PROJECT – INFORMATION SYSTEM FOR A SMART-GRID

lectures on information technology for energy systems

ENSE3/2A/SEM and master SGB

stephane.ploix@grenoble-inp.fr

2016-2017

Abstract

The project aims at simulating the interactions between actors of a smart micro-grid i.e. a micro-grid with communication capabilities. Different tools are going to be manipulated: UML modeling, RESTful Web Services, XML formalism, Relational Database and SQL language. Java and SQLlite technologies are going to be used.

From a pedagogical point of view, this work leads to the skills required for the so-call smart-energy area, which is located in between energy and computer science.

Prerequisites in Java are required.

Contents

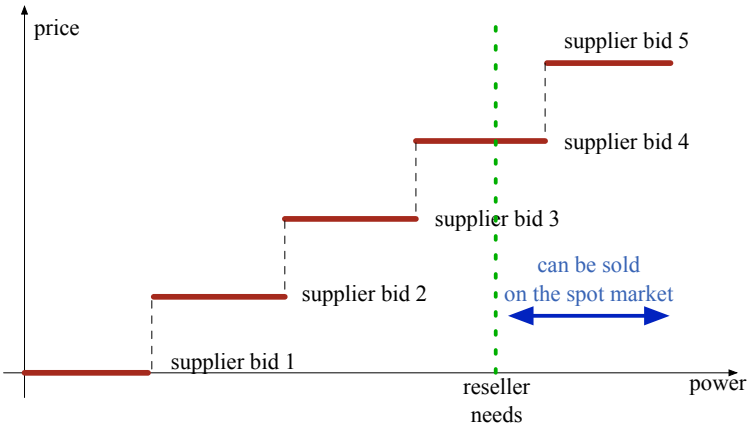
Part I
Main actors in a Smart-Grid

1. General objective of the project

The general objective of the project is to develop a modular web-oriented Java simulator of actors in a smart micro-grid, representing in particular economical exchanges between actors.
The different actor roles involved in a smart-grid are given below.

1.1. Scheduling Coordinator or Balance Responsible Party

A *scheduling coordinator* or *balance responsible party* [1] is a crucial actor connected with several electricity suppliers and one or several electricity resellers. His responsibility is to equilibrate production with consumption. If the suppliers, with whom he is dealing with, cannot provide energy, he can buy electricity on the spot market. He uses to own his own production means. In this case, he cumulates the roles of *balance responsible party* and electricity supplier[2]. If the consumption he is managing is lower than its production, he can also resell energy on the spot market. He is in relation with the transmission system operator [3] and distribution system operators[4] in order to avoid congestions on the grid. At a given time, a scheduling operator is characterized by a curve representing his supplier energy bids and his electricity reseller needs:



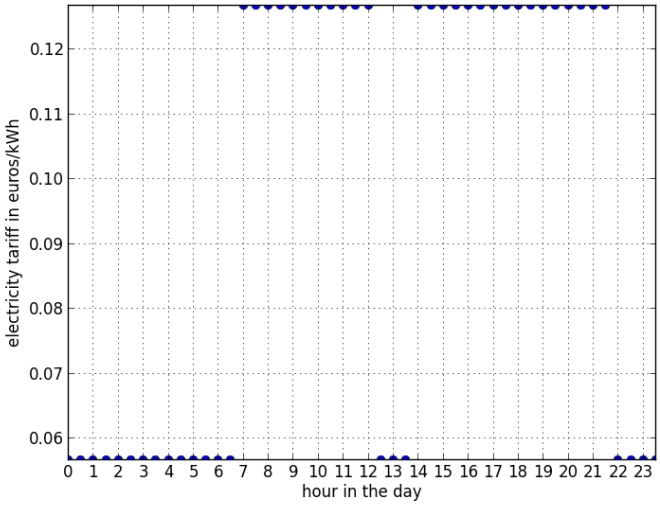
1.2. Supplier

In this study, it is assumed that an *electricity supplier*[5] is dealing with one and only one *scheduling coordinator*. Some electricity suppliers directly sell energy on the spot market. It is also assumed that *supplier* is a role corresponding to only one production means. Several roles can be cumulated by a single actor. Production means are for instance: windmill, photovoltaic plant, hydro-power plant over water, hydro-power plant in mountains, gas turbine, nuclear plant,... Each 60 minutes, he proposes a production at a given price depending on the time of the day. If the production plant he is managing is off, there is usually a cost to start it up.

[1] responsables d'équilibre
[2] he may also cumulate a role of reseller and aggregator but it is not considered in the next
[3] gestionnaire du réseau de transport
[4] gestionnaire d'un réseau de distribution
[5] producteurs d'électricité

1.3. Reseller/Aggregator

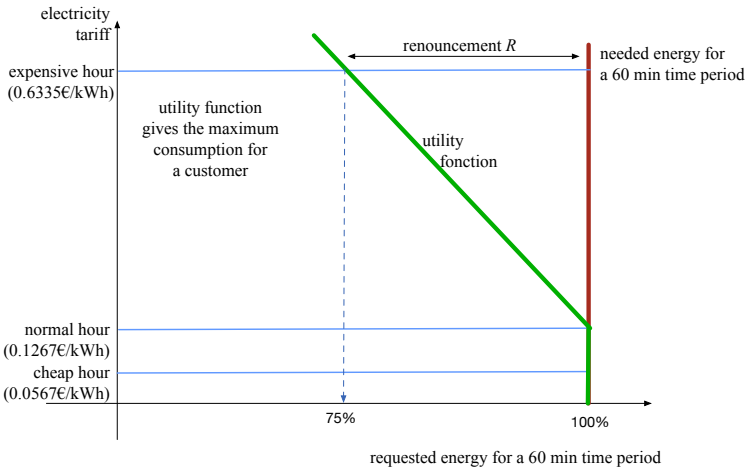
A *reseller/aggregator*[6] is in business with a *scheduling coordinator*. He has got his own customers with their energy needs. Let's focus on the new potential aggregator role (see energynet.pdf document). Various roles and players – such as electricity suppliers and balance responsible parties – are clearly defined in the market rules. First and foremost, it is recommended to introduce 'aggregator' as part of the market design of the electricity market. It is therefore recommended to base this on the following definition: an aggregator has entered into an agreement with an electricity customer on access to disposing of the electricity customer's flexible consumption and/or generation in the electricity market. The aggregator pools flexibility from customers and converts it into electricity market services, for example for use by the TSO, DSO and/or BRP. Whether a reseller is also an aggregator, it can influence its customers for them to reduce their consumption and therefore modify their energy needs (implicit flexibilities with variable tariffs or explicit flexibilities with targeted requests can be requested). Regarding his customers, electricity tariffs are announced 24 hours before application. In this work, the tariff follows the French 2 levels standard "heures creuses" at 0.0567€/kWh and "heures pleines" at 0.1267€/kWh but in addition, he can announced the day before an expensive 2 hours period at 0.6335€/kWh. The next figure represents the distribution of the "heures creuses/heures pleines":



