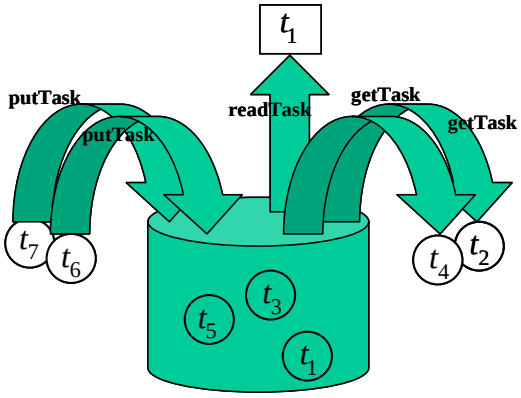
Prof. Dr. J. Reichardt

Fachbereich Informatik
Hochschule Darmstadt

Praktikum
Verteilte Systeme

TASK BAG



Im Rahmen dieses Praktikums soll in verschiedenen technischen Varianten eine Auftragsverwaltung implementiert werden, welche von entfernten Clients Aufträge (tasks) entgegennimmt und an entfernte Clients Aufträge zur Bearbeitung herausgibt.

Jeder Auftrag ist ein Objekt mit folgenden Attributen:

```
class Task {

      int id;
      String type;
      String descr;
      int done;
}
```

Drei Operationen sollen dazu dienen, Aufträge zu vergeben (*putTask*), zu übernehmen (*getTask*) und zu inspizieren (*readTask*):

```
public interface TaskBagService {

    public String putTask(String type, String descr);
    public String getTask(String type);
    public String readTask(String type);
}
```

Die Operation *putTask()* trägt eine *task*-Struktur in den *task*-Container (Task Bag) ein. Operation *getTask()* entnimmt eine *task*-Struktur aus dem Container, wobei diese Struktur gleichzeitig im Task Bag als „erledigt“ markiert wird (*done = 1*). Die Operation *readTask()* unterschei-

det sich von *getTask()* lediglich dadurch, dass der gelesene Task weiter unmarkiert im Task Bag verbleibt (*done = 0* unverändert).

getTask() und *readTask()* sind zunächst client- bzw. serverseitig blockierend zu implementieren. Das bedeutet, dass ein synchroner rufender Client seine Arbeit erst fortsetzen kann, nachdem ein Auftrag mit dem gewünschten *Type* im *Task*-Container vorhanden war und zum Client übertragen wurde. Während ein Auftragnehmer-Client blockiert, soll gleichzeitig ein Auftraggeber-Client einen Auftrag vom gewünschten *Type* im Container platzieren können, wodurch die Blockade des Auftragnehmers wieder aufgehoben wird. Im Laufe des Praktikums sind sowohl client- bzw. serverseitig synchron-blockierende als auch asynchron-nicht-blockierende, d.h. lose gekoppelte Lösungen zu entwickeln.

Verschiedene Auftraggeber- und Auftragnehmer-Clients sollen jeweils plattformübergreifend einen Task-Bag benutzen. Jede Praktikumsgruppe hat einen Task-Bag zu implementieren. Da der Schwerpunkt des Praktikums auf der Kommunikation und den Schnittstellen liegt, sollte die Implementierung des TaskBag möglichst einfach sein; für Variante 1 (siehe unten) beispielsweise als Datei mit Sätzen fester Länge, für Variante 2 als Singleton Session Bean mit einer HashMap. Es muss lediglich gewährleistet sein, dass die evtl. mehrfachen Server-Instanzen auf genau einer Speicher-Instanz des TaskBag arbeiten. Variante 3 benutzt keine eigene TaskBag-Implementierung, sondern stützt sich ausschließlich auf die Speicherkapazität des Java Messaging Service. Unter zusätzlicher Verwendung von Message Driven Beans soll so ein nicht-blockierender, vollständig asynchroner Service realisiert werden.

Praktikumsdurchführung:

- Für das Praktikum sind gute Programmierkenntnisse erforderlich. Die Praktikumsaufgaben sind jeweils bis zum Praktikumstermin konzeptionell vorzubereiten, insbesondere ist ein detaillierter Entwurf für Programmkonstrukte wie Dateizugriff oder HashMap-Zugriff zum Praktikumstermin mitzubringen.

- Die Praktikumsaufgaben werden im wesentlichen mit der JavaEE-Entwicklungsumgebung NetBeans durchgeführt. Ausgenommen ist die erste Aufgabe, die am Beispiel des Sun-RPC das Prinzip der IDL-Compilers veranschaulichen soll. Der oben beschriebene Task-Bag-Service ist in folgenden Varianten zu implementieren.

