****

**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: 20.10.2016**

**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 |  |  |  |
| Labor 2: Anpassung der native-c DLL |  |  |  |
| Labor 2: Konsolenapplikation |  |  |  |
| Labor 2: C# Assembly |  |  |  |
| Labor 3: Anpassung der native-c DLL (Kunden/Konten) |  |  |  |
| Labor 3: WPF Applikation |  |  |  |
| Labor 3: C# Applikation - Netzwerkkommunikation mit RabbitMQ implementiert |  |  |  |
|  |  |  |  |

Inhaltsverzeichnis

[1 Aufgabenstellung 1](#_Toc467505548)

[1.1 Labor 1 - Dynamic Link Libraries 1](#_Toc467505549)

[1.2 Labor 2 - Assemblies und Wrapper 1](#_Toc467505550)

[1.3 Labor 3 – Netzwerkkommunikation 1](#_Toc467505551)

[2 Komponentenübersicht 2](#_Toc467505552)

[3 Bankanwendung (GUI) 2](#_Toc467505553)

[4 C# Assembly 3](#_Toc467505554)

[4.1 BankManagement - Class 3](#_Toc467505555)

[5 Native-C DLL 3](#_Toc467505556)

[5.1 Schnittstelle – bank.dll 3](#_Toc467505557)

[6 Zusammenfassung der 3 Laboreinheiten 3](#_Toc467505558)

# Aufgabenstellung

<Genereller Aufbau der Labore bis zur finalen Abgabe – Zusammenfassung>

<Überleitung zu den 3 Laboraufgabenstellungen>

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 ist 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“.

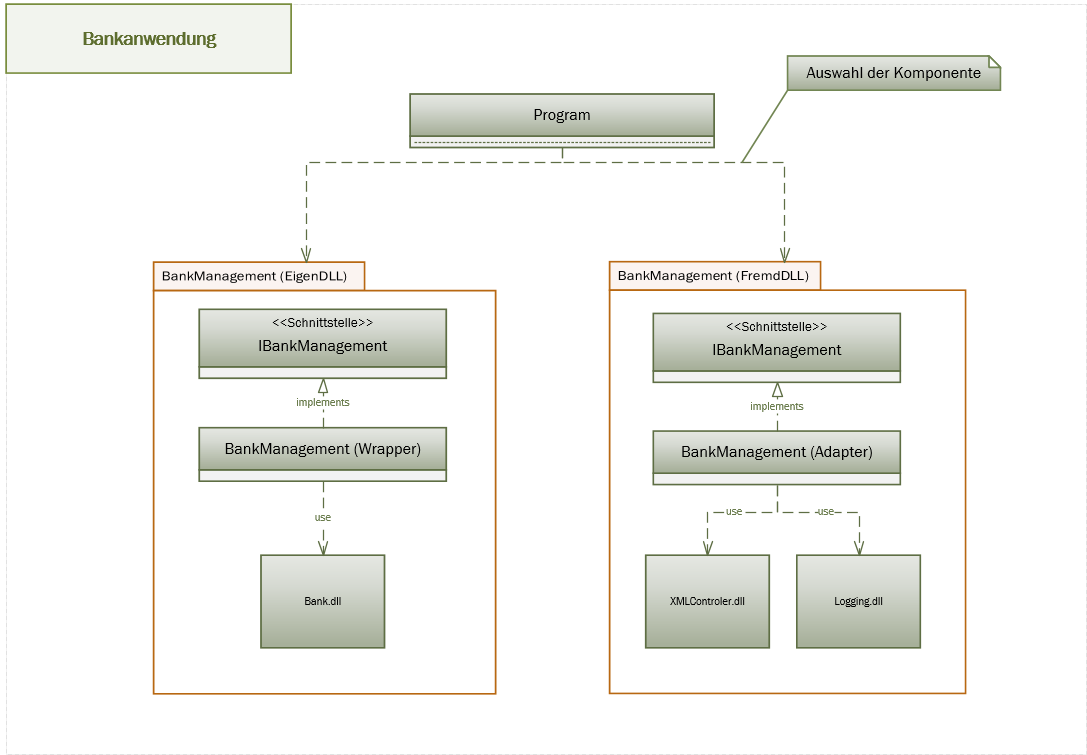
Testen Sie Ihre Assemblies wieder ausgiebig mit eigens dafür erstellten Testprogrammen und/oder UnitTests.

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 mit den austauschbaren Komponenten von unserer und einer anderen Gruppe dar.



cJSON

BankApplikation (GUI)

Nicht

Abbildung 1: Bankanwendung – Komponentendiagramm

Bei der Bankapplikation wird die Assembly vom der 2 Laboreinheit verwendet. Für die finale version wurde sie noch angepasst und für eine Netzwerkkommunikation zum RabbitMQ Server fit gemacht. Dadurch kann man nun zwischen den Banken mit einem definierten Protokoll überweisungen tätigen.

# Bankanwendung (GUI)

Verwendet c# Assembly das wiederum die native c dll nutzt.

Im dll sind auch funktionen der lib cJSON genutzt.

Im Assembly sind die funktionen der c dll „ge’Wrappt“ und es sind zusätlich noch funktionen für die Netzwerkanbindung umgesetzt.

# C# Assembly

Wrapper der Native C Dll, die Bankfunktionalitäten lassen sich mit dem Assembly Objektorientiert verwenden.

BankManagement - Class

<Schnittstellenfunktionen>

# Native-C DLL

Persistenz & Bankfunktionalitäten „low lvl“ abgebildet

Schnittstelle – bank.dll

<Schnittstellenfunktionen>

# Zusammenfassung des Projektes der 3 Laboreinheiten

Projekt war aufbauend und hatte 3 Stufen

Implementierungsstufen:

1. Native-c DLL, (cJSON)
2. 2 Assemblies + Konsolenanwendung mit Verwendung der Austauschbaren Assemby‘sl.
3. C# WPF Applikation mit Verwendung des Assembly’s von uns. Das Assembly beinhaltet nun auch Funktionen um über das Netzwerk, Überweisungen zu anderen Banken zu tätigen. zur Netzwerkkonnektivität und für die Überweisung zu dem anderen Banken mit einer RabbitMQ Verbindung.

Probleme und gewonnene Erfahrungen:

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