1.4. Electricity consumer

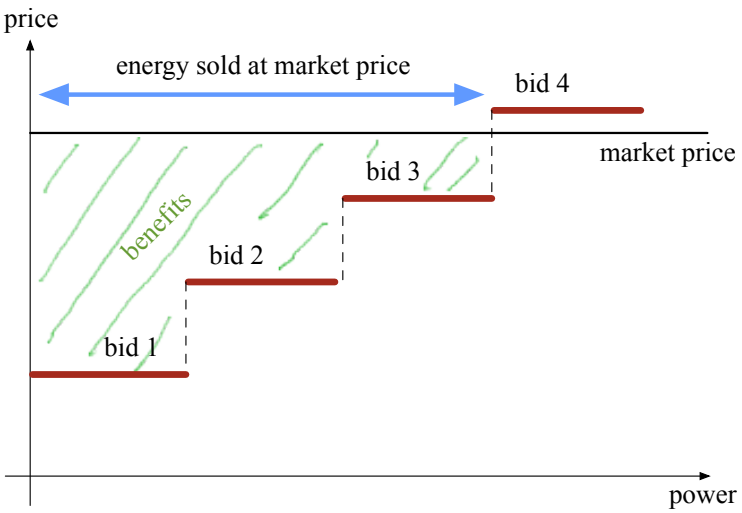
Electricity consumer is the electricity end-user. It is characterized by a typical electricity need curve with a 60 minutes time resolution. Depending on the announcement of a 2 hours expensive period, he may modify its electricity needs. It is modeled here by a utility function that represents the maximum consumption he accepts according to the electricity tariff.

[6] commercialisateur / aggrégateur



1.5. Spot market

Regarding the spot market, only the day ahead market is considered. Energy is traded for each 60 min period. To reach an equilibrium, a scheduling operator can either sell or buy energy on the spot market in addition to the balancing out he has done between his own electricity suppliers and reseller needs. It is a one turn market. At each 60 min, some scheduling operators may want to buy a given quantity of energy or to sell electricity by providing a curve looking like:



Question 1: Draw a general UML class diagram that points out how actors are structured.

Question 2: Identify actors involves and summarize information into a use case UML diagram: *equilibrate*[7] *consumption and production*, *trade*[8] *at day ahead spot market* and *reduce customer*[9] *consumptions*. See 'UML-tutorial.pdf' for help on UML and, possibly, 'UML-doc.pdf' for more details.

Question 3: Detail use cases and identify the interactions between involved actors (Reseller/Aggregator, Scheduling Coordinator, Supplier and Spot Market) using UML sequence diagrams. Each actor own activity should appear.

[7] main actor is scheduling coordinator
[8] main actor is scheduling coordinator
[9] main actor is reseller/aggregator

Part II
Dataset

2. Relational database

When manipulating huge bunches of data, spreadsheets are no longer sufficient but databases are preferred: they are much faster to access a data because of internal mechanisms like hash tables, but also because they can handle many interconnected tables and they generally provide a web access to data. There are different kinds of database: hierarchical (like file systems), object, relational, object relational,... but the relational database are widely used in information systems, for data archiving or in dynamic web sites. A relational database can be seen as a set of interdependent tables called relations. Some database systems can be accessed through the Internet such as Oracle[10], Microsoft SQL server[11], MySQL[12] or PostgreSQL[13], some others are just local database systems such as SQLite[14] or HSQLDB[15].

Globally speaking, a database system has the following characteristics:

- physical independence: the data access is standardized whereas the way data are stored depends on the database system;
- logical independence: a set of data can be seen in different ways, with different access rights, for different users;
- standardized access to large bunches of data;
- fast data access;
- for Internet database, it offers a unique access point to large bunches of data that may be physically distributed;
- for Internet database, remote access to data, concurrent data access management (using transactions) and access right management.

Almost all relational databases can be accessed thanks the Structured Query Language (SQL), which can be decomposed into 3 subsets:

- the Data Description Language for the creation of databases
- the Data Manipulation Language for searching, adding, updating or removing data
- the Data Control Language for managing transactions and integrity

The mostly used data types in SQL are:

- TEXT or VARCHAR(n): strings of characters;
- INTEGER: integer values;
- FLOAT, REAL or NUMERIC: real numbers.

SQL language is standardized but, depending on the database system, not all elements of the language are supported.