    - Variante 1: Realisierung als Sun-RPC mithilfe des IDL-Compilers *rpcgen*.
    - Variante 2: Realisierung als Web-Service (JAX-WS) mit Java-Clients.
    - Variante 3: Realisierung als asynchroner Publisher-Subscriber-Dienst basierend auf Message Driven Beans und Java Messaging Service.

Literatur:

1. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems - Concepts and Design. 4th edition, Pearson-Education 2009
2. Goncalves, A.: Beginning Java EE 6 Platform with GlassFish 3 – From Novice to Professional, Apress 2010
3. Sun-Tutorial Java EE 6: http://download.oracle.com/javaee/6/tutorial/doc/
4. NetBeans-Tutorial: http://netbeans.org/kb/docs/websvc/jax-ws.html

1. Praktikumsübung

IDL-Compiler

1. In dieser Praktikumsaufgabe soll die Kommunikation zwischen Client und Server mittels Sun-RPC realisiert werden. Ausgangspunkt der Implementierung ist die IDL-Beschreibung der Service-Prozeduren *getTask(), putTask()* und *readTask()* in der Datei *taskbag.x*.

2. Ein Beispiel für eine Schnittstellen-Beschreibung befindet sich bereits in der **Datei *taskbag.x*** (unter *http://141.100.42.156/VS-Reichardt/Praktikum_1/*). Sie enthält die IDL-Spezifikation einer einfachen Echofunktion. Als Erstes ist diese Echofunktion auszuprogrammieren: a) Erzeugen und Übergabe der RPC-Parameter auf Client-Seite. b) Entnahme der RPC-Parameter auf Server-Seite und Zuweisung eines Returnwertes. c) Ausdrucken des Returnwertes auf Client-Seite. Durch Anpassung dieses Beispiels sind dann die TaskBag-Signaturen *getTask(), putTask()* und *readTask()* zu spezifizieren. Beachten: Mehrere Funktionsparameter müssen stets zu einer Parameter-Struktur zusammengefasst werden.

3. Für die einzelnen Implementierungsschritte ist die in der Vorlesung herausgegebene **Versuchsanleitung *VS-Vorlesung/VS-Praktikum/VS-Prak-Aufgabe-1-ANLEITUNG. doc*** zu verwenden, insbesondere das darin beschriebene Make-Skript ***taskbag_make*** (aus demselben Verzeichnis zu kopieren wie *taskbag.x*).

4. Der IDL-Compiler *rpcgen* generiert aus der IDL-Spezifikation die Client- und Server-Stubs, eine Header-Datei, die Parameter-Konvertierungsroutinen, sowie zwei Programm-Templates für Client und Server.

5. Das Client-Programm ist mit den Aufrufen der TaskBag-Funktionen zu ergänzen, das Server-Programm mit der Implementierung der TaskBag-Funktionen. Die Methoden *getTask()* und *readTask()* sind **clientseitig-blockierend** zu implementieren. Das heißt, dass der Client-Prozess in einer Warteschleife solange die Methoden *getTask()* bzw. *readTask()* aufruft bis sie einen Returnwert ungleich Null zurückliefern.

6. Danach ist das Programm entsprechend der Anleitung zu compilieren und zu testen. Dabei ist zu beachten: 1) Clients und Server können nur dann miteinander kommunizieren, wenn sie dieselbe Schnittstelle inklusive der dort angegebenen Identifier benutzen. 2) Das Sun-RPC-Binding benötigt einen *Portmapper*; auf eigenem Laptop evtl. nachzuinstallieren.
    a. Der Server ist mit ***./taskbag_server*** zu starten.
    b. Ein lokaler Client wird mit ***./taskbag_client localhost [arg1 arg2]*** gestartet,
    c. ein entfernter Client mit ***./taskbag_client <server-IP-address> [arg1 arg2]***.

7. Für den Test ist folgendes Nutzungsszenario durchzuspielen: Der Server wird mit leerem TaskBag gestartet. Danach wird ein Client mit *getTask()*-Aufrufschleife gestartet. Die Schleife kann vom Client-Prozess nicht verlassen werden, da der TaskBag noch keinen passenden Task zurückliefern kann. Nun wird ein weiterer Client mit *putTask()*-Aufruf und passendem *Type*-Parameter gestartet. Dieser Auftrag wird in den TaskBag eingetragen. Der nächste *getTask()*-Aufruf kann nun den Auftrag aus dem TaskBag lesen und ihn dem wartenden Client-Prozess zustellen, der damit entblockiert wird.

2. Praktikumsübung

Webservice mit Java-Clients

1. In dieser Praktikumsaufgabe soll die Kommunikation zwischen TaskBag-Server und Clients mithilfe des Protokolls SOAP/HTTP realisiert werden. Dazu wird das Java API für XML Web Services (JAX-WS) verwendet.

