tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

Systemtechnik Labor

xHIT 2017/18

Message Oriented Middleware (MOM)

Laborprotokoll

Mario Fentler

5. April 2018

Bewertung:		Version:	0.1
Betreuer:	Micheler	Begonnen:	14.3.18
		Beendet:	14.3.18

Inhaltsverzeichnis

1 Einführung

Die Übung soll die Funktionsweise und Implementierung von einer Message Oriented Middleware(MOM) mit Hilfe des Frameworks Apache Active MQ demonstrieren. Neben IPC, RemoteObjects und RPC ist MOM eine weitere Möglichkeit 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 SSUCCESS"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 ßur 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

2 Aufgabenvorarbeit

Damit alles funktioniert muss man folgende Schritte machen:

1. Systemumgebungsvariable erstellen:
   Die JAVA_HOME Umgebungsvariable soll zu dem Verzeichnis zeigen wo jre installiert ist.

2. ActiveMQ:
   Die Datei entpacken und dann mit der CMD in den Ordner (Bei mir im MOM-Ordner) und bin/activemq start -> wenn kein Fehler -> localhost:8161 -> Manage MQ (admin:admin)

3. Ant installieren:
   In C: entpacken, ANT_HOME zu den Umgebungsvariablen hinzufügen und %ANT_% \bin zum Path dranhängen

4. Projektumgebung aufbauen:
   Das JMSChat.jar File in Eclipse importieren.

3 Aufgabendurchführung

Nachdem der ActiveMq Server konfiguriert wurde habe ich ihn mit dem vorhandenen Programm getestet. Somit konnte man überprüfen, ob der Server läuft und funktioniert. Man startet ihn mit den Befehlen:

- cd <Installationsordner>
- \bin activemq start

Man kann auf das Webinterface vom Server über localhost:8616\admin zugreifen.

Um alles zu testen benötigt man 3 Konsolen Fenster: eines in dem der Server gestartet wird, einen Parkrechner und einen Zentralrechner.
Damit man eine Nachricht am Receiver bekommt muss dieser als erstes gestartet werden und danach der Sender.
Man startet sie mit dem Befehl: ant run-Receiver / ant run-Sender. Daraufhin sendet der Parkrechner eine Nachricht an den Zentralrechner, welche dann auf beiden Konsolenfenstern angezeigt wird.

- ant compile: Wenn Änderungen am Code gemacht wurden
- ant run-receiver: Startet den Receiver
- ant run-sender: Startet den Sender

Um die vorgegebene Aufgabenstellung zu lösen mussten wir dieses Programm anpassen. Der Parkrechner sollte XML Dateien senden und der Zentralrechner diese dann wieder als Text ausgeben und auch in ein File speichern. Weiters ist das XML File vom Parkrechner zufällig generiert, damit nicht immer die selben Werte rauskommen.

4 Code-Anpassungen/Erweiterungen

4.1 Parkrechner (Sender)

In der JMSChatSender Klasse habe ich folgende Änderungen durchgeführt:

- **Methode randomGenerator(double low, double high):**
  Sie liefert mit random Werte zwischen den Parametern high und low. Somit wird erreicht, dass nicht immer die selben Werte generiert werden.
- **Methode RandomXML(windparkID, anzahlWindRaeder):**
  Sie erstellt mir einen XML String, dabei wird die Methode randomGenerator() für die Werte verwendet. Die Methode erstellt mir die Daten eines zufälligen Windparks mit einer zufälligen Anzahl von Windrädern. Zusätzlich wird noch ein Timestamp hinzugefügt.

```
1        public static String randomXML(int windparkId, int anzahlWindRaeder) {
2            Timestamp timestamp = new Timestamp(System.currentTimeMillis());
3            //Gibt die Zeit dazu, damit man später besser nachvollziehen kann von wann
             ↪  die Nachricht gekommen ist.
4            String xml = "\n<!-- Data sent at "+timestamp+" -->\n";
5            xml += "<windpark id=\"NOE" + windparkId + "\">";
6            for (int i = 1; i <= anzahlWindRaeder; i++) {
7                xml += "\n\t<windrad id=\"" + i + "\">\n\t\t<power unit=\"kWh\">" +
                 ↪  randomGenerator(100, 500)
8                + "</power>\n\t\t<blindpower unit=\"kWh\">" + randomGenerator(100, 500)
                 ↪  + "</blindpower>" + "\n\t\t<windspeed unit=\"km/h\">" +
                 ↪  randomGenerator(0, 100) + "</windspeed>\n\t\t<amk unit=\"m/s\">" +
                 ↪  randomGenerator(0.1, 0.7) + "</amk>\n\t\t<temperature unit=\"C\">"
                 ↪  + randomGenerator(10, 35) + "</temperature>"
9                + "\n\t\t<bladeposition unit=\"deg\">" + randomGenerator(0, 360) +
                 ↪  "</bladeposition>\n\t\t<transfertime unit=\"ms\">" +
                 ↪  randomGenerator(1, 3000) + "</transfertime>\n\t</windrad>\n";
10           }
11           xml += "</windpark>\n";
12
13           return xml;
14       }
```