[10] `http://www.oracle.com`
[11] `http://www.microsoft.com/en-us/server-cloud/products/sql-server/`
[12] `http://www.mysql.com`
[13] `http://www.postgresql.org`
[14] `http://www.sqlite.org`
[15] `http://hsqldb.org`

2.1. Data Description Language

In order to create or remove a table:

```
CREATE TABLE client (numClient INTEGER, name TEXT, surname TEXT, city TEXT);
CREATE TABLE client (numClient INTEGER, name TEXT, surname TEXT, PRIMARY KEY(numClient) );
DROP TABLE client;
```

Note the primary key, which can be composed of 1 or several columns (called attributes of a relation): it makes it possible to reference in a unique way each row (called a tuple of a relation) of a table. The primary key is an identifier used to represent relations between tables. Indeed, representing a client in an invoice table can be done by adding a reference to the client identifier 'numClient':

```
CREATE TABLE invoice (numInvoice INTEGER, description TEXT, numClientRef INTEGER,...
  PRIMARY KEY(numInvoice), FOREIGN KEY(numClientRef) REFERENCES client(numClient) ) ;
```

In order to add a column:

```
ALTER TABLE client ADD income FLOAT;
```

In order to rename a table:

```
ALTER TABLE client RENAME TO infoclient;
```

2.2. Data Manipulation Language

In order to insert a row in a table:

```
INSERT INTO client VALUES(2,'olivier','Martin','Grenoble');
INSERT INTO table (nom_colonne_1,nom_colonne_2,...) VALUES ('valeur␣1','valeur␣2',...);
```

In order to update a row in a table:

```
UPDATE client SET ville='Sophia-Antipolis' WHERE ville='Nice';
```

In order to suppress a row in a table:

```
DELETE FROM client WHERE ville='Sophia-Antipolis';
```

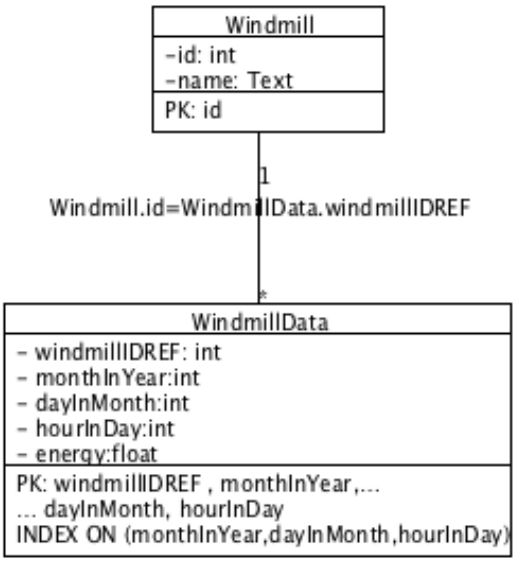
In order to get data from a table:

```
SELECT nom, prenom FROM client;
SELECT * FROM client WHERE ville='Lyon';
SELECT * FROM commande WHERE quantity>=3;
SELECT * FROM commande WHERE prix BETWEEN 50 AND 100;
SELECT * FROM commande WHERE quantite IS NULL;
SELECT * FROM client WHERE ville LIKE 'Saint\%';
SELECT prenom FROM Client WHERE nom IN('Dupont', 'Durand', 'Martin');
SELECT * FROM commande WHERE client.numClient=5 ORDER BY prix ASC;
SELECT * FROM commande WHERE client.numClient=5 ORDER BY prix DESC;
```

The following keywords can be used to built up requests: NOT BETWEEN, NOT NULL, NOT LIKE or NOT IN.
In order to get data from several tables:

```
SELECT nom, date, quantite FROM client, commande WHERE client.numClient=commande.numClient;
SELECT Ct.numClient, nom AS nm, date, quantite FROM client AS Ct, commande AS Cd...
    ... WHERE Ct.numClient=Cd.numClient ORDER BY nom;
SELECT nom FROM client WHERE numClient IN(SELECT numClient FROM commande...
    ... WHERE date='16-04-2013');
SELECT DISTINCT numClients FROM commande WHERE quantite > ...
    ... (SELECT quantite FROM commande WHERE numClient=1);
SELECT numClient, SUM(quantite) FROM commande GROUP BY numClient;
SELECT numClient, COUNT(numCommande) FROM commande GROUP BY numClient;
SELECT numcommande FROM commande WHERE prix<100 UNION SELECT numcommande ...
    ... FROM commande WHERE numClient=5;
```

The keyword AS makes it possible to create an alias for the sake of simplification. Aggregation functions are available: AVG(), SUM(), MIN(), MAX() or COUNT().
A request covering multiple tables can be done thanks to a key exchange mechanism. Let's consider the 2 following tables (attributes stand for columns):



These tables have been generated by the following SQL code:

```
CREATE TABLE Windmill (id INTEGER UNIQUE NOT NULL, name TEXT, PRIMARY KEY (id));
CREATE TABLE WindmillData (windmillIDREF INTEGER NOT NULL, monthInYear INTEGER,...
... dayInMonth Integer, hourInDay int, energy FLOAT,...
... PRIMARY KEY (windmillIDREF, monthInYear, dayInMonth, hourInDay),...
... FOREIGN KEY (windmillIDREF) REFERENCES Windmill(id));
```

