VGU Project

Background

Due to the massive expansion of renewable energy, many European countries launched a conversion to the next generation power grid. The large-scale use of distributed power generators such as wind and photovoltaic energy in low-voltage networks places new challenging demands on electricity grids: While low-voltage networks in traditional power grids are conventionally unmonitored, the decentralized nature of volatile and renewable energies results in a strong need to control and monitor actors in order to react timely to the variable energy demands:

The electricity grid is an unstable balance in which power generation and power consumption must balance each other at every time. When the mains frequency decreases, consumers slow down energy production more than electricity generators can generate. As the mains frequency increases, there is more power generated than needed. Thus, the mains frequency is the reference value of the grid stability. For longer-term frequency decreases, additional energy needs to be fed into the grid and, conversely, frequency increases require power plants to rapidly reduce their capacity. A loss of stability occurs when the grid is no longer able to return to a stable operating point following the occurrence of a disturbance that results in a large imbalance between power generation and load. This can then cause persistent oscillations of the grid frequency with automatic shut-downs of generating units and hence blackouts or in the worst case it can damage critical infrastructure. In order to keep the grid frequency stable at 50 Hz, it requires an intelligent supply-demand mechanism and, in the case of over-frequency or under-frequency, of a functioning control energy system, which is summarized as 'Demand Side Management' (DSM).

Objective

Simulations can be used to analyse and evaluate various control mechanisms to improve DSM. They allow to generate unorthodox consumer profiles or abnormal scenarios, that are not easy to reproduce in real world environments. The objective of this project is to develop a mains frequency simulator for a small metropolitan power grids. The resulting application should allow the user to simulate and visualize the stability of a power grid with various parameters.

Implementation

The project is conducted in groups with student project managers. Each group will develop a simulator, which is divided into different components. Each component has a specified interface and is its own Java-Project build to an exchangeble JAR-file. The components are as follows: Energy Generator (e.g. power generators, batteries), Energy Consumer (e.g. electrical devices, household clusters), Control System (e.g. power meters, control circuits) and Graphical User Interface. The pre-defined interfaces, JUnit test cases and runnables are delivered in additional Java projects.

Firstly, each group develops a project plan and subsequently implements the components according to the defined milestones. During the project, components may be interchanged between different student groups. Knowledge in the areas ’Software Development’ and ’Software Engineering’ is practially used. The result of the project will be the software and a detailed report which will also be the foundation for the grading of the project.

Requirements

The simulator provides a grid environment with generators and consumers. Each component may show individual automatic or externally driven behaviour. Additionally, the state of these components is, according to individual capabilities, measured and altered every two hours by a control model - which is in the following called an iteration. The GUI aims to summarize the important details of each iteration and may give control over environment, components and external conditions. The total amount of running consumers can be defined as parameter in percentage.

Objective
- simulate a power grid over 24 hours (12 Iterations)
- implement different types of consumer models and generator models
- implement following scenarios with 100 different consumers with average consumption of 100W

Iteration	Running
0	50%
1	20%
2	15%
3	45%
4	75%
5	60%
6	55%
7	40%
8	45%
9	65%
10	95%
11	75%

Table 1: Typical Consumption Pattern

Scenario 1: Regular Day
- The consumers show a consumption according to table 1.
- The frequency should at no time exceed 50Hz +- 2 Hz

Scenario 2: Outage
- The consumers show a consumption according to table 1.
- Due to an outage, generators will unregister at a iteration between 0 and 5
- The frequency should stabilize after some more iterations

The interfaces and JUnit test cases, which are described later, aim to fullfill following requirements:

Generator Model
- MUST be registered at the Control Model
- MUST provide an interface to measure the momentary power
- MUST provide the minimum and maximum supply
- MUST provide the maximum supply-change per iteration
- MUST provide an interface for the Control Model to request a supply change
- MAY change supply due to internal or external conditions (e.g. time, weather, price ..)
- MAY turn automatically off after a nr. of iterations (e.g. battery, pump power plant)

Consumer Model
- MUST be registered at the Control Model
- MUST have an On and Off state, each with defined power
- MAY be registered as cluster with other consumers (e.g. households)
- MUST provide an interface to measure the power
- MUST provide an interface for remote changes
- MAY change state due to internal or external conditions (e.g. time, weather, price ..)

