J. Reichardt Fachbereich Informatik Hochschule Darmstadt

ANLEITUNG ZUM 2. PRAKTIKUMSVERSUCH VERTEILTE SYSTEME

TaskBag-Webserver mit JAX-WS und NetBeans

TEIL A: Zunächst als einfaches Beispiel einen Echo-Webservice realisieren.

A.1 Echo-Webserver erstellen

Projektdateien und -verzeichnisse erzeugen •

• File -> New Project (beachten: Java Web und Web Application auswählen)

🗊 New Project		×
Steps	Choose Project	
1. Choose Project 2	Categories: Java Web Java Web Java EE Java Card Java ME Maven Groovy NetDeans Modules B.C. Samples	Projects: Web Application Web Application with Existing Sources Web Pree-Form Application
	Description: Creates an empty Web application in a IDE-generated build script to build, run	a standard IDE project. A standard project uses an , and debug your project.
	< <u>B</u> ack	Next > Enish Cancel Help

Next Ο

Einstellungen\Apollo\Eigene Dateien\NetBeansProjects Browse
gen\Apollo\Eigene Dateien\NetBeansProjects\EchoServer
toring Libraries
Bgowse
s and projects can share the same compilation Help for details).

o Next



- o Finish
- o Folgende Projektdateien und -verzeichnisse wurden erzeugt:

Projects	I × J Files
Echos	Ferver
📄 🖨 🐨	eb Pages
I i i i i i i i i i i i i i i i i i i i	WEB-INF
	index.jsp
📮 🔂 So	urce Packages
	<default package=""></default>
🗼 💼 宿 Те	est Packages
📋 🕀 違 Lit	oraries
🕒 📴 Те	st Libraries
📄 🗄 🗟 Co	onfiguration Files
- W	

• Echo-Klasse erzeugen

• Rechte Maustaste auf Ordner ,Source Packages' -> New -> Web Service

🗊 New Web Service		×
Steps	Name and Location	
 Choose File Type Name and Location 	Web Service Name: Echo	
	Project: EchoServer	
	Location: Source Packages	•
	Package: echosy	•
	Create <u>Web</u> Service from Scratch	
	C Create Web Service from Existing Session Bean	
	Enterprise Bean:	Browse
	Implement Web Service as Stateless Session Bear	
	<back next=""> Einish Cancel</back>	Help

Der Web Service Name , Echo' wird von NetBeans intern noch durch das Wort , Service' ergänzt.
Die Implementierung als Stateless Session Bean ist für Teil B (Dependency Injection und Namensgebung der Methoden) wesentlich.

- o Finish
- o Im Verzeichnis ,Source Packages' wurde die Datei ,Echo.java' mit der Klasse ,Echo' erzeugt:



• Echo-Methode erzeugen

- o Datei ,Echo.java' öffnen.
- o ,Design' wählen:

Start Page × 🕐 Echo,java × Source Design 🔯 🖬 🍈 100% 💌 🔍 🔍 💭	
EchoService	
Operations (0)	Add Operation Remove Operation

o ,Add Operation' wählen:

🗊 Add Operation			×
<u>N</u> ame: echo			
Return Type: java.lang.String			Browse
Parameters Exceptions			
Name	Type	Final	Add
call	java.lang.String		
			<u>Remove</u>
			Do <u>w</u> n
-			
			1
		OK	Cancel

- o In Datei ,Echo.java' wähle ,Source'.
- o In Webmethode ,echo' ersetze ,return null' durch ,return call' ('call' ist der Eingabeparameter).

• Echo-Server bereitstellen

o Rechte Maustaste auf Ordner 'EchoServer' -> Deploy. Der Output-Log von GlassFish Server 3 ist:



- Die URL in der Ausgabe des GlassFish Server 3 wird in Teil A.2 für den Zugriff auf die WSDL-Datei des Servers benötigt; deshalb bitte diese URL vormerken! - Diese URL wird ebenso für den folgenden Aufruf des generierten Webservice-Testers benötigt.

• Echo-Server testen

- Bevor der Echo-Client erstellt wird, kann der Server bereits durch ein von NetBeans generiertes Testtool (Tester) getestet werden:
- Browser starten. Obige URL um ,?Tester ' ergänzen: <u>http://localhost:8080/Echo-Service/Echo?Tester</u> (beim Start des Testers von einer entfernten Plattform aus ist in der URL die Angabe ,localhost' durch die IP-Adresse des Servers zu ersetzen):



• Echo-Parameter eingeben und Echo-Funktion starten:

