HAW HAMBURG

Referat

Andre Brand- 2286926

Middleware: Ein Blick hinter die Kulissen

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Andre Brand

Middleware: Ein Blick hinter die Kulissen

Referat eingereicht im Rahmen der Veranstaltung Verteilte Systeme
im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Christoph Klauck

Eingereicht am: 09.01.2019

Inhaltsverzeichnis

1 Einleitung **1**

2 Entwurf **2**

2.1 Protokoll . . . 3

2.1.1 Nameservice . . . 3

2.1.2 Middleware . . . 4

2.1.3 Middleware . . . 5

2.2 Nameservice . . . 5

2.2.1 Rebind . . . 6

2.2.2 Resolve . . . 6

2.3 Middleware . . . 7

2.3.1 Object Broker . . . 8

2.3.2 Dispatcher . . . 8

2.3.3 Socket Controller . . . 8

2.3.4 Nameservice Interface . . . 9

2.3.5 Object Warehouse . . . 11

2.3.6 MWareObject . . . 13

2.4 IDL-Compiler . . . 16

2.4.1 Abstrakte Klasse . . . 16

2.4.2 Ausführbare Klasse . . . 17

3 Lösungsweg **18**

3.1 Nameservice . . . 18

3.1.1 Ablauf innerhalb des Nameservice . . . 18

3.1.2 Ablauf rebind . . . 19

3.1.3 Ablauf resolve . . . 19

3.2 Middleware . . . 20

3.2.1 Object Broker . . . 20

3.2.2 ObjectWarehouse . . . 21

3.2.3 MWareObject 21
3.2.4 LocalObject 21
3.2.5 RemoteObject 22
3.3 IDL-Compiler 22
3.3.1 Ablauf 22

4 Fazit **24**

Selbstständigkeitserklärung **25**

1 Einleitung

In diesem Referat geht es um die Erstellung einer Middleware, die dafür sorgt, dass ein entferntes Objekt wie ein lokales angesprochen werden kann.
Dazu müssen insgesamt drei Teilprogramme erstellt werden. Zum einen die Middleware an sich, einen Namensdienst, der die entfernten Objekte verwaltet und einen Compiler, der idl-Dateien in ein Interface übersetzt.
Bis auf den Namensdienst müssen alle Anwendungen in Java geschrieben werden. Der Namensdienst darf nicht in Java geschrieben werden.

2 Entwurf

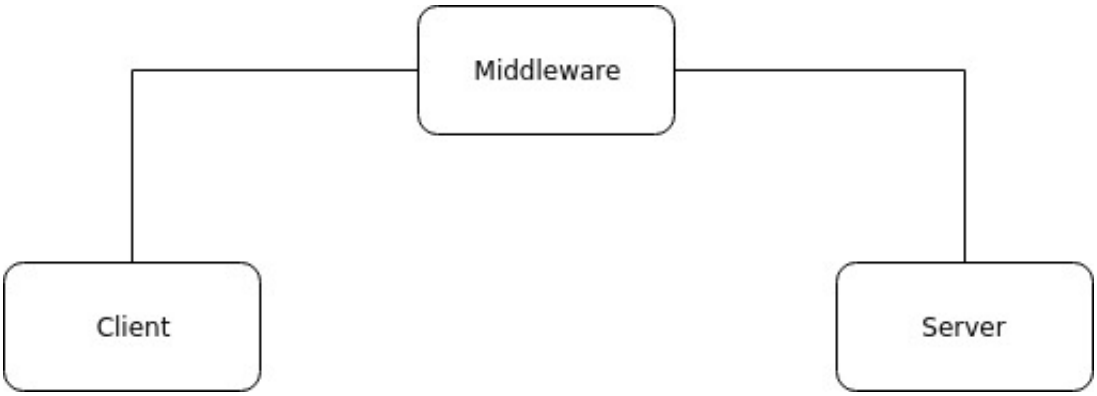

Abbildung 1: Komponentendiagramm

Die Middleware soll dafür sorgen, dass ein entferntes Objekt wie ein lokales behandelt werden kann, ohne, dass der Anwender dies merkt. Dazu muss mit Hilfe eines Compilers und einer idl-Datei ein Stub-Objekt erzeugt werden.
Dieses Stub-Objekt kommuniziert über den Nameservice mit dem ObjectBroker des Servers, auf dem das richtige Objekt liegt. Die Antworten des Objektes werden direkt an ein RemoteObject, dass im Stub-Objekt die Kommunikation übernimmt, gesendet und vom Stub-Objekt in den richtigen Datentypen geparst, damit das Ergebnis direkt vom Client genutzt werden kann.
Der ObjectBroker vom Server, der im Nameservice hinterlegt ist, nimmt die Anfrage des Clients an und startet einen Dispatcher. Der Dispatcher leitet die Anfrage des Clients an das passende LocalObject im MWareObjectWarehouse weiter, welches diese Anfrage parst und mittels Reflection im Objekt aufruft. Das Ergebnis dieses Aufrufs wird als String zurückgegeben und vom Dispatcher an den Client zurückgesendet.

