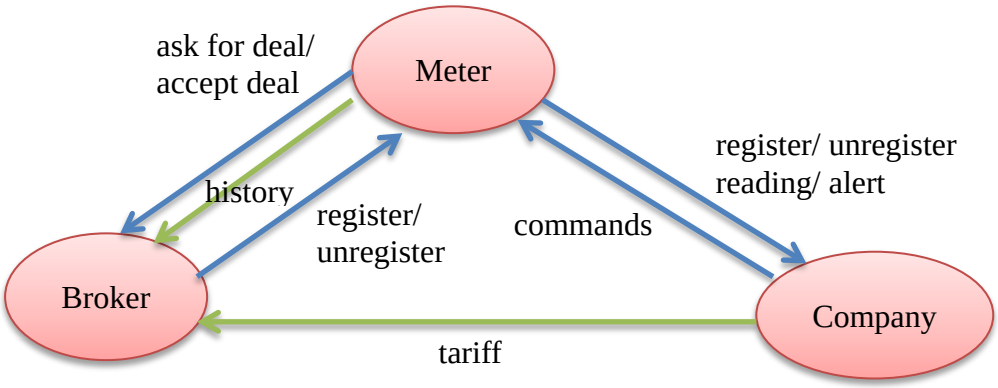
Report of COMP2207 Main Coursework
A smart metering system using Java RMI

Student Name: Kuan-Yu Li Student ID: 27440656 Email: kl1u13@soton.ac.uk

1. Introduction and overview of the design

- Relationships between three servers
  Blue lines = pushed messages; Green lines = pulled messages.



- A group of three objects represents a broker/company/meter
  - Broker.java, BrokerInterface.java, and BrokerServer.java are used to represent a broker.
  - Meter.java, MeterInterface.java, and MeterServer.java are used to represent a meter.
  - Company.java, CompanyInterfaca.java, and CompanyServer.java are used to represent a company.
  - Take a meter as an example, MeterInterface.java lists the functions that can be called by other objects. Meter.java implements all these functions and also keeps some variables to store data. MeterServer.java will be run on the machine. A user can directly interact with a meter through MeterServer.java.

- Design of a server

It consists of 2 main parts:

  - First part is about naming. You can input the name of the server, and it will bind the name to its remote object. After this step, other objects can find this object by looking up the correct url(e.g. rmi://localhost/meter_name).
  - Second part is an infinite while loop that keeps asking the user what to do next. The user can input a number to choose from the available actions, such as registering a company, sending an alert, etc.

- How the system works
  - One important assumption I made is that the broker is omniscient. It knows all existing meters and companies, and every meter and company also know the broker and its url. So the

BrokerServer must be the first to be started. After it is started, it will create a Broker object and then go to the infinite loop.

- Then you can start a CompanyServer and a MeterServer. When started, they will create corresponding objects and ask the user to name these remote objects. After finishing naming, they will inform the broker about their names and remote object urls. Finally they will go to the infinite loop, waiting for commands from the user.

- To run the system on a Linux machine, use the following commands
  - rmiregistry [port] & - Start rmi object registry service on the machine. You can specify the port number. By default, it is 1099.
  - javac – Use javac to compile all java codes and generate class files.
  - java BrokerServer – BrokerServer must be started first.
  - java MeterServer / java CompanyServer – Then you can start MeterServer and CompanyServer as many as you want in any order.

```
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> rmiregistry &
[1] 2849
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac Broker.java
Note: Broker.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac BrokerInterface.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac BrokerServer.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac Company.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac CompanyInterface.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac CompanyServer.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac Meter.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac MeterInterface.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code> javac MeterServer.java
fishlinghu@linux-t2xy:~/Dropbox/Lectures_2014fall/Distributed Systems and Networ
ks/exercise/Main Coursework/code>
```

2. How a Meter changes Power Company

- Register with a company by the meter itself
  - When the user gives the command of registering with a company, the meter first takes the information (i.e. names and urls) of all companies from the broker and lists the names of companies.
  - Then the user can choose one company, and the meter can directly register with it.
  - If the meter is already registered with a company, it will unregister with the old one automatically before registering with the new company.

```
Successfully register with HTC!
Register successfully!

What do you want to do?
1. Register with a new company.
2. Unregister with the company.
3. Request for a better deal.
4. Accept the provided deal.
5. Send alert to the company.
6. Send reading to the company.
7. Set reading value.
8. Which company is the meter registered to?
1
Following are all the companies:
Company name: HTC
==================================================
Company name: Sony
==================================================
Which company do you want to register with? Enter the name: Sony

Unregister with the old company: HTC
Unregister successfully!
Successfully register with Sony!
Register successfully!
```

↑ A meter can only register with one company at the same time. So every time it registers with a new company, it unregisters with the old one first.