🥹 Method invocation trace - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
K 🕹 📮 🚱 C X 🏠 🕞 http://docalbact/0000/EchoSapica/Echo?Tactor
Zurück Vor Neues Fenster Neuer Tab Neu laden Stopp Startseite
Method invocation trace
echo Method invocation
Method parameter(s)
Type Value java lang String Hallo Testeri
Method returned
java.lang.String: "Hallo Tester!"
SOAP Request
xml version="1.0" encoding="UTF-8"? <\$:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"> <\$:Header/> <\$:Header/>
<ns2:echo xmlns:ns2="http://echosv/"></ns2:echo>
<call>Hallo Tester!</call>
SOAP Response
xml version="1.0" encoding="UTF-8"?
<s:envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"> <s:body></s:body></s:envelope>
<ns2:echoresponse xmlns:ns2="http://echosv/"></ns2:echoresponse>
<return>Hallo Tester!</return>

- o Inspizieren Sie das WSDL-File zum weiteren Kennenlernen des SOAP-Protokolls:
 - Überzeugen Sie sich, dass die <soap:address> des Echo-Service mit der vom GlassFish Server ausgegebenen URL übereinstimmt.
 - Öffnen Sie im Browser mit der unter <schemaLocation> angegebenen URL (http://localhost:8080/EchoService/Echo?xsd=1) die XMLSchema-Datei, welche die Syntaxregeln für die SOAP-Kodierung der Echo-Methode enthält.
 - Überzeugen Sie sich, dass der obige SOAP-Request und die SOAP-Response diese Syntaxregeln erfüllen.

A.2 Echo-Java-Client erstellen

- Projektdateien und -verzeichnisse erzeugen
 - File -> New Project (beachten: Java und Java Application auswählen)



o Next

📦 New Java Application			×
Steps	Name and Local	tion	
1. Choose Project 2. Name and Location	Project <u>N</u> ame:	[EchoClient]	
	Project Location:	C:\Dokumente und Einstellungen\Apollo\Eigene Dateien\NetBeansProjects	Browse
	Project Fol <u>d</u> er:	nte und Einstellungen\Apollo\Eigene Dateien\NetBeansProjects\EchoClient	
	Use Dedicated	l Folder for Storing Libraries	
		an	Bro <u>w</u> se
		Different users and projects can share the same compilation libraries (see Help for details).	
	🔽 🖸 create Main C	lass echoclient.Main	
	🔽 Set as <u>M</u> ain Pr	roject	
June			
		<back next=""> Einish Cancel</back>	Help

o Finish

• Folgende Projektdateien und -verzeichnisse wurden erzeugt:



• WSDL-File des Echo-Servers importieren

 Rechte Maustaste auf Ordner ,EchoClient' -> New -> Other (beachten: Web Services und Web Service Client auswählen)

🇊 New File		×
Steps	Choose File Type	
2	Categories: File	Types:
	Java Card 3 Platform Java Java Swing GUI Forms JavaBeans Objects AWT GUI Forms Junit Persistence Groovy Hibernate Web Services XML	RESTful Java Client Web Service Client Logical Handler Message Handler
	Description:	
	Creates a web service client that is compliant wil	h JSR-109.
	< Back	ext > Enish Cancel Help

o Next

Steps	WSDL and Clie	nt Locatio	on				
1. Choose File Type	Specify the WSD	Specify the WSDL file of the Web Service.					
2. WSDL and Client Location	C Project:						
	C Local File:						Bjøwse
	• WSDL URL:	http://loc	alhost:8080/E	choService/Ec	ho?wsdl		Set Proxy
	Specify a packa	je name wh	iere the client	java artifacts	will be generated:		
	Project:	EchoClien	t				
	P <u>a</u> ckage:	echoclier	ht				-
	Client <u>S</u> tyle:	JAX-WS	5tyle				
	🗖 Generate Di	spatch code	e				
				1	1	1	1
			< <u>B</u> ack	Next >	Einish	Cancel	Help

- WSDL URL: Eingabe der URL, die beim Deployen des Echo-Servers im GlassFish Server-Logfile ausgegeben wurde (siehe oben). -> Finish
- o Das WSDL-File wurde importiert und eine Service-Referenz erzeugt:

🇊 EchoClient - NetBea	ns IDE 6.9	.1	
File Edit View Naviga	te Source	Refactor	Run Debu
1 🔁 🔁 😼	5	Contraction (Contraction)	ult config>
Projects 40 × ∫ Fi	les] Se	rvices
EchoClient			
🚊 🕣 Source Packa	jes		
😟 🔠 META-INF			
🖨 📻 META-INF	.wsdl.localh	ost_8080.E	choService
Echo.	wsdl		
Echo.	xsd_1.xsd		
🖻 🔚 echoclient			
🚳 Main. j	ava		
🕀 💼 Test Packages	5		
🕀 🔂 Generated So	urces (jax-w	s)	
📄 🔂 Web Service P	References		
🖻 🔟 Echo			
🖻 🧕 EchoS	ervice		
🖻 – 📷 Ed	hoPort:		
	echo		

• Aufruf der Service-Methode *echo()*

- Öffnen der EchoClient-Datei ,Main.java' und Expandieren der ,Web Service References' bis zum Methoden-Port ,echo'.
- Leerzeile in ,main()' einfügen, Klick auf den Methoden-Port (rote Kugel) und Drag&Drop des Ports in die Leerzeile von ,main()'.



