1. Dezember 2014

Steffen Giersch & Maria Lüdemann

BAI 5 Verteilte Systeme bei Herrn Professor Klauck

HAW Hamburg

Entwurf

Verteilte Systeme Praktikum 3

**Team:** 2 Steffen Giersch & Maria Lüdemann

**Aufgabenaufteilung:**

|  |  |
| --- | --- |
| Aufgabe | Teammitglied |
| Entwurfsplanung | Steffen Giersch |
| Entwurfsplanung | Maria Lüdemann |
| Implementation | Steffen Giersch |
| Implementation | Maria Lüdemann |

**Bearbeitungszeitraum:**

|  |  |  |
| --- | --- | --- |
| Datum | Dauer | Teammitglied |
| 17.11.2014 | 4 Stunden | Steffen |
| 20.11.2014 | 2 Stunden | Steffen & Maria |
| 21.11.2014 | 2 Stunden | Maria |
| 25.11.2014 | 6,5 Stunden | Steffen & Maria |
| 26.11.2014 | 7 Stunden | Steffen |
| 27.11.2014 | 2 Stunden | Steffen & Maria |
| 28.11.2014 | 1 Stunde | Maria |
| 29.11.2014 | 4,5 Stunden | Steffen & Maria |
| 1.12.2014 | 3 Stunden | Steffen & Maria |

**Gesamt: 33 Stunden**

**Quellen**: DieSocketConnection ist in Anlehnung an den Code von der Gruppe Birger Kampf & Fabian Sawatzki entstanden.

**Aktueller Stand:**

* + - Entwurfsdokument fertig samt Nachtrag
    - Implementation in Arbeit

# Inhaltsverzeichnis

[1 Inhaltsverzeichnis 2](#_Toc405228702)

[2 Funktionalität 3](#_Toc405228703)

[3 Komponenten 3](#_Toc405228704)

[3.1 Bank\_Access 3](#_Toc405228705)

[3.2 Cash\_Access 4](#_Toc405228706)

[3.3 mware\_Lib 4](#_Toc405228707)

[3.4 Globaler Name Service 5](#_Toc405228708)

[4 Fehler Behandlung 6](#_Toc405228709)

[4.1 Exceptions 6](#_Toc405228710)

[5 Abblauf - Sequenzdiagramm 6](#_Toc405228711)

[6 Kommunikations Übersicht 6](#_Toc405228712)

[7 Klassendiagramm 6](#_Toc405228713)

# Funktionalität

Es soll eine objektorientiere Middleware konstruiert werden, die entfernte Methodenaufrufe ermöglicht und dabei auch nebenläufige Aufrufe unterstützt. Dabei wird ein globaler Namensdienst genutzt der die Objektnamen zu Objektreferenzen übernimmt. Außerdem werden Schnittstellen für die externen Anwendungen bank\_acess und cash\_acess angeboten.

Die Middleware wurde primär in vier Pakete aufgeteilt.

1. **bank\_access** bietet die Stubs für Objekte der bank\_access Anwendung.
2. **cash\_access** bietet die Stubs für Objekte der bank\_access Anwendung.
3. **mware\_lib,** beinhaltet alle Klassen die zum Betrieb der Middleware notwendig sind.
4. **name\_service**, das den Namensdienst bereit stellt, das auf einem separaten Rechner liegt und die Namensanfragen auflöst

Genauere Definition der Pakete im nächsten Kapitel.

# Komponenten

## Bank\_Access

Beinhaltet:

* AcountImplBase
* AccountImplementation
* ManagerImplBase
* ManagerImplementation
* InvalidParamException
* Overdraftexception

Das Package cash\_Access implementiert den Stub für die Account und Manager Funktionalität der Bank dabei beinhaltet es jeweils ein Interface und eine Implementation der Klasse sowie die beiden Fehlermeldungen die geworfen werden können sollen.

Dabei ist das prinzipielle Vorgehen, dass der Stub die Parameter übernimmt und in eine Liste verpackt um mit dem Namen des Objekts, dem Funktionsnamen und der Liste der Argumenten mittels einer SocketConnection an einen ObjectBrokerDispatcher der Instanz die das eigentliche Objekt verwaltet zu senden der den Aufruf verteilt.

Die gebotenen Funktionen definieren sich in der abstrakten Klasse und sind:

**ManagerImplBase:**

transfer(double amount) **throws** OverdraftException :: Double -> void

public double getBalance() :: void -> double

**Account:**

String createAccount(String owner, String branch) throws InvalidParamException) :: String x String -> String

## Cash\_Access

* TransactionImplBase
* TransactionImplementation
* InvalidParamException
* Overdraftexception

Das Package Cash\_Access implementiert den Stub für die Transaktionsfunktionalität und beinhaltet somit das Interface und eine Stubimplementierung für das Objekt sowie die beiden Exceptions die geworfen werden können sollen.