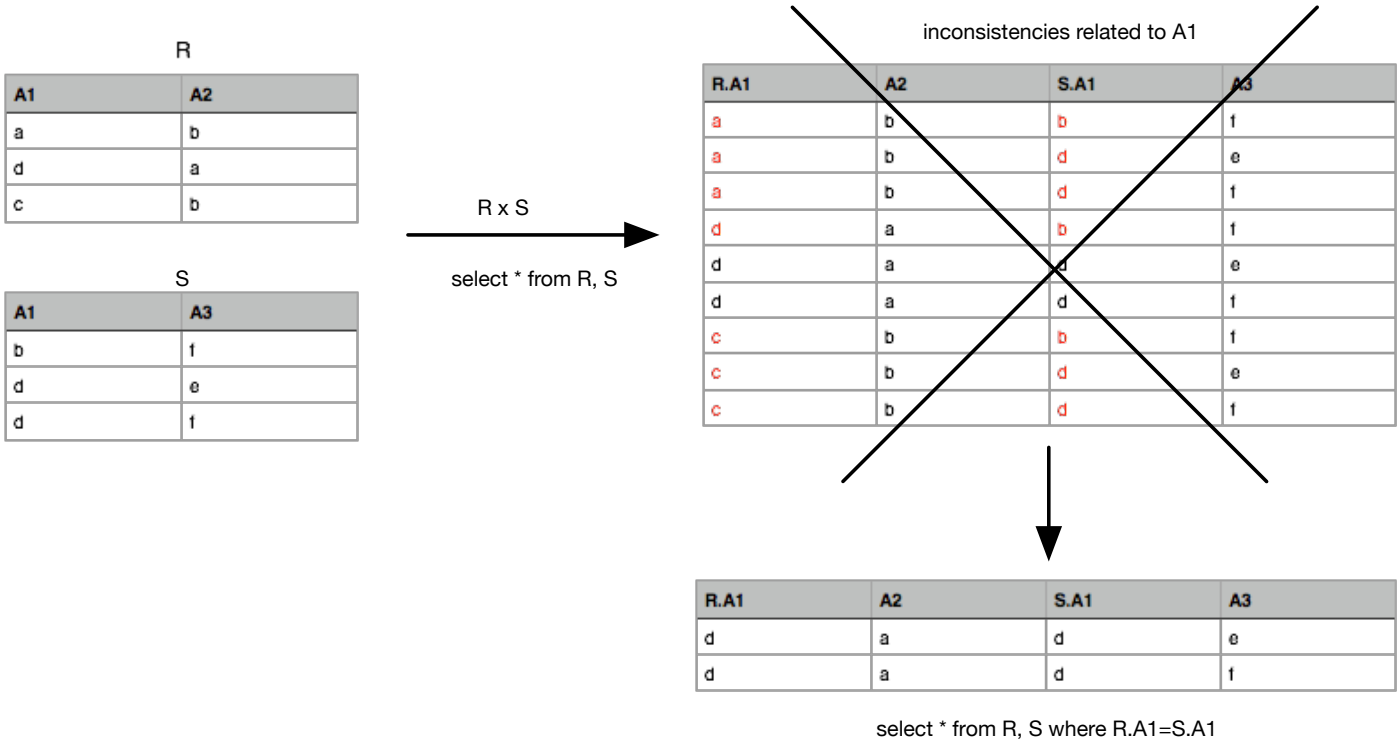
Concerning the table 'Windmill', a *primary key* has been defined. It corresponds to the attribute (column) 'id'. It is thus specified that each tuple (row) will be identified by a unique number 'id'.
Concerning the table 'WindmillData', a *primary key* has been been defined but here it corresponds to several attributes (columns): it means that the tuple (windmillIDREF, monthInYear, dayInMonth, hourInDay) has to be unique for each row.
In addition, a foreign key has been defined. It means literally that each value used for the attribute 'windmillIDREF' in table 'WindmillData' must correspond to an existing value 'id' in table 'Windmill'. If 'windmillIDREF' value is inserted and if this value is not present in the 'id' column in table 'Windmill', an error appears.
A request covering these 2 tables is given by:

```
SELECT name, windmillIDREF, monthInYear, dayInMonth, hourInDay, energy ...
    ... FROM Windmill, WindmillData ...
    ... WHERE id=windmillIDREF AND name='1312' AND monthInYear='1';
```

Note the compulsory restriction 'id=windmillIDREF' that represents the link between the two tables.
SELECT ... **FROM** Windmill, WindmillData ...; stands for a cartesian product. It results in a table with the following colunns: Windmill.id, Windmill.name, WindmillData.windmillIDREF, WindmillData.monthInYear, WindmillData.dayInMonth, WindmillData.hourInDay. Because there are no ambiguities, table prefixes can be removed: id, name, windmillIDREF, monthInYear, dayInMonth, hourInDay.
Only some columns are selected by:

```
SELECT name, windmillIDREF, monthInYear, dayInMonth, hourInDay, energy FROM ...;
```

Nevertheless, a restriction is necessary. Indeed, consider the following example:



A restriction (row selection) is required to keep consistencies in resulting tables when they are common variables, in particular, commun keys.
Regarding the previous example, the link is done tanks to:

```
SELECT name, windmillIDREF, monthInYear, dayInMonth, hourInDay, energy FROM Windmill, ...
    ... WindmillData WHERE id=windmillIDREF ...;
```

Additional row selection (restrictions) are carried out with ... **AND** name='1312'**AND** monthInYear='1';

Question 4: Why choosing the tuple (windmillIDREF, monthInYear, dayInMonth, hourInDay) as primary key for table 'WindmillData'? Is there an alternative?

In the UML diagram, an index appears. It has been defined by:

```
CREATE INDEX WindmillTimeData ON WindmillData ...
    (monthInYear ASC, dayInMonth ASC, hourInDay ASC)
```

This index will speed up searches concerning 'monthInYear', 'dayInMonth' and 'hourInDay' thanks to the creating of internal hash table (see `https://en.wikipedia.org/wiki/Hash_table`). It has been defined because there is a big quantity of timed data.

Question 5: The database in file 'system.sqlite3' can be accessed using the application 'SQLite Database Browser'[16]. Using 'SQLite Database Browser', analyze the structure of the database. Justify the use of indices.

Question 6: Using the application 'SQLite Database Browser', create 2 SQL requests to get, for a given day in a month: (1) the hourly energy production for a given windmill defined by its name (2) the hourly energy consumption for a given consumer defined by its name.

2.3. Data Control Language

The GRANT command can yield access rights to a user in Internet database system. REVOKE removes access rights.

[16] `http://sqlitebrowser.org`

Data control language contains also a mean to manage concurrent accesses thanks to transactions. It amounts to set a lock to block access to some tables for others users, until the transaction is finished. If the transaction cannot succeed, a rollback mechanism restore the initial state.

