

Spring 2014

Cmpe-283 Virtualization Technologies

**Project-1 Report**

**Submitted by:**

Cheng-Ju Chuang <jeremhh@gmail.com>

1. Introduction

The report indicates the result of Cmpe-283 project 1. It contains the basic architecture design of programming and provides the resource and reference I need to this project.  
Goal:

* To gain experience of capabilities which provided by VI API.
* Trying resolve real-industry problem.
* To facilitate interoperability between cluster and program.

Needs:

* Established cluster and hosts which have vCenter, vHosts and VMs.
* Java programming environment.
* Authority to access cluster vLan.

1. Background

This project basing on java can monitor and managing the VMs under specific cluster system. When VM failed to provide computing resources, the program will recover the failure immediately. Basically, this project is designed to handle multiple VMs and make sure they can always provide the computing resources and improve cluster availability and stability.

1. Requirements

These requirements are essential features of project. In functional features, we can see the operations provide functionality for user to handle and manage their VM under cluster. According to functional requirements, we can scale the size of project and match what functions that availability manager should provide.

Non-functional features indicate advantages which used to approaching project goal from the project. Most of them claim how the use case and scenarios working out are. They also give the rule of designing the architecture.

Functional Requirements:

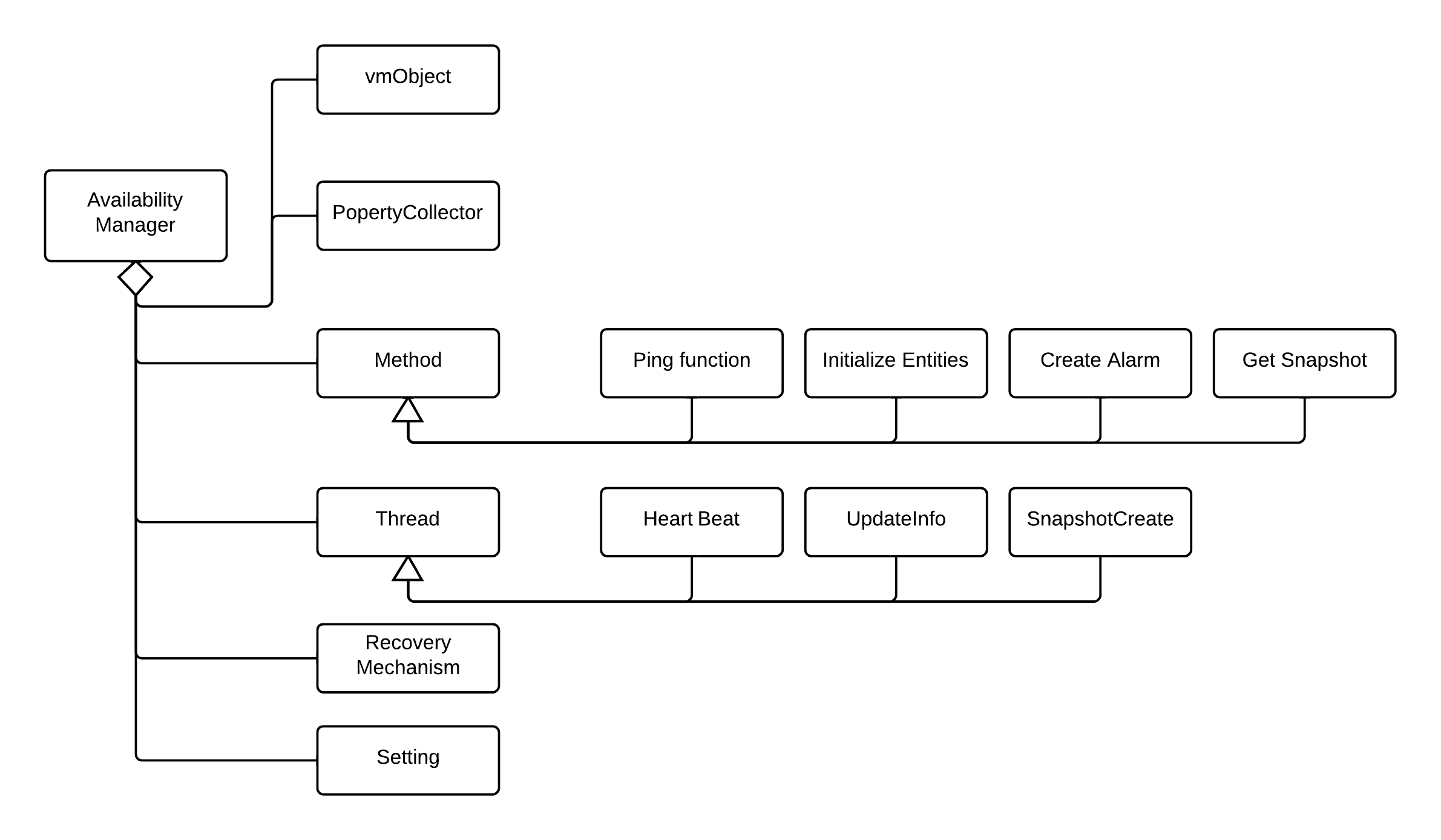
* Monitoring
* Recovery
* Adding Hosts
* Remove Hosts
* Revert to Snapshot
* Cold Migration
* Clone VM

