# Verteilte Systeme

**Entwurf zur Aufgabe 4**

Florian Weiß, Ditmar Lange

15. Januar 2019

# Inhaltsverzeichnis

# 1 Organisatorisches

**Team:** Ditmar Lange, Florian Weiß

## 1.1 Aufgabenaufteilung

- Compiler: Florian Weiß
- Restliche Programmierung: Beide

## 1.2 Bearbeitungszeitraum

Ditmar Lange:

- 26.12.18 Entwurf: 2 Std.
- 27.12.18 Programmieren: 2 Std.
- 08.01.19 Programmieren: 12 Std.
- 10.01.19 Programmieren: 7 Std.
- 11.01.19 Programmieren: 5 Std.
- 11.01.19 Entwurf: 2 Std.
- 12.01.19 Entwurf: 2 Std.
- 12.01.19 Programmieren: 3 Std.
- 13.01.19 Programmieren: 7 Std.
- 14.01.19 Programmieren: 6 Std.
- 15.01.19 Programmieren: 3 Std.
- 15.01.19 Entwurf überarbeiten

Florian Weiß:

- 26.12.18 Entwurf: 2 Std.
- 27.12.18 Programmieren: 2 Std.
- 30.12.18 Programmieren: 3 Std.
- 08.01.19 Programmieren: 12 Std.
- 10.01.19 Programmieren: 6 Std.
- 11.01.19 Programmieren: 5 Std.

- 12.01.19 Programmieren: 2 Std.
- 13.01.19 Programmieren: 12 Std.
- 14.01.19 Programmieren: 4 Std.
- 15.01.19 Programmieren: 3 Std.

## 1.3 Aktueller Stand

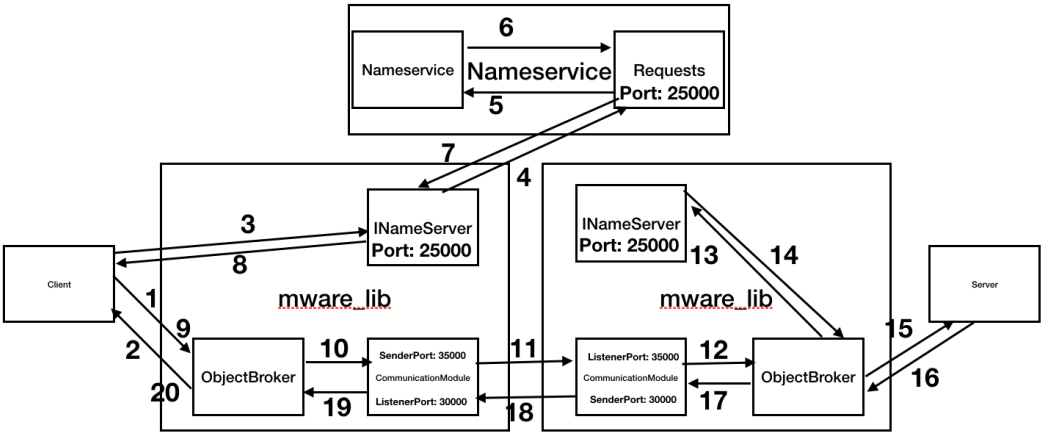
- Alles funktioniert, soweit wir es am Testsystem erkennen können, einwandfrei.

## 1.4 Änderung des Entwurfs

- 10.01.19: Konkreter geworden bei einigen Methoden.

# 2 Struktur, Vorgaben und Ablauf

Abbildung 1: Aufbau des Systems mit beispielhaften Ports und Ablauf




## 2.1 Grober Ablauf der Kommunikation

1. Der Client holt sich vom ObjectBroker den lokalen Namensdienst der Middleware.
2. Der ObjectBroker liefert dem Client den lokalen Namensdienst.
3. Der Client sendet ein resolve(name) an den lokalen Namensdienst, um die Referenz des Objektes zu bekommen.
4. Der lokale Namensdienst leitet die Anfrage an den RequestHandler-Thread des globalen Namendienstes.
5. Der requestHandler-Thread sendet das resolve(name) an den globalen Namensdienst.
6. Dieser sendet ihm die Referenz, sofern sie in seiner Tupel-Liste vorhanden ist.
7. Der requestHandler-Thread antwortet nun dem lokalen Namensdienst.
8. Dieser sendet die Referenz an den Client.
9. Nachdem das Stellvertreterobjekt mittels narrowCast erzeugt wurde, sendet der Client einen remoteCall an den ObjectBroker, mittels dem eine Methode aufgerufen werden soll.
10. Der ObjectBroker kontaktiert hierfür das CommunicationModule.

