implemented language. In order to, model the structure and behaviour of the system, we will use 2 types of diagram:

- ***Class Diagram:*** a class diagram in UML is a type of structure diagram which can be used to describe the structure of a system. The diagram contains system's classes, their attributes and operations as well as relationship between objects. This is very useful if we want to model our system.

- ***Sequence Diagram:*** it is an interaction diagram which can be used to show how objects interact with each other arranged in time sequence. It contains the objects and classes that involves in the scenario and how messages are exchanged between them. This is very useful if we want to model how our system's components interact with each other.

5. Implementation:

Java will be used as our implementation language for the project. One of the reason for this decision is because EVEREST is written in Java, therefore in order to interact effectively with it, the new component will be written in Java. The second reason is because Java is backed by a large choice of open source library such as Hibernate for database access, or Spring MVC for developing web application. The last reason is simply because of my proficiency with Java.

Spring Framework's Spring Core and Spring Bean library will be used by our project, mainly because it's powerful dependency injection capabilities. Thus will improve the loosely couple nature between components which make it easier for testing.

As for database access layer, we can use **Hibernate** or take advantage of **Spring JPA** library to implement this layer. Other alternative is using OpenStack's Database Service **Trove-API** as a way to store data in a relational database.

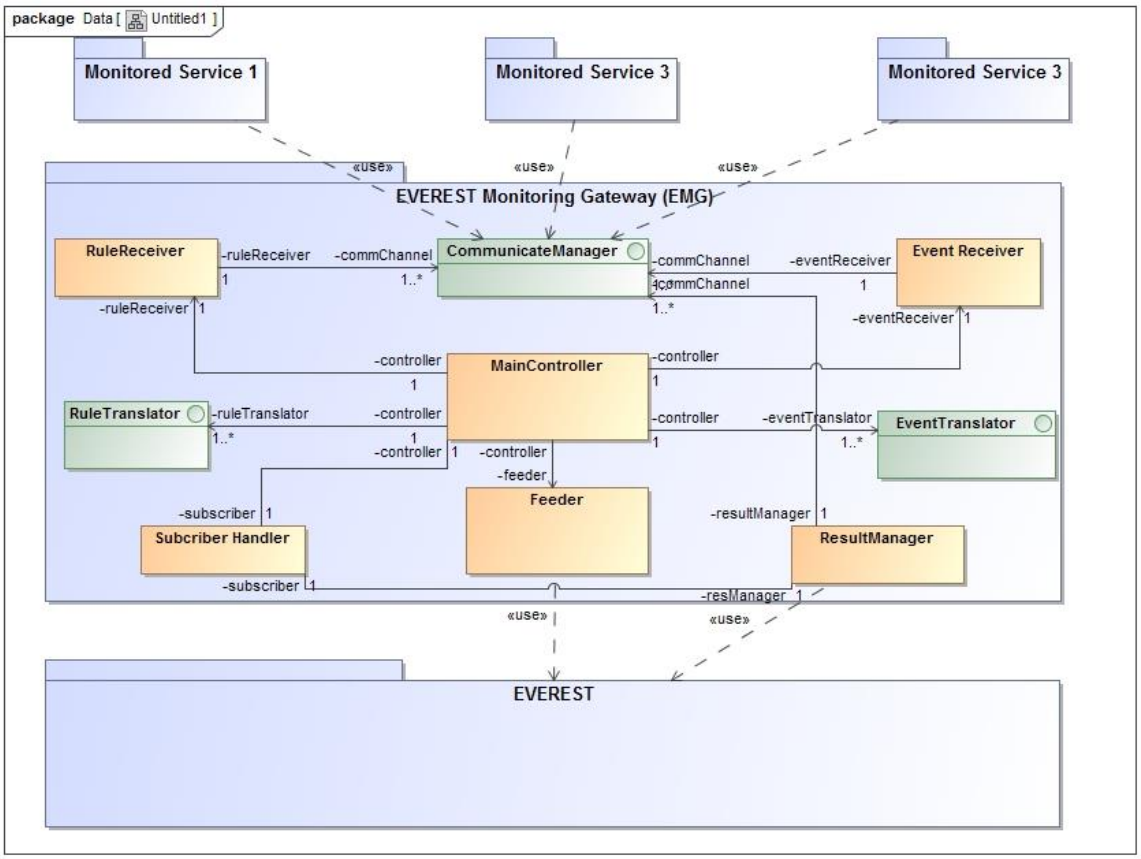
For the communication between the monitored service and EVEREST's gateway component, I will use **JMS** or **XMPP** to establish communication channel between the gateway and monitored services. Other alternative solutions is making use of **RMI**, the external services will obtain the remote reference to the gateway through JNDI. This approach can be implemented using RMI or through an Application Server which can provide **EJB** remote interface for the gateway. However, this approach will restrict the service to be implemented in Java, and the use of an Application Server is too heavy weight and unnecessary.

6. Testing:

JUnit will be used for testing the behaviour of the implemented component. However, in order to actually test the monitoring capabilities of EVEREST, we need to implement a test service. The test service will also be implemented in Java. In order to observe the result of the monitoring, a GUI will be developed to display the monitor's result. The GUI will be developed as a Webpage using **Spring MVC** framework and **JSP/Servlet API**.

7. An initial design of the future system:

In order to integrate EVEREST into the cloud, we need to develop a component which will act as a gateway to the EVEREST engine; we call this component EVEREST Monitor Gateway (EMG). This gateway component will help to decouple EVEREST from receiving event, rules and sending result responsibilities. Thus improving EVEREST cohesion and following Single-Responsibilities design principles, the EVEREST only need to concentrate on analysing and detecting violation. The EMG will take charge in communicating with monitored service, converting message to formula and event, and delivering monitored result to different subscriber. The model below illustrates EMG and its sub components:



The EMG is separated into 9 sub component:

- **CommunicateManager**: this is the interface that will be used to communicate with external services for receiving event, receiving Rule and sending result. There can be different implementations of this component based on different communication protocol. (This component is based on PubSubManager idea in the EVEREST-RCG).
- **RuleReceiver**: this component will receive rule messages generated by external services and pass the rules to EVEREST through the MainController.
- **EventReceiver**: this component will receive event messages generated by external services and pass the events to EVEREST through the MainController.
- **RuleTranslator**: this component will take charge in translate the rule messages to Formula object s which is accepted by the EVEREST monitor engine. There can be different implementations of RuleTranslator based on format of Rule messages.
- **EventTranslator**: this component will take care the translation of event messages to TypeOfEvent objects which is accepted by the EVEREST monitor engine. There can be different implementations of EventTranslator based on format of Event messages.
- **MainController**: this component act as a coordinator for EMG. It will receive the event/rule messages from Event/RuleReceiver and translate the event/rule

messages to TypeOfEvent/Formula objects using Rule/EventTranslator and feed the event/rule to EVEREST using Feeder component.

- **Feeder**: this component will communicate with EVEREST and feed the engine with Formula and TypeOfEvent: object for violation detection.
- **ResultManager**: this component will query the monitor result from EVEREST and send it to corresponding services.
- **SubscriberHandler:** which will store the subscriber channel identifier and it's corresponding templateId , in order to return the monitoring result to the correct recipients.

IV. Plan Of Work:

V. Risks Analysis:

Risk	*Probability*	*Impact*	*Effect*	*Response Action*
Requirement Gathering Stage				

Requirements are ambiguous.	Medium	High	Requirements are unclear and may lead to wrong interpretation.	Discuss with supervisor about the requirement and checked thoroughly until all the parts is understood.
Requirements are incomplete.	Low	High	Can cause wrong design decision thus delay the project schedule. Redesign action will have to be taken.	Check with supervisor in order to spot any holes in the requirement. In case if this risk happened, redesign and reschedule the project.
Environment Setup				
Hardware and resources are insufficient for setting up the environment.	Low	High	Can cause delay to the schedule and affect other tasks.	Purchase additional hardware and resource or rely on external cloud provider.
System Design				
Design is infeasible.	Low	High	Cannot proceed to implementation and cause delay to the schedule.	Discuss with supervisor and review the design in order to redesign the system. Reschedule.
Poor design.	Medium	Medium	Poor design makes implementation and maintenance harder.	Ask for review from supervisor and fix poor design part.
Fail design review.	Low	Medium	Project delayed.	Redesign any parts reviewed by the supervisor and reschedule.
Implementation				
Deadline missed implementation.	Medium	Medium	Project delayed.	Reduce the scope of the project and remove some less important features. Reschedule
Chosen libraries and tools not support required features as expected.	Low	High	Project delayed.	Change to another tools or libraries. In the worst-case scenarios, I have to implement the required features.
Fail to implement components because design is not possible.	Low	High	Cause project delayed and affect subsequent tasks.	Discuss with supervisor and redesign the failed components. Reschedule.

Testing				
Error-prone component requires more testing, design and implementation work than expected.	Medium	High	Cause project delayed.	Put extra time into the schedule in order to meet the project deadline. Reschedule.
Strict requirement on the system may require more testing, design and implementation work.	Medium	High	Cause project delayed.	Lower the requirement expectation or put extra time into the schedule.
Developer				
Big learning curves for new skills.	Medium	Medium	Reduce productivity thus delay the schedule.	Use extra time not in the schedule to learn about the skills.
Sickness.	Medium	Medium	Reduce productivity and delay schedule.	Reschedule.