Non-Functional Requirements:

* Availability
* Stability
* Controllability

1. Design

Following diagrams describe architecture and main work flow of this project. Main components are vmObject, Methods, Thread and Setting. vmObject is a class which contain necessary information of instance (like ServiceInstance, VirtualMachine…etc.) which can be easily access and refer. Methods class is a function pool which provides important function to implement essential feature such as ping function, create alarm and get snapshot tree. Thread could be seen as execution of light weight process which is handled independently. Basically, main programing calls thread to monitor instances. If any failure occur and reduce of resource accessibility, main program will invoke recovery mechanism to restore the availability of cluster. Setting stores some static strings and information like URL, account and password. It provides easy scaling if we change to another cluster system.

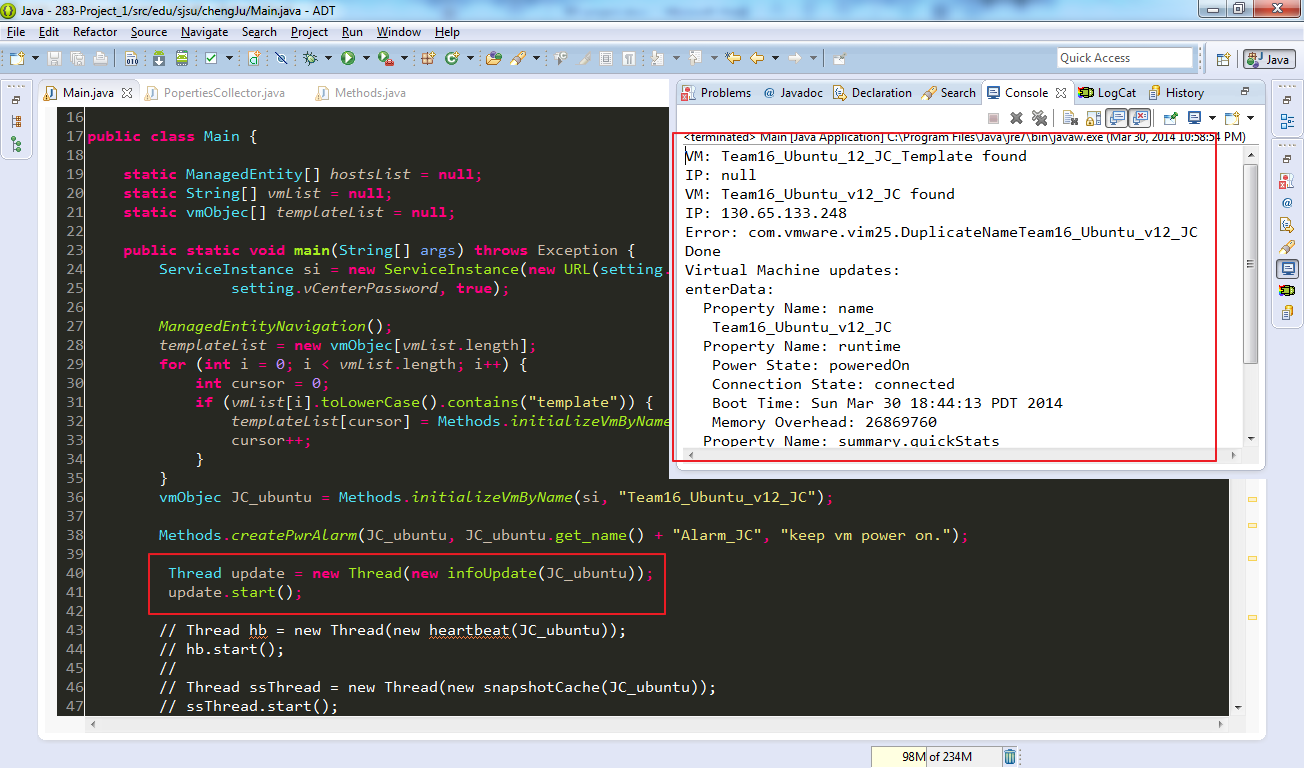


*Fig1. Component Tree*

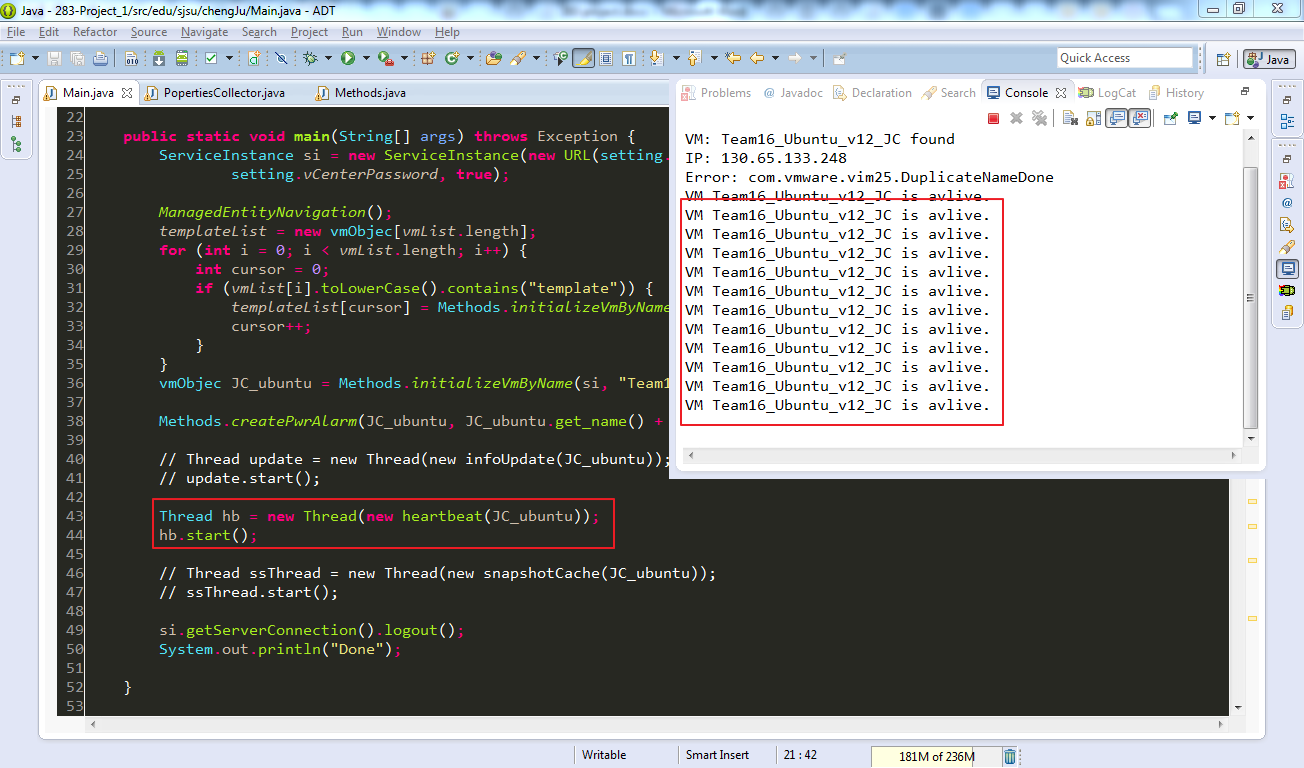
1. Implementation

We use VI Java SDK this time so I set up Java environment with eclipse IDE and no plug-in usage. Development OS is Windows 7 but program could run under OSX/UNIX system with sound Java environment. The program should be executed under LAB room eng-276 with team16 cluster system.

