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

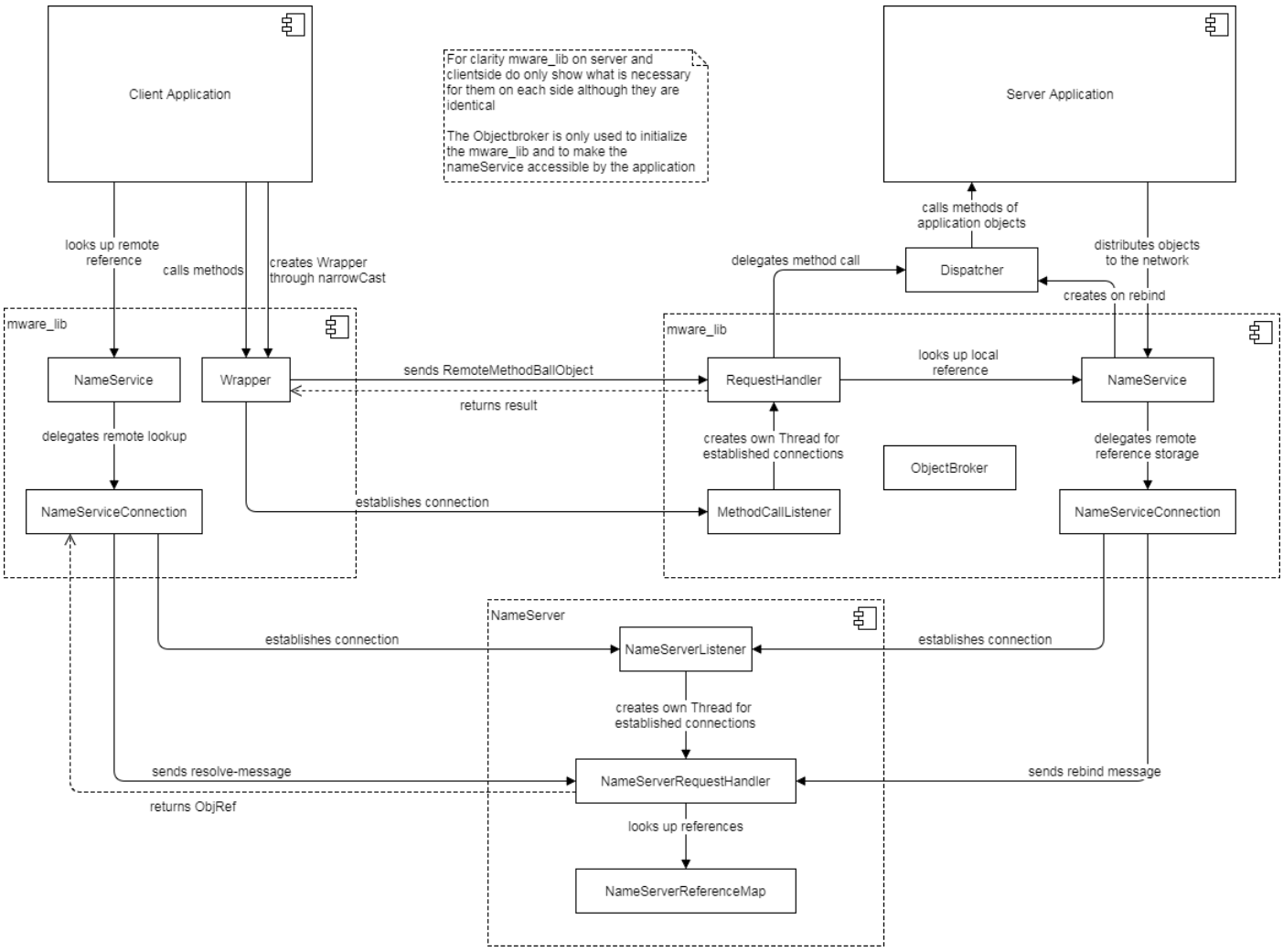
Aktueller Stand:

- Der Entwurf ist fertig

Änderungen des Entwurfs: -

Komponenten

Übersicht



Das Projekt besteht aus vier separat startbaren Anwendungen, nämlich dem Client, dem Server, dem IDLcompiler (der zum größten Teil durch das gegebene Beispiel vorgegeben ist) und dem NameServer, wobei der Client, der vom IDLcompiler erstellte 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 für die Wrapperklasse, die für den entfernten Methodenaufruf zuständig ist.

