

Systemtechnik Labor

4AHIT 2017/18

# **Message Oriented Middleware**

## **Laborprotokoll**

Marc Rousavy

26. April 2018

Bewertung:

Betreuer: Thomas Micheler

Version: 1.1

Begonnen: 26. April 2018

Beendet: 26. April 2018

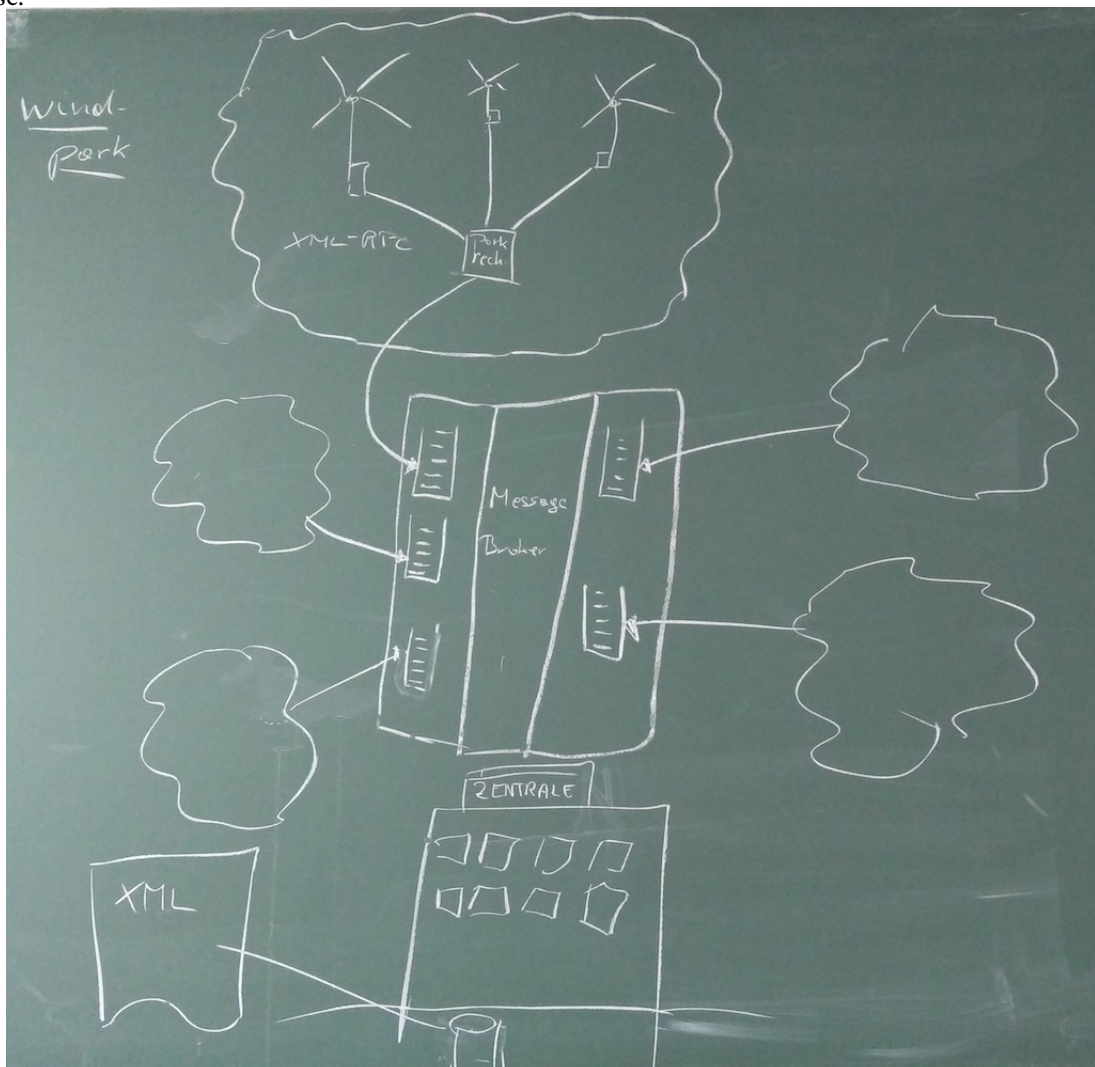
## Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
1.1	Ziele . . . . .	4
1.2	Voraussetzungen . . . . .	4
1.3	Aufgabenstellung . . . . .	4
1.4	Bewertung . . . . .	5
1.5	Fragestellung für Protokoll . . . . .	5
1.5.1	Links & Dokumente . . . . .	5
<b>2</b>	<b>Lösung</b>	<b>7</b>
2.1	ActiveMQ . . . . .	7
2.1.1	Allgemein . . . . .	7
2.1.2	Installieren . . . . .	7
2.1.3	Java . . . . .	8
2.2	Windfarm . . . . .	9
2.2.1	Windrad . . . . .	9
2.2.2	Windpark . . . . .	11
2.2.3	Zentrale . . . . .	14

## 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 Windkraftanlage. Ein Windkraftanlage (Windrad) ist immer Teil eines Verbunds, genannt Windpark. Jede Windkraftanlage beinhaltet einen Rechner, der die Daten der Windkraftanlage 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.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.

