****

**FH - Studiengang für**

**Informationstechnik und System-Management**

**Salzburg**

**ITS**

**Übungen in**

**Spezielle Softwaretechnologien**

**Protokoll**

Gegenstand der Übung gemäß Anleitung:

**Softwarekomponenten – Netzwerkkommunikation**

**Version: 1**

**Datum der Übung: 3.11.2016**

**Datum der Abgabe: 22.11.2016**

**Autoren: Christopher Wieland, Martin Wieser, Stephanie Kaschnitz**

**Unterschrift des Autors / der Autorin:**

Historie

|  |  |  |  |
| --- | --- | --- | --- |
| Änderung | Datum: | Autor: | Version: |
| Labor 1: Bank.dll | 18.10.16 | SK, CW, MW | 1.0 |
| Labor 2: Anpassung der native-c DLL | 27.10.16 | MW, CW | 1.1 |
| Labor 2: Konsolenapplikation | 29.1016 | SK | 1.2 |
| Labor 2: C# Assembly | 1.11.16 | CW, SK | 2.0 |
| Labor 3: Anpassung der native-c DLL (Kunden/Konten) | 17.11.16 | MW | 2.1 |
| Labor 3: WPF Applikation | 17.11.16 | CW | 2.2 |
| Labor 3: C# Applikation - Netzwerkkommunikation mit RabbitMQ implementiert & getestet | 21.11.16 | SK, MW | 3.0 |
|  |  |  |  |

Inhaltsverzeichnis

[1 Aufgabenstellung 3](#_Toc467612441)

[1.1 Labor 1 - Dynamic Link Libraries 3](#_Toc467612442)

[1.2 Labor 2 - Assemblies und Wrapper 4](#_Toc467612443)

[1.3 Labor 3 – Netzwerkkommunikation 5](#_Toc467612444)

[2 Komponentenübersicht 6](#_Toc467612445)

[3 Bankanwendung (GUI) 7](#_Toc467612446)

[4 C# Assembly mit Netzwerkanbindung 7](#_Toc467612447)

[4.1 BankManagement 7](#_Toc467612448)

[4.2 Netzwerkanbindung 8](#_Toc467612449)

[4.2.1 RabbitMQ 8](#_Toc467612450)

[4.2.2 Protokoll 8](#_Toc467612451)

[4.3 Schnittstelle 8](#_Toc467612452)

[4.3.1 IBankManagement 8](#_Toc467612453)

[5 Bank.dll 8](#_Toc467612454)

[5.1 Schnittstelle 9](#_Toc467612455)

[5.1.1 Customer 9](#_Toc467612456)

[5.1.2 Kreditkonto 9](#_Toc467612457)

[5.1.3 Sparkonto 9](#_Toc467612458)

[5.1.4 Bank & Konten 9](#_Toc467612459)

[5.1.5 Überweisung 9](#_Toc467612460)

[5.1.6 Währungsmodul 9](#_Toc467612461)

[6 Fazit des Projektes der 3 Laboreinheiten 9](#_Toc467612462)

[6.1 Implementierungsstufen: 9](#_Toc467612463)

[6.2 Probleme und gewonnene Erfahrungen: 9](#_Toc467612464)

# Aufgabenstellung

Dieses Protokoll ist das Abschlussprotokoll von den ersten drei Laboreinheiten in Spezielle Software-Technologien. Bei diesem Laborprojekt ging es um das Abbilden der Funktionalitäten einer Bank, in austauschbare Komponenten. Mit diesen Komponenten wurde dann eine einfache Bankapplikation realisiert welche mit den Banken der anderen Gruppen über das Netzwerk kommunizieren, und so Überweisungen tätigen kann. Dies wurde in drei Stufen erledigt.

In den folgenden Kapiteln sind die 3 Laboraufgaben genau beschrieben.

Labor 1 - Dynamic Link Libraries

Die Aufgabenstellung bestand darin, grundlegende Funktionen einer Bank mithilfe von nachladbaren Dynamic Link Libraries (DLL) in C/C++ zu entwickeln.

Die Anzahl der Komponenten sowie die Umsetzung sind selbst zu gestalten.

Um die Interoperabilität mit verschiedenen Programmiersprachen sicherzustellen, sind die Komponenten in C zu entwerfen.

D.h. alle Funktionen der DLLs müssen als C-Funktionen aus der DLL exportiert werden, um sie aus beliebigen Programmiersprachen aufrufen zu können.

