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

Aufgabenaufteilung:

1. Ralf:
2. Eugen:

Quellenangaben: -/-

Bearbeitungszeitraum:

## Aktueller Stand:

* Der Entwurf ist fertig

## Änderungen des Entwurfs: -

# TODO/Anmerkungen

• Der RequestHandler enthält den NameService (mware\_lib)? Das ist rein von der Lokalität her unschön,

weil ich als Außenstehender von einem Handler nicht erwarten würde, dass dieser noch einen NameService

mit der Map enthält. Andersherum wäre das schon passender.

Vom RequestHandler selbst würde ich erwarten, dass er sich wirklich nur um eingehende Daten und ausgehende

Daten kümmert und nicht um Berechnungen, um die konkreten Anfragen zu erfüllen.

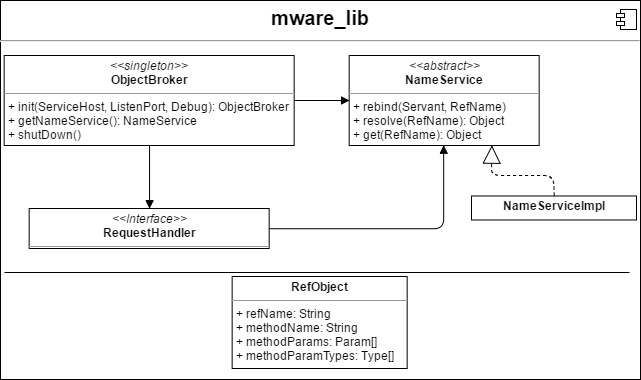
Aus dem Grund finde ich es auch etwas unpassend, dass der RequestHandler Reflection benutzt, um die Methode

aufzurufen. Wenn man es dabei belässt, dass der RequestHandler den NameService enthält, so würde ich ihn

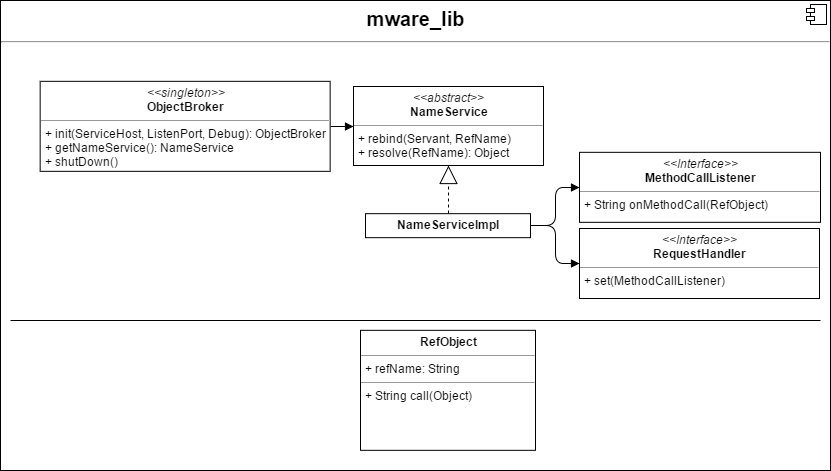
das rmiObj an diesen weiterschicken lassen, woraufhin der NameService dem rmiObj das konkrete Objekt aus seiner Map gibt und das rmiObj regelt dann den Reflection Methodenaufruf. rmiObj impliziert eigentlich schon, dass hier der Methodenaufruf stattfindet.

• NameService hast du mal klein geschrieben (Service), aber bei NameServiceImpl wiederrum Ordnungsgemäß in PascalCase. (Selbes bei refName und refObj heißt es bei rebind, wohingegen es bei resolve objRef heißt)

# A



# B



# 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

## NameServer

Der Namensdienst bietet Auskunft über den Speicherort von Objekten im Netzwerk. Dazu wird ein referenzierender Name zusammen mit der IP-Addresse und dem Port des Servers gespeichert, welcher dann von anderen Nutzern (Clients) ausgelesen werden kann.

