Final Year Project Report

Inspector Cloud

Project Team

Muhammad Awais Ali(i12-0429)

Muhammad Umar Yousaf(i12-0492)

Shahzaib Hayat Khan(i12-0184)

Supervised by

Dr. Uzair Khan

Department of Computer Science

National University of Computer and Emerging Sciences,

Islamabad, Pakistan

2015-2016

1 CONTENTS

2 INTRODUCTION:

We are going to develop an infrastructure that will enable us to collect any desired information of the OpenStack projects to reliably monitor physical and virtual resources comprised of deployed clouds and virtual machines running on them for monitoring, metering, billing and maintenance purpose.

Open stack in an open source cloud computing platform that provide services to both private and public clouds regardless of their need for size and is easy to implement and is also scalable.

It is a cloud operating system that manages a large set of computational, networking and storage resources for a data center via a single web interface called horizon or dashboard.

Traditional data centers heavily rely upon the hardware and physical servers. This infrastructure is available for a singular purpose and has limited storage and computational capacity. These datacenters are also restricted by the physical size of the space they occupy.

OpenStack simply solves all of the problems faced by the traditional data centers. It has an option to download, modify and deploy custom applications regardless of the resources they need to run. It even provides full responsibility to organizations for its deployment, maintenance and support to develop private clouds. Thus it reduces the demand of managing and supporting the infrastructure to run private clouds within an organization.

3 PROJECT VISION:

3.1 PROBLEM STATEMENT:

OpenStack consists of many components like swift, glance, nova, cinder etc. these components have many parameters which should be checked constantly to have smooth working of the virtual machine running on the OpenStack like cpu , memory, storage etc. These components and their parameters can be monitored via a non-graphical user interface or terminal through commands which is really a hectic method to get the status of every component running in an OpenStack cloud.
Thus nowadays no such tool or application exists that can effectively monitor all the components and return result to the user.

3.2 BUSINESS OPPORTUNITY:

This tool will provide administrators of the OpenStack cloud to easily monitor the components of OpenStack via a graphical user interface. They don't need to write every single command to check the status of any of the components of OpenStack cloud like currently how many virtual machines are running on OpenStack cloud, what is the status of cpu consumption, how much storage has been consumed by the virtual machines e.tc For all these question administrator of OpenStack cloud does not need to write commands now status of every single component of OpenStack will be visible to the administrator upon a single click. This tool will even solve the billing problem for most of the users as it will generate billing details on the consumption of the resources used. Thus it's a golden opportunity for us because no such tool or application exists providing such functionalities.

3.3 OBJECTIVE:

To develop an infrastructure that will enable us to collect any desired information of the OpenStack projects to reliably monitor physical and virtual resources comprised of deployed clouds and virtual machines running on them for monitoring, metering billing and maintenance purpose.

3.4 PROJECT SCOPE:

Our product will mainly focus upon the monitoring of various components of OpenStack cloud, monitoring its physical and virtual resources, their metering and to produce billing information and to help the administrator in the maintenance of the OpenStack cloud.

3.5 CONSTRAINTS:

- This tool is applicable for the monitoring of OpenStack cloud running different virtual machines.
- It will generate billing information only for the resources being used by different virtual machines running on OpenStack.

4 STAKE HOLDER DESCRIPTION:

4.1 STAKE HOLDER SUMMARY:

User: people who usually work in data centers which implements open stack cloud like data center administrators, who usually monitors the resource utilization in these centers and perform any maintenance tasks.

4.2 KEY HIGH LEVEL GOALS AND PROBLEMS OF STAKE HOLDERS:

Now a day no such tool exists that allows OpenStack cloud administrators to effectively get the status and billing information of all the resources being utilized by the open stack cloud. Thus it provides an attractive and easy to use graphical user interface for the administrators to perform metering, rating and billing tasks effectively.

5 SOFTWARE REQUIREMENT SPECIFICATIONS

5.1 LIST OF FEATURES:

- Admin could find all the resource quotas available to a particular tenants including floating ips, vms, cores, and ram e.tc.
- Admin has the functionality to know all the statistics related to all the meters available associated with an instance as well as a component.
- Admin could get the information about all the events that have been occurred since the nodes are started including when a particular instance booted, restarted, and updated etc.
- Admin could associate an alarm with a desired meter by providing a certain threshold value, on crossing which the alarm will be raised and particular action could be performed automatically including booting another instance to balance the load.
- Admin has the authority to create a customized bill category that could be used to bill tenants depending upon the flavors associated with them.
- Admin could generate invoice for the tenants utilizing its resources on behalf of particular flavors available to them.

- Admin has a sort of troubleshooting functionality as all the errors or warnings are being parsed on a single place that could provide him the issues related to all the components.

5.2 Functional Requirement

5.2.1 Monitor VM Statistics:

Admin could have the information about all the resource quotas associated with a particular tenant that could tell him how much of the resources are being consumed out of the available.

5.2.2 Create Customized Bill:

Admin could create a certain customized bill category for billing its tenants after setting his/her desired pricing.

5.2.3 Generate Bill:

After having one or more billing categories available an admin could generate invoices for all the tenants to charge for the services provided.

5.2.4 Debugging:

In order to troubleshoot any issue and instead of the only available but a non-trivial way to finding the issue admin could have a place where all the error or warnings are waiting to tell the issuing node.

5.3 Non-functional requirements:

5.3.1 Availability:

The application must be available at all times to the user. For this purpose we will need a backup server in case of any crash.

5.3.2 Easy to use:

The application must provide the best user experience to the user as user will be using it a lot so we wouldn’t want user to get bored. For this purpose we will design the interface according to HCI rules and will make it as perfect as possible.

5.3.3 Reliability:

The application must be reliable and error free. And in case of any malfunction, it must have the ability to recover soon. Data will be successfully transferred from server to the user.

5.3.4 Performance:

The response time must be low. Application should be highly responsive.

5.3.5 Security:

The data of the user must

6 HIGH LEVEL USE CASES

6.1 MONITOR VM STATISTICS:

Use case	Monitor VM statistics
Actor	Admin
Type	Primary
	Admin will select the resource quotas to get current status like how many resources are used and how many of them are free. Resources includes (Bandwidth, IPs, Storage and Memory). This is like an eagle-eye view of our whole cloud and machines attached to it.

6.2 GENERATE BILL:

Use case	Generate Bill
Actor	Admin
Type	Primary
	Admin will select generate bill and resultantly system will ask the admin to select the virtual machine whom he want to generate bill and then admin have to select the desired (admin-defined) bill category to apply on it to bill its user and then system will apply the rate defined in that strategy to bill its client.

6.3 CREATE CUSTOMIZED BILL:

Use case	Create Customized Bill
Actor	Admin
Type	Primary
	Admin will select create customized bill and thus system will allow admin to make his own bill scheme using available flavors for Virtual Machine. Admin can store defined his own rates to different flavors and then use this to bill his users. Admin will be able to store as many bill schemes as possible with different rates and categories and use that to bill its client.

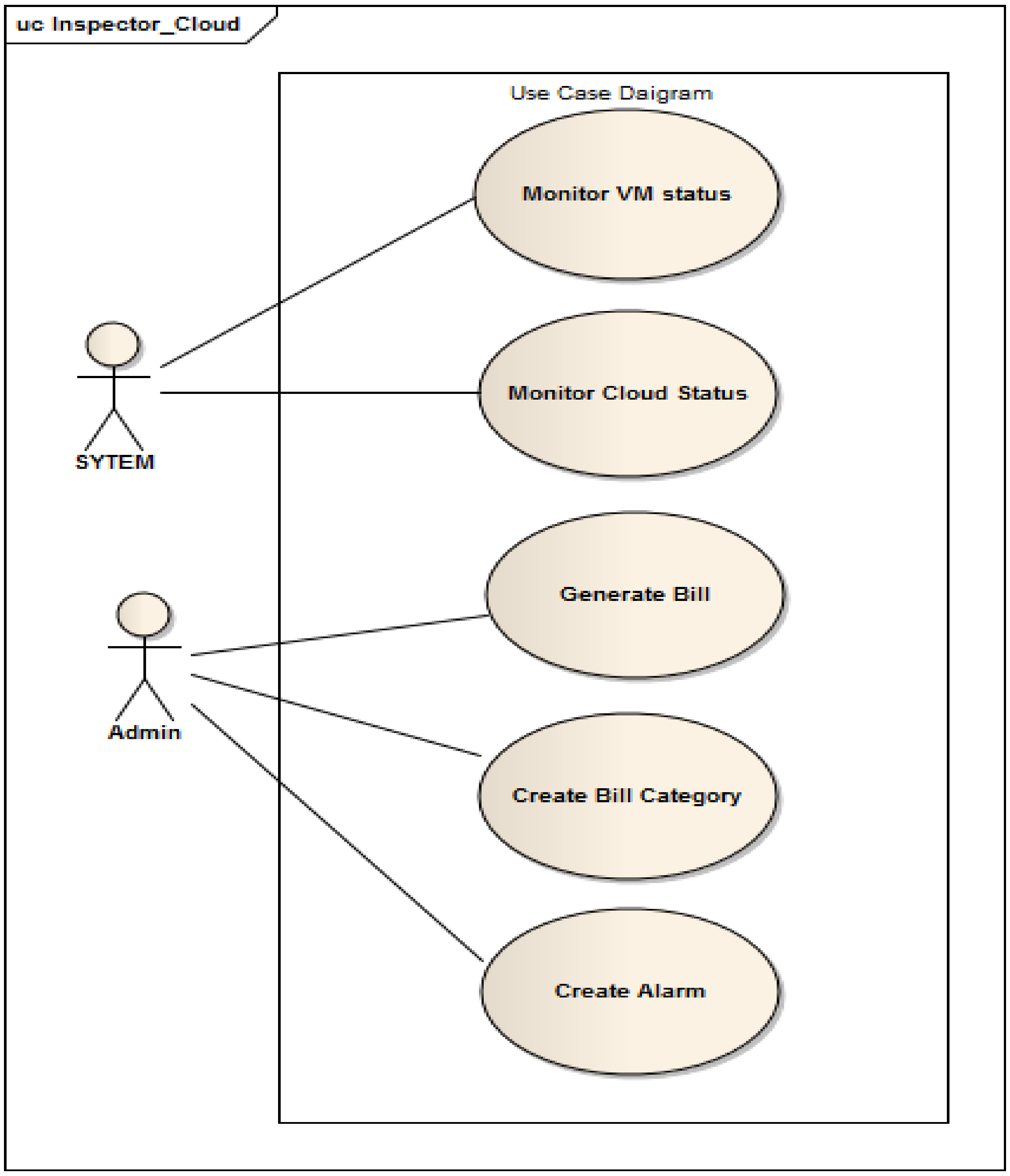
6.4 CREATE ALARM:

Use case	Create Alarm
Actor	Admin
Type	Primary
	Admin will be able to apply certain threshold value on the usage on its client machines so when those machines crosses that thresh value it will notify admin and will perform the defined task upon triggering alarm.