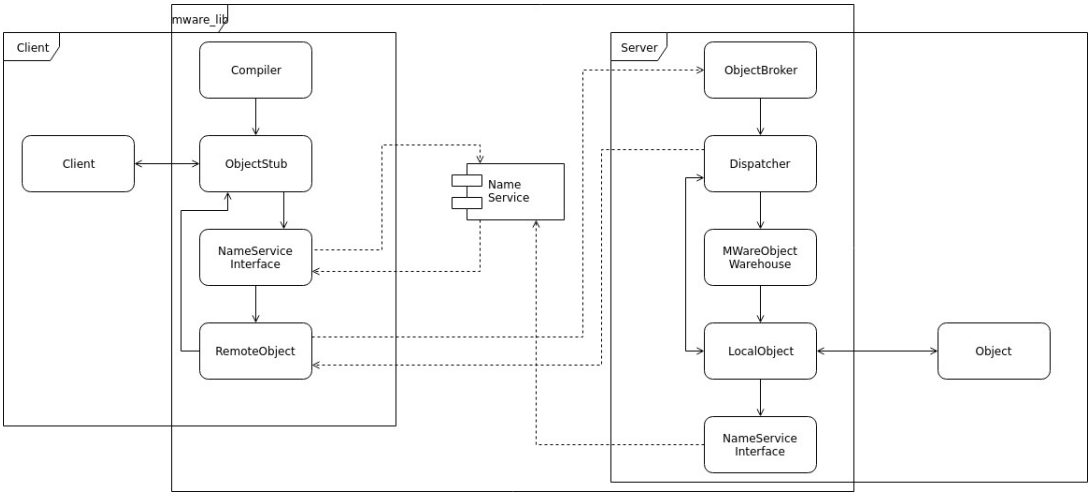

Abbildung 2: Bausteindiagram Middleware

2.1 Protokoll

Die Kommunikation der Middleware läuft über TCP-Sockets, daher muss ein eigenes Protokoll erstellt werden. Im folgenden wird das Protokoll beschreiben.

2.1.1 Nameservice

Rebind

Rebind bindet ein Objekt über die Hostadresse und den Port des Servers, auf dem das Objekt liegt, an den Nameservice

- **Aufruf:** rebind!<Name>:<Hostadresse>:<Port>
- **Antwort:** ok

Resolve

Resolve fragt ein Objekt vom Nameservice ab. Zurückgegeben werden die Hostadresse und der Port des Servers, auf dem das angefragte Objekt liegt.

- **Aufruf:** resolve!<Name>
- **Antwort:** ok:<Name>:<Hostadresse>:<Port>

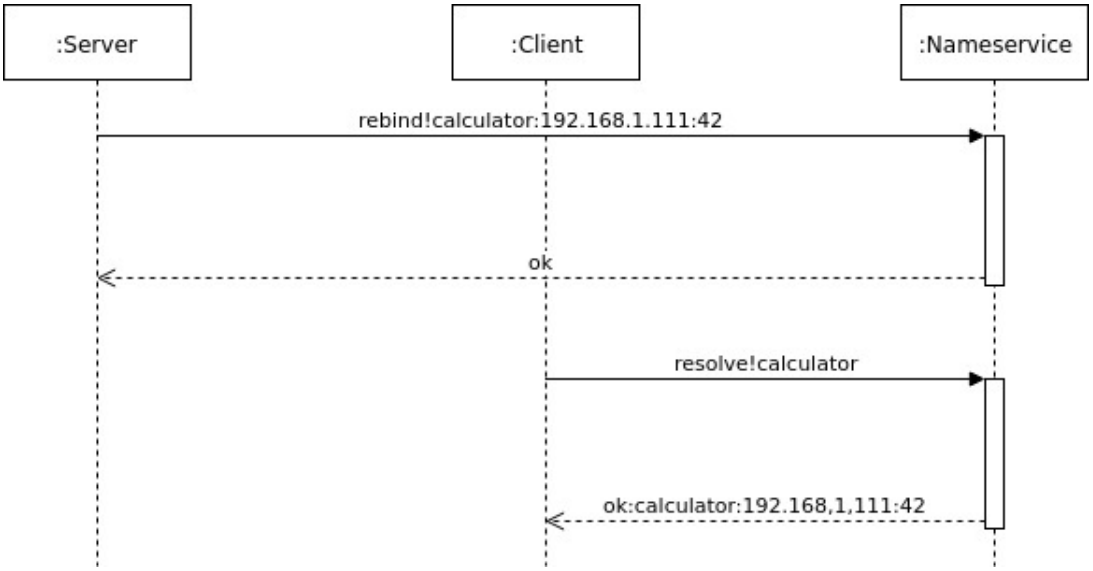

Abbildung 3: Flowchart Nameservice

2.1.2 Middleware

In der Middleware kommunizieren die RemoteObjekte mit den lokalen Objekten mit Hilfe des folgenden Protokolls:

- **Methodenaufruf:** <Objektname>!<Funktionsnname>![<Parameter>:<Datentyp>]*
- **Antwort:** result:<Ergebnis>

