tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

Systemtechnik Labor

4AHIT 2017/18

Message Oriented Middleware

Laborprotokoll

Marc Rousavy

3. Mai 2018

Bewertung:

Betreuer: Thomas Micheler

Version: 1.1

Begonnen: 26. April 2018

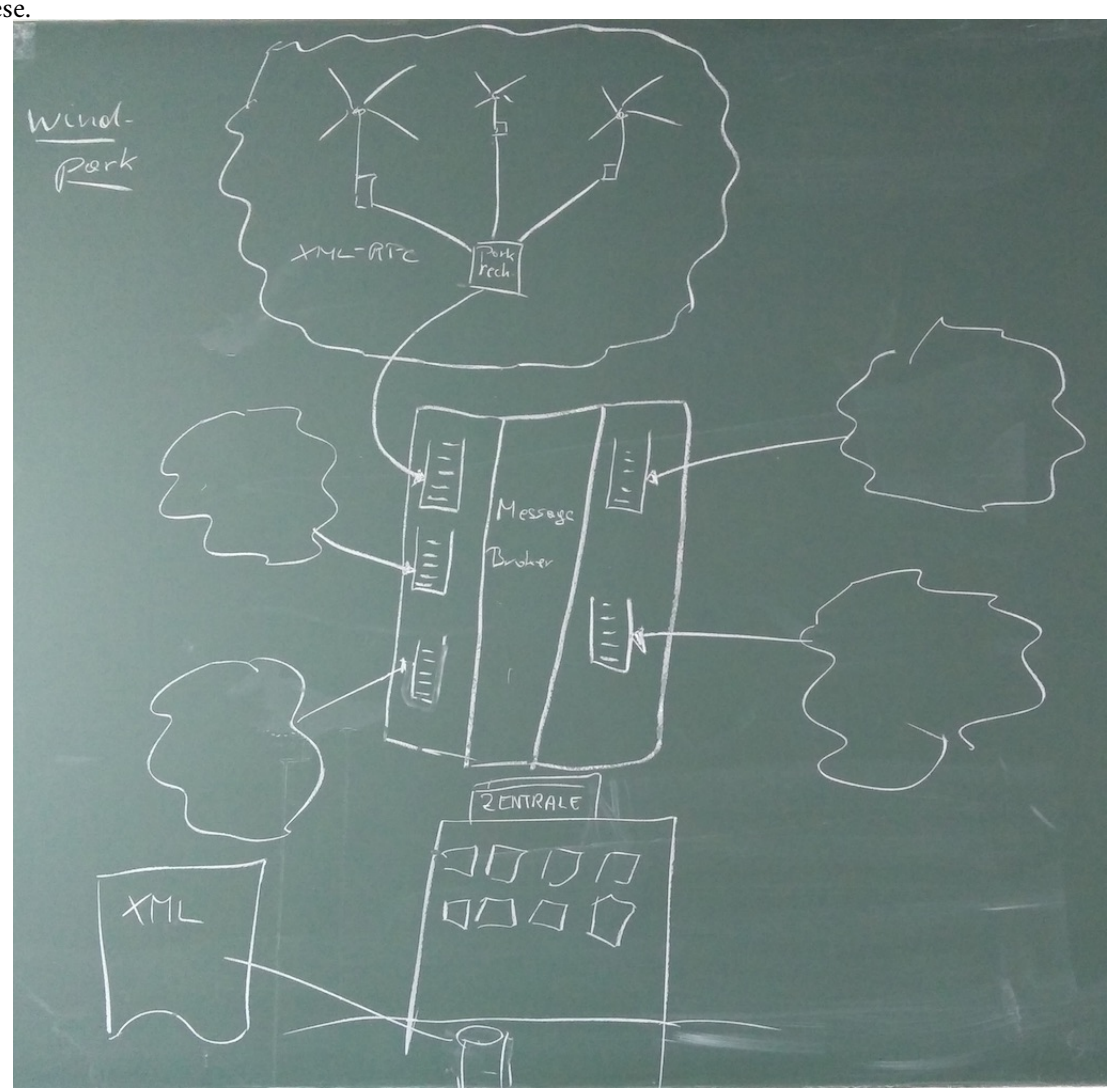
Beendet: 26. April 2018

Inhaltsverzeichnis

1 Einführung

Diese Übung soll die Funktionsweise und Implementierung von eine Message Oriented Middleware (MOM) mit Hilfe des Frameworks Apache Active MQ demonstrieren. Message Oriented Middleware (MOM) ist neben InterProcessCommunication (IPC), Remote Objects (RMI) und Remote Procedure Call (RPC) eine weitere Moeglichkeit um eine Kommunikation zwischen mehreren Rechnern umzusetzen.

Die Umsetzung basiert auf einem praxisnahen Beispiel einer Windkraftanalage. Ein Windkraftanalage (Windrad) ist immer Teil eines Verbunds, genannt Windpark. Jede Windkraftanlage beinhaltet einen Rechner, der die Daten der Windkraftanalage aufzeichnet und diese steuern kann. Die Daten werden in einer XML-Struktur abgespeichert. Die Daten aller Windkraftanlagen eines Windparks werden von einem Parkrechner gesammelt und abgespeichert. Der Parkrechner kommuniziert mit einem Rechner der Zentrale. Eine Zentrale kommuniziert mit mehreren Windparks und steuert diese.



1.1 Ziele

Das Ziel dieser Übung ist die Implementierung einer Kommunikationsplattform von mehreren Windparks mit einer zentralen Stelle unter Verwendung einer Message Oriented Middleware (MOM). Hier sollen nun die Parkrechner von mehreren Windparks die gesammelten Daten an eine zentrale Stelle übertragen. Aufgrund der Offenheit von nachrichtenbasierten Protokollen werden hier Message Queues verwendet. So kann gewährleistet werden, dass in Zukunft weitere Anlagen hinzugefügt bzw. Kooperationspartner eingebunden werden können.

