UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Studienarbeit

Prototypisierte Implementation des Sticky-Logging-Formalismus und Integration in JBoss

vorgelegt von:
Martin Schnorr Matr.-Nr.: 204 210 011

Betreuer:
Prof. Dr. Steffen Staab
Dipl.-Inform. Christoph Ringelstein

Koblenz, den 30. Mai 2008

Abstract

In dieser Studienarbeit werden neben den Grundlagen der Web Services, Komponenten und APIs zur Realisierung des Sticky Loggings aufgezeigt. Es wird ein Szenario zum Testen des Sticky Loggings beschrieben und als Web Services implementiert. Der Sticky-Logging-Formalismus wird erklärt und es wird eine API zur Erstellung der StickyLogs implementiert. Die StickyLogs werden innerhalb des SOAP-Attachments der SOAP-Nachrichten zwischen den Web Services ausgetauscht. Dazu wird eine Realisierung mit einem Messagehandler unter JAX-WS programmiert und erläutert.

Gliederung

1. Einleitung **5**

2. Grundlagen **6**

2.1. Service-orientierte Architektur 6

2.2. Web-Service-Architektur 7

2.2.1. SOAP 8

2.2.2. WSDL 12

2.2.3. UDDI 12

3. Komponenten zur Implementation eines Szenarios und des Sticky Loggings **14**

3.1. Server: JBoss AS 14

3.2. Entwicklungsumgebung: Eclipse WTP 14

3.3. Tool: wsimport 15

3.4. Verwendete APIs 15

3.4.1. JAX-WS 15

3.4.2. Jena 17

4. Szenario **18**

4.1. Beschreibung des Szenarios 18

4.2. Szenario als Web Services 19

4.2.1. Akteure 19

4.2.2. Datenströme 20

4.2.3. Erläuterung der Webmethoden 21

4.3. Implementation des Szenarios 22

5. Sticky Logging **24**

5.1. Beschreibung des Sticky-Logging-Formalismus 24

5.2. Implementation des Sticky-Logging-Formalismus 27

5.2.1. Implementation einer Sticky Logging API zur Erstellung der Sticky Logs 27

5.2.2. Austausch der Sticky Logs mit Hilfe der SOAP-Nachrichten 29

5.2.3. Einsatz des Sticky Loggings am Beispiel des Szenarios 29

6. Zusammenfassung **31**

Literaturverzeichnis **32**

A. Quellcode **33**

A.1. Implementation des Szenarios 33

A.1.1. Web Service LogisticInc 33
A.1.2. Web Service WebShop 34
A.1.3. Web Service Client Customer 36
A.2. Implementation der Sticky Logging API 36
A.3. Messagehandler zur Modifikation von ein- und ausgehenden SOAP-Nachrichten 45
A.4. Einbinden des Messagehandlers 50

B. Consolenausgaben **52**
B.1. Ausgabe beim Client 52
B.2. Ausgabe beim Server 53

Abbildungsverzeichnis

1. SOA Tempel (Quelle: nach [WS05]) 7
2. Web-Service-Dreieck (Quelle: nach [WS05]) 8
3. Web Service Stack (Quelle: [WA04] 9
4. Aufbau einer SOAP-Nachricht (Quelle: nach [WS05]) 10
5. Übersichtsgrafik zu JAX-WS (Quelle: siehe Fußnote zu JAX-WS) 16
6. Darstellung eines RDF-Graphen (Quelle: siehe Fußnote zur Jena-API) . . . 17
7. Zusammensetzen (Mergen) von 2 Modellen (Quelle: siehe Fußnote zum Jena-Tutorial) 18
8. Szenario eines Bestellprozesses mit Übermittlung personenbezogener Daten von einem Webshop zu einem Logistikdienstleister 19
9. Grafik zur Verdeutlichung der Datenströme 21
10. UML-Klassendiagramm zur Sticky Logging API 28
11. Sequenzdiagramm zur Verdeutlichung der Reihenfolge der Messagehandleraufrufe 30

1. Einleitung

In dieser Studienarbeit wurde ein Prototyp zum Sticky-Logging-Formalismus implementiert. Dazu folgt zunächst in Kapitel 2 eine umfassende Beschreibung zu Grundlagen der Web Services, um die Begriffe, die im Zusammenhang mit dieser Studienarbeit stehen, zu erläutern. Kapitel 3 stellt die Komponenten, die zur Umsetzung eines Szenarios und des Sticky Logging benötigt werden, dar. Dazu zählen ein Server, eine Entwicklungsumgebung und ein Tool zur automatischen Generierung der Ports zum Ansprechen der Web Services. Es findet eine Beschreibung der verwendeteten APIs JAX-WS und Jena statt. In Kapitel 4 wird ein Szenario eingeführt, mit dessen Hilfe die Funktionalität des Sticky-Logging-Prototyps überprüft wird. Die Umsetzung des Szenarios als Web Services mit den auftretenden Akteuren, Datenströmen und Webmethoden wird beschrieben. Es folgt die Erläuterung der Implementation des Szenarios. In Kapitel 5 wird der Formalismus des Sticky Loggings mit seinen Konzepten erklärt und die Implemenation aufgeführt, sowie der Austausch der Sticky Logs anhand des Szenarios demonstriert. Die Studienarbeit wird mit einer Zusammenfassung abgeschlossen.

2. Grundlagen

2.1. Service-orientierte Architektur

> „Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“ [WS05]

Dostal et. al. stellen in ihrem Buch „Service-orientierte Architekturen mit Web Services“ (siehe Literaturverzeichnis unter [WS05]) die wesentlichen Merkmale einer service-orientierten Architektur als Tempel dar (siehe Abbildung 1). Anhand des Tempelaufbaus wird hier der Begriff „Service-orientierte Architektur“ erklärt. Die Basis der SOA wird von offenen Standards, Sicherheit und Einfachheit gebildet. Um Dienste anderer Anbieter komfortabel nutzen zu können, ist es wichtig, dass **offene Standards** verwendet werden. So kann der Nutzer den Dienst leichter verstehen. Der Begriff **Einfachheit** drückt sich durch die leichte Wiederverwendung eines Dienstes in verschiedenen Umgebungen aus. Eine hohe **Sicherheit** kann durch verschiedene Verfahren wie z.B. Verschlüsselung und digitale Signatur erreicht werden. Die Säulen zeigen folgende Merkmale einer service-orientierten Architektur: Verteiltheit, Lose Kopplung, Verzeichnisdienst und Prozessorientiert. Dienste befinden sich an verschiedenen Stellen, daher der Begriff **Verteiltheit**. Unter **Loser Kopplung** versteht man das Suchen, Finden und Einbinden von Diensten zur Laufzeit. Das heißt, Dienste werden dynamisch verwendet. Dies wird auch **dynamisches Binden** genannt. Ein weiterer Begriff ist der **Verzeichnisdienst**. Er entspricht einem Telefonbuch oder den Gelben Seiten. In dem Verzeichnisdienst wird nach registrierten Diensten bzw. Methoden gesucht. Fehlt noch die Säule **Prozessorientiert**. Mit service-orientierten Architekturen lassen sich hervorragend Geschäftsprozesse, die von externen Ereignissen (Events) beeinflusst bzw. gesteuert werden, realisieren (z.B. das Überweisen eines Geldbetrages von einem Konto auf ein Zielkonto, welches unter einem gewissen Betrag gefallen ist). Man spricht in diesem Zusammenhang auch von ereignisgesteuerter Architektur (engl. „Event-driven Architecture“).

Die grundlegenden Komponenten einer service-orientierten Architektur sind

- Kommunikation,
- Dienstbeschreibung und
- Verzeichnisdienst.

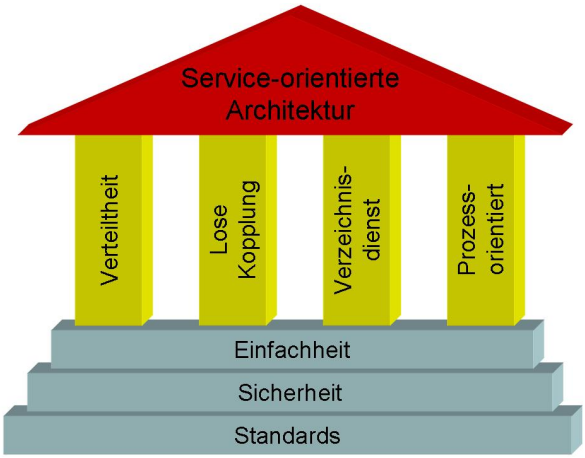

Abbildung 1: SOA Tempel (Quelle: nach [WS05])

2.2. Web-Service-Architektur

Eine bekannte Ausprägung von service-orientierten Architekturen ist die Web-Service-Architektur. Die Komponenten einer Web-Service-Architektur werden durch folgende Spezifikationen beschrieben:

- Kommunikation: **SOAP**
- Dienstbeschreibung: **WSDL**
- Verzeichnisdienst: **UDDI**

Abbildung 2 zeigt den Ablauf einer Dienstnutzung mit den benötigten Komponenten:

1. Ein Dienstanbieter veröffentlicht einen Dienst, indem er an einen UDDI-basierten Dienstverzeichnis ein WSDL-Dokument übermittelt, welches die Schnittstellenbeschreibung in XML-Darstellung enthält.
2. Eine Dienstnutzer sucht nun nach diesem Dienst in dem Verzeichnisdienst. Der Verzeichnisdienst stellt hierfür eine SOAP-Schnittstelle zur Verfügung. Findet er den Dienst, fordert er vom Verzeichnisdienst den Aufenthaltsort des WSDL-Dokumentes an.
3. Der Verzeichnisdienst liefert eine Referenz in Form eines URI[1] auf das WSDL-Dokument.
4. Der Dienstnutzer fordert mit diesem URI das WSDL-Dokument beim Dienstanbieter an.
5. Jetzt können Dienstnutzer und Dienstanbieter mit Hilfe der WSDL-Beschreibung mittels SOAP kommunizieren.

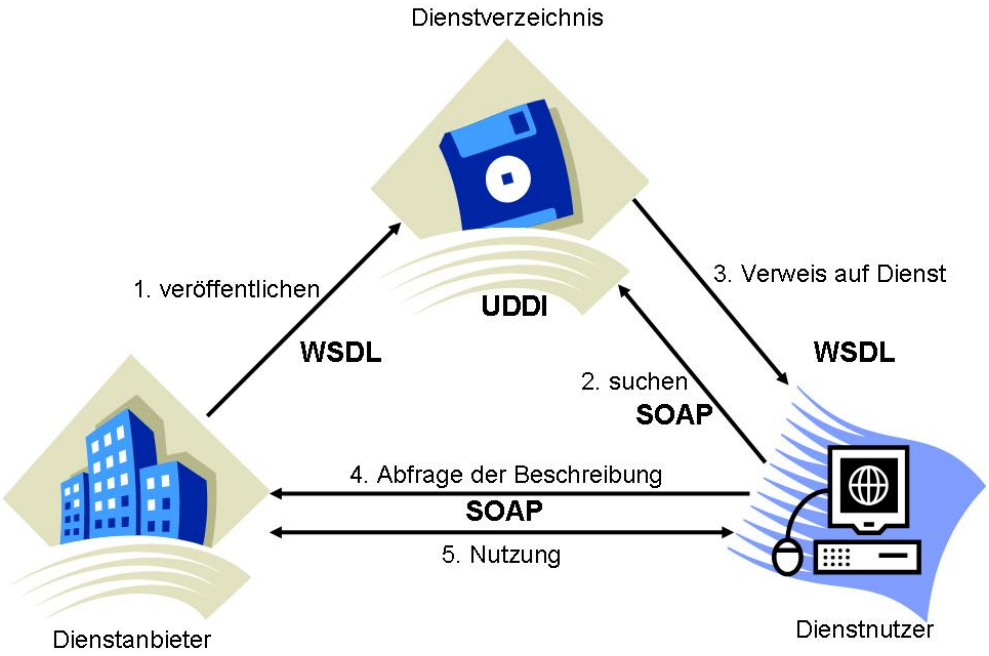

Abbildung 2: Web-Service-Dreieck (Quelle: nach [WS05])

Abbildung 3 zeigt einen Web Service Stack. Ein Web Service Stack ist ein Schichtenmodell, welches die Anforderungen, die an eine SOA gestellt werden, beschreibt. Es gibt verschiedene Varianten von Web Services Stacks. Das hier dargestellte Schichtenmodell stammt vom „World Wide Web Consortium“ (W3C). Die unterste Ebene bildet die Transportschicht. Web Services können unterschiedliche Transportprotokolle verwenden (HTTP, SMTP, ...), um XML-Nachrichten auszutauschen. Als Grundlage eines gemeinsamen Datenmodells dienen XML, Schemataabeschreibungen und Namensräume. In der nächsten Schicht befinden sich die SOAP-Spezifikationen. Sie beschreiben Syntax und Semantik einer XML-Nachricht. SOAP besitzt einige Erweiterungsmöglichkeiten (engl. SOAP Extensions). Diese können für eine Vielzahl an Funktionalitäten genutzt werden, z.B. zur Realisierung von Sicherheitskonzepten wie Verschlüsselung und digitaler Signatur. Aufbauend auf dieser Schicht ist die Beschreibung des Web Services mit Hilfe von WSDL zu finden. Den Abschluss bildet die Prozessschicht. Hier werden Prozesse modelliert und koordiniert.

2.2.1. SOAP

Das **Simple Object Access Protocol** (SOAP) ist ein auf XML basierendes Protokoll, welches die Interaktion bei Web Services regelt. Die SOAP-Spezifikation vom W3C legt lediglich fest, wie eine Nachricht aufgebaut sein muss. Eine konkrete Implementati-

[1]URI = Uniform Resource Identifier (dt. „einheitlicher Bezeichner für Ressourcen“

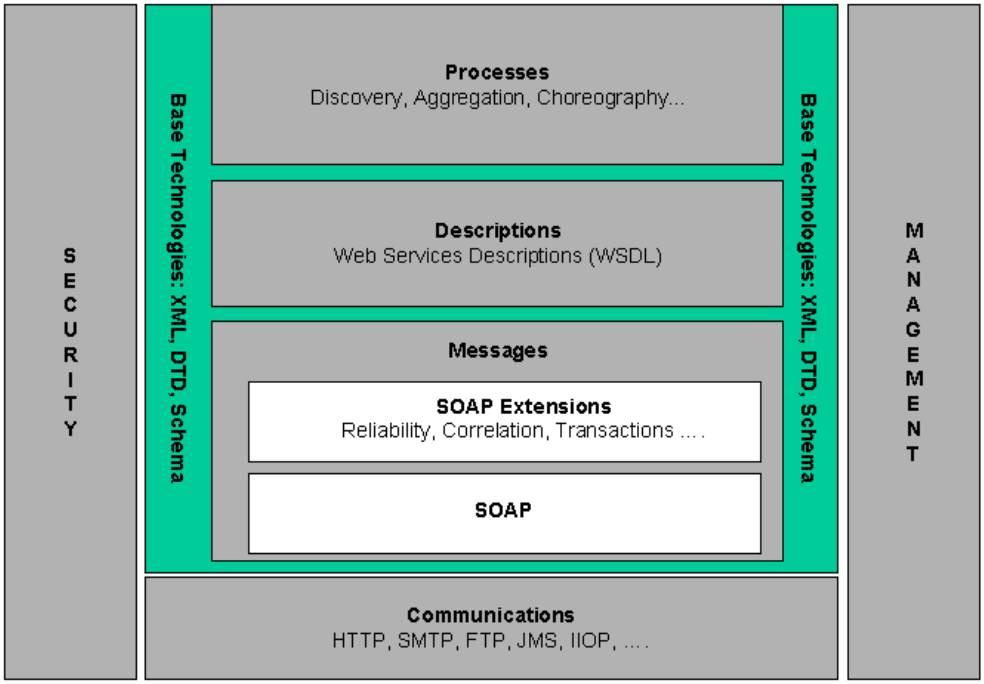

Abbildung 3: Web Service Stack (Quelle: [WA04]

on wird nicht vorgeschrieben, sondern bleibt dem Nutzer selbst überlassen. Die SOAP-Spezifikation liegt momentan in Version 1.2 vor.[2] Die SOAP-Spezifikation ist unterteilt in vier Dokumente:

1. **Part 0 (Primer):** Dieses Dokument entspricht einer Einführung in die SOAP Spezifikation der Version 1.2 mit vielen technischen Beispielen (vgl. [S007]).

2. **Part 1 (Messaging Framework):** Im Teil 1 der Spezifikation werden Elemente einer Nachricht beschrieben (vgl. [S107]).

3. **Part 2 (Adjuncts):** Im Teil 2 der Spezifikation wird ein Datenmodell, ein Kodierungsschema für „remote procedure calls“ (RPC) und die Anbindung an das Transportprotokoll HTTP beschrieben (vgl. [S207]).

4. **Specification Assertions and Test Collection:** In dieser ergänzenden Spezifikation werden die Zusicherungen aus Teil 1 und 2 noch einmal aufgelistet. Danach wird eine Test Collection aufgestellt, um die zuvor aufgestellten Zusicherungen zu überprüfen (vgl. [S307]).

