Tutorial how to use Structured Simulation framework

General idea of the framework

The framework was created for automated simulation to analyze adaptive systems. The goal of this framework is to facilitate the generation of a large number of simulations in a structure way. The idea of this tool is to be as generic as possible to be used in different fields of simulation (traffic, human behavior, energy, etc…). To use it the user need to create the "glue code" and provide input files. This is a multi-threading framework developed in Java.

Functionalities

Here a list of functionalities provide by the framework :

- Creation of a list of simulation with modifiers applied on a base scenario
- Environment of simulation created sort by their probability of happening
- Possibility to define a cut off to stop the framework
    - Until a define probability value
    - After an amount of time
    - After a number of simulation
- Possibility to extract measures from generated results files
    - Creation of a file with all extracted measures
- Creation of one folder by simulation environment with
    - Parameters file(s)
    - Results file(s)
    - Measures file(s)
- Generation of a summary file to know which modifiers were applied and in which order for each simulation

Use of the framework

First, you need to download the jar of the framework "structSimV1.jar" on GitHub at this address: https://github.com/CaTaram/structSim

Then, add it to your java project.

To use the tool, you need to create some files:

- Class main
- Gluecode class
- Properties file
- And the desired number of modifier classes

Below, explanations with more precise information for each classes needed. In addition, at the end of this document you will found some examples of the use of the framework.

Class main

The main class is the class that will launch all the simulation process. This class needs to extend the "StartProgram" abstract class to have access to the "startProgram" method. The startProgram method takes in parameters : a String that is the path to the config file and an instance of the gluecode class.

Gluecode class

The glue code class need to extend the "ASimulationSystemHandler" abstract class.

Then the core of these methods need to be create according to your simulator :

- startSimulation : to start the simulator
- stopProgram : if needed to stop the simulator
- ExtractMeasures : to extract "some" measures from result files of simulations
- initiateModifierClass : to return a list of modifier classes that need to be used to do simulations
- readParametersFile : to read parameters file and return a vector of parameter
- writeParametersFile : to write the parameters file base on a vector of parameter

Properties file

Some properties must be completed to ensure the proper functioning of the framework. The information needed are :

- pathOUT : the folder path where files for each simulation are saved
- pathParameters : the path to the parameters file
- pathSimulator : the path where the simulator is installed on the computer
- cuttOfPlanning : the value of the cut of planning
- typeCuttOfPlanning : the type of the cut of planning (int : number of iteration, hours, minutes, day, criteria : base on the probability of the simulation, must to be between 0 and 1)

BE CAREFULL, it needs to be a file with the ".properties" extension.

Modifier Classes

You can create an infinite number of modifier classes. Each of these classes make modification to one or multiple parameters.

A modifier class have to extends the AModifier abstract class, and you need to fill the override method "applyModifier".

In a modifier class the parameter to change and the key of this parameter need to be declared. And for each modifier a probability to happened must be defined in the constructor.

Examples

A very simple Example :

For this very simple example, we do not really use a simulator. But like a simulator, we use a list of parameters with two modifiers to modify the value of the parameter “val2” each time a “simulation” is done.

The “parameters.txt” file contain the list of parameters for the simulation and looks like:

```
val1=1
val2=2
val3=3
val4=4
val5=5
```

For our simulations, all saved files have to be saved in the /ResultSim folder and we decide to do only 10 simulations, it’s why the cuttOfPlanning is 10 and the typeCuttOfPlanning is INT. Below the properties.config file in more details for our example :

```
# For all path the folder must be created
pathOUT = ./ResultSim
pathParameters = ./data/simulation/parameters.txt
pathSimulator = c:/mySimulator
#the place where the simulator create automatically results file for a simulation.
pathToSimulatorResultFile = c:/mySimulator/results/results.txt
# CuttOfPlanning = How many time must the planning run.
cuttOfPlanning = 10
# type of cuttOfPlanning : INT, HOURS, MINUTES, DAY, CRITERIA
# INT = iteration or laps.
# Criteria = based on the probability. The numbers must be between 1 and 0. ex.:0.5
typeCuttOfPlanning = INT
```

To create the simulation planning, we created two different modifiers :

