Team: 6, Mert Siginc

Quellenangaben:
- http://users.informatik.haw-hamburg.de/~klauck/verteiltesysteme.html

Bearbeitungszeitraum:
Ca. 70 Stunden insgesamt

Aktueller Stand:
Fertig.

Änderungen des Entwurfs:
-

Entwurf:

Gliederung:

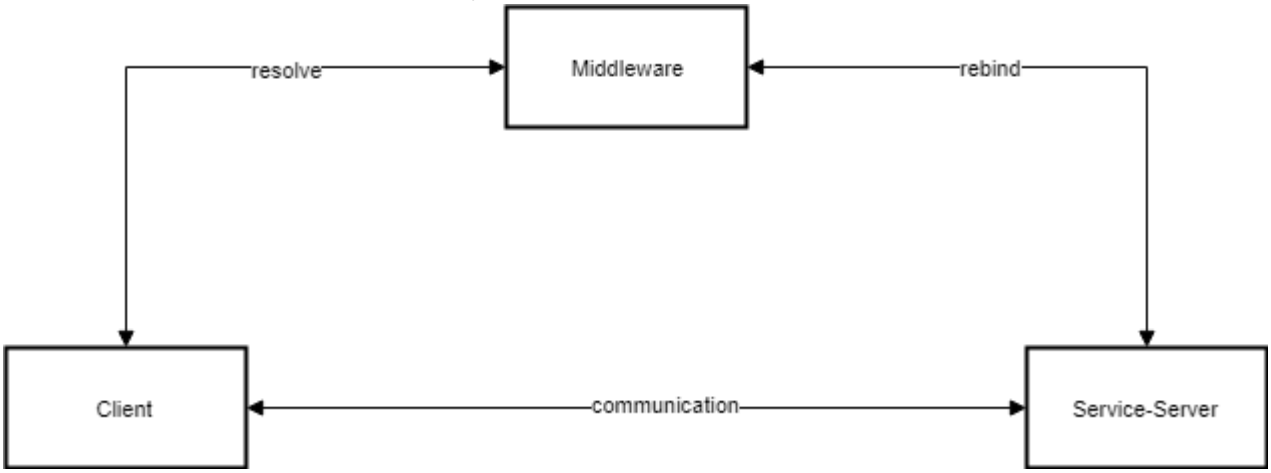
1. Allgemeine Beschreibung
2. Wie ist die Architektur aufgebaut?
3. Wie sehen die einzelnen Komponenten aus?
    a. Middleware
    b. Client
    c. Service
4. Wie sieht das Kommunikationsprotokoll aus?
5. Klassenbeschreibungen
6. Kritische abschnitte
7. IDL-Compiler
8. Exceptions

Allgemeine Beschreibung:

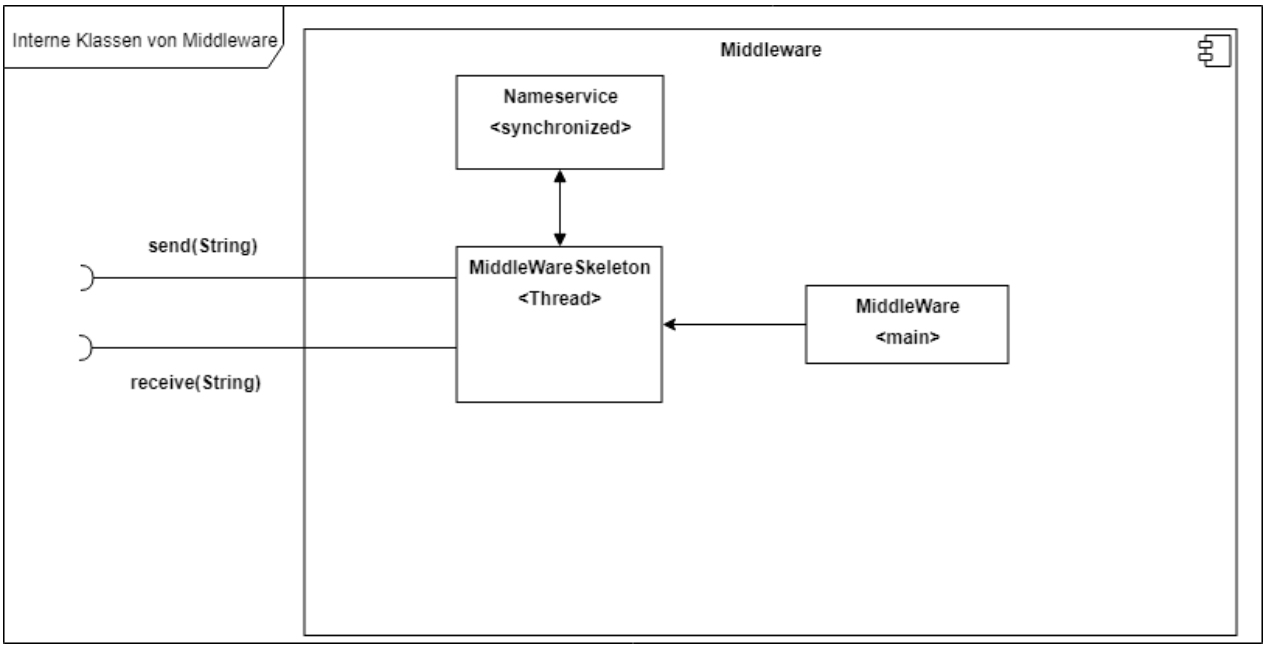
Die Anwendung, die ich zu realisieren habe, ist eine CORBA Anwendung, welches ein Programmiersprachen offenes RMI ist. RMI (Remote Method Invocation) ist ein verfahren, Methoden auf verteilten Objekten lokal aufzurufen. Dabei wird dem Client und dem Service-Server die selben IDL-Daten zur Verfügung gestellt, die sie dann Ihrer Sprachegehörig parsen und Compilern können, um mit diesen arbeiten zu können.

