
Laborprotokoll

DEZSYS-L05 Message Queues

Systemtechnik Labor
4CHIT 2016/17

Marvin Ertl

Note:
Betreuer:

Version 1.0
Begonnen am 9. Juni 2017
Beendet am 16. Juni 2017

Inhaltsverzeichnis

1	Einführung	1
1.1	Ziele	1
1.2	Voraussetzungen	1
1.3	Aufgabenstellung	1
2	Ergebnisse	3
2.1	Starten des ActiveMQ	3
2.2	Beispiel ausführen	4
2.3	Design	5
2.4	Parkrechner	7
2.5	Zentralrechner	8

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.

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). Das Beispiel baut auf dem Beispiel DEZSYS-L04 "Datenuebertragung von Windkraftanlagen zum Parkrechner via XML-RPCäuf. Hier sollen nun die Parkrechner von mehreren Windparks die gesammelten Daten an eine zentrale Stelle uebertragen. Aufgrund der Offenheit von nachrichtenbasierten Protokollen werden hier Message Queues verwendet. So kann gewaehrleistet werden, dass in Zukunft weitere Anlagen hinzuzufuegt bzw. Kooperationspartner eingebunden werden koennen.

1.2 Voraussetzungen

- Grundlagen Architektur von verteilten Systemen
- Grundlagen zur nachrichtenbasierten Systemen / Message Oriented Middleware
- Verwendung des Message Brokers Apache ActiveMQ (Link)
- Verwendung der XML-Datenstruktur aus der Aufgabenstellung DEZSYS-L04
- Verwendung der JMSChat.jar JAVA Applikation als Startpunkt fuer 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 aus Beispiel DEZSYS-L04 in diese Message Queue.
- Der Zentralrechner laedt aus einer Konfigurationsdatei die Namen (Message Queues) aller Parkrechner.
- Der Zentralrechner verbindet sich mit allen Message Queues und empfaengt 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 Uebertragungszeit abgelegt werden.
- Bei erfolgreicher Uebertragung der Daten wird dem Parkrechner die Nachricht SSUCCESSübertragen. Die Umsetzung der Rueckmeldung ist vom Software-Entwickler zu entwerfen und umzusetzen.

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

2 Ergebnisse

2.1 Starten des ActiveMQ

Um Messages Queues verwenden zu können muss die Middleware gestartet werden. Dies ist in unserem Fall der Apache ActiveMQ Broker. Um diesen starten zu können, müssen die nötigen Daten heruntergeladen werden und entpackt werden.

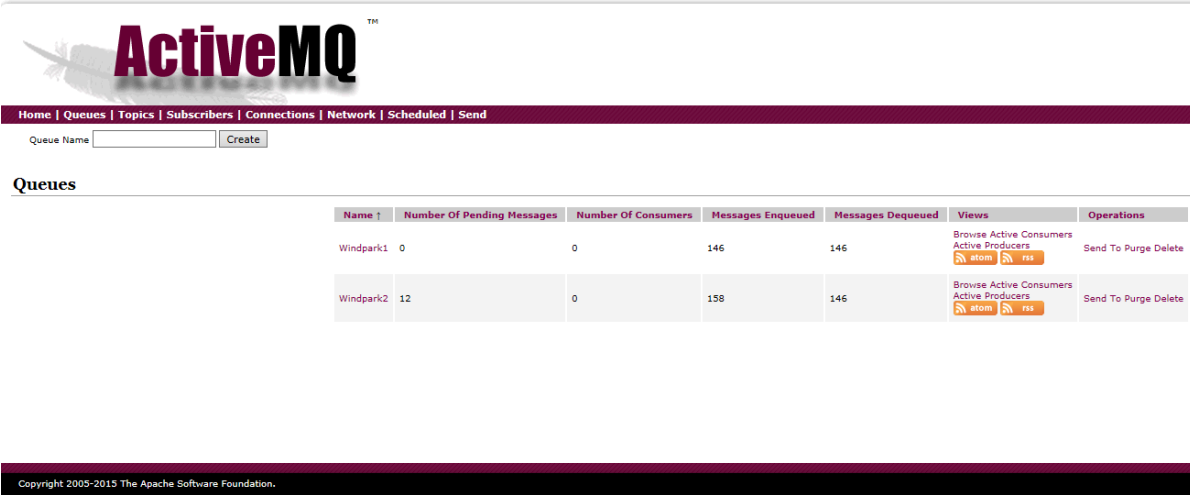
Nun kann man den Broker starten, hierfür geht man in den Unterordner bin und nun kann man mittels folgenden Befehl unter Windows die Middleware starten:

```
1 java -jar .\activemq.jar start
```

