Appendix

OCSO dynamic threshold triggering algorithm

```
1   package Green;
2
3   import java.io.IOException;
4   import java.util.Calendar;
5   import java.util.LinkedList;
6   import java.util.Timer;
7   import java.util.TimerTask;
8
9   import org.w3c.dom.Element;
10
11  import Green.OCSOOptimizationFacilitor.UtilizationRegulator;
12  import Green.OCSOOptimizationExecutor.ServiceOptimizer.IaaSOptimization;
13  import Green.OCSOOptimizationExecutor.ServiceOptimizer.PaaSOptimization;
14  import Green.OCSOOptimizationExecutor.RuleEvolution;
15  import OCSOOperation.FileOperation;
16  import OCSOOperation.RuleParser;
17
18  public class ScalingOptimization {
19
20      private LinkedList<Element> optimizationRules = null;
21      private LinkedList<Integer> optimizationFrequencies = null;
22      private LinkedList<Integer[]> optimizationCounters = null;
23      private static LinkedList<Integer[]> finalOptimizationThresholds = null;
24      private static LinkedList<Integer> finalOptimizationUpLimits = null;
25      private static LinkedList<Integer> finalOptimizationDownLimits = null;
26      private LinkedList<String> optimizationIDs = null;
27      private LinkedList<String> optimizationTypes = null;
28      private LinkedList<String> optimizationServices = null;
29      private LinkedList<String> applications = null;
30      private LinkedList<String> instances = null;
31      private LinkedList<Timer> optimizationTimers = null;
32      private LinkedList<Schedule> optimizationSchedules = null;
33      private static String configFilePath = "src/configuration.txt";
34      private static String ruleFilePath = "src/rules.xml";
35      private static String iaaSLogFilePath = "src/GreenIaaS/log.txt";
36      private static String paaSLogFilePath = "src/GreenPaaS/log.txt";
37      private boolean[] onGoingOptimization = new boolean[2];
38      private final static int milisecond = 60000;
39      private final static int delay = 1000;
40      private FileOperation fo = new FileOperation();
41      private RuleParser rp;
42
43      public ScalingOptimization() {
44
45          rp = new RuleParser(ruleFilePath);
46          optimizationRules = rp.getAllOptimizationRules();
47          optimizationIDs = rp.getAllRuleIDs();
48          optimizationFrequencies = rp.getAllOptimizationFrequencies();
49          optimizationCounters = rp.getAllOptimizationCounters();
50          finalOptimizationUpLimits = rp.getAllOptimizationUpLimits();
51          finalOptimizationDownLimits = rp.getAllOptimizationDownLimits();
52          finalOptimizationThresholds = rp.getAllOptimizationThresholds();
53          optimizationTypes = rp.getAllOptimizationTypes();
54          optimizationServices = rp.getAllOptimizationServices();
55
56          if (optimizationRules.size() > 0 &&
57                  optimizationFrequencies.size() ==
58                  optimizationRules.size() &&
59                  optimizationRules.size() ==
60                  optimizationIDs.size() &&
61                  optimizationCounters.size() ==
62                  optimizationIDs.size() &&
63                  finalOptimizationUpLimits.size() ==
64                  optimizationTypes.size() &&
65                  finalOptimizationDownLimits.size() ==
66                  optimizationTypes.size() &&
67                  optimizationRules.size() ==
68                  optimizationTypes.size() &&
69                  optimizationRules.size() ==
70                  optimizationServices.size()) {
71              System.out.println(
72                      "*** " + optimizationRules.size() +
73                      " rules initialized ***" + optimizationRules);
```

```
74                optimizationSchedules = new LinkedList<Schedule>();
75                optimizationTimers = new LinkedList<Timer>();
76                applications = new LinkedList<String>();
77                instances = new LinkedList<String>();
78                //initiate objects
79                int instanceNo = 0; int applicationNo = 0;
80                onGoingOptimization[0] = false;
81                onGoingOptimization[1] = false;
82                for (int i = 0; i < optimizationRules.size(); i++) {
83                    if (optimizationTypes.get(i).contains("IaaSOptimization")) {
84                        instanceNo++;
85                        instances.add(rp.getIaaSInstance(optimizationRules.get(i)));
86                        //create new schedule for IaaS optimisation
87                        optimizationSchedules.add(new Schedule(
88                                        optimizationRules.get(i), i, instanceNo-1));
89
90                        //create new timer for IaaS optimisation
91                        optimizationTimers.add(new Timer());
92                        GreenIaaSPanel.refreshConsole(
93                                " Initiating Monitor schedule for " +
94                           optimizationIDs.get(i));
95                        fo.addToLog(iaaSLogFilePath,
96                                "Initiating Monitor schedule for " +
97                                 optimizationIDs.get(i));
98                    } else if (
99                             optimizationTypes.get(i).contains("PaaSOptimization")) {
100                       applicationNo++;
101                       applications.add(rp.getPaaSApplication(
102                        optimizationRules.get(i)));
