1. DEZEMBER 2014

ENTWURF

VERTEILTE SYSTEME PRAKTIKUM 3

STEFFEN GIERSCH & MARIA LÜDEMANN
BAI 5 VERTEILTE SYSTEME BEI HERRN PROFESSOR KLAUCK
HAW Hamburg

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: Die SocketConnection ist in Anlehnung an den Code von der Gruppe Birger Kampf & Fabian Sawatzki entstanden.

Aktueller Stand:

- Entwurfsdokument fertig samt Nachtrag
- Implementation in Arbeit

1 INHALTSVERZEICHNIS

2 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.

3 KOMPONENTEN

3.1 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

3.2 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

3.3 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.

3.4 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.

4 Fehler Behandlung

4.1 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.

5 Abblauf - Sequenzdiagramm

Siehe Anhang 1

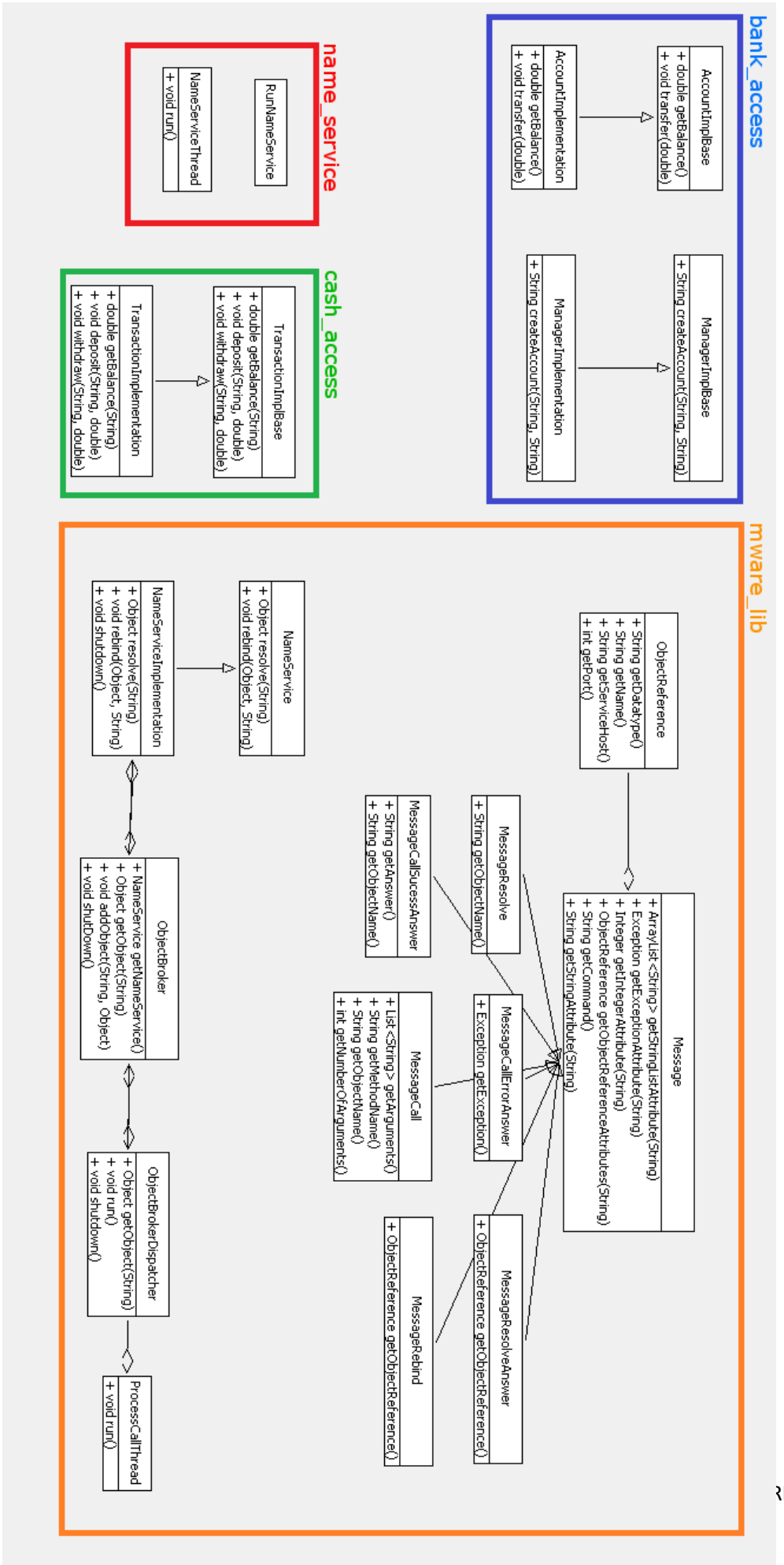
6 Kommunikations Übersicht

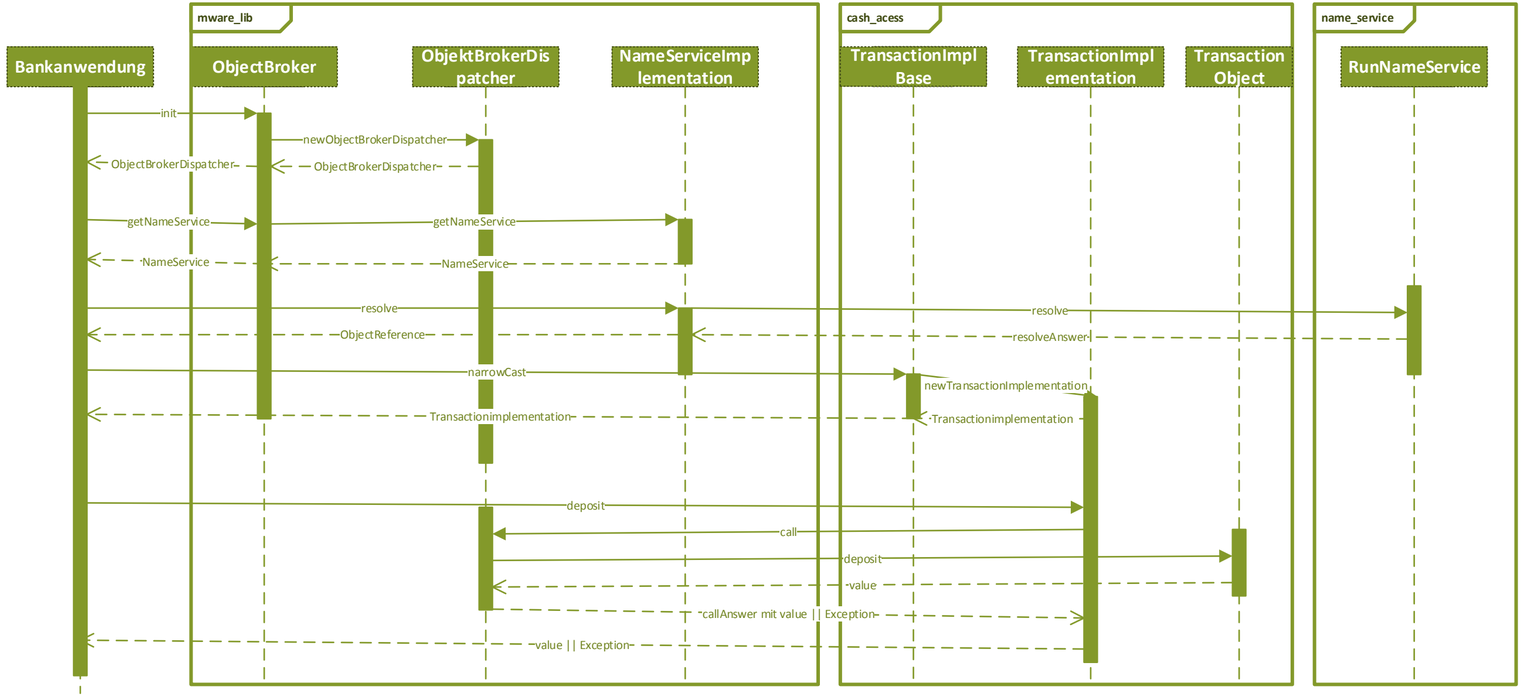
Siehe Anhang 2

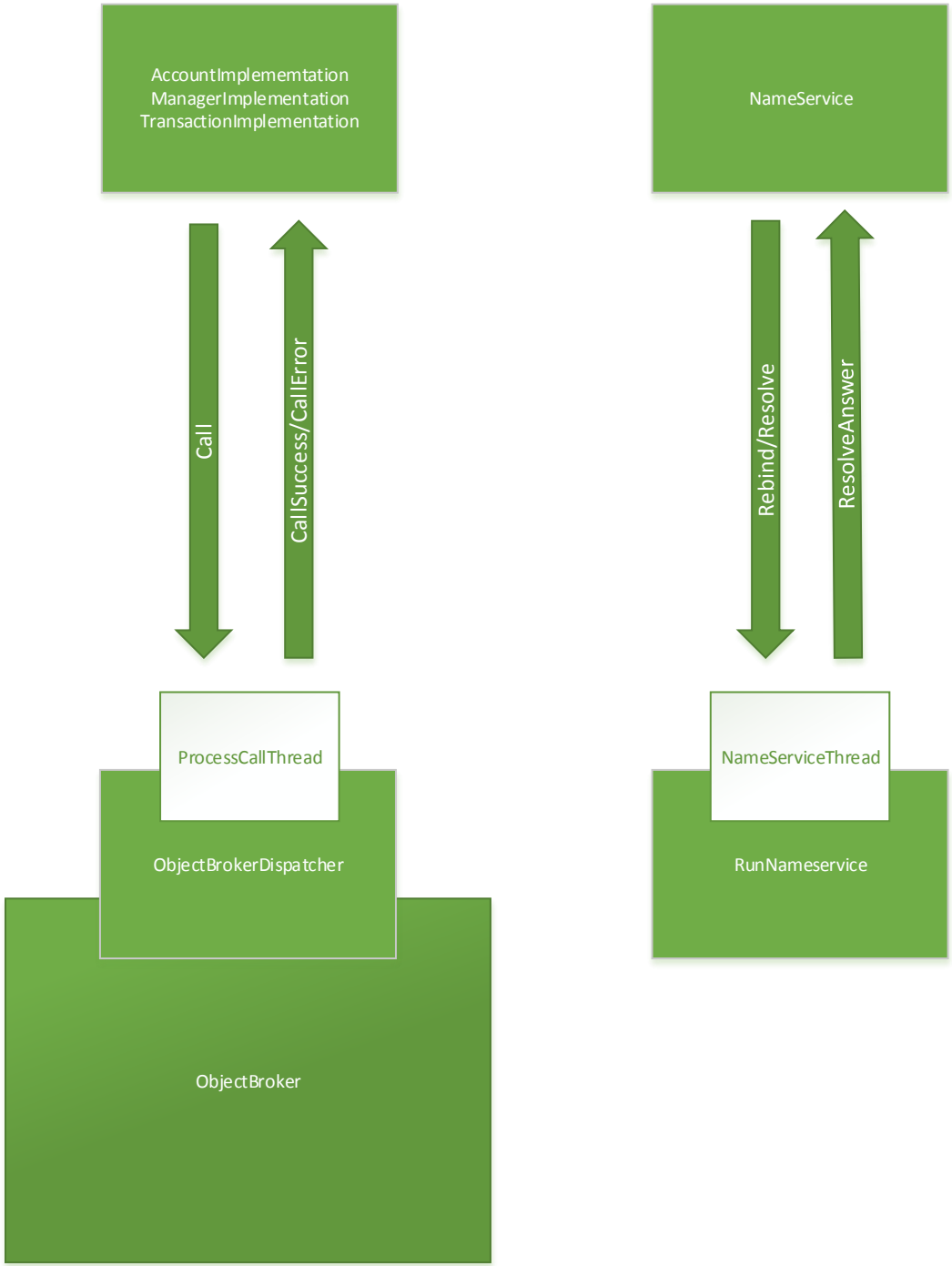
Siehe Anhang 3 – Paket_Kommunikationsdiagramm zeigt auf, wie die Kommunikation zwischen den Paketen definiert ist und welchem Protokol sie folgen.

