Übung 3: SOAP Implementierung

# Grundidee

Die dritte Übung basiert mehr oder weniger auf den Prinzipien der ersten beiden Übungen:

1. Das Interface soll komplett objektbasiert gestaltet werden
2. Die Übertragung der Befehle soll mit einem Serializer und Deserializer über ein Netzwerk erfolgen. Es soll also kein zeichen- oder tokenbasiertes Protokoll erbastelt werden, da dieses schlecht wartbar, fehleranfällig und zu oft zu unsicher und nicht robust genug implementiert wird

Da ich aufgrund der Betreuung unseres Lehrlings für seine Lehrabschlussarbeit (IPA) sehr wenig Zeit aufwenden kann, habe ich den Code aus der zweiten Übung übernommen und rasch angepasst.

Dies führt dazu, dass zwar eine SOAP Schnittstelle verwendet wird, meine Lösung aber alles andere alles als SOAP mässig konzipiert ist (Ich serialisiere Objekte und versende sie via SOAP). Es gibt also einen einzigen Endpoint, was nicht der Grundidee von SOAP entspricht.

Für die vierte Übung möchte ich aber wieder Vollgas geben und diese konzeptkonform umsetzen.

# Implementierung

Die Implementierung der dritten Übung basiert auf der zweiten Übung – die Objekte werden jetzt einzig via SOAP versendet ((De)Serialisiert als Bytearray, da ich die Grundentitäten nicht ändern wollte).

Die konkrete Implementierung des Drivers sieht also wie folgt aus:

|  |
| --- |
| package ch.fhnw.vesys.shared.soap;  import ch.fhnw.vesys.shared.core.Driver;  import ch.fhnw.vesys.shared.core.Sender;  import ch.fhnw.vesys.shared.core.Task;  import javax.xml.namespace.QName;  import javax.xml.ws.Service;  import java.net.URL;  public class SoapDriver extends Driver {  private static final SoapSender sender = new SoapSender();  public SoapDriver() {  super(sender);  }  @Override  public void connect(String[] args) {  SoapSender.hostname = args[0];  SoapSender.port = Integer.parseInt(args[1]);  System.out.println("Connected to soap driver.");  }  @Override  public void disconnect() {  System.out.println("Disconnected from soap driver");  }  private static class SoapSender implements Sender {  static String hostname;  static int port;  @Override  public Task sendTask(Task task) {  try {  URL wsdlurl = new URL("http://" + hostname + ":" + port + "/task?wsdl");  QName qname = new QName("http://soap.shared.vesys.fhnw.ch/", "TaskServiceImplService");  Service service = Service.create(wsdlurl, qname);  TaskService taskservice = service.getPort(TaskService.class);  return TaskServiceImpl.fromByteArrayToTask(taskservice.handleTask(TaskServiceImpl.fromTaskToByteArray(task)));  } catch (Exception exception) {  exception.printStackTrace();  return new Task.InvalidTask();  }  }  }  } |

Dabei wird auf den SOAP Serviceinterface und die konkrete Implementierung für die Tasks zurückgegriffen (Dieser wird als Endpoint durch den Server geladen):

|  |
| --- |
| package ch.fhnw.vesys.shared.soap;  import javax.jws.WebMethod;  import javax.jws.WebService;  import javax.jws.soap.SOAPBinding;  @WebService  @SOAPBinding(style = SOAPBinding.Style.RPC)  public interface TaskService {  @WebMethod  public byte[] handleTask(byte[] data) throws Exception;  } |

Dazu die Implementierung:

|  |
| --- |
| package ch.fhnw.vesys.shared.soap;  import ch.fhnw.vesys.shared.api.BankDriver;  import ch.fhnw.vesys.shared.core.Task;  import ch.fhnw.vesys.shared.local.LocalDriver;  import javax.jws.WebService;  import java.io.\*;  @WebService(endpointInterface = "ch.fhnw.vesys.shared.soap.TaskService")  public class TaskServiceImpl implements TaskService {  private final BankDriver bankdriver = new LocalDriver();  @Override  public byte[] handleTask(byte[] data) throws Exception {  Task task = fromByteArrayToTask(data);  task.executeHandledTask(bankdriver);  return fromTaskToByteArray(task);  }  static byte[] fromTaskToByteArray(Task task) throws IOException {  ByteArrayOutputStream byteoutputstream = new ByteArrayOutputStream();  ObjectOutputStream objectoutputstream = new ObjectOutputStream(byteoutputstream);  objectoutputstream.writeObject(task);  byte[] data = byteoutputstream.toByteArray();  objectoutputstream.close();  byteoutputstream.close();  return data;  }  static Task fromByteArrayToTask(byte[] data) throws Exception {  ByteArrayInputStream byteinputstream = new ByteArrayInputStream(data);  ObjectInputStream inputstream = new ObjectInputStream((byteinputstream));  Task task = (Task) inputstream.readObject();  inputstream.close();  byteinputstream.close();  return task;  }  } |

Der restliche Code ist dem Projekt zu entnehmen.

# Verbesserungsmöglichkeiten

Wie bereits eingangs erwähnt, entspricht meine Umsetzung nicht dem SOAP Gedanken, system- und implementationsunabhängige Schnittstellen zu verwenden – zeitbedingt war mir dieser aber nur bedingt möglich.

Im Hinblick auf die kommende vierte Übung muss ich mir also überlegen, ob ich der Eleganz halber nicht für jede Aktion eine kurze Implementierung neu schreiben möchte. Dies würde zu etwas Mehraufwand führen, das Ganze aber viel konformer und übersichtlicher gestalten.

# Source Code

Der Source Code ist hier zu finden: <https://github.com/swaechter/fhnw/tree/master/Module/vesys/bank>