11. Der Sender dieses CommunicationModules kontaktiert den Listener des CommunicationModules der Adresse, an der das Objekt gelagert ist.
12. Hier wird dann ein localCall an den dortigen ObjectBroker gesendet.
13. Dieser sendet dem lokalen Namensdienst den Befehl, die Referenz des gespeicherten Objekts zurückzugeben.
14. Dies wird dann auch getan, sofern das Objekt da tatsächlich gespeichert ist.
15. Der ObjectBroker ruft daraufhin die Methode beim Objekt auf.
16. Und bekommt das Ergebnis.
17. Dieses sendet er an das CommunicationModule.
18. Dieses sendet mit dem Sender das Ergebnis an den Listener des CommunicationModules des Clients.
19. Das Ergebnis wird dann dem ObjectBroker des Clients übermittelt.
20. Zuletzt wird das Ergebnis dann dem Client übermittelt.

## 2.2 Nameservice

- Der Namservice hat eine Tupel-Liste, die aus den Namen und Referenzen sowie der Hostadresse und dem Port der angemeldeten Objekte besteht.
- Er hört auf einen bestimmten Port, der als Parameter bei der Initialisierung angegeben wird.
- Sobald eine Verbindung mit dem Nameservice aufgebaut wird, startet der Nameservice einen neuen Thread (request) und übergibt ihm die TupelListe und die angekommene Anfrage.
- Der Nameservice darf nicht in JAVA implementiert werden, also wird er in Erlang implementiert.

## 2.3 ObjectBroker

- Der ObjectBroker dient als Schnittstelle zwischen dem Client/Server und der Middleware.
- Es kann nur ein ObjectBroker initialisiert werden (Singleton)
- Der ObjectBroker muss in JAVA implementiert werden

## 2.4 INameService

- Der INameService ist die Klasse innerhalb der mware_lib, die die Kommunikation mit dem Nameservice übernimmt. Anfragen wie rebind und resolve werden an den INameService gestellt, von dem dann an den Nameservice, und die Antworten des Nameservice werden vom INameService dann auch zurückgegeben.
- Beim rebind werden die konkreten Objekte auch lokal gespeichert, und bei resolveLocally wird das konkrete Objekt zurückgegeben.
- Der INameService muss in JAVA implementiert werden.
- Der INameService ist eine Subklasse des Interfaces NameService.

## 2.5 CommunicationModule

- Das CommunicationModule dient der Kommunikation zwischen verschiedenen Middlewares.
- Es kann entfernte Methodenaufrufe versenden, aber auch welche empfangen, und dessen Durchführung in der eigenen Middleware beauftragen.
- Das CommunicationModule braucht zwei separate Threads, ähnlich wie beim Nameservice, Einer (Listener) muss auf einen bestimmten Port lauschen um Anfragen annehmen zu können und der andere (Sender) muss Aufrufe auf andere Ports zu anderen CommunicationModule-Listeners machen.

## 2.6 Sender

- Der Sender öffnet ein Socket zum Port des empfangenden CommunicationModules.
- Dann wird die Nachricht gesendet, und die daraus resultierende Antwort zurückgegeben.

## 2.7 Listener

- Der Listener hört auf einen vom CommunicationModule zufällig gewählten Port.
- Alle ankommenden Nachrichten wandelt der Listener in ein passendes Format um, und sendet es dann dem CommunicationModule zur Bearbeitung.

## 2.8 Compiler

- Der Compiler soll die vorgegebenen IDL Dateien in JAVA-Klassen konvertieren.
- Die neuen JAVA-Klassen bekommen den Namen "_<name>ImplBase"
- Die JAVA-Klassen bekommen ausserdem eine narrowCast() Methode, die zu einer von resolve() gelieferten Objektreferenz eine klassenspezifische Referenz liefert.