Die folgenden Basisfunktionalitäten sollten beinhaltet sein:

* Kunden anlegen, löschen, umbenennen, ändern (Adressdaten, ...)
* Konten für Spar- und Kreditgeschäfte anlegen, schließen, verwalten
  + Ein Kunde kann mehrere Konten haben
  + Ein Konto kann mehrere Kontoverfüger haben
* Überweisungen von einem Konto zum anderen, Abhebung/Einzahlung, Kontoauszüge, Kontoabschlüsse
* Währungsmodul (Umrechnung, Kursverwaltung, etc.)
* Persistente Datenhaltung

Sonstige Basis-/Hilfsfunktionen nach eigenem Ermessen

Labor 2 - Assemblies und Wrapper

Entwerfen Sie nun auf Basis des/der von einem anderen Team übernommen DLLs und

der eigenen DLLs ein gemeinsames C#/.NET Interface und darauf basierend .NET Assemblies, die – diesmal objektorientiert unter C# und .NET – dem eigenen DLL und

dem übernommen DLL als Wrapper/Adapter/Bridge dienen. Sie sollten die volle bzw.

geforderte Funktionalität der DLLs anbieten, ohne jedoch Details der DLLs und deren

API kennen zu müssen.

Letztlich sollten Sie mehrere .NET Assemblies implementieren, welche eine möglichst

einfache Verwendung der DLLs in C#/.NET Anwendungen und – da wir ja Komponenten

haben – den Austausch der verwendeten DLLs erlaubt. Das Ziel ist, dass Ihre zukünftige

Applikation durch den Austausch von Assemblies und der DLLs sowohl mit den eigenen

DLLs, als auch mit den DLLs des anderen Teams arbeiten kann (nicht gleichzeitig), da die

Schnittstellen beider Assembly-Gruppen in beiden Fällen ja gleich sind.

Einschränkung: Ein dynamisches Laden/Entladen der Assemblies bzw. DLLs während der

Runtime wird nicht benötigt.

Die Interfaces, Klassen, Typen,.. und das Fehlerhandling sollte so ausgelegt sein, dass die

DLLs für die Anwender völlig transparent und ohne Detailwissen über die DLLs

angebunden werden können. Aus der Nutzung der Assemblies darf in keiner Art und

Weise ersichtlich werden, dass es sich um native Dynamic Link Libraries handelt, die

lediglich ein primitives C-API anbieten.

Labor 3 – Netzwerkkommunikation

Ziel ist es nun, dass Sie Ihre Bankgeschäfte und Überweisungen auch bankübergreifend tätigen können. Dadurch sollte es möglich werden, Überweisungen und Abbuchungen auch von einem Konto einer Bank zu einem Konto einer beliebigen andren Bank durchzuführen zu können. Hierfür muss jede Bank mit einer anderen Bank (alle Banken der Lehrveranstaltung) kommunizieren und Daten austauschen können.

Führen sie Bankkennung (BankID’s ähnlich einer BIC oder Bankleitzahl; bitte nicht die IP Adresse der Bank dafür verwenden!!!) ein, die eine netzwerkübergreifende, eindeutige Identifikation einer (Bank-) Geschäftsstelle ermöglichen. Falsche IDs sollten frühzeitig erkannt und entsprechend behandelt werden.

Diskutieren Sie mit Ihren Kollegen: Überlegen Sie sich, wie sich die Banken im Netzwerk möglichst ohne manuellen Eingriff automatisch finden können und in welcher Form (Protokoll) Sie die Daten untereinander austauschen.

1. Die Kommunikation über das Netzwerk mit den Fremdbanken übernehmen wieder eine oder mehrere Komponenten. Ihr System sollte auf maximale Stabilität ausgelegt werden.

Geld darf keinesfalls und auch nicht im Fehlerfall „verschwinden“.

Testen Sie Ihre Assemblies wieder ausgiebig mit eigens dafür erstellten Testprogrammen und/oder UnitTests.

2. Erweitern Sie nun Ihre .NET Applikation und die Benutzerschnittstelle so ab, dass netzübergreifende Überweisungen und Abbuchungen möglich sind. Diese Applikation sollte für den Endanwender folgende zusätzliche Optionen anbieten:

* Überweisungen von einem Konto zu einem anderen (in Fremd und Eigenwährungen)
* Verwaltung und Verarbeitung von Abbuchungsaufträgen
* Hilfsfunktionen wie z.B. Abfrage aller im Netzwerk erreichbaren/verfügbaren Banken

# Komponentenübersicht

Die folgende Abbildung stellt die Komponentenübersicht dar. Diese wird unten kurz beschrieben.