In „SOAP Version 1.2 Part 0: Primer (Second Edition)“ wird der Aufbau einer SOAP-Nachricht beschrieben. Der Inhalt einer SOAP-Nachricht ist nicht Teil der SOAP Spezifikation. Er wird in begleitenden Spezifikationen z.B. „Resource Representation Header

[2]Website des W3C zu SOAP: http://www.w3.org/TR/soap/

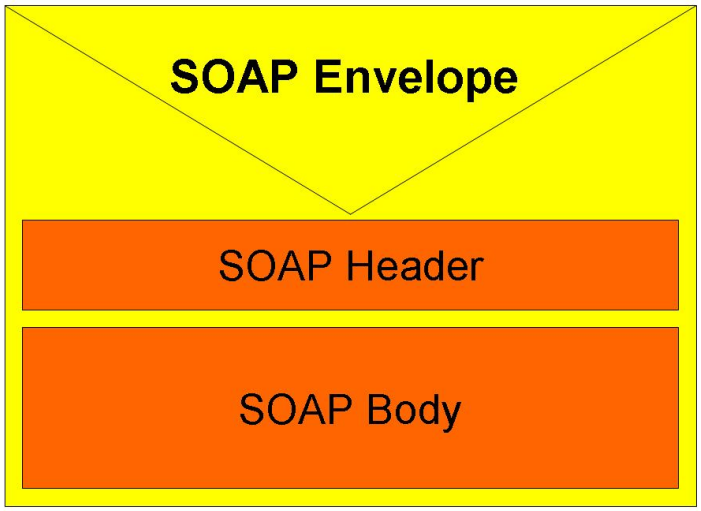

Abbildung 4: Aufbau einer SOAP-Nachricht (Quelle: nach [WS05])

Block“ beschrieben. Abbildung 4 zeigt die drei Teile, aus denen eine SOAP-Nachricht besteht:

- SOAP Envelope
- SOAP Header
- SOAP Body

Der **SOAP Envelope** entspricht einem Briefumschlag, der den eigentlichen Brief enthält. Er bildet das Wurzelelement eines XML-Dokumentes. Listing 1 zeigt den generellen Aufbau eines SOAP Envelope.

Listing 1: Aufbau eines SOAP Envelope (Quelle: [WS05])

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <!-- SOAP Header -->
   <!-- SOAP Body -->
</env:Envelope>
```

Mit der Angabe des URI http://www.w3.org/2003/05/soap-envelope wird bestimmt, welche Version der SOAP-Spezifikation verwendet wird. Außerdem wird der Namensraum (engl. namespace) für das Präfix *env* festgelegt. Der **SOAP Header** ist ein optionales Element. Er kann nur genau einmal, und dann als erstes Kindelement vom SOAP Envelope, vorkommen. SOAP-Nachrichten können auch über Zwischenstationen vom Sender zum Empfänger gelangen. Dazu ist es erforderlich, dass die Zwischenstation den SOAP Header lesen kann. Listing 2 zeigt ein Beispiel zu einem SOAP Header.

Listing 2: Aufbau eines SOAP Header (Quelle: [S107])

```
<env:Header>
```

```
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
```

Das Beispiel zeigt den SOAP Header einer Flugreservierung. Es gibt hier zwei SOAP Header Blöcke: *reservation* und *passenger*, die spezielle Attribute haben. Für jedes Kindelement des SOAP Headers können Attribute zur Angabe der Verarbeitung eines Header-Elementes gemacht werden:

- **encodingStyle:** Hier können Serializations-Regeln für SOAP-Nachrichten angegeben werden.
- **role:** Mit diesem Attribut wird der Empfänger bzw. die Zwischenstation, die dieses Header-Element verarbeiten darf, spezifiziert.
- **mustUnderstand:** Dieses optionale Attribut kann nur die Werte *true* und *false* annehmen. Bei *true* muss der Knoten (Empfänger oder Zwischenstation) den Header-Eintrag verarbeiten können, sonst wird eine Fehlermeldung zurück zum Absender geschickt und die weitere Bearbeitung der SOAP-Nachricht abgebrochen. Ist das Attribut auf *false* gesetzt, so ist die Auswertung des Kindelementes optional.
- **relay:** Dieses optionale Attribut leitet bei *true* den Headerblock weiter.

Das zweite Kindelement des SOAP Envelope ist obligatorisch: der **SOAP Body**. Der SOAP Body enthält die eigentlichen Nutzdaten einer SOAP-Nachricht. Der SOAP Body muss ein wohl geformtes XML-Dokument sein. Listing 3 zeigt ein Beispiel zu einem SOAP Body.

Listing 3: Aufbau eines SOAP Body (Quelle: [S107])

```
<env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
```

```
    <p:arriving>New York</p:arriving>
    <p:departureDate>2001-12-20</p:departureDate>
    <p:departureTime>mid-morning</p:departureTime>
    <p:seatPreference/>
  </p:return>
 </p:itinerary>
 <q:lodging
  xmlns:q="http://travelcompany.example.org/reservation/hotels">
  <q:preference>none</q:preference>
 </q:lodging>
</env:Body>
```

Das Beispiel zeigt den SOAP Body einer Flugreservierung. Hier sind die eigentlichen Daten zu finden. Die Bedeutung der Daten (Semantik) wird, wie oben bereits erwähnt, vom SOAP nicht berücksichtigt, sondern muss über zusätzliche Spezifikationen beschrieben werden. Jedem Element muss ein eindeutiger Namensraum zugewiesen werden (ein URI).

2.2.2. WSDL

Eine weitere Komponente von Service-orientierten-Architekturen ist die Dienstbeschreibung. Diese wird bei Web-Services-Architekturen mit der **Web Service Description Language** (WSDL) spezifiziert. WSDL enthält ein XML-Vokabular, welches die Schnittstelle eines Dienstes mit seinen Operationen beschreibt. Ein WSDL-Dokument enthält folgende Komponenten:

- abstrakter Schnittstellenteil
  - **Types**: Abstrakte Beschreibung von komplexeren Datentypen
  - **Message**: Abstrakte Beschreibung der Nachricht beim Kommunikationsaustausch
  - **Operation**: Abstrakte Beschreibung der Web-Services-Funktionen
  - **Port Type**: Menge von abstrakten Endpunkten
- konkreter Implementierungsteil
  - **Binding**: Festlegung eines Netzwerkprotokolls und eines Datenformats für einen abstrakten Endpunkt (Port Type)
  - **Port**: Kombination eines Bindings und einer Netzwerkadresse
  - **Service**: Zusammenfassung mehrerer Ports

Weitere Informationen siehe: http://www.w3.org/TR/wsdl.

2.2.3. UDDI

Die dritte Komponente einer Service-orientierten-Architektur ist der Verzeichnisdienst. Dieser wird bei Web-Services-Architekturen mit **Universal Description, Discovery**

and Integration (UDDI) spezifiziert. UDDI ist ein XML-basiertes Protokoll zum Aufsetzen eines Verzeichnisdienstes. Ein Service-Anbieter kann seinen Web-Service auf einem Verzeichnisdienst registrieren, wo er dann von Service-Konsumenten gefunden werden kann.

3. Komponenten zur Implementation eines Szenarios und des Sticky Loggings

Am Anfang der Studienarbeit war zu klären, welche Komponenten zur Umsetzung des Prototypen benötigt werden. Soll kommerzielle oder Open Source Software eingesetzt werden? Es wurde sich für Open Source Software entschieden. Eine bekannte Serversoftware auf diesem Gebiet ist der JBoss Application Server (JBoss AS). Zur Umsetzung des Szenarios (siehe Kapitel 4) und Realisierung des Sticky Loggings wurde mit der Entwicklungsumgebung Eclipse WTP gearbeitet. Zur automatischen Generierung der Ports aus den WSDL-Dokumenten zum Ansprechen der Web Services wurde das Tool wsimport eingesetzt.

3.1. Server: JBoss AS

JBoss bietet eine große Auswahl an Softwareprodukten an.[3] Zur Umsetzug des Prototyps kam als Server der „JBoss Application Server“ (JBoss AS) zum Einsatz.[4] JBoss AS ist ein Open Source Java 2 Enterprise Edition (J2EE) Applikationsserver. JBoss AS besitzt als Fundament eine serviceorientierte Architektur (SOA). In JBoss AS GA ist das Web Service Framework „JBoss Web Services“ (JBoss WS) integriert. JBoss WS implementiert die JAX-WS Spezifikationen (siehe Kapitel 3.4.1), die ein Programmmodell und eine Runtime-Architektur definieren, um Web Services in Java zu implementieren. Der Download des JBoss Application Server **JBoss-4.2.2.GA** ist hier zu finden: (siehe Fußnote[5]). Bevor man den JBoss AS benutzen kann, ist das Java JDK 5.0 zu installieren. Danach muss noch eine Umgebungsvariable **JAVA_HOME** gesetzt werden.[6] Mit der Adresse http://localhost:8080/ kann der gestartete Server angesprochen werden.

3.2. Entwicklungsumgebung: Eclipse WTP

Das Eclipse Web Tools Platform (WTP) Projekt[7] erweitert die Eclipse Plattform um Tools für die Entwicklung von Web und Java EE Applikationen. Eclipse WTP umfasst Source- und graphische Editoren für eine große Anzahl von Sprachen, Wizards und Built-in Applikationen um die Entwicklung zu vereinfachen. Es umfasst des Weiteren Tools und APIs zur unterstützenden Erstellung, Ausführung und Tests von Applikationen. Eclipse WTP kann als Standaloneversion[8] oder als Update einer bereits bestehenden Eclipsever-

[3] http://labs.jboss.com/resources/

[4] „Documentation Library:“ http://labs.jboss.com/jbossas/docs/index.html

[5] Download des JBoss AS: http://labs.jboss.com/downloading/?projectId=jbossas&url=http://sourceforge.net/project/showfiles.php?group_id=22866&package_id=16942&release_id=548923

[6] Weitere Details zu Umgebungsvariablen siehe: http://labs.jboss.com/file-access/default/members/jbossas/freezone/docs/Installation_Guide/beta422/pdf/Installation_Guide.pdf

[7] Webseite zu WTP: http://www.eclipse.org/webtools/

[8] Download von Eclipse WTP: http://download.eclipse.org/webtools/downloads/

sion[9] installiert werden. In Eclipse WTP kann man (nach erfolgreicher Einrichtung) den JBoss Application Server per Knopfdruck starten und ihm Projekte zuweisen.

3.3. Tool: wsimport

Das Tool **wsimport** von Sun Microsystems generiert aus WSDL-Dateien die notwendigen Javaklassen zum Ansprechen der Webservices. Das Hilfsprogramm wsimport ist bei der JDK 6.0 - Installation vorhanden. Ein Beispielbefehl, welcher auch unter Kapitel 4.3 zu finden ist, sieht folgendermaßen aus: **wsimport -d src -keep -p de.unikoblenz.schnorr** http://127.0.0.1:8080/LogisticInc/Delivery?wsdl. Mit diesem Befehl werden unter anderem die Javaklassen Delivery.java und DeliveryService.java im Ordner *src/de/unikoblenz/schnorr* erzeugt[10]. Diese können dann in das Projekt übernommen werden. Eine Dokumentation der verschiedenen Operationen und Parameter ist hier zu sehen: http://java.sun.com/javase/6/docs/technotes/tools/share/wsimport.html.

3.4. Verwendete APIs

Die Web Services wurden mit JAX-WS realisiert. Zur Erzeugung und Modifikation der RDFs kam die API Jena zum Einsatz.

3.4.1. JAX-WS

JAX-WS (Java API for XML - Web Services) ist ein Framework zum Erstellen von Web Services. Abbildung 5 stellt die Bestandteile, die zu JAX-WS gehören, dar. Tools sind hier z.B. wsimport. APT ist ein Javaprogramm, um Annotationen in Javaprogrammen einzufügen. Der Annotation Processor übersetzt mit Hilfe der JAX-WS-Annotationen Java-Code in Web Services [11].

Zur Realisierung der Web Services (siehe Kapitel 4.3) werden Annotationen eingesetzt, die mit Java 1.5 bzw. Java 5.0 eingeführt wurden. Annotationen sind zusätzliche Daten oder Informationen zu einem Programm, die das Programm nicht direkt beeinflussen[12]. Andere Programme (z.B. Compiler) können mit den Annotationen arbeiten. Im Prototyp werden die JAX-WS-Annotationen @WebService,@SOAPBinding, @WebMethod und @WebParam verwendet.

- **@WebService:** Unter http://java.sun.com/javaee/5/docs/api/javax/jws/WebService.html wird die Verwendung der Annotation @WebService erläutert. Mit dieser Annotation wird eine Java-Klasse zu einem Web Service deklariert. Es kann ein Name

[9] Update einer vorhandenen Eclipse-Version: http://download.eclipse.org/webtools/updates/

[10] Weitere Beispiele zu wsimport sind hier zu finden: http://www.tutego.com/blog/javainsel/2006/04/insel-webservice-in-java-6-wsimport.html

[11] Quelle und weitere Informationen: https://jax-ws-architecture-document.dev.java.net/nonav/doc/?jaxws/package-summary.html

[12] Webseite zu Annotationen: http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html

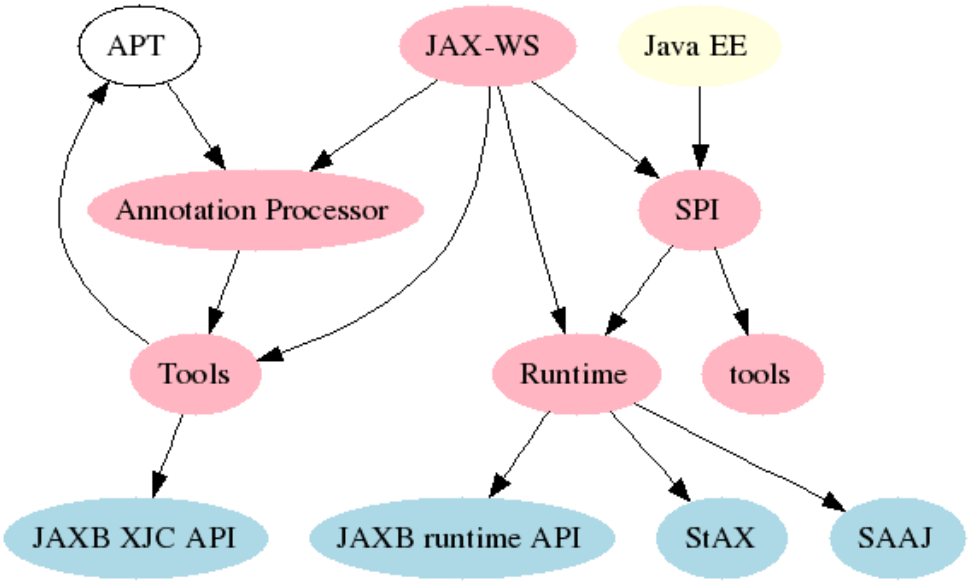

Abbildung 5: Übersichtsgrafik zu JAX-WS (Quelle: siehe Fußnote zu JAX-WS)

des Web Services (name) angegeben werden, der zur Beschreibung des Web Services im WSDL-Dokument Verwendung findet (bei WSDL 1.1: wsdl:portType).

- **@SOAPBinding:** Die Annotation @SOAPBinding wird unter http://java.sun.com/javaee/5/docs/api/javax/jws/soap/SOAPBinding.html beschrieben. Mit Hilfe dieser Annotation kann die Verwendung des SOAP Message Protokolls beeinflusst werden. Es gibt drei Parameter: style, use und parameterStyle. Der Parameter style definiert das Encoding der Nachrichten zum Senden und Empfangen für den Web Service. In der Studienarbeit wird der default-Wert SOAPBinding.Style.DOCUMENT verwendet. Der zweite Parameter use definiert die Formatierung der Nachrichten zum Senden und Empfangen für den Web Service. Auch hier wird der default-Wert SOAPBinding.Use.LITERAL verwendet. Der letzte Parameter parameterStyle legt fest, ob die Methodenparameter den ganzen Nachrichten-Body darstellen oder ob die Methodenparameter Elemente innerhalb eines Toplevel-Elements, benannt nach der Methode, sind. Auch hier wird wieder innerhalb der Studienarbeit der default-Wert SOAPBinding.ParameterStyle.WRAPPED verwendet.

- **@WebMethod:** Die Annotation @WebMethod wird unter http://java.sun.com/javaee/5/docs/api/javax/jws/WebMethod.html beschrieben. Mit dieser Annotation wird eine Methode als eine Webmethode zu dem Web Service deklariert.

- **@WebParam:** Die Annotation @WebParam wird unter http://java.sun.com/javaee/5/docs/api/javax/jws/WebParam.html beschrieben. In der Studienarbeit wird damit der Parametername (name) der Webmethoden festgelegt.

3.4.2. Jena

Jena[13] ist ein Framework, um Semantic Web Anwendungen zu entwickeln. Es unterstützt die Erstellung und Modifikation von RDFs mit Hilfe einer RDF-API. Das Resource Description Framework (RDF) ist eine Möglichkeit, Informationen (Resourcen) im Web darzustellen [RD04]. Dazu werden Triple gebildet, bestehend aus Subjekt, Prädikat und Objekt. Jedes Triple kann als Graph dargestellt werden. Der Startpunkt ist das Subjekt, die Beschriftung des Graphen entspricht dem Prädikat und der Endpunkt ist das Objekt. Abbildung 6 zeigt einen RDF-Graphen. Das Beispiel ist aus dem Tutorial zur Jena-API entnommen[14]. Das Subjekt wird mit seinem Uniform Resource Identifier (URI) identifiziert. Eine URI muss dabei keine real existiernde Internetadresse sein.

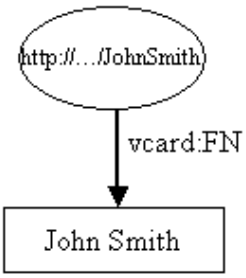

Abbildung 6: Darstellung eines RDF-Graphen (Quelle: siehe Fußnote zur Jena-API)

Jena unterstützt das Zusammensetzen (Mergen) von Modellen (zusammengesetzten RDF-Graphen). Abbildung 7 zeigt zwei Modelle, die mit der Funktion *model1.union(model2)* zu einem Modell zusammengesetzt werden. Listing 4 enthält den entsprechenden Java-Code. Die Modelle model1 und model2 werden hierbei über den InputStreamReader eingelesen.
Zur Realisierung des Prototyps wurde die Version Jena 2.5.5 verwendet.

Listing 4: Java-Code zum Mergen von Jena-Modellen