- **Main Methode:**
  In der Main Methode habe ich nur das Topic gegen eine Queue getauscht und die vorher genannte Methode aufgerufen.

```
1        destination = session.createQueue(subject);
2
3        // Create the message
4        int randomParkID = (int)(Math.random()*10+1);
5        int randomWindRaeder = (int)(Math.random()*7+1);
6        TextMessage message = session.createTextMessage(randomXML(randomParkID,
         ↪  randomWindRaeder));
7        producer.send(message);
```

4.2 Zentralrechner (Receiver)

Die JMSChatReceiver Klasse ist dafür zuständig, die Nachrichten aus der Queue vom Server ausgelesen und in ein XML File geschrieben werden. Hierzu habe ich folgende Änderungen vorgenommen:

- **Methode writeToXML(str):**
  Sie wird verwendet um den erhaltenen XML String in das XML File dazuzuhängen.

```
1     public static void writeToXML(String str) throws IOException {
2         System.out.println(str);
3
4         File yourFile = new File("output.xml");
5         yourFile.createNewFile();
6         FileOutputStream outputStream = new FileOutputStream(yourFile, true);
7         byte[] strToBytes = str.getBytes();
8         outputStream.write(strToBytes);
9
10        outputStream.close();
11    }
```

- **Änderungen in der Main Methode:**
  Nachfolgendes habe ich in die Main-Methode angehängt. Der Receiver läuft solange, bis man ihn mit STRG+C beendet. Alternativ könnte man hier mit Threads arbeiten, der eine der Received und der andere der einfach eine gewisse Zeit läuft und dann den Receiver schließt und immer wenn eine weitere Nachricht ankommt wird der Counter zurück gesetzt. Somit kann man verhindern, dass der Receiver ewig läuft.

```
1     // Start receiving
2     while (true) {
3         TextMessage message = (TextMessage) consumer.receive();
4         Timestamp timestamp = new Timestamp(System.currentTimeMillis());
5         writeToXML(message.getText());
6
7         if (message != null) {
8             message.acknowledge();
9         }
10
11    }
```

5 Fragen

Nennen Sie mindestens 4 Eigenschaften der Message Oriented Middleware.

- Parallele Verarbeitung von Nachrichten möglich
- Asynchrone/Synchrone Kommunikation
- Lose Kopplung
- Der Receiver muss nicht sofort verfügbar sein.

Was versteht man unter einer transienten und synchronen Kommunikation?

- Transiente Kommunikation:
  Es erfolgt eine Übertragung nur dann, wenn sowohl Sender als auch Empfänger zum Zeitpunkt der Nachrichtenübermittlung aktiv sind. D.h. es werden die Nachrichten nur solange im Kommunikationssystem zwischenspeichert, solange beide Prozesse aktiv sind.
- Synchrone Kommunikation:
  Sender blockiert bis receive auf Empfängerseite erfolgt ist. Empfänger blockiert bis send auf Senderseite erfolgt ist.

Beschreiben Sie die Funktionsweise einer JMS Queue?
Die Nachricht wird am Server hinterlegt und von einem Client abgerufen.

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

- ConnectionFactory:
  Man meldet sich bei einem MOM-Service an.
- Connection:
  Es wird über die ConnectionFactory eine Verbindung mit dem MOM-Service hergestellt.
- Session:
  Diese ermöglicht die Kommunikation des Programmes mit dem Server
- MessageProducer:
  Erstellt eine Nachricht und fügt sie in die Queue hinzu. Die Nachricht braucht die Destination als Attribut.
- MessageConsumer:
  Empfänger der Nachricht.

Beschreiben Sie die Funktionsweise eines JMS-Topic?
Ein Topic schickt Nachrichten über einen, in der Session festgelegten Begriff, und lässt nur Sender und Receiver dieses Topics miteinander kommunizieren.

Was versteht man unter einem loose gekoppelten verteilten System? Nennen Sie ein Beispiel dazu. Warum spricht man hier von lose?
Lose gekoppeltes Systen, weil das einzige was ich wissen muss ist, wie ich mich mit der Middleware verbinden kann.