# 3 Methodendokumentation

## 3.1 Nameservice

**init()** Initialisiert den Nameservice, der dann permanent auf Nachrichten vom request-Handler wartet.

## 3.2 requestHandler

**init(Port)** Lauscht auf dem angegebenen Port, initialisiert den Nameservice, und behandelt mittels loop und handleRequest die beim Nameservice ankommenden Anfragen.

**loop(Socket, NameservicePID)** Hört auf dem Socket und startet für jede ankommende Nachricht einen neuen handleRequest-Thread.

**handleRequest(Socket, NameservicePID** Bearbeitet die angekommende Nachricht.

- Die angekommene Nachricht wird anhand der Semikolons in 4 Teile getrennt: Befehl, Objektname, Hostadresse, Port.
- Bei rebind wird dem Nameservice der Befehl {rebind, self(), {Name, Host, Port}} gesendet. Der Nameserver fügt daraufhin den Namen und seine Adresse in die Tupelliste ein und sendet ein ok zurück. Dieses ok wird dann an den INameServer weitergeleitet. Dann ruft sich handleRequest selber wieder auf bis der INameService die Verbindung schließt.
- Bei resolve wird dem Nameservice der Befehl {resolve, self(), Name} gesendet. Der Nameservice findet dann die Referenz mittels keyfind in der TupleListe und gibt diese zurück. Die wird dann an den INameService gesendet. Wurde das Objekt nicht gefunden, wird ein null gesendet. Dann ruft sich handleRequest selber wieder auf bis der INameService die Verbindung schließt.
- Fehlernachrichten werden geloggt.

## 3.3 INameService

**rebind(Object servant, String name)**

1. Holt sich Host und Port vom CommunicationModule, den er über den ObjectBroker erreicht.
2. Schickt eine Nachricht mit folgenden Format an den Nameservice auf den bei der Initialisierung angegebenen Port: {rebind;<name>;<host>;<port>}
3. Er empfängt daraufhin ein ok und speichert dann außerdem das Objekt in einer HashMap.

**resolve(String name)** Sendet dem Nameservice den Namen mit einer Nachricht folgendes Formats:
{resolve;<name>; ; }
Er bekommt dann die Hostadresse und dem Port des INameService, auf dem sich die konkrete Referenz des Objektes befindet und gibt diese zurück. Sollte der Nameservice die Referenz nicht gefunden haben, so wird ein null zurückgegeben.

**resolveLocally(String name)** Gibt das Objekt zurück, sofern es auf diesem INameService gespeichert ist. Ansonsten wird null zurückgegeben.

## 3.4 ObjectBroker

**init(String serviceHost, int port, boolean debug)**

1. Initialisisert den ObjectBroker und gibt ihn als Objekt zurück. Es wird sichergestellt, dass nur ein ObjectBroker erstellt werden kann (Singleton).
2. Die Variable debug entscheidet darüber, ob geloggt werden soll.
3. Der INameService wird mit host, port, debug, und der Referenz auf den ObjectBroker initialisiert.
4. Das CommunicationModule wird initialisiert mit host, INameService, debug und der Referenz auf den ObjectBroker als Parametern.

**getCom()** Gibt das im Konstruktor erstellte CommunicationModule zurück.

**getNameService()** Gibt den im Konstruktor erstellten INameService zurück.

**shutdown()** Beendet sich.

**remoteCall(String name, String host, int port, String methodName, Object... args)** Führt ein invoke(name, host, port, methodName, args) auf dem CommunicationModule aus, und gibt dann dessen Ergebnis zurück.

**localCall(String name, String methodName, Object... args**

- Holt sich vom INameService das lokal gelagerte Objekt name.
- Stellt die Klassentypen der args fest.
- Dann wird mittels invoke() (von Java) die Methode ausgeführt.
- Das Ergebnis der Methode wird zurückgegeben (möglicherweise eine RuntimeException).

## 3.5 CommunicationModule

**Konstruktor(String host, INameService nameService, boolean debug, ObjectBroker ob)**

1. Erstellt einen serverSocket, der einen zufälligen Port zugewiesen bekommt.
2. Startet den Listener-Thread mit einer Referenz auf dieses CommunicationModule, dem Socket, und dessen Host und Port.