- In der Klasse "Main' wurde außerhalb der Methode "main()' eine Definition für die Methode "echo()' eingefügt. Die Leerzeile in "main()' ist weiterhin leer. In diese Leerzeile kann nun ein Aufruf der Methode "echo()' eingefügt werden, z.B. innerhalb einer Ausgabeanweisung mit "System.out.println()'.
- Starten des EchoClients
 - Rechte Maustaste auf Ordner ,EchoClient' -> Run.

A.3 Remote Echo-Java-Client erstellen

• Projektdateien und -verzeichnisse erzeugen (wie für lokalen Client)

• WSDL-File des entfernten Echo-Servers importieren

 Anstatt ,localhost' wird in der WSDL-URL die IP-Adresse des entfernten Servers angegeben. Net-Beans setzt voraus, dass der Server unter derselben IP-Adresse wie die WSDL-Datei zu erreichen ist (in der WSDL-Datei unter <soap:address> nachzuprüfen). Alles weitere wie in A.2.

teps	WSDL and Clie	nt Location		
1. Choose File Type	Specify the WSDL file of the Web Service.			
. WSDL and Llient Location	C Project:	Browsg		
	C Local File:	Biowse		
	• WSDL <u>U</u> RL:	http://141.100.42.99:8080/EchoService/Echo?wsdl Set Proxy		
	Specify a packag	e name where the client java artifacts will be generated:		
	Project:	RemoteEchoClient		
	P <u>a</u> ckage:	remoteechoclient		
	Client <u>S</u> tyle:	JAX-WS Style		
	🥅 Generate Di	spatch code		

TEIL B: TaskBag-Webservice realisieren

B.1 TaskBag-Webserver erstellen

- Projektdateien und -verzeichnisse erzeugen (wie für Echo-Service)
- TaskBag-Klasse erzeugen (wie Echo-Klasse)
- TaskBag-Methoden putTask(), getTask() und readTask() erzeugen (wie echo()-Methode)
- Task als Tabellenstruktur erstellen
 - o Rechter Mausklick auf "Source Packages/taskbagserver' -> New -> Java Class

🗊 New Java Class		>
Steps	Name and Location	
 Choose File Type Name and Location 	Class Name: Task	
	Project: TaskBagServer	-
	Location: Source Packages	•
	Package: taskbagserver	•
	Created File: ollo\Eigene Dateien\NetBeansProjects\TaskBagServer\src\java\taskbagserver\Task.ja	ava
X		
	< Back Next > Enish Cancel Help	

- o Finish
- o Datei 'Task.java' öffnen und Strukturelemente Id, Type, Descr und Done (siehe Aufgabenstellung) als Attribute der Klasse Task einfügen. Danach eine Leerzeile einfügen.
- Rechter Mausklick auf die Leerzeile -> Insert Code -> **Getter and Setter**. Für alle Attribute Get- und Set-Methoden generieren lassen. Damit ist die Task-Struktur vollständig definiert.
- TaskBagTable als Singleton-Speichermedium (an Stelle einer Datei) erstellen
 - o Rechter Mausklick auf ,Source Packages/taskbagserver' -> New -> SessionBean

🗊 New Session Bean	×
Steps	Name and Location
1. Choose File Type 2. Name and Location	EJB Name: TaskBagTable
	Project: TaskBagServer
	Location: Source Packages
	Package: taskbagserver
	Session Type: C StateEss C Stateful C Bingleton Create Interface: Local Remote
	< <u>Back</u> Next > Einish Cancel <u>H</u> elp

o Finish

- o In der Datei , TaskBagTable.java' die Schnittstelle für die , TaskBagTable' definieren. Vorschlag:
 - void addTask(String type, String descr);
 - String getNextTaskByType(String type);
 - String readNextTaskByType(String type);
- Dazu rechter Mausklick innerhalb der Klasse 'TaskBagTable' -> Insert Code -> Add Business Method

Add Business Method			×
Name: addTask Return Type: void Parameters Exceptions			Browge
Name	Туре	Final	Add
type	java.lang.String		
descr	java.lang.String		<u>Felliove</u>
			Up Down
Use in Interface: 🌀 Loca	I 🔿 Rgmote 🔿 Both		
		ОК	Cancel