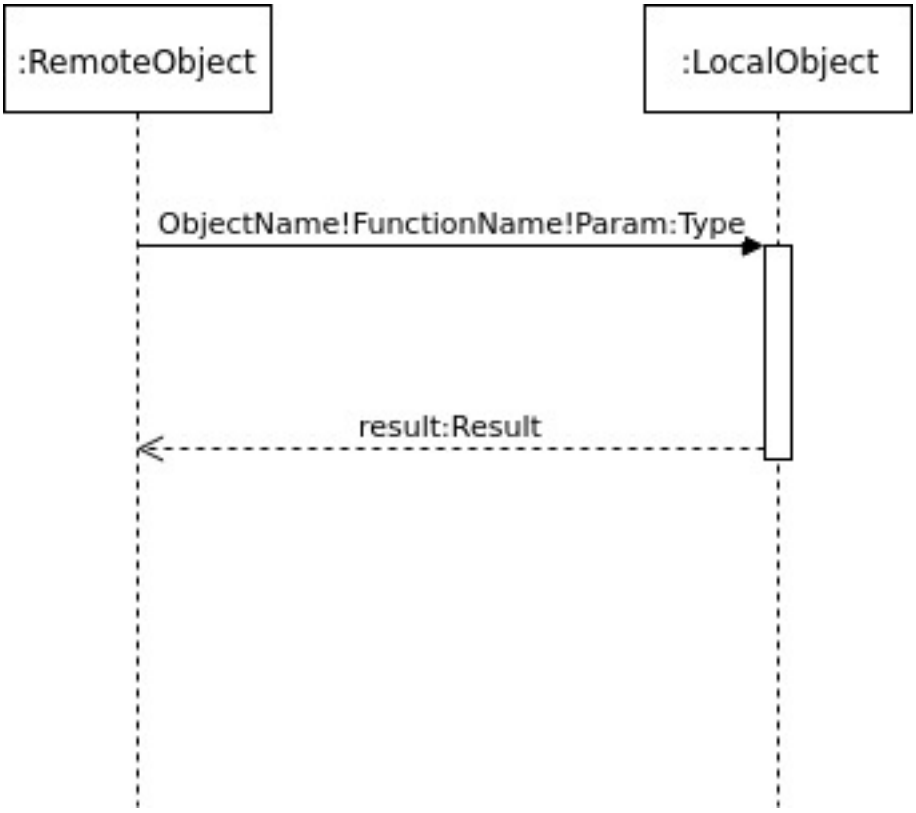

Abbildung 4: Flowchart Objektkommunikation

2.1.3 Middleware

Aufruf einer Methode

- **Aufruf:** resolve!<Name>
- **Antwort:** ok:<Name>:<Hostadresse>:<Port>

2.2 Nameservice

Der Nameservice bildet ab, welches Objekt sich auf welchem Server befindet. Mit dem Namen des Objekts kann der Host und der Port des Servers, auf welchen das Objekt liegt, abgefragt und eine Verbindung dahin aufgebaut werden. Der Nameservice ist als

entferntes Objekt zu implementieren, sodass die Nachrichtenübermittlung mittels TCP realisiert wird.

2.2.1 Rebind

Beschreibung:
Bindet ein Objekt mit dem übergebenen Namen im Nameservice.
Die zugehörigen Informationen sind Hostadresse und Port des Servers, auf dem das Objekt liegt. Bereits vorhandene Objekte mit dem selben Namen werden überschrieben.

Anfrage: rebind!<Name>:<Hostadresse>:<Port>

Antwort: ok

Beispiel:

Server:	rebind!calculator:192.168.1.111:42
Nameservice:	ok

Ablauf

1. Auslesen von Name, Hostadresse und Port
2. Speichern der Hostadresse und des Ports unter angegebenen Namen
3. Rückgabe von ok

2.2.2 Resolve

Beschreibung:
Fragt die zugehörigen Informationen zum Objekt mit dem übergebenen Namen ab.
Als Antwort werden Hostadresse und Port, die für den übergebenen Namen gespeichert wurden, zurückgegeben.

Anfrage: resolve!<Name>

Antwort: ok:<Name>:<Hostadresse>:<Port>

Beispiel:

Client:	resolve!calculator
Nameservice:	ok:calculator:192.168.1.111:42

Ablauf

1. Auslesen vom Namen
2. Lesen der Hostadresse und des Ports unter angegebenen Namen
3. Rückgabe von ok:<Name>:<Hostadresse>:<Port>