```
// read the RDF/XML files
model1.read(new InputStreamReader(in1), "");
model2.read(new InputStreamReader(in2), "");

// merge the Models
Model model = model1.union(model2);
```

[13]Jena Website: http://jena.sourceforge.net/index.html
[14]Tutorial zur Jena-API: http://jena.sourceforge.net/tutorial/RDF_API/index.html

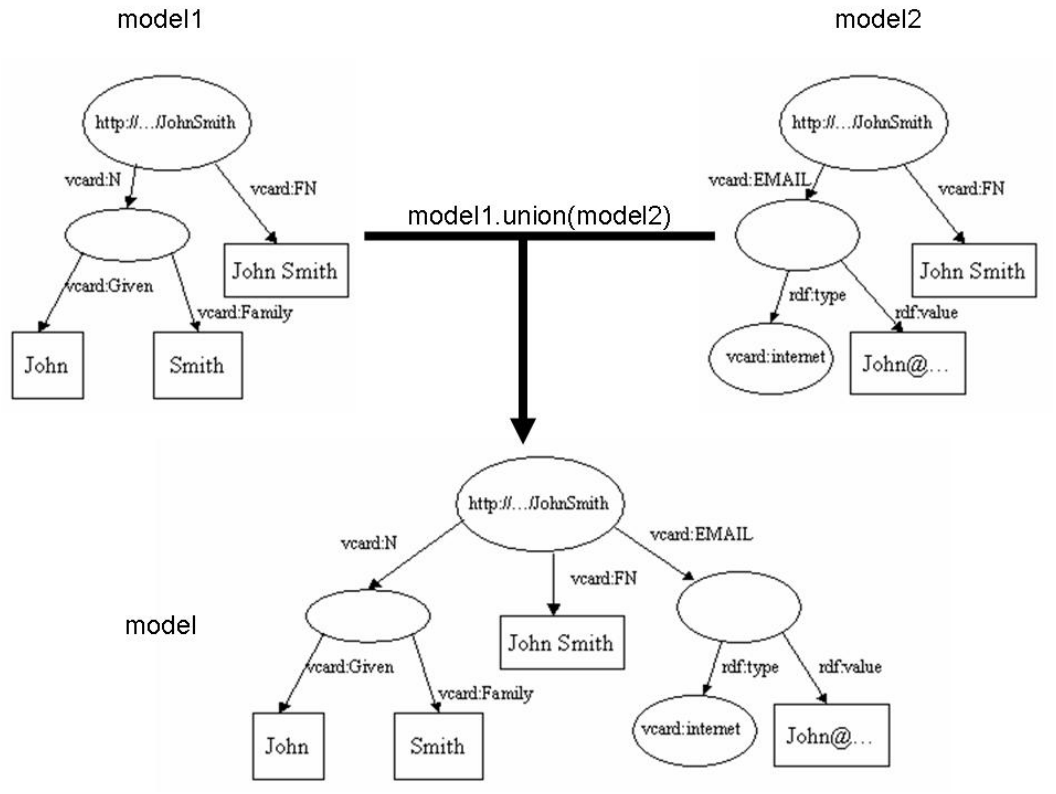

Abbildung 7: Zusammensetzen (Mergen) von 2 Modellen (Quelle: siehe Fußnote zum Jena-Tutorial)

4. Szenario

4.1. Beschreibung des Szenarios

Nach dem Bundesdatenschutzgesetz[15] hat jede natürliche Person das Recht, Auskunft über die Verwendung seiner personenbezogenen Daten von einer Organisation zu erhalten (**Recht auf informationelle Selbstbestimmung**). Bei verteilten Arbeitsprozessen ist dies für eine Organisation nicht immer einfach. Die Weitergabe der personenbezogenen Daten eines Betroffenen an externe Dienstleister (z.B. Logistikdienstleister) oder an andere interne Bereiche führt dazu, dass die Organisation, die die Daten erhoben hat, eventuell nur noch unter erheblichem Aufwand eine Angabe über die Weitergabe der personenbezogenen Daten machen kann.

Abbildung 8 zeigt ein Anwendungsbeispiel für den Sticky-Logging-Formalismus. Ein Webshop hat die Lagerung und Auslieferung seiner Produkte an einen Logistikdienstleister outgesourct. Bestellt nun ein Kunde beim Webshop Produkte, so wird seine Adresse an den Logistikdienstleister zur Abwicklung der Auslieferung übermittelt. Ein Kunde hat das Recht Informationen über die Verwendung seiner personenbezogenen Daten zu erhalten.

[15]BDSG §1, Abs. 1: „Zweck dieses Gesetzes ist es, den Einzelnen davor zu schützen, dass er durch den Umgang mit seinen personenbezogenen Daten in seinem Persönlichkeitsrecht beeinträchtigt wird.“

Mit Hilfe des Sticky Logging kann der Kunde nun diese Information bekommen. Der Prototyp wird dies demonstrieren.

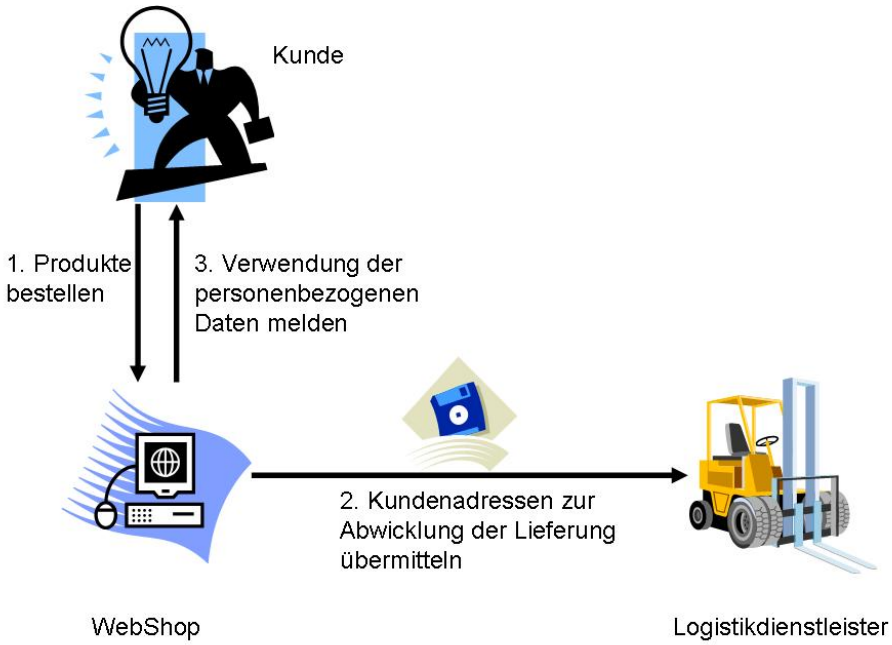

Abbildung 8: Szenario eines Bestellprozesses mit Übermittlung personenbezogener Daten von einem Webshop zu einem Logistikdienstleister

4.2. Szenario als Web Services

Zur Verdeutlichung der Funktionsweise des Sticky Loggings wurde das Szenario aus Abbildung 8 von Seite 19 als Web Services realisiert. Im Kapitel 4.3 wird die Vorgehensweise zur Implementation des Szenarios aufgezeigt. Es folgt eine detaillierte Servicebeschreibung, in der die Akteure, die auftretenden Datenströme und die verwendeten Webmethoden aufgeführt werden.

4.2.1. Akteure

Nach Abbildung 8 von Seite 19 sind drei Akteure auszumachen: **Kunde**, **WebShop** und **Logistikdienstleister**.

Kunde (Customer)

Der Kunde wird als **Web Service Client** mit Name **Customer** realisiert. Er stößt den Bestellprozess an, indem er die Methode *order* beim **WebShop** aufruft und ihm seinen Namen und seine Lieferadresse übermittelt.

WebShop

Der WebShop wird als **Web Service** mit Name **WebShop** realisiert. Er besitzt eine Methode *order*, die vom **Customer** aufgerufen wird. Die Methode *order* erwartet zwei Eingabeparameter: Name und Lieferadresse des Kunden. Sie ruft die Methode *deliver* vom **Logistikdienstleister** auf und übergibt ihr den Namen und die Lieferadresse des Kunden. Die Methode *order* bestätigt dem **Customer** den Eingang seiner Bestellung.

Logistikdienstleister (LogisticInc)

Der Logistikdienstleister wird als **Web Service** mit Name **LogisticInc** realisiert. Er besitzt eine Methode *deliver*, die vom **WebShop** aufgerufen wird. Die Methode *deliver* erwartet zwei Eingabeparameter: Name und Lieferadresse des zu beliefernden Kunden. Die Methode *deliver* bestätigt dem **WebShop** den Eingang des Lieferauftrages.

4.2.2. Datenströme

Abbildung 9 verdeutlicht die vorhandenen Datenströme, die in diesem Szenario entstehen. Der Kunde (Customer) ruft beim Web Service WebShop zunächst die Methode *order* auf, um Name (*name*) und Lieferadresse (*address*) zu hinterlassen. Welches Produkt der Kunde kauft, wird hier nicht berücksichtigt. Innerhalb der Methode *order* wird daraufhin die Methode *deliver* beim Logistikdienstleister aufgerufen, um den Namen (*name*) und die Adresse (*address*) des zu beliefernden Kunden zu übermitteln. Der Logistikdienstleister liefert als Rückgabewert einen String, der den Namen und die Adresse des Kunden, sowie zusätzliche Textinformationen enthält (Exakter Wortlaut: *"LogisticInc answers: We will send the package to " + name + " who lives in " + address + "."*). Jetzt liefert die Methode *order* eine Antwort an den Kunden. Mit der Antwort wird dem Kunden mitgeteilt, dass die Lieferung seiner Ware angestoßen wird (Exakter Wortlaut: *"WebShop answers: Hello " + name + ". You bought the package in our shop. We will send the package to " + address + "."*). Der Kunde bekommt also als Antwort auf den Aufruf der Methode *order* folgenden Text:
"WebShop answers: Hello " + name + ". You bought the package in our shop. We will send the package to " + address + "."
"LogisticInc answers: We will send the package to " + name + " who lives in " + address + "."

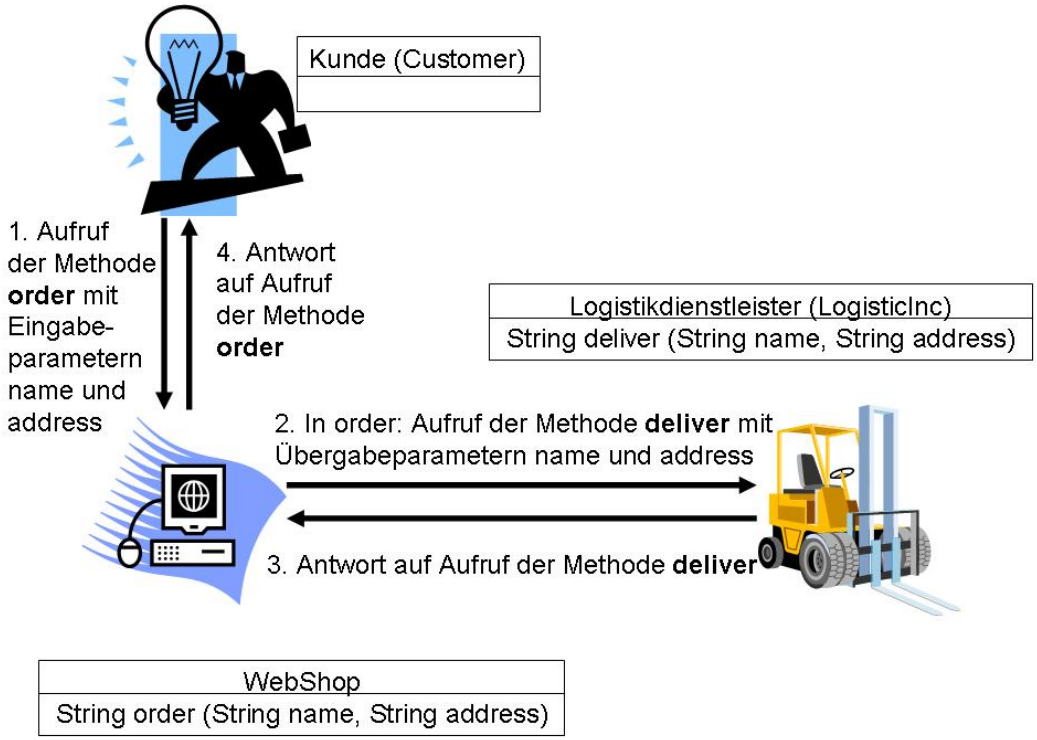

Abbildung 9: Grafik zur Verdeutlichung der Datenströme

4.2.3. Erläuterung der Webmethoden

Im Folgenden sind die Methodensignaturen der Web Services **WebShop** und **LogisticInc**, sowie eine Beschreibung der Methoden aufgeführt:

- **WebShop:**
  - **String order (String name, String address)**
    Die Methode *order* erwartet als Eingabeparameter zwei Strings: *name* und *address*, also den Namen und die Lieferadresse des Kunden. Die Methode ruft die Methode *deliver* vom Logistikdienstleister auf und übergibt ihr ihre beiden Eingabeparameter. Sie wartet auf die Antwort der Methode *deliver*. Die Methode *order* liefert als return-Wert einen String bestehend aus den Eingabeparametern und zusätzlichen Textbausteinen.
- **LogisticInc (Logistikdienstleister):**
  - **String deliver (String name, String address)**
    Die Methode *deliver* erwartet als Eingabeparameter zwei Strings: *name* und *address*, also den Namen und die Lieferadresse des zu beliefernden Kunden. Die Methode *deliver* liefert als return-Wert einen String bestehend aus den Eingabeparametern und zusätzlichen Textbausteinen.

4.3. Implementation des Szenarios

Zur Realisierung des Szenarios (siehe Abbildung 8) wurden drei dynamische Webprojekte mit Eclipse WTP angelegt:

- LogisticInc
- WebShop
- Customer

Im Anhang A.1 ist der Quellcode zu den Web Services und zum Client zu finden. Im Folgenden wird der Implementationsvorgang zu den Web Services LogisticInc und WebShop, sowie zum Web Service Client Customer vorgestellt.

Web Service LogisticInc

Das dynamische Webprojekt LogisticInc realisiert den Web Service LogisticInc aus dem Szenario. Listing 5 im Anhang A.1.1 zeigt den Java-Code der Delivery.java-Datei aus dem package de.unikoblenz.schnorr des src-Ordners. Die Methode deliver liefert zu den Parametern name und address einen String mit den Parametern und zusätzlichem Text zurück. Listing 6 besteht aus der modifizierten web.xml, welche im Ordner WebContent/WEB-INF zu finden ist. Diese Datei wird zum Registrieren des Web Services auf dem JBoss-Applicationserver benötigt. Nach der Veröffentlichung auf dem JBoss-Server kann der Web Service unter Verwendung des WSDL-Dokumentes unter http://127.0.0.1:8080/LogisticInc/Delivery?wsdl angesprochen werden.

Web Service WebShop

Das dynamische Webprojekt WebShop realisiert den Web Service WebShop aus dem Szenario. Listing 7 im Anhang A.1.2 enthält den Javacode des Web Services Order. In der Methode order wird auf die Methode deliver des Web Services Delivery zugegriffen. Dies wird mit dem Tool wsimport realisiert (siehe Kapitel 3.3). Mit dem Befehl **wsimport -d src -keep -p de.unikoblenz.schnorr** http://127.0.0.1:8080/LogisticInc/Delivery?wsdl werden unter anderem die Javaklassen Delivery.java und DeliveryService.java im Ordner *src/de/unikoblenz/schnorr* erzeugt. Diese können dann in das Projekt WebShop in Eclipse WTP kopiert und verwendet werden. In der return-Anweisung wird die WebMethode deliver vom Web Service Delivery verwendet und ihr die Parameter der Methode order (name und address) übermittelt. Listing 8 zeigt die web.xml-Datei, die zum Hochladen auf den Server benötigt wird. Nach der Veröffentlichung auf dem JBoss-Server kann der Web Service unter Verwendung des WSDL-Dokumentes unter http://127.0.0.1:8080/WebShop/Order?wsdl angesprochen werden.

Web Service Client Customer

Zur Erstellung des Clients kann auch das Tool **wsimport** verwendet werden. Mit dem Befehl **wsimport -d src -keep -p de.unikoblenz.schnorr** http://127.0.0.1:8080/WebShop/Order?wsdl werden unter anderem die Javaklassen Order.java und OrderService.java im Ordner *src/de/unikoblenz/schnorr* erzeugt. Diese können dann nach Eclipse WTP kopiert und verwendet werden. Listing 9 enthält den Javacode, der als Java-Applikation gestartet werden kann. Der Client ruft die Methode order des Web Service Order auf und übermittelt ihr einen Namen und eine Adresse. Das Ergebnis dieser Java-Anwendung stellt Listing 10 dar. Die Ausgabe entspricht dem zu erwartenden Ergebnis.

5. Sticky Logging

5.1. Beschreibung des Sticky-Logging-Formalismus

Christoph Ringelstein und Steffen Staab beschreiben in ihrer Veröffentlichung „Logging in Distributed Workflows“ (siehe Literarturverzeichnis unter [SL07]) die Funktionsweise des Sticky Logging. Wie der Name „sticky“ (dt. klebrig) schon sagt, werden die Loginformationen an die Daten als Metadaten gehängt und nicht in einer seperaten Logdatei gespeichert. Dies wird realisiert, indem die Loginformationen zusammen mit den Nutzdaten innerhalb der SOAP-Nachrichten verschickt werden. Eine detaillierte Beschreibung der Sticky-Logging-Ontologie ist hier zu sehen:
http://isweb.uni-koblenz.de/Research/SOA/StickyLogging.

Konzepte

Nachfolgend sind die Konzepte und Eigenschaften des Sticky Logging beschrieben.

Data Instances

Jedes Datum wie z.B. die Adresse eines Kunden kann mehrere Data Instances haben, wobei jede seinen eigenen Sticky Log hat. Eine Data Instance hat folgende Eigenschaften:

- **isPrimary:** Ein *xsd:boolean*, der angibt, ob Data Instance die primäre Instanz eines Datums ist.
- **isCopyOf:** Falls Data Instance eine Kopie von einer anderen Data Instance ist, gibt **isCopyOf** die URI zu dieser Data Instance an.
- **hasCopy:** URIs zu den direkten Kopien dieser Data Instance.
- **isAboutPerson:** Eine *foaf:person* zu der Person, für die diese Data Instance erstellt wurde.
- **hasSource:** Referenziert zu der Quelle (source) dieser Data Instance.
- **hasUUID:** Eine UUID[16] zur eindeutigen Identifikation der Data Instance.

Action on Data Instances

Mit diesem Konzept werden Aktionen auf Data Instances beschrieben. Eine Action hat folgende Eigenschaften:

- **hasPurpose:** Eine URI, die auf ein Ontologiekonzept zur Definition des Verwendungszweckes einer Action referenziert.

[16] Javadoc zu UUID: http://java.sun.com/j2se/1.5.0/docs/api/java/util/UUID.html

- **performedOnDataInstance:** Eine URI, die auf die Data Instance referenziert, auf die sich die Action bezieht.
- **performedBy:** Eine URI, die sich auf die Entity bezieht, die die Action ausgelöst hat.
- **leadsTo:** Eine URI, die sich auf die (falls vorhanden) nachfolgende Action bezieht.
- **hasSequentialNumber:** Eine *xsd:integer*, die die Nummer dieser Action angibt. Die erste Action hat die Nummer 1. Jede neue Action bekommt eine um eins größere Nummer zugeordnet.
- **hasUUID:** Eine UUID zur eindeutigen Identifikation der Action.
- **hasTimeStamp:** Eine *xsd:dateTime*, die Datum und genaue Uhrzeit der Ausführung der Action angibt. Sie muss in UTC angegeben werden.

Log Entry

Log Entries enthalten alle Informationen zu einer oder mehreren Actions auf ein Datum. Ein Log Entry hat folgende Eigenschaften:

- **logsAction:** Eine URI, die die Action angibt, die von Log Entry geloggt wird.
- **loggedBy:** Eine URI, die die Entity angibt, die den Log ausführt.
- **hasUUID:** Eine UUID zur eindeutigen Identifikation der Log Entry.
- **hasTimeStamp:** Eine *xsd:dateTime*, die Datum und genaue Uhrzeit der Erstellung der Log Entry angibt. Sie muss in UTC angegeben werden.

Entity

Jede Action, die auf eine Data Instance ausgeführt wird, wird von einer Person, einer Entity, ausgeführt. Eine Entity kann eine Organisation oder eine natürliche Person sein. Die handelnde Entity muss dabei der loggenden Entity entsprechen. Eine Entity hat folgende Eigenschaften:

