Team: 08, Dieter Pisarewski, Maxim Rjabenko

Aufgabenaufteilung:

Maxim Rjabenko und Dieter Pisarewski:
Gemeinsame Entwurf und Implementierung

Quellenangaben:

Base64 Coder: http://stackoverflow.com/questions/917604/decode-base64-string-java-5/917692#917692

Bearbeitungszeitraum:

28.11.2013 – 15.01.2013

Aktueller Stand:

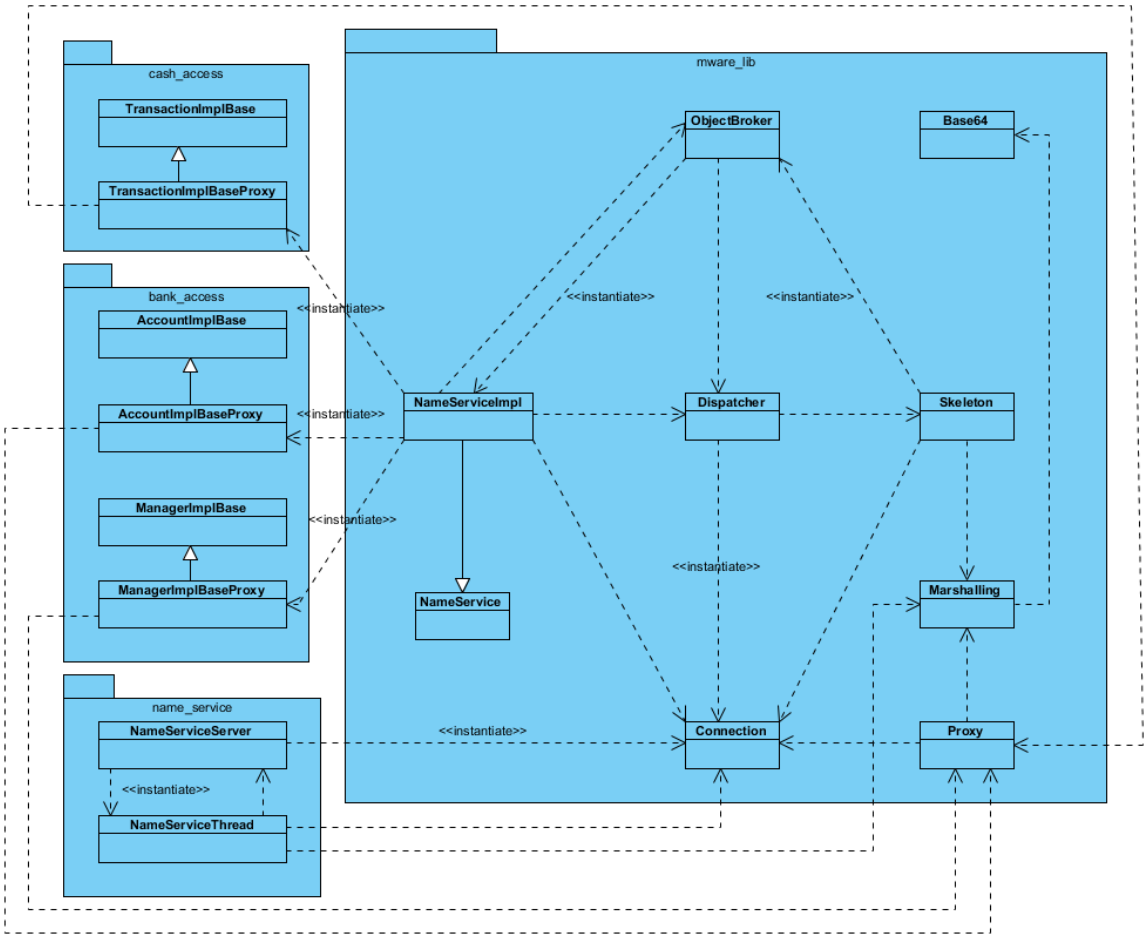
Implementiert und getestet mit GUI Applikationen und mit Testapplikationen.

Änderungen im Entwurf:

Beschreibung der Klassen vervollständigt.