- Register with a company through the broker
  - When the user gives the command of requesting for a better deal, the meter first asks the broker for a better deal.
  - Then the broker will ask every company for their tariff and pick the best one for the meter.
  - After telling the meter about the deal, the broker doesn't wait for the meter to accept it. Instead, it continues to serve other meters. The meter can accept or refuse the deal at any time.
  - If the meter decides to accept it, it sends a message to the broker. Then the broker will help the meter to register with the new company.

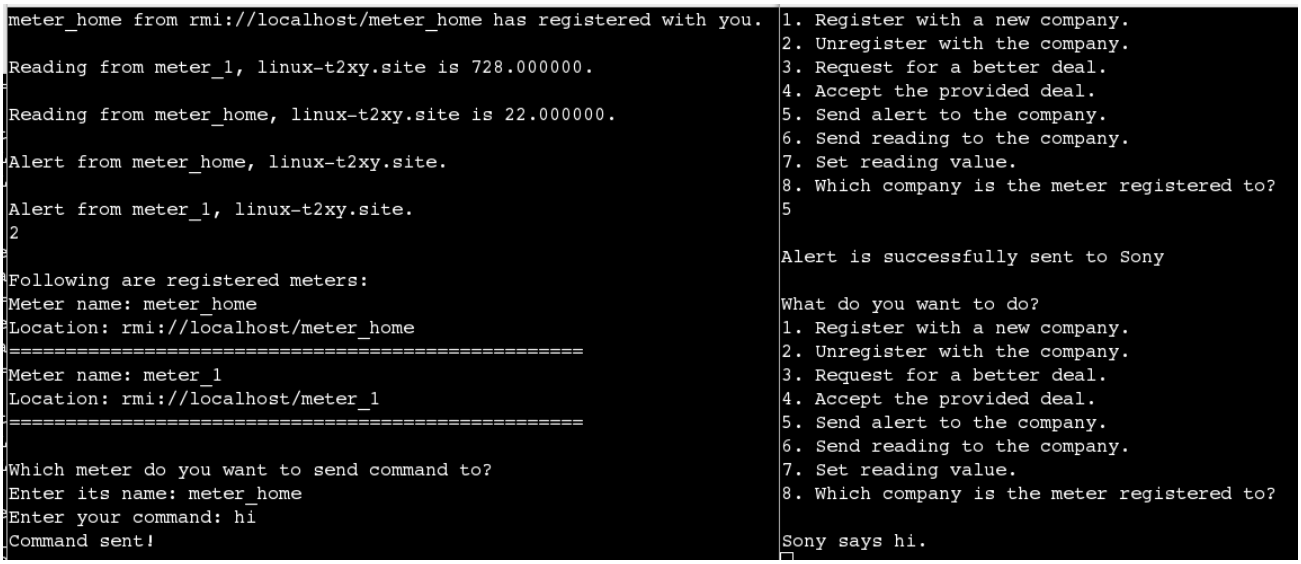
```
2. List all the meters.
2

Meter name: meter_home
Location: rmi://localhost/meter_home
==================================================
Meter name: meter_1
Location: rmi://localhost/meter_1
==================================================

What do you want to do?
1. List all the companies.
2. List all the meters.

Receive the request for a new deal from meter_home, linux-t2xy.site. Wait for me
ter_home's history of readings.
Receive the readings: -1.000000.

Sony's tariff: 37.000000.

HTC's tariff: 55.000000.

meter_home from rmi://localhost/meter_home accpets the new deal.
Successfully register meter_home with the new company: Sony.
```

```
7. Set reading value.
8. Which company is the meter registered to?
3
Successfully sent request to the broker for new deal.

Broker requests for the history of readings.
Sent the history of readings to the Broker.
Broker found a new deal: Sony's tariff: 37.0.

What do you want to do?
1. Register with a new company.
2. Unregister with the company.
3. Request for a better deal.
4. Accept the provided deal.
5. Send alert to the company.
6. Send reading to the company.
7. Set reading value.
8. Which company is the meter registered to?
4
Do you accept the deal: Sony's tariff: 37.0 ? (Y/N): y

Unregister with the old company: Sony
Unregister successfully!
Successfully register with Sony!
```

↑ The meter (right) requests for a deal. Then the broker (left) request for the meter's reading. Then it returns a deal to the meter. If the meter accepts it (see the bottom-right part), the broker can help the meter to register with the new company.