Dabei ist das prinzipielle Vorgehen, dass der Stub die Parameter übernimmt und in eine Liste verpackt um mit dem Namen des Objekts, dem Funktionsnamen und der Liste der Argumenten mittels einer SocketConnection an einen ObjectBrokerDispatcher der Instanz die das eigentliche Objekt verwaltet zu senden der den Aufruf verteilt.

Die gebotenen Funktionen definieren sich in der abstrakten Klasse und sind:

**Transaction:**

public void deposit(String accountID, double amount)

public void withdraw(String accountID, double amount)

public double getBalance(String accountID) throws InvalidParamException

## mware\_Lib

Das Package mware\_lib implementiert die eigentliche Middleware und stellt die dafür notwendigen Funktionalitäten. Sie setzt sich zusammen aus:

* ObjectBroker
* ObjecktBrokerDispatcher
* NameService
* NameServiceImplementation
* Message
* MessageCall
* ObjectReference
* SocketConnection
* Log
* Constants

**Messages:**

Nachrichten verschicken wir als Objekte, dafür gibt es eine Klasse Messages und eine Implementation für jede Art von Nachricht die versendet werden kann z.B MessageCall.

**SocketConnection:**

Implementiert einige Funktionen für das Auf und Abbauen von Socket Verbindungen und das lesen und senden von Objekten sowie Message Objekten.

**Log:**

Eine Implementation eines Logs mit dem wir die Arbeitsweise des Systems loggen wollen.

**Constants:**

Constants beinhaltet einige Konstanten die in der Implementation der Messages benötigt werden sowie den Port auf dem der Nameservice läuft.

**ObjektBroker:**

In der Middleware ist das zentrale Stück der ObjectBroker, er startet und beendet die Middleware, kontrolliert den Nachrichten Verkehr und den lokalen NameService und hält eine Liste der bekannten Objekte.

**ObjektBorkerDispatcher:**

Nimmt Anfragen für Objekte an die in seiner Instanz liegen und beantwortet diese, bzw. leitet sie weiter. Dabei wird für jede Nachricht ein Thread gestartet um Nebenläufigkeiten zu gewährleisten.

## Globaler Name Service

Der NameService kann auf einem separaten Rechner laufen und verwaltet die Verbindungen zwischen Objektnamen und Objektreferenzen. Dabei bietet die offene Schnittstelle die Funktionen :

rebind(servant, name) :: Object x String -> void

**servant** -> Ist das Objekt das angemeldet werden soll

**name** -> Ist der Name unter dem das Objekt zu finden sein soll.

Die Funktion nimmt ein Objekt und den dazugehörigen Namen und speichert dies als Referenz in der eigens dafür gehaltenen HashMap ab.

resolve(name) :: String -> Object

**name** -> Ist der Objekt Name der aufgelöst werden soll

Die Funktion nimmt den Namen entgegen und versucht ihn mittels der intern gehaltenen HashMap aufzulösen und die Referenz auf das gesuchte Objekt zurück zu geben.

Um auch nebenläufige Anfragen zu ermöglichen startet der Name Service für Anfragen eigene Threads die dann die jeweiligen Anfragen abhandeln.

# Fehler Behandlung

## Exceptions

Die Anwendungsspezifischen Exceptions werden, sollten sie geworfen werden, so an die jeweiligen Stubs (cash\_acess und bank\_acess) weitergegeben von ihnen erkannt und ausgeworfen.

Das System wird des Weiteren geloggt um eine Fehlerbehandlung zu ermöglichen und die Funktion einfacher nach zu vollziehen. Dabei erzeugt jede Komponente der Middleware (mware\_lib) eine Log Datei in der die Logging Nachrichten gespeichert werden.

# Abblauf - Sequenzdiagramm

Siehe Anhang 1

# Kommunikations Übersicht

Siehe Anhang 2

Siehe Anhang 3 – Paket\_Kommunikationsdiagramm zeigt auf, wie die Kommunikation zwischen den Paketen definiert ist und welchem Protokol sie folgen.

# Klassendiagramm

Das Klassendiagramm zeigt die wichtigsten Klassen und ihre Abhänigkeiten, um zu verhindern dass das Diagramm zu unübersichtlich wird haben wir die Klassen ohne Abhänigkeiten weg gelassen.