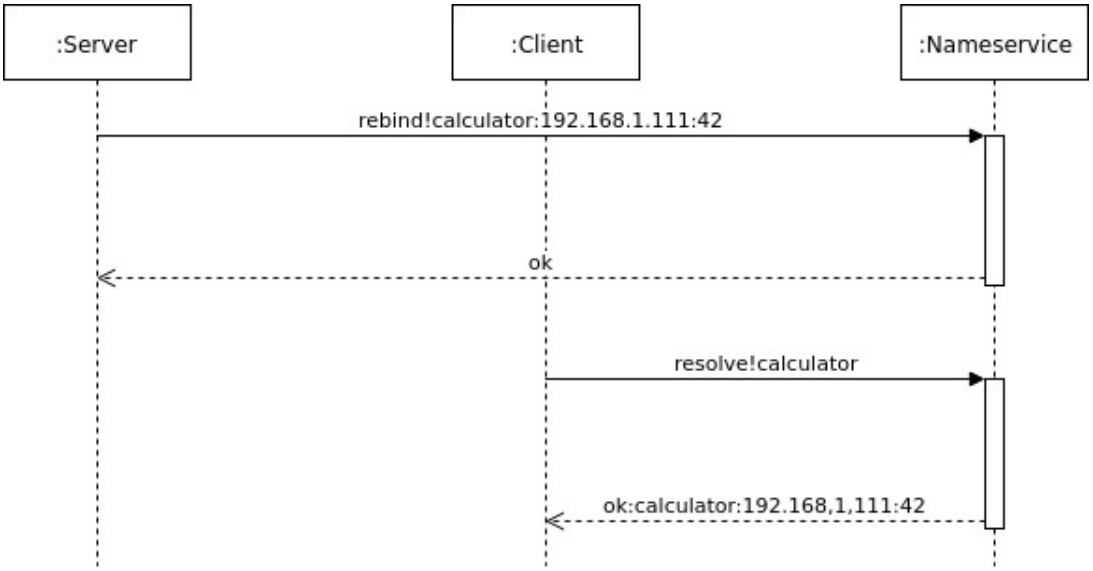

Abbildung 5: Flowchart Nameservice

2.3 Middleware

Aufgabe der Middleware ist es, den Zugriff auf entfernte Objekte so zu ermöglichen, dass die benutzenden Objekte nicht merken, dass es sich um ein entferntes Objekt handelt. Im folgenden werde ich alle Module, die die Middleware beinhaltet, vorstellen.

2.3.1 Object Broker

Das "Frontend"der Middleware.
Wird clientseitig von dem Stub-Objekt, das vom IDL-Compiler erzeugt wurde, angesprochen. Dieser clientseitige Object Broker sendet eine Anfrage mit dem Namen des Stub Objektes an den Nameservice. Dieser liefert, sofern bekannt, die Hostadresse und den Port des Servers, auf dem das angefragte Objekt liegt, zurück.
Bekommt der anfragende ObjectBroker eine Hostadresse und einen Port zurück, stellt er dort eine Anfrage zur Ausführung der gewünschten Funktion. Der serverseitige Object Broker nimmt diese Anfrage entgegen und startet einen Dispatcher mit dem Socket des Clients.

2.3.2 Dispatcher

Der Dispatcher verarbeitet die Anfrage des Clients und ruft die angeforderte Methode des Objektes mit den übergebenen Parametern aus und leitet das Ergebnis oder auftretende Exceptions an den Client weiter. Danach wird die Verbindung geschlossen.

Ablauf

1. Empfangen der Nachricht vom Client
2. Auslesen des Namens vom Objektes, das angesprochen werden soll.
3. Anfrage an das Object Warehouse nach dem Objekt mit dem gegebenen Namen
4. Aufruf der angegebenen Methode
5. Rückgabe des Ergebnisses an den Client als Nachricht

2.3.3 Socket Controller

Ist für die eigentliche Kommunikation verantwortlich. Sendet und empfängt Nachrichten über einen übergebenen Socket.

Send Message

Beschreibung:
Sendet eine Nachricht an das Objekt auf der änderen Seite"des Sockets.

Aufruf: sendMessage(String message)

Variablen

1. **message:** Nachricht, die an den Kommunikationspartner gesendet werden soll

Ablauf

1. Senden der übergebenen Nachricht an den Kommunikationspartner

Receive Message

Beschreibung:
Empfängt eine Nachricht vom Kommunikationspartner.

Aufruf: receiveMessage()

Ablauf

1. Empfangen einer Nachricht an den Kommunikationspartner
2. Rückgabe der empfangenen Nachricht

2.3.4 Nameservice Interface

Interface, dass den Nameservice anspricht.

Rebind

Beschreibung:
Spricht die Funktion Rebind des Nameservice an. Das Objekt wird mit Hilfe eines MWare-Objektes im Object Warehouse gespeichert und die IP-Adresse und der Port des Servers mit dem Namen des Objektes beim Nameservice registriert.

Aufruf: rebind(Object servant, String name)

Variablen

- **servant:** Objekt, dass beim Nameservice registriert wird
- **name:** Name des Objektes, mit dem es beim Nameservice registriert wird

Ablauf

1. Erzeugen eines MWare-Objektes mit dem übergebenen Objekt
2. Einfügen des MWare-Objektes in das MWare Warehouse
3. Rebind-Anfrage an den Nameservice
    - **Anfrage:** rebind!<Name>:<Hostadresse>:<Port>
        - Name: Name des zu registrierenden Objektes
        - Hostadresse: Eigene IP-Adresse, auf der der Server horcht
        - Port: Port, auf dem der Server horcht
    - **Antwort:** ok

Resolve

Beschreibung:
Spricht die Funktion Resolve des Nameservice an. Die IP-Adresse und der Port des Servers, auf dem das Objekt mit dem übergebenen Namen liegt, wird zurückgegeben.

Aufruf: resolve(String name)

Variablen

- **name:** Name des Objektes, dass beim Nameservice abgefragt wird

Rückgabe: Objektreferenz des angefragten (entfernten) Objektes

Ablauf

1. Erzeugen eines MWare-Objektes mit dem übergebenen Objekt
2. Einfügen des MWare-Objektes in das MWare Warehouse
3. Rebind-Anfrage an den Nameservice
    - **Anfrage:** rebind!<Name>:<Hostadresse>:<Port>
        - Name: Name des zu registrierenden Objektes
        - Hostadresse: Eigene IP-Adresse, auf der der Server horcht
        - Port: Port, auf dem der Server horcht
    - **Antwort:** ok

2.3.5 Object Warehouse

Verwaltet sowohl lokale aus auch entfernte Objekte für den Zugriff.