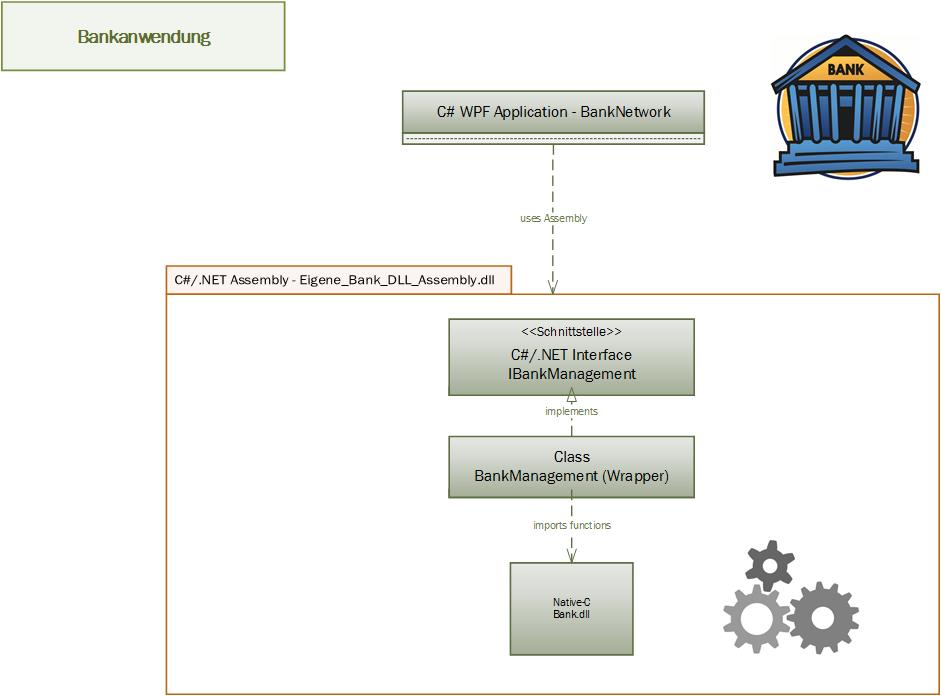


Abbildung 1: Bankanwendung – Komponentendiagramm

Es wurde die WPF Applikation BankNetwork entwickelt die die Komponente Eigene\_Bank\_DLL\_Assembly.dll verwendet. Da die BankManagement Klasse ein Wrapper von der Native-C Bank.dll ist, können die Funktionalitäten ab dieser Komponente Objektorientiert genutzt werden. Zudem wurde die Assembly noch für die finale Version angepasst und für eine Netzwerkkommunikation zum RabbitMQ Server fit gemacht. Dadurch kann man nun zwischen den Banken (mit einem von unserer Gesamten Laborgruppe definierten Protokoll) Überweisungen tätigen. Das Verwalten der Konten und Kunden mitsamt der Persistierung wurde in der Bank.dll realisiert.

# Bankanwendung (GUI)

Es wurde eine einfache intuitive Bankanwendung entwickelt mit der man einfache Bankgeschäfte abwickeln kann.

Hauptfunktionen sind:

1. Verwaltung der Kunden / Konten
2. Überweisungen zu eigenen und zu Konten von anderen Banken
3. Bankauszug

Die Bankanwendung hat als Abhängigkeit die genutzte C#/.NET Assembly, die wiederum als Abhängigkeit die Bank.dll von der ersten Einheit hat.

# C# Assembly mit Netzwerkanbindung

Hier sind die Imports vom der bank.dll enthalten, d.h. die Eigene-Bank\_Dll\_Assembly.dll ist ein Wrapper vom Bank.dll. Das Assembly implementiert das Interface IBankManagement. Die Bankfunktionalitäten lassen sich mit dieser Komponente Objektorientiert verwenden. Da in der Eigene\_Bank\_DLL\_Assembly.dll ein relativer Pfad auf die Bank.dll verweist muss die Bank.dll sich im Pfad der exe befinden.

BankManagement

Die Klasse BankManagement importiert die Funktionen aus der Bank.dll. Zudem Implementiert sie die Methoden vom Interface IBankManagement die die externen Funktionen dann verwenden.

Das folgende Beispiel zeigt den Import der Funktionen NeuesSparkonto und Sparkontoentfernen.