3. Java and SQLite database

The problem data have been stored in a SQLite database, a SQL database without Internet access, because it does not require specific rights for firewall filters on your computer. A SQLite database is contained in a file.
The database in file 'system.sqlite3' can also be accessed from Java. The Java driver for SQLite has to be downloaded from `https://bitbucket.org/xerial/sqlite-jdbc/downloads` and added into your project library. In order to open the database in file 'system.sqlite3' from Java, do (see file 'SQLiteConnector1.zip'):

```
import java.sql.*;

public class SQLliteConnector {
    public static void main( String args[] ) {
        Connection c = null;
        try {
            Class.forName("org.sqlite.JDBC");
            c = DriverManager.getConnection("jdbc:sqlite:system.sqlite3");
        } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
        }
        System.out.println("Opened database successfully");
    }
}
```

In order to read the database from Java, use for instance (see file 'SQLiteConnector2.zip'):

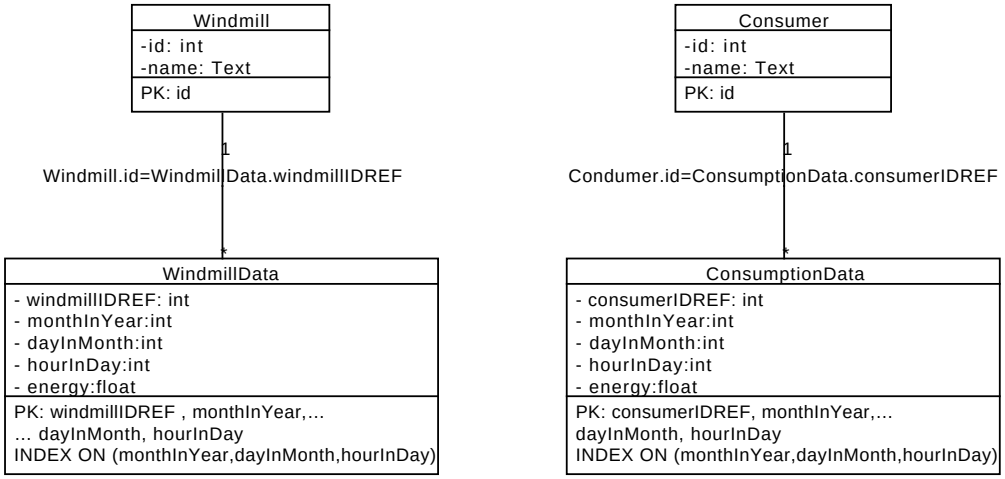
```
import java.sql.*;

public class SQLliteConnector {
    public static void main( String args[] ) {
        Connection connection = null;
        Statement statement = null;
        try {
            Class.forName("org.sqlite.JDBC");
            connection = DriverManager.getConnection("jdbc:sqlite:system.sqlite3");
        } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
        }
        System.out.println("Opened database successfully");
        try {
            statement = connection.createStatement();
            ResultSet rs = statement.executeQuery(
                "SELECT name, windmillIDREF, monthInYear, dayInMonth, hourInDay, energy
                FROM Windmill, WindmillData WHERE id=windmillIDREF AND name='1312'
                AND monthInYear='1';" );
            while ( rs.next() ) {
                String  name = rs.getString("name");
                int monthInYear  = rs.getInt("monthInYear");
                int dayInMonth  = rs.getInt("dayInMonth");
                int hourInDay  = rs.getInt("hourInDay");
                float energy = rs.getFloat("energy");
                System.out.println(name+","+monthInYear+","+dayInMonth+","+
                    hourInDay+": "+energy+"MWh");
            }
            rs.close();
```

```
            statement.close();
            connection.close();
        } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
        }
        System.out.println("Operation done successfully");
    }
}
```

4. Available data

The structure of the SQLite database 'system.sqlite3' is given by:



The data cover a full year either for Windmill energy production in MW and for energy consumption in MW. Note that consumers here stand for a large sets of users.

Question 7: Implement SQL statements of the previous question into Java (see 'SQLConnector3.zip').

Question 8: Using JFreeChart and the database, plot the energies related to consumer C2000903 and to Windmill 143, for January 5th and analyze them.

Clue for question 8: To plot the data, use the JFreeChart (jfreechart-1.0.9.jar) and Jcommon (jcommon-1.0.12.jar): see 'Plotting.zip' and documentation 'jfreechart-install.pdf' and 'jfreechart-manual.pdf'.

Part III
Web Services

There are many different ways to exchange data through the Internet. Nevertheless, different requirements have to be taken into account:

- interoperability: information has to be easy to extract. As a consequence, XML standard has become widely used and information that can be found inside a message are depicted;
- security: for the sake of security, in most organizations, most ports of the IP protocole are filtered, except the port 80 because it is the port used by the Web;
- robustness: unlike UDP[17], TCP contains robust standard error control mechanisms

According to these requirements, information exchanges tend gradually to use the HTTP protocole using XML data format and port 80.
Two kinds of web services are widely used:

- the RESTful (REpresentational State Transfer) web services (see `http://en.wikipedia.org/wiki/Representational_state_transfer`), which is inspired from the web and, in particular, from the HTTP protocole (see `http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol`);
- the SOAP web services (see `SimpleObjectAccessProtocol`), which is an XML based remote procedure call (RPC)[18].

5. RESTful web services

We are going to use RESTful web services (see documentation at `https://jersey.java.net/nonav/documentation/1.12/index.html`) because they are easier to use. Indeed, to test such a web service, a browser is enough. A RESTful web service is organized like the WEB i.e. everything is a resource defined by an Universal Resource Locator (for instance, `http://www.mysite.org/myappli/myobject/myfun/myparam1/myparam2`): an object, a method and even a parameter.

