Prof. Dr. Arnd Poetzsch-Heffter
Dipl.-Inf. Jan Schäfer
Dipl.-Inf. Jean-Marie Gaillourdet
Dipl.-Inform. Markus Reitz

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Aufgabenblatt 1: Praktikum Komponententechnik (WS 06/07)

Ausgabe: 24. Oktober 2006

Motivation

In diesem Aufgabenblatt geht es darum die Tools die im Laufe des Praktikums verwendet werden einmal in der Praxis anzuwenden. Dazu sollen Sie ein einfaches Client-Server-Protokoll implementieren. Die Hauptaufgabe besteht darin die volle Tool-Chain aufzubauen, die für die Softwareentwicklung nötig ist. Die Implementierung des Protokolls dient dazu, dass Sie sich mit den Java-Klassen vertraut machen die zur Client-Server-Kommunikation nötig sind. Dieses Aufgabenblatt sollen Sie jeweils in zweier/dreier Gruppen komplett bearbeiten.

Aufgabe 1 Versionsverwaltung

Als Erstes werden Sie das Versionsverwaltungstool *Subversion* (`subversion.tigris.org`) kennenlernen. Sie sollten alle einen Subversion-Account von Nicole Rauch erhalten haben. Auf den SCI-Rechnern kann Subversion mit dem Befehl `svn` aufgerufen werden.

a) Checken Sie das Verzeichnis `https://softech.informatik.uni-kl.de:2443/svn/Studenten/trunk/06_WS_Praktikum/Intro/Gruppe<Nummer>` mit Subversion aus, wobei `<Nummer>` durch die Nummer ihrer Gruppe ersetzt werden muss.

b) Legen Sie in diesem Verzeichnis eine leere Datei mit dem Namen `build.xml` an und fügen Sie sie zum Subversion-Repository hinzu.

c) Committen Sie diese Änderung mit dem Kommentar „build.xml Datei hinzugefügt".

d) Legen Sie folgende Verzeichnisse an, fügen Sie sie dem Repository hinzu und committen Sie mit passenden Kommentaren: `server`, `client`, `common`.

Aufgabe 2 Build-Infrastruktur

Nun geht es darum eine Build-Infrastruktur mit dem Werkzeug *Ant* (`ant.apache.org`) aufzubauen.

a) Erstellen Sie die folgenden Java-Dateien mit entsprechenden Klassen: `server/src/server/Main.java`, `client/src/client/Main.java`. Beide Klassen sollen eine `main`-Methode enthalten, um jeweils den Server und den Client zu starten.

b) Erstellen Sie jeweils ein Ant-Target für Server und Client in der `build.xml`-Datei, um die jeweiligen Programme zu übersetzen. Die `.class`-Dateien sollen dabei in das Verzeichnis `server/bin`, bzw. `client/bin` erzeugt werden und *nicht* in die `src`-Verzeichnisse. Fügen Sie zusätzlich ein Target „clean" ein, mit dem man die generierten `.class` Dateien wieder löschen kann. Es soll danach möglich sein die folgenden Kommandos in ihrem Verzeichnis auszuführen:

- `ant build-server` - übersetzt den Server
- `ant build-client` - übersetzt den Client
- `ant clean` - entfernt alle `.class`-Dateien (Achtung: Vor dem Testen Sicherheitskopie machen!)

Aufgabe 3 Implementierung eines einfachen Client-Server-Protokolls

Implementieren Sie nun das folgende Client-Server-Protokoll. Dokumentieren Sie dabei alle Klassen und Methoden mit *JavaDoc* (`http://java.sun.com/j2se/javadoc/`).

Das Protokoll sieht folgendermaßen aus. Der Client kann an den Server eine Liste von positiven ganzen Zahlen schicken und der Server sortiert die Zahlen der Größe nach und schickt die sortierte Liste an den Client zurück. Eine Anfrage besteht aus mehreren Text-Zeilen. Die erste Zeile lautet immer `REQUEST START`, die letzte Zeile immer `REQUEST END`. Dazwischen kann der Client Zeilen mit je einer Zahl senden. Eine Anfrage könnte z.B. so aussehen:

```
REQUEST START
2
3
1
4
REQUEST END
```