103                       //create new timer for PaaS optimisation
104                       optimizationSchedules.add(
105                               new Schedule(optimizationRules.get(i), i,
106                                 applicationNo-1));
107                       optimizationTimers.add(new Timer());//create new timer for PaaS
108   optimisation
109                       GreenPaaSPanel.refreshConsole(
110                               " Initiating Monitor schedule for " +
111                                optimizationIDs.get(i));
112                       fo.addToLog(paaSLogFilePath,
113                               "Initiating Monitor schedule for " +
114                                 optimizationIDs.get(i));
115                   } else {
116                       System.err.println(
117                               "Err validating optimization type #1 " +
118                                optimizationTypes.get(i) +
119                               " id: " + optimizationIDs.get(i) +
120                               Calendar.getInstance().getTimeInMillis());
121                   }
122               }
123               /* begin periodical schedule according to specified
124                running frequencies for each services; */
125               for (int i = 0; i < optimizationFrequencies.size(); i++) {
126                   String id = optimizationIDs.get(i);
127                   int frequency = optimizationFrequencies.get(i);
128                   System.out.println(id + " will be running at " + frequency);
129                   optimizationTimers.get(i).scheduleAtFixedRate(
130                           optimizationSchedules.get(i), delay,
131                           (int)Math.round((milisecond*frequency)));
132                   System.out.println(" Initiating Monitor schedule for " +
133                           optimizationIDs.get(i));
134               }
135
136           } else {
137               System.err.println("Error reading rules... " +
138                Calendar.getInstance().getTimeInMillis());
139           }
140       }
141
142       public LinkedList<Timer> returnAllTimer() {
143           return optimizationTimers;
144       }
145
146       public LinkedList<Green.ScalingOptimization.Schedule> returnAllSchedule() {
147           return optimizationSchedules;
148       }
149
150       public boolean[] ongoingOptimisation() {
151           return onGoingOptimization;
```

```
152         }
153
154         public class Schedule extends TimerTask {
155
156             //initialise and retrieving data from individual rules:
157             public Integer[] threshold = new Integer[2];
158             public LinkedList<String> metrics = new LinkedList<String>();
159             public LinkedList<Integer> periods = new LinkedList<Integer>();
160             public LinkedList<String> statistics = new LinkedList<String>();
161             public String ruleID = new String("");
162             public Integer[] counter = new Integer[2];
163             public int downLimit= 0;
164             public int upLimit = 0;
165             public int frequency = 0;
166             public int monitorInternalInterval = 0;
167             public int monitorInternalValuesCount = 0;
168             public int latestMonitorValue = 0;
169             public int i = 0; public int j = 0;
170             public String optimizationType = new String("");
171             public String optimizationService = new String();
172             public String instanceID = new String("");
173             public String applicationName = new String("");
174             Integer[] continueScalingCount = {0,0};
175             UtilizationRegulator ur;
176             RuleEvolution re;
177             public LinkedList<Integer> filteredMonitorValues =
178                     new LinkedList<Integer>();
179             public LinkedList<Integer> monitorDataValues =
180                     new LinkedList<Integer>();
181
182             Schedule(Element rule, int i, int j) {
183                 super();
184              //fetch additional parameters from rules for each optimisation
185                 this.i = i; this.j = j;
186                 this.ruleID = optimizationIDs.get(i);
187                 threshold = rp.getAllOptimizationThresholds().get(i);
188                 counter = optimizationCounters.get(i);
189                 metrics = rp.getTheOptimizationMetrics(ruleID);
190                 statistics = rp.getTheOptimizationStatistics(ruleID);
191                 downLimit = rp.getAllOptimizationDownLimits().get(i);
192                 upLimit = rp.getAllOptimizationUpLimits().get(i);
193                 frequency = optimizationFrequencies.get(i);
194                 periods = rp.getTheOptimizationPeriods(ruleID);
195                 this.monitorInternalInterval =
196                         (int) Math.round(periods.get(0)/frequency/2);
197                 this.monitorInternalValuesCount =
198                         (int) Math.round(periods.get(0)/frequency);
199                 optimizationType = optimizationTypes.get(i);
200                 optimizationService = optimizationServices.get(i);
201                 re = new RuleEvolution(ruleFilePath);
202                 //utilisation data regulator
203                 ur = new UtilizationRegulator(fo, configFilePath);
204                 boolean bindingServiceSuccess =
205                         ur.setOptimizationTypeAndProvider(