6.5 MONITOR CLOUD STATUS:

Use case	Monitor Cloud Status
Actor	Admin
Type	Primary
	Admin will select debugging and as a result of which system will parse all logs of OpenStack components for finding errors and critical warnings that can cause serious trouble to system so the admin don't have to look for issues in files for himself. System will show him the detailed view with all the errors and warning of separate components in separate view box.

7 USE CASE DIAGRAM



8 EXPANDED USE CASES

8.1 MONITOR VM STATISTICS

Scope	InspectorCloud
Use case	Monitor VM Statistics
Level	User goal
Primary Actor	Admin
Stakeholders and Interests	Admin, Tenants
Preconditions	None
Postconditions	System responds with the resource quotas of the corresponding tenant including.

Main Success Scenario:

Actor	System
1. Admin selects resource quotas for viewing resources associated with the corresponding tenant.	
	2. System displays all the pie charts of resources like cores, ram, floating ips, vms, security groups, key pairs

Extensions:

None

8.2 MONITOR CLOUD STATUS

Scope	InspectorCloud
Use case	Monitor Cloud Status
Level	User goal
Primary Actor	Admin
Stakeholders and Interests	Admin, Tenants
Preconditions	None
Postconditions	System responds with the errors and warning if available anywhere in the datacenter.

Main Success Scenario:

Actor	System
1. Admin selects debugging for viewing or examining any of the errors or warnings available on any of the node in the datacenter.	
	2. System displays all the log files that could possibility report for the desired errors or warnings after parsing files like ceilometer.log, ceilometer-alarm-notifier.log, and ceilometer-api.log e.tc.

Extensions:

None

8.3 CREATE ALARM

Scope	InspectorCloud
Use case	Create Alarm
Level	User goal
Primary Actor	Admin
Stakeholders and Interests	Admin, Tenants
Preconditions	At least one VM must be running
Postconditions	An Alarm is created successfully on the desired meter value.

Main Success Scenario:

Actor	System
1. Admin selects create alarm for making a desired action.	
	2. System presents user a form containing required fields to be filled for making an alarm to trigger on the desired meter if it crosses the provided threshold.
3. Admin fills in the required fields including the meter name and a specific threshold value for making an alarm to trigger after crossing it.	
	4. System creates an alarm on the provided meter and provides feed back to the user.

Extensions:

3a User fill fields like instance-name, alarm-name, meter-name, threshold, evaluation period, time period incorrect or out of domain

1. System will highlight the text box and ask for correct attribute.

2. User repeat the step

3b User left mandatory text box empty.

1. System will highlight text box.

2. System didn't accept form except those information

4a. System disconnect while submitting form

1. System reload the registration form.

8.4 CREATE CUSTOMIZED BILL

Scope	InspectorCloud
Use case	Create Customized Bill
Level	User goal
Primary Actor	Admin
Stakeholders and Interests	Admin, Tenants
Preconditions	At least one flavor must be available showing resources scheme
Postconditions	A customized bill is created that would be available anytime we will be billing tenants.

Main Success Scenario:

Actor	System
1. Admin selects create customized bill	
	2. System presents user a form containing required fields to be filled for associating desired price on the resources schemes available known as flavors.
3. Admin fills in the required fields including category name and a specific pricing value with each of the flavor available.	
	4. System creates a bill-category with the provided name after associating pricing values with all the available flavors.

Extensions:

3a User fill fields name and pricing value incorrect or out of domain

1. System will highlight the text box and ask for correct attribute.

2. User repeat the step

3b User left mandatory text box empty.

1. System will highlight text box.

2. System didn’t accept form except those information

4a. System disconnect while submitting form

1. System reload the registration form.

8.5 Generate Bill

Scope	InspectorCloud
Use case	Generate Bill
Level	User goal
Primary Actor	Admin
Stakeholders and Interests	Admin, Tenants
Preconditions	At least one billing category and instance must be available
Postconditions	Bill will be generated for the provided instance on the basis of selected bill category.

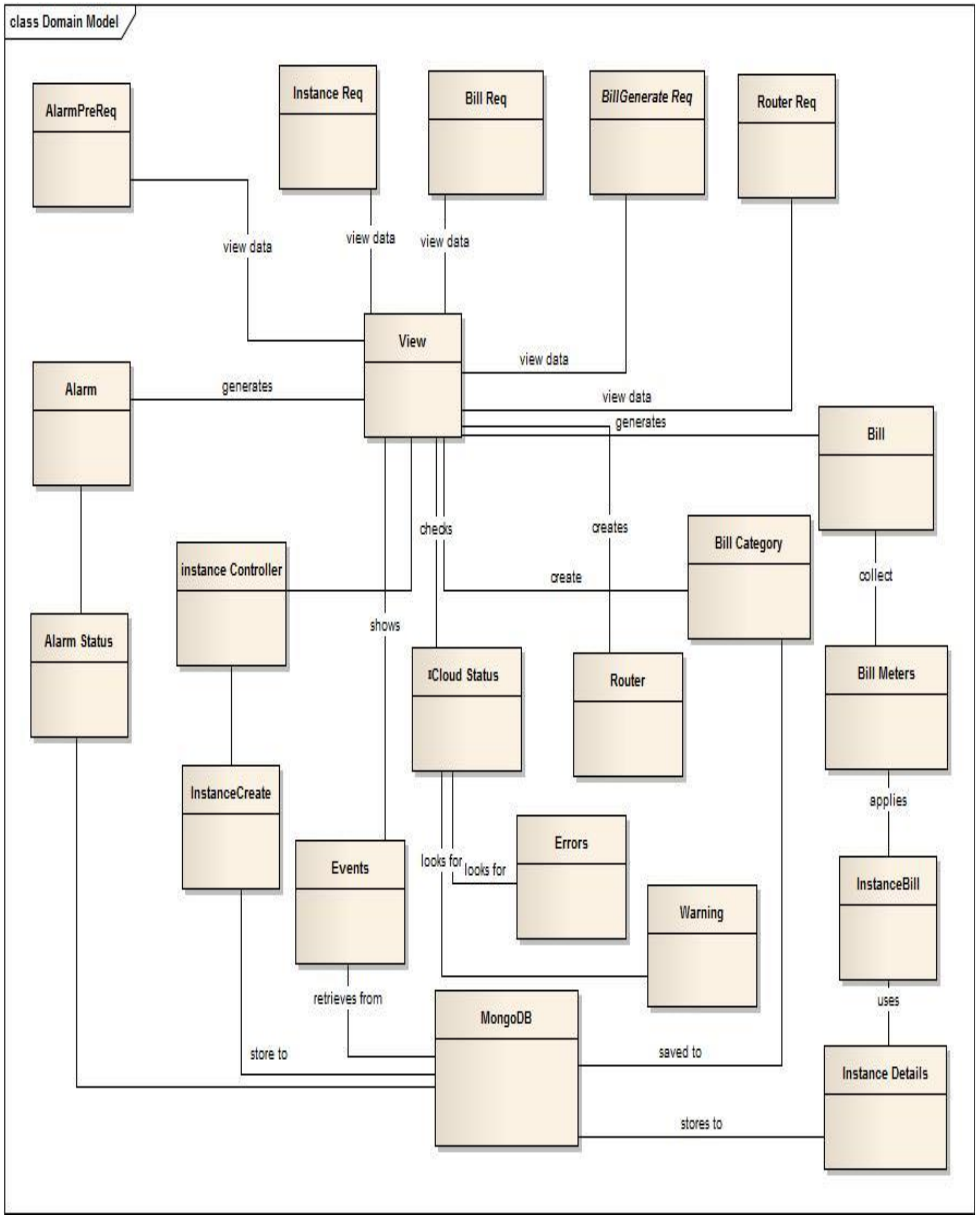
Main Success Scenario:

Actor	System
1. Admin selects generate bill for billing a tenant	
	2. System asks the user for selecting required instance name and billing category for making bill.
3. Admin provides with the required information including instance name and billing category.	
	4. System generates the invoice for billing purposes showing all the information available regarding its tenant and its instance flavor and also provide the option for printing.

Extensions:

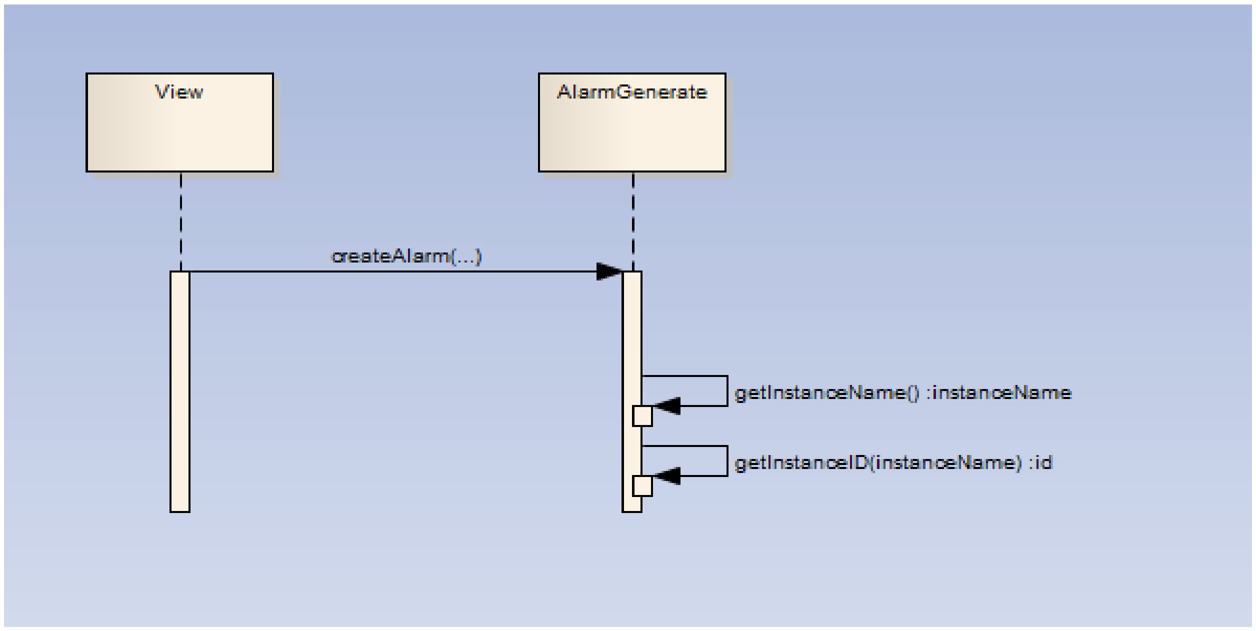
None

9 DOMAIN MODEL

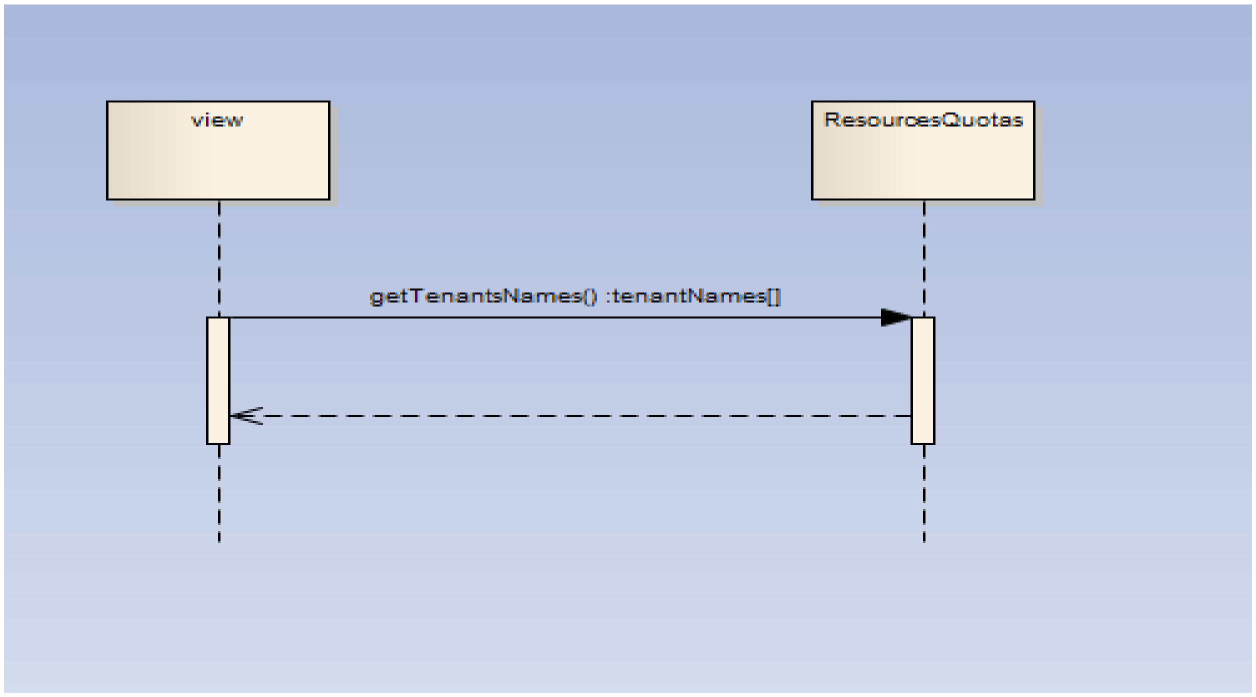


10 SYSTEM SEQUENCE DIAGRAMS

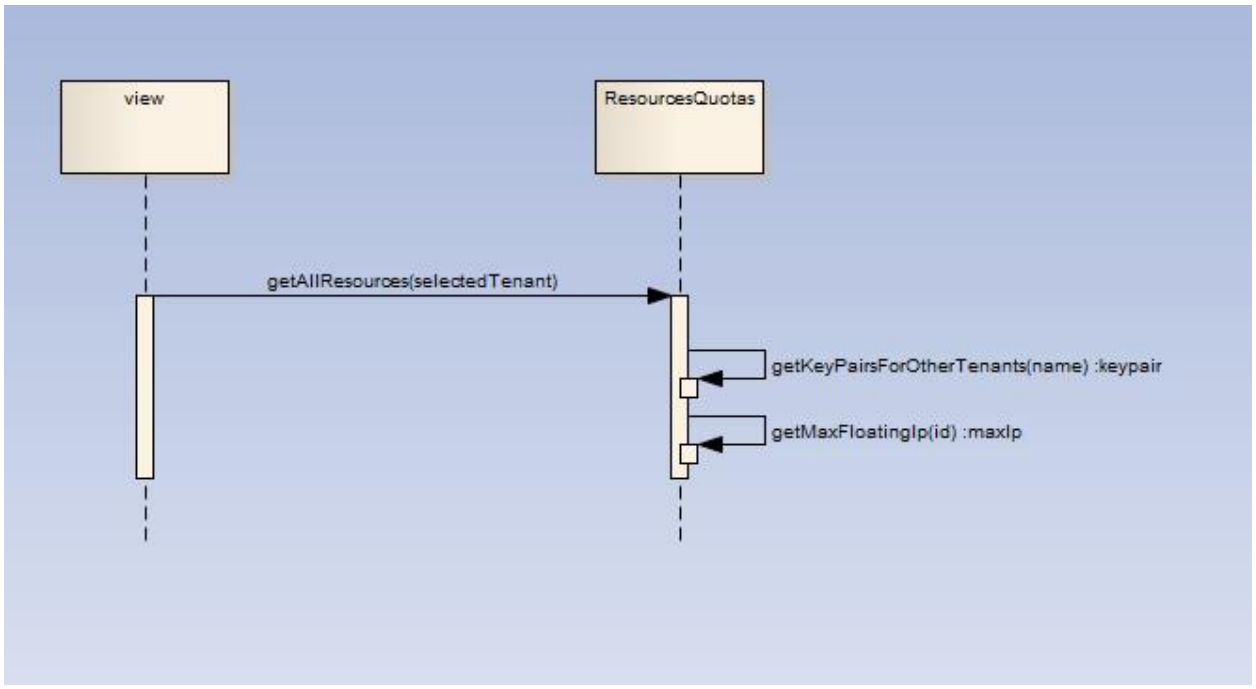
10.1 ALARM_1


View
AlarmGenerate
createAlarm(...)
getInstanceName() :instanceName
getInstanceID(instanceName) :id

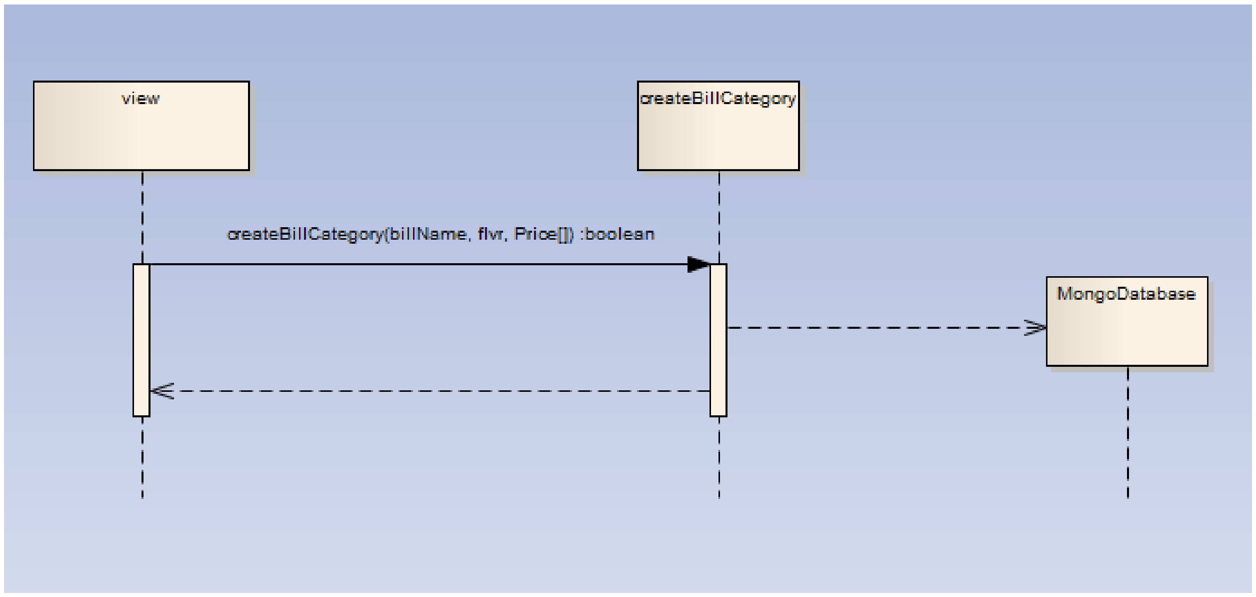
10.2 CLOUD_STATS_1



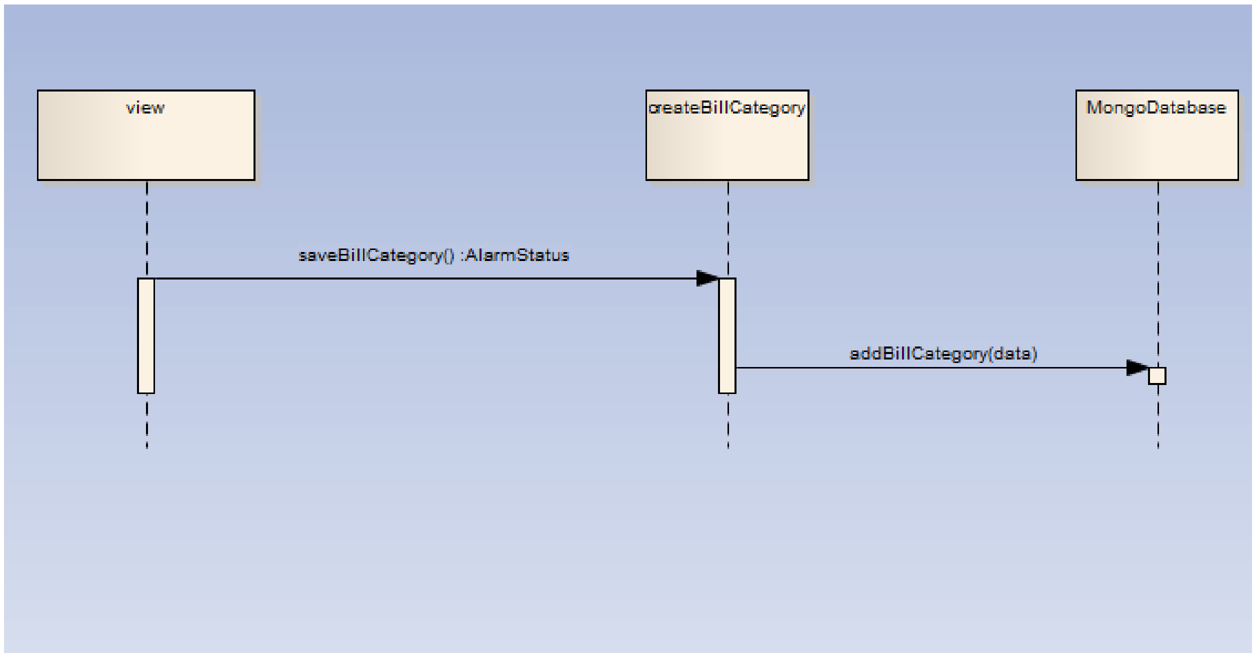
10.3 CLOUD_STATS_2


view
ResourcesQuotas
getAllResources(selectedTenant)
getKeyPairsForOtherTenants(name) :keypair
getMaxFloatingIp(id) :maxIp