2. Die Entwicklungsumgebung ist NetBeans 6.9.1 für Java EE 6 und Enterprise JavaBeans 3.1. Nach dem Hochfahren von NetBeans ist die Java-Datenbank Derby zu starten (unter ‚Services': rechte Maustaste auf ‚Databases/JavaDB' -> Start Server), danach der Application-Server GlassFish 3 (unter ‚Services': rechte Maustaste auf ‚Servers/GlassFish Server 3' -> Start).

3. Alle weiteren Implementierungsschritte sind in der in der Vorlesung herausgegebenen **Versuchsanleitung** ***/VS-Vorlesung/VS-Praktikum/VS-Prak-Aufgabe-2-ANLEITUNG.doc*** beschrieben.

4. Die Praktikumsaufgabe besteht aus zwei Teilen. Im Teil A soll zunächst als einfaches Beispiel eine Echo-Funktion als Webservice realisiert werden. Im Teil B sollen dann dieselben Implementierungsschritte angewandt werden, um damit die Grobstruktur des TaskBag-Servers zu erstellen. Diese Grobstruktur ist dann um eine Speicher-Komponente zu erweitern. Dazu wird eine Singleton-SessionBean als gemeinsames Speichermedium für alle evtl. gleichzeitig laufenden Instanzen des TaskBag-Servers benutzt.

5. Die Methoden *getTask()* und *readTask()* sind serverseitig-blockierend zu implementieren. Das heißt, dass der Server-Prozess in einer Warteschleife solange die *get-* bzw. *read-*Methode der Speicher-Komponente aufruft bis sie einen Returnwert ungleich Null zurückliefert. Erst dann wird dieser Returnwert an den Client weitergegeben, der damit entblockiert wird.

6. Für den Test ist entsprechend folgendes Nutzungsszenario durchzuspielen: Der Server wird mit leerem TaskBag gestartet. Danach wird ein Client mit *getTask()*-Aufruf gestartet, der server-seitig blockiert. Der Aufruf eines weiteren Clients mit *putTask()*-Aufruf trägt einen Task mit passendem *Type*-Parameter in die TaskBagTable ein. Der nächste *getTask()*-Aufruf kann nun den Auftrag aus dem TaskBag lesen und ihn dem wartenden Client-Prozess zustellen, der damit entblockiert wird.

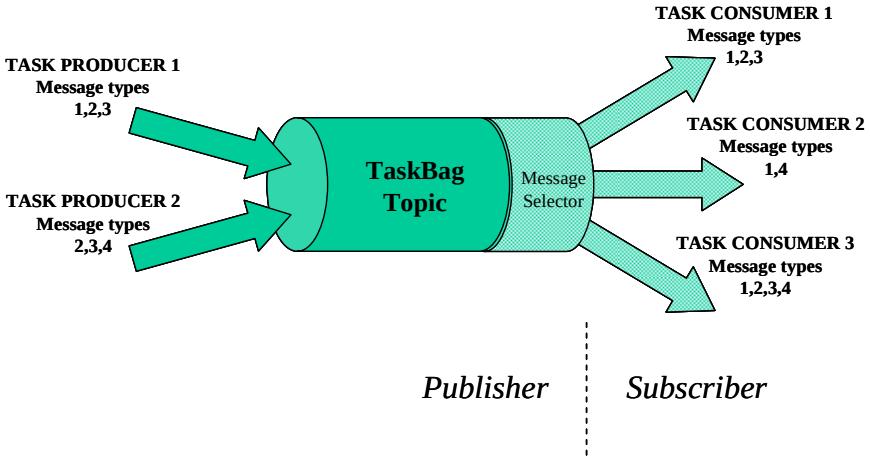
7. Für den Echo- und den TaskBag-Service sind sowohl lokale als auch entfernte Clients zu implementieren. Für einen entfernten Client ist jeweils das WSDL-File des Servers über das Netz zu importieren. Der obige Test ist auch für die entfernten Clients durchzuführen.

3. Praktikumsübung

Publisher-Subscriber
mit JMS und Message Driven Beans

1. In dieser Praktikumsaufgabe wird die Kommunikation zwischen TaskBag-Server und Clients nach dem Prinzip der losen Kopplung, d.h. ohne Blockieren auf Client- oder Serverseite realisiert. Für die Annahme, Speicherung und Verteilung der Tasks soll hierzu die Speicher- und Funktionsschnittstelle des Java Messaging Service genutzt werden, die komfortabel über Message Driven Beans mit ihren Annotationen zur Verfügung gestellt wird.