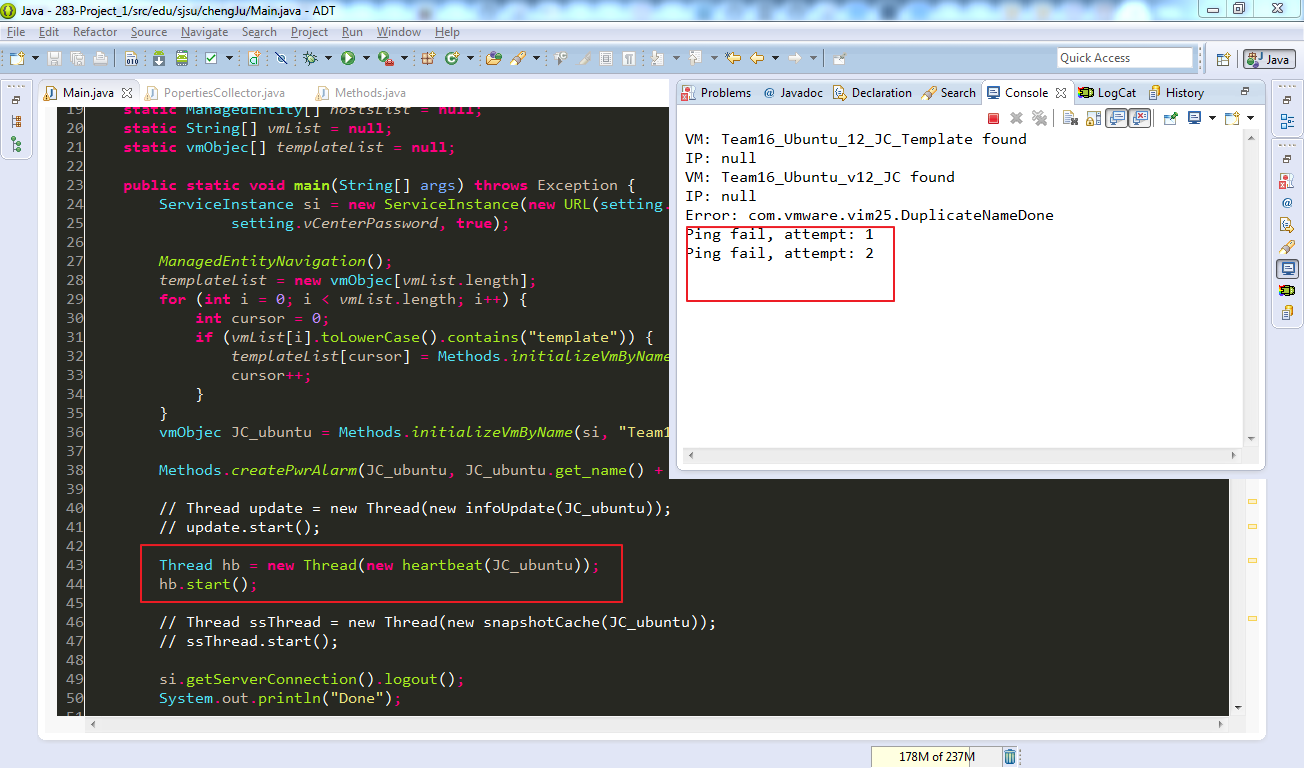
Next part contains a sequence of screenshot to show the capability of this project.



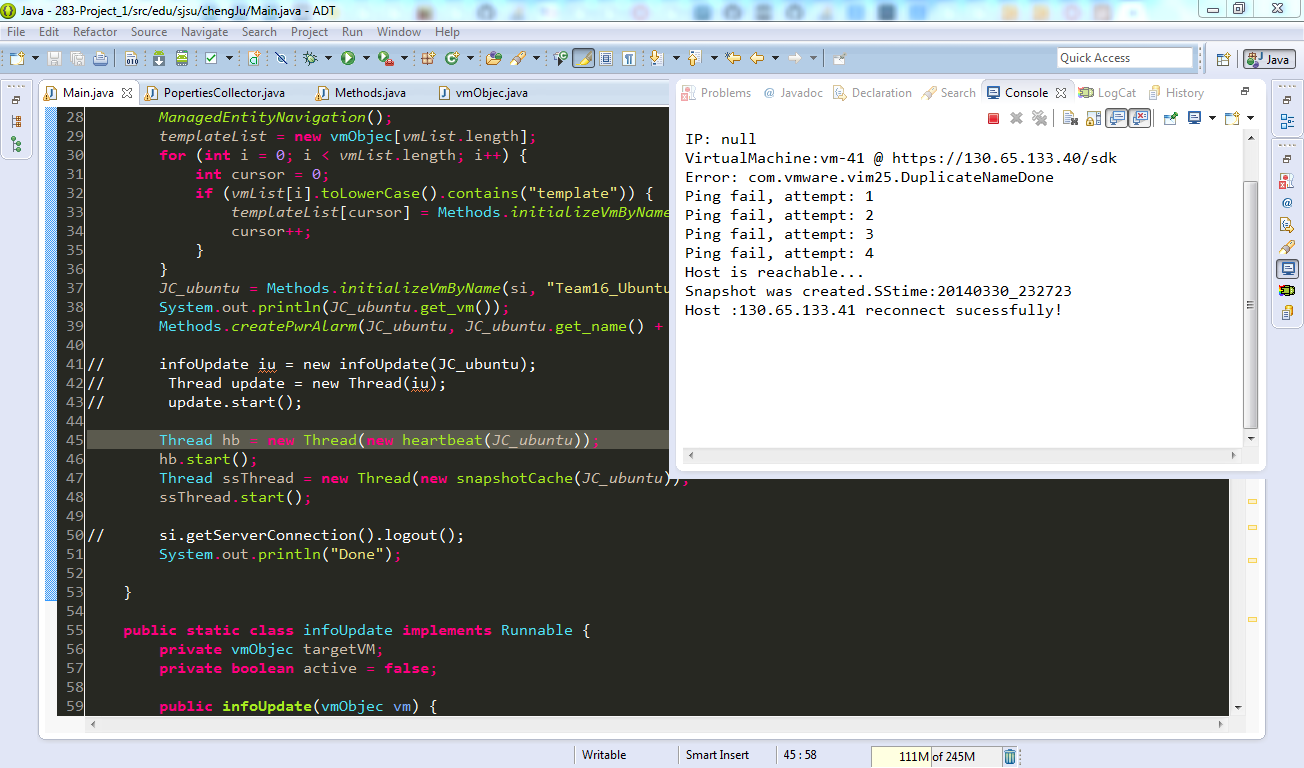
Monitoring the target VM.



