JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

DEZSYS-L06 RMI

Systemtechnik Labor
4CHIT 2016/17

Marvin Ertl

Version 1.0
Note:
Betreuer:
Begonnen am 16. Juni 2017
Beendet am 23. Juni 2017

Inhaltsverzeichnis

1 Einführung

Diese Übung soll die Funktionsweise und Implementierung einer verteilten Applikation mit Hilfe von Java Remote Message Invocation (RMI) demonstrieren. Java Method Invocation (RMI) ist neben InterProcessCommunication (IPC), Message Oriented Middleware (MOM) und Remote Procedure Call (RPC) eine weitere Moeglichkeit um eine Kommunikation zwischen mehreren Rechnern umzusetzen.

1.1 Ziele

Das Ziel dieser Übung ist die Implementierung einer zentrale Steuerungsplattform von Windkraftanlagen der Windparks unter Verwendung von Java Remote Method Invocation (RMI). Das Beispiel baut auf den Beispiel DEZSYS-L04 "Datenuebertragung von Windkraftanlagen zum Parkrechner via XML-RPCünd DEZSYS-L05 "Nachrichtenbasierte Kommunikation (Message Queues) in Windparksäuf. Hier wird nun ein System entwickelt, dass es der Zentrale ermoeglicht Kommandos an die Windkraftanlagen der einzelnen Windparks zu senden und deren Bestaetigung zu empfangen. Mit Hilfe dieser direkten Steuerung kann auf Veraenderung der Umwelteinfluesse bzw. des Energiebedarfs des Stromnetzes von zentraler Stelle reagiert werden.

1.2 Voraussetzungen

- Grundlagen Architektur von verteilten Systemen
- Grundlagen zu Java Remote Method Invocation
- Verwendung der Architektur der Aufgabenstellungen DEZSYS-L04 und DEZSYS-L05
- Kennenlernen von Java RMI anhand des folgenden Tutorials "RMI Tutorial mit DateServer Beispiel"
- Verwendung der RemoteDate.jar Java Applikation als Startpunkt fuer diese Aufgabenstellung

1.3 Aufgabenstellung

Implementieren Sie das zentrale Windpark-Steuerungssystem mit Hilfe von verteilten Objekten. Die Applikation soll folgende Windkraftanlagen Kommandos umsetzen:

- Blattwinkel anpassen
- Gondelwinkel (Azimut) verstellen
- Blindleistung anpassen
- Wirkleistung anpassen

- Abschalten
- Starten

Die Applikation ist über das Netzwerk mit anderen Rechnern zu testen!

2 Ergebnisse

2.1 Starten der RMI Registry

Um RMI verwenden zu können muss der RMI Service gestartet werden. Um diese leichter starten zu können, kann der Pfad zur RMI Registry als Systemumgebungsvariable in Windows hinzugefügt werden.

Hierfür öffnet man über die Systemsteuerung im Unterpunkt System gibt es die Option Systemumgebungsvariablen bearbeitet. Dort kann man beim Path eine neue Systemvariable einfügen und zwar fügt man den Pfad zur JDK bin Ordner an.

Nun kann man die RMI Registry im Unterordner bin des Projektes starten, dies geschieht wie folgt:

```
1 rmiregistry.exe
```

Listing 1: Starten der RMI Registry

2.2 Design

Auf Basis des Designs der letzten Übungen wurde das Programm erweitert.

Das Windrad hat nun eine Implementation der RemoteControlImpl, diese wird über RMI gebinden. Mittels dieser Implementation stehen dem Zentralrechner dann einige Funktionen zu verfügen, unter anderem das Verändern der Attribute und das Auslesen einer Statusinformation. Die Windräder bekommen einen automatisch generierten Namen, dabei wird eine statische Variable beim Erzeugen jedes Windrades um eins erhöht.

Der Zentralrechner verwaltet in einer Hashmap die Verbindungen zu den gebindeten Windräder, als Key wird der Name der Windräder verwendet. Beim Start werden alle in der Registry gebindeten Namen ausgelesen und versucht zu diesen zu verbinden. Nun kann man über die Konsole die Attribute der einzelnen Windräder verändert werden, sowie das Anzeigen aller Namen in der Registry und das Verbinden zu neuen RMI Einbindungen.