1.2 Voraussetzungen

- Grundlagen Architektur von verteilten Systemen
- Grundlagen zur nachrichtenbasierten Systemen / Message Oriented Middleware
- Verwendung des Message Brokers Apache ActiveMQ
- Verwendung der XML-Datenstruktur eines Parkrechner "parknodedata.xml"
- Verwendung der JMSChat.jar JAVA Applikation als Startpunkt für diese Aufgabenstellung

1.3 Aufgabenstellung

Implementieren Sie die Windpark-Kommunikationsplattform mit Hilfe des Java Message Service. Verwenden Sie Apache ActiveMQ (http://activemq.apache.org) als Message Broker Ihrer Applikation. Das Programm soll folgende Funktionen beinhalten:

- Jeder Windpark (Parkrechner) erstellt eine Message Queue mit einem vorgegeben Namen.
- Der Parkrechner stellt die gesammelten Daten der Windkraftanlagen diese Message Queue.
- Der Zentralrechner lädt aus einer Konfigurationsdatei die Namen (Message Queues) aller Parkrechner.
- Der Zentralrechner verbindet sich mit allen Message Queues und empfängt die Daten der Windparks.
- Der Zentralrechner sammelt die Daten der Windparks und legt diese erneut in einer XML-Datei ab. Hier wird die XML-Struktur dynamisch erweitert, indem in der XML-Struktur der Name des Parkrechners und die Übertragungszeit abgelegt werden.
- Bei erfolgreicher Übertragung der Daten wird dem Parkrechner die Nachricht ”SUCCESS” übertragen. Die Umsetzung der Rückmeldung ist vom Software-Entwickler zu entwerfen und umzusetzen.