Wie ist die Architektur aufgebaut?

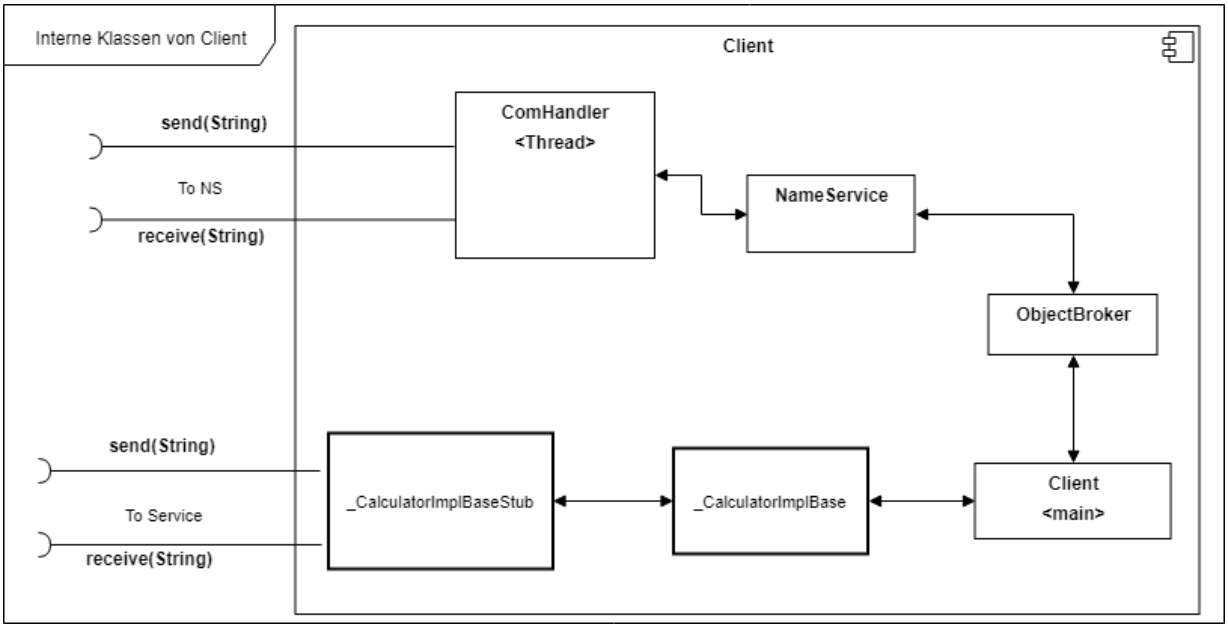
Die Architektur meiner Anwendung sieht vor, dass es eine Middleware einen Service-Server und einen Client gibt. Zu den einzelnen Punkten werde ich noch weiter unten im Entwurf detaillierter eingehen. Im Folgenden werde ich die Service-Server als Service/Services benennen.