3. How multiple remote objects connect and communicate to each other

- How one company connects to multiple meters
  - Every company object keeps a hash map of registered meters' names and urls. So it can directly send messages to all registered meters without the help of a broker. When a meter unregisters with the company, the meter's information will be removed from the hash map.
  - As for meters, every meter keeps the information of its own registered company. So it can also directly send messages to the company without the help of a broker.
  - Since I did not write the object as multi-threaded, all operations are carried out sequentially. Luckily, all operations in this project are quite simple. So there won't be much delay even if multiple meters interact with a same company at the same time.

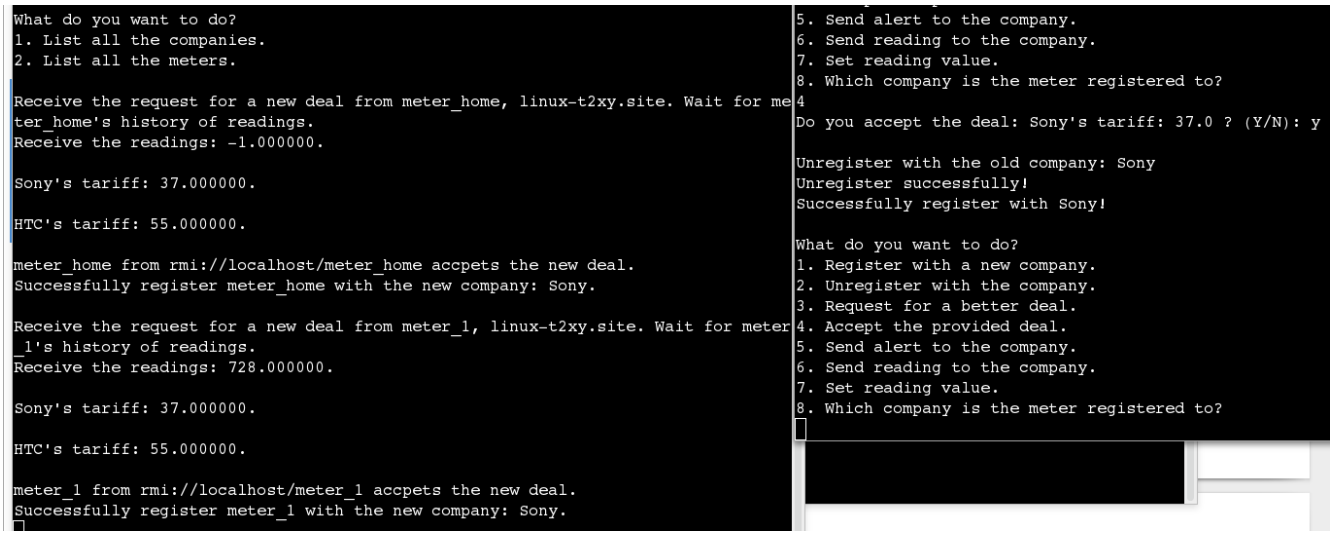


↑ You can see that the company (left) can receive messages from multiple meters, and can send messages to any registered meter.

- How one broker connects to multiple meters and companies

As I said before, the most important assumption of my system is that the broker knows about all existing companies and meters. To implement this, the broker must be started first. And every time a company or a meter starts, it has to inform the broker. Then the broker will store the new object's name and url in a hash map. With all objects' urls, the broker can connect to every existing company and meter. Although the broker is not required to have any operations initiated by the user, I implemented two operations to list all meters and companies. This is the proof that the broker knows about every meter and company.

←The broker knows about every meter and company.