VI. References List:

1. Khaled Mahbub, Geogre Spanoudakis and Theocharis Tsigkritis (2011), *Translation of SLAs into Monitoring Specification*. In Service Level Agreements for Cloud Computing, page.79.
2. Geogre Spanousdakis, Christos Kloukinas, Khaled Mahbub (2009), *the SERENITY Runtime Monitoring Framework*. In Security and Dependability for Ambient Intelligence, pages 213-237.
3. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy KonWinSki,Gunho Lee, David Patterson, Ariel Rabkin, ion Stoica, and Matei Zaharia (2010), *A View of Cloud Computing*. In Magazine Communication of the ACM, Volume 53 Issue 4, page 50, April 2010.
4. Munassar, Nabil Mohammed Ali; Govardhan, A (2010), *A Comparison Between Five Models Of Software Engineering*. In International Journal of Computer Science Issues (IJCSI), Vol. 7 Issue 5, p94-101, Sep2010.

5. Jim Arlow & Ila Neustadt (2005*). UML2 and the Unified Process: Practical Object---Oriented Analysis and Design*. In USA: Addison Wesley.

6. Openstack Documentation.[Online] Available from : https://wiki.openstack.org .

7. Wikipedia. Cloud Computing. [Online] Available from: http://en.wikipedia.org/wiki/Cloud_computing .

VII. Ethical, Professional and Legal Issues :

Research Ethics Checklist

School of Informatics BSc, MSc, MA Projects

If the answer to any of the following questions (1 – 3) is NO, your project needs to be modified.		*Delete as appropriate*
1.	Does your project pose only minimal and predictable risk to you (the student)?	**Yes**
2.	Does your project pose only minimal and predictable risk to other people affected by or participating in the project?	**Yes**
3.	Is your project supervised by a member of academic staff of the School of Informatics or another individual approved by the module leaders?	**Yes**
If the answer to either of the following questions (4 – 5) is YES, you MUST apply to the University Research Ethics Committee for approval. (You should seek advice about this from your project supervisor at an early stage.)		*Delete as appropriate*
4.	Does your project involve animals?	**No**
5.	Does your project involve pregnant women or women in labour?	**No**
If the answer to the following question (6) is YES, you MUST complete the remainder of this form (7 – 19). If the answer is NO, you are finished.		*Delete as appropriate*
6.	Does your project involve human participants? For example, as interviewees, respondents to a questionnaire or participants in evaluation or testing?	**No**
If the answer to any of the following questions (7 – 13) is YES, you MUST apply to the Informatics Research Ethics Panel for approval and your application may be referred to the University Research Ethics Committee. (You should seek advice about this from your project supervisor at an early stage.)		*Delete as appropriate*
7.	Could your project uncover illegal activities?	**Yes/No**
8.	Could your project cause stress or anxiety in the participants?	**Yes/No**
9.	Will you be asking questions of a sensitive nature?	**Yes/No**
10.	Does your project rely on covert observation of the participants?	**Yes/No**
11.	Does your project involve participants who are under the age of 18?	**Yes/No**
12.	Does your project involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)?	**Yes/No**
13.	Does your project involve participants who have learning difficulties?	**Yes/No**
The following questions (14 – 16) must be answered YES, i.e. you MUST COMMIT to satisfy these conditions and have an appropriate plan to ensure they are satisfied.		*Delete as appropriate*
14.	Will you ensure that participants taking part in your project are fully informed about the purpose of the research?	**Yes/No**

15. Will you ensure that participants taking part in your project are fully informed about the procedures affecting them or affecting any information collected about them, including information about how the data will be used, to whom it will be disclosed, and how long it will be kept? **Yes/No**

16. When people agree to participate in your project, will it be made clear to them that they may withdraw (i.e. not participate) at any time without any penalty? **Yes/No**

The following questions (17 – 19) must be answered and the requested information provided.	*Delete as appropriate*

17. Will consent be obtained from the participants in your project? **Yes/No**

    Consent from participants will be necessary if you plan to gather personal, medical or other sensitive data about them. "Personal data" means data relating to an identifiable living person; e.g. data you collect using questionnaires, observations, interviews, computer logs. The person might be identifiable if you record their name, username, student id, DNA, fingerprint, etc.

    *If **YES**, provide the consent request form that you will use and indicate who will obtain the consent, how are you intending to arrange for a copy of the signed consent form for the participants, when will they receive it and how long the participants will have between receiving information about the study and giving consent, and when the filled consent request forms will be available for inspection (**NOTE**: subsequent failure to provide the filled consent request forms will automatically result in withdrawal of any earlier ethical approval of your project):*

18. Have you made arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential? **Yes/No**

    Provide details:

19. Will the research be conducted in the participant's home or other non-University location? **Yes/No**

Appendix B: Environment Setup Document

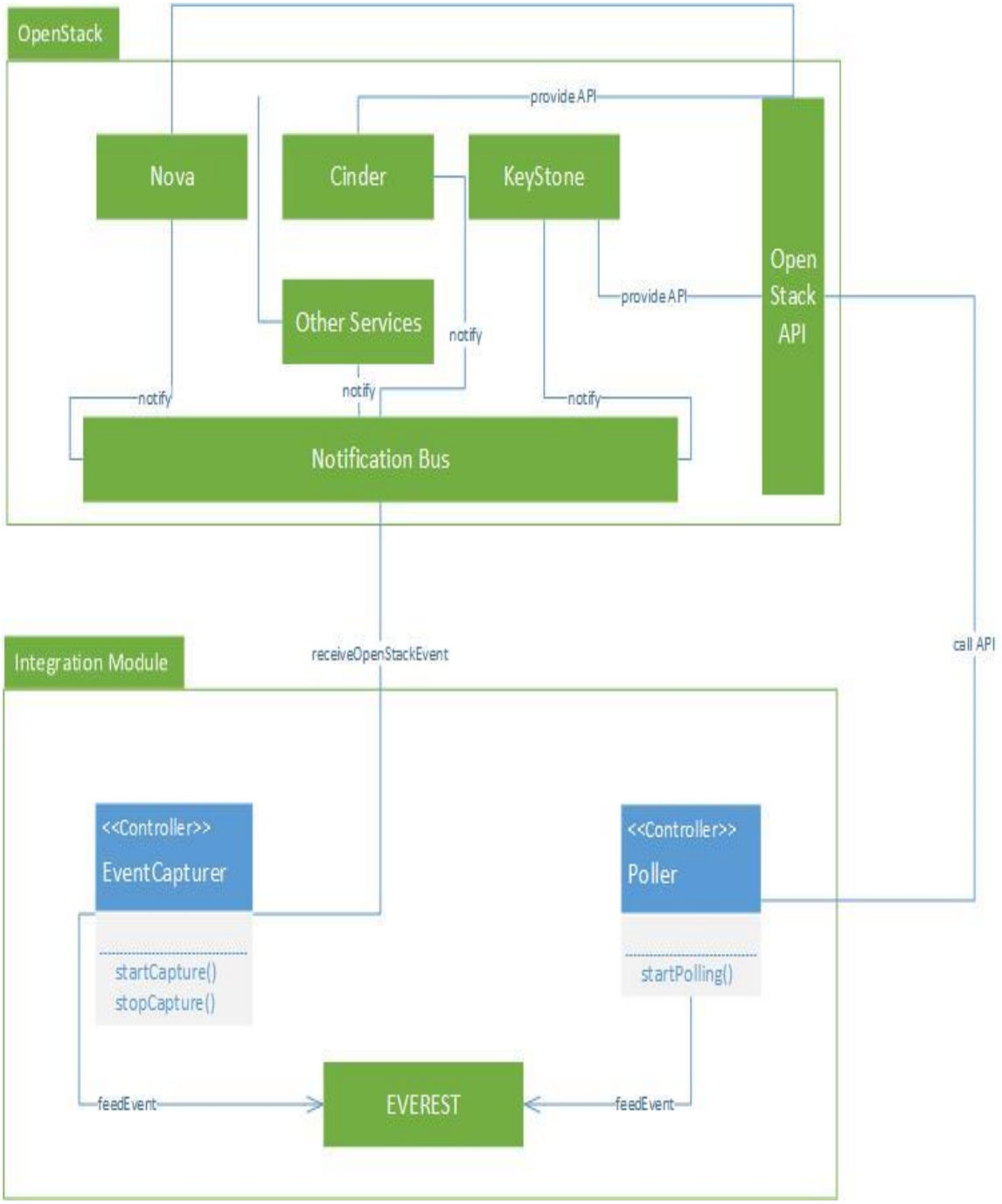
Configuration detail for setting up OpenStack using Devstack.

```
[[post-config|$NOVA_CONF]]
[DEFAULT]
use_syslog = True
notification_driver=nova.openstack.common.notifier.rpc_notifier
nova.openstack.common.notifier.rabbit_notifier
notification_topics=notifications,monitor
notify_on_state_change=vm_and_task_state
notify_on_any_change=True
instance_usage_audit=True
instance_usage_audit_period=hour

[[post-config|$KEYSTONE_CONF]]
[DEFAULT]
notification_driver=keystone.openstack.common.notifier.rpc_notifier
default_notification_level=INFO
notification_topics=notifications,monitor
control_exchange=identity
notify_on_any_change=True
instance_usage_audit=True
instance_usage_audit_period=hour

[[post-config|$GLANCE_API_CONF]]
[DEFAULT]
notification_driver=glance.openstack.common.notifier.rpc_notifier
default_notification_level=INFO
control_exchange =glance
notification_topics=notifications,monitor
notify_on_any_change=True
instance_usage_audit=True
instance_usage_audit_period=hour

[[post-config|$CINDER_CONF]]
[DEFAULT]
notification_driver=cinder.openstack.common.notifier.rpc_notifier
default_notification_level=INFO
notification topics=notifications,monitor
notify_on_any_change=True
instance_usage_audit=True
instance_usage_audit_period=hour
control_exchange=cinder

[[local|localrc]]
FLOATING_RANGE=192.168.1.224/27
FIXED_RANGE=10.11.12.0/24
FIXED_NETWORK_SIZE=256
FLAT_INTERFACE=eth0
ADMIN_PASSWORD=admin
MYSQL_PASSWORD=mysql
RABBIT_PASSWORD=rabbit
SERVICE_PASSWORD=service
LOGFILE=$DEST/logs/stack.sh.log

# Enable the ceilometer metering services
enable_service ceilometer-acompute ceilometer-acentral ceilometer-
anotification ceilometer-collector
```