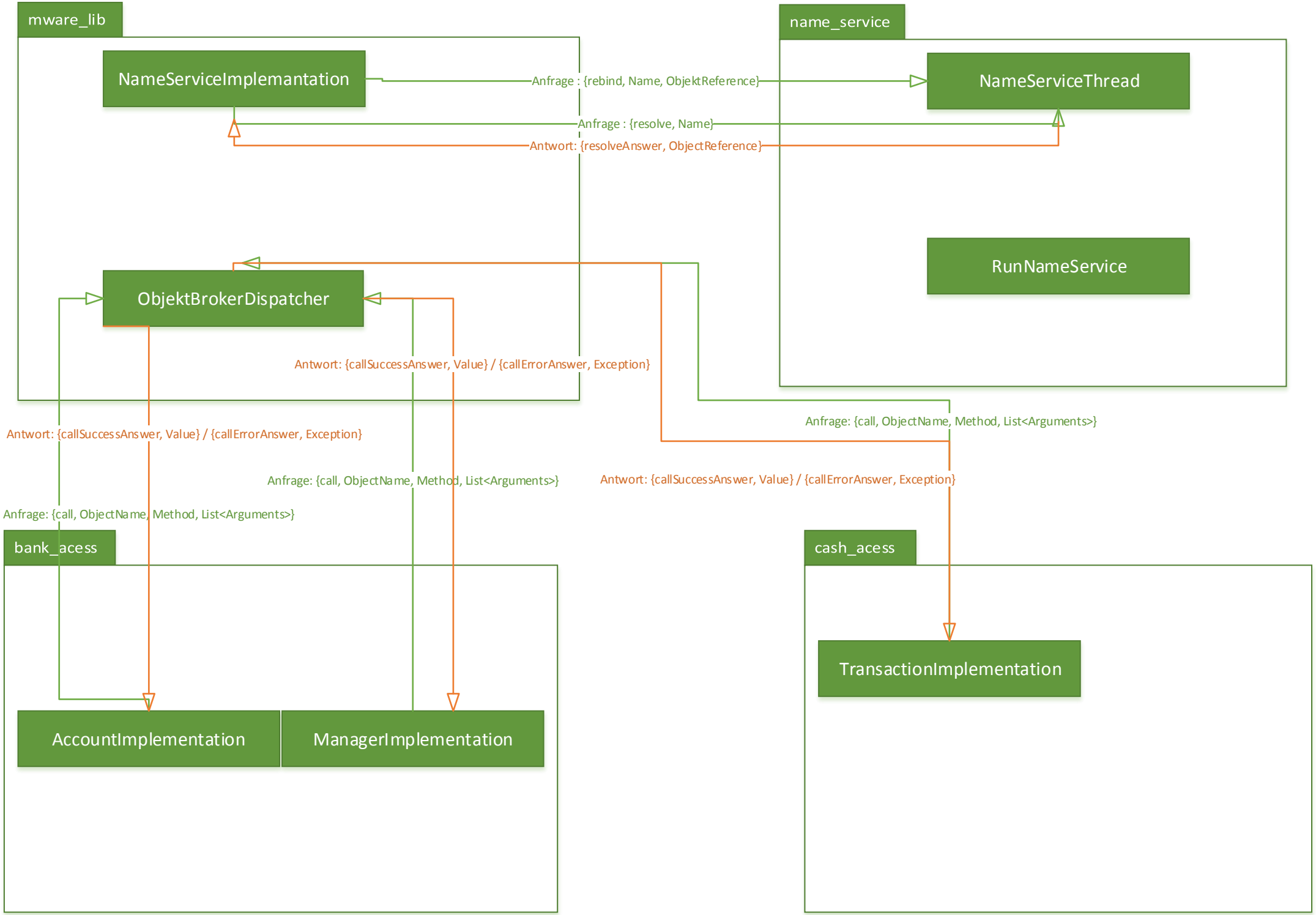
7 Klassendiagramm

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.