206                                 optimizationType, optimizationService);
207
208                 if (ur == null || !bindingServiceSuccess ||
209                         monitorInternalInterval <= 0 || periods.size() <= 0 ||
210                         frequency <= 0 || upLimit <= 0 || re == null ||
211                         downLimit >= upLimit || statistics.size() <= 0 ||
212                         metrics.size() == 0 || threshold[0] <= 0 ||
213                         threshold[1] <= 0 ||  ruleID.length() == 0 ||
214                         optimizationType.length() == 0 ||
215                         monitorInternalValuesCount <= 0) {
216                     System.err.println(
217                             "Some data formats are wrong, or initiation failed... ");
218
219                 }
220                 System.err.println(
221                         "Info -> Utilization Regulator: " + ur + " " +
222                         bindingServiceSuccess);
223                 System.out.println(
224                         "Info -> MonitorInternal: " + monitorInternalInterval);
225                 System.out.println("Info -> Period: " + periods);
226                 System.out.println("Info -> Frequency: " + frequency);
227                 System.out.println("Info -> UpLimit: " + upLimit);
228                 System.out.println("Info -> DownLimit: " + downLimit);
229                 System.out.println("Info -> Statistics: " + statistics);
```

```
230                 System.out.println("Info -> Optimization Type: " + optimizationType);
231                 System.out.println("Info -> Metrics: " + metrics);
232                 System.out.println("Info -> Threshold[0]: " + threshold[0]);
233                 System.out.println("Info -> Threshold[1]: " + threshold[1]);
234                 System.out.println("Info -> RuleID: " + ruleID);
235
236                 System.out.println(
237                         "<< Monitor schedule initiated: " + ruleID +
238                         " at Frequency: " + frequency + " >>");
239                 if (optimizationType.contains("IaaSOptimization")) {
240                     GreenIaaSPanel.refreshConsole(
241                             "<< Monitor schedule created: " + ruleID +
242                             " at Frequency: " + frequency + " >>");
243                     fo.addToLog(iaaSLogFilePath,
244                             "<< New Monitor schedule created: " + ruleID +
245                             " at Frequency: " + frequency + " >>");
246                 } else if (optimizationType.contains("PaaSOptimization")) {
247                     GreenPaaSPanel.refreshConsole(
248                             "<< Monitor schedule created: " + ruleID +
249                             " at Frequency: " + frequency + " >>");
250                     fo.addToLog(paaSLogFilePath,
251                             "<< New Monitor schedule created: " + ruleID +
252                             " at Frequency: " + frequency + " >>");
253                 } else {
254                     System.err.println(
255                                         "Err validating optimization type #2: " +
256                         optimizationType + Calendar.getInstance().getTimeInMillis());
257                 }
258
259             }
260
261             public void run() {
262
263                 Integer[] Counter = {0,0}; //clear Counter before each count starts
264                 continueScalingCount[0]++; continueScalingCount[1]++;
265                 if (continueScalingCount[0] > Math.round(threshold[0]*1.1) ||
266                         continueScalingCount[1] > Math.round(threshold[1]*1.1)) {
267                     //restore original scaling parameters in case of intermittent
268 optimisation
269                     System.out.println(
270                         ">>> This " + ruleID + " will be an intermittent scaling, "
271                                 + "restoreing original parameters <<<");
272                     /*check whether current rule parameters are the same
273                      * as the original one before rule evolution
274                      */
275                     if (finalOptimizationThresholds.get(i)[0] !=
276                             rp.getTheOptimizationThreshold(ruleID)[0] ||
277                             finalOptimizationThresholds.get(i)[1] !=
278                             rp.getTheOptimizationThreshold(ruleID)[1] ||
279                             finalOptimizationUpLimits.get(i) !=
280                             rp.getTheOptimizationUpLimit(ruleID) ||
281                             finalOptimizationDownLimits.get(i) !=
282                             rp.getTheOptimizationDownLimit(ruleID)) {
283                         //intermittent scaling needs to restore original values
284                         optimizationSchedules.get(i).cancel();
285                         if (re.updateThresholds(ruleID,
286                         finalOptimizationThresholds.get(i)) &&
287                         re.updateUpLimit(ruleID, finalOptimizationUpLimits.get(i)) &&
288                                 re.updateDownLimit(ruleID,
289                         finalOptimizationDownLimits.get(i))) {
290                             System.out.println(
291                                     ">>> " + ruleID +
292                                     " original parameters have been restored  <<<");