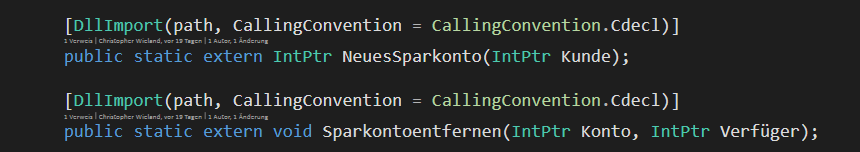


Abbildung 2: Beispielimport von Funktionen aus der Bank.dll

Netzwerkanbindung

Für die Netzwerkanbindung der Banken, mussten einige Themen besprochen werden. Die Wichtigsten werden in den folgenden drei Kapiteln kurz beschrieben.

RabbitMQ

Es wurde von Stefan Binna ein RabbitMQ Server aufgesetzt der Nachrichten entgegennehmen und in die Queue für einen Empfänger einfügen kann. Durch den Parameter „routingKey“ wird die Nachricht an eine bestimmte Bank weitergeleitet. Der routingKey war in unserem Falle der für jede Gruppe definierte BIC. Die Nachrichten konnten dann von der Queue Abgeholt und verarbeitet werden.

Protokoll

Um die Nachrichten zu vereinheitlichen wurde die Transaction Klasse erstellt die die benötigten Daten verarbeitet und diese JSON-Serialisiert.

**Transaction Klasse**

Die Transaction Klasse beinhaltet die Daten für die Überweisungen. Diese werden dann zum Senden an den RabbitMQ Server,– JSON-Serialisiert und übertragen.

Beim Empfang einer Nachricht von einer anderen Bank oder einer Antwort (durch Abholen vom RabbitMQ Server) wird der JSON-String wieder deserialisiert und als Transaction Objekt verarbeitet um Überweisungen zu ermöglichen.

Schnittstelle

Das Interface IBankManagement wird von BankManagement implementiert und definiert die Methoden der Klasse. Im zweiten Teil der Laborübung war dies speziell für die Austauschbarkeit wichtig, somit konnten zwei Austauschbare Komponenten geschaffen werden.

IBankManagement

**Customer Managements**

**int createCustomer(String \_Vorname, String \_Nachname, String \_Geburtsdatum, String \_adresse, String \_Wohnort, String \_Telefon);**

Mit createCustomer wird ein neuer Kunde angelegt

**void changeCustomer(int \_id, String \_Vorname, String \_Nachname, String \_adresse, String \_Wohnort, String \_Telefon);**

Diese Funktion ändert die Daten des übergebenen Kunden

**int getCustomer(String \_firstName, String \_lastName, String \_birthDate);**

Mit getCustomer wird mit Suchparametern der Kunde aus der Datenbank gesucht und dessen ID zurückgegeben.

**void deleteCustomer(int \_id);**

DeleteCustomer löscht einen Kunden.

**Account Management**

**string createSavingsAccount(int \_id);**

Erstellt ein Sparkonto mit der ID vom Customer

**string createCreditAccount(int \_id);**

Erstellt ein Kreditkonto mit der ID vom Customer

**void deleteSavingsAccount(String \_sNumber, int \_id);**

Löscht das Sparkonto vom User

**void deleteCreditAccount(String \_cNumber, int \_id);**

Löscht das Kreditkonto vom User

**void depositCreditAcc(String \_cNumber, String \_usage, double \_amount);**

Fügt dem KreditKonto Geld hinzu

**void depositSavingsAcc(String \_sNumber, String \_usage, double \_amount);**

Fügt dem SparKonto Geld hinzu

**void withdrawCreditAcc(String \_cNumber, double \_amount);**

Hebt Geld vom KreditKonto ab

**void transfer(String \_cNumber, String \_toAccNumber, String \_usage, double \_amount);**

Hier kann eine Überweisung mit einem Kreditkonto getätigt werden

**void addSavingsAccountUser(String \_sNumber, int \_id);**

Hier kann ein Verfüger dem SparKonto hinzugefügt werden

**void addCreditAccountUser(String \_cNumber, int \_id);**

Hier kann ein Verfüger dem KreditKonto hinzugefügt werden

**string getBankAccountNumbers(int \_id);**

Mit dieser Funktion bekommt man die Kontonummer eines Übergebenen Kontos

**int getAccType(String Accnumber);**

Gibt den Typen des Kontos mit der Kontonummer zurück

**double getDepositkontostand(String snumber);**

Holt den Kontostand eines Sparkontos

**double getCreditkontostand(String snumber);**

Holt den Kontostand eines Kreditkontos

**int deleteCreditAccUser(String Kontonummer, int additionalUser);**

Löscht einen Usereintrag von einem Kreditkonto

**int deleteSavingsAccUser(String Kontonummer, int additionalUser);**

Löscht einen Usereintrag von einem Sparkonto

**void withdrawSavingsAcc(String \_snumber, double \_amount);**

Hebt vom Sparkonto ab

**Währungsmodul und Kontoauszug**

**void createBankStatement(String \_accNumber);**

Erstellt einen Kontoauszug.