Nachfolgendes UML Diagramm visualisiert die beschriebene Struktur:

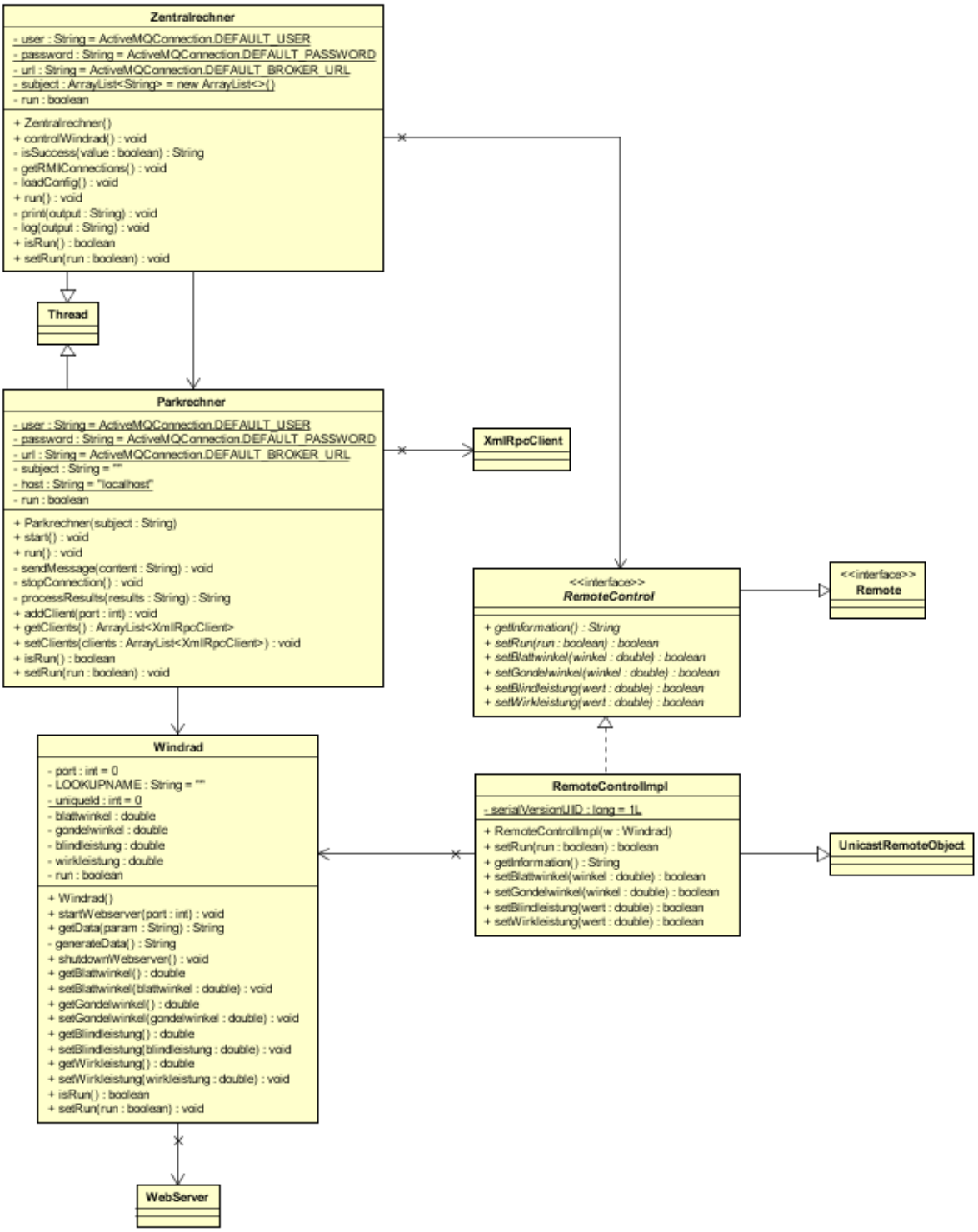

Abbildung 1: UML Diagramm des Design

2.3 Windrad

Das Windrad wurde um die gefragten Attribute erweitert und weiteres wird eine Attribute der RemoteControlImpl eingefügt. Diese ermöglicht dann den Zentralrechner die Attribute zu verändern.