**getHost()** Gibt den Host zurück

**getPort()** Gibt den Port zurück

**invoke(String name, String host, int port, String method, Object... args)** Startet einen neuen Sender mit den selben Parametern. Führt auf diesem Sender invoke aus und gibt dessen Ergebnis dann zurück.

**remoteCall(String objectName, String methodName, Object... arg)** Sendet dem ObjectBroker einen localCall(objectName, methodName, args) Befehl, und erwartet dann das Resultat, welches dann auch sofort zurückgegeben wird.

## 3.6 Sender

**invoke()**

1. Öffnet einen Socket mit dem Host und Port des anderen Listeners.
2. Sendet die Nachricht um beim anderen CommunicationModule remoteCall aufzurufen.
3. Das Ergebnis (ein String) wird darauf überprüft, ob es ein String, int, oder double ist, und dementsprechend in ein solches umgewandelt und dann zurückgegeben.

## 3.7 Listener

**run()**

- Wartet auf eine ankommende Verbindung.
- Fängt die eintreffende Nachricht ab und splittet sie an den Semikolons (;).
- Ist der erste Substring "remotecall" und hat die Liste 5 oder 6 Substrings, so werden dann die Parameter, so fern es welche gibt (6. Substring) in ihre Klassen zurückgewandelt. Parameter sind durch Doppelpunkte (:) getrennt.
- Daraufhin wird remoteCall auf dem CommunicationModule ausgeführt.
- Das Ergebnis wird zurück an den Sender geschickt, der die Anfrage an den Listener gesendet hatte.

## 3.8 Compiler

**main(String args[])** Liest die IDL-Datei zeilenweise ein und schreibt die äquivalenten JAVA-Zeilen in die neu erstellte JAVA-Datei. Erstellt außerdem eine narrowCast()-Methode, die sicherstellt, dass das Stellvertreterobjekt das gleiche Verhalten wie die richtige Implementation der Klasse aufweist.

**Funktionsweise der narrowCast(Object ref)-Methode** narrowCast(ref) wird so geschrieben, dass sie einen String als Parameter übergeben bekommt, und diesen an den Kommas splittet nachdem alle Anführungszeichen (") entfernt wurden. Daraus entstehen Name, Host und Port. Mit diesen Informationen wird ein neues Objekt angelegt, welches für die Methoden das invoke im CommunicationModule aufruft.
Innerhalb des returns vom narrowCast werden die Methoden erzeugt in denen mittels invoke das CommunicationModule aufgerufen und folglich das eigentliche Objekt angesprochen wird.

# 4 Request-/Reply-Protokoll

## 4.1 requestHandler des Nameservice

**"<Befehl>;<Name>;<Hostadresse>;<Port>"** Befehl ist entweder

- rebind
- resolve
- andere

Bei rebind wird ein ok zurückgegeben

Bei resolve wird entweder die Hostadresse und der Port des zuständigen INameService zurückgegeben, oder ein null, sollte dieser Name nicht bekannt sein.

Bei anderen wird nichts zurückgegeben.

## 4.2 Nameservice

**{rebind, RH_PID, {Name, Host, Po}}** .
Bei Erfolg: Ein String "ok" wird an die RH_PID zurückgegeben.
Bei Fehler: Keine Rückgabe

**{resolve, RH_PID, Name}** .
Bei Erfolg: Die Adresse von Name wird als String zurückgegeben. Host und Port sind durch ein Komma im String getrennt.
Bei Fehler: Ein null wird zurückgegeben

## 4.3 CommunicationModule (Listener)

**"remotecall;<host>;<port>;<objektname>;<methodenname>;<parameter geteilt durch :>"** .
Bei Erfolg: Wird vom Sender das Resultat zurückgesendet.
Bei Fehler: Wird die RuntimeException zurückgesendet.

# 5 Test-Dokumentation

## 5.1 Testkonzept

Die mware_lib und der Nameservice werden mit den Server und Client Dateien von Prof. Klauck getestet.

## Literatur

[1] https://de.wikibooks.org/wiki/Java_Standard:_Socket_und_ServerSocket_(java.net)

[2] VS-Vorlesungsfolien

[3] https://stackoverflow.com/questions/160970/how-do-i-invoke-a-java-method-when-given-the-method-name-as-a-string

[4] learnyousomeerlang.com

[5] erlang.org

[6] javabeginners.com