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

1.4 Bewertung

- Gruppengrösse: 1 Person
- Anforderungen "überwiegend erfüllt"
  - Implementierung der Kommunikation zwischen einem Parkrechner und dem Zentralrechner (JMS Queue)
- Anforderungen "zur Gänze erfüllt"
  - Implementierung der Kommunikation mit mehreren Parkrechner und dem Zentralrechner
  - Rückmeldung des Ergebnisses der Übertragung vom Zentralrechner an die Parkrechner (JMS: Topic)
  - Zusammensetzung der Daten aller Windparks in eine zentrale XML-Struktur

1.5 Fragestellung für Protokoll

- Nennen Sie mindestens 4 Eigenschaften der Message Oriented Middleware?
- Was versteht man

  Richard Wutscher, [26.04.18 14:10] unter einer transienten und synchronen Kommunikation?
- Beschreiben Sie die Funktionsweise einer JMS Queue?
- JMS Overview - Beschreiben Sie die wichtigsten JMS Klassen und deren Zusammenhang?
- Beschreiben Sie die Funktionsweise eines JMS Topic?
- Was versteht man unter einem lose gekoppelten verteilten System? Nennen Sie ein Beispiel dazu. Warum spricht man hier von lose?

1.5.1 Links & Dokumente

- Grundlagen Message Oriented Middleware: Praesentation
- XML-Datenstruktur eines Parkrechner: parknodedata.xml
- Middleware: Apache ActiveMQ Installationspaket
- Beispiel Quellcode: JMSChat.jar
- Apache ActiveMQ & JMS Tutorial:
  - http://activemq.apache.org/index.html
  - http://www.academictutorials.com/jms/jms-introduction.asp
  - http://docs.oracle.com/javaee/1.4/tutorial/doc/JMS.html#wp84181

- http://www.onjava.com/pub/a/onjava/excerpt/jms_ch2/index.html
- http://www.oracle.com/technetwork/systems/middleware/jms-basics-jsp-135286.html
- http://www.oracle.com/technetwork/articles/java/introjms-1577110.html

2 Lösung

2.1 ActiveMQ

2.1.1 Allgemein

Für die Message Oriented Middleware verwende ich **ActiveMQ**, ein open source 'Message Broker' von Apache.

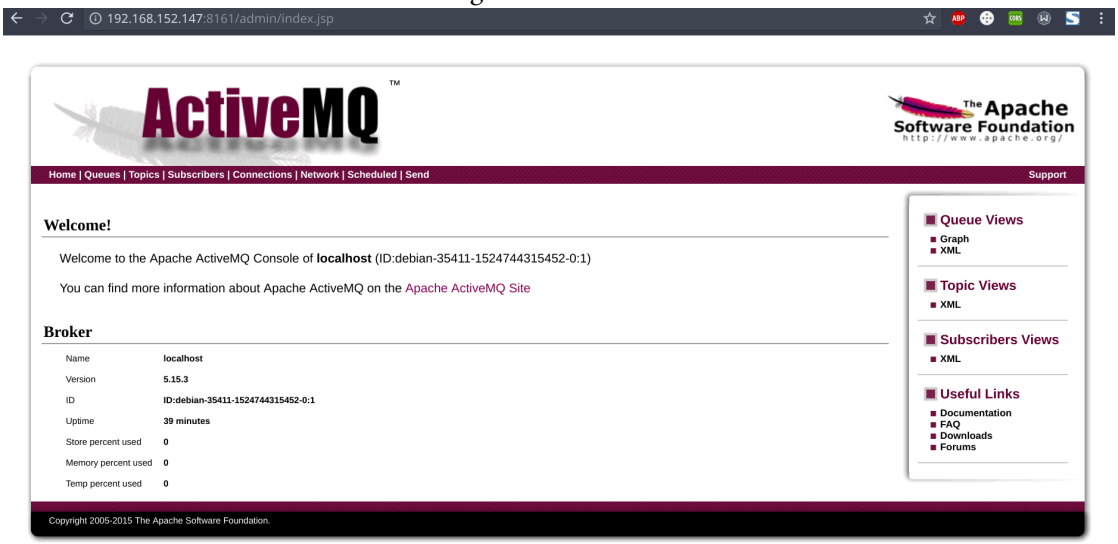
Der neueste Build ist zum Download verfügbar auf der ActiveMQ Download Seite, in meinem Fall ist das **ActiveMQ 5.15.3 (Unix/Linux)**.