Im Konstruktor wird dafür das Windrad in der Registry gebindetet. Dies geschieht wie folgt:

```
this.LOOKUPNAME = "w" + uniqueId++;
try {
    this.im = new RemoteControlImpl(this);
    Naming.rebind(this.LOOKUPNAME, this.im);
} catch (RemoteException e) {
    System.out.println("Error while creating RemoteControlImpl");
} catch (MalformedURLException e) {
    System.out.println("URL Exception");
}
```

Listing 2: Binden des Windrades in der Registry

2.3.1 Remote Control Implementation

Die Klasse RemoteControlImpl erbt von UnicastRemoteObject, dies wird benötigt damit die Klasse anschließend an die Registry gebindet werden kann, weiteres wird das Interface RemoteControl implementiert. Dies beinhaltet Methoden für das Ändern von allen Attributen.

```
public interface RemoteControl extends java.rmi.Remote {
    public String getInformation() throws java.rmi.RemoteException;
    public boolean setRun(boolean run) throws java.rmi.RemoteException;
    public boolean setBlattwinkel(double winkel) throws java.rmi.RemoteException;
    public boolean setGondelwinkel(double winkel) throws java.rmi.RemoteException;
    public boolean setBlindleistung(double wert) throws java.rmi.RemoteException;
    public boolean setWirkleistung(double wert) throws java.rmi.RemoteException;
}
```

Listing 3: Interface RemoteControl

Alle Methoden haben als Rückgabewert einen boolean Wert, dieser liefert den Zentralrechner zurück ob die Änderungen erfolgreich übernommen worden sind.

In der RemoteControlImpl Klasse werden diese Methoden nun realisiert nachfolgend ist eine Methode als Beispiel aufgeführt, die anderen Methoden sind nach den selben Prinzip implementiert.

```
@Override
public boolean setBlattwinkel(double winkel) throws RemoteException {
    this.w.setBlattwinkel(winkel);
    return true;
}
```

Listing 4: Realisierung der Methoden

2.4 Zentralrechner

Der Zentralrechner verwaltet in einer Hashmap die Verbindungen zu den gebindeten Client. Um alle Clients zu bekommen wird aus der Registry die Namen gelesen und anschließend versucht eine Verbindung herzustellen.

```
private void getRMIConnections() {
    try {
        Registry registry = LocateRegistry.getRegistry("localhost", 1099);
        for (String name : registry.list()) {
            this.netConn.put(name, (RemoteControl)Naming.lookup(name));
        }
    } catch (RemoteException e) {
        this.print("RemoteControl exception, check if RMI is running\nError message: " + e.getMessage())
            ;
    } catch (MalformedURLException | NotBoundException e) {
        this.print("Not bound exception");
    }
}
```

Listing 5: Herstellen der Verbindungen

2.4.1 Konsolen eingaben

Da nun die Verbindung zwischen den Zentralrechner und den Windrädern vorhanden ist, kann man über die Konsole diese ansteuern. Dabei wird in einer Endlosschleife der Benutzer abgefragt, die Eingabe wird als erstes nach jeden Leerzeichen geteilt und in einen Array gespeichert.

Nun wird das Array als erstes nach der Länge geprüft und anschließend die weiteren Attribute. Ein Array mit der Länge 1 kann unteranderem help oder exit sein.

```
System.out.print(">>>");
String temp = s.nextLine();
this.log(">>>" + temp);

String[] command = temp.split(" ");

if (command.length == 1) {
    if (command[0].equals("exit")) break;
    if (command[0].equals("help")) {
        this.print("Commands:\nshow names\tList all names\nupdate connections\tConnects to all given
            names\n[name of windrad] status\tList all attributes\n[name of windrad] [typ of settings] [
            value]\tSets the attribute of the windrad to given value\nexit\tExit the program");
    }
}
```

Listing 6: Verarbeiten von Input mit der Länge 1

Ein Input der Länge 2 kann unteranderem der Name eines Windrades plus status, dann werden alle Attribute des Windrades ausgegeben, oder show names, dadurch werden alle Namen von gebindeten Clients angezeigt. Unter mittels den Befehl update connections können neue gebindeten Clients hinzugefügt zu der Hashmap hinzugefügt werden.

```
if (command.length == 2) {
    try {
        if (command[1].equals("status")) this.print(this.netConn.get(command[0]).getInformation());
        if (command[0].equals("show") && command[1].equals("names")) {
            Registry registry = LocateRegistry.getRegistry("localhost", 1099);
            String[] boundNames = registry.list();
            for (String name : boundNames) {
                this.print(name);
            }
        }
        if (command[0].equals("update") && command[1].equals("connections")) this.getRMIConnections();
    } catch (RemoteException e) {
        this.print("Remote exception while trying to execute following commando: " + temp.toLowerCase())
            ;
    } catch (NullPointerException e) {
        this.print("Windrad with used name not existing");
```

```
    }
}
```