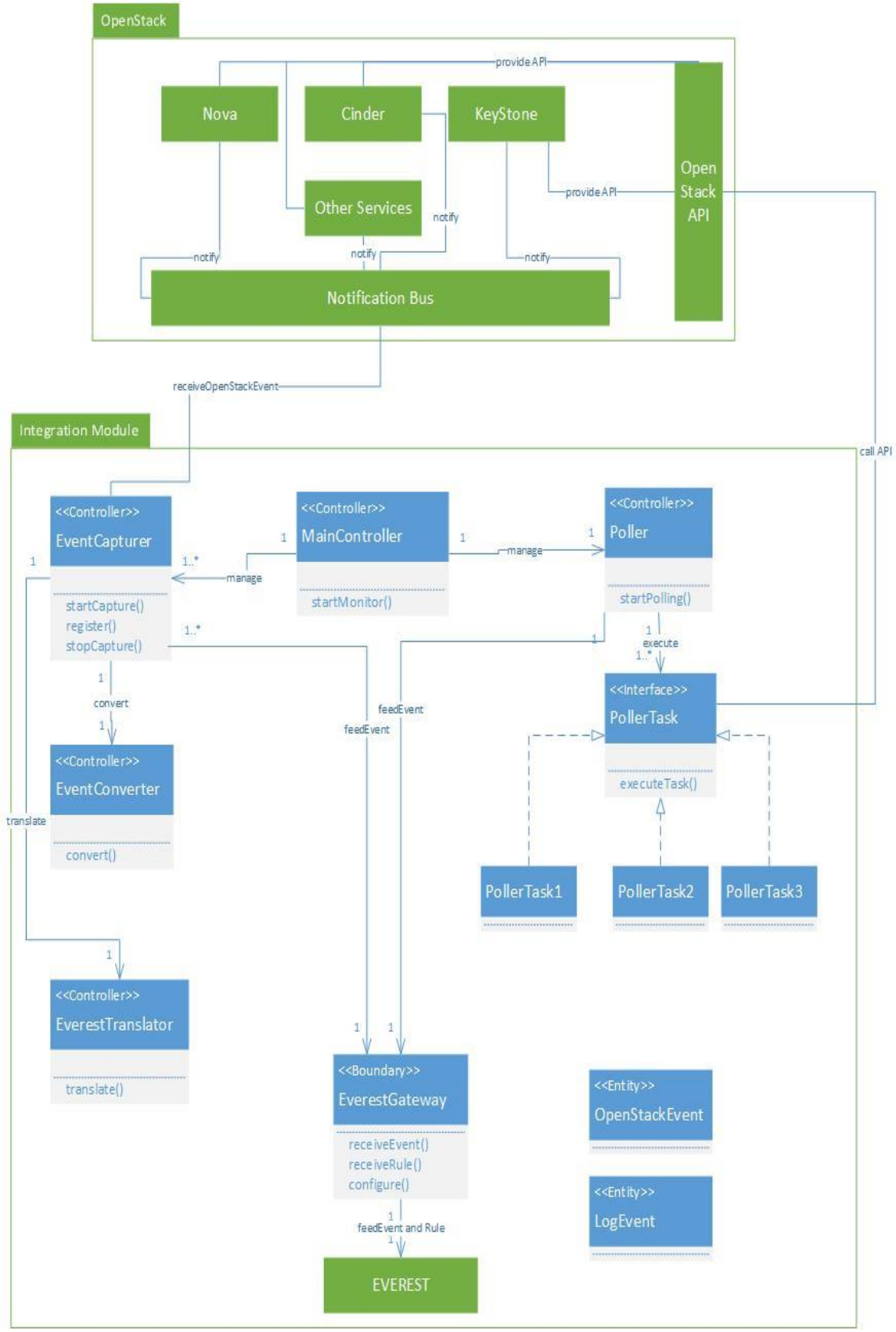
```
# Enable the ceilometer alarming services
enable_service ceilometer-alarm-evaluator,ceilometer-alarm-notifier
# Enable the ceilometer api services
enable_service ceilometer-api
```

Appendix C: Analysis Document

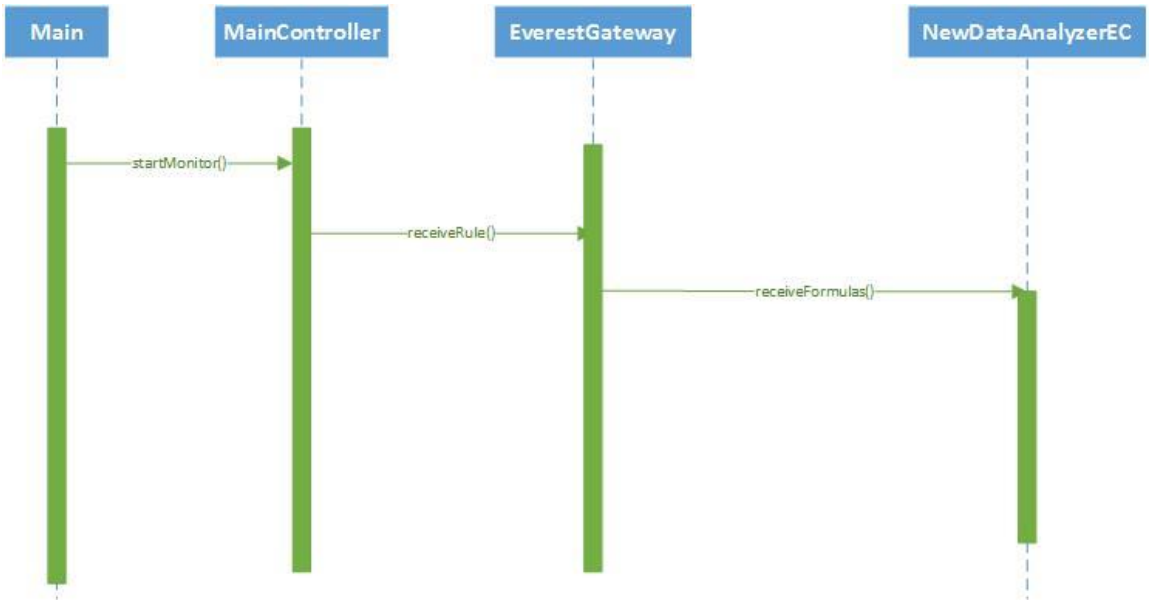
Architectural Diagram



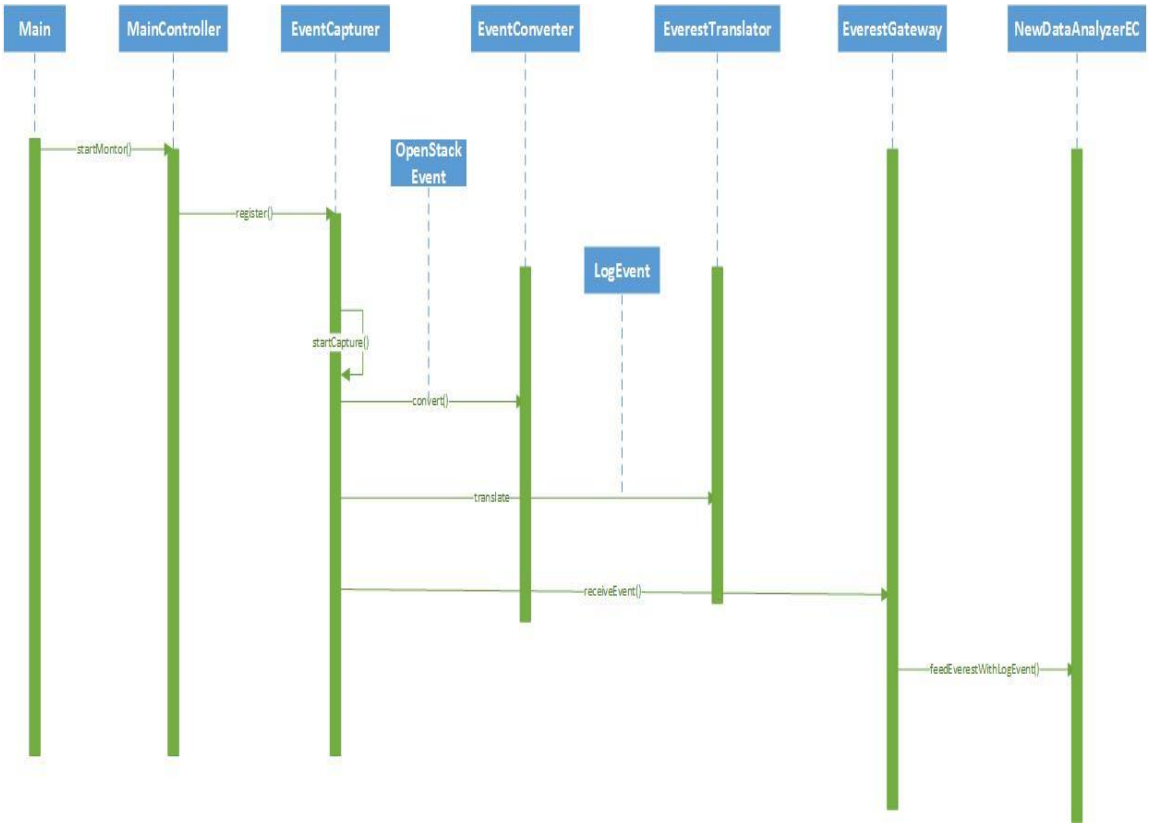
Analysis Class Diagram:



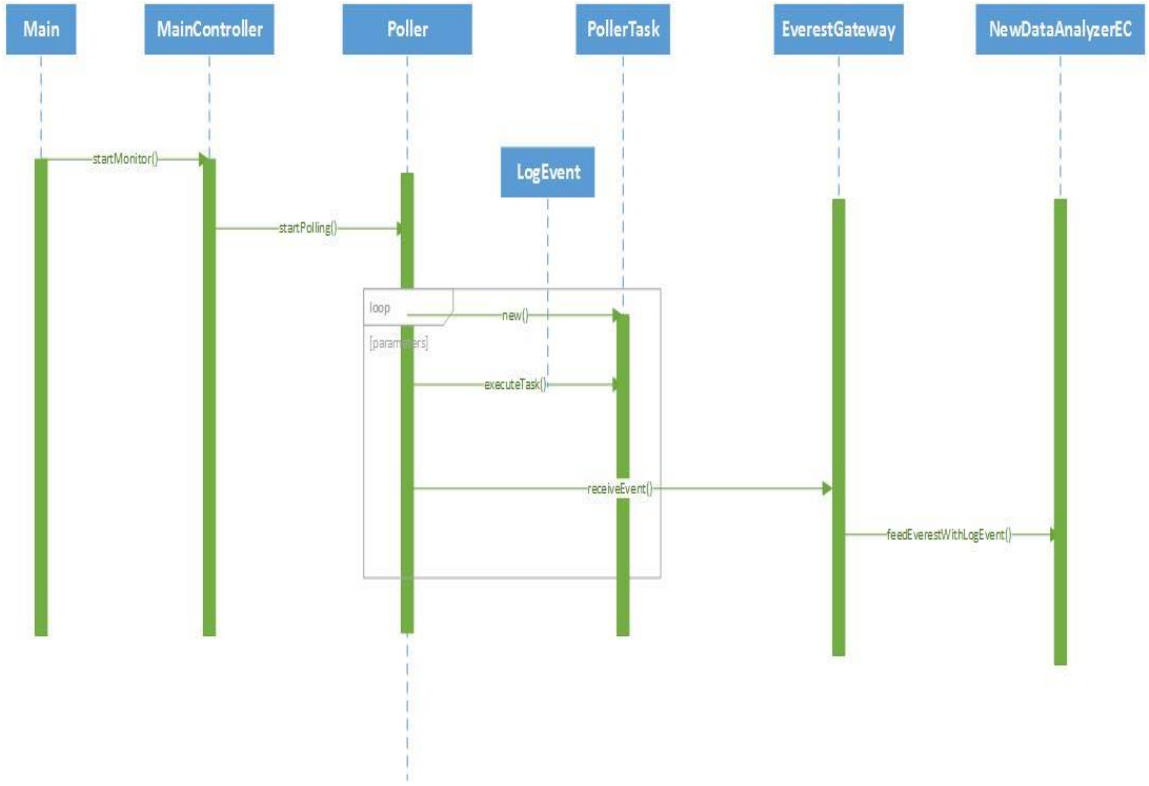
Submit Rule Sequence Diagram:



Event Capturer Sequence Diagram:



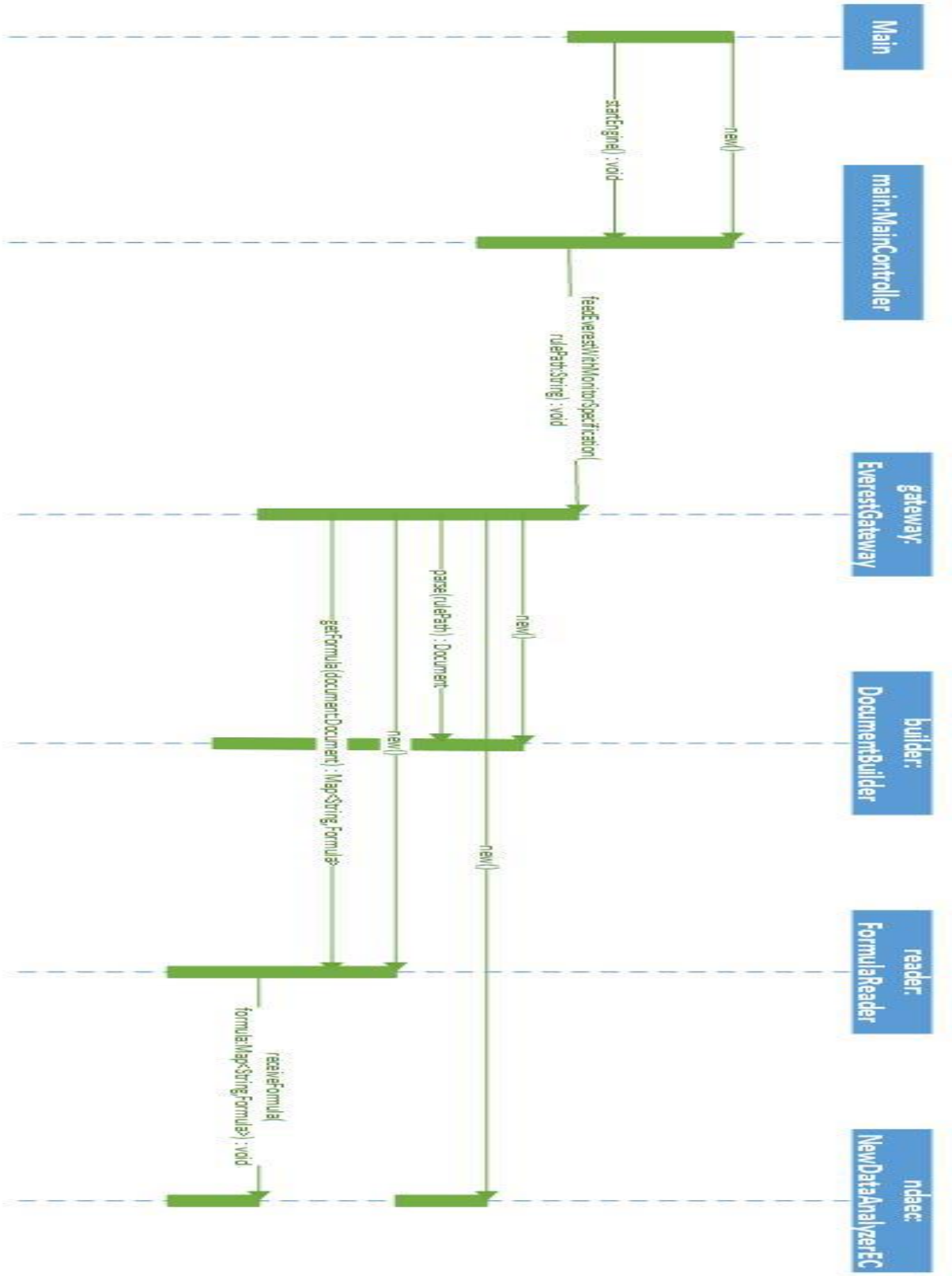
Poller Sequence Diagram:



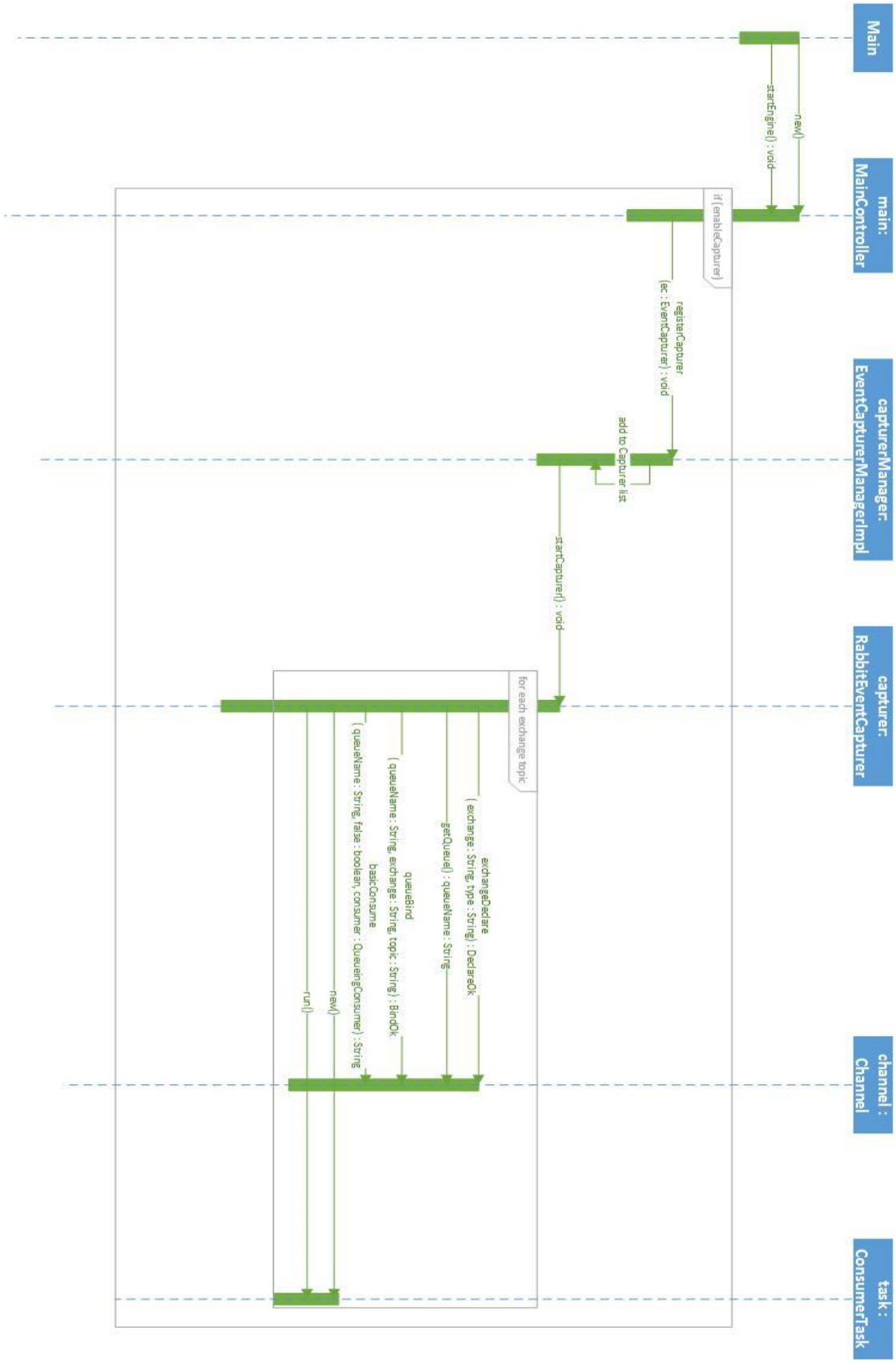
Appendix D: Design Document

Design Class Diagram:

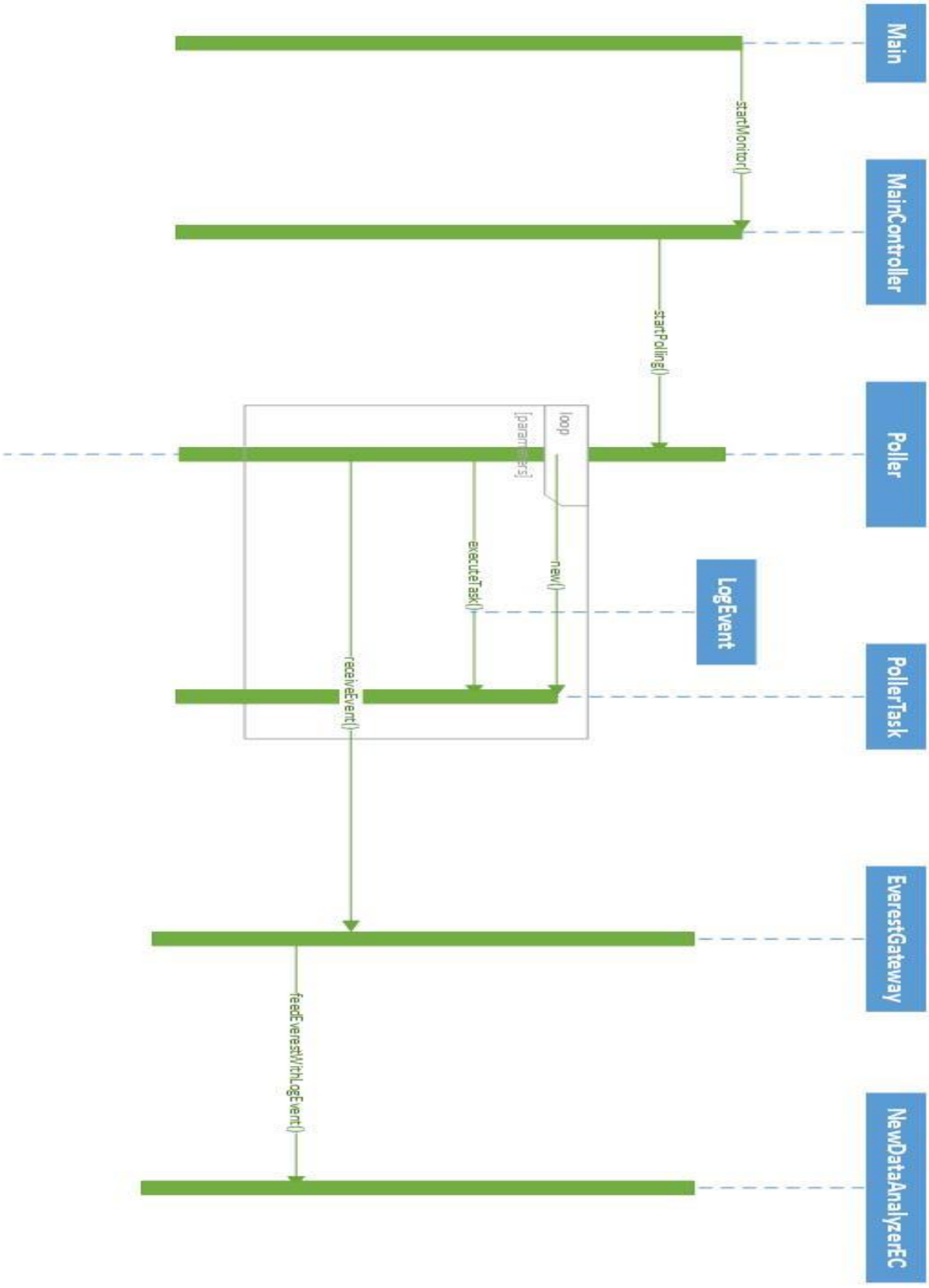
Submit Rule Design Sequence Diagram:



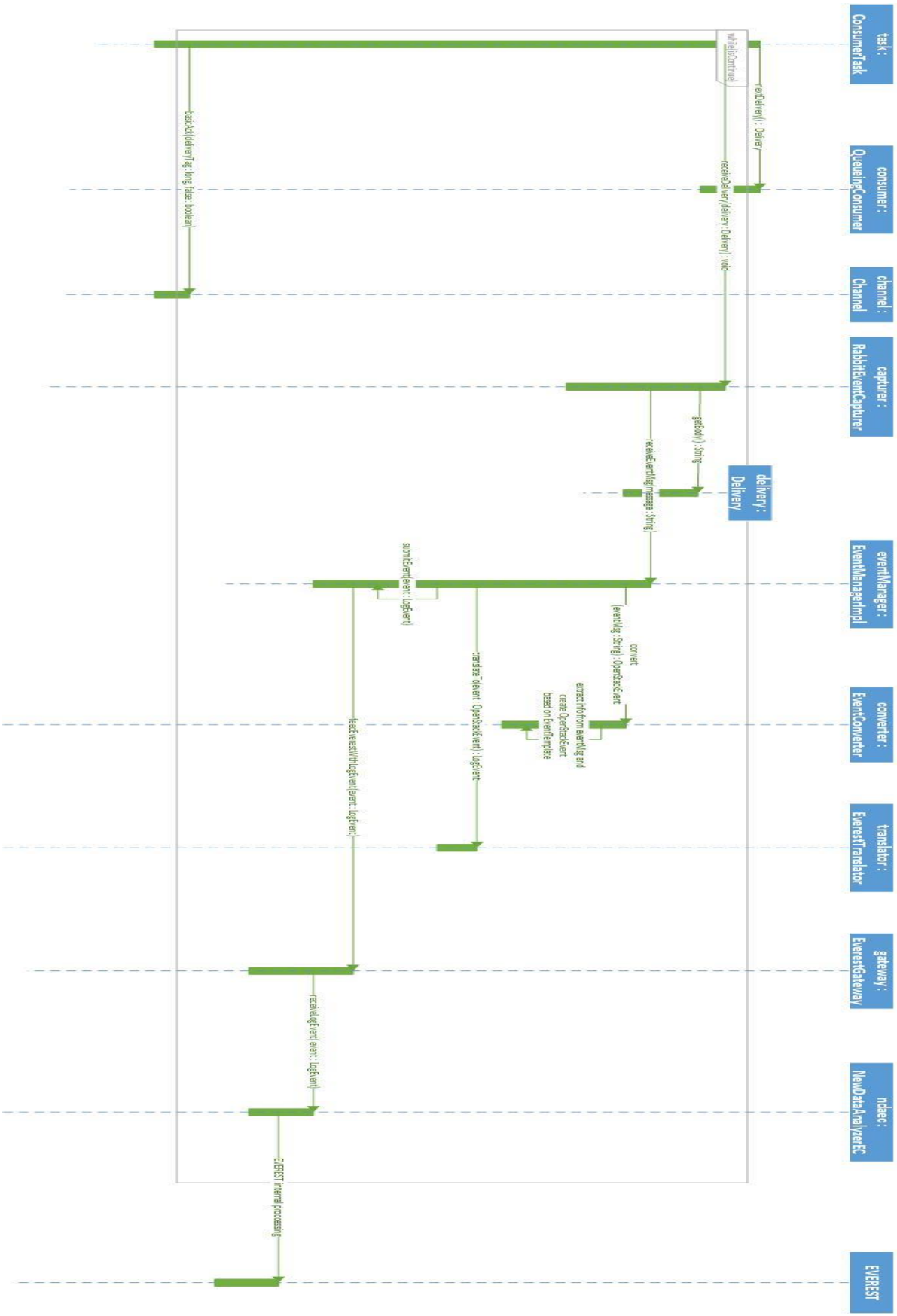
Starting EventCapturer Sequence Diagram:



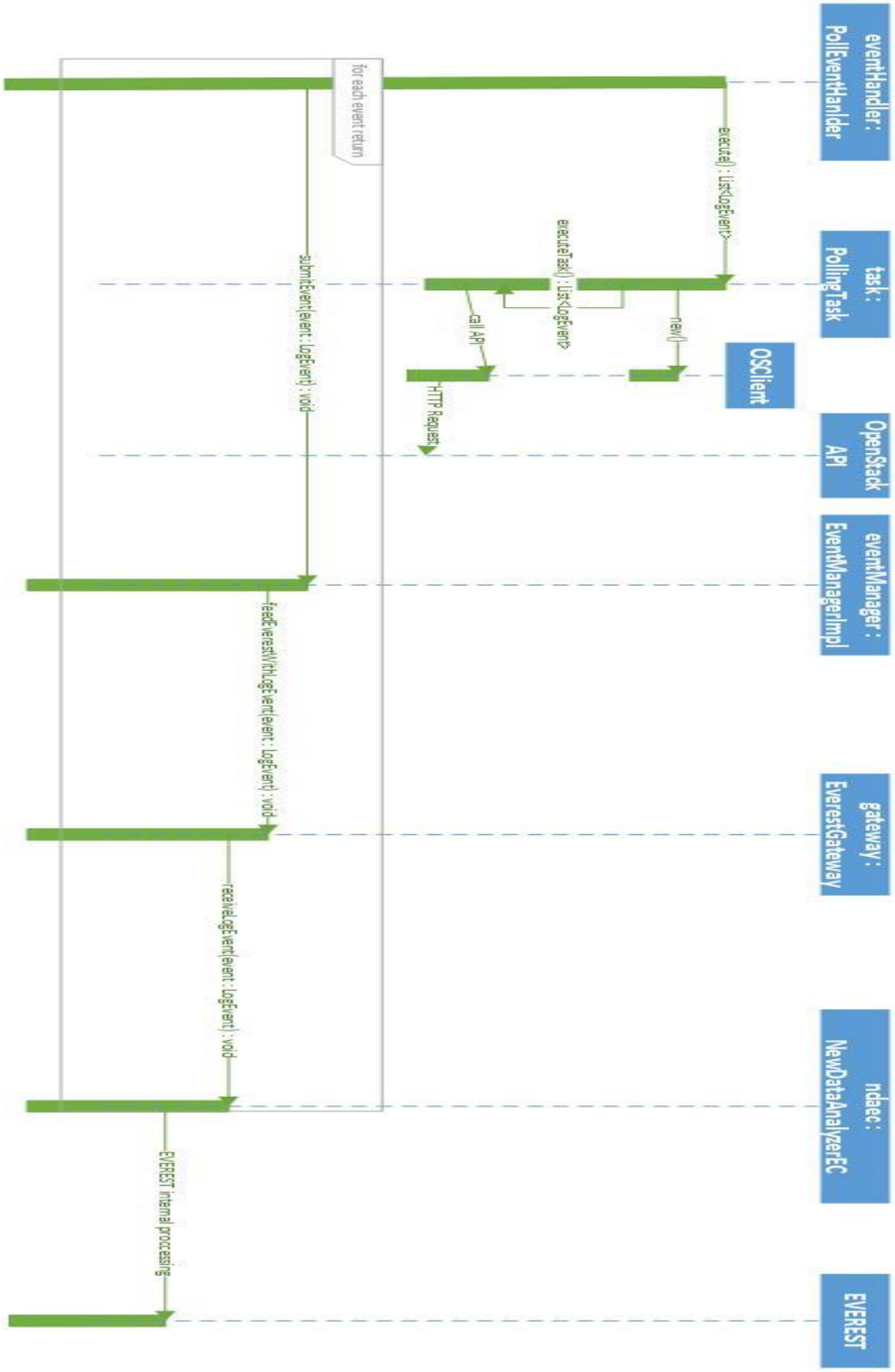
Starting Poller Sequence Diagram:



Event Capturer Sequence Diagram:



Poller Sequence Diagram:



Appendix E: Implementation Document

Spring configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config></context:annotation-config>

    <bean id="rabbitProps"
class="org.springframework.beans.factory.config.PropertyPlaceholderConf
igurer">
        <property name="location" value="classpath:META-
INF/rabbit.properties"></property>
    </bean>

    <bean id="rabbitConfigure"
class="city.os.core.capture.rabbit.RabbitConfigure">
        <property name="host" value="${rabbit.host}"/>
        <property name="username" value="${rabbit.username}"/>
        <property name="password" value="${rabbit.password}"/>
        <property name="exchange" value="${rabbit.exchange}"/>
    </bean>

    <bean id="eventCapturerManager"
class="city.os.core.capture.EventCapturerManagerImpl"></bean>
    <bean id="eventManager"
class="city.os.core.controller.EventManagerImpl">
    </bean>
    <bean id="pollerManager"
class="city.os.core.poll.PollingEngineImpl"></bean>

    <bean id="rabbitCapturer"
class="city.os.core.capture.rabbit.RabbitEventCapturer">
        <constructor-arg ref="rabbitConfigure"></constructor-arg>
    </bean>
    <bean id="eventConverter"
class="city.os.core.converter.EventConverterImpl"></bean>
    <bean id="everestTranslator"
class="city.os.core.translator.EverestTranslatorImpl"></bean>
    <bean id="pollerConsumer"
class="city.os.core.poll.PollingTaskConsumerImpl"></bean>
    <bean id="pollerProducer"
class="city.os.core.poll.PollingTaskProducerImpl"></bean>
    <bean id="taskGenerator"
class="city.os.core.poll.PollingTaskGenerator"></bean>
    <bean id="mainController"
class="city.os.core.controller.MainController"></bean>

    <bean id="everestGateway"
class="city.os.everest.EverestGateway"></bean>

</beans>
```

Log4j configuration file:

```
log4j.rootLogger=DEBUG, stdout, FILE

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
%5p (%L) - %m%n
#%d{yyyy-MM-dd HH:mm:ss} %5p (%c.java:%L) - %m%n

# Define the file appender
log4j.appender.FILE=org.apache.log4j.DailyRollingFileAppender
# Set the name of the file
log4j.appender.FILE.File=$${env:OSEI_HOME}/log/log.out

# Set the immediate flush to true (default)
log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode
log4j.appender.FILE.Threshold=debug

# Set the append to false, should not overwrite
log4j.appender.FILE.Append=true

# Set the DatePattern
log4j.appender.FILE.DatePattern='.' yyyy-MM-dd-a

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

Rabbit configuration files:

```
rabbit.host=localhost
rabbit.username=guest
rabbit.password=rabbit
rabbit.exchange=nova:notifications.info;keystone:notifications.info;cin
der:notifications.info;glance:notifications.info
```

Maven Pom file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.city.monitor.openstack-integration</groupId>
    <artifactId>OpenStackEverestIntegration</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>

        <!-- Generic properties -->
        <java.version>1.6</java.version>
```

```
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>

        <!-- Spring -->
        <spring-framework.version>3.2.3.RELEASE</spring-
framework.version>
        <slaatsoi.framework.version>0.1-
SNAPSHOT</slaatsoi.framework.version>

        <!-- Hibernate / JPA -->
        <hibernate.version>4.2.1.Final</hibernate.version>

        <!-- Logging -->
        <logback.version>1.0.13</logback.version>
        <slf4j.version>1.7.5</slf4j.version>

        <!-- Test -->
        <junit.version>4.11</junit.version>

    </properties>

    <dependencies>
        <!-- Spring and Transactions -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring-framework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring-framework.version}</version>
        </dependency>

        <!-- Logging with SLF4J & LogBack -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>${logback.version}</version>
            <scope>runtime</scope>
        </dependency>

        <!-- Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

        <!-- RAabbitMQ -->
        <dependency>
            <groupId>com.rabbitmq</groupId>
            <artifactId>amqp-client</artifactId>
```