Listing 1: Starten der Middleware

Jetzt kann man mittels einen Webbrowser auf das Admininterface zugreifen, hierfür ruft man localhost mit den Port 8161 auf. Wird man nun nach einen Benutzernamen und Passwort gefragt, muss man bei beiden admin eingeben.

Auf der Seite hat man eine Übersicht über alle Queues und Topics, hier hat man auch die Möglichkeit manuell Messages eingeben.



The screenshot shows the ActiveMQ web interface. At the top is the ActiveMQ logo. Below it is a navigation bar with links: Home, Queues, Topics, Subscribers, Connections, Network, Scheduled, and Send. There is a search bar for Queue Name and a Create button. The main section is titled 'Queues' and contains a table with the following data:

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
Windpark1	0	0	146	146	Browse Active Consumers Active Producers atom rss	Send To Purge Delete
Windpark2	12	0	158	146	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

At the bottom of the interface, there is a footer that reads: Copyright 2005-2015 The Apache Software Foundation.

Abbildung 1: Übersicht von allen Queues im Webinterface

2.2 Beispiel ausführen

Es wird ein neues Java Projekt erstellt, in dieses kann dann über die Projekt Properties, Java Build Path die externen Libraries einbinden. Dies sieht wie folgt aus:

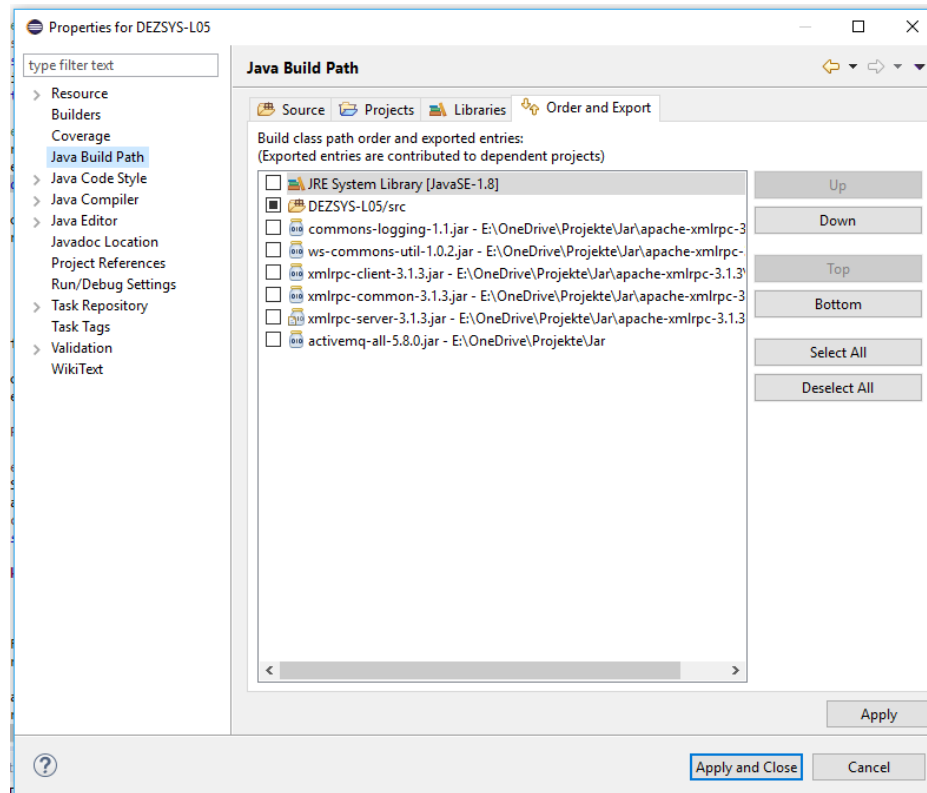


Abbildung 2: Einfügen der externen Libraries

Nachdem man die Middleware wie im ersten Kapitel beschrieben gestartet hat, kann man den Reciver und Sender starten. Jetzt sollte folgendes in der Konsole angezeigt werden.

```
2017-06-16 01:03:33,392 INFO org.apache.activemq.transport.failover.FailoverTransport - Successfully connected to tcp://localhost:61616
2017-06-16 01:03:33,392 INFO org.apache.activemq.transport.failover.FailoverTransport - Successfully connected to tcp://localhost:61616
MaxMustermann [ xxx.xxx.xxx.xxx ]: This message was sent at (ms): 1497567813412
```

Abbildung 3: Nachricht wurde übertragen

2.3 Design

Auf Basis des Designs der letzten Übung wurde das Programm erweitert. Dabei gab es keine Änderungen beim Windrad.

Der Parkrechner wurde aber erweitert. Dieser besitzt nun eine Verbindung zur Queue, über diese liefert er die XML mit den Windrädern an den Zentralrechner. Weiteres gibt es eine Verbindung zu einem Topic, über dieses wird der Parkrechner informiert ob die neuen Informationen über die Queue am Zentralrechner angekommen sind.

Der Zentralrechner spielt hier die wichtigste Rolle, da dieser in einer ArrayList alle Message Consumer zu den Queues verwaltet, um die neuen Informationen von dem Parkrechner zu bekommen und in einer weiteren ArrayList alle Message Producer für die Topics, über welche die Parkrechner informiert werden, ob die Informationen angekommen sind. Dabei verwendet der Zentralrechner, wie auch der Parkrechner, einen eigenen Thread in dem die Queues andauernd abgefragt werden.

Weiteres werden alle Parkrechner die in der config.xml sind zu dem Zentralrechner hinzugefügt. Außerdem werden die Informationen anschließend verarbeitet und in eine XML Datei geschrieben.

Nachfolgendes UML Diagramm visualisiert die beschriebene Struktur, wobei der Parkrechner der Client ist und jedes Windrad ein Server ist.