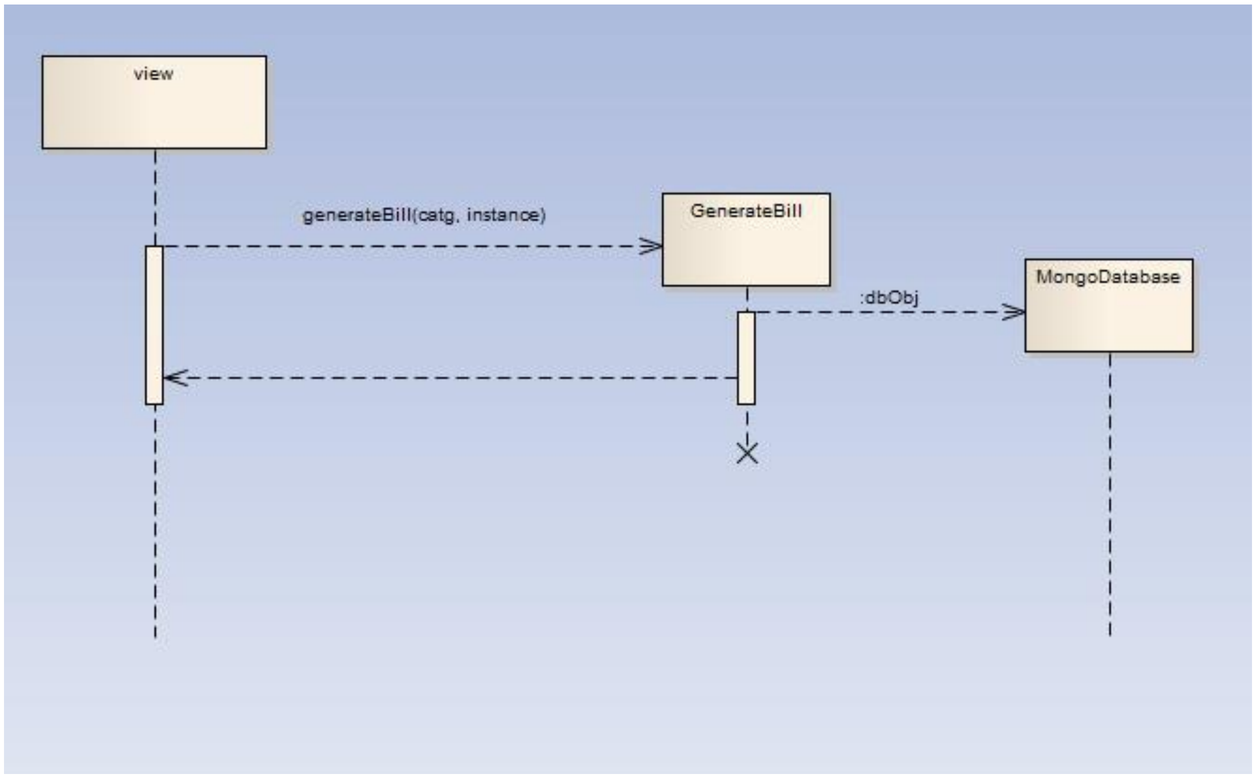
10.4 CREATE_BILL_CAT_1



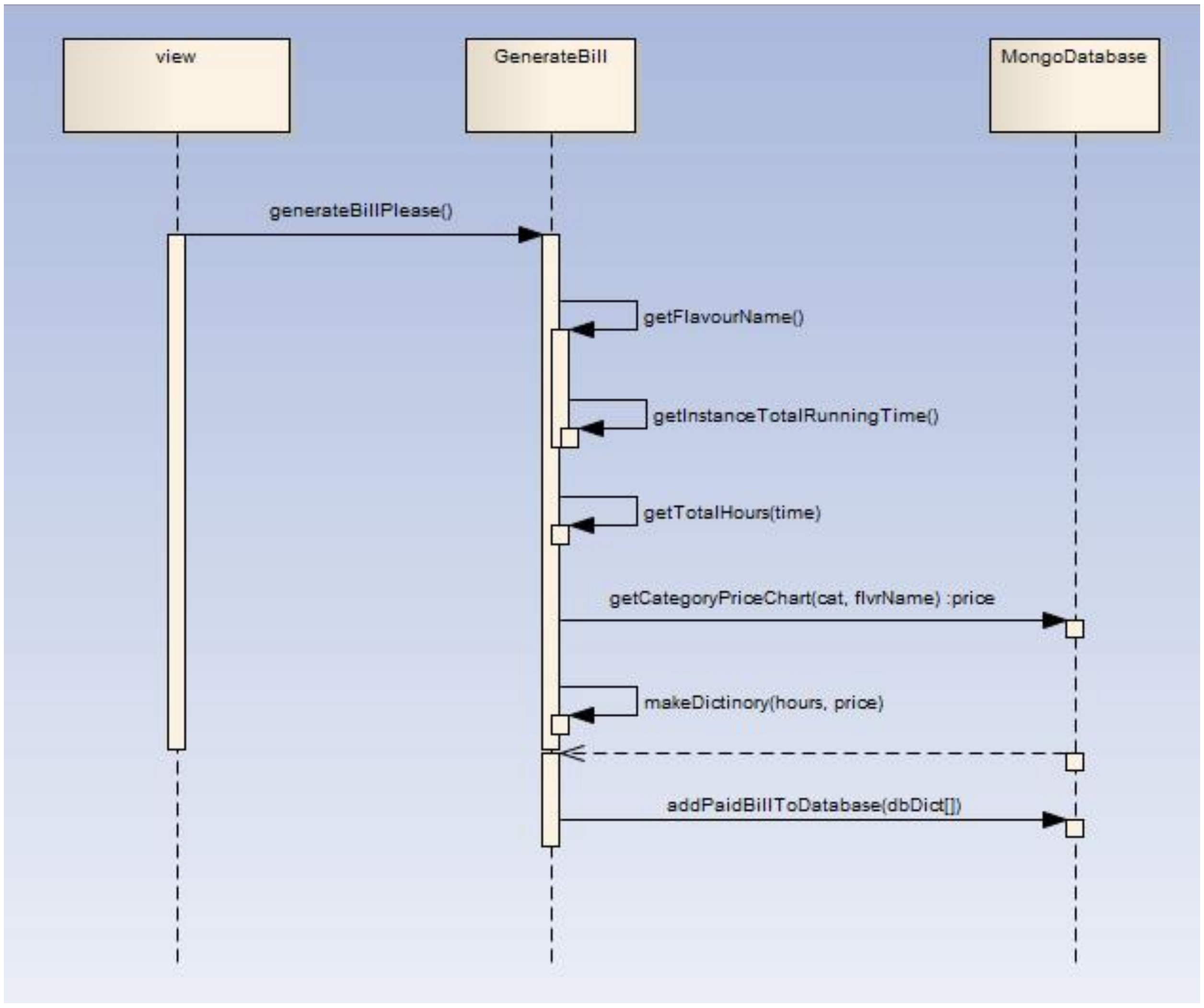
10.5 Create_Bill_Cat_2



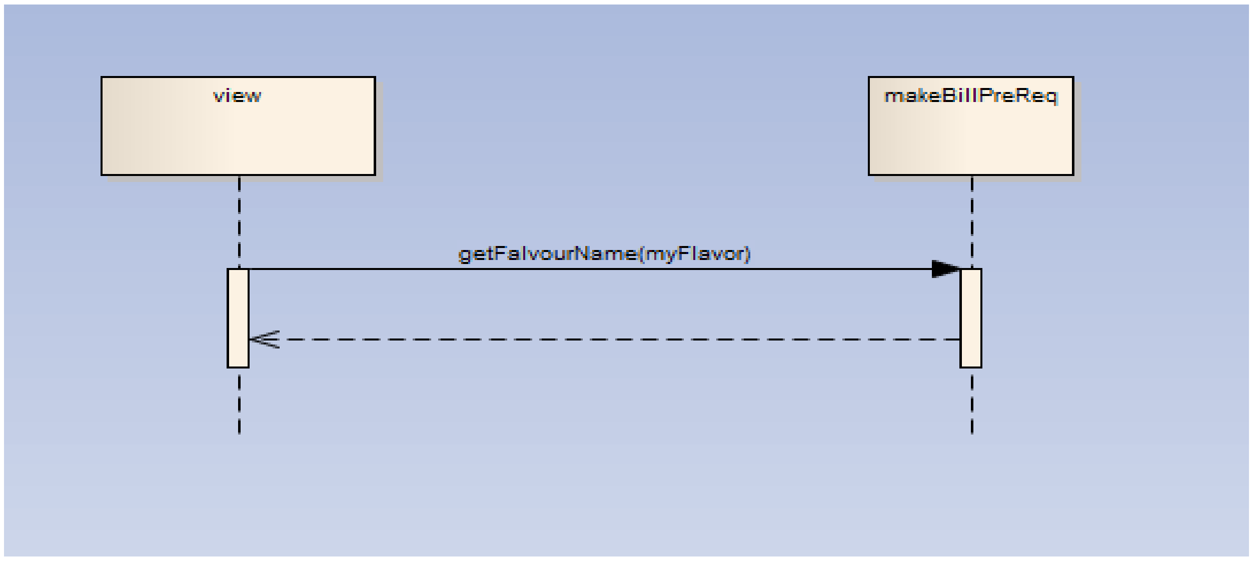
10.6 Generate_Bill_1



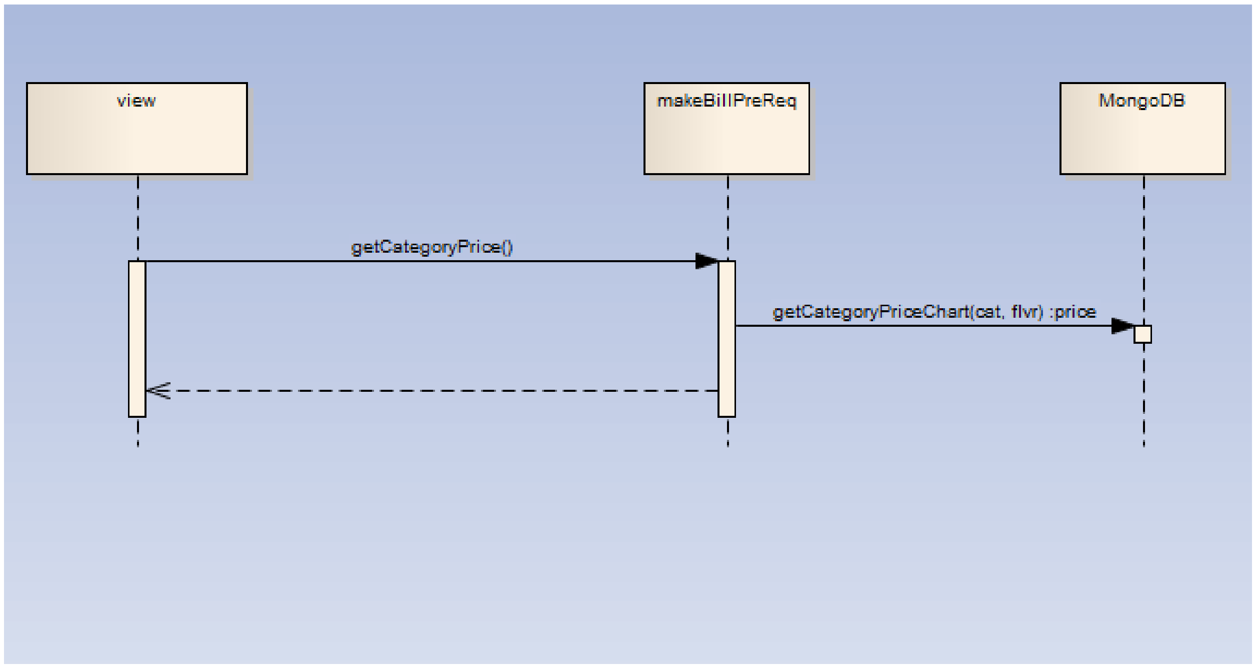
10.7 GENERATE_BILL_2


view
GenerateBill
MongoDatabase
generateBillPlease()
getFlavourName()
getInstanceTotalRunningTime()
getTotalHours(time)
getCategoryPriceChart(cat, flvrName) :price
makeDictinory(hours, price)
addPaidBillToDatabase(dbDict[])