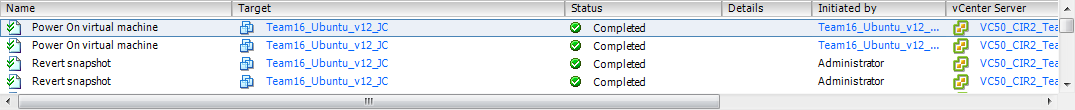
Tracking Heartbeat



Ping fail when VM interface down.



Reconnect the vHost.



Make VM’s power on after reverting to current snapshot.

*Fig2. Sequence of Screenshot*

1. Questions

* **In case of failure, what is a good approach during Disaster Management of Virtual Machines-**
  + **Check the Host first, then the Virtual Machine.**
  + **Check the Virtual Machine, then the Host.**

**Justify your answer with sufficient reasons.**

My answer is to check the host first, then virtual machine. Main reason is when VM fails; we don’t know where the problem is. In hierarchy structure of cluster, VMs are mounted under vHosts. In other words, VM always fails when host is down. The case will be more complicated if host failed then so we should check the availability of host first to prevent to waste too much time on necessary node checking. Better case is reconnecting vHost if vHost disconnected or reverting VM from current snapshot when vHost is alive. Those two cases use less resource and step to keep the cluster works fine. Since we check VM first, we still have to check vHost’s status. Reducing the potential causes in every steps is more efficient than we thought.

* **How many threads do you think are required for the optimal functioning of your Availability Manager? Support your answer with reasons.**

My answer is 2: Heartbeat thread and Snapshot Create thread. Basically, availability manager detect entities status by tracking heartbeat so Heartbeat thread is absolutely needed. Then we can follow up if failure occurs. Snapshot provides another solution for back up and reverting VM.

In order to tack entity continuously, Heartbeat and Snapshot are executed regularly until exception appears. That indicates that when function being executed repeatedly, the risk will increase because of dead-lock. On top of that, I try to decrease same time running threads to prevent dead-lock happening. For monitoring and management, heartbeat and create snapshot is fair enough for my case. If there’s other requirement which needs to create new thread, I tend to use one time thread and finish it immediately.

* **How did you set and use the Alarms to aid your Availability Manager? Where else can you use the alarms? Explain**