- **hasName:** Ein *xsd:string*, der den Namen der Entity enthält.
- **hasID:** Ein *xsd:string*, der eine effektive Identifikationsnummer für den Entity enthält, z.B. international securities identifying number (ISIN).
- **hasAddress:** Die Adresse der Entity.
- **hasLogged:** Eine URI, die auf die Log Entries verweist, die von der Entity erstellt wurden.

- **hasPGPCertificate:** Eine URI, die auf das PGP-Zertifikat der Entity verweist.
- **hasTimeStamp:** Eine *xsd:dateTime*, die Datum und genaue Uhrzeit der Erstellung der Entity angibt. Sie muss in UTC angegeben werden.

Signature

Die Signature unterzeichnet Log Entities und alle Resourcen, auf die sich die Entity bezieht.

- **signs:** URIs, die auf alle Log Entries verweisen, die von dieser Signature unterzeichnet sind.
- **hasValue:** Ein *xsd:string*, der die Signature der Entity enthält.
- **hasUUID:** Eine UUID zur eindeutigen Identifikation der Signature.
- **hasTimeStamp:** Eine *xsd:dateTime*, die Datum und genaue Uhrzeit der Erstellung der Signature angibt. Sie muss in UTC angegeben werden.

Arten von Actions

In der Sticky Logging Ontologie werden folgende Arten von Actions unterschieden:

1. Lesender Zugriff auf Daten zum Benutzen
   - **UseAction:** Datum lesen, um es zu benutzen
2. Lesender Zugriff auf Daten zum Kopieren
   - **CopyAction:** Datum lesen, um es zu kopieren
3. Schreibender Zugriff
   - **CreateAction:** Erzeugen einer neuen Data Instance
   - **ChangeAction:** Ändern einer Data Instance
   - **DeleteAction:** Löschen einer Data Instance
4. Senden einer Data Instance von einer Entity zu einer anderen
   - **SendAction:** Senden einer Data Instance von einer Entity zu einer anderen. Das Konzept Action muss hierfür um eine Eigenschaft erweitert werden: *hasDestination:* Eine URI, die das Ziel der Senden-Action angibt.
   - **ReceiveAction:** Empfangen einer Data Instance von einer anderen Entity
5. Actions auf Log-Einträge nach Löschen der Data Instance
   - **LogReturnAction:** Senden der Log-Einträge zurück an den Service Client

- **LogReceiveAction:** Empfangen der gesendeten Log-Einträge beim Service Client
- **LogMergeAction:** Zusammensetzen (Mergen) der Log-Einträge beim Service Client

5.2. Implementation des Sticky-Logging-Formalismus

Zur Umsetzung des Prototyps wurde eine API zur Erstellung der StickyLogs implementiert. Die StickyLogs werden innerhalb der SOAP-Nachricht als SOAP-Attachment zwischen den Web Services ausgetauscht. Eine Implementation des Sticky-Logging-Austausches fand anhand des Szenarios statt.

5.2.1. Implementation einer Sticky Logging API zur Erstellung der Sticky Logs

Im Rahmen der Studienarbeit wurde eine API zur Erstellung der Sticky Logs programmiert. Die API ermöglicht eine einfache, typgeprüfte Erstellung der Konzepte des Sticky Loggings. Dazu zählen DataInstances, Actions, Entities, LogEntries und Signatures. Jede Klasse hat der Ontolgie entsprechende Attribute.
Abbildung 10 zeigt das UML-Klassendiagramm zur Sticky Logging API. Das UML-Diagramm wurde mit dem Tool EclipseUML von OMONDO[17] erstellt.

Im Anhang unter A.2 ist der Quellcode zu den Klassen RDF und StickyLog zu finden. Die Klasse RDF (siehe Listing 11) verwendet die Jena-API, um das Erzeugen und Bearbeiten von RDFs zu ermöglichen. Die Libraries müssen hierbei im Projekt und im Ordner WebContent/WEB-INF/lib (zum Hochladen auf den Server) hinterlegt werden. Die Klasse RDF definiert die Properties, die zum Schreiben der RDFs für das Sticky Logging benötigt werden. Es wird der namespace der StickyLogs auf „sl:“ festgelegt. Eine Funktion zur Rückgabe aller RDFs im String-Format ist enthalten. Die Funktion createModelFromString(String s) parst einen String zu einen RDF-Modell. Die Funktion mergeModels(Model model1, Model model2) greift auf die Jena-API zurück. Es wird die Funktion model1.union(model2) verwendet um zwei Modelle zusammenzusetzen (siehe hierzu auch Kapitel 3.4.2 Jena).
Die Erzeugung der StickyLogs findet mit Hilfe der Klasse StickyLog statt (siehe Listing 12). Die Klasse bietet Funktionen zum Erzeugen von DataInstances, Entities, Actions, LogEntries und Signatures, in denen jeweils auch die Erstellung der RDFs stattfindet. Die Funktion mergeStickyLogs greift auf die Funktion mergeModels der Klasse RDF zurück. Hiermit werden zwei Modelle zusammengesetzt und RDFs zur Umsetzung des Action-Konzeptes LogMergeAction geschrieben. Mit der Funktion getRDFString() können alle RDFs als String zurückgegeben werden.

[17] Webseite: http://www.omondo.com/

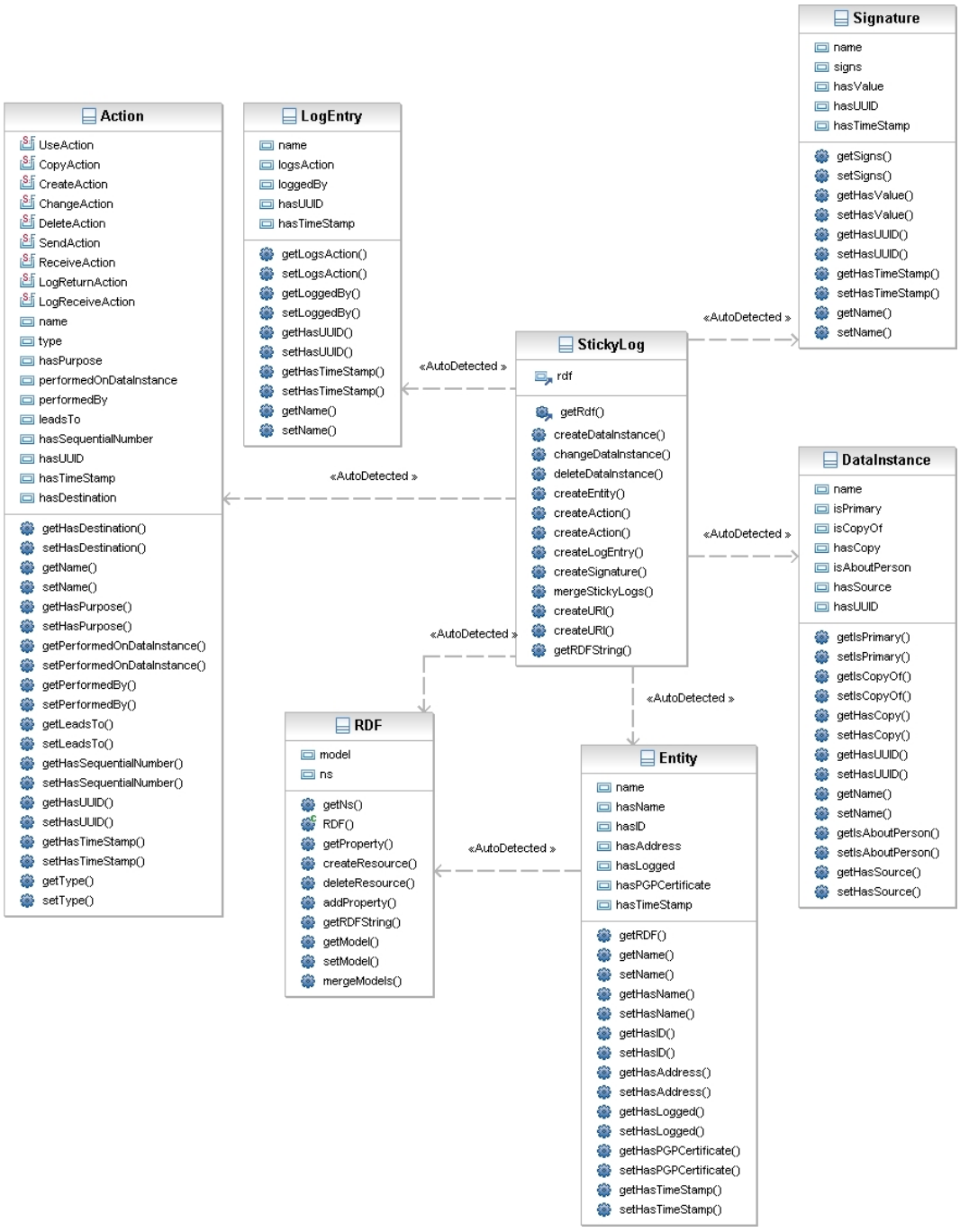

Abbildung 10: UML-Klassendiagramm zur Sticky Logging API

5.2.2. Austausch der Sticky Logs mit Hilfe der SOAP-Nachrichten

Der Austausch der Sticky Logs innerhalb der SOAP-Nachrichten wird mit dem Prototyp demonstriert. Es wurden zwei Messagehandler zum Bearbeiten der SOAP-Nachrichten beim Empfangen und Senden der Nachricht implementiert.

Messagehandler

JAX-WS bietet zur Modifikation der SOAP-Nachrichten Handler an. Zur Umsetzung des Prototyps wurde jeweils ein Handler für WebShop und LogisticInc implementiert.

WebShop

Listing 13 zeigt den Messagehandler beim WebShop. Mit der Funktion handleMessage (SOAPMessageContext messageContext) werden ein- und ausgehende SOAP-Nachrichten abgefangen. Die Boolean-Variable outbound gibt an, ob es sich um eine ausgehende Nachricht handelt. Ist dies der Fall, werden die StickyLogs für den Web Service WebShop erzeugt und als SOAP-Attachment in den Anhang der SOAP-Nachricht gepackt. Bei eingehenden Nachrichten (not outbound) wird der StickyLog mit der Methode getStickyLogFromAttachment(att) aus dem SOAP-Attachment ausgelesen. Die Methode getStickyLogFromAttachment(att) verwendet die Methode createModelFromString(slString) der Klasse RDF um die StickyLogs aus dem String zu parsen.

LogisticInc

Listing 14 zeigt den Messagehandler bei LogisticInc. Mit der Funktion handleMessage (SOAPMessageContext messageContext) werden ein- und ausgehende SOAP-Nachrichten abgefangen. Die Boolean-Variable outbound gibt an, ob es sich um eine ausgehende Nachricht handelt. Ist dies der Fall, werden die StickyLogs für den Web Service LogisticInc erzeugt, mit den eingegangen StickyLogs zusammengesetzt und als SOAP-Attachment in den Anhang der SOAP-Nachricht gepackt. Bei eingehenden Nachrichten (not outbound) wird der StickyLog mit der Methode getStickyLogFromAttachment(att) aus dem SOAP-Attachment ausgelesen. Die Methode getStickyLogFromAttachment(att) verwendet die Methode createModelFromString(slString) der Klasse RDF um die StickyLogs aus dem String zu parsen.

5.2.3. Einsatz des Sticky Loggings am Beispiel des Szenarios

An dieser Stelle wird der Einsatz des Sticky Logging Formalismus an dem oben beschriebenen Szenario demonstriert. Der Customer ruft die Methode order des Web Services WebShop auf. Der WebShop erzeugt mit Hilfe des Messagehandlers (siehe Listing 13)

die StickyLogs und fügt sie als SOAP-Attachment an die SOAP-Nachricht. Die SOAP-Nachricht wird an LogisticInc übermittelt. LogisticInc schreibt mit Hilfe des Messagehandlers auch StickyLogs, setzt sie mit den enthaltenen StickyLogs der eingehenden SOAP-Nachricht zusammen und fügt die zusammengesetzten StickyLogs als SOAP-Attachment an die ausgehende SOAP-Nachricht an. Diese werden zusammen mit den Nutzdaten wieder zurück an den WebShop übermittelt. Dem Customer wird nun mit Hilfe der StickyLogs die Auskunft über die Verwendung seiner Daten mitgeteilt. Unter A.4 ist der Java-Code zum Web Service WebShop zu finden. Der Messagehandler wird in die Handlerchain eingebunden. Mit der Methode mergeStickyLogs(..) der Sticky Logging API werden die StickyLogs zusammengesetzt und an den Customer übermittelt. Die Methode mergeStickyLogs verwendet die Methode mergeModels (Model model1 , Model model2) der Klasse RDF, die wiederrum auf die Methode model1.union(model2) der Jena-API zurückgreift (siehe hierzu auch Kapitel 3.4.2 zum Mergen von zwei Modellen).

Der Customer bekommt als Ausgabe die Consolenausgabe im Anhang unter B.1. Er kann nun sehen, welche Organisationen und Personen, Zugriff auf seine Daten hatten. Anhang B.2 zeigt die Consolenausgabe vom Server bei Ausführung von Client.java. Der Weg der SOAP-Nachrichten und der StickyLogs wird hierbei deutlich. Abbildung 11 stellt diesen Verlauf noch einmal in Anlehnung an ein Sequenzdiagramm dar.

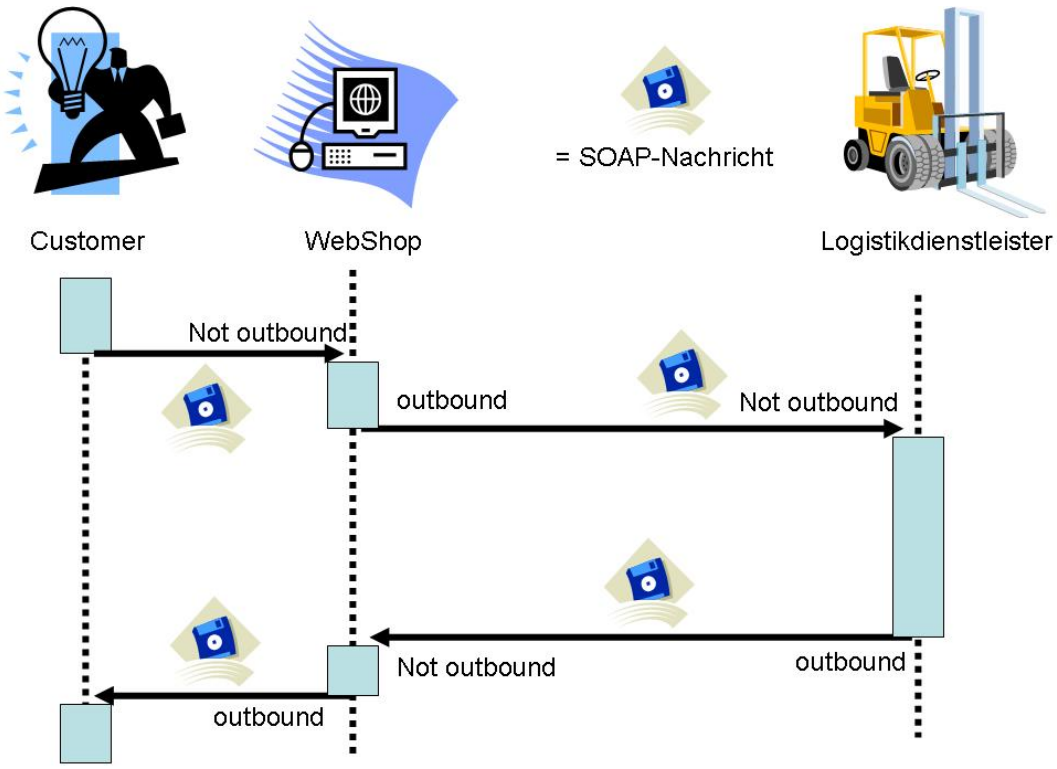

Abbildung 11: Sequenzdiagramm zur Verdeutlichung der Reihenfolge der Messagehandleraufrufe

6. Zusammenfassung

In dieser Studienarbeit wurden **Grundlagen** zu service-orientierten Architekturen und im Speziellen zu Web Services beschrieben. Die drei Komponenten einer Web-Service-Architektur (Kommunikation: SOAP, Dienstbeschreibung: WSDL und Verzeichnisdienst: UDDI) wurden erklärt.

Es fand eine Auswahl von **Komponenten** zur Implementation des Szenarios und des Sticky Loggings statt. Dazu zählt der JBoss Applicationserver, die Entwicklungsumgebung Eclipse WTP und das Tool wsimport zum Ansprechen der Web Services WebShop und LogisticInc. Die APIs, die zur Umsetzung des Szenarios als Web Service und zur Erstellung der Sticky Logging API, benötigt wurden, wurden vorgestellt. Das waren JAX-WS und Jena.

Es wurde ein **Szenario** zum Testen des Prototyps beschrieben und implementiert. Als Szenario wurde eine Übermittlung von Kundendaten vom Kunden über einen WebShop an einen Logistikdienstleister zur Simulation eines Liefervorgangs gewählt.

Im darauffolgenden Kapitel wurden die Konzepte des **Sticky Loggings** aufgezeigt. Darauf aufbauend fand die Implementation einer Sticky Logging API zur Erstellung der Sticky Logs statt. Mit Hilfe der SOAP-Nachrichten können die StickyLogs innerhalb des SOAP-Attachments zwischen den Web Services ausgetauscht werden. Eine Implementation dieses Austauschvorgangs zeigt der Prototyp.

Literatur

[SL07] Christoph Ringelstein und Steffen Staab: *Logging in Distributed Workflows*. Busan, South-Korea. Proceedings of the Workshop on Privacy Enforcement and Accountability with Semantics, November 2007. http://www.uni-koblenz.de/~cringel/pub/Ringelstein2007LID.pdf

[WA04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris und David Orchard: *Web Services Architecture*. NOTE-ws-arch-20040211, W3C - World Wide Web Consortium, Februar 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[WS05] Wolfgang Dostal, Mario Jeckle, Ingo Melzer und Barbara Zengler: *Service-orientierte Architekturen mit Web Services*. Elsevier, Spektrum Akademischer Verlag, 2005.

[S007] Nilo Mitra und Yves Lafon: *SOAP Version 1.2 Part 0: Primer (Second Edition)*. REC-soap12-part0-20070427, W3C - World Wide Web Consortium, April 2007. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[S107] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques´Moreau, Henrik Frystyk Nielsen, Anish Karmarkar und Yves Lafon: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. REC-soap12-part1-20070427, W3C - World Wide Web Consortium, April 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[S207] Martin Gudgin, Marc Hadley, Jean-Jacques Moreau und Henrik Frystyk Nielsen: *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. REC-soap12-part2-20070427, W3C - World Wide Web Consortium, April 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[S307] Hugo Haas, Oisin Hurley, Anish Karmarkar, Jeff Mischkinsky, Mark Jones, Lynne Thompson und Richard Martin: *SOAP Version 1.2 Specification Assertions and Test Collection (Second Edition)*. REC-soap12-testcollection-20070427, W3C - World Wide Web Consortium, April 2007. http://www.w3.org/TR/2007/REC-soap12-testcollection-20070427/

[RD04] Dave Beckett und Brian McBride: *RDF/XML Syntax Specification (Revised)*. REC-rdf-syntax-grammar-20040210, W3C - World Wide Web Consortium, Februar 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

A. Quellcode

A.1. Implementation des Szenarios

Zur Realisierung des Szenarios (siehe Abbildung 8) wurden drei dynamische Webprojekte mit Eclipse WTP angelegt:

- LogisticInc
- WebShop
- Customer

Nachfolgend ist der Java-Code und das XML-Dokument web.xml zum Registrieren der Web Services LogisticInc und WebShop auf dem JBoss-ApplicationServer zu finden. Außerdem wird der Java-Code des Customers zum Aufrufen der Methode order vom Web Services WebShop und das ausgegebene Ergebnis aufgeführt.

A.1.1. Web Service LogisticInc

Listing 5: Java-Code zu Delivery.java