- o OK (zwei mal), und für jede weitere Methode ,Add Business Method' wiederholen.
- In der Datei ,TaskBagTable.java' als Attribut der Klasse ,TaskBagTable' die physikalische TaskBag-Tabelle definieren. Vorschlag:
 private Map<Integer, Task> taskbag = new HashMap<Integer, Task>();
 private int maxkey = 0; // zeigt stets auf den ersten freien Eintrag
- Die Methoden der TaskBagTable-Schnittstelle sind auszuprogrammieren.
- Singleton ,TaskBagTable' benutzen
 - In Datei ,TaskBag.java' rechte Maustaste auf eine Leerzeile am Anfang der TaskBag-Klasse -> Insert Code -> Call Enterprise Bean

🗊 Call Enterprise Bean			X	J
Select an enterprise bean from open projects.				
B	er			
Reference Name:	TaskBagTable			
Referenced Interface:		C Local	C Remote	
	ОК	Cancel	Help	

• OK. Dadurch wird in der Webservice-Klasse ,TaskBag' in Form einer EJB-Annotation eine Referenz auf die ,TaskBagTable' hergestellt, die beim Deployen per Dependency-Injection initialisiert wird.

o Die Webservice-Klasse ,TaskBag' sollte nun folgende Form haben:

```
@WebService()
@Stateless()
public class TaskBag {
@EJB
private TaskBagTable taskBagTable;
/**
* Web service operation
*/
@WebMethod(operationName = "putTask")
public String putTask(@WebParam(name = "type")
String type, @WebParam(name = "descr")
String descr) {
      //TODO write your implementation code here:
      return "ok";
}
•••
}
```

o Insgesamt wurden bisher folgende Projektverzeichnisse und -dateien erzeugt:



- Rechte Maustaste auf ,TaskBagServer' -> Clean and Build.
- Damit steht die Architektur des TaskbagServers bereit, mit allen Komponenten und ihren Kommunikations- und Benutzungsbeziehungen. Jetzt sind nur noch die Methoden der TaskBag-Schnittstelle auszuprogrammieren.

TaskBag-Methoden ausprogrammieren

 Die TaskBag-Methoden getTask() und readTask() sind server-seitig blockierend zu implementieren, das heißt, dass die Implementierung von getTask() innerhalb einer Schleife solange die TaskBagTable-Methode getNextTaskByType() aufruft bis diese einen Returnwert ungleich null zurückgibt. Erst dann erhält ein Client den Returnwert von getTask() und wird damit entblockiert. Entsprechendes gilt für die Methode readTask().

Die TaskBagTable-Methoden getNextTaskByType(), usw. sind dagegen nicht-blockierend zu imple-0 mentieren, d.h. ihr Aufruf liefert immer sofort einen Returnwert (null oder ungleich null).

TaskBagServer bereitstellen und testen

- Rechte Maustaste auf TaskBagServer -> Deploy
- Die URL des TaskBagServers in der Ausgabe von GlassFishServer3 für den späteren WSDL-Import 0 auf Client-Seite vormerken (wie für Echo-Server und -Client)
- Tester-Aufruf (wie für Echo-Server) 0
- Blockieren und Entblockieren der Serveraufrufe testen: 0
 - putTask(garten, rasenmaehen) // Trägt einen Task ein
 - getTask(garten)
 - getTask(garten)
- // Liest und löscht den Task
- putTask(garten, rasenmaehen)
- // 2. Aufruf blockiert, da kein passender Task vorhanden
- // Entblockiert den 2. Aufruf durch Eintrag eines passen-// den Task

B.2 TaskBag-Java-Clients erstellen und testen

Es sind zwei Clients zu erstellen, die später separat in unterschiedlichen Reihenfolgen gestartet werden, um das Blockieren bzw. Entblockieren der Aufrufe zu testen. Der erste Client soll zunächst nur getTask() aufrufen, der zweite Client putTask().

- Projektdateien und -verzeichnisse für Client 1 erzeugen (wie für Echo-Client) .
- In Client 1 das WSDL-File des Echo-Servers importieren (wie für Echo-Client)
- Aufruf der Service-Methode getTask() (wie echo()-Methode) •
- Projektdateien und -verzeichnisse für Client 2 erzeugen (wie oben) •
- In Client 2 das WSDL-File des Echo-Servers importieren (wie oben)
- Aufruf der Service-Methode putTask() (wie oben)

Testen des TaskBag-Webservice mit den beiden Java-Clients gemäß Aufgabenstellung. .

Zur Info: Für jeden Aufruf einer Web-Methode wird vom GlassFish-Application-Server aus einem Pool eine neue TaskBagBean geholt (vergleichbar mit dem Start eines neuen Prozesses oder Threads). Alle diese Komponenten greifen stets auf dieselbe ,TaskBagTable' (Singleton) zu. Diese Zugriffe sind, im Gegensatz zu den Dateizugriffen in Aufgabe 1, default-mäßig voll synchronisiert (container-managed concurrency).

B.3 Remote TaskBag-Java-Clients erstellen und testen Wie in A.3.