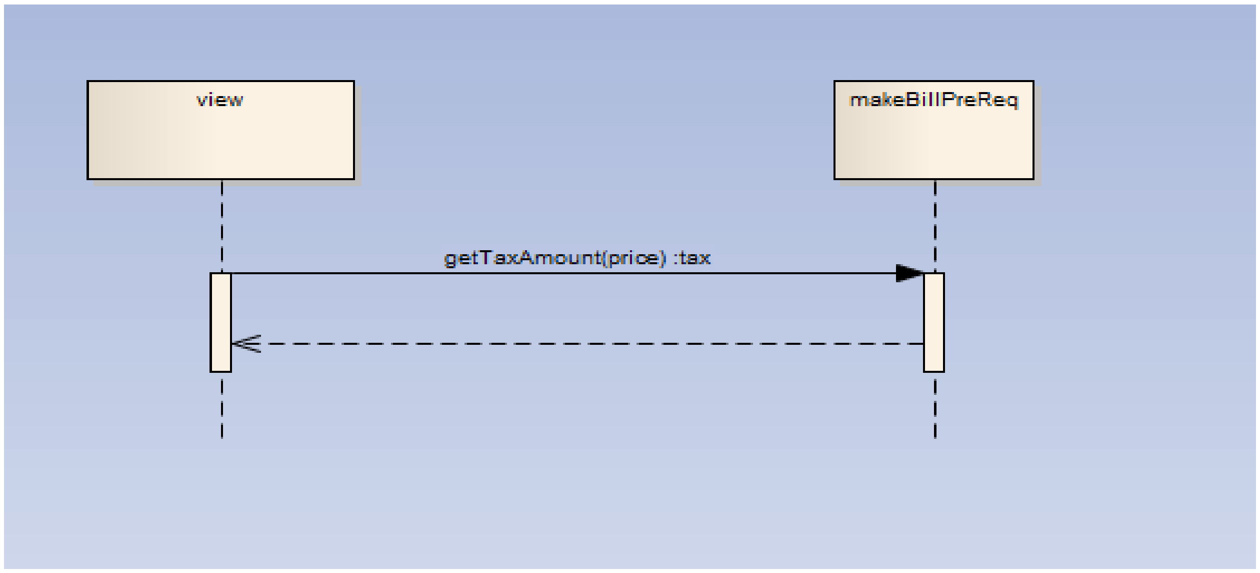
10.8 Generate_Bill_3



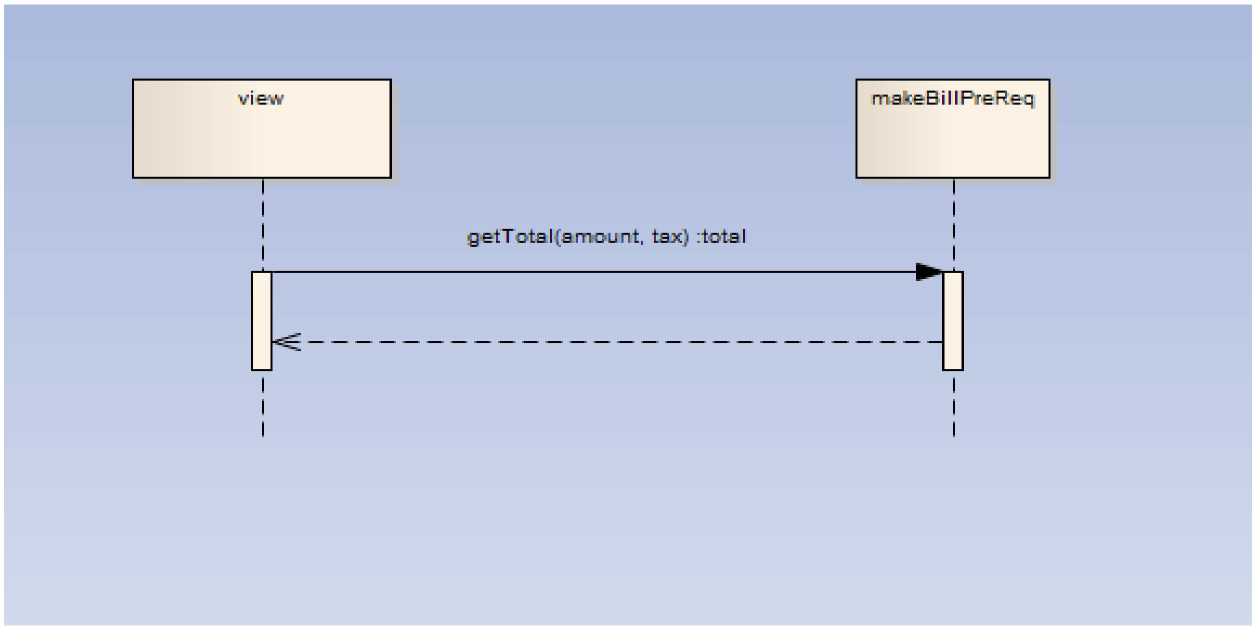
10.9 Genereate_Bill_4



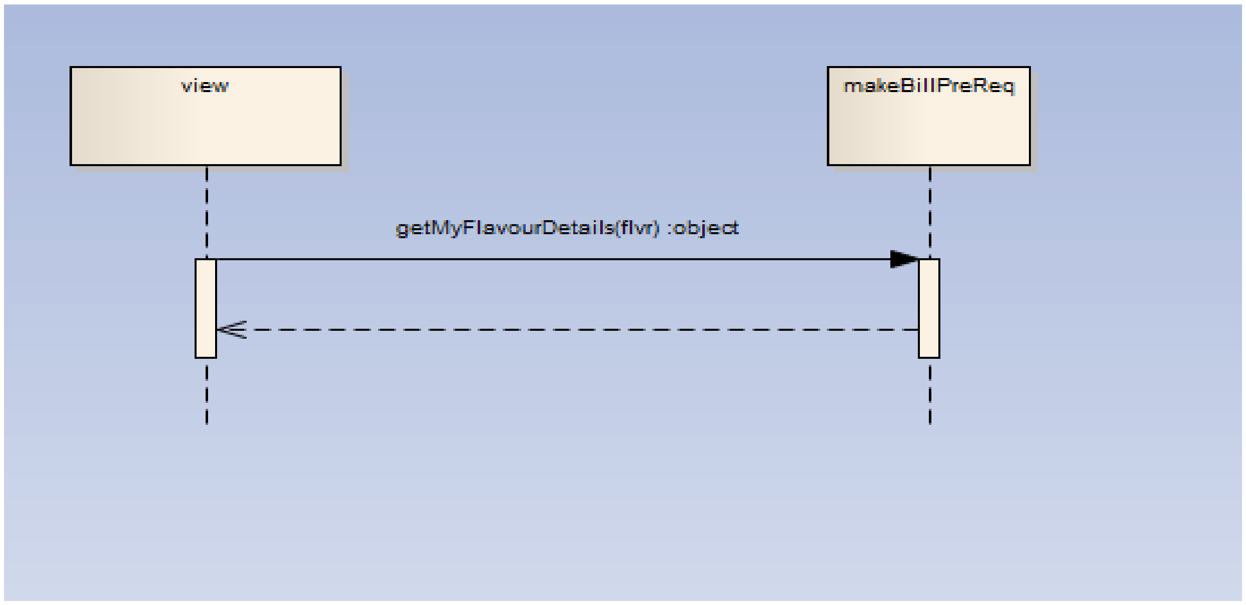
10.10 Generate_Bill_5



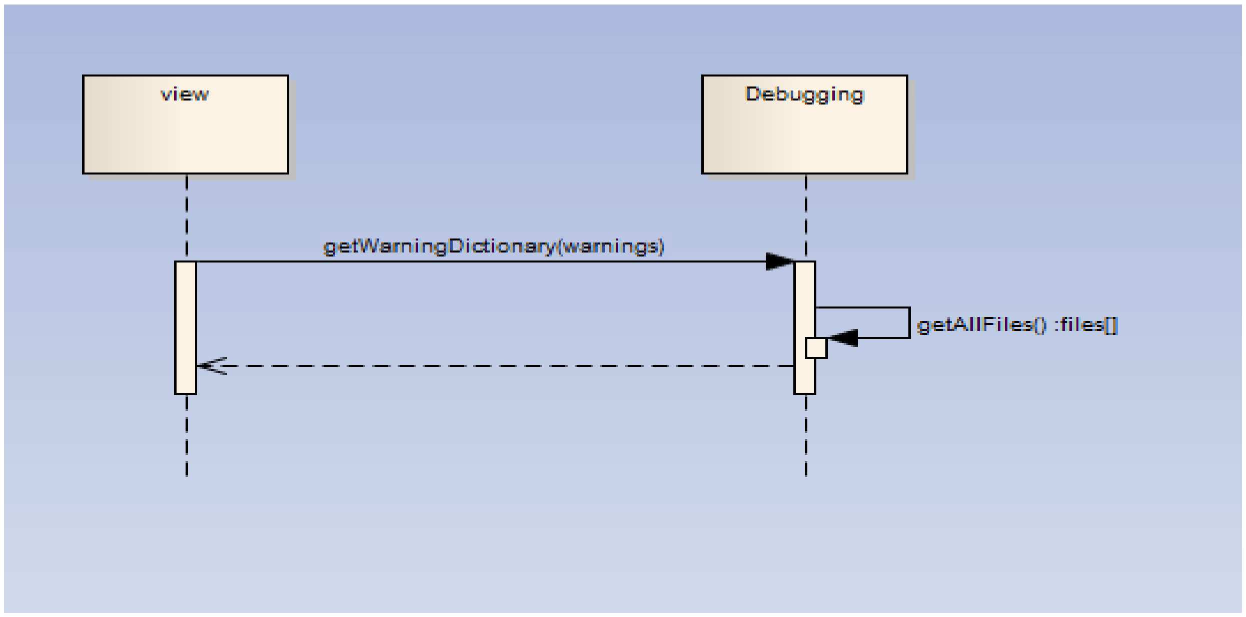
10.11 Generate_Bill_6



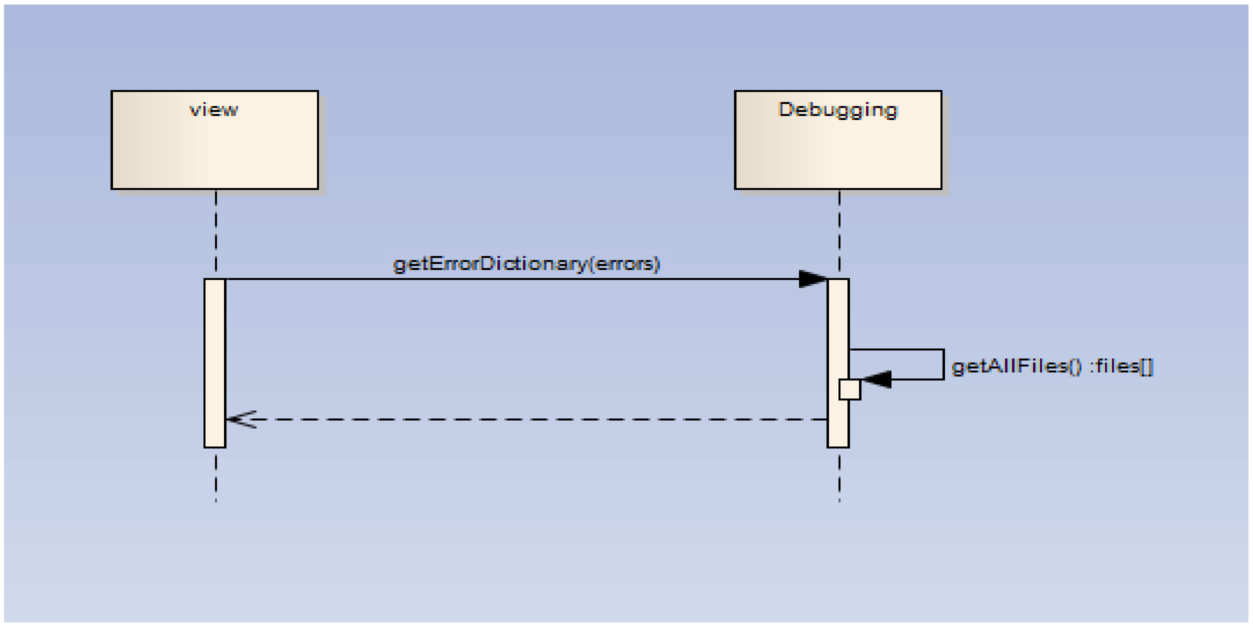
10.12 GENERATE_BILL_7



10.13 VM_1



10.14 VM_2


view
Debugging
getErrorDictionary(errors)
getAllFiles() :files[]

11 OPERATION CONTRACTS

11.1 CREATE ALARM:

Name	createAlarm(args[])
Cross Reference	Create Alarm
Pre-condition	At least 1 instance is available.
Post condition	• Alarm Generate Object Created • No association is formed/destroyed • No deletion occurred.

11.2 CREATE BILL CATEGORY

Name	createBillCategory(billName,flavor,price)
Cross Reference	Create Bill Category
Pre-condition	
Post condition	• Bill, MongoDB Object created • Association with MongoDB is formed. • No deletion occurred.

11.3 SAVE BILL CATEGORY

Name	SaveBillCategory()
Cross Reference	Create Bill Category
Pre-condition	-
Post condition	• Bill, MongoDB object created. • MongoDB and billCategory row is updated. • Deletion of MongoDB object occurred.

11.4 Generate Bill

Name	generateBill(catg,instance)
Cross Reference	Generate Bill
Pre-condition	Instance is available.
Post condition	• MongoDB object is created. • No association formed • No deletion occurred.

Name	generateBillPlease()
Cross Reference	Generate Bill
Pre-condition	-