Der Server sortiert dann die Zahlen und antwortet mit einer Liste von Zahlen, die in `ANSWER START` und `ANSWER END` eingeschlossen ist. In dem Beispiel würde der Server antworten mit:

```
ANSWER START
1
2
3
4
ANSWER END
```

Aufgabe 3.1 Implementierung des Server

Der Server soll auf einem beliebigen per Kommandozeilen-Parameter einstellbaren Port lauschen und auf Client-Anfragen warten. Wenn eine Client-Anfrage eintrifft soll der Server die Zahlen der Nachricht sortieren und an den Client zurücksenden. Falls das Format der Anfrage fehlerhaft war soll der Server die Zeile „ERROR: Wrong message format" an den Client schicken.

a) Implementieren Sie den Server. Legen Sie alle Quelldateien des Servers im Verzeichnis `server/src` ab, falls Sie glauben, dass bestimmte Teile der Implementierung auch vom Client verwendet werden könnten, legen Sie diese Teile im Verzeichnis `common/src` ab. Ändern Sie die `build.xml`-Datei so ab, dass Klassen aus dem `common/src`-Verzeichnis, nach `common/bin` kompiliert werden.

b) Erstellen Sie ein Shell-Skript mit dem Namen `server.sh`, so dass der Server mit dem Aufruf `./server.sh 30000` gestartet werden kann und auf Anfragen von Clients auf Port 30000 wartet.

Aufgabe 3.2 Implementierung des Client

Der Client soll auf der Kommandozeile die Addresse des Servers, den Port und eine Liste von Zahlen erwarten. Er soll dann die Zahlenliste an den Server schicken und auf die Antwort des Servers warten. Wenn der Server antwortet soll auf der Standardausgabe die sortierte Liste ausgeben werden. Bei falschen Parametern oder wenn der Server nicht erreichbar ist, soll der Client entsprechende Fehlermeldungen ausgeben.

a) Implementieren Sie den Client. Verfahren Sie mit den Quelldateien wie beim Server.

b) Erstellen Sie ein Shell-Skript mit dem Namen `client.sh`, so dass der Client mit dem Befehl `./client.sh localhost 30000 3 4 2 1` gestartet werden kann und eine Anfrage an den Server mit der Adresse „localhost" und Port 30000 stellt. Wenn der Server vorher entsprechend gestartet wurde, soll der Client die Zahlenliste des Servers auf der Standardausgabe in einer Zeile ausgeben: `1 2 3 4`.

Aufgabe 3.3 JavaDoc

Fügen Sie zur `build.xml`-Datei ein neues Target „javadoc" hinzu, das es ermöglicht aus den JavaDoc-Annotationen eine API-Dokumentation zu generieren. Die API-Dokumentation soll dabei in die Verzeichnisse `<modul>/doc/api` der jeweiligen Module generiert werden.

Aufgabe 4 Erstellen von Unit-Tests

Nach der Implementierung geht es nun ans Testen. Eigentlich sollte man parallel zur Entwicklung auch die Unit-Tests schreiben. Wir verfahren hier nun außnahmsweise mal so, dass die Tests nach der eigentlichen Entwicklung geschrieben werden.

Zum Testen der Implementierung verwenden wir das Werkzeug *JUnit* in der Version 3.8.x (`www.junit.org`). Eine gute Einführung zu JUnit ist unter `http://www.linux.ie/articles/tutorials/junit.php` zu finden. Alle Quelldateien der Unit-Tests sollen im `<modul>/test/src`-Verzeichnis der jeweiligen Module untergebracht werden. Die `.class`-Dateien sollen in die Verzeichnisse `<modul>/test/bin` erzeugt werden.

a) Erstellen Sie einen JUnit-Test zum Testen der Anfrage-Parser-Komponente ihrer Implementierung. Testen Sie dabei, dass ihr Parser gültige Anfragen richtig parst und bei ungültigen Anfragen entsprechend fehlschlägt.

b) Erstellen Sie einen JUnit-Test, um die Sortierfunktion des Servers zu überprüfen.

c) Passen sie die `build.xml`-Datei so an, dass man mit dem Befehl `ant test`, alle JUnit-Tests ausführen kann. Achten Sie darauf, dass die Tests vorher auch übersetzt werden.

Aufgabe 5 Alle Änderungen committen

Committen Sie alle Ihre Änderungen bis spätestens zum nächsten Treffen am 31.10.06.

Viel Spaß!