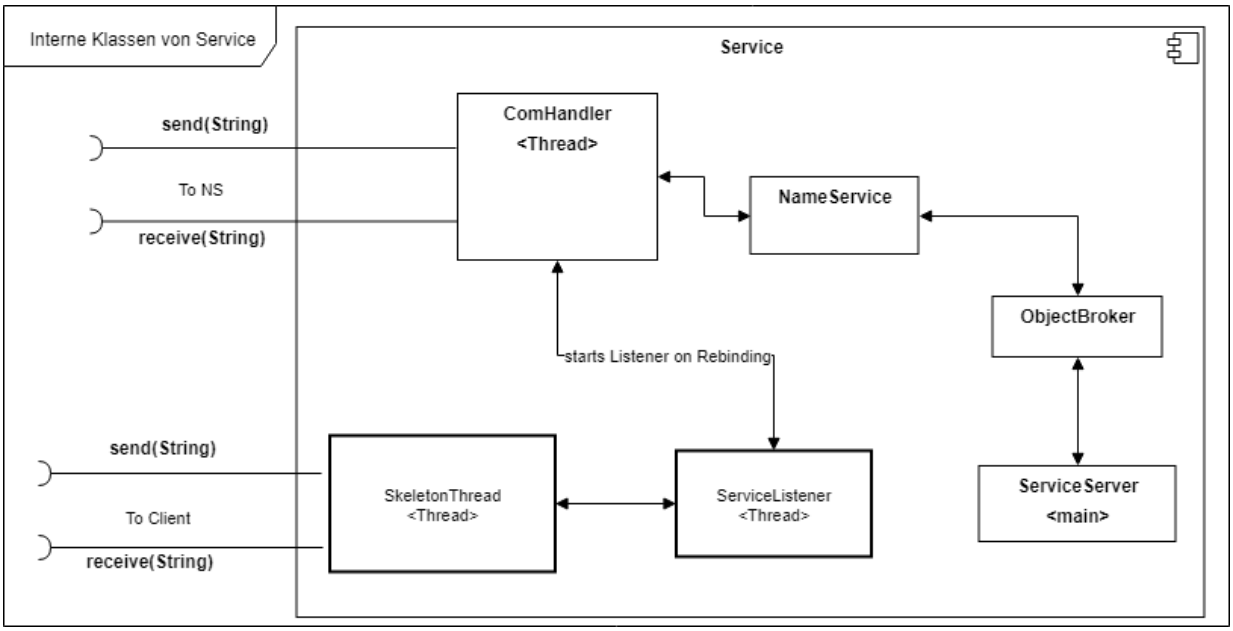
Zum Anmelden und erfragen eines Services wird die Schnittstelle der Middleware angesprochen, welches den Namensservice verwaltet. Der Service übergibt seine IP und Port und den eindeutigen Servicenamen, welche im Middleware im Nameservice hinterlegt wird. Der Client fragt die Middleware, ob es einen Service mit einem bestimmten Namen gibt. Ist dies der Fall bekommt dieser die IP und Port vom Service.
Von nun an kommunizieren Client und Service unter sich.



Die Middleware ist folgendermaßen aufgebaut:
Sie verfügt über einen MiddleWareSkeleton, welches die ankommenden Anfragen beantwortet und den Namenservice an sich, welches die Services verwaltet.



Der Client ist folgendermaßen aufgebaut:
Der Client hat einen ObjectBroker, welches die Schnittstelle zur Middleware und somit zum Nameservice ist. Wird eine Nameservice Methode aufgerufen, wird diese an den ComHanlder weitergegeben, um die Anfrage an die Middleware weiterzuleiten. Zudem besitzt der Client die aus einer IDL-Datei generieten Klassen, in diesem Fall _CalculatorImplBase und _CalculatorImplBaseStub. _CalculatorImplBaseStub ermöglicht uns hierbei den entfernten Aufruf, welches durch das narrowCast eingebunden wird. Dieser kommuniziert dann mit dem Service über seine Schnittstelle und gibt die Antwort an den Client zurück.



Der Service ist folgendermaßen aufgebaut:
Wie auch beim Client hat der Server den ObjectBroker, welches als Schnittstelle zur Middleware und somit zum Nameservice ist. In diesem Fall gibt es nur einen unterschied. Beim „rebind“ Aufruf erzeugt der ComHandler einen ServiceListener, welches von nun an die Anfragen der Clients über den SkeletonThread entgegennimmt und beantwortet.

Wie sehen die einzelnen Komponenten aus?

Middleware

Beim startet der Middleware wird eine TCP-Schnittstelle geöffnet worüber der Client und die Services ihre Anfragen an die Middleware senden können.
Für Jede eingegangene Verbindung wird ein neuer „Arbeiter“ mit dem Namen „MiddlewareSkeleton“ erzeugt, welches die die eingegangenen Anfragen verarbeitet und beantwortet, hiernach wird diese wieder geschlossen. Die „Arbeiter“ arbeiten mit dem Nameservice, welches die Services mit jeweils dessen Namen, IP und PORT abspeichert. In Java bietet sich hierbei eine Map an. In anderen Sprachen könnte man auch eine Liste verwenden.
In Java würde sich folgende Struktur in der Map anbieten:
„ServiceName“(String) -> „IP|PORT“(String), hierbei dient das „|“ als Trennzeichen.