- ModifierClass1 that add 1 to the value of the parameter “val2”
- ModifierClass2 that add 10 to the value of the parameter “val2”

```
public class ModifierClass1 extends AModifier{
    Parameter paramToChange;
    String keyToChange = "val2";

    public ModifierClass1(){
        this.probability = 0.2;
        this.name = "Modifier1";
    }

    @Override
    public Environment applyModifier(Environment env) {


        ConcurrentHashMap<String, Double> arrayParam = new ConcurrentHashMap<>(env.getLisOfParameters());
        double valueToChange = arrayParam.get(keyToChange);
        valueToChange += 1;
        arrayParam.put(keyToChange, valueToChange);

        env.setLisOfParameters(arrayParam);

        return env ;
    }


}
```

```java
public class ModifierClass2 extends AModifier{

    Parameter paramToChange;
    String keyToChange = "val2";

    public ModifierClass2(){
        this.probability = 0.5;
        this.name = "Modifier2";
    }

    @Override
    public Environment applyModifier(Environment env) {


        ConcurrentHashMap<String, Double> arrayParam = new ConcurrentHashMap<String, Double>(env.getLisOfParameters());
        double valueToChange = arrayParam.get(keyToChange);
        valueToChange = valueToChange + 10;
        arrayParam.put(keyToChange, valueToChange);

        env.setLisOfParameters(arrayParam);

        return env ;
    }
}
```

For this example, we called the glue code class “SimpleSimulationHandler”. Since we do not use an external simulator, we don’t need to fill all overridden methods. In the glue code we :

- create an instance of each modifier class
- put them in a list of modifier in the initateModifierList method
- we didn’t use startSimulation and stopProgram method for this example
- we adapt the readParameterFile and writeParamterFile according for the parameter file template of this example
- and we adapt the extractMeasure method to extract method from result file of this example.

```java
public class SimpleSimulationHandler extends ASimulationSystemHandler{

    private ModifierClass1 mc1 = new ModifierClass1();
    private ModifierClass2 mc2 = new ModifierClass2();
    /**
     * Constructor
     */
    public SimpleSimulationHandler() {
    }

    @Override
    public List<AModifier> initiateModifierList() {
        listModifierClass.add(mc1);
        listModifierClass.add(mc2);

        return listModifierClass;
    }


    @Override
    public void startSimulation(String pathToInputFile) {


    }


    @Override
    public void stopProgram() {
        // TODO Auto-generated method stub

    }
```

```java
@Override
public Vector<Parameter> readParametersFile(String parametersFilePath) {
    String separator = "=";
    Vector<Parameter> parametersList = new Vector<Parameter>();

    String key;
    double value;

    BufferedReader in;
    try {
        in = new BufferedReader(
                new InputStreamReader(new FileInputStream(parametersFilePath), "UTF-8"));

        String line = "";

        while ((line = in.readLine()) != null) {

            int pos = line.indexOf(separator);
            key = line.substring(0, pos);
            value = Double.parseDouble(line.substring(pos + 1, line.length()));
            parametersList.add(new Parameter(key, value));

        }
        in.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return parametersList;

}
@Override
public void writeParametersFile(Vector<Parameter> setOfParameters, String locationToStore) {
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(locationToStore + "/myParamFile.txt"));
        for(Parameter p : setOfParameters){
            bw.write(p.getKey() + "=" + p.getValue());
            bw.newLine();
        }
        bw.flush();
        bw.close();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}
@Override
public Vector<Measure> extractMeasures(String resultsFile) {

    String separator = "=";
    Vector<Measure> measuresList = new Vector<Measure>();

    String measureKey;
    String measureValue;

    BufferedReader in;
    try {
        in = new BufferedReader(
                new InputStreamReader(new FileInputStream(resultsFile), "UTF-8"));

        String line = "";

        while ((line = in.readLine()) != null) {

            int pos = line.indexOf(separator);
            measureKey = line.substring(0, pos);
            measureValue = (line.substring(pos + 1, line.length()));
            measuresList.add(new Measure(measureKey, measureValue));

        }
        in.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return measuresList;

}
```

We named the main class of this example “simulation”. In this class, we define the String of the properties file path and we create an instance to the glue code. Then we pass these two elements to the startProgram method.

```java
public class Simulation extends StartProgram{

    // Mock-up code for a simple simulation.
    public static void main(String[] args) throws IOException {

        //Path of the config file
        String pathConfigFile = "./src/main/java/ch/hevs/silab/structuredsim/simulators/mockUpSim/config.properties";

        //Custom Class
        SimpleSimulationHandler ssh = new SimpleSimulationHandler();

        startProgram(pathConfigFile, ssh);

    }

}
```