```
package de.unikoblenz.schnorr;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

/**
 * This is a webservice class exposing a method called deliver which takes two
 * input parameters and returns them with additional text.
 *
 * @author schnorr
 */

/*
 * @WebService indicates that this is webservice interface and the name
 * indicates the webservice name.
 */
@WebService(name = "Delivery")
/*
 * @SOAPBinding indicates binding information of soap messages. Here we have
 * document-literal style of webservice and the parameter style is wrapped.
 */
@SOAPBinding
    (
            style = SOAPBinding.Style.DOCUMENT,
            use = SOAPBinding.Use.LITERAL,
            parameterStyle = SOAPBinding.ParameterStyle.WRAPPED
     )
public class Delivery
{
        /**
```

```
         * deliver returns a String with the input parameters name and address and adds
            some text
         * @param name
         *              name of customer
         * @param address
         *              address of customer
         * @return
         *              name and address of customer plus additional text
         */
    @WebMethod
    public String deliver( @WebParam(name = "name")
    String name, @WebParam(name = "address") String address )
    {
       return "LogisticInc answers: We will send the package to " + name + " who lives in
           " + address + ".";
    }

}
```

Listing 6: web.xml von LogisticInc

```
 <?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com
    /xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
    javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>Delivery</display-name>
  <servlet>
    <servlet-name>Delivery</servlet-name>
    <servlet-class>de.unikoblenz.schnorr.Delivery</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Delivery</servlet-name>
    <url-pattern>/Delivery</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

A.1.2. Web Service WebShop

Listing 7: Java-Code zu Order.java

```
package de.unikoblenz.schnorr;


import javax.jws.WebMethod;
```

```
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

/**
 * This is a webservice class exposing a method called order which takes two
 * input parameters and returns them with additional text.
 *
 * @author schnorr
 */

/*
 * @WebService indicates that this is webservice interface and the name
 * indicates the webservice name.
 */
@WebService(name = "Order")
/*
 * @SOAPBinding indicates binding information of soap messages. Here we have
 * document-literal style of webservice and the parameter style is wrapped.
 */
@SOAPBinding
    (
          style = SOAPBinding.Style.DOCUMENT,
          use = SOAPBinding.Use.LITERAL,
          parameterStyle = SOAPBinding.ParameterStyle.WRAPPED
     )
public class Order
{
        /**
         * order returns a String with the input parameters name and address and adds
             some text
         * @param name
         *              name of customer
         * @param address
         *              address of customer
         * @return
         *              name and address of customer plus additional text
         */
    @WebMethod
    public String order( @WebParam(name = "name")
    String name, @WebParam(name = "address") String address )
    {
          Delivery port = new DeliveryService().getDeliveryPort();

          return "WebShop answers: Hello " + name + ". You bought the package in our shop
              . We will send the package to " + address + ".\n " + port.deliver( name,
              address );
    }

}
```

Listing 8: web.xml von WebShop

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com
    /xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
    javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>Order</display-name>
```

```
  <servlet>
    <servlet-name>Order</servlet-name>
    <servlet-class>de.unikoblenz.schnorr.Order</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Order</servlet-name>
    <url-pattern>/Order</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

A.1.3. Web Service Client Customer

Listing 9: Java-Code zu Client.java

```
package de.unikoblenz.schnorr;

public class Client {
  public static void main( String[] args ){
   Order port = new OrderService().getOrderPort();
   System.out.printf( "Your result: %s",
   port.order( "Martin Schnorr","Sommerstr. 12, 56133 Fachbach" ));
  }
}
```

Listing 10: Ergebnis zu Client.java

```
Your result:
 WebShop answers: Hello Martin Schnorr. You bought the package in our shop. We will send
     the package to Sommerstr. 12, 56133 Fachbach.
 LogisticInc answers: We will send the package to Martin Schnorr who lives in Sommerstr.
     12, 56133 Fachbach.
```

A.2. Implementation der Sticky Logging API

Listing 11: Java-Code zu RDF.java

```
package StickyLog;

import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;
```

```
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;

/**
 * class for working with RDF
 * @author Schnorr
 *
 */
public class RDF {
        private Model model = ModelFactory.createDefaultModel();
        //namespace for StickyLogging
        private String ns = "sl:";

        public String getNs() {
                return ns;
        }

        public RDF(){
                //create the properties for StickyLogging
                model.createProperty("rdf:type");
        model.createProperty(ns + "isPrimary");
        model.createProperty(ns + "isCopyOf");
        model.createProperty(ns + "hasCopy");
        model.createProperty(ns + "isAboutPerson");
        model.createProperty(ns + "hasSource");
        model.createProperty(ns + "hasUUID");
        model.createProperty(ns + "hasPurpose");
        model.createProperty(ns + "performedOnDataInstance");
        model.createProperty(ns + "performedBy");
        model.createProperty(ns + "leadsTo");
        model.createProperty(ns + "hasSequentialNumber");
        model.createProperty(ns + "hasTimeStamp");
        model.createProperty(ns + "logsAction");
        model.createProperty(ns + "loggedBy");
        model.createProperty(ns + "hasName");
        model.createProperty(ns + "hasID");
        model.createProperty(ns + "hasAddress");
        model.createProperty(ns + "hasLogged");
        model.createProperty(ns + "hasPGCertificate");
        model.createProperty(ns + "signs");
        model.createProperty(ns + "hasValue");
        model.createProperty(ns + "hasContent");
        model.createProperty(ns + "hasDestination");
        }

        public Property getProperty(String name){
                return model.getProperty(name);
        }


        public Resource createResource(String res) {
                return model.createResource(res);
        }

        public void deleteResource(Resource res,Property prop, RDFNode r){
                model.remove(res,prop,r);
        }
```

```
        public void addProperty(Resource res, Property prop, String value){
                res.addProperty(prop, value);
        }

        /*
         * getRDFString returns the RDF in a String
         */
        public String getRDFString(){
                String result ="";
                // list the statements in the Model
        StmtIterator iter = model.listStatements();

        // print out the predicate, subject and object of each statement
        while (iter.hasNext()) {
            Statement stmt      = iter.nextStatement();  // get next statement
            Resource  subject   = stmt.getSubject();     // get the subject
            Property  predicate = stmt.getPredicate();   // get the predicate
            RDFNode   object    = stmt.getObject();      // get the object

            result += subject.toString();
            result += " " + predicate.toString() + " ";
            if (object instanceof Resource) {
               result += object.toString();
            } else {
                // object is a literal
                result += "\"" + object.toString() + "\"";
            }
            result += " .\n";
        }
        return result;
        }
        public void createModelFromString(String s){
                String[] sType = s.split(" .\n");

                for (int i = 0; i < sType.length; i++) {
                        String[] s2 = sType[i].split(" ");
                        Resource res = model.createResource(s2[0]);
                        res.addProperty(getProperty(s2[1]),s2[2].substring(1, s2[2].
                            length()-1));//cut footnotes from start and ending
                }
        }

        public Model getModel() {
                return model;
        }

        public void setModel(Model model) {
                this.model = model;
        }

        public Model mergeModels(Model model1, Model model2) {
                return model1.union(model2);
        }
}
```