Control Model
- MUST be able register an arbitary number of generators and consumers
- MUST be able to un-register each component
- MUST compute the total demand every iteration
- MUST compute the total cost every iteration
- MUST compute the mains frequency every iteration
 (50Hz minus the difference of demand-supply, whereas 10% equals 1Hz)
- MUST unregister 10% Generators, if mains frequency > 51Hz (overload)
- MUST unregister 15% of the consumers if mains frequency < 49Hz (blackout)
- MUST unregister components if mains frequency > 51Hz < 49 Hz for 3 Iterations (defect)
- MAY request state changes in consumers for demand side management
- MAY request supply and state changes in generators for demand side management
- MAY request to start or shutdown generators for demand side management
- MAY change the electricity price

GUI
- MUST be capable to control current the iteration
- MUST be capable to show current demand, supply and frequency
- MUST be capable to show the number of consumer and generators
- MUST be capable to show current weather and electricity price
- MUST be capable to show name and power of individual consumer or generator

Interfaces

AbstractComponent

```
	protected String name;
	private double maxPower;
	private double minPower;
	private double maxChange;
	private double minChange;
	public abstract double getPower();
	public abstract void setPower(double power);
	public abstract void next();
	public abstract double getCost();
	public double getMinChange() {
		return minChange;
	}
	public void setMinChange(double minChange) {
		this.minChange = minChange;
	}
	public double getMaxChange() {
		return maxChange;
	}
	public void setMaxChange(double maxChange) {
		this.maxChange = maxChange;
	}
	public double getMaxPower() {
		return maxPower;
	}
	public void setMaxPower(double maxPower) {
		this.maxPower = maxPower;
	}
	public double getMinPower() {
		return minPower;
	}
	public void setMinPower(double minPower) {
		this.minPower = minPower;
	}
```

IControl

```
	void addGenerator(AbstractComponent generator);
	void removeGenerator(AbstractComponent generator);
	List<AbstractComponent> getGenerators();
	void addConsumer(AbstractComponent consumer);
	void removeConsumer(AbstractComponent consumer);
	List<AbstractComponent> getConsumers();
	double getTotalDemand();
	double getTotalSupply();
	double getFrequency();
	double getCost();
```

Unit Test Cases:

TestLowFrequency
- register a 100W Device
- register a 90W Generator (minimum and maximum supply)
- I0: Check Frequency equals 49Hz

TestHighFrequency
- register a 90W Device
- register a 100W Generator (minimum and maximum supply)
- I0: Check Frequency equals 51Hz

TestDemandSupplyComputation
- register two 90W Device
- register two 100W Generator (minimum and maximum supply)
- I0: Check total demand equals 180W and total supply equal 200W

TestSupplyAdaption
- register a 100W Device
- register a Generator with 90W, minimum 90W, maximum 100W, max-change 5W
- I0: check supply is 90W
- I1: check supply > previous supply
- I2: check supply > previous supply

TestControlledBlackout
- register 100 individual 1W Device
- register a 70W Generator (minimum and maximum supply)
- I0: Check Frequency is 47Hz, demand 100W, supply 70W -> 15 Devices will be unregistered
- I1: Check Frequency is ~48.7Hz, demand 85W, supply 70W -> 13 Devices will be unregistred
- I2: Check frequency is ~49.8Hz demand 72W, supply 70W

TestDeviceCluster
- register a cluster with 100 x 1W Device
- register a generator with 70W (minimum and maximum supply)
- I0: Check cluster with all devices will unregister

TestGeneratorTurnOff
- register a 100W Device
- register a 1000W Generator (minimum and maximum supply)
- I0: Check Generator will unregister

TestOverDemand
- register 100 individual 1W Device
- register a 10W Generator (minimum and maximum supply)
- I4: Check all Generator and devices will unregister

TestOverSupply
- register a 10W Device
- register 100 individual 10W Generator (minimum and maximum supply)
- I4: Check all Generators and Devices will unregister

Addional Task: Pricing Model

Implement an pricing model for generators and consumers, with 100 different consumers of an avarage consumption of 100W. Consumers provides 1 Euro per Watt.

A Generator consume 50 Cent for every 1 Watt power provided. However, generators have following constrains, a generator is more expensive if:

a) +25 Cent if the max-power is less then 2500W.
b) +25 Cent if the max-change is more then 50% of the max-power
c) +25 Cent if the min-change is less then 50% of the max-power

Implement your generators and control model in a way to ensure an overall positive balance.

Grading System

Total Points 150

Practical Tasks (final presentation)

a) Show scenario 1	10
b) Show scenario 2	10
c) Exchange a library with another group	10
d) Complete of all unit test cases	10
e) Show the additional task	10
Total:	**50 Points**

Documentation (written Report)

a) Presentation (e.g. figures, format, document organization)	10
b) Describtion of Architecture (Concepts, Structure, Interfaces)	10
c) Code Documentation (Input/Output, Control Flow, Classes)	10
d) User Documentation (UI, Usecases, Examples)	10
e) Evaluation (Solution for scenario 1,2 and optional task)	10
Total:	**50 Points**

Other **50 Points**

Other Points are given for e.g. project management, innovative extensions or exceptional effords in other areas (to the group or individual students).