CCRG Cluster Manager

AOOP Term Project Report

Project Members: İlke Muhtaroğlu

Nitel Muhtaroğlu

Melih Koca

**Hyperic**

**Hyperic is a Web Application Monitoring and Management software (**<http://www.hyperic.com/>**). It watches the agents (the servers) for CPU, Memory and other resources. It also provide application level metrics for known applications such as Tomcat, MySQL, Hadoop. It is open for writing for third party plugins (for weka) for watching the application status.**

**At our project we aimed getting the server agent status for CPU, Memory in order to evaluate in which server it is effective to run/schedule the job. In order to do the scheduling decision we need the CPU, Memory information for evaluating in the scheduling algorithm, this information is gathered via agents deployed on servers and transfered to the Hyperic server via web services. Again these agents information is queried from the Hyperic server via a web service API. This api is the key for getting information from the hyperic server and its agents.**

**The API is provided at the Jar : *hqapi1-4.1.0.M1.jar***

**This jar depends on additional jars that resides at HQapi user library.**

**tr.edu.ozyegin.ccrg.clustermanager.cluster**

*HypericStateQuery -* is the singletoned class that is responsible for communicating to the Hyperic server and querying the agent’s cpu, memory and other metric statuses. The class has the singleton pattern applied.

The *agentApi()* method of the class returns the List of the Agent IP addresses.

The *memoryStatus(String)* method takes the Agent IP as the parameter and returns the last 100000 milisecond statistic of the agents memory in Byte.

**Junit Tests**

**We have used Junit for testing the written code for integration purposes. For example the connection to the Hyperic is higly error prone due to integration to a unknown third party sytem. Therefore starting by ClusterTest Junit class, the public methods are tested whether they provide the promissed functionality.**

**At the Junit class the annotations drive the tasks done in methods. For example via @Before annotation, the method is annotated to make preparation for test methods. For example the classes, necessary for test methods, can be generated at the method annotated via @Before.**

@Before

**public** **void** setUp() **throws** Exception {

}

**Then at the methods annotated via @Test, the aimed tests can be written. For example at the method “testCluster” annotated via @Test, the Cluster class tested for creation (Cluster class is singletoned!), the testing way is totaly semantic. For example at this method it is checked whether cluster object is null or not ... The other methods are tested for the purposes they are expected to serve.**

@Test

**public** **void** testCluster() {

Cluster cluster = Cluster.*getSingletonCluster*();

*assertTrue*(cluster != **null** );

cluster = **null**;

//fail("Not yet implemented");

}

@After

**public** **void** tearDown() **throws** Exception {

Cluster cluster = **null**;

}

There is also a class containing “ public static void main ( String [ ] args ) ” for creating the pattern objects and classic testing : System.out.println(“--result” + result);

Test related codes are under package:

tr.edu.ozyegin.ccrg.clustermanager.test

the classes are:

ClusterStatusTest – Testing the ClusterStatus and State Pattern objects.

ClusterTest – Testing the Observable pattern applied to Cluster class. Via this test class Cluster, Node, Component, Memory, CPU, NIC, Storage classeses integrity is tested.

MainMethod – Testing the classes via creation through Class \_class = new Class(); and then printing the result to the console via System.out.println();

* Junit is introduced byKent Beck at extreme programming TDD technique. There is also Mock Object technique which is used for creating mock objects that simulate the expected/necessary object that is required in a class to satisfy the expected behaviour.

Patterns

tr.edu.ozyegin.ccrg.clustermanager.cluster

At this package the UML diagram depicted at slide page number 5 is implemented. There is only one domain model entity called “State.java” at this package. Please check the project source code jar delivered for classes that are implemented.

tr.edu.ozyegin.ccrg.clustermanager.states

At this package the UML diagram depicted at slide page number 8 is implemented.

tr.edu.ozyegin.ccrg.clustermanager

At this package the UML diagrams depicted at slide page number 7, 9, 10, 12, 13 are implemented.

Slide page number 7 is implemented at Shell.java, ClusterInterface.java

Slide page number 9 is implemented at Job.java, JobCreator.java, HadoopJob.java, WekaJob.java, CcxJob.java, HadoopJobCreator.java, WekaJobCreator.java, CcxJobCreator.java

Slide page number 10,13 are implemented at Scheduler.java, HadoopScheduler.java, WekaScheduler.java, CcxScheduler.java

Slide page number 12 is implemented at ClusterInterface.java, ClusterInterface.java