Post condition	• GenerateBill instance is created. • Association of instance with GenerateBill is formed. • No deletion occurred.

Name	getFalvorName()
	getTaxAmount()
	getTotalAmount()
	getInstanceRunningTime()

Cross Reference	Generate Bill
Pre-condition	-
Post condition	-

Name	getCategoryPrice()
Cross Reference	Generate Bill
Pre-condition	Category already available in Database.
Post condition	• BillPreReq, MongoDB instance is created. • Association with MongoDB is formed. • Nothing is deleted.

11.5 Monitor VM statistics:

Name	getWarningDictionary(warnings)
Cross Reference	Monitor VM status
Pre-condition	Log Files should present.
Post condition	• Debugging instance created • No association formed • No deletion occurred.

Name	getErrorDictionary(warnings)
Cross Reference	Monitor VM status
Pre-condition	Log Files should present.
Post condition	• Debugging instance created • No association formed • No deletion occurred.

11.6 MONITOR CLOUD STATUS:

Name	getTenantNames()
Cross Reference	Monitor Cloud status
Pre-condition	Tenant already presents
Post condition	• No instance created. • No association formed • No deletion occurred.

Name	getAllResources(tenant)
Cross Reference	Monitor Cloud status
Pre-condition	-
Post condition	• ResoiurceQuotas instance created • No association formed • No deletion occurred.

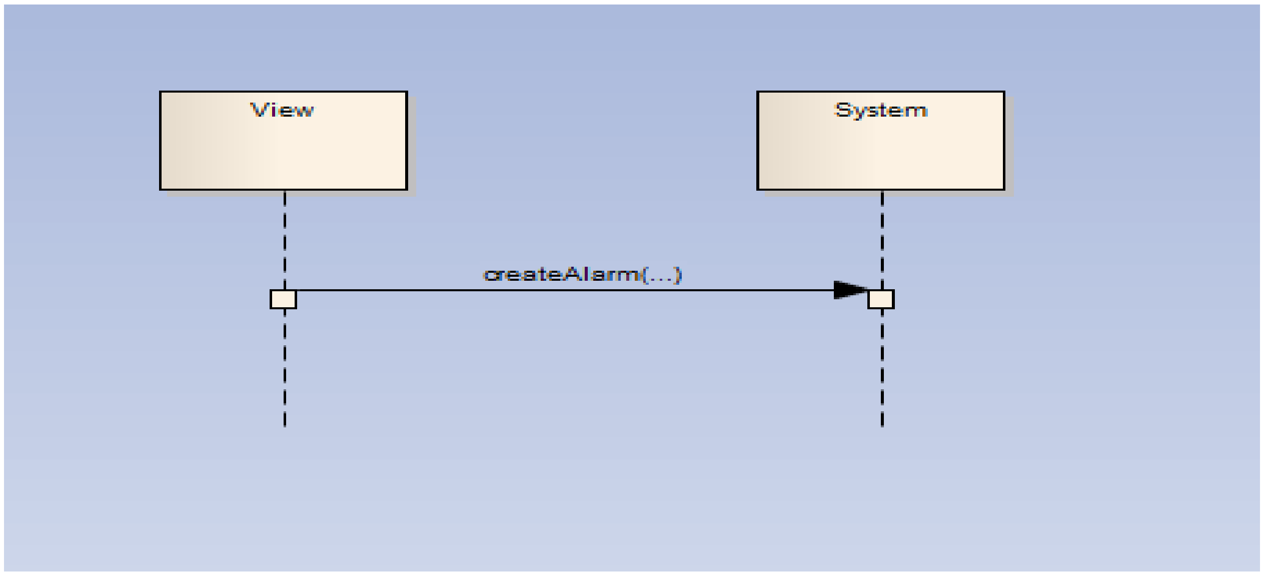
Name	getKeyPairsToTenant()
Cross Reference	Monitor Cloud status
Pre-condition	-
Post condition	• No instance created. • No association formed. • No deletion occurred.