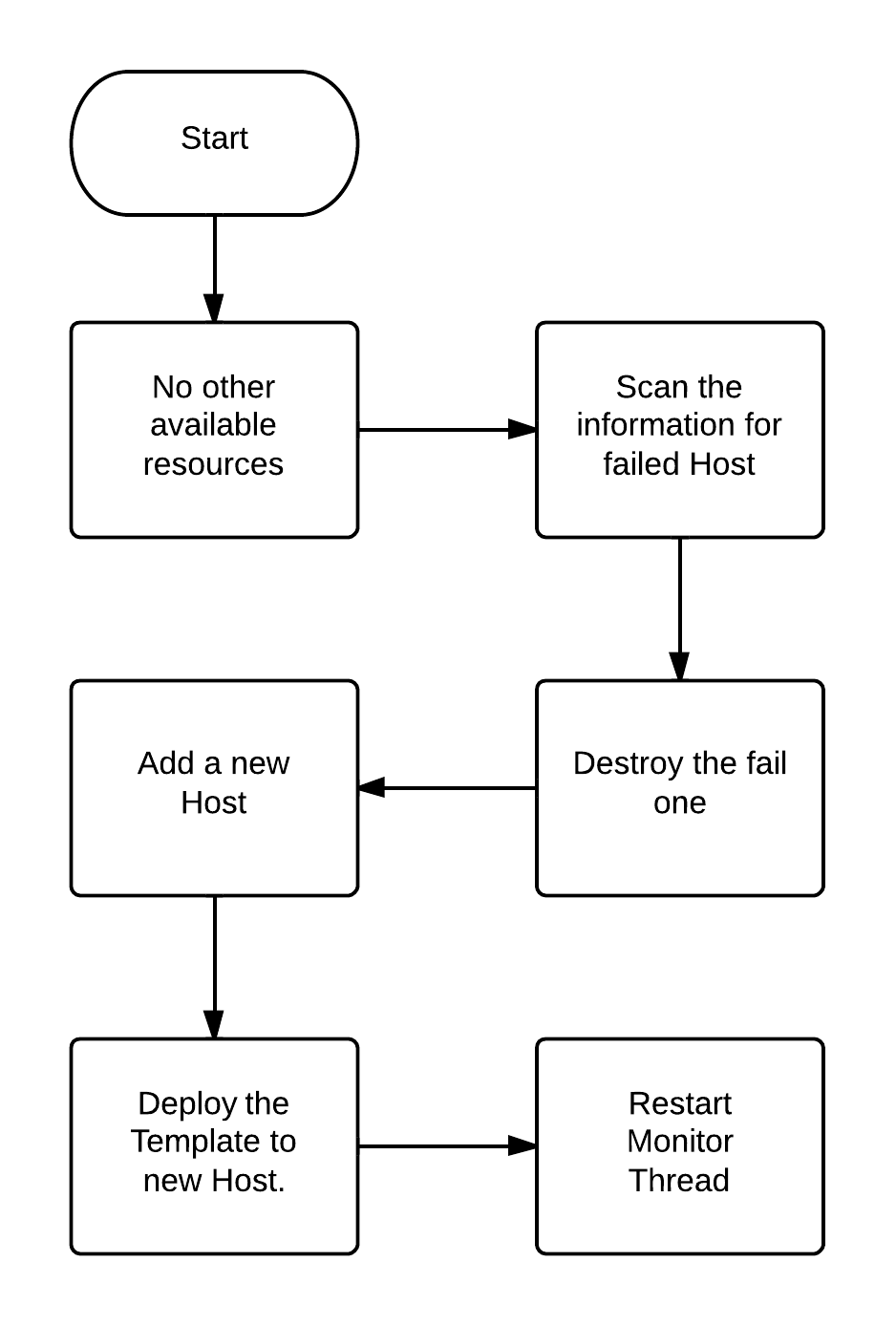
In my case, I set an alarm which is triggered by power state. If power is off, it will create an action task to set VM power on. It prevents heartbeat give a wrong decision due to legally power off / reset the VM by user. In assign duration, heartbeat function will ignore the failure of time slice because of restarting VM.

Alarm also can be used on entity property monitoring such as CPU and memory. For example, VMWare has a default alarm called CPU alarm. If CPU usage exceeds 90 percent, alarm will set VM as red state and we can see the warning icon in vSphere client. Alarm is a good tool to avoid simple failure happens and recovers machine right after that. But if there is a complicated scenario involving different condition, alarm might not be our first choice.