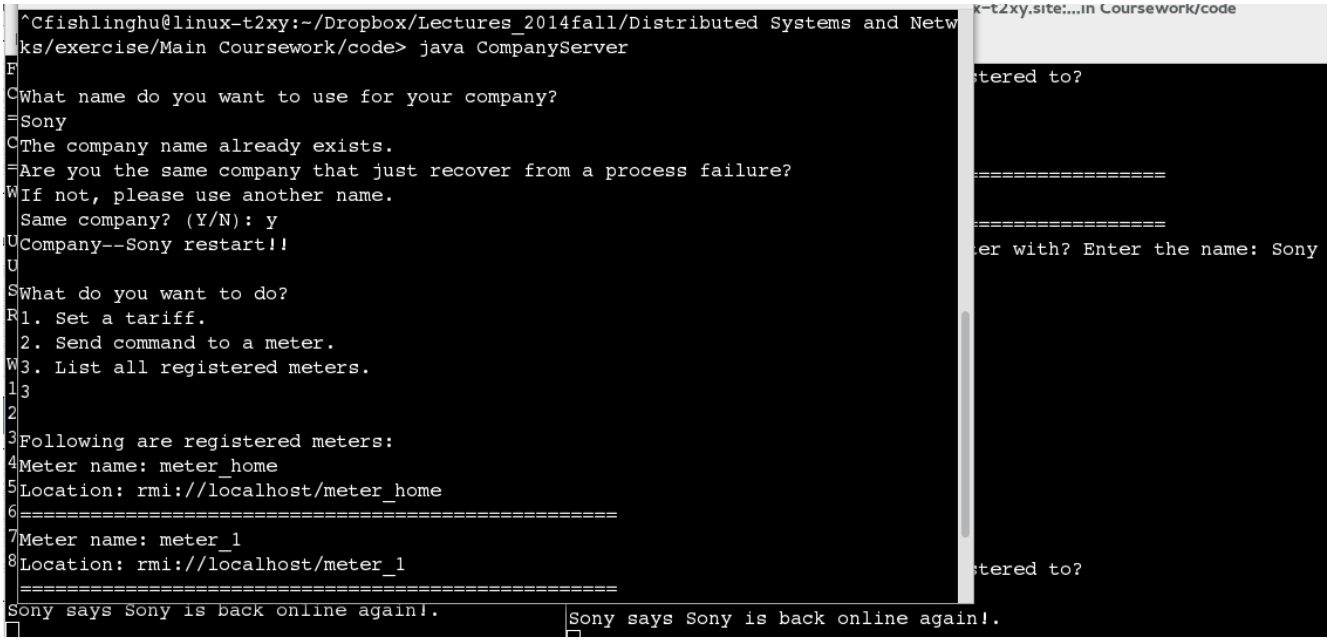


↑ The broker (left) can communicate with multiple companies to get their tariff information. And it can also help multiple meters to find a new deal.

4. How a connection lost is handled

- Restart Process

If a MeterServer process or CompanyServer process fails, you can restart the process and enter the same name you used before. The broker will then ask you if this is the meter/company that already exists. If you enter yes, the meter/company will retrieve all old data from the broker and broadcast to all the meters/companies that registered with it before, saying that it is back online. Then the system can work as before. This is how the system deals with a process failure.



↑ You can see from the picture that a company process can restart, gain the old data, then broadcast to all registered meters.

- Connection Lost

I use a lot of try-catch to deal with the connection lost. If a meter sends messages to a company and detect that it can't connect to the company object, then the user can decide whether to unregister with the company or not. If a company detects connection lost with a meter, it will jusr ignore it, since a company can't unregister a meter directly.

5. How the system be implemented on several machines

- How to modify

To implement the system on several machines, just modify the url (rmi://localhost/) used for local host to (rmi://machine's real IP/). When running servers on different machines, the objects will be bind to this kind of urls. Then all objects can communicate with each other successfully.

- Where to modify
  - You can just search for the comment “modify the url here” in every file.
  - In Broker.java, the two functions that used to add the name and url of a meter/company to the hash map.
  - In Meter.java, the meter object constructor that look-up the remote broker object.
  - In MeterServer.java, the main function that look-up the remote broker object and generate the url for the meter object.
  - In Company.java, the company object constructor that look-up the remote broker object.
  - In CompanyServer.java, the main function that look-up the remote broker object and generate the url for the company object.

6. Conclusions and critical evaluation

- Evaluation of my design
  - Pros:
    - The system I implement successfully meets the requirements and restriction of the Smart Metering System.
    - I write the codes in a strongly object-oriented way. There are not many codes in ObjectServer.java. Every function is written in Object.java. This made my codes easy to be debugged and understood.
    - The interface is also very user-friendly, which makes the system easy to be understood and tested. There are a lot of instructions telling the user what to input and what options to choose from.
    - To make it a robust system, I use many if-else in my codes, and I test it for many times. So an illegal input won't make the system crash.
    - The system can tolerate process failures. And once the process restarts, it can get back all the lost data.
  - Cons:
    - The system recovery heavily relies on information stored in the broker object. So the BrokerServer can't be shut down.
    - The system is not written in a multi-threaded way.

- Evaluation of RMI implementation for the system

  I think RMI is an appropriate implementation for the system. It is very easy for objects to access a remote object. Once the remote object is successfully referenced, you can just treat it as a local object and invoke its functions freely. The only problem for RMI is that it is not easy to detect a connection lost instantly. You have to reference the object then find that it is out of connection.

7. Environment information

- java –version
  - openjdk version "1.8.0_40"
  - OpenJDK Runtime Environment (build 1.8.0_40-b10)
  - OpenJDK 64-Bit Server VM (build 25.40-b14, mixed mode)

- javac –version
  - javac 1.8.0_25

- Operating system
  - OpenSUSE 13.2 (Linux)