2.1.2 Installieren

Ich habe mir ActiveMQ als soft-link hinzugefügt, sodass ich es von überall aufrufen kann:

```
$ tar -xzf apache-activemq-5.15.3-bin.tar.gz
$ mv apache-activemq-5.15.3-bin ~/activemq
$ cd ~/activemq/bin
$ chmod +x activemq
$ sudo ln -s /home/mrousavy/activemq/bin/activemq /usr/bin/activemq
$ activemq start
```

Nun ist der **ActiveMQ** Apache server unter `http://localhost:8161/admin` erreichbar. Benutzername und Passwort sind standardmäßig `admin` und `admin`.



Der Status des ActiveMQ services kann mittels

```
$ activemq status
```

abgerufen werden.

2.1.3 Java

Als nächstes muss ein Java Projekt erstellt werden, wozu ich **IntelliJ IDEA** verwende.

Nun muss die **ActiveMQ** library zu dem Projekt hinzugefügt werden, hierbei wird die library `activemq-all-5.15.3.jar` bereits in dem `apache-activemq-5.15.3-bin.tar.gz` archiv mitgegeben, jedoch verwende ich **Maven** um den Prozess der Libraries zu vereinfachen.

Die **ActiveMQ** library ist mittels folgendem `dependency` tag in dem `pom.xml` file hinzuzufügen:

```
<dependencies>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-all</artifactId>
        <version>5.8.0</version>
    </dependency>
</dependencies>
```

Ich verwende das Package `activemq-all`, da hier alle Artifakte inkludiert sind (`activemq-core`, ..).

2.2 Windfarm

Für eine **Windfarm** benötigen wir ein individuelles Windrad 2.2.1 welches Daten misst, einen Windpark 2.2.2 welcher Daten aller Windräder misst und eine Zentrale 2.2.3 welche die Daten aller Windparks misst.

2.2.1 Windrad

Das Windrad (*Windmill*) benötigt eine ID sowie eine Referenz zu dem Windpark (*Windfarm*).

Es soll die Windgeschwindigkeit, die Rotationsgeschwindigkeit, die Einheit der Geschwindigkeit, den Strom-output, die Einheit des Strom-outputs, die Blatt-position und die Latenz messen.

Außerdem wäre es hilfreich einen Zeitstempel mitzuspeichern.

Die Implementation folgt in der Klasse `Windmill.java`:

```java
// Windmill.java

@XmlRootElement
public class Windmill {
    @XmlTransient
    public Windfarm farm = null;
    @XmlElement
    public int id = 0;
    @XmlElement
    public double windSpeed = 0.0;
    @XmlElement
    public SpeedUnit speedUnit = SpeedUnit.KMH;
    @XmlElement
    public double powerOutput = 0.0;
    @XmlElement
    public PowerUnit powerUnit = PowerUnit.MEGA_WATT;
    @XmlElement
    public double rotationSpeed;
    @XmlElement
    public double bladePosition;
    @XmlElement
    public double latency;

    @XmlElement(name = "timestamp")
    public String getTimestamp() {
        DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
        return dateFormat.format(new Date());
    }
}
```

Wobei die Enums folgendermaßen definiert sind:

```java
// Windmill.java

// Measurement of Speed
```

```java
public enum SpeedUnit {
    KMH,
    MPH
}

// Measurement of Power
public enum PowerUnit {
    KILO_WATT,
    MEGA_WATT,
    GIGA_WATT
}
```

Um das ganze zu einem XML zu *parsen*, habe ich folgende Methode geschrieben:

```java
// Windmill.java

public String buildXml() {
    try {
        JAXBContext jaxbContext = JAXBContext.newInstance(Windmill.class);
        Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
        jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        StringWriter sw = new StringWriter();
        jaxbMarshaller.marshal(this, sw);
        return sw.toString();
    } catch (JAXBException e) {
        e.printStackTrace();
        return "";
    }
}
```

2.2.2 Windpark

Der Windpark (*Windfarm*) benötigt eine ID sowie eine Liste aus Windrädern.