**void convertMoney(String \_cNumber, string \_currency);**

Rechnet das Geld in eine andere Währung um.

**void showChangeOfCourse(String \_cNumber);**

Zeigt die momentane Währung an.

**double currencyConversion(int \_currency, double \_value);**

Rechnet die Währung um.

**BIC**

**int getBankBIC();**

Gibt den BIC der Bank zurück.

**Netzwerkkommunikation**

**void send(Transaction transaction);**

Sendet eine Transaktion

**void receive();**

Empfängt Nachrichten aus der Queue

# Bank.dll

In der bank.dll Komponente werden die die Grundfunktionalitäten einer Bank abgebildet.

Zudem werden die Daten der Kunden/Konten hier im JSON Format Persistiert. Als externe Libary dafür wurde cJSON verwendet.

Schnittstelle

Es werden die folgenden Schnittstellenfunktionen bereitgestellt.

Customer

**CUSTOMER\* NeuerKunde(char\* \_Vorname,**

**char\* \_Nachname, char\* \_Geburtsdatum,**

**char\* \_adresse, char\* \_Wohnort, char\* \_Telefon);**

Einen neuen Kunden anlegen

**void Kundenvornamenänderung(CUSTOMER \*Kunde, char\* Vorname);**

Kunden ändern

**void Kundennachnamenänderung(CUSTOMER \*Kunde, char\* \_Nachname);// Customeränderung**

Kunden ändern

**void Kundenadressänderung(CUSTOMER \*Kunde, char\* \_Adresse);**

Kunden ändern

**void Kundenwohnortsänderung(CUSTOMER \*Kunde, char\* \_Wohnort);**

Kunden ändern

**void Kundentelefonänderung(CUSTOMER \*Kunde, char\* \_Telefon); // Customeränderung**

Kunden ändern

**void Kundendatenabfrage(CUSTOMER \* Kunde);**

Hilfsfunktion - Output ins Logging file

**void Kundeentfernen(CUSTOMER\* Kunde);**

Entfernt den Kunden

**int searchUser(char\* vorname, char\* nachname, char\* geb);**

Rückgabe der Userid

**CUSTOMER\* readUser(int id);**

Rückgabe eines Users mit der ID

**int UserKontoExists(int userid, char\* ktnr);**

Gibt es das Konto dieses Users?