293                             optimizationSchedules.add(i, new Schedule(
294                                     rp.getAllOptimizationRules().get(i) , i, j));
295                             optimizationTimers.get(i).scheduleAtFixedRate(
296                                     optimizationSchedules.get(i), delay,
297                                     (int)Math.round((milisecond*frequency)));
298                         } else {
299                             /*intermittent scaling and does not need restore,
300                             continue with current parameters*/
301                             System.err.println(
302                                     "!!! Error occured while restoring
303                                       original rule parameters");
304                         }
305                     } else {
306                         System.out.println(">>> " + ruleID +
307                                 " is with original parameters already <<<");
```

```
                    }
                }

                if (optimizationType.contains("IaaSOptimization")) {//IaaS optimisation
                    this.instanceID = instances.get(j);
                    if (instanceID.length() == 0) {
                        System.err.println("Error retrieving instance ID... " +
                    Calendar.getInstance().getTimeInMillis());
                    } else {
                        boolean iaaSMonitorStatisticsAvailble =
                                ur.isIaaSMonitorDataAvailable(optimizationType,
                                   instanceID, metrics, statistics, periods,
                                         frequency, monitorInternalInterval);

                        if (iaaSMonitorStatisticsAvailble) {
                            //get latest utilisation data for VM instance
                            latestMonitorValue = ur.getIaaSLatestMonitorStatistics();
                            System.out.println("Monitor data: " + latestMonitorValue);
                            fo.addToLog(iaaSLogFilePath,
                                    "# " + ruleID + " with instance " + instanceID +
                                    " RV: " + latestMonitorValue + " at " +
                                            Calendar.getInstance().getTime());
                            //adding data to list for threshold count
                            filteredMonitorValues.add(latestMonitorValue);
                            while (filteredMonitorValues.size() >=
                                    monitorInternalValuesCount) {
                                filteredMonitorValues.removeFirst();
                            }
                            //threshold count
                            for (int value : filteredMonitorValues) {
                                if (value >= upLimit || value <= downLimit) {

                                    if (value >= upLimit) {

                                        Counter[1]++;

                                        System.out.println(
                                                "# Updates: Due to " + value + " >=
                                         " + upLimit + ", " + "Uplimit Counter updated
                                         for " + instanceID);

                                    } else if (value <= downLimit) {
                                        Counter[0]++;
                                        System.out.println(
                                                "# Updates: Due to " + value + " <=
                                         " + downLimit + ", " + "Downlimit Counter
                                         updated for " + instanceID);

                                    }
                                } else {
                                    System.out.println(
                                            "# Updates: Monitor value is within limits
                                     for " + instanceID + " in rule: " + ruleID);

                                }
                            }
                            //update counters
                            optimizationCounters.set(i, Counter);
                            re.updateCounters(ruleID, Counter);
                            GreenIaaSPanel.refreshConsole("# Counter Updates for : " +
                             ruleID + ": " +  instanceID + Counter[0]  + " " + Counter[1]);
                            fo.addToLog(iaaSLogFilePath, "# Counter Updates for : " +
                             ruleID + ": " + instanceID + Counter[0]  + " " + Counter[1]);

                            while (Counter[1] >= threshold[1] ||
                                Counter[0] >= threshold[0]) {
                                //threshold violated
                                //counter reset;
                                Counter[0] = 0; Counter[1] = 0;
                                optimizationCounters.set(i, Counter);
                                re.updateCounters(ruleID, Counter);
                                optimizationSchedules.get(i).cancel();//cancel
current schedule
                                onGoingOptimization[0] = true;
                                IaaSOptimization is = new IaaSOptimization(
                                 this, onGoingOptimization[0]);

                                System.out.println("***Optimized Scaling initialised!