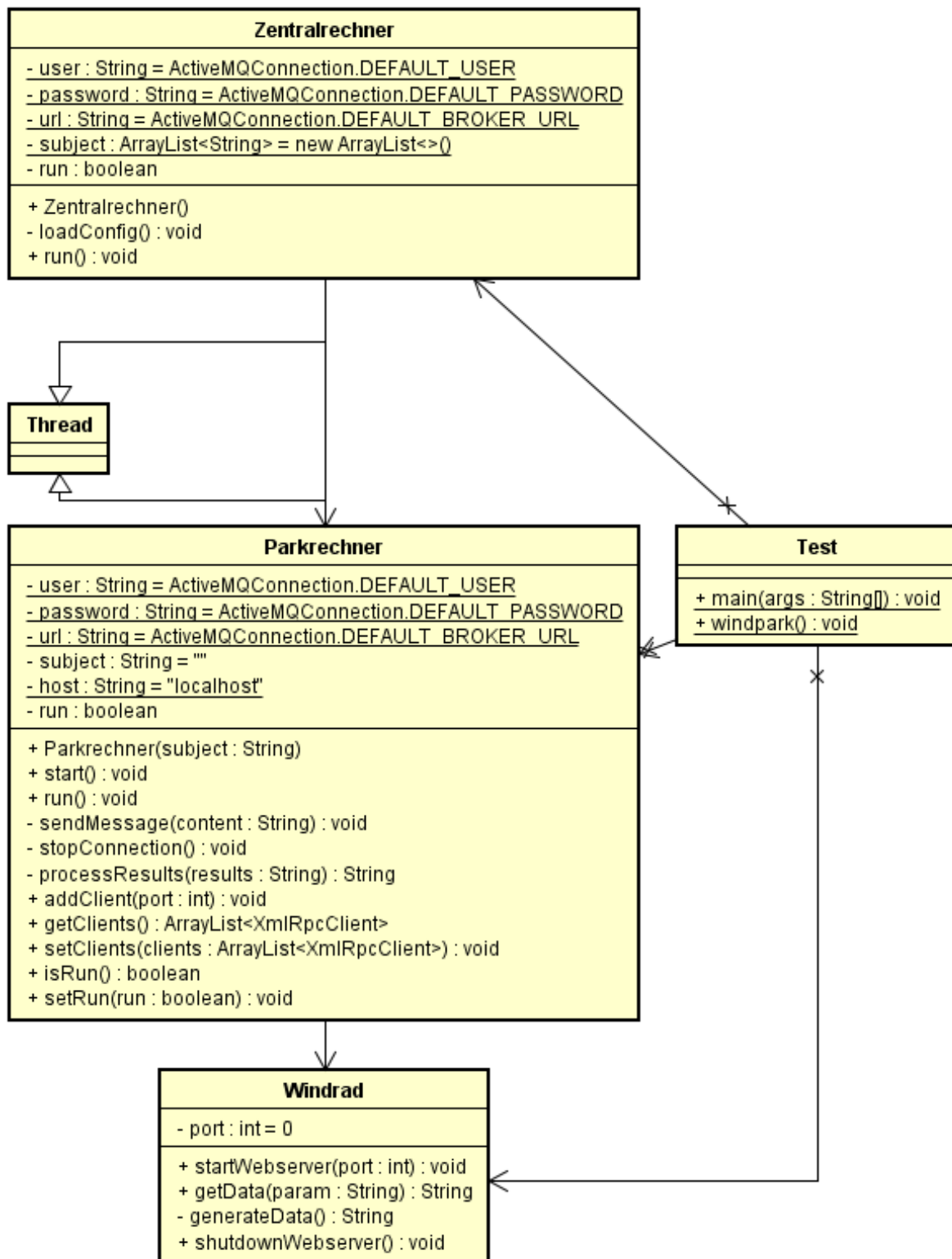


Abbildung 4: UML Diagramm

2.4 Parkrechner

Im Konstruktor wird nun die Verbindung zu einer Queue hergestellt über diese sendet der Parkrechner die Informationen, die über RPC von den Windräder kommen und als XML aufbereitet werden. Der Name der Queue muss als Parameter angegeben werden.

```

1  try {
    ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(user, password, url);
3   this.connection = connectionFactory.createConnection();
    this.connection.start();

5   this.session = this.connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
7   this.destination = this.session.createQueue(this.subject);

9   this.producer = this.session.createProducer(this.destination);
    this.producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
11 } catch (Exception e) {
    System.out.println("[MessageProducer] Caught: " + e);
13 }

```

Listing 2: Erstellen der Verbindung zur Queue

Weiteres wird auch eine Verbindung zum Topic mit dem selben Namen hergestellt, über dieses empfängt der Parkrechner die Information ob die neusten Informationen beim Zentralrechner angekommen sind. Die Verbindungen werden dabei als Objektattribute gespeichert.

```

1  try {
    ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(user, password, url);
3   this.connectionTopic = connectionFactory.createConnection();
    this.connectionTopic.start();

5   this.sessionTopic = this.connectionTopic.createSession(false, Session.AUTO_ACKNOWLEDGE);
7   this.destinationTopic = this.sessionTopic.createTopic(subject);

9   this.consumerTopic = this.sessionTopic.createConsumer(this.destinationTopic);
11 } catch (Exception e) {
    System.out.println("[MessageConsumer] Caught: " + e);
}

```

Listing 3: Erstellen der Verbindung zum Topic

Nachdem die Informationen der Windräder als XML abgespeichert worden sind, werden diese mittels der Queue zum Zentralrechner übertragen, dieses erfolgt in einer eigenen Methode, welche als Parameter die Informationen als XML erwartet.

```

private void sendMessage(String content) {
2   try {
    TextMessage message = this.session.createTextMessage(content);
4    this.producer.send(message);
    } catch (Exception e) {
6     System.out.println("[MessageProducer] Caught: " + e);
    this.stopConnection();
8   }
}