**int rmvKontofromUser(int userid, char\* ktnr);**

Löscht einen Kontoeintrag von einem User

**int addKontotoUser(int userid, char\* ktnr);**

Fügt einen Kontoeintrag einem User hinzu

**char\* getUserKonten(int id);**

Gibt das JSON Array mit den Konten des Users Zurück

**int getUserId(CUSTOMER\* cust);**

Gibt die UserId des Customers zurück

Kreditkonto

**KREDITKONTO\* NeuesKreditkonto(CUSTOMER\* Kunde);**

Neues Kreditkonto anlegen

**KREDITKONTO\* readKreditKonto(char\* ktnr);**

Lesen und Rückgabe der Kreditkontostruktur

**void doAbheben(KREDITKONTO\* zielkonto, double betrag);**

Einen Betrag vom Kreditkonto abheben

**void Kreditkontoentfernen(KREDITKONTO\* Konto);**

Ein Kreditkonto entfernen

**void doEinzahlen(KREDITKONTO\* zielkonto,**

**char\* verwendungszweck, double betrag);**

Zahlt auf das Konto den gewünschten Betrag ein

**char\* getKreditKontonummer(KREDITKONTO\* konto);**

Holt die Kreditkontonummer

**double getKreditkontostand(KREDITKONTO\* konto);**

Holt den Kreditkontostand

**int removeKreditKontoVerfüger(int userid, char\* ktnr);**

Entfernt einen Verfügereintrag von einem Kreditkonto

**int addKreditkontoVerfüger(int userid, char\* ktnr);**

fügt einen Verfügereintrag zum Kreditkonto hinzu

**int kkverfügerExists(char\* ktnr, int userid);**

überprüft ob ein Usereintrag in einem Kreditkonto ist

**void KreditkontoVerfügerLoeschen**

**(KREDITKONTO\* Konto, CUSTOMER\* cust);**

overloaded - Löscht einen Verfügereintrag vom Kreditkonto

**void KreditkontoVerfügerHinzufügen**

**(KREDITKONTO\* Konto, CUSTOMER\* cust);**

overloaded - Sparkontoverfüger Hinzufügen zum Kreditkonto

Sparkonto

**SPARKONTO\* NeuesSparkonto(CUSTOMER\* Kunde);**

Neues Sparkonto anlegen

**SPARKONTO\* readSparKonto(char\* ktnr);**

Lesen und Rückgabe der Sparkontostruktur

**void Sparkontoentfernen(SPARKONTO\* Konto);**

Entfernen des Sparkontoeintrages

**double getSparkontostand(SPARKONTO\* konto);**

Sparkontostand holen

**char\* getSparKontonummer(SPARKONTO\* konto);**

Sparkontonummer holen

**int addSparkontoVerfüger(int userid, char\* ktnr);**

Sparkontoverfüger Hinzufügen zum Sparkonto

**int skverfügerExists(char\* ktnr, int userid);**

Ist die CustomerID in diesem Konto vermerkt?

**int removeSparKontoVerfüger(int userid, char\* ktnr);**

Löscht einen Verfügereintrag vom Sparkonto

**void doSparbuchabheben(SPARKONTO\* zielkonto, double betrag);**

Einen Betrag vom Sparbuch abheben

**void SparkontoVerfügerHinzufügen**

**(SPARKONTO\* Konto, CUSTOMER\* cust);**

Sparkontoverfüger Hinzufügen zum Sparkonto

void SparkontoVerfügerLoeschen

**(SPARKONTO\* Konto, CUSTOMER\* cust);**

Löscht einen Verfügereintrag vom Sparkonto

**void doSparen(SPARKONTO\* zielkonto,**

**char\* verwendungszweck, double betrag);**

Zahlt einen Betrag ein -> spart

### Bank & Konten

**int getAccountType(char\* kontonummer);**

Gibt den Kontotyp zurück

**int getBIC();**

Gibt BIC der Bank zurück

### Überweisung

**UEBERWEISUNG\* NeueUeberweisung**

**(KREDITKONTO\* quellkonto, KREDITKONTO\* zielkonto,**

**double betrag, char\* verwendungszweck);**

Überweisung von einem zum anderen Konto

### Währungsmodul

**void doUmrechnung(WAEHRUNGSMODUL\* waehrungsmmodul, char\* waehrung);**

Umrechnungen

**void doKursverwaltung(WAEHRUNGSMODUL\* waehrungsmodul);**

Kurs Verwalten

**WAEHRUNGSMODUL\* NeuesWaehrungsmodul(KREDITKONTO\* konto);**

Währungsmodul erstellen

**double waehrungsumrechnung(int \_currency, double \_value);**

Umrechnung der Währung 0 = EUR, 1 = USD, 2 = GBG

# Fazit des Projektes der 3 Laboreinheiten

Das Bankprojekt war mit seinen Aufgaben immer sehr spannend und konnte am Schluss erfolgreich mit dem RabbitMQ Server mit einer anderen Bank kommunizieren und Überweisungen tätigen.

Implementierungsstufen:

1. Native-c DLL zum Abbilden der Bankfunktionalitäten und mit der Libary cJSON für das Persistieren der Daten.
2. 2 Assemblies + Konsolenanwendung mit Verwendung der Austauschbaren Assemblies.
3. C# WPF Applikation mit Verwendung des Assemblies von der 2 Laboreinheit. Das Assembly beinhaltet nun auch Funktionen um über das Netzwerk, Überweisungen zu anderen Banken zu tätigen. Für die Kommunikation mit anderen Banken wurde ein RabbitMQ Server aufgesetzt der die Transaktionsanfragen an die anderen Banken weiterleitet.

Probleme und gewonnene Erfahrungen:

Aufgrund der provozierten Probleme und der Aufgabenstellung/Abwicklung konnten einige Erfahrungen mitgenommen werden.

Einige davon sind:

* DLL Hell
* Dllimports
* Wrapper/Adaptoren um 2 Eigene Komponenten austauschbar zu machen
* Wiederverwendbarkeit
* Netzwerkprogrammierung
* RabbitQM
* Protokolle Definieren
* Team/Gruppen/Aufteilung Organisieren
* Bankdomäne besser verstehen