```

```
                                    " + ruleID + "***");
                                GreenIaaSPanel.refreshConsole(
                                        "***Optimized Scaling initialised! " +
                                   ruleID + "***");
                                fo.addToLog(iaaSLogFilePath,
                                        "***Optimized Scaling initialised! " +
                                 ruleID + "***");
                                if (is.scalingSucceeded()) {//success

                                    instances.add(j,
                                rp.getInstanceIDWithRuleIDFromRuleElement(ruleID));
                                    //get new instance from updated rule

                                    if (re.ruleEvolutionSucceeded(ruleID,
                                         filteredMonitorValues)) {
                                        System.out.println(
                                                "###Rule parameters evolved,
                                          optimization complete! " +
                                        ruleID + "###");
                                        GreenIaaSPanel.refreshConsole(
                                                "Rule parameters evolved,
                                           optimization complete! " +
                                        ruleID + "###");
                                        fo.addToLog(iaaSLogFilePath,
                                                "Rule parameters evolved,
                                           optimization complete! " +
                                        ruleID + "###");
                                    } else {
                                        System.err.println(
                                                "Rule parameters failed to evolve, "
                                                + "optimization finished with minor
                                                 errors! " + ruleID);
                                    }
                                    optimizationSchedules.add(i, new Schedule(
                                             rp.getAllOptimizationRules().get(i) ,
                                             i, j));//begin new schedule
                                    optimizationTimers.get(i).scheduleAtFixedRate(
                                             optimizationSchedules.get(i), delay,
                                         (int)Math.round((milisecond*frequency)));
                                    onGoingOptimization[0] = false;
                                } else {
                                    System.err.println("Noooooooo...Error happend " +
                                Calendar.getInstance().getTimeInMillis());
                                    onGoingOptimization[0] = false;
                                    GreenIaaSPanel.refreshConsole(
                                             "!!! Error happened while optimising: "
                                                    + ruleID + " when trying to scale:
                                                      " + instanceID + ", " + "rety next time");
                                    fo.addToLog(iaaSLogFilePath,
                                             "!!! Error happened while optimising: "
                                            + ruleID + " when trying to scale: " +
                                            instanceID + ", " + "rety next time");
                                }
                            }
                        } else {
                            System.out.println("# Awaiting monitor data...
                              for instance: " + instanceID);
                            GreenIaaSPanel.refreshConsole("# Awaiting monitor data...
                              for instance: " + instanceID);
                        }
                    }

                } else if (optimizationType.contains("PaaSOptimization")) {//PaaS
optimisation
                    this.applicationName = applications.get(j);
                    if (applicationName.length() == 0) {
                        System.err.println(
                                "Error retrieving Application Name... " +
                        Calendar.getInstance().getTimeInMillis());
                    } else {
                        boolean paaSMonitorStatisticsAvailable =
                                //get latest utilisation data for application
                                ur.isPaaSMonitorDataAvailable(optimizationType,
                                applicationName, metrics, statistics, periods,
                                        frequency, monitorInternalInterval);
                        if (paaSMonitorStatisticsAvailable) {
                            //get latest utilisation data for application
                            latestMonitorValue =
```

```
                                ur.getPaaSLatestDominantMonitorStatistics();
                                System.out.println("Monitor data: " + latestMonitorValue);
                                fo.addToLog(paaSLogFilePath, "# " + ruleID + " with
                                 application " +
                                applicationName + " RV: " + latestMonitorValue + " at " +
                                        Calendar.getInstance().getTime());

                                filteredMonitorValues.add(latestMonitorValue);
                                //adding data to list for threshold count
                                while (filteredMonitorValues.size() >=
                                monitorInternalValuesCount) {
                                    filteredMonitorValues.removeFirst();
                                }
                                //threshold count
                                for (int value : filteredMonitorValues) {
                                    if (value >= upLimit || value <= downLimit) {

                                        if (value >= upLimit) {

                                            Counter[1]++;

                                            System.out.println("# Updates: Due to " +
                                             value + " >= " +