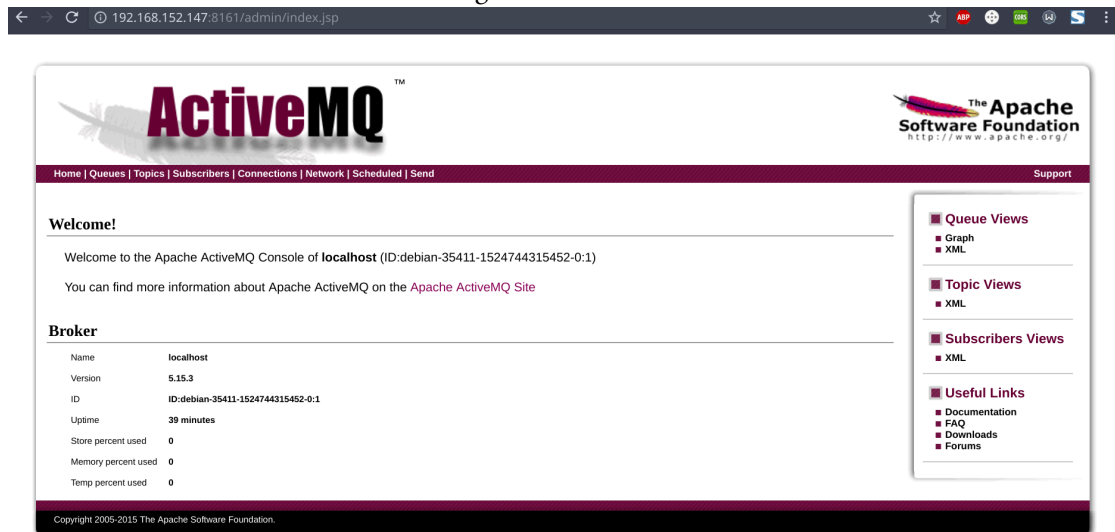
Der neueste Build ist zum Download verfügbar auf der [ActiveMQ Download Seite](#), in meinem Fall ist das **ActiveMQ 5.15.3 (Unix/Linux)**.

#### 2.1.2 Installieren

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

```
1 $ tar -xzf apache-activemq-5.15.3-bin.tar.gz
2 $ mv apache-activemq-5.15.3-bin ~/activemq
3 $ cd ~/activemq/bin
4 $ chmod +x activemq
5 $ sudo ln -s /home/mrousavy/activemq/bin/activemq /usr/bin/activemq
6 $ 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

```
1 $ 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:

```
1 <dependencies>
2   <dependency>
3     <groupId>org.apache.activemq</groupId>
4     <artifactId>activemq-all</artifactId>
5     <version>5.8.0</version>
6   </dependency>
7 </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`:

```
1 // Windmill.java
2
3 @XmlElement
4 public class Windmill {
5     @XmlTransient
6     public Windfarm farm = null;
7     @XmlElement
8     public int id = 0;
9     @XmlElement
10    public double windSpeed = 0.0;
11    @XmlElement
12    public SpeedUnit speedUnit = SpeedUnit.KMH;
13    @XmlElement
14    public double powerOutput = 0.0;
15    @XmlElement
16    public PowerUnit powerUnit = PowerUnit.MEGA_WATT;
17    @XmlElement
18    public double rotationSpeed;
19    @XmlElement
20    public double bladePosition;
21    @XmlElement
22    public double latency;
23
24    @XmlElement(name = "timestamp")
25    public String getTimestamp() {
26        DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
27        return dateFormat.format(new Date());
28    }
29 }
```

Wobei die Enums folgendermaßen definiert sind:

```
1 // Windmill.java
2
3 // Measurement of Speed
```

```
4 public enum SpeedUnit {
5     KMH,
6     MPH
7 }
8
9 // Measurement of Power
10 public enum PowerUnit {
11     KILO_WATT,
12     MEGA_WATT,
13     GIGA_WATT
14 }
```

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

