**Team**: 05, Eugen Deutsch, Ralf von der Reith

**Aufgabenaufteilung**:

Die Aufgaben wurden gemeinsam bearbeitet

**Quellenangaben**: -/-

**Bearbeitungszeitraum**:

* + - 1. 12.06.2017 – 3 Stunden
      2. 13.06.2017 – 4 Stunden
      3. 14.06.2017 – 2 Stunden
      4. 16.06.2017 – 2 Stunden
* 11 Stunden

**Aktueller Stand**:

* Der Entwurf ist fertig

**Änderungen des Entwurfs**: -

# Komponenten

Das Projekt besteht aus vier separat startbaren Anwendungen, nämlich dem Client, dem Server, dem IDLcompiler und dem NameServer, wobei der Client, der Wrapper und der Server die mware\_lib einbinden.

Im groben Ablauf sieht es dann wie folgt aus: Der IDLcompiler wird mit dem Dateinamen der idl-Datei als Argument gestartet und erzeugt daraus Java Code.

Unabhängig davon werden der NameServer, der Server und der Client gestartet. Zunächst erstellt und registriert der Server mithilfe der mware\_lib ein Objekt beim NameServer, woraufhin der Client mithilfe seiner mware\_lib und dem vom IDLcompiler erstellten Wrapper Methodenaufrufe darauf durchführen kann.

## mware\_lib

Die mware\_lib bietet dem Anwender Komponenten zum Arbeiten mit entfernten Objekten. Auf der Serverseite kann man mithilfe der mware\_lib Objekte beim NameServer anmelden und sie für entfernte Aufrufe zur Verfügung stellen, wohingegen man sich als Anwender die Referenz für so ein Objekt holen kann. Sie besteht ausfolgenden Teilen:

ObjectBroker: Vorgegebene Klasse, die als Frontend für den Nutzer dient. Beim Aufruf von init werden hier die Dienste initialisiert und der NameService wird zur Verfügung gestellt.

NameService: Vorgegebene Klasse, die für das registrieren von Objekten unter bestimmten Namen und das holen einer Referenz du diesen dient. Außerdem enthält diese Klasse eine Map, die unter gegebenen Referenznamen Dispatcher speichert (serverseitig).

NameServiceConnection: Eine Klasse, die vom NameService benutzt wird, um Befehle an den NameServer zu schicken.

MethodCallListener: Horcht auf eingehende entfernte Methodenaufrufe, erstellt für jede solche Anfrage einen RequestHandler und übergibt ihm das eingehende rmiObject mit einer Referenz zum NameService.

RequestHandler: Holt sich aus dem NameService mit dem Namen des rmiObjects einen Dispatcher und übergibt ihm das rmiObject um das Ergebnis für den Methodenaufruf zu erhalten und diesen zurückzuschicken.

Dispatcher: Ein Hilfsobjekt, dass jeweils ein konkretes Objekt enthält und ein rmiObject annimmt, um über Java Reflection die Methode aufzurufen.

RmiObject: Enthält den Namen der Referenz, den Namen der aufzurufenden Methode, ihre Parameter und Parametertypen. Wird vom Wrapperobjekt (IDLcompiler) verschickt und serverseitig durch die mware\_lib akzeptiert, womit dann der Methodenaufruf stattfinden kann.

## NameServer

## IDLcompiler

Erzeugt mithilfe einer Idl-Datei Java Code.

Aus…

**module** Math {  
 **class** Calculator {  
 **int** add(**int** a, **int** b);  
 **double** div(**int** a, **int** b);  
 **string** asString(**int** a);  
 };  
};

wird…

**package** Math;  
  
**import** ...  
  
**public abstract class** \_CalculatorImplBase {  
 **private static** \_CalculatorImplBase narrowCast(Object objectRef) { ... }  
   
 **public abstract int** add(**int** a, **int** b) **throws** Exception;  
 **public abstract double** div(**int** a, **int** b) **throws** Exception;  
 **public abstract** String toString(**int** a) **throws** Exception;  
   
 **private class** Calculator **extends** \_CalculatorImplBase {  
 @Override  
 **public int** add(**int** a, **int** b) **throws** Exception { ... }  
  
 @Override  
 **public double** div(**int** a, **int** b) **throws** Exception { ... }  
  
 @Override  
 **public** String asString(**int** a) **throws** Exception { ... }  
 }  
}

Diese Klasse kann nun benutzt werden, um Methodenaufrufe auf entfernt liegenden Objekten durchzuführen. Dazu holt man sich durch narrowCast zunächst ein Objekt des Typen und führt dann darauf die Methoden aus. Diese Aufrufe werden vom Objekt weitergeschickt und vom Server abgefangen, der ihm dafür das Ergebnis zurückschickt.

Um eventuelle Exceptions an den Nutzer zu geben, wird vom Objekt geprüft, ob das erhaltene Objekt vom Typ Exception ist und gegebenenfalls geworfen. Ansonsten bekommt der Nutzer das Ergebnis.

## Sonstiges

Zum Testen wird von uns die im IDLcompiler Abschnitt erwähnte Idl-Datei benutzt. Diese enthält alle zu unterstützenden Datentypen. Im Falle von div wird eine Exception geworfen, sofern a == 0 ist, wir somit eine Division durch null in Auftrag gegeben haben.