Architektur:

Classendiagramm



Wir haben folgende Klassen implementiert:

- Proxys:
  - AccountImplBaseProxy
  - ManagerImplBaseProxy
  - TransactionImplBaseProxy
- Vordefinierte Classen:
  - AccountImplBase
  - ManagerImplBase
  - TransactionImplBase
  - ObjectBroker
  - NameService
- Eigene Classen:
  - NameServiceServer
  - NameServiceImpl
  - Dispatcher
  - Skeleton
  - Connection
  - Marshalling
  - Proxy

ObjectBroker ist als Singleton realisiert, der beim Aufruf der Methode **init** instanziiert wird. Er verwaltet eine Tabelle von lokalen Objekten. Wenn der **ObjectBroker** initialisiert wird, startet er auch einen **Dispatcher**.

NameServiceServer öffnet einen Server-Socket und wartet auf Anfragen. Wenn er eine Anfrage bekommt, startet er einen **NameServerThread**, der diese Anfrage abarbeitet. Außerdem verwaltet er eine Tabelle mit global registrierten Objekten.

NameServerThread bearbeitet Kommandos fürs Binden bzw. Auflösen von entfernten Objekten. Dabei registriert er die Objekte bei **NameServiceServer**.

NameServiceImpl sendet Anfragen zum Binden bzw. Auflösen eines entfernten Objekts an den NameServiceServer.

Beim Binden registriert **NameServiceImpl** ein lokales Objekt beim **ObjectBroker** und überträgt an den NameServiceServer einen Objektnamen, den Klassennamen des Objekts, eigenen Hostnamen und Port des lokalen Dispatchers.

Beim Auflösen erhält er die oben genannten Daten vom NameServiceServer und instanziiert durch Reflektion ein Proxyobjekt für entferntes Objekt. Damit das Objekt Anfragen an Server(Dispatcher) senden kann merkt er den Namen dieses Objektes, Host und Port, die ihm **NameServiceImpl** mitteilt.

Klassenennamen für Proxyobjekte müssen folgendem Muster entsprechen: <Klassenname>Proxy.

Proxy dient als Helper-Klasse für entfernte Methodenaufrufe und enthält deswegen nur statische Methoden. Sie wird von **AccountImplBaseProxy**, **ManagerImplBaseProxy** und **TransactiontImplBaseProxy** verwendet.

Sie implementiert Methode **invoke**, die eine Verbindung zum Host eines entfernten Objekts herstellt und Kommando „INVOKE“ mit dazugehörigen Namen des Objekts, Methodennamen und Methodenparametern sendet.

Danach liest Sie die Antwort vom Server, deserialisiert das Ergebnis und liefert es zurück.

AccountImplBaseProxy erbt von AccountImplBase, delegiert Methodenaufrufe an die Proxy Klasse.

ManagerImplBaseProxy erbt von ManagerImplBase, delegiert Methodenaufrufe an die Proxy Klasse.

TransactiontImplBaseProxy erbt von TransactiontImplBase, delegiert Methodenaufrufe an die Proxy Klasse.

Dispatcher ist ein Thread und ist als Singleton realisiert. Er öffnet einen Server-Socket und wartet auf Anfragen von Clients. Er nimmt entfernte Methodenaufrufe an und startet einen Skeleton für jeden Aufruf.