1. Discussion

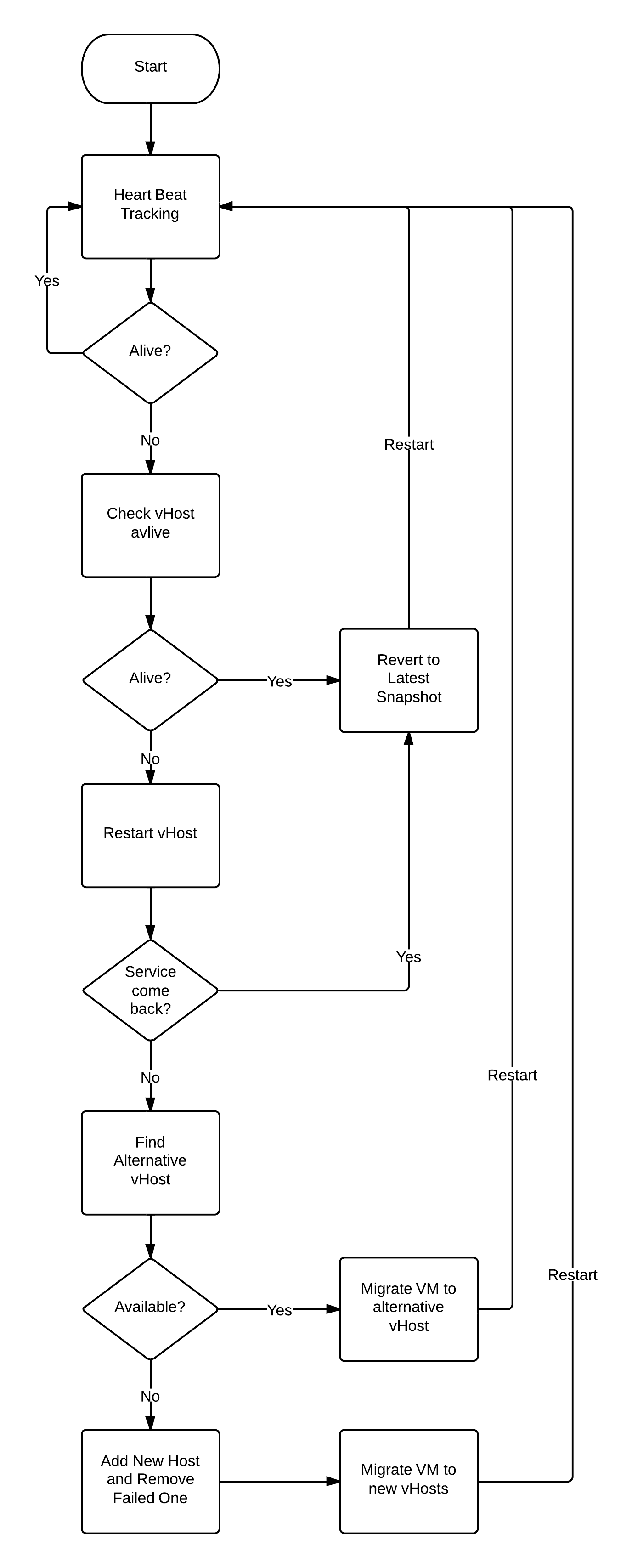
* The host add/remove mechanism

In the figure 2, we can see the flowchart of add/remove mechanism. As I mentioned before, I tend to reduce potential factor step by step to reduce the complexity of failover procedure. To complete the cycle of availability manager mechanism, the program will start the thread again to keep monitoring.



*Fig3. Flowchart of add/remove mechanism*

* The approach used to configure the failure detection for each VM

*Fig4. Flowchart of Failure Detection*

This long flowchart explains the main approaching of my failure detection. Even it is not the perfect mechanism due to time limitation of this project; hopefully it should solve most failure case in cluster system.