Assume the request:
`http://www.mysite.org/myappli/myobject/myfun/myparam1/myparam2`
is sent i.e. the following HTTP message is sent:

```
GET /myappli/myobject/myfun/myparam1/myparam2 HTTP/1.1
Host: www.mysite.org
```

The following reply could come:

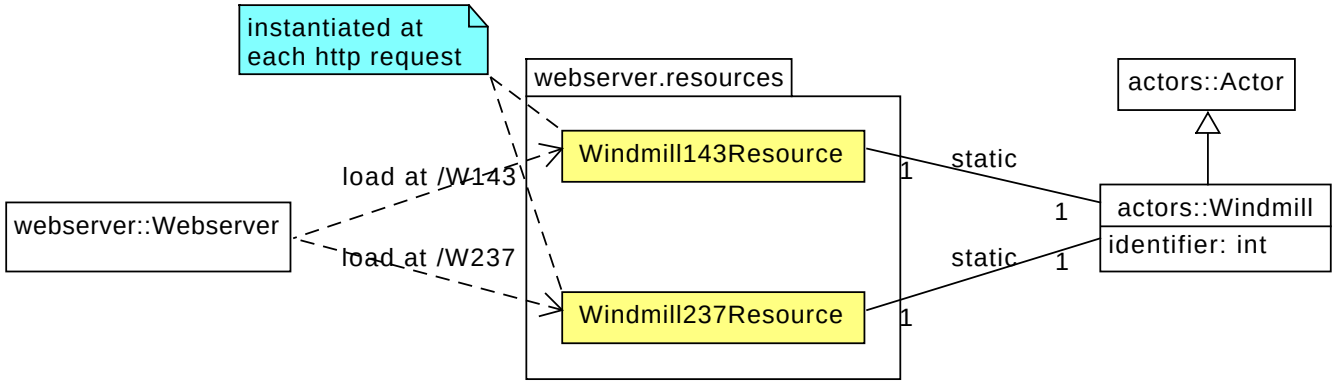
```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/xml; charset=UTF-8
Content-Length: 122
Accept-Ranges: bytes
Connection: close
```

[17] UDP protocole may also contain error control mechanisms but they have to be developed specifically. It is therefore not standardized.

[18] like Java Remote Method Invocation except that it is not based on HTTP and XML

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
    <data>12.3</data>
    <data>18.9</data>
    <data>2.4</data>
</result>
```

In order to keep it simple, we are going to use Jersey implementation of the JAX-RS JSR-311 entitled "Java API for RESTful Web Services". It is an easy way to manage RESTful web services.
Let's illustrate the power of this technology with the following example which create a web object: 'helloworld', i.e. an object that can be published and accessed through the Web. The object is given in 'basic-restful.zip'[19].
The proposed architecture for the smart grid is given by:



Such an architecture models what could be in reality except that the webservices would not be gathered in the same place but distributed in actors' places.

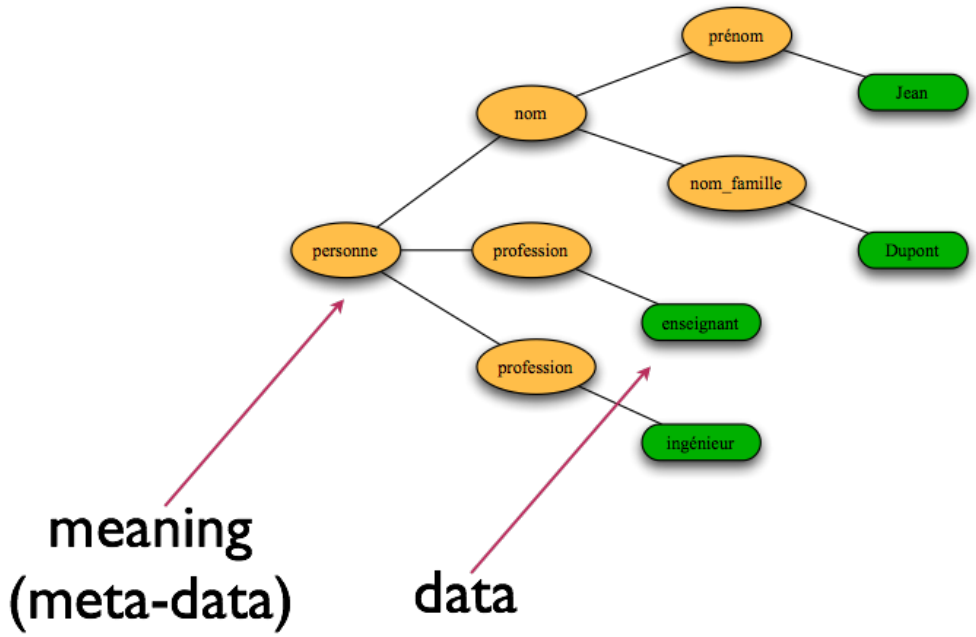
6. XML syntax

XML (see `http://en.wikipedia.org/wiki/Extensible_Markup_Language`) is a markup language that relies on tags to indicate metadata i.e. to give information about the data. Thanks to the tags, it is much easier to retrieve data in a text message. Consider, for instance, the following message:

```
<?xml version="1.0" encoding="UTF-8"?>
<personne>
        <nom>
                <prenom>Jean</prenom>
                <nom_famille>Dupont</nom_famille>
        </nom>
        <profession>enseignant</profession>
        <profession>ingenieur</profession>
</personne>
```

Thanks to tags, it is easy to understand that 'Jean' is a given name and that he is a teacher. Note that couple of identical tags are never imbricated: they have to follow a strict tree architecture:

[19] Do not forget to load the libraries 'asm-3.1.jar', 'jersey-core-1.12.jar', 'jsr311-api-1.1.1.jar', 'jersey-server-1.12.jar', 'jersey-client-1.12.jar'.



XML document can contain many kinds of nodes:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- '''Commentaire''' -->
  <element-document xmlns="http://exemple.org/" xml:lang=";fr">
    <element attribut="valeur">Texte</element>
    <element>élément répété</element>
    <element>
      <element><![CDATA[<ceci> est un noeud]]></element>
    </element>
    <element>Texte avec<element>un élément</element>inclus</element>
    <element/><!-- élément vide -->
    <element attribut="valeur"></element>
  </element-document>
<?xml-stylesheet href="transform.xsl" type="text/xsl"?>
```

comment node
root node or document
text node
escape node
element node
attribute node
instruction node