Add Object

Beschreibung:
Fügt ein Objekt zum Warehouse hinzu.

Aufruf: addObject(String name, MWareObject object)

Variablen

- **name:** Name des Objektes, dass hinzugefügt werden soll
- **object:** Objektes, dass hinzugefügt werden soll

Rückgabe: Boolean, ob das Objekt hinzugefügt wurde, oder nicht.

Ablauf

1. Prüfen, ob bereits ein Objekt mit dem Namen vorhanden ist
    - Wenn ja: Rückgabe von false
    - Wenn nein: Hinzufügen des Objekts unter angegebenen Namen und Rückgabe von true

Remove Object

Beschreibung:
Entfernt ein Objekt vom Warehouse.

Aufruf: removeObject(String name)

Variablen

- **name:** Name des Objektes, dass entfernt werden soll

Rückgabe: Boolean, ob das Objekt entfernt wurde, oder nicht.

Ablauf

1. Prüfen, ob ein Objekt mit dem Namen vorhanden ist
    - Wenn ja: Entfernen des Objektes und Rückgabe von true
    - Wenn nein: Rückgabe von false

Replace Object

Beschreibung:
Fügt ein Objekt zum Warehouse hinzu.

Aufruf: replaceObject(String name, MWareObject object)

Variablen

- **name:** Name des Objektes, dass hinzugefügt werden soll
- **object:** Objektes, dass hinzugefügt werden soll

Ablauf

1. Prüfen, ob bereits ein Objekt mit dem Namen vorhanden ist
    - Wenn ja: Ersetzen des vorhandenen Objektes mit dem übergebenen
    - Wenn nein: Hinzufügen des Objekts unter angegebenen Namen

Get Object

Beschreibung:
Liefert das Objekt mit dem übergebenen Namen zurück, sofern vorhanden.

Aufruf: getObject(String name)

Variablen

- **name:** Name des Objektes, dass hinzugefügt werden soll

Rückgabe: Objekt mit dem übergebenen Namen, wenn vorhanden.

Ablauf

1. Prüfen, ob bereits ein Objekt mit dem Namen vorhanden ist
    - Wenn ja: Rückgabe des Objekts
    - Wenn nein: Rückgabe von null

2.3.6 MWareObject

Kann sowohl ein lokales, als auch ein entferntes Objekt repräsentieren und ruft die Methoden des Objektes auf.

Call Method

Beschreibung:
Ruft die in der übergebenen Nachricht angegebene Methode mit den angegebenen Parametern auf.

Aufruf: callMethod(String message)

Variablen

- **message:** String mit allen Informationen zur aufgerufenen Methode
  - **Format:** <Name>!<Methode>![Parameter:Datentyp]*

Rückgabe: Ergebnis des Methodenaufrufs als String.
Format: result:<Ergebnis>.

Ablauf

1. Parsen des übergebenen Strings mittels des oben angegebenen Formats
2. Aufruf der angegebenen Funktion mit den übergebenen Parametern
3. Rückgabe des Ergebnisses im Format: result:<Ergebnis>