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.

Im Folgenden werden die Aufgaben der Komponenten Beschrieben:

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 das 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

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 Map genutzt. Die Namen dienen als Schlüssel, über den sich die Adresse 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.
2. 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 Javas ObjectStreams.

resolve

Anfrage: String: resolve/<Referenz>

Sucht nach einem Eintrag zur übergebenen Referenz. Bei Treffer wird diese wie folgt zurückgegeben:

Antwort: ObjRef

rebind

Anfrage: String: rebind/<Referenz>/<IP-Adresse>/<Port>

Trägt die Adresse 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 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

Server

```
ObjectBroker objBroker = ObjectBroker.init(host, port, debug);
NameService nameService = objBroker.getNameService();
nameService.rebind((Object) new Calculator(), "calculator");
```

Client

```
ObjectBroker objBroker = ObjectBroker.init(host, port, debug);
NameService nameService = objBroker.getNameService();
Óbject refObj = nameService.resolve("calculator");
_CalculatorImplBase wrapper = _CalculatorImplBase.narrowCast(refObj);

// Und dann können über den Wrapper die entfernten Methodenaufrufe durchgeführt werden
try {
    wrapper.add(0, 0);
} catch (Exception e) {
    ...
}
```

Ablauf

Rebind

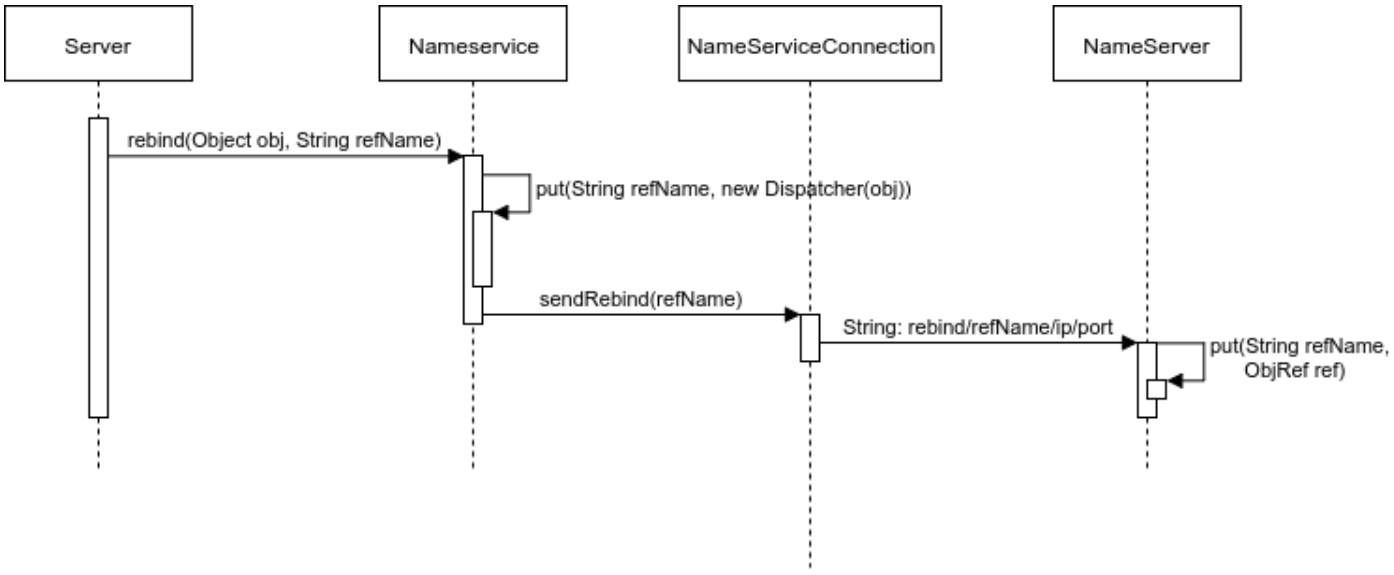


Abbildung 1 Rebind

1) Es wird rebind mit dem zu registrierenden Objekt und einem zugehörigen Namen als Referenz aufgerufen.
2) Der NameService speichert sich das Objekt in einem Dispatcherobjekt unter dem Namen und schickt einen String, der wie folgt aufgebaut ist über die NameServiceConnection 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

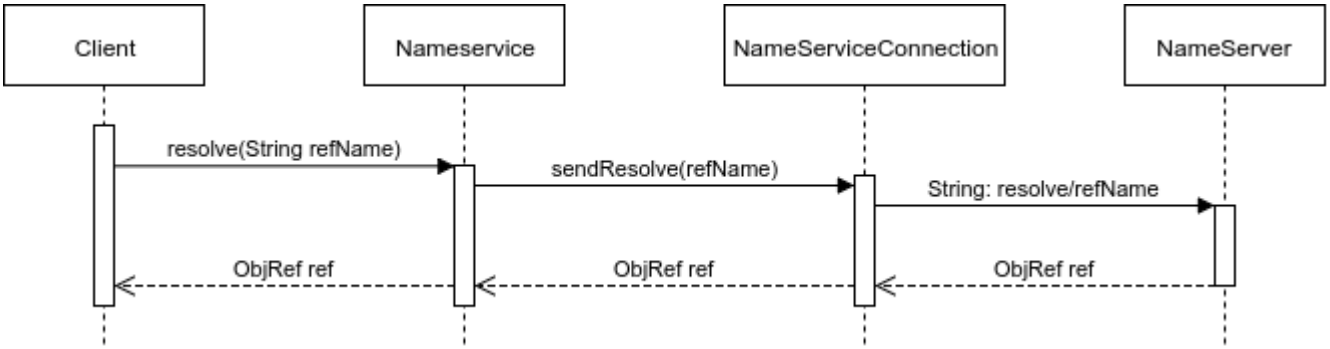


Abbildung 2 Resolve

1) Der Client fragt per resolve mit dem entsprechenden refName nach dem refObj.
2) Der NameService übergibt die Aufgabe an die NameServiceConnection. Diese 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) Die NameServiceConnection erhält das Objekt, gibt es an den NameService zurück und dieser gibt es wiederrum an den Client zurück.