bank_access
AccountImplBase
+ double getBalance()
+ void transfer(double)
AccountImplementation
+ double getBalance()
+ void transfer(double)
ManagerImplBase
+ String createAccount(String, String)
ManagerImplementation
+ String createAccount(String, String)
name_service
RunNameService
NameServiceThread
+ void run()
cash_access
TransactionImplBase
+ double getBalance(String)
+ void deposit(String, double)
+ void withdraw(String, double)
TransactionImplementation
+ double getBalance(String)
+ void deposit(String, double)
+ void withdraw(String, double)
mware_lib
ObjectReference
+ String getDatatype()
+ String getName()
+ String getServiceHost()
+ int getPort()
Message
+ ArrayList <String> getStringListAttribute(String)
+ Exception getExceptionAttribute(String)
+ Integer getIntegerAttribute(String)
+ ObjectReference getObjectReferenceAttributes(String)
+ String getCommand()
+ String getStringAttribute(String)
MessageResolve
+ String getObjectName()
MessageCallSucessAnswer
+ String getAnswer()
+ String getObjectName()
MessageCallErrorAnswer
+ Exception getException()
MessageCall
+ List <String> getArguments()
+ String getMethodName()
+ String getObjectName()
+ int getNumberOfArguments()
MessageResolveAnswer
+ ObjectReference getObjectReference()
MessageRebind
+ ObjectReference getObjectReference()
NameService
+ Object resolve(String)
+ void rebind(Object, String)
NameServiceImplementation
+ Object resolve(String)
+ void rebind(Object, String)
+ void shutdown()
ObjectBroker
+ NameService getNameService()
+ Object getObject(String)
+ void addObject(String, Object)
+ void shutDown()
ObjectBrokerDispatcher
+ Object getObject(String)
+ void run()
+ void shutdown()
ProcessCallThread
+ void run()


mware_lib
cash_acess
name_service
Bankanwendung
ObjectBroker
ObjektBrokerDispatcher
NameServiceImplementation
TransactionImplBase
TransactionImplementation
TransactionObject
RunNameService
init
newObjectBrokerDispatcher
ObjectBrokerDispatcher
ObjectBrokerDispatcher
getNameService
getNameService
NameService
NameService
resolve
resolve
ObjectReference
resolveAnswer
narrowCast
newTransactionImplementation
Transactionimplementation
Transactionimplementation
deposit
call
deposit
value
callAnswer mit value || Exception
value || Exception


AccountImplememtation
ManagerImplementation
TransactionImplementation
Call
CallSuccess/CallError
ProcessCallThread
ObjectBrokerDispatcher
ObjectBroker
NameService
Rebind/Resolve
ResolveAnswer
NameServiceThread
RunNameservice


mware_lib
name_service
NameServiceImplemantation
NameServiceThread
Anfrage : {rebind, Name, ObjektReference}
Anfrage : {resolve, Name}
Antwort: {resolveAnswer, ObjectReference}
RunNameService
ObjektBrokerDispatcher
Antwort: {callSuccessAnswer, Value} / {callErrorAnswer, Exception}
Anfrage: {call, ObjectName, Method, List<Arguments>}
Antwort: {callSuccessAnswer, Value} / {callErrorAnswer, Exception}
Anfrage: {call, ObjectName, Method, List<Arguments>}
Antwort: {callSuccessAnswer, Value} / {callErrorAnswer, Exception}
Anfrage: {call, ObjectName, Method, List<Arguments>}
bank_acess
cash_acess
TransactionImplementation
AccountImplementation
ManagerImplementation