```
        <version>3.3.5</version>
    </dependency>

    <!-- Logging -->
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.16</version>
    </dependency>

    <!-- JSON Parser -->
    <dependency>
        <groupId>com.googlecode.json-simple</groupId>
        <artifactId>json-simple</artifactId>
        <version>1.1</version>
    </dependency>

    <!-- MYSQL -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.13</version>
    </dependency>

    <!-- Openstack4J -->
    <dependency>
        <groupId>org.pacesys</groupId>
        <artifactId>openstack4j</artifactId>
        <version>1.0.1</version>
    </dependency>

    <!-- Test Artifacts -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>${spring-framework.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.0.1</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.osgi</groupId>
        <artifactId>spring-osgi-annotation</artifactId>
        <version>1.2.0</version>
        <type>jar</type>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>xerces</groupId>
```

```
            <artifactId>xercesImpl</artifactId>
            <version>2.9.1</version>
        </dependency>

        <dependency>
            <groupId>xalan</groupId>
            <artifactId>xalan</artifactId>
            <version>2.7.1</version>
        </dependency>

        <dependency>
            <groupId>javax.xml</groupId>
            <artifactId>jaxrpc-api</artifactId>
            <version>1.1</version>
        </dependency>

        <dependency>
            <groupId>org.apache.axis</groupId>
            <artifactId>axis</artifactId>
            <version>1.4</version>
        </dependency>

        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.0</version>
        </dependency>

        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-impl</artifactId>
            <version>2.1.5</version>
        </dependency>

        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-xjc</artifactId>
            <version>2.0.5</version>
        </dependency>

        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-libs</artifactId>
            <version>1.0.5</version>
        </dependency>

        <dependency>
            <groupId>com.sun.org.apache</groupId>
            <artifactId>jaxp-ri</artifactId>
            <version>1.4</version>
        </dependency>