Name	getAllFlaotingIps()
Cross Reference	Monitor Cloud status
Pre-condition	-
Post condition	• No instance created

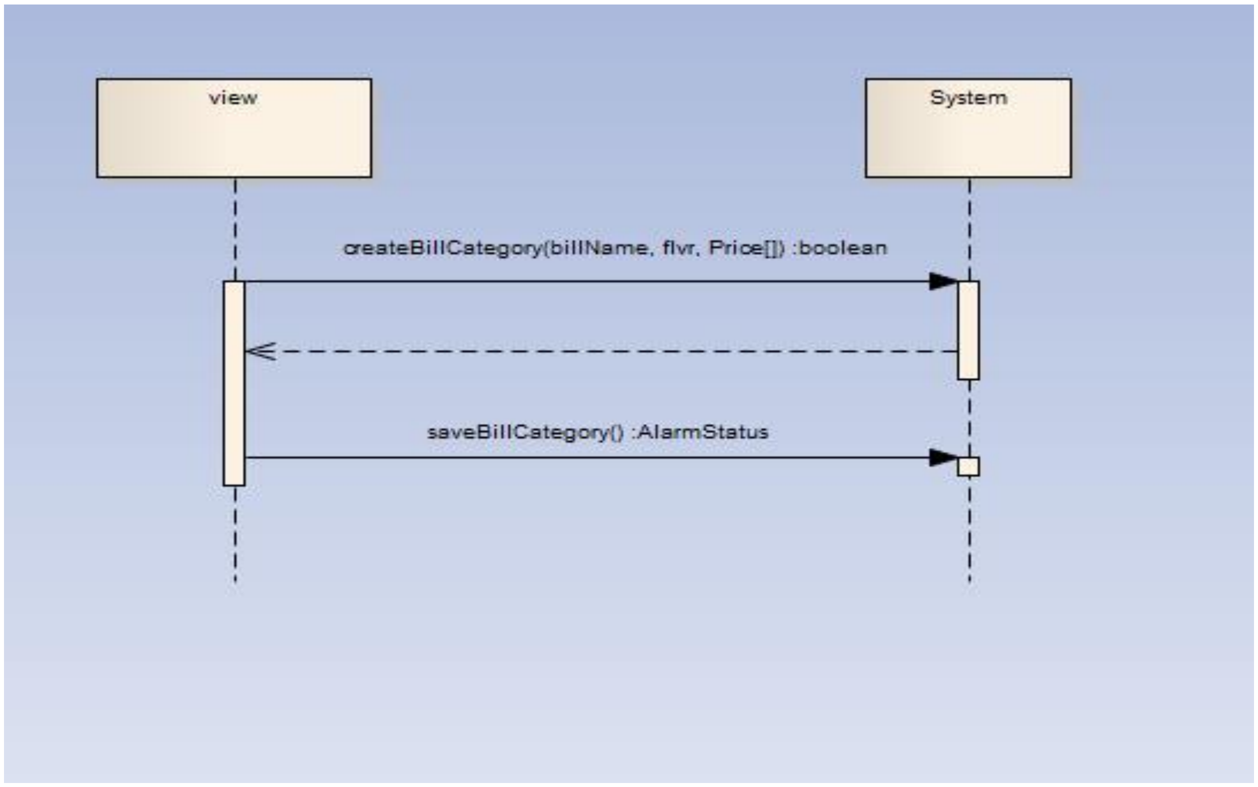
	• No association formed • No deletion occurred.

12 SEQUENCE DIAGRAMS

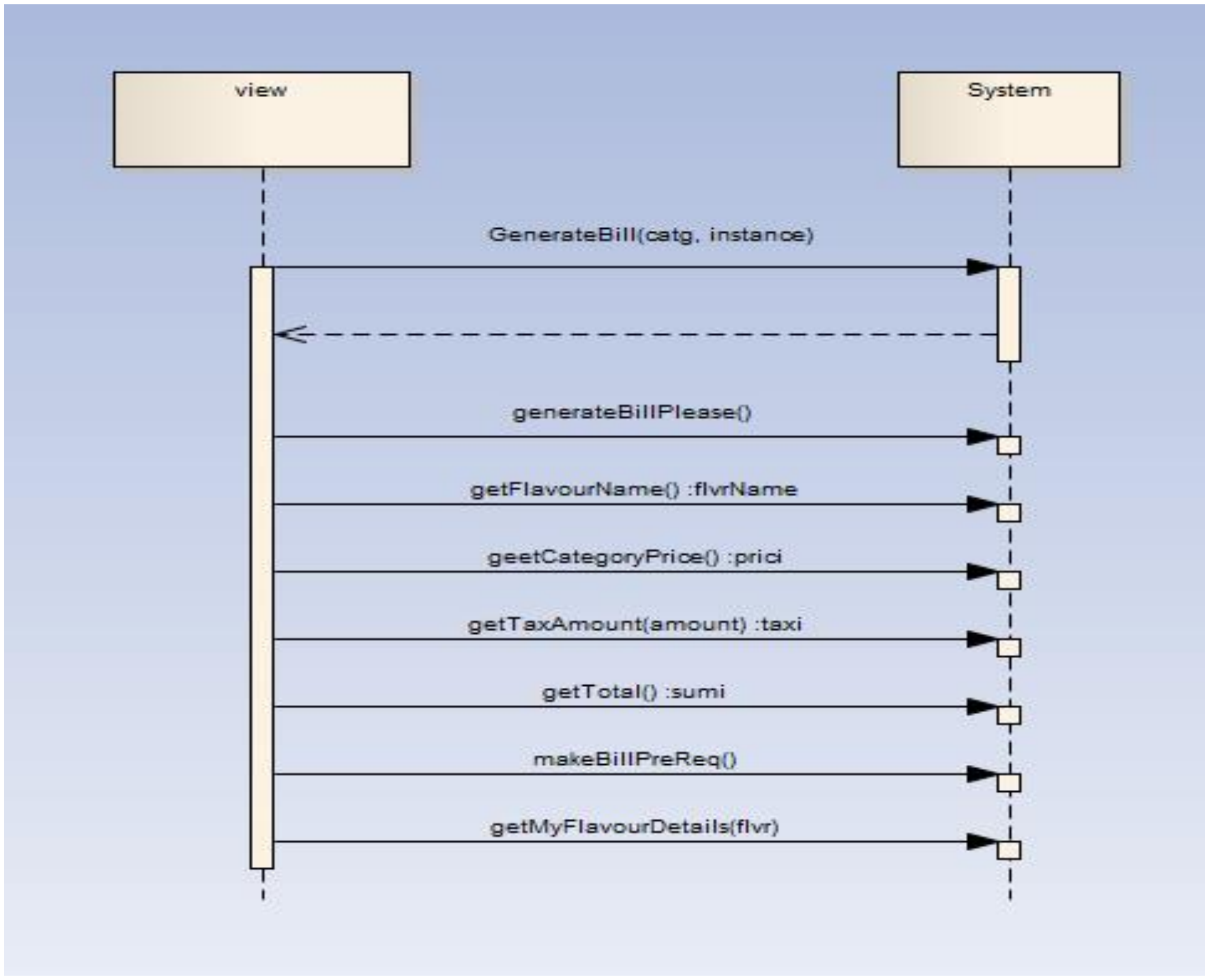
12.1 CREATE_ALARM



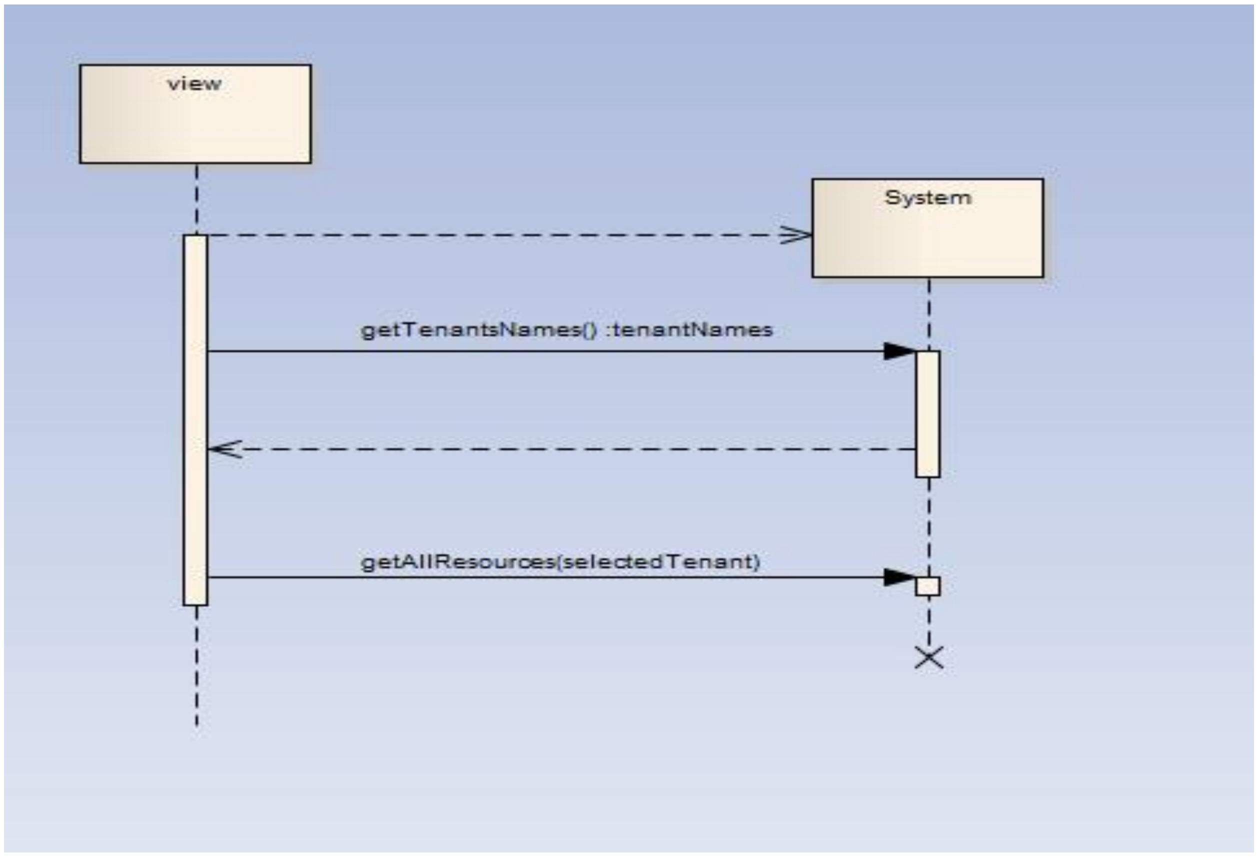
12.2 CREATE_BILL_CATEGORY


view
System
createBillCategory(billName, flvr, Price[]) :boolean
saveBillCategory() :AlarmStatus