                                            upLimit + ", Uplimit Counter updated for " +
                                              applicationName);

                                        } else if (value <= downLimit) {
                                            Counter[0]++;
                                            System.out.println("# Updates: Due to " +
                                               value + " <= " +
                                            downLimit + ", Downlimit Counter updated for
                                          " + applicationName);

                                        }
                                    } else {
                                        System.out.println(
                                                "# Updates: Monitor value is within limits
                                         for " + applicationName + " in rule: " + ruleID);

                                    }
                                }
                                //counter updates
                                optimizationCounters.set(i, Counter);
                                re.updateCounters(ruleID, Counter);
                                GreenPaaSPanel.refreshConsole(
                                        "# Counter Updates for : " + ruleID + ": " +
                                applicationName + Counter[0]  + " " + Counter[1]);
                                fo.addToLog(paaSLogFilePath,
                                        "# Counter Updates for : " + ruleID + ": " +
                                applicationName + Counter[0]  + " " + Counter[1]);

                                while (Counter[1] >= threshold[1] ||
                                  Counter[0] >= threshold[0]) {
                                    //threshold violated
                                    //counter reset
                                    Counter[0] = 0; Counter[1] = 0;
                                    optimizationCounters.set(i, Counter);
                                    re.updateCounters(ruleID, Counter);
                                    optimizationSchedules.get(i).cancel();//cancel
current schedule
                                    onGoingOptimization[1] = true;
                                    PaaSOptimization ps = new PaaSOptimization(
                                     this, onGoingOptimization[1]);

                                    System.out.println("***Optimized Scaling initialised!
                                    " + ruleID + "***");
                                    GreenIaaSPanel.refreshConsole(
                                            "***Optimized Scaling initialised! " + ruleID
                                           + "***");
                                    fo.addToLog(iaaSLogFilePath,
                                            "***Optimized Scaling initialised! " + ruleID
                                         + "***");
                                    if (ps.scalingSucceeded()) {//success

                                        if (re.ruleEvolutionSucceeded(ruleID,
                                       filteredMonitorValues)) {
                                            System.out.println(
                                                    "Rule parameters evolved,
```

```
                                        optimization complete! " +
                                    ruleID + "###");
                                    GreenPaaSPanel.refreshConsole(
                                            "Rule parameters evolved,
                                           optimization complete! " +
                                    ruleID + "###");
                                    fo.addToLog(paaSLogFilePath,
                                            "Rule parameters evolved,
                                            optimization complete! " +
                                    ruleID + "###");
                                } else {
                                    System.err.println("Rule parameters failed to
                                  evolve, optimization finished with minor errors!
                                    " + ruleID);
                                }
                                optimizationSchedules.add(
                                                                        i, new
                                Schedule(rp.getAllOptimizationRules().get(i) , i,
                                j));
                                //begin new schedule
                                optimizationTimers.get(i).scheduleAtFixedRate(
                                        optimizationSchedules.get(i), delay,
                                     (int)Math.round((milisecond*frequency)));
                                onGoingOptimization[1] = false;
                            } else {
                                System.err.println("Noooooooo...Error happend " +
                            Calendar.getInstance().getTimeInMillis());
                                onGoingOptimization[1] = false;
                                GreenPaaSPanel.refreshConsole(
                                        "!!! Error happened while optimising: "
                                 + ruleID + " when trying to scale: " +
                                applicationName + ", " + "rety next time");
                                fo.addToLog(paaSLogFilePath,
                                        "!!! Error happened while optimising: "
                                + ruleID + " when trying to scale: " +
                                applicationName + ", " + "rety next time");
                            }
                        }
                    } else {
                        System.out.println("# Awaiting monitor data... "
                                + "for application: " + applicationName);
                        GreenPaaSPanel.refreshConsole(
                                "# Awaiting monitor data... "
                                + "for application: " + applicationName);
                    }
                }
            } else {
                System.err.println("Err validating optimization type #3 " +
            Calendar.getInstance().getTimeInMillis());
            }
        }
    }

}
```