```
1 // Windmill.java
2
3 public String buildXml() {
4     try {
5         JAXBContext jaxbContext = JAXBContext.newInstance(Windmill.class);
6         Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
7         jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
8         StringWriter sw = new StringWriter();
9         jaxbMarshaller.marshal(this, sw);
10        return sw.toString();
11    } catch (JAXBException e) {
12        e.printStackTrace();
13        return "";
14    }
15 }
```

### 2.2.2 Windpark

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

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

```
1 // Windfarm.java
2
3 @XmlElement
4 public class Windfarm {
5     @XmlElement
6     public int id = 0;
7     @XmlElement(name = "windmill")
8     public ArrayList<Windmill> mills = new ArrayList<Windmill>();
9 }
```

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

```
1 // Windfarm.java
2
3 public String buildXml() {
4     try {
5         JAXBContext jaxbContext = JAXBContext.newInstance(Windfarm.class);
6         Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
7         jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
8         StringWriter sw = new StringWriter();
9         jaxbMarshaller.marshal(this, sw);
10        return sw.toString();
11    } catch (JAXBException e) {
12        e.printStackTrace();
13        return "";
14    }
15 }
```

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

```
1 // Windfarm.java
2
3 @XmlTransient
4 private Session session = null;
5 @XmlTransient
6 private Connection connection = null;
7 @XmlTransient
8 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.

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

```
1 // Windfarm.java
2
3 public void connect() throws JMSEException {
4     ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(Statics.USER,
5     ↪ Statics.PASSWORD, Statics.URL);
6     connection = factory.createConnection();
7     connection.start();
8
9     // Create the session
10    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
11    Destination destination = session.createTopic(Statics.SUBJECT);
12
13    // Create the producer.
14    producer = session.createProducer(destination);
15    producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
16 }
```

..sowie folgende Methode zum senden der derzeitigen Werte:

```
1 // Windfarm.java
2
3 public void send() throws JMSEException {
4     TextMessage message = session.createTextMessage(buildXml());
5     System.out.println(message.getText());
6     producer.send(message);
7 }
```

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

```
1 // Windfarm.java
2
3 public void stop() throws JMSEException {
4     connection.close();
5     producer.close();
6     session.close();
7 }
```

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:

```
1 // Windfarm.java
2
3 public static void main(String[] args) {
4     try {
5         System.out.println("Starting Windfarm..");
6     } catch (Exception e) {
7         e.printStackTrace();
8     }
9 }
```

```
6
7     Windfarm farm = new Windfarm();
8     farm.connect();
9     // send XML every half second, 10 times
10    for (int i = 0; i < 10; i++) {
11        System.out.println("Sending XML..");
12        farm.send();
13        Thread.sleep(500);
14    }
15    // stop service
16    farm.stop();
17
18    System.out.println("Windfarm finished!");
19 } catch (JMSEException ex) {
20     System.out.println("[Windfarm] error: " + ex);
21     ex.printStackTrace();
22 } catch (InterruptedException e) {
23     e.printStackTrace();
24 }
25 }
```

### 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:

```
1 // Headquarter.java
2
3 public void connect() throws JMSEException {
4     ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(Statics.USER,
5     ↪ Statics.PASSWORD, Statics.URL);
6     connection = factory.createConnection();
7     connection.start();
8
9     // Create the session
10    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
11    Destination destination = session.createTopic(Statics.SUBJECT);
12
13    // Create the consumer
14    consumer = session.createConsumer(destination);
15 }
```

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

```
1 // Headquarter.java
2
3 public void receive() throws JMSEException {
4     // Start receiving
5     TextMessage message = (TextMessage) consumer.receive();
6     if (message != null) {
7         System.out.println("[Headquarter] Message received: " + message.getText());
8         message.acknowledge();
9     }
10 }
```

..und eine Methode zum beenden:

```
1 // Headquarter.java
2
3 public void stop() throws JMSEException {
4     connection.close();
5     consumer.close();
6     session.close();
7 }
```

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

```
1 // Headquarter.java
2
3 public static void main(String[] args) {
4     try {
5         System.out.println("Starting Headquarter..");
6
7         Headquarter hq = new Headquarter();
8         hq.connect();
9         // receive a message 10 times in total
10        for (int i = 0; i < 10; i++) {
11            hq.receive();
12        }
13        // stop the service
14        hq.stop();
15
16        System.out.println("Headquarter finished!");
17    } catch (JMSEException e) {
18        e.printStackTrace();
19    }
20 }
```

## Akronyme

**MOM** Message Oriented Middleware. [12](#), [14](#)