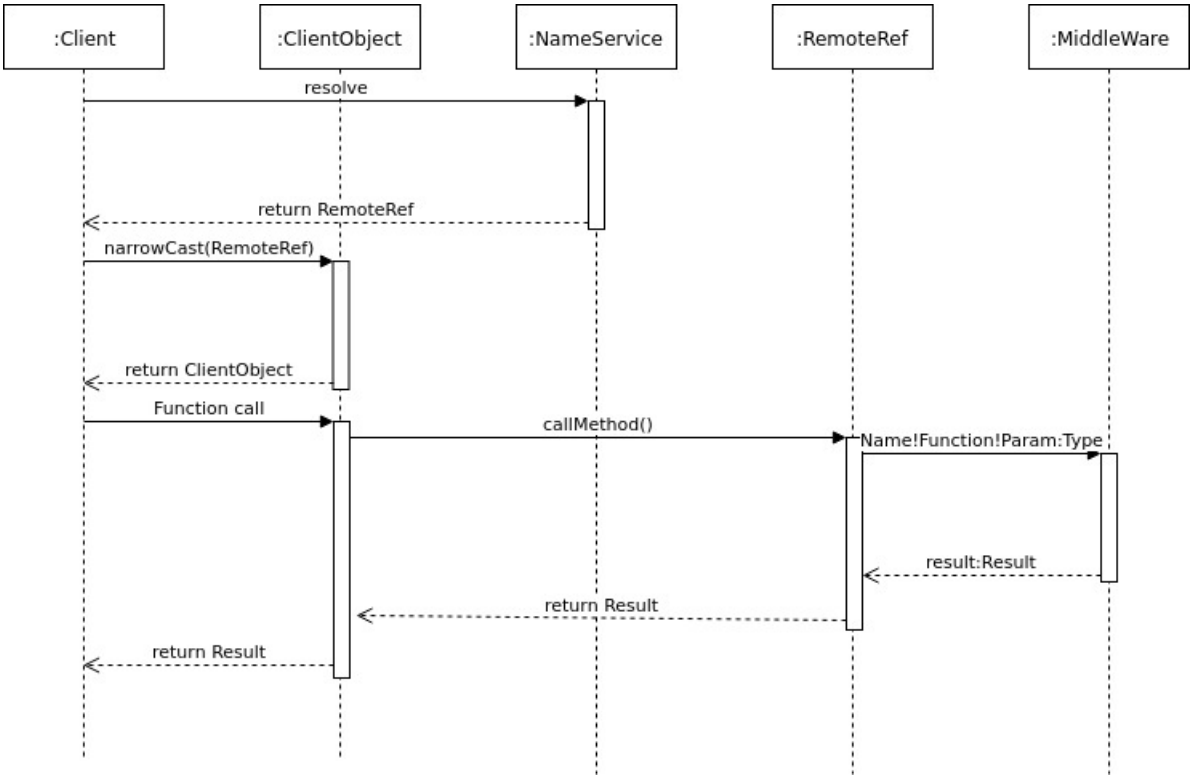

Abbildung 4: Flowchart Client

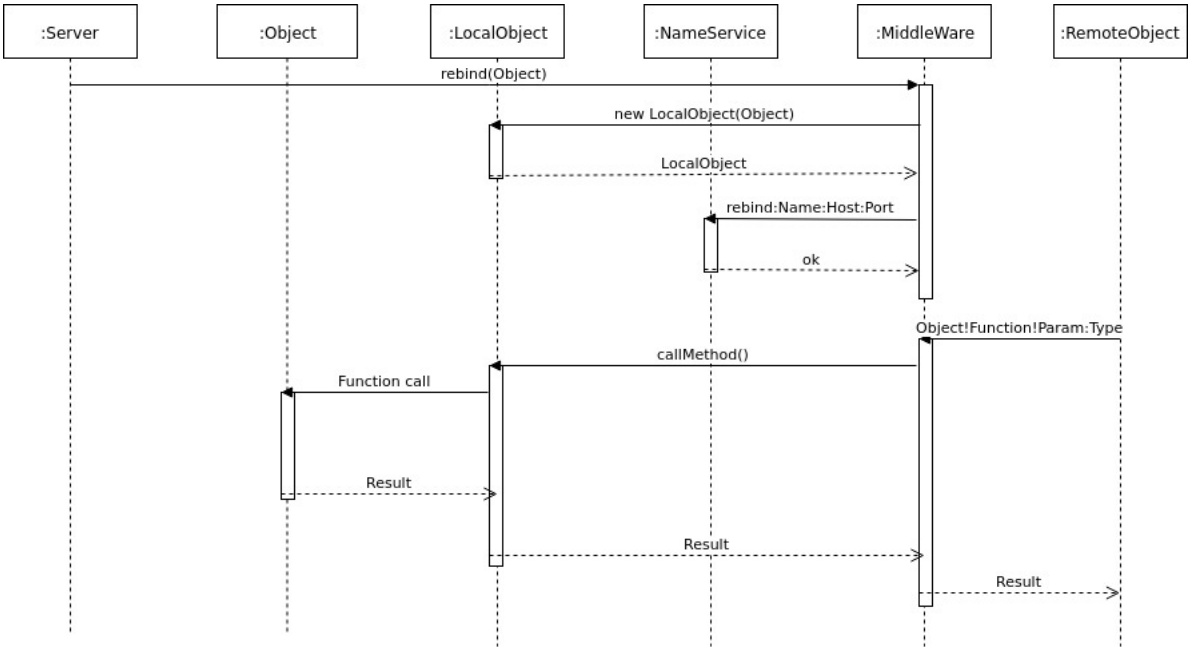

Abbildung 4: Flowchart Server

2.4 IDL-Compiler

Der IDL-Compiler muss die IDL-Datei einlesen und daraus zwei Java-Dateien erzeugen.

1. _ClassNameImplBase.java - Abstrakte Klasse, die die Methoden der Klassen aus der IDL-Datei als abstrakte Methoden beinhaltet

   a) Muss Methode narrowCast beinhalten, die ein Objekt zurückliefert, welches mittels des Objektes aus dem Resolve-Aufruf das RemoteObject ansprechen kann

2. ClassName.java - Klasse, die die jeweiligen Methoden des entfernten Objektes aufruft.

Die IDL-Datei ist folgendermaßen aufgebaut:

```
module modulename {     // Name des Moduls
        class ClassA {     // Name der Klasse
                string methodA(int a);     // Methode der Klasse
        };     // Klassenende
        class ClassB {     // Zweite Klasse
                double methodA(double param1, double param2);
                int methodB(int param1);
        };
};     // Modulende
```

Das Modul beschreibt ein Java-Package, dass mehrere Klassen beinhalten kann. Dies muss mittels
package modulename;
in den Java-Klassen vermerkt werden.

2.4.1 Abstrakte Klasse

narrowCast

Beschreibung:
Liefert ein Objekt zurück, dass ein Objekt der erzeugten Klasse zurückliefert.

Aufruf: sendMessage(Object rawObjectReference)

Variablen

1. **rawObjectReference:** Objekt, dass vom Nameservice zurückgeliefert wird (MWareObject)

2.4.2 Ausführbare Klasse

Die ausführbare Klasse muss alle Funktionen inklusive der Implementierung des Methodenaufrufs beinhalten. Siehe hierfür das Kapitel 2.3.6 CallMethod.

