Projektname:	Bankomponeten
Projektnummer:	0002
Kurzbeschreibung:	Erstellung auf Basis der von einem anderen Team übernommenen DLLs und der eigenen DLLs ein gemeinsames C#/.NET Interface und das darauf basierend .NET Assemblies, die objektorientiert unter C# und .NET – dem eigenen DLL und dem übernommenen DLL als Wrapper/Adaptor/Bridge dienen. Es wird die volle bzw. geforderte Funktionalität der DLLs anbieten, ohne jedoch Details der DLLs und deren API kennen zu müssen.
Version:	1.2.2
Vorgelegt von:	Reimar Klammer, Johannes Probst, Daniel Komohorov
Author:	Daniel Komohorov
Auftraggeber:	DI (FH) DI Roland Graf, MSc
Erstelldatum:	25.10.2016

DOKUMENTENHISTORIE

Autor	Datum	Version	Änderung
Komohorov	26.10.2016	1.2.0	Erstellung der Dokumentation
Klammer	27.10.2016	1.2.1	Ergänzung Erklärung zu Moduls und Namspaces
Klammer	30.10.2016	1.2.2	Ergänzung Erklärung zu Moduls und Namspaces
•	•	•	•
•	•	•	•
•	•	•	•

I Inhaltsverzeichnis

1 Dokumentation der Softwarekomponenten

1.1 Funktionsweise der Implementierung

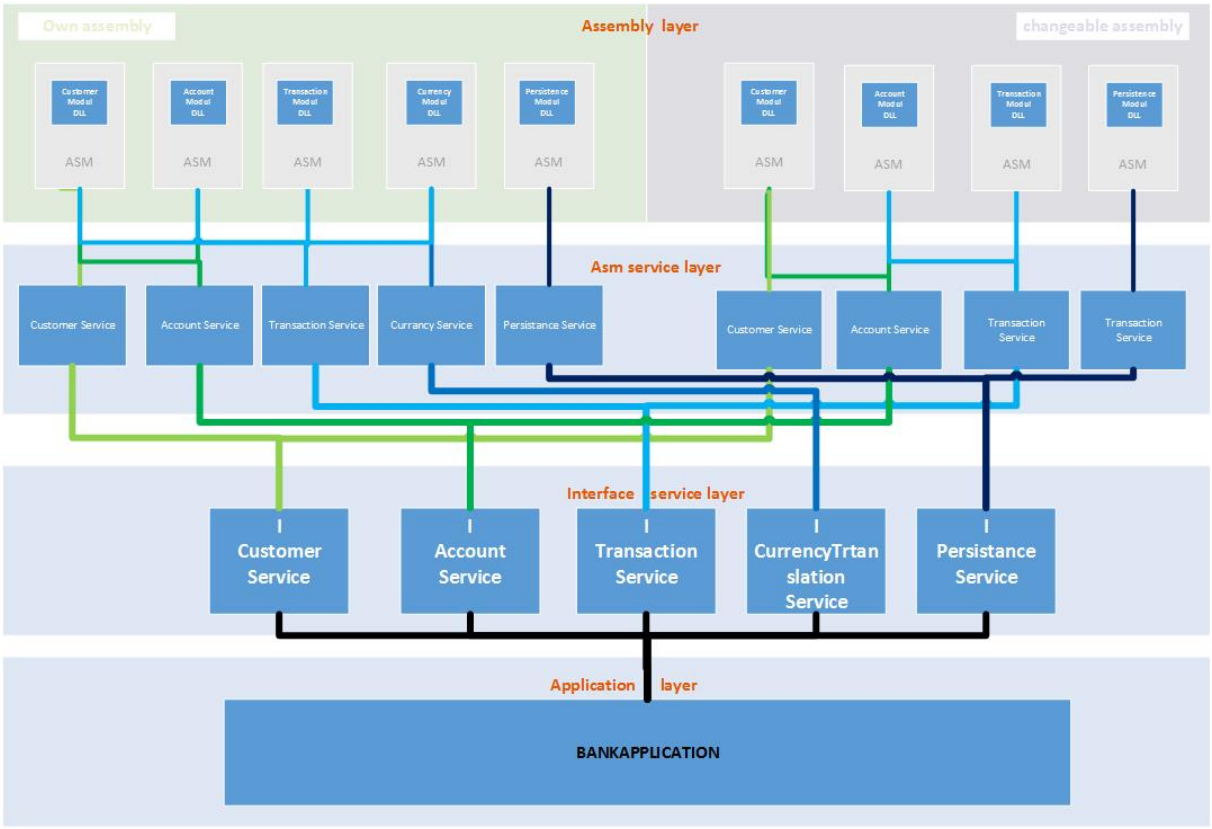

Abbildung 1: Komponentenübersicht

Die gesamte Implementierung ist auf 4 Layer aufgeteilt (Siehe Bild Komponentenübersicht).
Der **Assembley Layer** wurde in zwei Schritten erstellt:
Schritt 1) Eigene und fremde C/C++ DLL's in C# übersetzt (InternalWrapper)
Schritt 2) Die entstandene InternalWrapper in Übergeordnete Wrapper gepackt, damit diese nicht mehr über die Errorcodes, sonder Exeptions gehandelt werden.
(im Kapitel 1.3, Punkt 'Components.Common' das Exeptionhandling näher erläutert)

Im **Assembley Service Layer** ist die Anforderung von den jeweiligen Interfaces Implementiert.