Listing 12: Java-Code zu StickyLog.java

```
package StickyLog;
```

```
import java.net.URI;
import java.util.UUID;
import com.hp.hpl.jena.rdf.model.Resource;


/*
 * main class for StickyLogging
 * @author Schnorr
 */
public class StickyLog {
        private RDF rdf = new RDF();

        public RDF getRdf() {
                return rdf;
        }

        /*
         * creates DataInstance and RDF for DataInstance
         */
        public DataInstance createDataInstance(String name, Boolean isPrimary, URI
            isCopyOf, URI [] hasCopy, URI isAboutPerson, URI hasSource, UUID hasUUID){
                DataInstance d = new DataInstance();
                d.setName(name);
                d.setIsPrimary(isPrimary);
                d.setIsCopyOf(isCopyOf);
                d.setHasCopy(hasCopy);
                d.setIsAboutPerson(isAboutPerson);
                d.setHasSource(hasSource);
                d.setHasUUID(hasUUID);

                Resource res= rdf.createResource(name);
                //Create RDF in wrong direction (stack)
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasUUID"), hasUUID.
                    toString());
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasSource"), hasSource
                    .toString());
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "isAboutPerson"),
                    isAboutPerson.toString());
                for (int i = 0; i < hasCopy.length; i++) {
                        rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasCopy"),
                            hasCopy[i].toString());
                }
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "isCopyOf"), isCopyOf.
                    toString());
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "isPrimary"), isPrimary
                    .toString());
                rdf.addProperty(res,rdf.getProperty("rdf:type"), "sl:DataInstance");

                return d;
        }

        /*
         * changes DataInstance and add RDF for DataInstance and ChangeAction
         */
        public DataInstance changeDataInstance(DataInstance d, String name, Boolean
            isPrimary, URI isCopyOf, URI [] hasCopy, URI isAboutPerson, URI hasSource,
            UUID hasUUID, String changeActionName, URI changeActionHasPurpose, URI
            changeActionPerformedBy, URI changeActionLeadsTo, int
            changeActionHasSequentialNumber, UUID changeActionHasUUID, long
```

```
        changeActionHasTimeStamp){
            d.setName(name);
            d.setIsPrimary(isPrimary);
            d.setIsCopyOf(isCopyOf);
            d.setHasCopy(hasCopy);
            d.setIsAboutPerson(isAboutPerson);
            d.setHasSource(hasSource);
            d.setHasUUID(hasUUID);

            //TODO: delete old DataInstance from RDF

            //RDF for changed DataInstance
            Resource res= rdf.createResource(changeActionName);
            //Create RDF in wrong direction (stack)
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasUUID"), hasUUID.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasSource"), hasSource
                .toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "isAboutPerson"),
                isAboutPerson.toString());
            for (int i = 0; i < hasCopy.length; i++) {
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasCopy"),
                    hasCopy[i].toString());
            }
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "isCopyOf"), isCopyOf.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "isPrimary"), isPrimary
                .toString());
            rdf.addProperty(res,rdf.getProperty("rdf:type"), "sl:DataInstance");

            //RDF for ChangeAction
            Resource res2= rdf.createResource(name);
            //Create RDF in wrong direction (stack)
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                changeActionHasTimeStamp);
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasUUID"),
                changeActionHasUUID.toString());
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasSequentialNumber")
                , ""+changeActionHasSequentialNumber);
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "leadsTo"),
                changeActionLeadsTo.toString());
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "performedBy"),
                changeActionPerformedBy.toString());
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "
                performedOnDataInstance"), name);
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasPurpose"),
                changeActionHasPurpose.toString());
            rdf.addProperty(res2,rdf.getProperty("rdf:type"), "sl:ChangeAction");

            return d;
    }

    /*
     * deletes DataInstance and add RDF for DeleteAction
     */
    public DataInstance deleteDataInstance(DataInstance d, String deleteActionName,
        URI deleteActionHasPurpose, URI deleteActionPerformedBy, URI
        deleteActionLeadsTo, int deleteActionHasSequentialNumber, UUID
        deleteActionHasUUID, long deleteActionHasTimeStamp){
```

```
            //TODO: delete DataInstance from RDF

            //RDF for DeleteAction
            Resource res2= rdf.createResource(deleteActionName);
            //Create RDF in wrong direction (stack)
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                deleteActionHasTimeStamp);
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasUUID"),
                deleteActionHasUUID.toString());
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasSequentialNumber")
                , ""+deleteActionHasSequentialNumber);
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "leadsTo"),
                deleteActionLeadsTo.toString());
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "performedBy"),
                deleteActionPerformedBy.toString());
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "
                performedOnDataInstance"), d.getName());
            rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasPurpose"),
                deleteActionHasPurpose.toString());
            rdf.addProperty(res2,rdf.getProperty("rdf:type"), "sl:DeleteAction");

            return d;
        }

        /*
         * creates Entity and RDF for Entity
         */
        public Entity createEntity(String name, String hasName, String hasID, String
            hasAddress, URI hasLogged, URI hasPGPCertificate, long hasTimeStamp){
            Entity e = new Entity();
            e.setName(name);
            e.setHasName(hasName);
            e.setHasID(hasID);
            e.setHasAddress(hasAddress);
            e.setHasLogged(hasLogged);
            e.setHasPGPCertificate(hasPGPCertificate);
            e.setHasTimeStamp(hasTimeStamp);

            Resource res= rdf.createResource(name);
            //Create RDF in wrong direction (stack)
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                hasTimeStamp);
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasPGPCertificate"),
                hasPGPCertificate.toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasLogged"), hasLogged
                .toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasAddress"),
                hasAddress.toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasID"), hasID.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasName"), hasName.
                toString());
            rdf.addProperty(res,rdf.getProperty("rdf:type"), "sl:Entity");

            return e;
        }

        /*
         * creates Action and RDF for Action
```

```
     */
    public Action createAction(String name, String type, URI hasPurpose, URI
        performedOnDataInstance, URI performedBy, URI leadsTo, int
        hasSequentialNumber, UUID hasUUID, long hasTimeStamp){
            Action a = new Action();
            a.setName(name);
            a.setType(type);
            a.setHasPurpose(hasPurpose);
            a.setPerformedOnDataInstance(performedOnDataInstance);
            a.setPerformedBy(performedBy);
            a.setLeadsTo(leadsTo);
            a.setHasSequentialNumber(hasSequentialNumber);
            a.setHasUUID(hasUUID);
            a.setHasTimeStamp(hasTimeStamp);

            Resource res= rdf.createResource(name);
            //Create RDF in wrong direction (stack)
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                hasTimeStamp);
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasUUID"), hasUUID.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasSequentialNumber"),
                 ""+hasSequentialNumber);
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "leadsTo"), leadsTo.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "performedBy"),
                performedBy.toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "
                performedOnDataInstance"), performedOnDataInstance.toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasPurpose"),
                hasPurpose.toString());
            rdf.addProperty(res,rdf.getProperty("rdf:type"),"sl:"+type);

            return a;
    }

    /*
     * creates Action and RDF for Action with parameter hasDestination
     */
    public Action createAction(String name, String type, URI hasPurpose, URI
        performedOnDataInstance, URI performedBy, URI leadsTo, int
        hasSequentialNumber, UUID hasUUID, long hasTimeStamp, URI hasDestination){
            Action a = new Action();
            a.setName(name);
            a.setType(type);
            a.setHasPurpose(hasPurpose);
            a.setPerformedOnDataInstance(performedOnDataInstance);
            a.setPerformedBy(performedBy);
            a.setLeadsTo(leadsTo);
            a.setHasSequentialNumber(hasSequentialNumber);
            a.setHasUUID(hasUUID);
            a.setHasTimeStamp(hasTimeStamp);
            a.setHasDestination(hasDestination);

            Resource res= rdf.createResource(name);
            //Create RDF in wrong direction (stack)
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasDestination"), ""+
                hasDestination.toString());
```

```
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                hasTimeStamp);
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasUUID"), hasUUID.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasSequentialNumber"),
                 ""+hasSequentialNumber);
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "leadsTo"), leadsTo.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "performedBy"),
                performedBy.toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "
                performedOnDataInstance"), performedOnDataInstance.toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasPurpose"),
                hasPurpose.toString());
            rdf.addProperty(res,rdf.getProperty("rdf:type"), "sl:"+type);

            return a;
        }

        /*
         * creates LogEntry and RDF for LogEntry
         */
        public LogEntry createLogEntry(String name, URI logsAction, URI loggedBy, UUID
            hasUUID, long hasTimeStamp){
            LogEntry le = new LogEntry();
            le.setName(name);
            le.setLogsAction(logsAction);
            le.setLoggedBy(loggedBy);
            le.setHasUUID(hasUUID);
            le.setHasTimeStamp(hasTimeStamp);

            Resource res= rdf.createResource(name);
            //Create RDF in wrong direction (stack)
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                hasTimeStamp);
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasUUID"), hasUUID.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "loggedBy"), loggedBy.
                toString());
            rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "logsAction"),
                logsAction.toString());
            rdf.addProperty(res,rdf.getProperty("rdf:type"), "sl:LogEntry");

            return le;
        }

        /*
         * creates Signature and RDF for Signature
         */
        public Signature createSignature(String name, URI signs, String hasValue, UUID
            hasUUID, long hasTimeStamp){
            Signature s = new Signature();
            s.setName(name);
            s.setSigns(signs);
            s.setHasValue(hasValue);
            s.setHasUUID(hasUUID);
            s.setHasTimeStamp(hasTimeStamp);

            Resource res= rdf.createResource(name);
```

```
                //Create RDF in wrong direction (stack)
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                    hasTimeStamp);
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasUUID"), hasUUID.
                    toString());
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "hasValue"), hasValue);
                rdf.addProperty(res,rdf.getProperty(rdf.getNs() + "signs"), signs.
                    toString());
                rdf.addProperty(res,rdf.getProperty("rdf:type"), "sl:Signature");

                return s;
        }

        /*
         * merge StickyLogs and write RDF for LogMergeAction
         */
        public RDF mergeStickyLogs(RDF r1, RDF r2, String logMergeActionName, URI
            logMergeActionHasPurpose, URI logMergeActionPerformedBy, URI
            logMergeActionLeadsTo, int logMergeActionHasSequentialNumber, UUID
            logMergeActionHasUUID, long logMergeActionHasTimeStamp){
                rdf.setModel(rdf.mergeModels(r1.getModel(), r2.getModel()));

                //RDF for LogMergeAction
                Resource res2= rdf.createResource(logMergeActionName);
                //Create RDF in wrong direction (stack)
                rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasTimeStamp"), ""+
                    logMergeActionHasTimeStamp);
                rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasUUID"),
                    logMergeActionHasUUID.toString());
                rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasSequentialNumber")
                    , ""+logMergeActionHasSequentialNumber);
                rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "leadsTo"),
                    logMergeActionLeadsTo.toString());
                rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "performedBy"),
                    logMergeActionPerformedBy.toString());
                //TODO: rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "
                    performedOnDataInstance"), ???);
                rdf.addProperty(res2,rdf.getProperty(rdf.getNs() + "hasPurpose"),
                    logMergeActionHasPurpose.toString());
                rdf.addProperty(res2,rdf.getProperty("rdf:type"), "sl:LogMergeAction");

                return r1;
        }

        public URI createURI(String namespace, String localname){
                return URI.create(namespace+localname);
        }
        public URI createURI(String url){
                return URI.create(url);
        }
        public String getRDFString(){
                return rdf.getRDFString();
        }
}
```

A.3. Messagehandler zur Modifikation von ein- und ausgehenden SOAP-Nachrichten

Listing 13: Java-Code zu MsgHandler.java beim WebShop

```
package de.unikoblenz.schnorr;

import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
import javax.xml.namespace.QName;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;

import StickyLog.Action;
import StickyLog.DataInstance;
import StickyLog.Entity;
import StickyLog.LogEntry;
import StickyLog.RDF;
import StickyLog.Signature;
import StickyLog.StickyLog;

import java.net.URI;
import java.util.Date;
import java.util.Iterator;
import java.util.Set;
import java.util.UUID;

/**
 * MsgHandler for handling SOAP-Message
 * get StickyLogs from SOAP-Attachment of incoming SOAP-Message and
 * adds new and old StickyLogs to outgoing SOAP-Message
 *
 * @author Schnorr
 *
 */
public class MsgHandler  implements SOAPHandler<SOAPMessageContext>
{
    private StickyLog sl = new StickyLog();
    public StickyLog slOld = new StickyLog();
    private String namespace = "http://schnorr.unikoblenz.de/";

   public boolean handleFault(SOAPMessageContext messageContext)
    {
        return true;
    }

   public boolean handleMessage(SOAPMessageContext messageContext)
    {
        System.out.println ("HANDLER_WebShop!!!");
        SOAPMessageContext smc = (SOAPMessageContext)messageContext;
        SOAPMessage msg = smc.getMessage();

        Boolean outbound = (Boolean) messageContext.get(SOAPMessageContext.
            MESSAGE_OUTBOUND_PROPERTY);
        if (outbound) {
            // outgoing SOAP-Message
```

```
                System.out.println ("HANDLER WebShop outbound !!!");
                createStickyLog(namespace);
                if (slOld.getRDFString()!=""){
                        sl.mergeStickyLogs(sl.getRdf(), slOld.getRdf(), "action4", sl.
                            createURI("dofd:DeliverOrder"), sl.createURI(namespace, "
                            entity1"), sl.createURI(namespace, "none"), 4, UUID.
                            randomUUID(), new Date().getTime());
                }
                AttachmentPart att = msg.createAttachmentPart(sl.getRDFString(), "text/
                    plain");
                msg.addAttachmentPart(att);

        } else {
                //incoming SOAP-Message
                System.out.println ("HANDLER WebShop not outbound !!!");
                // get iterator of attachments
                Iterator<?> attIter = msg.getAttachments();
                // iterate through attachments
                if (attIter.hasNext()) {
                        while (attIter.hasNext()) {
                                // get the next attachment
                                AttachmentPart att = (AttachmentPart)attIter.next();
                                System.out.println("WebShop: The Web Service received
                                    attachment of type: " + att.getContentType() + "\n");
                                try {
                                        getStickyLogFromAttachment(att);
                                        } catch (SOAPException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                        }
                        }
                } else {
            System.out.println("WebShop: Did not receive any attachments.");
                }
        }
                try{
                        msg.writeTo (System.out);
                } catch (Exception e) {

                }
        return true;
    }

        //@Override
        public Set<QName> getHeaders() {
                // TODO Auto-generated method stub
                return null;
        }

        //@Override
        public void close(MessageContext arg0) {
                // TODO Auto-generated method stub
        }

        /**
     * creates StickyLogs for WebShop
     *
     * @param namespace
     * @return
```

```
     */
        public RDF createStickyLog(String namespace){
                    URI[] hasCopy = {sl.createURI(namespace, "none")};
                    hasCopy[0]= sl.createURI(namespace, "none");

                    DataInstance addressDI1 = sl.createDataInstance("addressDI1",true, sl.
                        createURI(namespace, "none"), hasCopy, sl.createURI(namespace, "
                        none"), sl.createURI(namespace, "none"), UUID.randomUUID());
                    Entity entity1 = sl.createEntity("entity1", "WebShop", "101020301", "
                        Koblenz", sl.createURI(namespace, "logEntry1"), sl.createURI("http
                        ://sbinc.com/cert.asc"), new Date().getTime());
                    Action action1 = sl.createAction("action1", Action.CreateAction, sl.
                        createURI("dofd:DeliverOrder"), sl.createURI(namespace, "
                        addressDI1"), sl.createURI(namespace, "entity1"), sl.createURI(
                        namespace, "none"), 1, UUID.randomUUID(), new Date().getTime());
                    LogEntry logEntry1 = sl.createLogEntry("logEntry1",sl.createURI(
                        namespace, "action1"), sl.createURI(namespace, "entity1"), UUID.
                        randomUUID(), new Date().getTime());
                    Signature signature1 = sl.createSignature("signature1", sl.createURI(
                        namespace, "logEntry1"), "HrdSDFc...", UUID.randomUUID(), new Date
                        ().getTime());

                    return sl.getRdf();
                }

      /**
         * uses method createModelFromString(String StickyLog) for
         * Merging 2 models
         *
         * @param att       the attachmentpart of a SOAP-Message
         * @throws SOAPException
         */
             public void getStickyLogFromAttachment(AttachmentPart att) throws SOAPException{
                    String slString = (String) att.getContent();
                    slOld.getRdf().createModelFromString(slString);
                 }

        public String getStickyLog() {
                return sl.getRDFString();
        }

        public StickyLog getSl() {
                return sl;
        }

        public StickyLog getSlOld() {
                return slOld;
        }

        public String getNamespace() {
                return namespace;
        }

        public void setNamespace(String namespace) {
                this.namespace = namespace;
        }
}
```

Listing 14: Java-Code zu MsgHandler.java bei LogisticInc

```
package de.unikoblenz.schnorr;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
import javax.xml.namespace.QName;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;

import StickyLog.Action;
import StickyLog.Entity;
import StickyLog.LogEntry;
import StickyLog.RDF;
import StickyLog.Signature;
import StickyLog.StickyLog;

import java.util.Date;
import java.util.Iterator;
import java.util.Set;
import java.util.UUID;

/**
 * MsgHandler for handling SOAP-Messages
 * get StickyLogs from SOAP-Attachment of incoming SOAP-Message and
 * adds new and old StickyLogs to outgoing SOAP-Message
 *
 * @author Schnorr
 *
 */
public class MsgHandler  implements SOAPHandler<SOAPMessageContext>
{
    StickyLog sl = new StickyLog();
    StickyLog slOld = new StickyLog();
    String namespace = "http://schnorr.unikoblenz.de/";

   public boolean handleFault(SOAPMessageContext messageContext)
    {
        return true;
    }

   public boolean handleMessage(SOAPMessageContext messageContext)
    {
        System.out.println ("HANDLER Delivery !!!");
        SOAPMessageContext smc = (SOAPMessageContext)messageContext;
        SOAPMessage msg = smc.getMessage();

        Boolean outbound = (Boolean) messageContext.get(MessageContext.
            MESSAGE_OUTBOUND_PROPERTY);
        if (outbound) {
                // outgoing SOAP-Message
                System.out.println ("HANDLER Delivery outbound !!!");

                createStickyLog(namespace);
```

```
                sl.mergeStickyLogs(sl.getRdf(), slOld.getRdf(), "action3", sl.createURI("
                    dofd:DeliverOrder"), sl.createURI(namespace, "entity2"), sl.createURI
                    (namespace, "none"), 3, UUID.randomUUID(), new Date().getTime());

                AttachmentPart att = msg.createAttachmentPart(sl.getRDFString(), "text/
                    plain");
                msg.addAttachmentPart(att);
        } else {
                //incoming SOAP-Message
                System.out.println ("HANDLER Delivery not outbound!!!");
                // get iterator of attachments
                Iterator<?> attIter = msg.getAttachments();

                // iterate through attachments
                if (attIter.hasNext()) {
                        while (attIter.hasNext()) {
                                // get the next attachment
                                AttachmentPart att = (AttachmentPart)attIter.next();
                                System.out.println("LogisticInc: The Web Service received
                                    attachment of type: " + att.getContentType() + "\n")
                                    ;
                                try {
                                                getStickyLogFromAttachment(att);
                                        } catch (SOAPException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                        }
                        }
                } else {
            System.out.println("LogisticInc: Did not receive any attachments.");
                }
        }
        try{
                msg.writeTo (System.out);
        } catch (Exception e) {

        }

        return true;
    }

  /**
     * creates StickyLogs for LogisticInc
     *
     * @param namespace
     * @return
     */
    public RDF createStickyLog(String namespace){
                Entity entity1 = sl.createEntity("entity2", "LogisticInc", "2378362136",
                    "Berlin", sl.createURI(namespace, "logEntry2"), sl.createURI("http://
                    sbinc.com/cert.asc"), new Date().getTime());
                Action action1 = sl.createAction("action2", Action.UseAction, sl.
                    createURI("dofd:DeliverOrder"), sl.createURI(namespace, "addressDI1")
                    , sl.createURI(namespace, "entity2"), sl.createURI(namespace, "none")
                    , 2, UUID.randomUUID(), new Date().getTime());
                LogEntry logEntry1 = sl.createLogEntry("logEntry2",sl.createURI(namespace
                    , "action2"), sl.createURI(namespace, "entity2"), UUID.randomUUID(),
                    new Date().getTime());
```

```
                Signature signature1 = sl.createSignature("signature2", sl.createURI(
                    namespace, "logEntry2"), "GfgfGFc...", UUID.randomUUID(), new Date().
                    getTime());

                return sl.getRdf();
    }

    /**
     * uses method createModelFromString(String StickyLog) for
     * Merging 2 models
     *
     * @param att       the attachmentpart of a SOAP-Message
     * @throws SOAPException
     */
    public void getStickyLogFromAttachment(AttachmentPart att) throws SOAPException{
        String slString = (String) att.getContent();
        slOld.getRdf().createModelFromString(slString);
    }

  /**
    * Is called after constructing the handler and before executing any other method.
    */
    @PostConstruct
    public void init() {

    }

        //@Override
        public Set<QName> getHeaders() {
                // TODO Auto-generated method stub
                return null;
        }

        /**
        * Is executed before this handler is being destroyed -
        * means after close() has been executed.
        */
        @PreDestroy
        public void destroy() {

        }

        //@Override
        public void close(MessageContext arg0) {
                // TODO Auto-generated method stub
        }
}
```