To extract the data from an XML document, there exists different standardized parsers. The main ones follow either a tree based approach with the DOM model (see `http://en.wikipedia.org/wiki/Document_Object_Model`) or an event driven approach with the SAX model (see `http://en.wikipedia.org/wiki/Simple_API_for_XML`). Because, the DOM approach is the easiest to learn, we are going to use it.

Question 9: Open and analyze the Java project 'windmill-restful.zip'. It consists of web-services related to windmills 143 and 237.
Assuming that day ahead production predictions correspond to data available in the SQLite database 'system.sqlite3', modify the project for webservices to provide the 24 values corresponding to the predictions of production for the next day. These values can be read from the database (currently, values are randomly generated; see webserver.actors.Windmill.java).

Question 10: Modify 'SimulationController.java' to decode the XML replies from Windmills, Resellers and Aggregators in order to get each set of predictions as a table of double values.

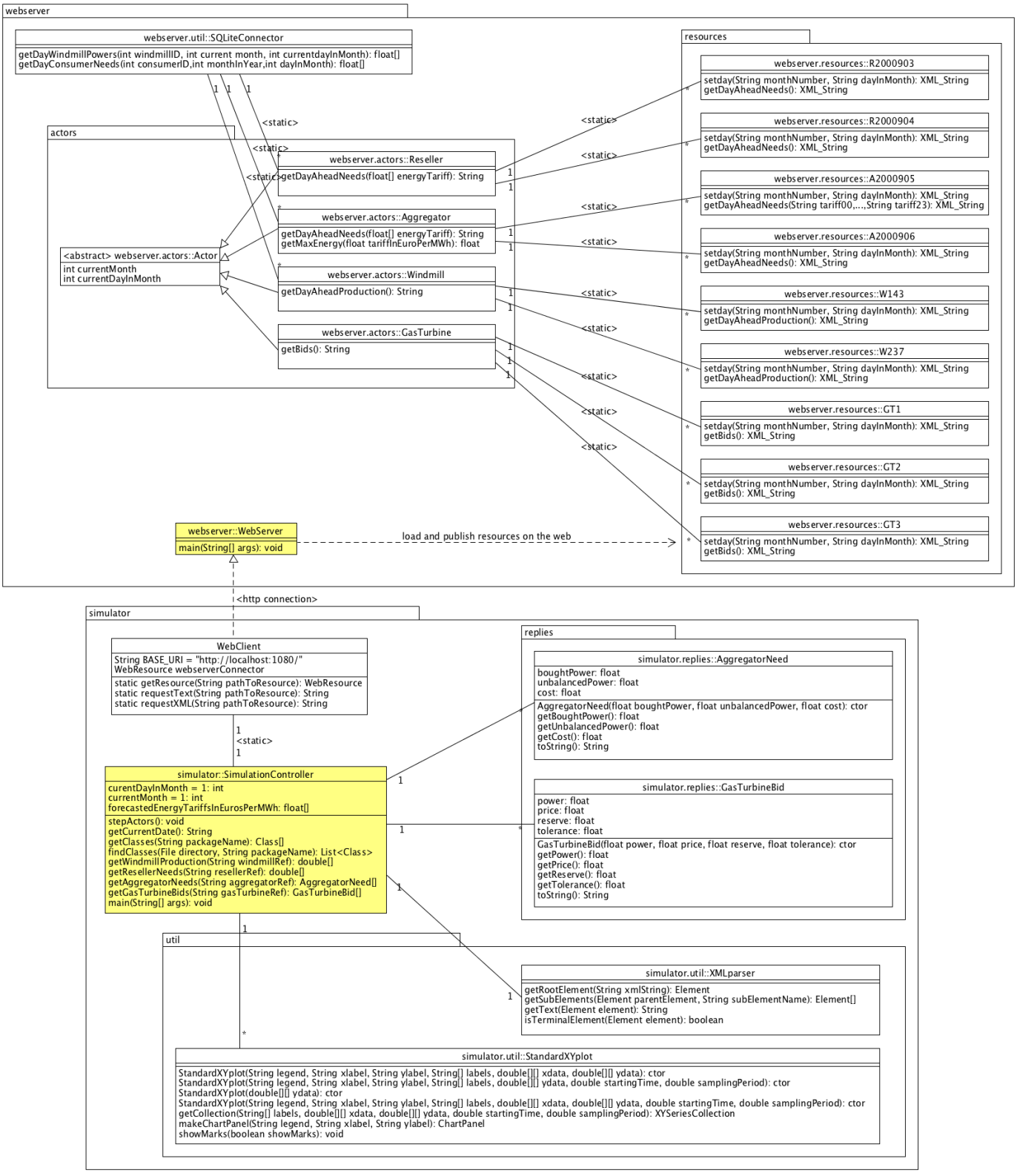
Clue for question 10: Use 'XMLparser.java' provided into the xml package of the project 'windmill-restful.zip'.

Question 11: In the same way, develop resellers 'R2000903' and 'R2000904' as well as resellers/aggregators 'A2000905' and 'A2000906' as big consumers providing their energy needs according to what is in the database, table *ConsumptionData*. 'R200090X' or 'A200090X' correspond to consumer '200090X'.