# Anwendung

## Client

## Server

# Ablauf

## Rebind

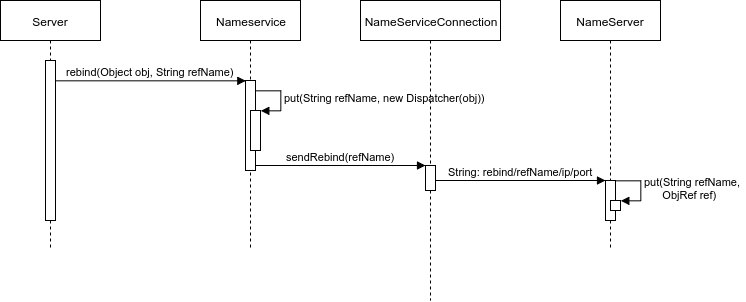


Abbildung 1 Rebind

1. Es wird rebind mit dem zu registrierenden Objekt und einem zugehörigen Namen als Referenz aufgerufen.
2. Der NameService speichert sich das Objekt unter dem Namen und schickt einen String, der wie folgt aufgebaut ist, los: rebind/refName/ip/port
3. Der NameServer erhält die Nachricht und baut den String entsprechend auseinander.
4. Der NameServer speichert sich unter dem refName ein refObj. Das refObj ist ein String in Form von: refName/ip/port.

## Resolve

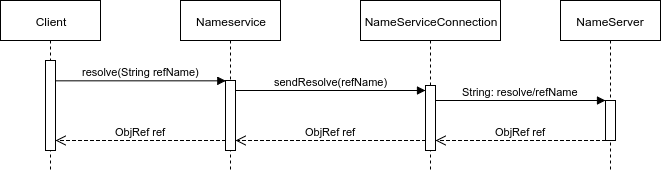


Abbildung 2 Resolve

1. Der Client fragt per resolve mit dem entsprechenden refName nach dem refObj.
2. Der NameService baut eine Anfrage bestehend aus resolve/refName zusammen und schickt diese ab.
3. Der NameServer erhält die Anfrage und schickt das zum refName passende, zuvor gespeicherte refObj zurück.
4. Der NameService erhält das refObj und gibt dieses zurück.

## NarrowCast

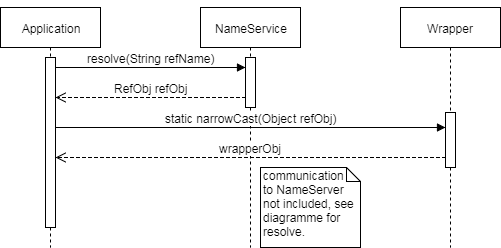


Abbildung 3 NarrowCast

1. Resolve wird ausgeführt (siehe Ablauf bei Resolve).
2. Der Client erhält ein refObj.
3. Der Client ruft beim vom IDL Compiler erstellten Objekt die narrowCast Methode auf, um ein konkretes Wrapperobjekt zu erhalten.

Auf dem Wrapperobjekt lassen sich nun die entsprechenden Methoden aufrufen (siehe Method call).

## Method call

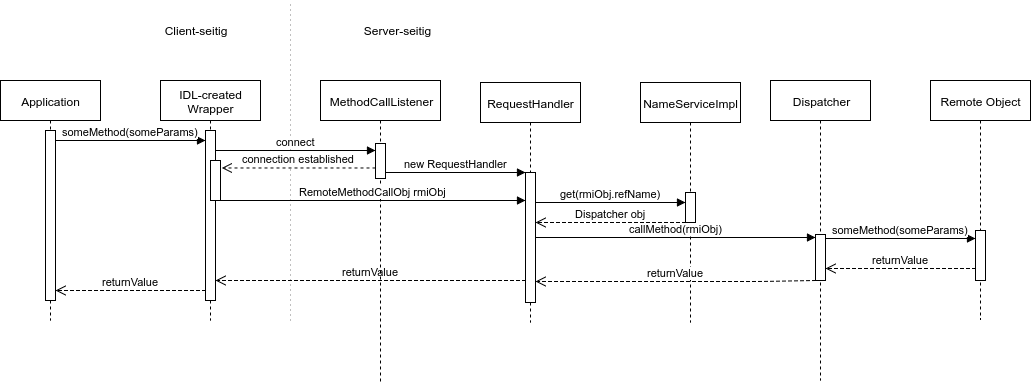


Abbildung 4 Method call

1. Die Client Applikation ruft die Methode eines vom IDL Compiler erstellten Objektes auf.
2. Das Wrapperobjekt stellt eine Verbindung her und sendet den Aufruf mit den entsprechenden Parametern weiter.
3. Der RequestHandler fängt diese ab und fragt beim NameService nach dem zugehörigen Objekt.
4. Per Java Reflection wird die Methode anhand der vom rmiObj gegebenen Daten (Methodenname, Parameter, Parametertypen) ausgeführt und das Ergebnis wird zurückgeschickt.
5. Der Wrapper erhält das Ergebnis und prüft, ob es vom Typ „Exception“ ist. Falls ja, so wird diese geworfen, wodurch der Client die Serverseitige Exception bekommt. Anderenfalls wird es zum entsprechenden Rückgabetyp der aufgerufenen Methode gecastet und zurückgegeben.

# Kommunikation

Zur Kommunikation zwischen Server, Client und NameServer werden Java Sockets verwendet. Die Nachrichten sehen dabei wie folgt aus:

Server über mware\_lib zu NameServer: rebind/refName/ip/port als String

Client über mware\_lib zu NameServer: resolve/refName als String

NameServer zu Client mware\_lib: refObj (als String: refName/ip/port)

Wrapper zu Server: rmiObject (durch ObjectOutputStream)

Server zu Wrapper: Object (double | string | int | Exception)