A.4. Einbinden des Messagehandlers

Listing 15: Java-Code zu Order.java beim WebShop

```
package de.unikoblenz.schnorr;

import java.net.URI;
import java.util.ArrayList;
import java.util.Date;
```

```
import java.util.List;
import java.util.UUID;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.ws.Binding;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.handler.Handler;

import StickyLog.RDF;
import StickyLog.StickyLog;

/**
 * This is a webservice class exposing a method called order which takes two
 * input parameters and returns them with additional text and StickyLogs.
 *
 * @author schnorr
 */

/*
 * @WebService indicates that this is webservice interface and the name
 * indicates the webservice name.
 */
@WebService(name = "Order")
/*
 * @SOAPBinding indicates binding information of soap messages. Here we have
 * document-literal style of webservice and the parameter style is wrapped.
 */
@SOAPBinding
   (
         style = SOAPBinding.Style.DOCUMENT,
         use = SOAPBinding.Use.LITERAL,
         parameterStyle = SOAPBinding.ParameterStyle.WRAPPED
    )
public class Order
{
        /**
         * order returns a String with the input parameters name and address
         * and adds some text and the StickyLogs
         * @param name
         *              name of customer
         * @param address
         *              address of customer
         * @return
         *              name and address of customer plus additional text and StickyLogs
         */
   @WebMethod
   public String order( @WebParam(name = "name")
   String name, @WebParam(name = "address") String address )
   {
         //Bind the MessageHandler and add it to the handlerChain
         Delivery port = new DeliveryService().getDeliveryPort();
         Binding binding = ((BindingProvider) port).getBinding();
      MsgHandler handler = new MsgHandler();
      List<Handler> handlerChain = new ArrayList<Handler>();
      handlerChain.add((Handler)handler);
      binding.setHandlerChain(handlerChain);
```

```
        //get StickyLogs from the MessageHandler
        StickyLog sl = handler.getSl();
        String namespace = handler.getNamespace();

        //return merged StickyLogs
            return "WebShop answers: Hello " + name + ". You bought the package in our shop
                . We will send the package to " + address + ".\n" + port.deliver( name,
                address ) + "\nStickyLog:\n"+sl.mergeStickyLogs(sl.getRdf(), handler.slOld.
                getRdf(), "action4", sl.createURI("dofd:DeliverOrder"), sl.createURI(
                namespace, "entity1"), sl.createURI(namespace, "none"), 4, UUID.randomUUID
                (), new Date().getTime()).getRDFString();
    }
}
```

B. Consolenausgaben

B.1. Ausgabe beim Client

Listing 16: Ausgabe beim Client bei Ausführung von Client.java

```
Your result:
WebShop answers: Hello Martin Schnorr. You bought the package in our shop. We will send
    the package to Sommerstr. 12, 56133 Fachbach.
LogisticInc answers: We will send the package to Martin Schnorr who lives in Sommerstr.
    12, 56133 Fachbach.

StickyLog:
entity2 sl:hasTimeStamp "1212408097920" .
entity2 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity2 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry2" .
entity2 sl:hasAddress "Berlin" .
entity2 sl:hasID "2378362136" .
entity2 sl:hasName "LogisticInc" .
entity2 rdf:type "sl:Entity" .
logEntry2 sl:hasTimeStamp "1212408097930" .
logEntry2 sl:hasUUID "5ffa354d-c24b-43ce-a893-d6f3278d27c3" .
logEntry2 sl:loggedBy "http://schnorr.unikoblenz.de/entity2" .
logEntry2 sl:logsAction "http://schnorr.unikoblenz.de/action2" .
logEntry2 rdf:type "sl:LogEntry" .
signature2 sl:hasTimeStamp "1212408097930" .
signature2 sl:hasUUID "4fa4aadc-e741-42ca-9baf-e4c7cc726be5" .
signature2 sl:hasValue "GfgfGFc..." .
signature2 sl:signs "http://schnorr.unikoblenz.de/logEntry2" .
signature2 rdf:type "sl:Signature" .
action3 rdf:type "sl:LogMergeAction" .
action3 sl:hasPurpose "dofd:DeliverOrder" .
action3 sl:performedBy "http://schnorr.unikoblenz.de/entity2" .
action3 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action3 sl:hasSequentialNumber "3" .
action3 sl:hasUUID "8d4644b6-c03c-4000-a89a-367e03563235" .
action3 sl:hasTimeStamp "1212408097930" .
action1 sl:hasTimeStamp "1212408096317" .
action1 sl:hasUUID "6b0c251e-feef-4ebc-bcfd-bb473f6b83b4" .
action1 sl:hasSequentialNumber "1" .
action1 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
```

```
action1 sl:performedBy "http://schnorr.unikoblenz.de/entity1" .
action1 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action1 sl:hasPurpose "dofd:DeliverOrder" .
action1 rdf:type "sl:CreateAction" .
entity1 sl:hasTimeStamp "1212408096317" .
entity1 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity1 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry1" .
entity1 sl:hasAddress "Koblenz" .
entity1 sl:hasID "101020301" .
entity1 sl:hasName "WebShop" .
entity1 rdf:type "sl:Entity" .
action4 rdf:type "sl:LogMergeAction" .
action4 sl:hasPurpose "dofd:DeliverOrder" .
action4 sl:performedBy "http://schnorr.unikoblenz.de/entity1" .
action4 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action4 sl:hasSequentialNumber "4" .
action4 sl:hasUUID "291c9bb4-d926-4352-bb91-b2f37478edc9" .
action4 sl:hasTimeStamp "1212408098060" .
logEntry1 sl:hasTimeStamp "1212408096317" .
logEntry1 sl:hasUUID "f21bf98b-d0d5-4977-9c4c-fb51272764b4" .
logEntry1 sl:loggedBy "http://schnorr.unikoblenz.de/entity1" .
logEntry1 sl:logsAction "http://schnorr.unikoblenz.de/action1" .
logEntry1 rdf:type "sl:LogEntry" .
signature1 sl:hasTimeStamp "1212408096317" .
signature1 sl:hasUUID "9c0b5c08-1f52-41fb-9a4f-e115bb28e95a" .
signature1 sl:hasValue "HrdSDFc..." .
signature1 sl:signs "http://schnorr.unikoblenz.de/logEntry1" .
signature1 rdf:type "sl:Signature" .
action2 sl:hasTimeStamp "1212408097930" .
action2 sl:hasUUID "64f2c9bb-30ab-4d49-a4a8-8a7855d8bd35" .
action2 sl:hasSequentialNumber "2" .
action2 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action2 sl:performedBy "http://schnorr.unikoblenz.de/entity2" .
action2 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action2 sl:hasPurpose "dofd:DeliverOrder" .
action2 rdf:type "sl:UseAction" .
addressDI1 sl:hasUUID "9e7edcf7-5189-4d1a-ad93-51e009d10874" .
addressDI1 sl:hasSource "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isAboutPerson "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasCopy "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isCopyOf "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isPrimary "true" .
addressDI1 rdf:type "sl:DataInstance" .
```