Client

Der Client verwendet die von uns zu erstellende Bibliothek mware_lib, welches alle nötigen Funktionen zur Kommunikation mit der Middleware und dem Service beinhaltet.
Beim Starten des Clients, wird wie in der Aufgabenbeschreibung vorgegeben ein ObjectBroker Objekt erzeugt, welches für uns als Schnittstelle zur Middleware dient.
Der ObjectBroker erzeugt eine ComHandler-Instanz, welches die Kommunikation zwischen Client und MW realisiert. Diese braucht der Nameservice um die Anfragen vom Client an die MW zu schicken.
Der ObjektBroker gibt uns bei Abfrage den Nameservice, welches mit dem ComHandler arbeitet.
Wird nun Resolve aufgerufen, wird die Anfrage über den ComHandler an die Middleware weitergeleitet und als Antwort erhalten wir die Referenz zum Service, welches wie in der Middleware beschrieben die IP und PORT der Services ist. Dies wird für die von einer IDL-Datei erzeugten Klassen benötigt.
Die Klassen erzeugen dann durch das narrowCast eine Schnittstelle zum Service, wie auch der ObjectBroker.
Wird nun vom Client eine Methode des IDL-Objektes aufgerufen, so wird diese an den Service über TCP weitergeleitet und die Antwort vom Service zurückgegeben.

Service

Der Service verwendet die von uns zu erstellende Bibliothek mware_lib, welches alle nötigen Funktionen zur Kommunikation mit der Middleware und dem Client beinhaltet.
Zudem implementiert er die von der IDL-Datei erzeugten Hauptklasse.
Beim Starten des Services, wird wie in der Aufgabenbeschreibung vorgegeben ein ObjectBroker Objekt erzeugt, welches für uns als Schnittstelle zur Middleware dient.
Der ObjectBroker erzeugt eine ComHandler-Instanz, welches die Kommunikation zwischen Service und MW realisiert. Diese braucht der Nameservice um die Anfragen vom Service an die MW zu schicken.
Der ObjektBroker gibt uns bei Abfrage den Nameservice, welches mit dem ComHandler arbeitet.
Wird nun über das Nameservice-Objekt Rebind aufgerufen, wird eine Schnittstelle nach außen geöffnet und die Informationen worüber diese Schnittstelle zu erreichen ist, (Das Protokoll wird weiter unten erklärt) an die Middleware weitergeleitet.
Von nun an ist der Service online und bereit Anfragen zu beantworten.
Verbindet sich nun ein Client, wird der SkeletonThrad aufgerufen, welches nun die Anfragen vom Client verarbeitet und beantwortet. Da wir nur auf Object-Klassen(Java) arbeiten, ist das verarbeiten der Anfragen und somit das aufrufen einer Methode eines Objects, so ohne weiteres nicht möglich. Hierbei bietet sich Reflection an, welches eine Methode in einer Object-Klasse sucht und bei Fund diese ausführt. Hiernach wird das Ergebnis an den Anfragesteller zurückgeschickt.

Wie sieht das Kommunikationsprotokoll aus?

Die TCP-Kommunikation sieht vor, dass ausschließlich Strings verschickt werden, so dass im CORBA-Umfeld jede Programmiersprache die Nachrichten verarbeiten kann.

Zur Middleware:

Beim rebind: „host|ip|nsName|command(rebind)“

Beim resolve: „null|null|nsName|command(resolve)“

Zum Service:

„methodeName|param1, param2, ..., paramX“

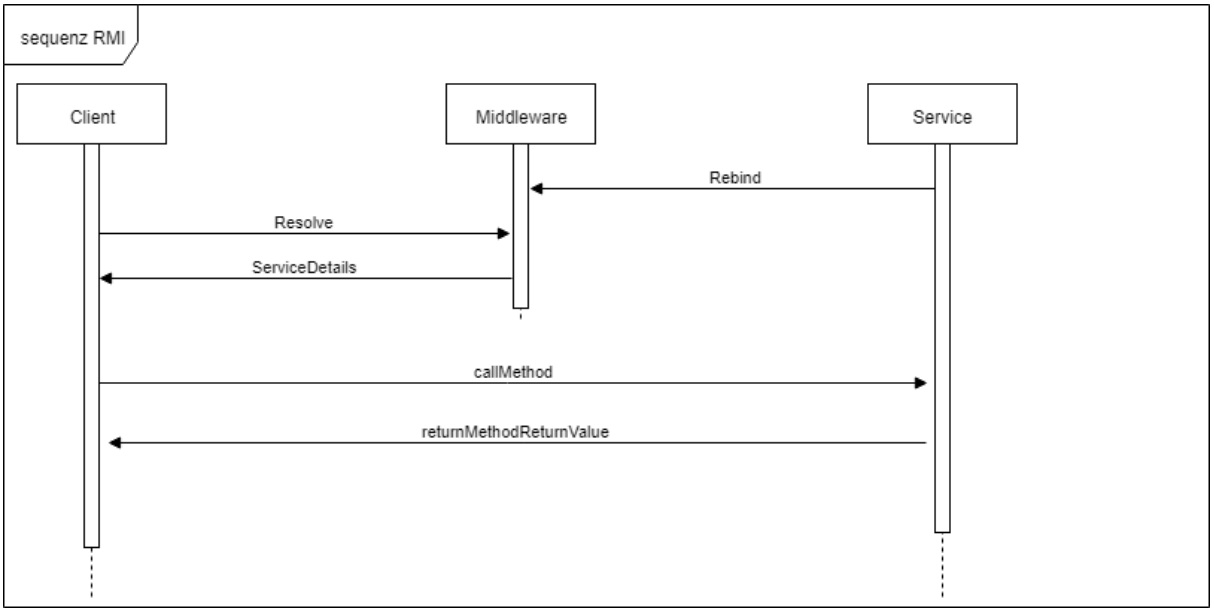
Zum Client:

Von der Middleware: „host|ip“

Vom Service: „answer“

Legende:

Commands:	„resolve“, „rebind“	Sagt der Middleware was er tun soll.
Host:	Host des Services.	
Ip:	Ip des Services.	
nsName:	Der Name wie ein Service im Nameservice hinterlegt werden soll.	
methodeName:	Name der Methode die aufgerufen werden soll.	
paramX:	Die mitgegebenen Parameter.	
answer:	Die Antwort vom Service auf den Methodenaufruf.	
null:	Einfach null als String, kann man aber durch ein beliebigen String ersetzen, da diese nicht verarbeitet werden.	



Klassenbeschreibungen

Mware_lib:

ComHandler: Der ComHandler erwartet host, ip und ein debugFlag, als Parameter.
Beim initialisieren des Objektes wird eine Verbindung zu den mitgegebenen Parametern aufgebaut.
Beim Aufruf von sendToNs oder SendToService, wird dann Protokollkonform ein String an die gegenstelle gesendet und ggf. auf eine Antwort gewartet. Falls eine Antwort erwartet wird und dieser erhalten wurde, wird diese zurückgegeben. Beim rebind erzeugt es einen ServiceListener.
INameService: Ist ein Interface, welches die Schnittstellen zum Nameservice der Middleware beschreiben.
NameService: Der NameService implementiert das INameService Interface und gibt die aufrufe seiner Methoden an den ComHandler weiter, so dass diese an die Middleware gesendet werden.
ObjectBroker: Der ObjectBroker erzeugt eine neue ComHanlder-Instanz, welches eine Verbindung zur Middleware aufbaut. Hiernach wird eine Nameservice-referenz-Instanz erstellt, welches mit dem ComHandler mit der Middleware kommuniziert.
ServiceListener: Der ServiceListener wird vom ComHandler gestartet, wenn ein rebind aufgerufen wird, welches dann eine Schnittstelle für die Clients bietet.
SkeletonThread: Der SkeletonThread wird vom ServiceListener gestartet, welches die Anfragen von den Clients entgegennimmt, dieser verarbeitet und beantwortet.
Util: Util bietet Hilfsfunktionen an.

Kritische Abschnitte

In unserer Anwendung gibt es einen kritischen Abschnitt, welches sich im Nameservice auf der Middleware befindet. Hier besteht die Möglichkeit, dass mehrere Thread gleichzeitig die selbe Methode aufrufen und es somit zu einem Fehler kommen könnte. Um dies vorzubeugen sind die Methoden synchronized.

IDL-Compiler

Der von mir implementiere IDL-Compiler erstellt zwei Dateien. Einmal die Interface-Klasse, welches das Interface beinhaltet und einmal eine Stub-Klasse, welches die Kommunikation zwischen Client und Service realisiert.

Exceptions

MwareException: Wird beim resolve geworfen, wenn der angefragte Service nicht zu finden ist.