2. Die ursprüngliche Aufgabenstellung wird leicht angepasst: Jeder Auftrag soll per Broadcast grundsätzlich allen Auftragnehmern gleichzeitig zugestellt werden können. Jeder Auftragnehmer soll allerdings auch individuell verhindern können, dass ihm unerwünschte Aufträge zugestellt werden (sowohl Broadcast als auch Message-Selector sind Grundfunktionen des Java Messaging Service).



3. Alle Implementierungsschritte sind in der in der Vorlesung herausgegebenen **Versuchsanleitung** ***/VS-Vorlesung/VS-Praktikum/VS-Prak-Aufgabe-3-ANLEITUNG. doc*** beschrieben.

4. Die Praktikumsaufgabe besteht aus zwei Teilen. Im Teil A ist ein TaskProducer-Client (als EJB Module) zu implementieren, der eine TextMessage in eine TaskBag-Topic schreibt. Ein TaskConsumer-Client ist als Message Driven Bean mit der *onMessage()*-Methode zu implementieren. Die TextMessage wird dem Client sofort nach ihrem Eintreffen in der Topic gemäß dem Push-Modell zugestellt. Der Text der Nachricht soll auf Producer- als auch auf Consumer-Seite jeweils ausgegeben werden.

5. Im Teil B wird die TaskBag-Topic um einen MessageSelector erweitert. Dazu soll der TaskProducer-Client nun eine längere Serie von Nachrichten (z.B. 16) von unterschiedlichem Typ erzeugen. Einer Nachricht wird jeweils mittels der Methode *setStringProperty („task Type“, „garten“)* eine willkürlich gewählte Typ-Variable zugeordnet (hier: *„task-Type“*), die auf willkürliche Werte gesetzt werden kann (hier: *garten, wohnung, auto* oder *hund*). Auf Consumer-Seite kann anschließend durch die Annotation *@ActivationConfig*

Property (propertyName = „messageSelector“, propertyValue = „taskType = ‚garten’“) dafür gesorgt werden, dass dem Consumer aus der Topic nur Nachrichten vom gewünschten Typ (hier: *‚garten’*) zugestellt werden. Der Java Messaging Service sieht vor, dass der *propertyValue* auch SQL-Ausdrücke enthalten kann. Im folgenden ist ein Ausschnitt der zulässigen *Selector expression syntax* angegeben:

Element	Valid values
Identifiers	• Property or header field reference (such as JMSPriority, myProperty1, and so on) • Cannot be NULL, TRUE, FALSE, NOT, AND, OR, BETWEEN, LIKE, IN, IS
Operators	AND, OR, LIKE, BETWEEN, =, <>, <, >, <=, >=, IS NULL, IS NOT NULL
Literals	• Boolean literals are TRUE and FALSE • Exact number literals have no decimal point (for example, +25, -399, 40, and so on) • Approximate number literals can use scientific notation or decimal (for example, -21.4E4, 5E2, +34.4928, and so on)
Examples	• shirtType IN ('polo', 'tee', 'sweater') • (JMSPriority > 4 AND shirtType LIKE '%polo%') OR shirtType = 'tee' • JMSPriority BETWEEN 4 and 7

Bilden Sie für den Consumer-Client verschiedene Selector-Bedingungen mit den Operatoren *OR*, *IN* und *LIKE*, und testen Sie diese.

6. Ergänzen Sie den bisherigen Consumer-Client um zwei weitere Consumer-Clients mit jeweils unterschiedlichen Selector-Bedingungen. Der Nachrichtenstrom soll nun den Clients entsprechend ihren Selector-Bedingungen zugestellt werden. Die Ausgabe auf Producer- und Consumer-Seite sollte so übersichtlich und aussagekräftig sein, dass die Erzeugung und Verteilung der Nachrichten leicht verfolgt werden kann.

7. Die lose Ankopplung eines Consumers, die bereits durch den asynchronen Aufruf der *onMessage()*-Methode bewirkt wurde, soll jetzt noch erweitert werden. Nachrichten für sogenannte „durable subscribers“, die vorübergehend nicht angekoppelt (d.h. undeployed) sind, werden in der JMS-Topic gespeichert und bei der nächsten Ankopplung (deploy) an den Consumer weitergeleitet. Dazu ist zunächst in der @MessageDriven-Annotation des Consumers die entsprechende ActivationConfigProperty zu setzen: *@ActivationConfigProperty(property Name = "subscriptionDurability", propertyValue = "Durable")*. Anschließend sind die folgenden Testschritte durchzuführen: (a) Undeploy der Consumer 1 und 2. (b) Run des Producers. (c) Deploy von Consumer 1. (d) Deploy von Consumer 2. Nach jedem Testschritt sind die Testausgaben zu überprüfen.