B.2. Ausgabe beim Server

Listing 17: Ausgabe beim Server bei Ausführung von Client.java

```
14:01:35,987 INFO  [STDOUT] HANDLER WebShop!!!
14:01:35,987 INFO  [STDOUT] HANDLER WebShop not outbound!!!
14:01:35,987 INFO  [STDOUT] WebShop: Did not receive any attachments.
14:01:35,987 INFO  [STDOUT] <soapenv:Envelope xmlns:ns1='http://schnorr.unikoblenz.de/'
    xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/' xmlns:xsd='http://www.w3.
    org/2001/XMLSchema'><soapenv:Body><ns1:order xmlns:ns1='http://schnorr.unikoblenz.de/
    '><name>Martin Schnorr
14:01:35,987 INFO  [STDOUT] </name>
14:01:35,987 INFO  [STDOUT] <address>Sommerstr. 12, 56133 Fachbach
```

```
14:01:35,987 INFO  [STDOUT] </address>
14:01:35,987 INFO  [STDOUT] </ns1:order>
14:01:35,987 INFO  [STDOUT] </soapenv:Body></soapenv:Envelope>
14:01:36,297 INFO  [STDOUT] HANDLER WebShop!!!
14:01:36,297 INFO  [STDOUT] HANDLER WebShop outbound!!!

14:01:36,418 INFO  [STDOUT] ------=_Part_5_27931988.1212408096388
14:01:36,418 INFO  [STDOUT] Content-Type: text/xml; charset=UTF-8
14:01:36,418 INFO  [STDOUT] Content-Transfer-Encoding: 8bit
14:01:36,418 INFO  [STDOUT] Content-ID: <rootpart@ws.jboss.org>
14:01:36,418 INFO  [STDOUT] <env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/
    envelope/'><env:Header></env:Header><env:Body><ns1:deliver xmlns:ns1="http://schnorr.
    unikoblenz.de/"><name>Martin Schnorr</name><address>Sommerstr. 12, 56133 Fachbach</
    address></ns1:deliver></env:Body></env:Envelope>
14:01:36,418 INFO  [STDOUT] ------=_Part_5_27931988.1212408096388
14:01:36,418 INFO  [STDOUT] Content-Type: text/plain
14:01:36,418 INFO  [STDOUT] Content-Transfer-Encoding: binary
14:01:36,418 INFO  [STDOUT] Content-Id: 0-1212408096418-31177387@ws.jboss.org
14:01:36,418 INFO  [STDOUT] action1 rdf:type "sl:CreateAction" .
action1 sl:hasPurpose "dofd:DeliverOrder" .
action1 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action1 sl:performedBy "http://schnorr.unikoblenz.de/entity1" .
action1 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action1 sl:hasSequentialNumber "1" .
action1 sl:hasUUID "6b0c251e-feef-4ebc-bcfd-bb473f6b83b4" .
action1 sl:hasTimeStamp "1212408096317" .
entity1 rdf:type "sl:Entity" .
entity1 sl:hasName "WebShop" .
entity1 sl:hasID "101020301" .
entity1 sl:hasAddress "Koblenz" .
entity1 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry1" .
entity1 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity1 sl:hasTimeStamp "1212408096317" .
logEntry1 rdf:type "sl:LogEntry" .
logEntry1 sl:logsAction "http://schnorr.unikoblenz.de/action1" .
logEntry1 sl:loggedBy "http://schnorr.unikoblenz.de/entity1" .
logEntry1 sl:hasUUID "f21bf98b-d0d5-4977-9c4c-fb51272764b4" .
logEntry1 sl:hasTimeStamp "1212408096317" .
signature1 rdf:type "sl:Signature" .
signature1 sl:signs "http://schnorr.unikoblenz.de/logEntry1" .
signature1 sl:hasValue "HrdSDFc..." .
signature1 sl:hasUUID "9c0b5c08-1f52-41fb-9a4f-e115bb28e95a" .
signature1 sl:hasTimeStamp "1212408096317" .
addressDI1 rdf:type "sl:DataInstance" .
addressDI1 sl:isPrimary "true" .
addressDI1 sl:isCopyOf "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasCopy "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isAboutPerson "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasSource "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasUUID "9e7edcf7-5189-4d1a-ad93-51e009d10874" .
14:01:36,418 INFO  [STDOUT] ------=_Part_5_27931988.1212408096388--
14:01:37,890 INFO  [STDOUT] HANDLER Delivery!!!
14:01:37,890 INFO  [STDOUT] HANDLER Delivery not outbound!!!
14:01:37,890 INFO  [STDOUT] LogisticInc: The Web Service received attachment of type:
    text/plain

14:01:37,910 INFO  [STDOUT] ------=_Part_6_4268202.1212408097910
14:01:37,910 INFO  [STDOUT] Content-Type: text/xml; charset=UTF-8
14:01:37,910 INFO  [STDOUT] Content-Transfer-Encoding: 8bit
```

```
14:01:37,910 INFO  [STDOUT] Content-ID: <rootpart@ws.jboss.org>
14:01:37,920 INFO  [STDOUT] <env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/
    envelope/'><env:Header></env:Header><env:Body><ns1:deliver xmlns:ns1='http://schnorr.
    unikoblenz.de/'><name>Martin Schnorr</name><address>Sommerstr. 12, 56133 Fachbach</
    address></ns1:deliver></env:Body></env:Envelope>
14:01:37,920 INFO  [STDOUT] ------=_Part_6_4268202.1212408097910
14:01:37,920 INFO  [STDOUT] Content-Type: text/plain
14:01:37,920 INFO  [STDOUT] Content-Transfer-Encoding: binary
14:01:37,920 INFO  [STDOUT] Content-Id: 0-1212408096418-31177387@ws.jboss.org
14:01:37,920 INFO  [STDOUT] action1 rdf:type "sl:CreateAction" .
action1 sl:hasPurpose "dofd:DeliverOrder" .
action1 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action1 sl:performedBy "http://schnorr.unikoblenz.de/entity1" .
action1 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action1 sl:hasSequentialNumber "1" .
action1 sl:hasUUID "6b0c251e-feef-4ebc-bcfd-bb473f6b83b4" .
action1 sl:hasTimeStamp "1212408096317" .
entity1 rdf:type "sl:Entity" .
entity1 sl:hasName "WebShop" .
entity1 sl:hasID "101020301" .
entity1 sl:hasAddress "Koblenz" .
entity1 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry1" .
entity1 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity1 sl:hasTimeStamp "1212408096317" .
logEntry1 rdf:type "sl:LogEntry" .
logEntry1 sl:logsAction "http://schnorr.unikoblenz.de/action1" .
logEntry1 sl:loggedBy "http://schnorr.unikoblenz.de/entity1" .
logEntry1 sl:hasUUID "f21bf98b-d0d5-4977-9c4c-fb51272764b4" .
logEntry1 sl:hasTimeStamp "1212408096317" .
signature1 rdf:type "sl:Signature" .
signature1 sl:signs "http://schnorr.unikoblenz.de/logEntry1" .
signature1 sl:hasValue "HrdSDFc..." .
signature1 sl:hasUUID "9c0b5c08-1f52-41fb-9a4f-e115bb28e95a" .
signature1 sl:hasTimeStamp "1212408096317" .
addressDI1 rdf:type "sl:DataInstance" .
addressDI1 sl:isPrimary "true" .
addressDI1 sl:isCopyOf "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasCopy "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isAboutPerson "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasSource "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasUUID "9e7edcf7-5189-4d1a-ad93-51e009d10874" .
14:01:37,920 INFO  [STDOUT] ------=_Part_6_4268202.1212408097910--
14:01:37,920 INFO  [STDOUT] HANDLER Delivery!!!
14:01:37,920 INFO  [STDOUT] HANDLER Delivery outbound!!!

14:01:38,030 INFO  [STDOUT] ------=_Part_7_25814543.1212408098030
14:01:38,030 INFO  [STDOUT] Content-Type: text/xml; charset=UTF-8
14:01:38,030 INFO  [STDOUT] Content-Transfer-Encoding: 8bit
14:01:38,030 INFO  [STDOUT] Content-ID: <rootpart@ws.jboss.org>
14:01:38,030 INFO  [STDOUT] <env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/
    envelope/'><env:Header></env:Header><env:Body><ns2:deliverResponse xmlns:ns2="http://
    schnorr.unikoblenz.de/"><return>LogisticInc answers: We will send the package to
    Martin Schnorr who lives in Sommerstr. 12, 56133 Fachbach.
</return></ns2:deliverResponse></env:Body></env:Envelope>
14:01:38,030 INFO  [STDOUT] ------=_Part_7_25814543.1212408098030
14:01:38,030 INFO  [STDOUT] Content-Type: text/plain
14:01:38,030 INFO  [STDOUT] Content-Transfer-Encoding: binary
14:01:38,030 INFO  [STDOUT] Content-Id: 0-1212408098030-14767255@ws.jboss.org
14:01:38,030 INFO  [STDOUT] entity2 sl:hasTimeStamp "1212408097920" .
```

```
entity2 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity2 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry2" .
entity2 sl:hasAddress "Berlin" .
entity2 sl:hasID "2378362136" .
entity2 sl:hasName "LogisticInc" .
entity2 rdf:type "sl:Entity" .
logEntry2 sl:hasTimeStamp "1212408097930" .
logEntry2 sl:hasUUID "5ffa354d-c24b-43ce-a893-d6f3278d27c3" .
logEntry2 sl:loggedBy "http://schnorr.unikoblenz.de/entity2" .
logEntry2 sl:logsAction "http://schnorr.unikoblenz.de/action2" .
logEntry2 rdf:type "sl:LogEntry" .
signature2 sl:hasTimeStamp "1212408097930" .
signature2 sl:hasUUID "4fa4aadc-e741-42ca-9baf-e4c7cc726be5" .
signature2 sl:hasValue "GfgfGFc..." .
signature2 sl:signs "http://schnorr.unikoblenz.de/logEntry2" .
signature2 rdf:type "sl:Signature" .
action3 rdf:type "sl:LogMergeAction" .
action3 sl:hasPurpose "dofd:DeliverOrder" .
action3 sl:performedBy "http://schnorr.unikoblenz.de/entity2" .
action3 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action3 sl:hasSequentialNumber "3" .
action3 sl:hasUUID "8d4644b6-c03c-4000-a89a-367e03563235" .
action3 sl:hasTimeStamp "1212408097930" .
action1 rdf:type "sl:CreateAction" .
action1 sl:hasPurpose "dofd:DeliverOrder" .
action1 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action1 sl:performedBy "http://schnorr.unikoblenz.de/entity1" .
action1 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action1 sl:hasSequentialNumber "1" .
action1 sl:hasUUID "6b0c251e-feef-4ebc-bcfd-bb473f6b83b4" .
action1 sl:hasTimeStamp "1212408096317" .
entity1 rdf:type "sl:Entity" .
entity1 sl:hasName "WebShop" .
entity1 sl:hasID "101020301" .
entity1 sl:hasAddress "Koblenz" .
entity1 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry1" .
entity1 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity1 sl:hasTimeStamp "1212408096317" .
logEntry1 rdf:type "sl:LogEntry" .
logEntry1 sl:logsAction "http://schnorr.unikoblenz.de/action1" .
logEntry1 sl:loggedBy "http://schnorr.unikoblenz.de/entity1" .
logEntry1 sl:hasUUID "f21bf98b-d0d5-4977-9c4c-fb51272764b4" .
logEntry1 sl:hasTimeStamp "1212408096317" .
signature1 rdf:type "sl:Signature" .
signature1 sl:signs "http://schnorr.unikoblenz.de/logEntry1" .
signature1 sl:hasValue "HrdSDFc..." .
signature1 sl:hasUUID "9c0b5c08-1f52-41fb-9a4f-e115bb28e95a" .
signature1 sl:hasTimeStamp "1212408096317" .
action2 sl:hasTimeStamp "1212408097930" .
action2 sl:hasUUID "64f2c9bb-30ab-4d49-a4a8-8a7855d8bd35" .
action2 sl:hasSequentialNumber "2" .
action2 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action2 sl:performedBy "http://schnorr.unikoblenz.de/entity2" .
action2 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action2 sl:hasPurpose "dofd:DeliverOrder" .
action2 rdf:type "sl:UseAction" .
addressDI1 rdf:type "sl:DataInstance" .
addressDI1 sl:isPrimary "true" .
addressDI1 sl:isCopyOf "http://schnorr.unikoblenz.de/none" .
```

```
addressDI1 sl:hasCopy "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isAboutPerson "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasSource "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasUUID "9e7edcf7-5189-4d1a-ad93-51e009d10874" .
14:01:38,030 INFO  [STDOUT] ------=_Part_7_25814543.1212408098030--
14:01:38,040 INFO  [STDOUT] HANDLER WebShop!!!
14:01:38,040 INFO  [STDOUT] HANDLER WebShop not outbound!!!
14:01:38,040 INFO  [STDOUT] WebShop: The Web Service received attachment of type: text/
    plain

14:01:38,050 INFO  [STDOUT] ------=_Part_8_10238269.1212408098050
14:01:38,050 INFO  [STDOUT] Content-Type: text/xml; charset=UTF-8
14:01:38,050 INFO  [STDOUT] Content-Transfer-Encoding: 8bit
14:01:38,050 INFO  [STDOUT] Content-ID: <rootpart@ws.jboss.org>
14:01:38,050 INFO  [STDOUT] <env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/
    envelope/'><env:Header></env:Header><env:Body><ns2:deliverResponse xmlns:ns2='http://
    schnorr.unikoblenz.de/'><return>LogisticInc answers: We will send the package to
    Martin Schnorr who lives in Sommerstr. 12, 56133 Fachbach.
</return></ns2:deliverResponse></env:Body></env:Envelope>
14:01:38,050 INFO  [STDOUT] ------=_Part_8_10238269.1212408098050
14:01:38,050 INFO  [STDOUT] Content-Type: text/plain
14:01:38,050 INFO  [STDOUT] Content-Transfer-Encoding: binary
14:01:38,050 INFO  [STDOUT] Content-Id: 0-1212408098030-14767255@ws.jboss.org
14:01:38,050 INFO  [STDOUT] entity2 sl:hasTimeStamp "1212408097920" .
entity2 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity2 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry2" .
entity2 sl:hasAddress "Berlin" .
entity2 sl:hasID "2378362136" .
entity2 sl:hasName "LogisticInc" .
entity2 rdf:type "sl:Entity" .
logEntry2 sl:hasTimeStamp "1212408097930" .
logEntry2 sl:hasUUID "5ffa354d-c24b-43ce-a893-d6f3278d27c3" .
logEntry2 sl:loggedBy "http://schnorr.unikoblenz.de/entity2" .
logEntry2 sl:logsAction "http://schnorr.unikoblenz.de/action2" .
logEntry2 rdf:type "sl:LogEntry" .
signature2 sl:hasTimeStamp "1212408097930" .
signature2 sl:hasUUID "4fa4aadc-e741-42ca-9baf-e4c7cc726be5" .
signature2 sl:hasValue "GfgfGFc..." .
signature2 sl:signs "http://schnorr.unikoblenz.de/logEntry2" .
signature2 rdf:type "sl:Signature" .
action3 rdf:type "sl:LogMergeAction" .
action3 sl:hasPurpose "dofd:DeliverOrder" .
action3 sl:performedBy "http://schnorr.unikoblenz.de/entity2" .
action3 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action3 sl:hasSequentialNumber "3" .
action3 sl:hasUUID "8d4644b6-c03c-4000-a89a-367e03563235" .
action3 sl:hasTimeStamp "1212408097930" .
action1 rdf:type "sl:CreateAction" .
action1 sl:hasPurpose "dofd:DeliverOrder" .
action1 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action1 sl:performedBy "http://schnorr.unikoblenz.de/entity1" .
action1 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action1 sl:hasSequentialNumber "1" .
action1 sl:hasUUID "6b0c251e-feef-4ebc-bcfd-bb473f6b83b4" .
action1 sl:hasTimeStamp "1212408096317" .
entity1 rdf:type "sl:Entity" .
entity1 sl:hasName "WebShop" .
entity1 sl:hasID "101020301" .
entity1 sl:hasAddress "Koblenz" .
```

```
entity1 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry1" .
entity1 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity1 sl:hasTimeStamp "1212408096317" .
logEntry1 rdf:type "sl:LogEntry" .
logEntry1 sl:logsAction "http://schnorr.unikoblenz.de/action1" .
logEntry1 sl:loggedBy "http://schnorr.unikoblenz.de/entity1" .
logEntry1 sl:hasUUID "f21bf98b-d0d5-4977-9c4c-fb51272764b4" .
logEntry1 sl:hasTimeStamp "1212408096317" .
signature1 rdf:type "sl:Signature" .
signature1 sl:signs "http://schnorr.unikoblenz.de/logEntry1" .
signature1 sl:hasValue "HrdSDFc..." .
signature1 sl:hasUUID "9c0b5c08-1f52-41fb-9a4f-e115bb28e95a" .
signature1 sl:hasTimeStamp "1212408096317" .
action2 sl:hasTimeStamp "1212408097930" .
action2 sl:hasUUID "64f2c9bb-30ab-4d49-a4a8-8a7855d8bd35" .
action2 sl:hasSequentialNumber "2" .
action2 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action2 sl:performedBy "http://schnorr.unikoblenz.de/entity2" .
action2 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action2 sl:hasPurpose "dofd:DeliverOrder" .
action2 rdf:type "sl:UseAction" .
addressDI1 rdf:type "sl:DataInstance" .
addressDI1 sl:isPrimary "true" .
addressDI1 sl:isCopyOf "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasCopy "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isAboutPerson "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasSource "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasUUID "9e7edcf7-5189-4d1a-ad93-51e009d10874" .
14:01:38,050 INFO  [STDOUT] ------=_Part_8_10238269.1212408098050--
14:01:38,080 INFO  [STDOUT] HANDLER WebShop!!!
14:01:38,080 INFO  [STDOUT] HANDLER WebShop outbound!!!

14:01:38,090 INFO  [STDOUT] ------=_Part_9_27762602.1212408098080
14:01:38,090 INFO  [STDOUT] Content-Type: text/xml; charset=UTF-8
14:01:38,090 INFO  [STDOUT] Content-Transfer-Encoding: 8bit
14:01:38,090 INFO  [STDOUT] Content-ID: <rootpart@ws.jboss.org>
14:01:38,090 INFO  [STDOUT] <env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/
    envelope/'><env:Header></env:Header><env:Body><ns2:orderResponse xmlns:ns2="http://
    schnorr.unikoblenz.de/"><return>WebShop answers: Hello Martin Schnorr. You bought the
     package in our shop. We will send the package to Sommerstr. 12, 56133 Fachbach.
LogisticInc answers: We will send the package to Martin Schnorr who lives in Sommerstr.
    12, 56133 Fachbach.

StickyLog:
entity2 sl:hasTimeStamp &quot;1212408097920&quot; .
entity2 sl:hasPGPCertificate &quot;http://sbinc.com/cert.asc&quot; .
entity2 sl:hasLogged &quot;http://schnorr.unikoblenz.de/logEntry2&quot; .
entity2 sl:hasAddress &quot;Berlin&quot; .
entity2 sl:hasID &quot;2378362136&quot; .
entity2 sl:hasName &quot;LogisticInc&quot; .
entity2 rdf:type &quot;sl:Entity&quot; .
logEntry2 sl:hasTimeStamp &quot;1212408097930&quot; .
logEntry2 sl:hasUUID &quot;5ffa354d-c24b-43ce-a893-d6f3278d27c3&quot; .
logEntry2 sl:loggedBy &quot;http://schnorr.unikoblenz.de/entity2&quot; .
logEntry2 sl:logsAction &quot;http://schnorr.unikoblenz.de/action2&quot; .
logEntry2 rdf:type &quot;sl:LogEntry&quot; .
signature2 sl:hasTimeStamp &quot;1212408097930&quot; .
signature2 sl:hasUUID &quot;4fa4aadc-e741-42ca-9baf-e4c7cc726be5&quot; .
signature2 sl:hasValue &quot;GfgfGFc...&quot; .
```

```
signature2 sl:signs &quot;http://schnorr.unikoblenz.de/logEntry2&quot; .
signature2 rdf:type &quot;sl:Signature&quot; .
action3 rdf:type &quot;sl:LogMergeAction&quot; .
action3 sl:hasPurpose &quot;dofd:DeliverOrder&quot; .
action3 sl:performedBy &quot;http://schnorr.unikoblenz.de/entity2&quot; .
action3 sl:leadsTo &quot;http://schnorr.unikoblenz.de/none&quot; .
action3 sl:hasSequentialNumber &quot;3&quot; .
action3 sl:hasUUID &quot;8d4644b6-c03c-4000-a89a-367e03563235&quot; .
action3 sl:hasTimeStamp &quot;1212408097930&quot; .
action1 sl:hasTimeStamp &quot;1212408096317&quot; .
action1 sl:hasUUID &quot;6b0c251e-feef-4ebc-bcfd-bb473f6b83b4&quot; .
action1 sl:hasSequentialNumber &quot;1&quot; .
action1 sl:leadsTo &quot;http://schnorr.unikoblenz.de/none&quot; .
action1 sl:performedBy &quot;http://schnorr.unikoblenz.de/entity1&quot; .
action1 sl:performedOnDataInstance &quot;http://schnorr.unikoblenz.de/addressDI1&quot; .
action1 sl:hasPurpose &quot;dofd:DeliverOrder&quot; .
action1 rdf:type &quot;sl:CreateAction&quot; .
entity1 sl:hasTimeStamp &quot;1212408096317&quot; .
entity1 sl:hasPGPCertificate &quot;http://sbinc.com/cert.asc&quot; .
entity1 sl:hasLogged &quot;http://schnorr.unikoblenz.de/logEntry1&quot; .
entity1 sl:hasAddress &quot;Koblenz&quot; .
entity1 sl:hasID &quot;101020301&quot; .
entity1 sl:hasName &quot;WebShop&quot; .
entity1 rdf:type &quot;sl:Entity&quot; .
action4 rdf:type &quot;sl:LogMergeAction&quot; .
action4 sl:hasPurpose &quot;dofd:DeliverOrder&quot; .
action4 sl:performedBy &quot;http://schnorr.unikoblenz.de/entity1&quot; .
action4 sl:leadsTo &quot;http://schnorr.unikoblenz.de/none&quot; .
action4 sl:hasSequentialNumber &quot;4&quot; .
action4 sl:hasUUID &quot;291c9bb4-d926-4352-bb91-b2f37478edc9&quot; .
action4 sl:hasTimeStamp &quot;1212408098060&quot; .
logEntry1 sl:hasTimeStamp &quot;1212408096317&quot; .
logEntry1 sl:hasUUID &quot;f21bf98b-d0d5-4977-9c4c-fb51272764b4&quot; .
logEntry1 sl:loggedBy &quot;http://schnorr.unikoblenz.de/entity1&quot; .
logEntry1 sl:logsAction &quot;http://schnorr.unikoblenz.de/action1&quot; .
logEntry1 rdf:type &quot;sl:LogEntry&quot; .
signature1 sl:hasTimeStamp &quot;1212408096317&quot; .
signature1 sl:hasUUID &quot;9c0b5c08-1f52-41fb-9a4f-e115bb28e95a&quot; .
signature1 sl:hasValue &quot;HrdSDFc...&quot; .
signature1 sl:signs &quot;http://schnorr.unikoblenz.de/logEntry1&quot; .
signature1 rdf:type &quot;sl:Signature&quot; .
action2 sl:hasTimeStamp &quot;1212408097930&quot; .
action2 sl:hasUUID &quot;64f2c9bb-30ab-4d49-a4a8-8a7855d8bd35&quot; .
action2 sl:hasSequentialNumber &quot;2&quot; .
action2 sl:leadsTo &quot;http://schnorr.unikoblenz.de/none&quot; .
action2 sl:performedBy &quot;http://schnorr.unikoblenz.de/entity2&quot; .
action2 sl:performedOnDataInstance &quot;http://schnorr.unikoblenz.de/addressDI1&quot; .
action2 sl:hasPurpose &quot;dofd:DeliverOrder&quot; .
action2 rdf:type &quot;sl:UseAction&quot; .
addressDI1 sl:hasUUID &quot;9e7edcf7-5189-4d1a-ad93-51e009d10874&quot; .
addressDI1 sl:hasSource &quot;http://schnorr.unikoblenz.de/none&quot; .
addressDI1 sl:isAboutPerson &quot;http://schnorr.unikoblenz.de/none&quot; .
addressDI1 sl:hasCopy &quot;http://schnorr.unikoblenz.de/none&quot; .
addressDI1 sl:isCopyOf &quot;http://schnorr.unikoblenz.de/none&quot; .
addressDI1 sl:isPrimary &quot;true&quot; .
addressDI1 rdf:type &quot;sl:DataInstance&quot; .
</return></ns2:orderResponse></env:Body></env:Envelope>
14:01:38,090 INFO  [STDOUT] ------=_Part_9_27762602.1212408098080
14:01:38,090 INFO  [STDOUT] Content-Type: text/plain
```

```
14:01:38,090 INFO  [STDOUT] Content-Transfer-Encoding: binary
14:01:38,090 INFO  [STDOUT] Content-Id: 0-1212408098090-2270907@ws.jboss.org
14:01:38,090 INFO  [STDOUT] action1 rdf:type "sl:CreateAction" .
action1 sl:hasPurpose "dofd:DeliverOrder" .
action1 sl:performedOnDataInstance "http://schnorr.unikoblenz.de/addressDI1" .
action1 sl:performedBy "http://schnorr.unikoblenz.de/entity1" .
action1 sl:leadsTo "http://schnorr.unikoblenz.de/none" .
action1 sl:hasSequentialNumber "1" .
action1 sl:hasUUID "5faefe38-7072-40db-bbe6-512cccb4ec62" .
action1 sl:hasTimeStamp "1212408098080" .
entity1 rdf:type "sl:Entity" .
entity1 sl:hasName "WebShop" .
entity1 sl:hasID "101020301" .
entity1 sl:hasAddress "Koblenz" .
entity1 sl:hasLogged "http://schnorr.unikoblenz.de/logEntry1" .
entity1 sl:hasPGPCertificate "http://sbinc.com/cert.asc" .
entity1 sl:hasTimeStamp "1212408098080" .
logEntry1 rdf:type "sl:LogEntry" .
logEntry1 sl:logsAction "http://schnorr.unikoblenz.de/action1" .
logEntry1 sl:loggedBy "http://schnorr.unikoblenz.de/entity1" .
logEntry1 sl:hasUUID "98fd858d-e8e3-4a7c-85c7-b446a16430f7" .
logEntry1 sl:hasTimeStamp "1212408098080" .
signature1 rdf:type "sl:Signature" .
signature1 sl:signs "http://schnorr.unikoblenz.de/logEntry1" .
signature1 sl:hasValue "HrdSDFc..." .
signature1 sl:hasUUID "5b396c37-47fc-4ce2-a3b2-727c0c62bfb7" .
signature1 sl:hasTimeStamp "1212408098080" .
addressDI1 rdf:type "sl:DataInstance" .
addressDI1 sl:isPrimary "true" .
addressDI1 sl:isCopyOf "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasCopy "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:isAboutPerson "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasSource "http://schnorr.unikoblenz.de/none" .
addressDI1 sl:hasUUID "1d2dc40d-a27f-4b8d-8276-543bdd4a6bd2" .
14:01:38,090 INFO  [STDOUT] ------=_Part_9_27762602.1212408098080--
```