        <dependency>
            <groupId>
                org.slasoi.monitoring-system.sla-level-monitoring.city
            </groupId>
            <artifactId>everest-core</artifactId>
            <version>${slaatsoi.framework.version}</version>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3</version>
                <configuration>
                    <source>${jdk.version}</source>
                    <target>${jdk.version}</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>appassembler-maven-plugin</artifactId>
                <configuration>
                    <programs>
                        <program>
                            <mainClass>city.os.main.Main</mainClass>
                            <name>main</name>
                        </program>
                    </programs>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Program shell file:

```
#!/bin/sh
# -----------------------------------------------------------------------
-------
#  Copyright 2001-2006 The Apache Software Foundation.
#
#  Licensed under the Apache License, Version 2.0 (the "License");
#  you may not use this file except in compliance with the License.
#  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
# -----------------------------------------------------------------------
-------
#
#   Copyright (c) 2001-2006 The Apache Software Foundation.  All rights
#   reserved.


# resolve links - $0 may be a softlink
PRG="$0"

while [ -h "$PRG" ]; do
  ls=`ls -ld "$PRG"`
  link=`expr "$ls" : '.*-> \(.*\)$'`
  if expr "$link" : '/.*' > /dev/null; then
```

```
    PRG="$link"
  else
    PRG=`dirname "$PRG"`/"$link"
  fi
done

PRGDIR=`dirname "$PRG"`
BASEDIR=`cd "$PRGDIR/.." >/dev/null; pwd`

# Reset the REPO variable. If you need to influence this use the
environment setup file.
REPO=


# OS specific support.  $var _must_ be set to either true or false.
cygwin=false;
darwin=false;
case "`uname`" in
  CYGWIN*) cygwin=true ;;
  Darwin*) darwin=true
           if [ -z "$JAVA_VERSION" ] ; then
             JAVA_VERSION="CurrentJDK"
           else
             echo "Using Java version: $JAVA_VERSION"
           fi
           if [ -z "$JAVA_HOME" ]; then
              if [ -x "/usr/libexec/java_home" ]; then
                  JAVA_HOME=`/usr/libexec/java_home`
              else

JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versions/${JAVA_V
ERSION}/Home
              fi
           fi
           ;;
esac

if [ -z "$JAVA_HOME" ] ; then
  if [ -r /etc/gentoo-release ] ; then
    JAVA_HOME=`java-config --jre-home`
  fi
fi

# For Cygwin, ensure paths are in UNIX format before anything is
touched
if $cygwin ; then
  [ -n "$JAVA_HOME" ] && JAVA_HOME=`cygpath --unix "$JAVA_HOME"`
  [ -n "$CLASSPATH" ] && CLASSPATH=`cygpath --path --unix "$CLASSPATH"`
fi

# If a specific java binary isn't specified search for the standard
'java' binary
if [ -z "$JAVACMD" ] ; then
  if [ -n "$JAVA_HOME"  ] ; then
    if [ -x "$JAVA_HOME/jre/sh/java" ] ; then
      # IBM's JDK on AIX uses strange locations for the executables
      JAVACMD="$JAVA_HOME/jre/sh/java"
    else
      JAVACMD="$JAVA_HOME/bin/java"
    fi
  else
```

```
    JAVACMD=`which java`
  fi
fi

if [ ! -x "$JAVACMD" ] ; then
  echo "Error: JAVA_HOME is not defined correctly." 1>&2
  echo "  We cannot execute $JAVACMD" 1>&2
  exit 1
fi

if [ -z "$REPO" ]
then
  REPO="$BASEDIR"/repo
fi

CLASSPATH="$BASEDIR"/etc:"$REPO"/org/springframework/spring-
context/3.2.3.RELEASE/spring-context-
3.2.3.RELEASE.jar:"$REPO"/org/springframework/spring-
aop/3.2.3.RELEASE/spring-aop-
3.2.3.RELEASE.jar:"$REPO"/org/springframework/spring-
beans/3.2.3.RELEASE/spring-beans-
3.2.3.RELEASE.jar:"$REPO"/org/springframework/spring-
core/3.2.3.RELEASE/spring-core-3.2.3.RELEASE.jar:"$REPO"/commons-
logging/commons-logging/1.1.1/commons-logging-
1.1.1.jar:"$REPO"/org/springframework/spring-
expression/3.2.3.RELEASE/spring-expression-
3.2.3.RELEASE.jar:"$REPO"/org/springframework/spring-
tx/3.2.3.RELEASE/spring-tx-
3.2.3.RELEASE.jar:"$REPO"/aopalliance/aopalliance/1.0/aopalliance-
1.0.jar:"$REPO"/org/slf4j/slf4j-api/1.7.5/slf4j-api-
1.7.5.jar:"$REPO"/ch/qos/logback/logback-classic/1.0.13/logback-
classic-1.0.13.jar:"$REPO"/ch/qos/logback/logback-core/1.0.13/logback-
core-1.0.13.jar:"$REPO"/org/hibernate/hibernate-
entitymanager/4.2.1.Final/hibernate-entitymanager-
4.2.1.Final.jar:"$REPO"/org/jboss/logging/jboss-logging/3.1.0.GA/jboss-
logging-3.1.0.GA.jar:"$REPO"/org/hibernate/hibernate-
core/4.2.1.Final/hibernate-core-
4.2.1.Final.jar:"$REPO"/antlr/antlr/2.7.7/antlr-
2.7.7.jar:"$REPO"/dom4j/dom4j/1.6.1/dom4j-
1.6.1.jar:"$REPO"/org/jboss/spec/javax/transaction/jboss-transaction-
api_1.1_spec/1.0.1.Final/jboss-transaction-api_1.1_spec-
1.0.1.Final.jar:"$REPO"/org/hibernate/javax/persistence/hibernate-jpa-
2.0-api/1.0.1.Final/hibernate-jpa-2.0-api-
1.0.1.Final.jar:"$REPO"/org/javassist/javassist/3.15.0-GA/javassist-
3.15.0-GA.jar:"$REPO"/org/hibernate/common/hibernate-commons-
annotations/4.0.1.Final/hibernate-commons-annotations-
4.0.1.Final.jar:"$REPO"/com/rabbitmq/amqp-client/3.3.5/amqp-client-
3.3.5.jar:"$REPO"/log4j/log4j/1.2.16/log4j-
1.2.16.jar:"$REPO"/com/googlecode/json-simple/json-simple/1.1/json-
simple-1.1.jar:"$REPO"/mysql/mysql-connector-java/5.1.13/mysql-
connector-java-
5.1.13.jar:"$REPO"/org/pacesys/openstack4j/1.0.1/openstack4j-
1.0.1.jar:"$REPO"/org/codehaus/jackson/jackson-mapper-
asl/1.9.13/jackson-mapper-asl-
1.9.13.jar:"$REPO"/org/codehaus/jackson/jackson-core-
asl/1.9.13/jackson-core-asl-
1.9.13.jar:"$REPO"/org/glassfish/jersey/core/jersey-client/2.0/jersey-
client-2.0.jar:"$REPO"/org/glassfish/jersey/core/jersey-
common/2.0/jersey-common-
2.0.jar:"$REPO"/javax/annotation/javax.annotation-
api/1.2/javax.annotation-api-1.2.jar:"$REPO"/org/glassfish/hk2/osgi-
```

```
resource-locator/1.0.1/osgi-resource-locator-
1.0.1.jar:"$REPO"/javax/ws/rs/javax.ws.rs-api/2.0/javax.ws.rs-api-
2.0.jar:"$REPO"/com/google/guava/guava/14.0.1/guava-
14.0.1.jar:"$REPO"/org/glassfish/hk2/hk2-api/2.1.88/hk2-api-
2.1.88.jar:"$REPO"/org/glassfish/hk2/hk2-utils/2.1.88/hk2-utils-
2.1.88.jar:"$REPO"/org/glassfish/hk2/external/javax.inject/2.1.88/javax
.inject-2.1.88.jar:"$REPO"/org/glassfish/hk2/hk2-locator/2.1.88/hk2-
locator-2.1.88.jar:"$REPO"/org/glassfish/hk2/external/asm-all-
repackaged/2.1.88/asm-all-repackaged-
2.1.88.jar:"$REPO"/org/glassfish/hk2/external/cglib/2.1.88/cglib-
2.1.88.jar:"$REPO"/org/glassfish/jersey/media/jersey-media-json-
jackson/2.0/jersey-media-json-jackson-
2.0.jar:"$REPO"/org/codehaus/jackson/jackson-jaxrs/1.9.11/jackson-
jaxrs-1.9.11.jar:"$REPO"/org/codehaus/jackson/jackson-
xc/1.9.11/jackson-xc-
1.9.11.jar:"$REPO"/com/google/code/findbugs/jsr305/2.0.0/jsr305-
2.0.0.jar:"$REPO"/org/apache/felix/maven-bundle-plugin/2.0.1/maven-
bundle-plugin-2.0.1.jar:"$REPO"/biz/aQute/bndlib/0.0.357/bndlib-
0.0.357.jar:"$REPO"/net/sf/kxml/kxml2/2.2.2/kxml2-
2.2.2.jar:"$REPO"/xmlpull/xmlpull/1.1.3.1/xmlpull-
1.1.3.1.jar:"$REPO"/org/apache/felix/org.osgi.core/1.0.0/org.osgi.core-
1.0.0.jar:"$REPO"/org/apache/felix/org.osgi.service.obr/1.0.1/org.osgi.
service.obr-1.0.1.jar:"$REPO"/org/apache/maven/maven-
project/2.0.7/maven-project-2.0.7.jar:"$REPO"/org/apache/maven/maven-
profile/2.0.7/maven-profile-2.0.7.jar:"$REPO"/org/apache/maven/maven-
plugin-registry/2.0.7/maven-plugin-registry-
2.0.7.jar:"$REPO"/org/apache/maven/maven-model/2.0.7/maven-model-
2.0.7.jar:"$REPO"/org/apache/maven/maven-plugin-api/2.0.7/maven-plugin-
api-2.0.7.jar:"$REPO"/org/apache/maven/maven-artifact/2.0.7/maven-
artifact-2.0.7.jar:"$REPO"/org/apache/maven/maven-archiver/2.2/maven-
archiver-2.2.jar:"$REPO"/org/apache/maven/maven-artifact-
manager/2.0.7/maven-artifact-manager-
2.0.7.jar:"$REPO"/org/apache/maven/maven-repository-
metadata/2.0.7/maven-repository-metadata-
2.0.7.jar:"$REPO"/org/apache/maven/maven-settings/2.0.7/maven-settings-
2.0.7.jar:"$REPO"/org/apache/maven/shared/maven-dependency-
tree/1.2/maven-dependency-tree-
1.2.jar:"$REPO"/org/apache/maven/wagon/wagon-provider-api/1.0-beta-
2/wagon-provider-api-1.0-beta-2.jar:"$REPO"/org/codehaus/plexus/plexus-
container-default/1.0-alpha-9-stable-1/plexus-container-default-1.0-
alpha-9-stable-1.jar:"$REPO"/classworlds/classworlds/1.1-alpha-
2/classworlds-1.1-alpha-2.jar:"$REPO"/org/codehaus/plexus/plexus-
archiver/1.0-alpha-7/plexus-archiver-1.0-alpha-
7.jar:"$REPO"/org/codehaus/plexus/plexus-utils/1.4.7/plexus-utils-
1.4.7.jar:"$REPO"/xerces/xercesImpl/2.9.1/xercesImpl-
2.9.1.jar:"$REPO"/xml-apis/xml-apis/1.3.04/xml-apis-
1.3.04.jar:"$REPO"/xalan/xalan/2.7.1/xalan-
2.7.1.jar:"$REPO"/xalan/serializer/2.7.1/serializer-
2.7.1.jar:"$REPO"/javax/xml/jaxrpc-api/1.1/jaxrpc-api-
1.1.jar:"$REPO"/org/apache/axis/axis/1.4/axis-
1.4.jar:"$REPO"/javax/xml/bind/jaxb-api/2.0/jaxb-api-
2.0.jar:"$REPO"/javax/xml/bind/jsr173_api/1.0/jsr173_api-
1.0.jar:"$REPO"/javax/activation/activation/1.1/activation-
1.1.jar:"$REPO"/com/sun/xml/bind/jaxb-impl/2.1.5/jaxb-impl-
2.1.5.jar:"$REPO"/com/sun/xml/bind/jaxb-xjc/2.0.5/jaxb-xjc-
2.0.5.jar:"$REPO"/com/sun/xml/bind/jaxb-libs/1.0.5/jaxb-libs-
1.0.5.jar:"$REPO"/com/sun/org/apache/jaxp-ri/1.4/jaxp-ri-
1.4.jar:"$REPO"/javax/xml/parsers/jaxp-api/1.4/jaxp-api-
1.4.jar:"$REPO"/org/slasoi/monitoring-system/sla-level-
monitoring/city/everest-core/0.1-SNAPSHOT/everest-core-0.1-
SNAPSHOT.jar:"$REPO"/org/slasoi/common/monitoringevent/monitoringevent/
```

```
0.1-SNAPSHOT/monitoringevent-0.1-
SNAPSHOT.jar:"$REPO"/org/dynamicjava/jsr/stax-api/1.0.0/stax-api-
1.0.0.jar:"$REPO"/org/dynamicjava/osgi/jaxb-api/1.0.0/jaxb-api-
1.0.0.jar:"$REPO"/org/dynamicjava/osgi/jaxb-impl/2.1.8/jaxb-impl-
2.1.8.jar:"$REPO"/org/dynamicjava/osgi/jpa-api/1.0.2/jpa-api-
1.0.2.jar:"$REPO"/org/apache/geronimo/specs/geronimo-
activation_1.1_spec/1.0.2/geronimo-activation_1.1_spec-
1.0.2.jar:"$REPO"/org/dynamicjava/osgi/osgi-commons/1.0.0/osgi-commons-
1.0.0.jar:"$REPO"/org/dynamicjava/osgi/da-launcher/1.0.2/da-launcher-
1.0.2.jar:"$REPO"/org/osgi/osgi-core/4.0/osgi-core-
4.0.jar:"$REPO"/stax/stax/1.2.0/stax-1.2.0.jar:"$REPO"/stax/stax-
api/1.0.1/stax-api-1.0.1.jar:"$REPO"/com/sun/tools/1.0/tools-
1.0.jar:"$REPO"/com/ibm/bpws4j/2.0/bpws4j-
2.0.jar:"$REPO"/org/city/monitor/openstack-
integration/OpenStackEverestIntegration/0.0.1-
SNAPSHOT/OpenStackEverestIntegration-0.0.1-SNAPSHOT.jar

ENDORSED_DIR=
if [ -n "$ENDORSED_DIR" ] ; then
  CLASSPATH=$BASEDIR/$ENDORSED_DIR/*:$CLASSPATH
fi

if [ -n "$CLASSPATH_PREFIX" ] ; then
  CLASSPATH=$CLASSPATH_PREFIX:$CLASSPATH
fi

# For Cygwin, switch paths to Windows format before running java
if $cygwin; then
  [ -n "$CLASSPATH" ] && CLASSPATH=`cygpath --path --windows
"$CLASSPATH"`
  [ -n "$JAVA_HOME" ] && JAVA_HOME=`cygpath --path --windows
"$JAVA_HOME"`
  [ -n "$HOME" ] && HOME=`cygpath --path --windows "$HOME"`
  [ -n "$BASEDIR" ] && BASEDIR=`cygpath --path --windows "$BASEDIR"`
  [ -n "$REPO" ] && REPO=`cygpath --path --windows "$REPO"`
fi

exec "$JAVACMD" $JAVA_OPTS  \
  -classpath "$CLASSPATH" \
  -Dapp.name="main" \
  -Dapp.pid="$$" \
  -Dapp.repo="$REPO" \
  -Dapp.home="$BASEDIR" \
  -Dbasedir="$BASEDIR" \
  city.os.main.Main \
  "$@"
```

Appendix F: Testing Document.

Testing document includes one OpenStack.xml template used for testing.The template include 4 rules which were used in our test.

Rule 1: R1.Nova.Openstack – the time taken for creating an instance must be less than or equal 5000ms.

Rule2: R1.Cinder.Openstack – the time taken for creating a volume must be less than or equal 3000ms.

Rule3: R1.Poller.QuotaCheck – the quota for a specified tenant must be 51200 MB RAM, 20 CPUs, 10 instances, 10 floatIps.

Rule4: R0.Poller.CheckInstanceError- there must be no instances which have status Error for a specified tenant.

The xml file is included in the submitted CDs and zip file of this project.

Appendix G: Source Code Document

The source code for the application can be found on the submitted CDs and zip file.

Appendix H: User and Developer Guide

The user and developer guide for the application can be found on the submitted CDs and zip file.