ENTWURF

ENTWURF zur aufgabe 4 aus der vorlesungsreihe “VERTEILTE SYSTEME”

TEAM 1: MICHAEL STRUTZKE, iGOR arkhipov

Aufgabenaufteilung

Alle Aufgaben wurden gemeinsam entworfen und bearbeitet.

## Quellenangaben

Aufgabenstellung, Vorstellung der Aufgabe in der Vorlesung.

## Bearbeitungszeitraum

|  |  |
| --- | --- |
| Datum | Dauer in Stunden |
| 03.12.16 | 5 |
| 04.12.16 | 3 |
| 10.12.16 | 5 |
| 11.12.16 | 5 |
| 13.12.16 | 3 |
|  |  |
| Gesamt | 21 |

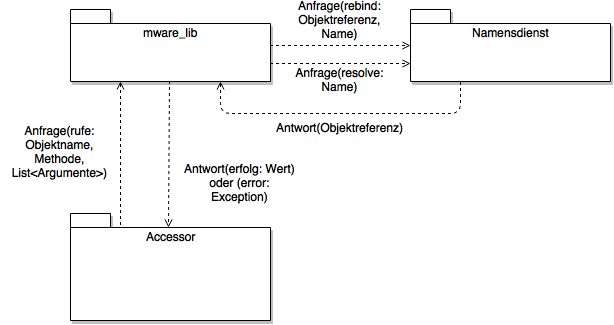
## Aktueller Stand

Der Entwurf ist fertig. Die Implementation wird eingesetzt.

## Änderungen im Entwurf

Noch keine.

## Details zum Entwurf:



Es soll eine Middleware implementiert werden, die es ermöglicht, Methoden auf entfernten Objekten auszuführen.

Dem Client stehen die Original-Klassen der entfernten Methoden nicht zur Verfügung.

Lediglich die Interface-Klassen, die mit einem mitgelieferten IDL-Compiler erzeugt werden, müssen auf dem Client-Rechner vorhanden sein. Der IDL-Compiler erzeugt aus „.idl“-Dateien Java-Klassen, die die Methoden der Server-Klassen simulieren. Die Return-Values dieser Methoden sind die Return-Values eines Referenz-Objekts auf einem entfernten Server.

Gemäß Vorgabe werden alle Kanalmitglieder zwischen 2 Uhrensynchronisationsklassen A und B unterschieden. Die Stationen der Klasse A haben die Uhren, die als hinreichend genau gelten, und die der Typ B müssen entsprechend mit der Klasse A synchronisiert werden.

## Komponentenübersicht

* Dieses Software-Paket beinhaltet folgende Komponenten:
* - Ein IDL-Compiler („idl\_compiler.jar“)
* - Einen Nameservice („name\_service.jar“)
* - Die Middleware („mware\_lib.jar”)
* - Einen Beispiel-Server mit einer Application „Calculator“ („math\_ops.jar“)
* - Einen Beispiel-Client, der einen entfernten Methoden-Aufruf mit dem Calculator-Objekt macht („accessor\_one.jar“)
* Zum Starten:

1. NameService starten:

* (bei Bedarf den Port vorher in "name\_service.config" ändern -
* bei zur Laufzeit belegtem Port wird automatisch ein freier Port gesucht
* und in der .config-Datei gespeichert)
* **java -jar name\_service.jar**

1. vom NameService geänderte Datei "**name\_service.config**" auf den/die AppServer und den/die Clients **kopieren** (in das gleiche Verzeichnis, in dem sich die jar befindet)
2. AppServer starten:

**java -jar math\_ops.jar**

1. Client starten:

**java -jar accessor\_one.jar**

Alle 3 Anwendungen können auf unterschiedlichen Computern laufen.

Auf den AppServern und Clients muss die Datei mware\_lib.jar vorhanden sein.

Mit der Datei "idl\_compiler.jar" werden die Interfaces für die in "idl.files" angegebenen idl-Dateien erzeugt.

## Phasenübersicht

**Nameservice**

Der Nameservice bietet einen Socket an, an dem er die Anfragen „rebind“ und „solve“ entgegennimmt.

Mit jeder Anfrage wird ein neuer Thread (Connection-Handler) gestartet, der die eingehen Anfrage bearbeitet.

Ist die Anfrage abgearbeitet oder eine vordefinierte Zeit (Timeout) vergangen sein, beendet sich dieser Thread.

Bei einer „rebind“-Anfrage speichert der Nameservice sich die Kombination aus Namen und Server, wobei ein Server durch die Kombination aus IP-Adresse und Port definiert ist. Den Port, über den das Objekt erreichbar ist, übermittelt der Server in der Nachricht.

War eine Bindung unter dem angegebenen Namen schon gespeichert, wird die alte Referenz durch die neue ersetzt (überschrieben).

Der Nameservice setzt voraus, dass der AppServer einen Socket auf dem angegebenen Port bereitstellt und das Objekt selbst wieder über den Namen auflösen kann, da ansonsten das Referenzobjekt nutzlos ist.

Erhält der Nameservice eine „solve“-Anfrage, antwortet er mit der Kombination aus IP und Port des AppServers.

**Middleware**

Die Middleware beinhaltet eine andere Implementierung des Nameservice. Diese Instanz vermittelt die Anfragen zwischen dem Frontend der Middleware und dem entfernten Nameservice.

Außerdem erstellt dieser Nameservice für jeden „rebind“-Befehl einen neuen Thread, der Methoden-Aufrufe für das gebundene Objekt abarbeitet.