Der **Interface Service Layer** stellt nun die Schnittstelle zwischen der Applikation und dem Services dar. Hier werden die geforderten Funktionalitäten nach außen abgebildet. Nach innen kennt der Interface Service Layer keine Services, sondern umgekehrt. So wurde die geforderte Kapselung umgesetzt. Zu dem definiertem Interface können beliebige Services (Komponenten) implementiert werden, so das nach außen nicht

ersichtlich ist, welcher Service tatsächlich aufgerufen wird.

Im **Application layer** wurde die Applikation 'BankApplication' umgesetzt. Um Sicher zu stellen, dass alle geforderten Funktionen dem Anwender zur Verfügung stehen, gibt es in diesem Fall die Möglichkeit einen eigenen oder einen fremden Service zu laden. Der Vorteil dabei ist, dass man unabhängig von dem Fertigstellungsgrad der fremden DLL's, die volle Funktionalität an die Anwender anbieten kann!

1.2 Beschreibung der Assemblies und deren Schnittstellen

Grundsätzliche Erklärung zu der Beschreibung: In den Schnittstellen wird definiert, was die jeweiligen Services können müssen. Beispiel: ICustomerService hat drei Methoden (CreateCustomer, DeleteCustomer und ModifyCustomer). Alle Services (egal ob eigene oder fremde), die die Schnittstelle bedienen, müssen diese Methoden zur Verfügung stellen.
Ein Beispielaufruf wird einmalig, anhand der Komponente 'Customer Service' und der dazugehörigen Methode 'CreateCustomer' dargestellt, und kann analog bei allen anderen angewendet werden:
- Aufruf auf dem Application Layer (Der erste Aufruf gibt den zuvor gewählten Serviceprovider, fremd oder eigen zurück):

```
1 var customerService = ServiceLocator.Instance().GetService<
      ICustomerService>();
2 customerService.CreateCustomer();
```

- Aufruf auf dem Interface Service Layer:

```
1 void CreateCustomer();
```

- Aufruf auf dem Assembly Service Layer:

```
1 var id = CustomerWrapper.CreateCustomer(firstName, lastName, street, zip
      );
```

- Aufruf auf dem Assembley Layer (Wrapper):

```
1 using cw = Components.Wrapper.Own.InternalCustomerWrapper;
2 cw.CreateCustomer(firstName, lastName, street, zip, out customerId);
```

- Aufruf auf dem Assembley Layer (InternalWrapper):

```
1 internal static extern int CreateCustomer(string firstName, string
      lastName, string street, int zip, out int id);
```

- Aufruf der darunter liegenden DLL, siehe Beschreibung der Version 1.1.0

Die Beschreibung geht aus von dem 'Interface Service Layer' und von eigen (own) Komponenten . Die Implementierung der Fremden Komponenten ist analog aufgebaut.

Die Klassen, Namespaces, Methoden und Parameter sind aus den Code-Ausschnitten ersichtlich und werden daher nicht explizit erläutert.Durch die Punkte '.....' wird signalisiert, dass es weitere Code-Teile gibt, die aber für die Beschreibung nicht relevant sind.

1.2.1 Interface Customer-Service

```
1 namespace Components.Contracts.Services
2 {
3     public interface ICustomerService
4     {
5         void CreateCustomer();
6         void DeleteCustomer();
7         void ModifyCustomer();
8     }
9 }
```

- Implementierung Service, 'Customer Service':

```
1 namespace Components.Service.Own
2 {
3     ..........
4     public class CustomerService : ICustomerService
5     {
6         public void CreateCustomer()
7         {
8           ..........
9             var id = CustomerWrapper.CreateCustomer(firstName, lastName,
                  street, zip);
10        }
11
12        public void DeleteCustomer()
13        {
14          ..........
15            CustomerWrapper.DeleteCustomer(customerId);
16        }
17
18        public void ModifyCustomer()
19        {
20          ..........
21            CustomerWrapper.ModifyCustomer(customerId, firstName,
                lastName, street, zip);
```

```
22
23          }
24      }
25  }
```

- Wrapper 'CustomerModul':

```
1   using cw = Components.Wrapper.Own.InternalCustomerWrapper;
2   namespace Components.Wrapper.Own
3   {
4       public static class CustomerWrapper
5       {
6           public static int CreateCustomer(string firstName, string
                lastName, string street, int zip)
7           {
8           ..........
9             cw.CreateCustomer(firstName, lastName, street, zip, out
                  customerId);
10             return customerId;
11          }
12
13          public static void DeleteCustomer(int id)
14          {
15          ..........
16           cw.DeleteCustomer(id);
17          }
18
19          public static void ModifyCustomer(int id, string firstName,
                string lastName, string street, int zip)
20          {
21          ..........
22           cw.ModifyCustomer(id,firstName,lastName,street, thezip);
23          }
24      }
25  }
```