3 Lösungsweg

3.1 Nameservice

Der Nameservice durfte gemäß den Vorgaben nicht in Java implementiert werden, daher entschied ich mich für Python. Abgebildet werden die Objekte in einem Dictionary, in dem Key-Value-Paare gespeichert werden können.
Der Schlüssel für das jeweilige Objekt ist der Name des Objektes und der Wert dazu ist wieder ein Dictionary. Dieses Dictionary hat zwei Keys, zum einen mit dem Key "host", die als Wert die IP-Adresse des Servers, auf dem das Objekt liegt, hat und den Key "port", der als Wert die Portnummer hat.

Geht eine Anfrage beim Nameserver an, wird ein Thread gestartet, in dem die eingehende Nachricht ausgewertet wird. Die jeweilige gewünschte Funktion (resolve oder rebind) wird ausgeführt und das Ergebnis zurückgegeben.
Wird ein Objekt beim Nameservice gebunden, werden Hostadresse und Portnummer mittels Protokoll übertragen. Diese beim Namesservice eingegangene Nachricht wird dann über eine Split-Funktion aufgeteilt und die Adresse und der Port zum übergebenen Namen hinterlegt.
Wird ein Objekt beim Nameservice angefragt, werden sofern vorhanden die IP-Adresse und der Port mittels des dafür angefertigten Protokolls zurückgegeben. Ist es nicht vorhanden, wird "nok:Unknown Objectßurückgegeben.

3.1.1 Ablauf innerhalb des Nameservice

- Entgegennahme der Anfrage
- Starten eines Threads mit dem Socket

- Empfangen der Nachricht
- Aufteilen der Nachricht mittels Split-Funktion beim Ausrufezeichen
- Auswerten des ersten Teils der Nachricht (Aufgerufene Funktion)
    1. rebind - Weitergabe des zweiten Nachrichtenteils an die Funktion rebind - Ergebnis = Rückgabewert
    2. resolve - Weitergabe des zweiten Nachrichtenteils an die Funktion resolve - Ergebnis = Rückgabewert
    3. Sonstiges - Ergebnis = "nok:unknown message"
- Senden des Ergebnisses über den Socket
- Schließen des Sockets

3.1.2 Ablauf rebind

- Aufsplitten des zweiten Nachrictenteils mittels Splitfunktion am Doppelpunkt
- Erstellen eines Dictionarys, das Host-Adresse und Port beinhaltet
- EIntragen des Dictionary in das des Nameservice mit dem Namen als Key
- Rückgabe von ök"

3.1.3 Ablauf resolve

- Abfrage des Objektes mit dem übergebenen Namen
    1. Wenn vorhanden, zusammenbau des Rückgabestrings: ök:<Name>:<Hostadresse>:<Port>"
    2. Wenn nicht vorhanden: Rückgabestring = "nok:no object"
- Rückgabe des Rückgabestrings

3.2 Middleware

3.2.1 Object Broker

Der ObjectBroker ist das FrontEnd der Middleware. Hier kann die Referenz zum Nameservice amgefragt werden. In meiner Implementierung kann auch direkt ein Objekt über den ObjectBroker mittels Funktion addObject(Object object) hinzugefügt werden. Diese ruft allerdings auch nur die Funktion rebind des Nameservice auf.
Wird ein ObjectBroker erzeugt, wird automatisch ein Thread gestartet, der auf einem zufällig gewählten Port horcht. Gehen Anfragen ein, wird ein Dispatcher gestartet, der Anfragen entgegen nimmt und die Funktionen der lokalen Objekte mittels Reflection aufruft.

Ablauf ObjectBroker

- Starten eines ServerSockets
- Horchen auf dem Port (hier ein zufällig erstellter)
- Bei eingehenden Anfragen: Erstellen eines neuen Dispatchers und starten des Dispatcher-Threads mittels start()

Ablauf Dispatcher

- Empfangen der Nachricht
- Splitten der Nachricht, um den Namen des angefragten Objektes zu bekommen
- Abfrage des Objektes im ObjectWarehouse
- Aufruf der Funktion callMethod(message) mit der empfangenden Nachricht als Parameter
- Rückgabe des Ergebnisses von callMethod

3.2.2 ObjectWarehouse

Das ObjectWarehouse speichert alle MWareObjekte, damit sowohl lokale als auch entfernte Objekte über die MiddleWare verwendet werden können. Zwar sollten keine lokalen Objekte über die MiddleWare benutzt werden, aber so wird unnötige Netzauslastung verhindert, falls doch ein lokales Objekt angesprochen werden soll.
Das ObjectWarehouse kann vom ObjectBroker, vom Dispatcher und vom Nameservice angesprochen werden.

3.2.3 MWareObject

Das MWareObject ist ein Interface, dass von den Klassen LocalObject und RemoteObject implementiert wird. Es schreibt folgende Methoden vor:

1. String getName(); - Liefert den Namen des Objekts zurück
2. String callMethod(String message); - Führt die Funktion aus, die laut der übergebenen Nachricht aufgerufen werden soll

3.2.4 LocalObject

Die Klasse LocalObject ist ein Wrapper für alle Objekte, die im Nameservice gebunden werden. Hier wird mittels Funktion callMethod die aufzurufende Funktion mit Hilfe von Reflection ausgeführt

Ablauf callMethod

1. Aufsplitten der Nachricht, um Methodennamen, Parameter und ihre Datentypen zu extrahieren
2. Suchen der Funktion durch Name und Class-Array mit Datentypen der Parameter
3. Aufruf der Funktion mittels Funktion invoke, Parameter: Referenz des Objekts und Object-Array mit Datentypen
4. Rückgabe des Ergebnisses aus dem Funktionsaufruf als String mit "result:äls Präfix