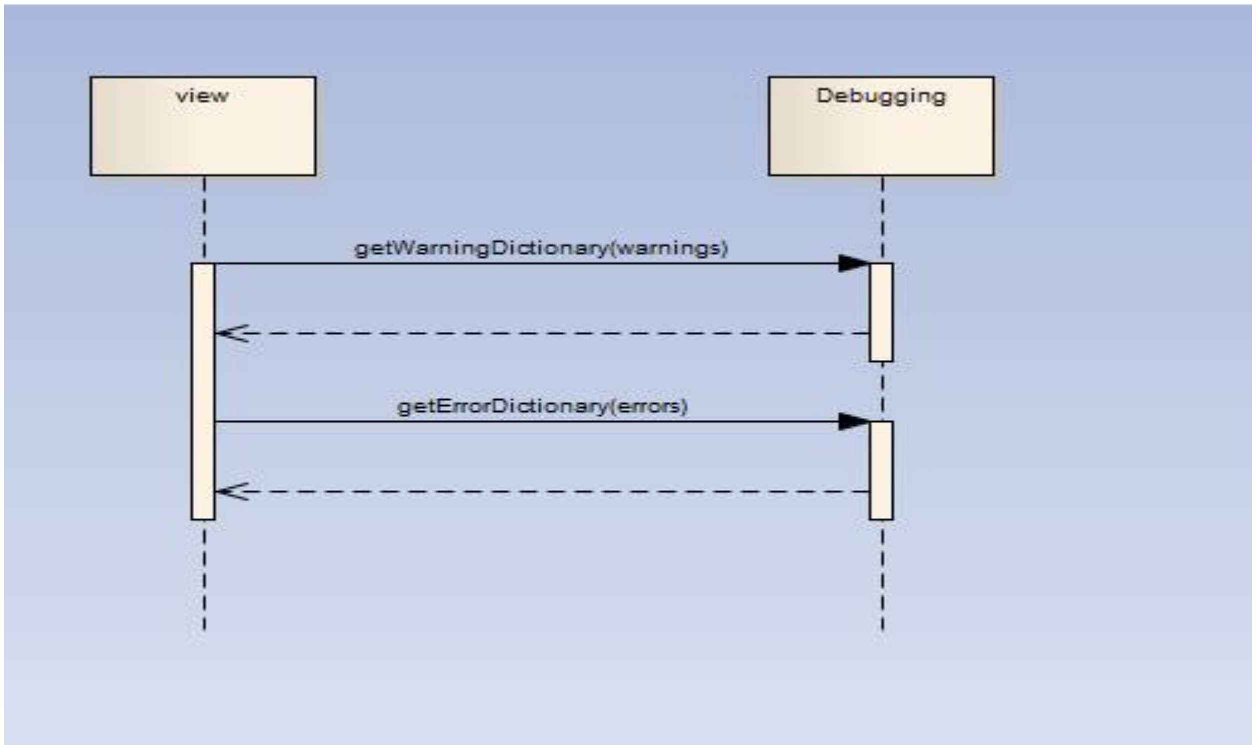
12.3 GENERATE_BILL


view
System
GenerateBill(catg, instance)
generateBillPlease()
getFlavourName() :flvrName
geetCategoryPrice() :prici
getTaxAmount(amount) :taxi
getTotal() :sumi
makeBillPreReq()
getMyFlavourDetails(flvr)

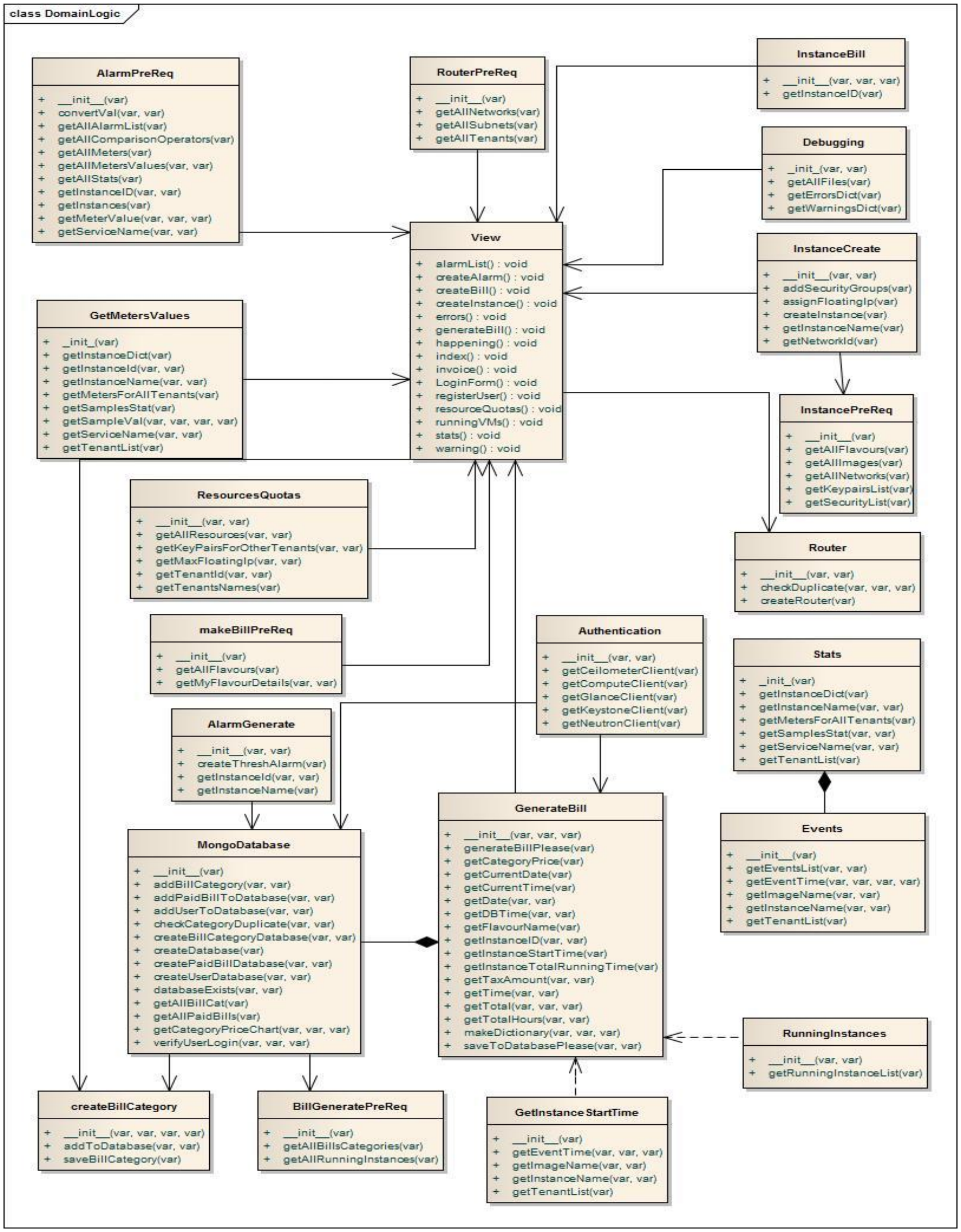
12.4 MONITOR_CLOUD_STATUS


view
System
getTenantsNames() :tenantNames
getAllResources(selectedTenant)

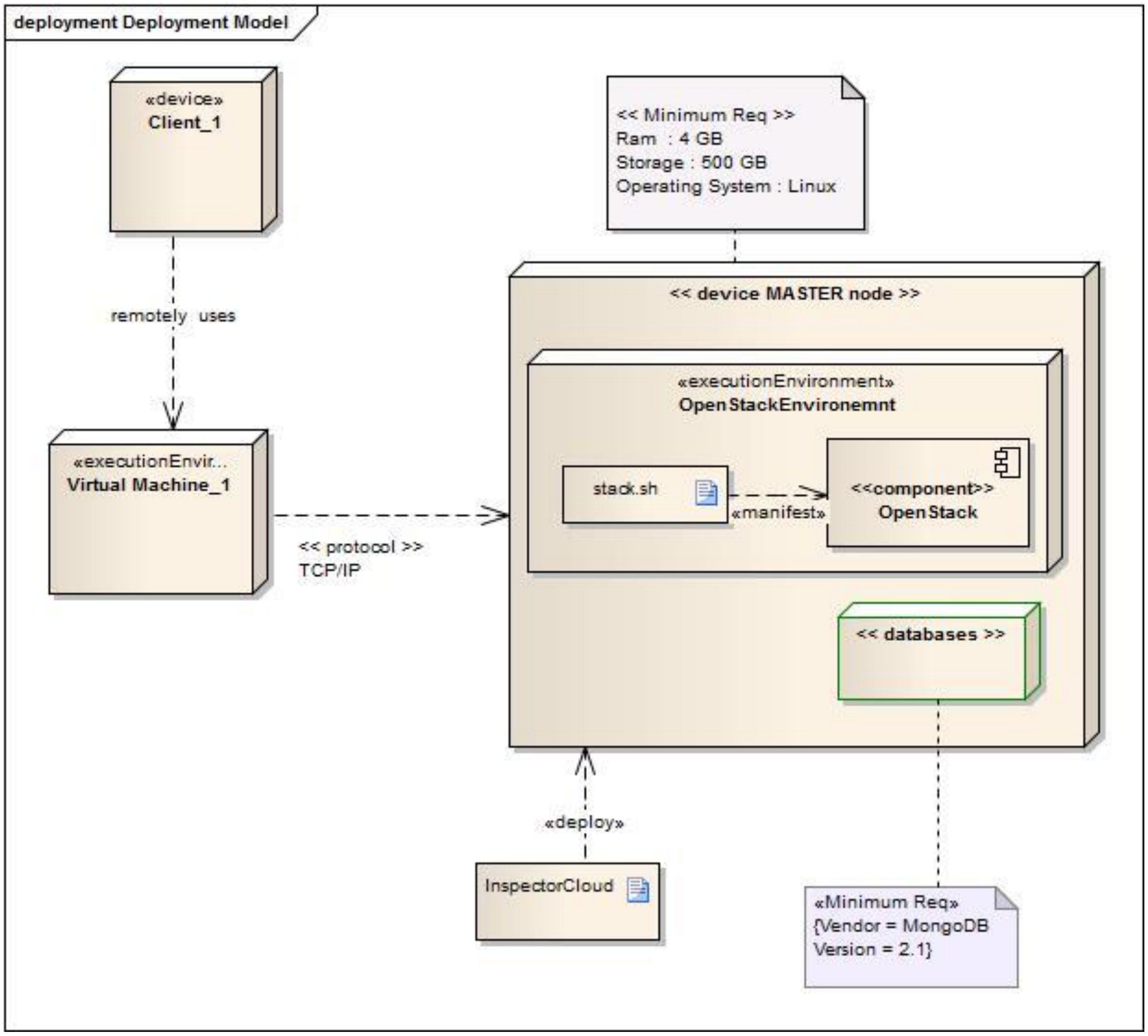
12.5 MONITOR_VM_STATUS



13 CLASS DIAGRAM



14 PACKAGE AND DEPLOYMENT DIAGRAM


deployment Deployment Model
«device»
Client_1
remotely uses
«executionEnvir...
Virtual Machine_1
<< protocol >>
TCP/IP
<< Minimum Req >>
Ram : 4 GB
Storage : 500 GB
Operating System : Linux
<< device MASTER node >>
«executionEnvironment»
OpenStackEnvironemnt
stack.sh
«manifest»
<<component>>
OpenStack
<< databases >>
«deploy»
InspectorCloud
«Minimum Req»
{Vendor = MongoDB
Version = 2.1}