Listing 7: Verarbeiten von Input mit der Länge 2

Und mit der Eingabe eines Names plus ein Attribute und einen Wert können die Attribute der einzelnen Windräder verändert werden. Weiteres kann auch mittels den Befehl Namen plus turn off/on ein Windrad ein bzw. ausgeschaltet werden.

```
if (command.length == 3) {
    try {
        switch (command[1]) {
            case "turn":            if (command[2].equals("on")) {
                this.print("Returns: " + isSuccess(this.netConn.get(command[0]).setRun(true)));
            } else if (command[2].equals("off")) {
                this.print("Returns: " + isSuccess(this.netConn.get(command[0]).setRun(false)));
            } else {
                this.print("Syntax error");
            }
            break;
            case "blattwinkel":     try {
                this.print("Returns: " + isSuccess(this.netConn.get(command[0]).setBlattwinkel(Double.
                    parseDouble(command[2]))));
            } catch (NumberFormatException e) {
                this.print("Syntax error, not a number");
            }
            break;
            case "gondelwinkel":    try {
                this.print("Returns: " + isSuccess(this.netConn.get(command[0]).setGondelwinkel(Double.
                    parseDouble(command[2]))));
            } catch (NumberFormatException e) {
                this.print("Syntax error, not a number");
            }
            break;
            case "blindleistung":   try {
                this.print("Returns: " + isSuccess(this.netConn.get(command[0]).setBlindleistung(Double.
                    parseDouble(command[2]))));
            } catch (NumberFormatException e) {
                this.print("Syntax error, not a number");
            }
            break;
            case "wirkleistung":    try {
                this.print("Returns: " + isSuccess(this.netConn.get(command[0]).setWirkleistung(Double.
                    parseDouble(command[2]))));
            } catch (NumberFormatException e) {
                this.print("Syntax error, not a number");
            }
            break;
            default:            break;
        }
    } catch (RemoteException e) {
        this.print("Remote exception while trying to execute following commando: " + temp.toLowerCase())
            ;
    } catch (NullPointerException e) {
        this.print("Windrad with used name not existing");
    }
```

Listing 8: Verarbeiten von Input mit der Länge 3

2.4.2 Konsole Beispiel

Nachfolgend zeigt ein Beispiel genauer wie die Befehle verwendet werden können.

Als erstes wird mittels den Befehl show names alle verbundenen Windräder angezeigt. Dannach wird mittels den Namen w1 und status alle Informationen zu dem Windrad ausgegeben. Anschließend wird es mittels den Befehl w1 turn on eingeschaltet und die Wirkleistung wird mittels den

Befehl w1 wirkleistung 4.5 auf 4.5 gesetzt, dies sieht man, wenn anschließend wieder die Informationen aufgerufen werden.

```
----------Control Windrad----------
>>>show names
w0
w1
w2
>>>w1 status
Windrad Informationen:
Running: false
Blattwinkel: 66.41616376000691
Gondelwinkel: 7.743305173634979
Blindleistung: 0.3964245484749188
Wirkleistung: 2.368539410701005
>>>w1 turn on
Returns: Success
>>>w1 wirkleistung 4.5
Returns: Success
>>>w1 status
Windrad Informationen:
Running: true
Blattwinkel: 66.41616376000691
Gondelwinkel: 7.743305173634979
Blindleistung: 0.3964245484749188
Wirkleistung: 4.5
>>>
```

Listing 9: Beispiel der Steuerung eines Windrades

2.4.3 Logging der Konsole

Um das Logging zu vereinfachen wurde die Methode print geschrieben, diese wurde bereits oben im Code für das Verwalten der Konsoleneingaben verwendet. Die Methode print gibt ganz normal den Paramter über die Konsole aus, schreibt diesen aber gleichzeitig auch in eine Log Datei.

```
private void print(String output) {
    System.out.println(output);
    this.log(output);
}
```

Listing 10: Print Methode logt und ausgabe

Die Methode print ruft eine Methode log auf, diese Methode kümmert sich um das Schreiben in die Log Datei. Die Methode log überprüft als erstes ob das File vorhanden ist, wenn nicht wird die Log Datei erzeugt. Anschließend wird der Paramter am Ende der Log Datei eingefügt.

```
private void log(String output) {
    try {
        File f = new File("log.txt");
        if(!f.exists() && !f.isDirectory()) {
            f.createNewFile();
        }
        BufferedWriter writer = new BufferedWriter(new FileWriter("log.txt", true));
        writer.newLine();
        writer.append(output);
        writer.flush();
    } catch (IOException e) {
        System.out.println("Error by writing to log.txt");
    }
}
```

Listing 11: Schreiben in die log Datei

In der Log Datei ist nun der selbe Inhalt, wie der der in der Konsole eingegeben worden ist.

Listings

Abbildungsverzeichnis