Die Implementation folgt in der Klasse `Windfarm.java`:

```
// Windfarm.java

@XmlRootElement
public class Windfarm {
    @XmlElement
    public int id = 0;
    @XmlElement(name = "windmill")
    public ArrayList<Windmill> mills = new ArrayList<Windmill>();
}
```

Außerdem habe ich folgende Methode geschrieben um die Windfarm zu XML zu *parsen*:

```
    // Windfarm.java

    public String buildXml() {
        try {
            JAXBContext jaxbContext = JAXBContext.newInstance(Windfarm.class);
            Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
            jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            StringWriter sw = new StringWriter();
            jaxbMarshaller.marshal(this, sw);
            return sw.toString();
        } catch (JAXBException e) {
            e.printStackTrace();
            return "";
        }
    }
```

Diese Klasse soll in der Lage sein, mit der Middleware (ActiveMQ) zu kommunizieren. Hierzu werden folgende Attribute benötigt:

```
// Windfarm.java

@XmlTransient
private Session session = null;
@XmlTransient
private Connection connection = null;
@XmlTransient
private MessageProducer producer = null;
```

Sie wurden als `@XmlTransient` markiert, weil sie von der `buildXml` Methode (und somit dem `Marshaller`) nicht erfasst und *geparsed* werden sollen.

Ich habe folgende Statische Variablen definiert:

```java
// Statics.java

public class Statics {
    public static final String USER = ActiveMQConnection.DEFAULT_USER;
    public static final String PASSWORD = ActiveMQConnection.DEFAULT_PASSWORD;
    public static final String URL = "failover://tcp://192.168.152.148:61616"; //
    ↪  Oder ActiveMQConnection.DEFAULT_BROKER_URL;
    public static final String SUBJECT = "Windfarm";
}
```

Für die Herstellung einer Verbindung zu dem **ActiveMQ** Service habe ich folgende Methode implementiert:

```java
// Windfarm.java

public void connect() throws JMSException {
    ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(Statics.USER,
    ↪  Statics.PASSWORD, Statics.URL);
    connection = factory.createConnection();
    connection.start();

    // Create the session
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    Destination destination = session.createTopic(Statics.SUBJECT);

    // Create the producer.
    producer = session.createProducer(destination);
    producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
}
```

..sowie folgende Methode zum senden der derzeitigen Werte:

```java
// Windfarm.java

public void send() throws JMSException {
    TextMessage message = session.createTextMessage(buildXml());
    System.out.println(message.getText());
    producer.send(message);
}
```

..und folgende Methode zum stoppen und schließen der Verbindung:

```java
// Windfarm.java

public void stop() throws JMSException {
    connection.close();
    producer.close();
    session.close();
}
```

Um das ganze zu testen, bastelte ich schnell diese Main Methode zusammen, welche jede halbe Sekunde das derzeitige XML an die Message Oriented Middleware (MOM) schickt:

```java
// Windfarm.java

public static void main(String[] args) {
    try {
        System.out.println("Starting Windfarm..");

        Windfarm farm = new Windfarm();
        farm.connect();
        // send XML every half second, 10 times
        for (int i = 0; i < 10; i++) {
            System.out.println("Sending XML..");
            farm.send();
            Thread.sleep(500);
        }
        // stop service
        farm.stop();

        System.out.println("Windfarm finished!");
    } catch (JMSException ex) {
        System.out.println("[Windfarm] error: " + ex);
        ex.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

2.2.3 Zentrale

Die Zentrale (*Headquarter*) soll lediglich die Daten von der MOM auslesen und anzeigen.
Die Implementation folgt in der `Headquarter.java` Datei.
Es wird wieder eine Methode zum Verbinden benötigt:

```
// Headquarter.java

public void connect() throws JMSException {
    ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(Statics.USER,
    ↪  Statics.PASSWORD, Statics.URL);
    connection = factory.createConnection();
    connection.start();

    // Create the session
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    Destination destination = session.createTopic(Statics.SUBJECT);

    // Create the consumer
    consumer = session.createConsumer(destination);
}
```

..sowie eine Methode zum empfangen bzw. auslesen von Messages:

```
// Headquarter.java

public void receive() throws JMSException {
    // Start receiving
    TextMessage message = (TextMessage) consumer.receive();
    if (message != null) {
        System.out.println("[Headquarter] Message received: " + message.getText());
        message.acknowledge();
    }
}
```

..und eine Methode zum beenden:

```
// Headquarter.java

public void stop() throws JMSException {
    connection.close();
    consumer.close();
    session.close();
}
```

In diesem Beispiel werde ich den Headquarter-service starten um 10 Messages von der MOM auszulesen:

```java
// Headquarter.java

public static void main(String[] args) {
    try {
        System.out.println("Starting Headquarter..");

        Headquarter hq = new Headquarter();
        hq.connect();
        // receive a message 10 times in total
        for (int i = 0; i < 10; i++) {
            hq.receive();
        }
        // stop the service
        hq.stop();

        System.out.println("Headquarter finished!");
    } catch (JMSException e) {
        e.printStackTrace();
    }
}
```

2.3 Fragen

2.3.1 Nennen Sie mindestens 4 Eigenschaften der Message Oriented Middleware

- Messages: Durch Queues oder Topics können Nachrichten Stackweise ausgetauscht werden.
- Persistente (Asynchrone) Kommunikation: Persistent bedeutet, dass die Nachrichten im Kommunikationssystem gespeichert. Asynchron heißt, dass die Sender gleich nach der Einreichung der Nachricht weiter arbeiten.
- Intermediate storage capacity for messages in the communication network
- Communication may take minutes (not ms)

2.3.2 Was versteht man unter einer transienten und synchronen Kommunikation?

- Transient: Die Nachricht ist solange auf dem SStacköder in der "Queue"bis sie gelesen (`acknowledge`) wurde.
- Synchron: Der Sender wird solange blockiert, bis ein Empfänger die Nachricht gelesen (`acknowledge`) hat.

2.3.3 Beschreiben Sie die Funktionsweise einer JMS Queue

Dies ist ein point-to-point Producer - Consumer System, in dem der Producer Nachrichten auf die Message Queue legt, welche sich der Consumer Stück für Stück abholen kann.

2.3.4 JMS Overview - Beschreiben Sie die wichtigsten JMS Klassen und deren Zusammenhang

- ConnectionFactory: Eine Factory um eine Connection zu erstellen.
- Connection: Stellt die Verbindung zur Middleware da, welche mit `start()` noch gestartet werden muss.
- Session: Stellt die derzeitige Sitzung dar, unter anderem welche Nachrichten bestätigt (`acknowledge`) worden sind, und welche Verbindung existiert.Mit *connection.session(false, Session.AUTO_ACKNOWLEDGE)* wird die Session erstellt. Der erste Parameter (`bool`) stellt fest dass es sich um Transaktionen handelt, welche mit commit und rollback behandelt werden können.
- Producer: Der Producer ist der Ersteller und Sender der Nachrichten. Der Producer wird mit *session.createProducer(destination)* erstellt.
- Consumer: Der Empfänger der Nachrichten. Er erhält die Nachrichten mit *consumer.receive()*.
- TextMessage: Eine Text (String) Nachricht. Mit *message.getText()* erhaltet man den String.

2.3.5 Beschreiben Sie die Funktionsweise einer JMS Topic

Ein Topic ähnelt stark einer Queue, jedoch sind die wesentlichen Unterschiede dass es bei Topics mehrere Consumer geben kann, und die Nachrichten nicht wie bei der Queue in einer bestimmten Reihenfolge ankommen können.

2.3.6 Was versteht man unter einem lose gekoppelten verteilten System? Nennen Sie ein Beispiel dazu. Warum spricht man hier von lose?

Lose Kopplung bedeutet, dass das System nicht abhängig von einzelnen Akteuren ist. Hier bedeutet das zum Beispiel, dass die Queue nicht von ihren einzelnen Receivern und Producern abhängig ist und diese jederzeit wieder gehen können, während andere dazukommen können. Ein Beispiel dazu wäre ein Broadcast von Informationen, welcher immer arbeitet, unabhängig von den Consumern.

Akronyme

MOM Message Oriented Middleware. 13, 14