Zur Speicherung der Referenzen wird als Datenstruktur eine Hashmap genutzt. Die Namen dienen als Schlüssel, über den sich die Addresse und der Port zurückgeben lassen.

Nach außen stellt der Namensdienst somit zwei Funktionen bereit:

1. Resolve:

* Nachschlagen des Speicherorts eines Objektes anhand seines Referenz-Namens.

1. Rebind:
   * Bereitstellen von Objekten für das System durch das Hinterlegen einer Referenz zum Speicherort.

Beim Start muss dem Namensdienst ein Port übergeben werden, über den dieser ansprechbar ist. Für den Abruf der Daten stellt das anfragende System dann eine TCP-Verbindung her und beginnt nun die Kommunikation entsprechend der unten definierten Schnittstelle.

Schnittstelle:

Die Übertragung der Daten geschieht mithilfe von Java‘s ObjectStreams

**resolve**

Anfrage:

String: resolve/<Referenz>

sucht nach einem Eintrag zur übergebenen Referenz. Bei Treffer wird diese wie folgt zurück gegeben:

Antwort:

ObjRef

**rebind**

Anfrage:

String: rebind/<Referenz>/<IP-Addresse>/<Port>

Trägt die Addresse und den Port unter der gegebenen Referenz ein. Überschreibt vorherige Einträge ohne Warnung.

## 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** toString(**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 toString(**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.

# Ablauf

## Rebind

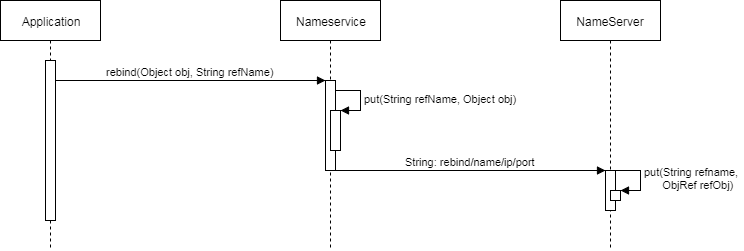


Abbildung 1 Rebind

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

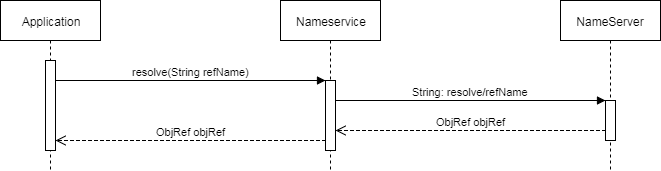


Abbildung 2 Resolve

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

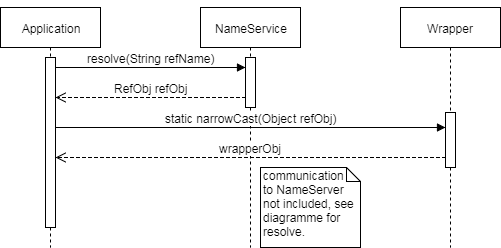


Abbildung 3 NarrowCast

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

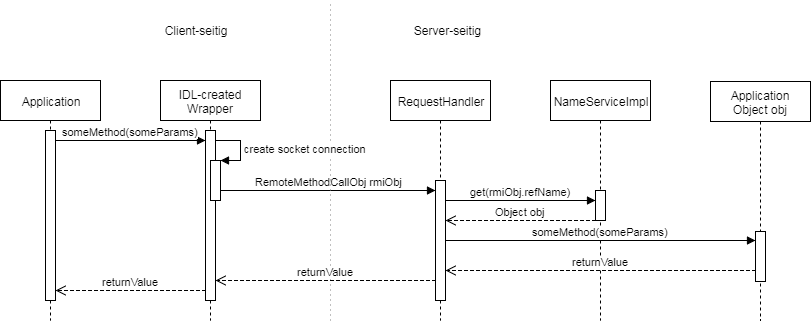


Abbildung 4 Method call

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)