```

Listing 4: Senden der Windpark Informationen

Wurden die Informationen gesendet, wartet der Parkrechner nun auf eine Rückmeldung via der Topic Verbindung.

```

1  try {
    TextMessage message = (TextMessage) consumerTopic.receive();
3   if (message != null) {
    System.out.println("Informationen empfangen: " + message.getText());
}

```

```

5      message.acknowledge();
6    }
7  } catch (JMSEException e1) {
8      System.out.println("Exception receiving topic");
9  }

```

Listing 5: Empfangen der Bestätigung

2.5 Zentralrechner

Im Zentralrechner werden alle Parkrechner Verbindungen über Queues und Topics verwaltet. Hierfür wird aus folgender config Datei die Namen der Queues gelesen:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <queues>
3   <queue>
4     Windpark1
5   </queue>
6   <queue>
7     Windpark2
8   </queue>
9 </queues>

```

Listing 6: Config Datei mit den Namen der Queues

Die Namen der Queues werden mittels folgender Methode ausgelesen und in einer ArrayListe abgespeichert.

```

1 private void loadConfig() {
2     try {
3         XMLInputFactory factory = XMLInputFactory.newInstance();
4         XMLEventReader eventReader = factory.createXMLEventReader(new FileReader("config.xml"));
5
6         while (eventReader.hasNext()) {
7             XMLEvent event = eventReader.nextEvent();
8             switch (event.getEventType()) {
9                 case XMLStreamConstants.CHARACTERS:
10                 Characters characters = event.asCharacters();
11                 if (characters.getData().trim().length() != 0) {
12                     subject.add(characters.getData().trim());
13                 }
14                 break;
15                 default: break;
16             }
17         }
18     } catch (FileNotFoundException e) {
19         System.out.println("File not found");
20         return;
21     } catch (XMLStreamException e) {
22         System.out.println("XML file invailed");
23         return;
24     }
25 }

```

Listing 7: Laden der config Datei

Nachdem die Namen der Queues bekannt sind, kann nun eine Verbindung zu allen diesen Queues erstellt werden. Diese Verbindungen werden wieder in ArrayListen gespeichert. Für die Queues werden Message Consumer verwendet, da der Zentralrechner über diese die Informationen als XML bekommt.

```

1 try {
2     for (String subject : subject) {

```

```

3      ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(user, password, url);
      Connection con = connectionFactory.createConnection();
5      con.start();
      this.connection.add(con);

7      Session ses = con.createSession(false, Session.AUTO_ACKNOWLEDGE);
9      this.session.add(ses);
      Destination des = ses.createQueue(subject);
11     this.destination.add(des);

13     this.consumer.add(ses.createConsumer(des));
    }
15 } catch (Exception e) {
    System.out.println("[MessageConsumer] Caught: " + e);
17     e.printStackTrace();
    this.run = false;
19 }

```

Listing 8: Erstellen der Verbindung zur Queue

Nun kann auch eine Verbindung zu allen Topics hergestellt werden, hierfür werden auch wieder die Namen aus der config Datei genommen. In diesem Fall werden Message Producer verwendet, da der Zentralrechner über die Topics die Parkrechner informiert, ob die neusten Informationen angekommen sind.

```

1  try {
2      for (String subject : subject) {
3          ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(user, password, url);
          Connection con = connectionFactory.createConnection();
5          con.start();
          this.connectionTopic.add(con);

7          // Create the session
9          Session ses = con.createSession(false, Session.AUTO_ACKNOWLEDGE);
          this.sessionTopic.add(ses);
11         Destination des = ses.createTopic(subject);
          this.destinationTopic.add(des);

13         // Create the producer.
15         MessageProducer pro = ses.createProducer(des);
          pro.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
17         this.producerTopic.add(pro);
    }
19 } catch (Exception e) {
    System.out.println("[MessageProducer] Caught: " + e);
21 }

```

Listing 9: Erstellen der Verbindung zum Topic

Nachdem alle Verbindungen hergestellt worden sind, kann man den Thread starten, in diesem wird in einer Endlosschleife alle Queues abgefragt, ob es neue Informationen gibt. Wird eine Message empfangen, wird der Parkrechner über die Topics informiert ob, die Nachricht angekommen ist.