Part IV

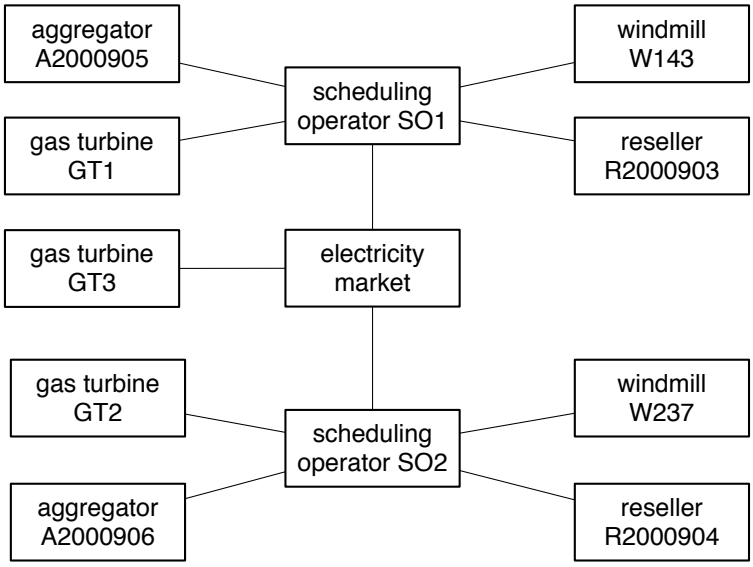
Simulation of interactions between micro-grid actors

A project architecture is proposed to help you (file "distrib.zip") but you can follow your own way. The following UML diagram represents the proposed project architecture:



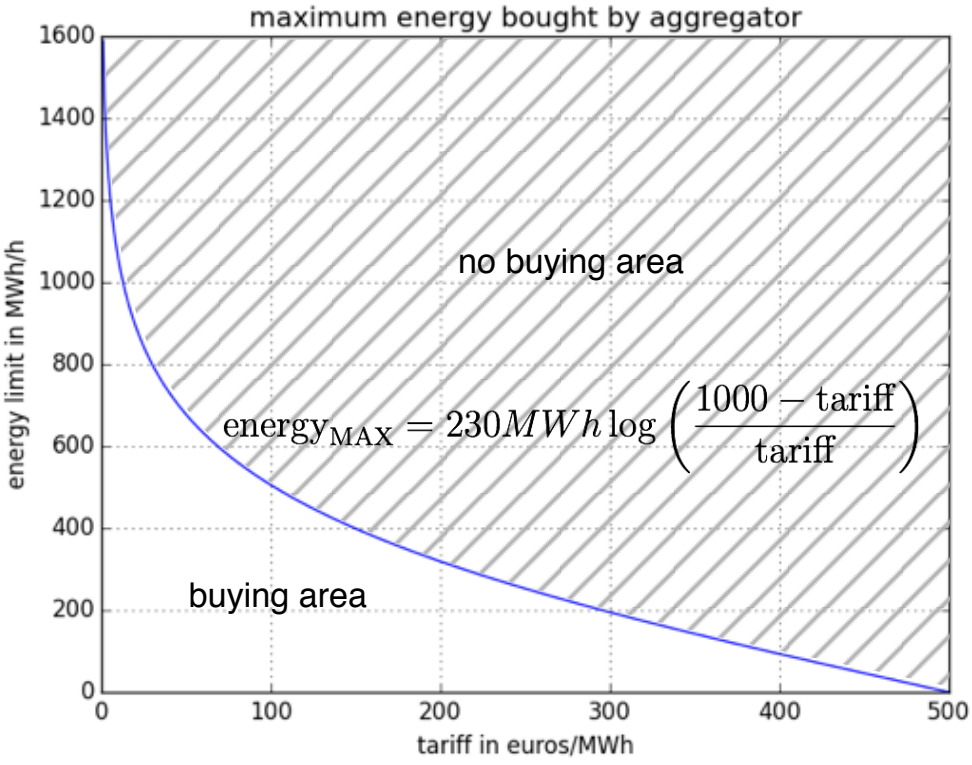
7. Micro-grid under consideration

Consider the micro-grid represented in:



7.1. Aggregators

Thanks to the 2 hours expensive tariff, an aggregator can modulate its overall customer consumption. Consider the overall modulation is given by the following utility function:



Question 12: Compute the energy needs for each aggregator when the forecasted energy tariff is:

hour	0	1	2	3	4	5	6	7	8	9	10	11
euros/MWh	60	60	60	60	60	60	60	300	400	400	200	200
hour	12	13	14	15	16	17	18	19	20	21	22	23
euros/MWh	300	200	100	60	100	200	300	400	400	200	100	60

Determine the costs and what will be unbalanced each hour.

7.2. Gas turbines

The gas turbines offer the following exclusive[20] bids:

GT1	bid1	bid2
power	50MWh	100MWh
price	10000 euros	15000 euros
reserve	25MWh	75MWh
tolerance	20MWh	20MWh

GT2	bid1	bid2
power	150MWh	250MWh
price	22000 euros	32000 euros
reserve	75MWh	200MWh
tolerance	20MWh	20MWh

GT3	bid1	bid2
power	300MWh	500MWh
price	34000 euros	50000 euros
reserve	150MWh	400MWh
tolerance	20MWh	20MWh

To accept a bid, the reserve power must be consumed at least. The power of a bid can vary within the range defined by the tolerance (+/-) without affecting the price.

Question 13: Develop actors gas turbines GT1, GT2 and GT3.

7.3. Scheduling operators

Question 14: Develop the *scheduling operators* and play the use case *equilibrate consumption and production*. Determine where the energy need is higher that production and conversely where production is higher that needs. Then, determine the local energy prices per hour considering.

Question 15: (Optional) Add the electricity market and simulate a possible overall behaviour. A bid on the market must be 20MWh minimum an hour. If a need is below that value, it will be solved on another market (adjustment market).

[20] One has to choose at most 1 bid per gas turbine.