NarrowCast

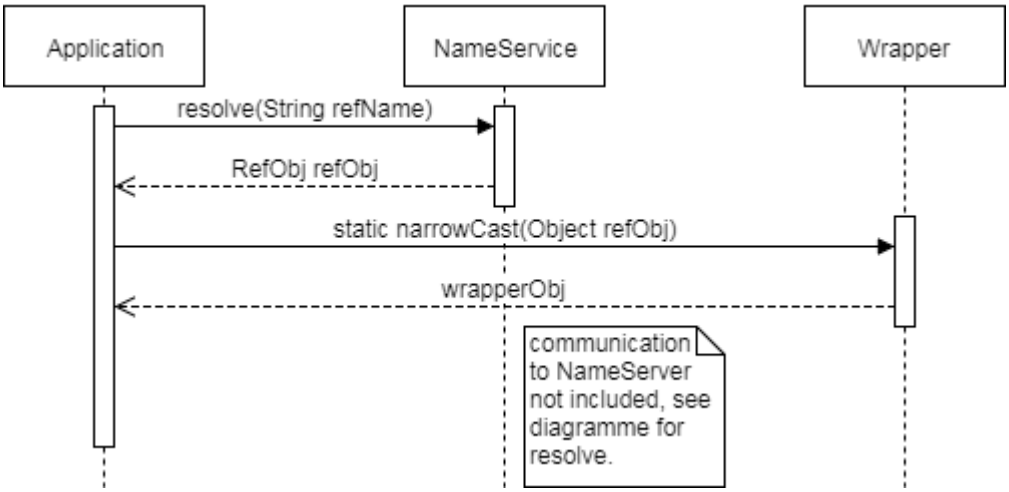


Abbildung 3 NarrowCast

1) Resolve wird ausgeführt (siehe Ablauf bei Resolve).
2) Der Client erhält ein refObj.
3) Der Client ruft bei der Wrapperklasse 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

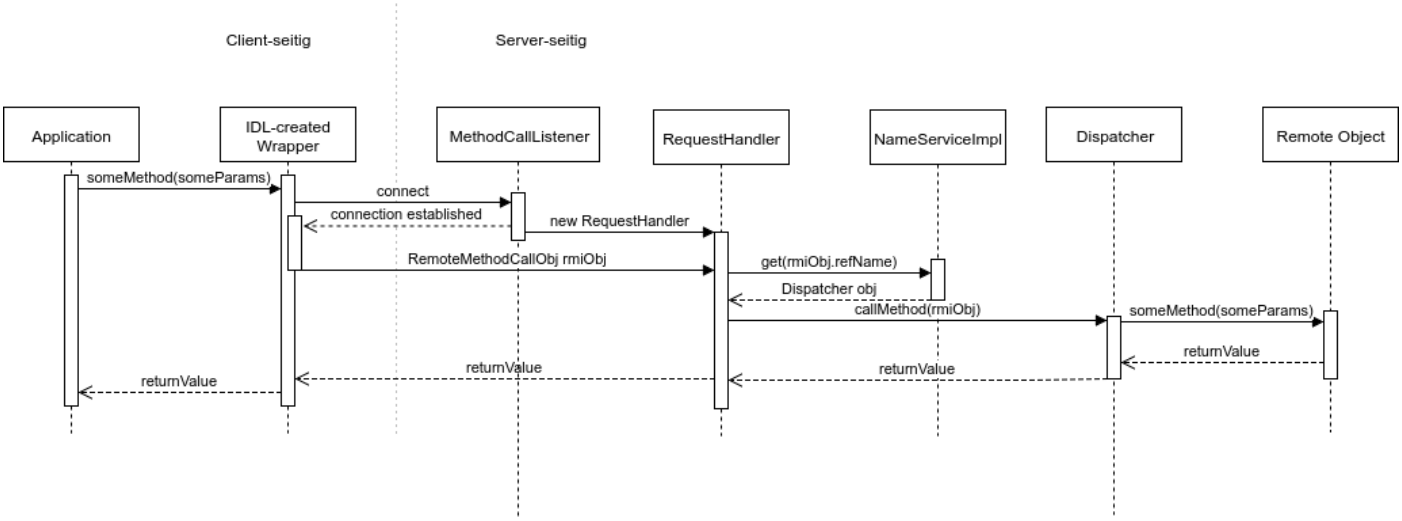


Abbildung 4 Method call

1) Die Client Applikation ruft die Methode des Wrappers auf.
2) Das Wrapperobjekt stellt eine Verbindung her und sendet den Aufruf mit den entsprechenden Parametern weiter.
3) Der MethodCallListener fängt den Aufruf ab und erstellt für die Anfrage einen RequestHandler mit einer Referenz zum NameService und dem rmiObj.
4) Der RequestHandler fragt beim NameService nach dem Dispatcher und gibt diesem dann das rmiObj.
5) Per Java Reflection wird die Methode anhand der vom rmiObj gegebenen Daten (Methodenname, Parameter, Parametertypen) im Dispatcher ausgeführt und das Ergebnis wird zurückgeschickt.
6) 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)