5. Treten beim Aufruf Exceptions auf, werden diese gefangen und der Aufrufer wird darüber mittels der Antwort ëxception:<ExceptionClass>:<ExceptionMessage>"darüber benachrichtigt

3.2.5 RemoteObject

Das RemoteObject ist der Kommunikationspartner des Dispatchers und wird vom NameServiceInterface zurückgeliefert. Es speichert die Hostadresse und den Port des Servers, auf dem das LocalObject liegt. Wird die Funktion callMethod aufgerufen, wird ein Socket erstellt und der Dispatcher dazu angeleitet, die Funktion des lokalen Objekts aufzurufen. Das Ergebnis wird direkt zurückgeliefert.

Ablauf callMethod

- Erstellen eines Sockets mit der gespeicherten Hostadresse und dem Port
- Senden der Nachricht über den Socket
- Empfangen der Antwort
- Rückgabe der Antwort

3.3 IDL-Compiler

Der IDL Compiler wandelt IDL-Dateien in die in der Datei beschriebenen Klassen um. Aus jeder beschriebenen Klasse wird jeweils eine Abstrakte Klasse und eine Klasse mit Implementierung erzeugt.

3.3.1 Ablauf

- Einlesen der Datei
- Speichern des Modulnamens
- Einlesen der Klassen und ihrer Funktionen mit Parametern
- Speichern der Klassen, Funktionen und Parametern als Objekte

- Umwandeln der Klassen mit ihren Funktionen und deren Parametern in die Abstrakten Klassen
- Schreiben der Funktion narrowCast, bei der ein Objekt aus dem NameService übergeben werden kann
    1. Das übergebene Objekt wird zu einem MWareObject gecastet und damit wird ein neues Objekt der Klasse mit Implementierung erzeugt und zurückgegeben
- Erzeugen der Klasse mit Implementierung, bei der die jeweilige Funktion mit ihren Parametern als String zusammengebaut wird, dass mit diesem die Funktion callMethod des MWareObjects aufgerufen werden kann.
    1. Aufruf der Funktion mit dem zusammengebauten String
    2. Casten der Antwort auf den Rückgabetypen der Funktion und Rückgabe des Ergebnisses

4 Fazit

Nach anfänglicher Planlosigkeit bei dieser Aufgabe, machte ich mir genauere Gedanken, wie ich diese angehen könnte und las mir noch einmal die Folien der Aufgabenvorstellung durch. Desto mehr ich mich damit beschäftigte, desto mehr Ideen kamen mir zur Implementierung. Ich entschied mich, diesen Lösungsansatz zu implementieren.
Danach ging der Entwurf erst relativ leicht von der Hand. Nachdem ich jedoch mit der Implementierung begann, musste ich den Entwurf einige male abändern, da meine Ideen so nicht ganz funktionierten.
Vor allem aber das Testen bereitete mir Probleme. Da die einzelnen Module relativ schwierig zu testen sind, musste ich erst alle Module fertigstellen, um vernünftig testen zu können. Zwar hätte man für jedes Modul ein ausgiebiges Teststsystem schreiben kännen, jedoch hätte dies meinen zeitlichen Rahmen gesprengt.
Dadurch, dass ein vernünftiges Testen erst möglich war, wart es auch schwierig, auftretende Fehler zu entdecken. Dies gelang jedoch dank der Debug-Ausgaben sehr gut.
Das gegebene Testszstem bereitete noch einmal Probleme, da mein System zwar mit meinen Tests funktionierte, hier jedoch anfänglich neue Fehler auftraten. Diese Fehler traten auf, weil z.B. mein Nameservice in einem Subpackage lag und falsch benannt war. Hier wurde als Fehler angezeigt, dass die Funktion getNameservice nicht vorhanden sei. Diese war aber implementiert. Ein Versuch, den Nameservice umzubenennen und in das Hauptpackage zu verschieben war jedoch erfolgreich.
Ein zweiter Fehler trat auf, da ich die Funktion shutDown nicht implementiert hatte. Diese konnte jedoch leicht nachimplementiert werden.
Der Compiler war ebenfalls eine Herausforderung. Hierbei orientierte ich mich an dem Compiler, der von Herrn Schulz bereitgestellt wurde. Ich entschied mich jedoch, nicht mit statischen Variablen zu arbeiten, da dies in meinen Augen keinen Sinn machte, da die Zeichen ziemlich kurz sind und ich so schneller implementieren konnte. Eine Änderung des Compilers war ja sowieso nicht vorgesehen.

ERKLÄRUNG ZUR SCHRIFTLICHEN AUSARBEITUNG DES REFERATES

Hiermit erkläre ich, dass ich diese schriftliche Ausarbeitung meines Referates selbstständig und ohne fremde Hilfe verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe sowie die aus fremden Quellen (dazu zählen auch Internetquellen) direkt oder indirekt übernommenen Gedanken oder Wortlaute als solche kenntlich gemacht habe. Zudem erkläre ich, dass der zugehörige Programmcode von mir selbständig implementiert wurde ohne diesen oder Teile davon von Dritten im Wortlaut oder dem Sinn nach übernommen zu haben. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Ort	Datum	Unterschrift im Original