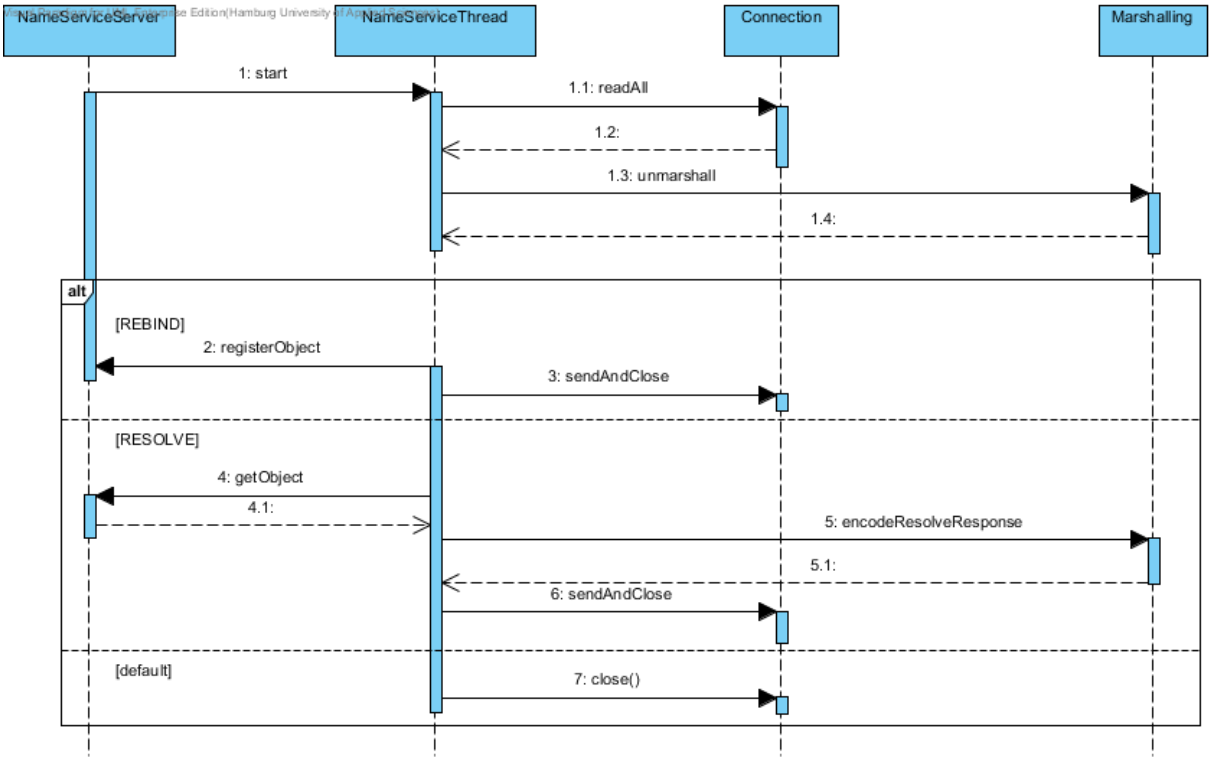
Skeleton ist ein Thread. Er verarbeitet einen entfernten Aufruf an einem lokalen Objekt. Dazu bekommt er vom Dispatcher eine **Connection**, aus der **Skeleton** einen Namen des Objekts, Methodennamen und übergebene Parameter ausliest, das lokales Objekt vom **ObjectBroker** bekommt und die Methode aufruft. Wenn ein Fehler beim Methodenaufruf passiert, sendet er diesen Fehler über vorhandene **Connection** zurück.

Connection implementiert Methoden für Netzwerkkommunikation.

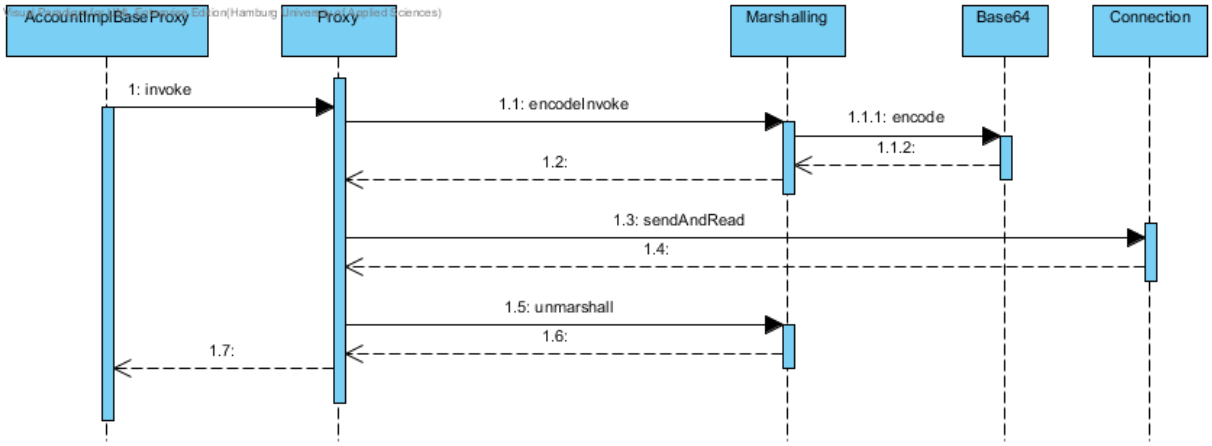
Marshalling dient als Helper-Klasse für Serialisierung und enthält deswegen nur statische Methoden. Für Marshalling benutzt sie Objektserialisierung von Java, wobei Binärdaten nach der Serialisierung als Text im Base64 Format kodiert werden.

Kommunikationsdiagramme

Ablauf beim Binden/Auflösen eines Objekts



Ablauf von einem Aufruf einer Methode an einem entfernten Objekt



Ablauf beim Ausführen eines entfernten Methodenaufrufs an einem lokalen Objekt