```

1  StringBuilder result = new StringBuilder();
  try {
3      for (int i = 0; i < this.consumer.size(); i++) {
          MessageConsumer consumer = this.consumer.get(i);
5          TextMessage message = (TextMessage) consumer.receive();
          boolean received = false;
7          if (message != null) {
              System.out.println("Received new Windpark informations.");
9              message.acknowledge();
              received = true;
11         }

13         //Topic
          String message3 = "not successfull";
15         if (received) message3 = "SUCCESS";
    }
  }

```

```

17     TextMessage message2 = this.sessionTopic.get(i).createTextMessage(message3);
    this.producerTopic.get(i).send(message2);

19     StringBuilder temp = new StringBuilder(message.getText());
    String windpark = "<windpark id=\"" + subject.get(i) + "\" time=\"" + System.currentTimeMillis()
    + "\">\n" + temp.substring(50, temp.length());
21     result.append(windpark);
}

```

Listing 10: Empfangen der Informatinen und senden der Rückmeldung

Nun werden die Empfangen Informationen zusammengefasst als XML. Weiteres wird überprüft ob die Datei Zentralrechner.xml vorhanden ist, in diese Datei werden alle Parkrechner Informationen gespeichert. Ist diese Datei nicht vorhanden, wird diese erstellt.

```

String[] temp = result.toString().split("\n");
2 String file = "";
for (String string : temp) {
4     file += "\t" + string + "\n";
}

6 File f = new File("Zentralrechner.xml");
8 if (!f.exists() && !f.isDirectory()) {
    f.createNewFile();
10     try (BufferedWriter bw = new BufferedWriter(new FileWriter("Zentralrechner.xml"))) {
        String content = "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<zentralrechner>\n";
12         bw.write(content);
    } catch (IOException e) {
14         System.out.println("Could not write to Zentralrechner.xml");
        return;
16     }
}

```

Listing 11: Erstellen der XML Datei

Nachdem die neuen Informationen vorbereitet worden sind, können diese nun in die XML Datei abgespeichert, hierfür muss das End Root Element entfernt werden, die neuen Informationen angefügt werden und anschließend wieder das End Root Element anfügen.

```

1 try {
    File inFile = new File("Zentralrechner.xml");
    File tempFile = new File(inFile.getAbsolutePath() + ".tmp");
    BufferedReader br = new BufferedReader(new FileReader("Zentralrechner.xml"));
    PrintWriter pw = new PrintWriter(new FileWriter(tempFile));

    String line = null;
    while ((line = br.readLine()) != null) {
        if (!line.trim().contains("</zentralrechner>")) {
9             pw.println(line);
            pw.flush();
11         }
    }
13     pw.close();
    br.close();

15     if (!inFile.delete()) {
        System.out.println("Could not delete file");
17         return;
    }
21     if (!tempFile.renameTo(inFile)) {
        System.out.println("Could not rename file");
23         return;
    }
25 } catch (FileNotFoundException ex) {
    System.out.println("File not found");
27     return;
} catch (IOException ex) {
29     System.out.println("IOException");
}

```

```
31     return;
32 }
33 try {
34     file += "\n</zentralrechner>";
35     Files.write(Paths.get("Zentralrechner.xml"), file.getBytes(), StandardOpenOption.APPEND);
36 } catch (IOException e) {
37     System.out.println("Could not write to Zentralrechner.xml");
38     return;
39 }
```

Listing 12: Anhängen der neuen Informationen

Listings

1	Starten der Middleware	3
2	Erstellen der Verbindung zur Queue	7
3	Erstellen der Verbindung zum Topic	7
4	Senden der Windpark Informationen	7
5	Empfangen der Bestätigung	7
6	Config Datei mit den Namen der Queues	8
7	Laden der config Datei	8
8	Erstellen der Verbindung zur Queue	8
9	Erstellen der Verbindung zum Topic	9
10	Empfangen der Informatinen und senden der Rückmeldung	9
11	Erstellen der XML Datei	10
12	Anhängen der neuen Informationen	10

Abbildungsverzeichnis

1	Übersicht von allen Queues im Webinterface	3
2	Einfügen der externen Libraries	4
3	Nachricht wurde übertragen	4
4	UML Diagramm	6