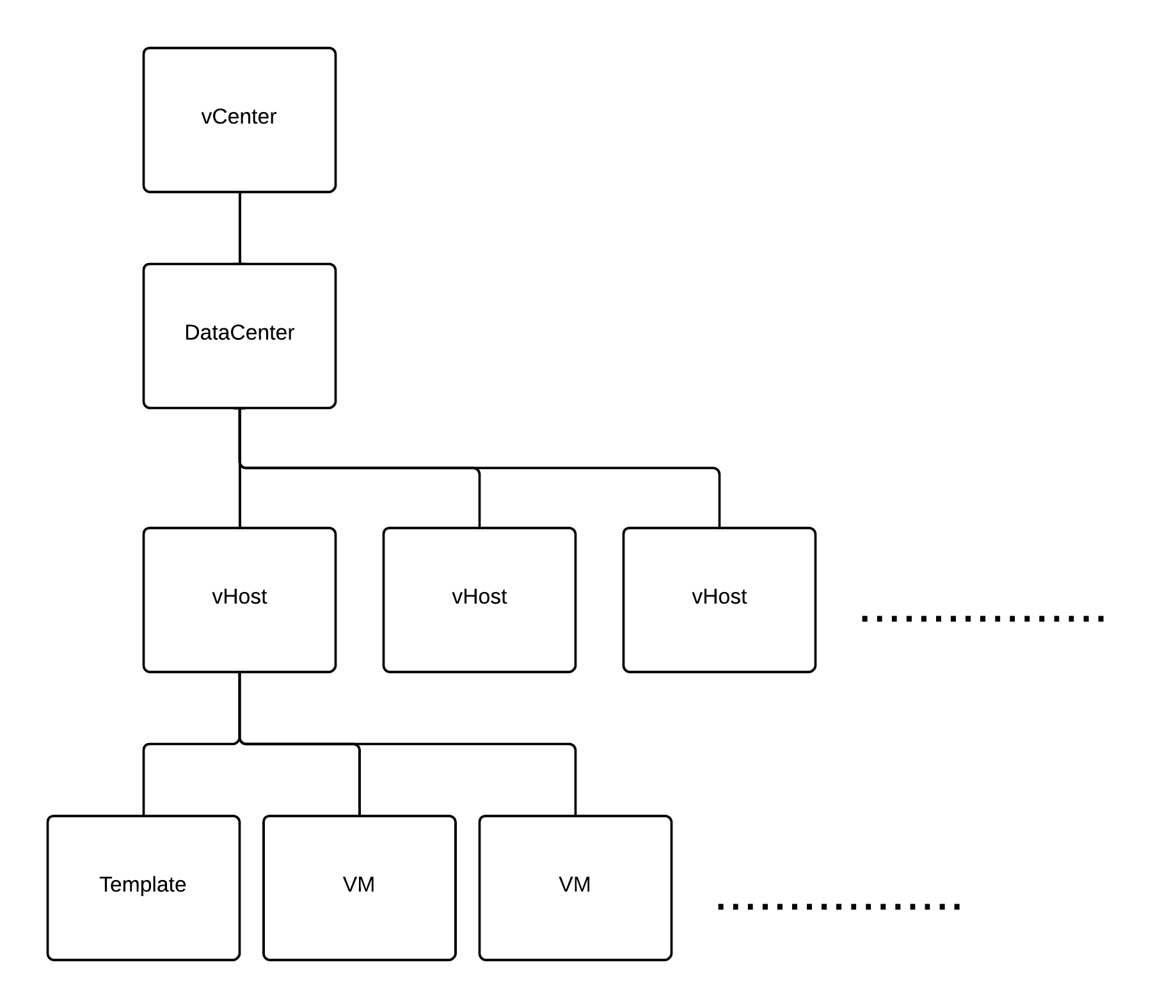
Generally, it’s a recycle Heartbeat process. If any failure happens, the process will stop and clarify the reason and apply with fine solution for it. It won’t stop for any natural condition until administrator force to shut down the program console.

* How host failures were detected

Basically, Heartbeat thread will check the ping state of each virtual machine to detect failure. I choose to put more fault-tolerance to prevent bad network condition or factor which cannot be handled easily. For instance, in this project it will be judge as failure if thread cannot receive normal termination of ping function five times in row per 5 seconds. This mechanism decreases the chance of wrong cases taking off programing resources.

* The mechanism used to convert between the image formats used by the hypervisors

There are some options for deploy, migrate and clone VM instance like OVF, Snapshot, VMDK and VM template. Due to time limitation I decide to choose snapshot and template file as my back up solution. Snapshot file can be cloned or revert to virtual machine if vHost is alive so I put it as my first priority to solve back up problem. Then I assign the origin VM template in each vHost as my plan B. If the vHost is not available anymore, we cannot access to VM resource because of connect exception. In that time we can deploy the template in every vHost to keep VM availability. As the figure below, the hierarchy of cluster meets the requirement for handling. Also we can extend this project to supply other VMware file resources.



*Fig5. Hierarchy of Cluster*

1. Conclusion

Challenge:

The hardest part in this project is VI SDK look up and threads resource allocation design. VI SDK is a pretty powerful development tool but it is also complicated, too. Deep hierarchy and huge variety classes increase the difficulty to use right API in right way. Official document and VI SDK forum help me a lot to call right API which does not waste resource. Threading is another issue here even I have designed threading program before. The potential risk of thread is dead-lock that I mentioned above so we have to clarify the logic as much as we can to avoid any harmful or inefficient situation. Flowchart helps is a good way to find out explicit steps to handle different scenario. Generally speaking, this project is worthy because it makes students start to resolve the problem in real world.

Lesson learned:

* VI SDK structure and overview.
* Experience of management cluster through vSphere client.
* Multi-Thread design.
* Researched in VMWare resource collection.

1. References

VI Java API forum: <http://sourceforge.net/p/vijava/discussion/826592/>

VMdev.info: <http://www.vmdev.info/>

Double Cloud: <http://www.doublecloud.org/>

VMware vSphere 5.1 official document:   
<http://pubs.vmware.com/vsphere-51/index.jsp>