- Internal Wrapper 'CustomerModul':

```
1   namespace Components.Wrapper.Own
2   {
3       internal static class InternalCustomerWrapper
4       {
5           [DllImport(Common.DllNames.OwnCustomerModuleName,
                CallingConvention = CallingConvention.Cdecl)]
6           internal static extern int CreateCustomer(string firstName,
                string lastName, string street, int zip, out int id);
```

```
7
8           [DllImport(Common.DllNames.OwnCustomerModuleName,
                CallingConvention = CallingConvention.Cdecl)]
9           internal static extern int DeleteCustomer(int id);
10
11          [DllImport(Common.DllNames.OwnCustomerModuleName,
                CallingConvention = CallingConvention.Cdecl)]
12          internal static extern int ModifyCustomer(int id, string
                firstName, string lastName, string street, int zip);
13      }
14  }
```

1.2.2 Interface Account-Service

```
1   namespace Components.Contracts.Services
2   {
3       public interface IAccountService
4       {
5           void CreateAccount();
6           void CloseAccount();
7           void AddDisposer();
8           void RemoveDisposer();
9       }
10  }
```

- Implementierung Service, 'Account Service':

```
1   namespace Components.Service.Own
2   {
3       [Export(typeof(IAccountService))]
4       public class AccountService : IAccountService
5       {
6           public void CreateAccount()
7           {
8           ..........
9               var accountNumber = AccountWrapper.CreateAccount(disposerId,
                    accountName, accountType == AccountType.LoanAccount ?
                    Wrapper.Own.AccountType.LoanAccount : Wrapper.Own.
                    AccountType.SavingsAccount);
10          }
11
12          public void CloseAccount()
13          {
14          ..........
15              AccountWrapper.CloseAccount(disposerId, accountNumber);
16          }
17
```

```
18          public void AddDisposer()
19          {
20          ..........
21                  AccountWrapper.AddDisposer(disposerId, accountNumber,
                        newDisposerId);
22          }
23
24          public void RemoveDisposer()
25          {
26          ..........
27                  AccountWrapper.RemoveDisposer(disposerId, accountNumber,
                        toRemoveDisposerId);
28
29          }
30      }
31  }
```

- Wrapper 'AccountModul':

```
1   using aw = Components.Wrapper.Own.InternalAccountWrapper;
2
3   namespace Components.Wrapper.Own
4   {
5       public static class AccountWrapper
6       {
7           public static int CreateAccount(int disposerId, string
                accountName, AccountType type)
8           {
9         aw.CreateAccount(disposerId, accountName, type, out accountNumber)
              );
10                return accountNumber;
11          }
12
13          public static void CloseAccount(int disposerId, int
                accountNumber)
14          {
15            aw.CloseAccount(disposerId, accountNumber));
16          }
17
18          public static void AddDisposer(int disposerId, int accountNumber
                , int newDisposerId)
19          {
20           aw.AddDisposer(disposerId, accountNumber, newDisposerId));
21          }
22
```

```
23          public static void RemoveDisposer(int disposerId, int
                accountNumber, int disposerToRemove)
24          {
25              aw.RemoveDisposer(disposerId, accountNumber, disposerToRemove
                    ));
26          }
27      }
28  }
```

- Internal Wrapper 'AccountModul':

```
1   namespace Components.Wrapper.Own
2   {
3       internal static class InternalAccountWrapper
4       {
5           [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
6           internal static extern int CreateAccount(int disposerId, string
                accountName, AccountType accountType, out int accountNumber);
7
8           [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
9           internal static extern int CloseAccount(int disposerId, int
                accountNumber);
10
11          [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
12          internal static extern int AddDisposer(int disposerId, int
                accountNumber, int newDisposerId);
13
14          [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
15          internal static extern int RemoveDisposer(int disposerId, int
                accountNumber, int disposerToRemove);
16      }
17  }
```

1.2.3 Interface Transaction-Service

```
1  namespace Components.Contracts.Services
2  {
3      public interface IAccountService
4      {
5          void CreateAccount();
6          void CloseAccount();
7          void AddDisposer();
8          void RemoveDisposer();
9      }
10 }
```

- Implementierung Service, ' TransactionService':

```
1  namespace Components.Service.Own
2  {
3      [Export(typeof(ITransactionService))]
4      public class TransactionService : ITransactionService
5      {
6          public void PayOut()
7          {
8          ..........
9              TransactionWrapper.PayOut(disposerId, accountNumber, amount,
                   currency);
10         }
11
12         public void PayIn()
13         {
14         ..........
15             TransactionWrapper.PayIn(disposerId, accountNumber, amount,
                   currency);
16         }
17
18         public void Transfer()
19         {
20         ..........
21             TransactionWrapper.Transfer(disposerId, sourceAccountNumber,
                   targetAccountNumber, amount, currency);
22         }
23
24         public void AccountStatement()
25         {
26         ..........
27             var transactions = TransactionWrapper.AccountStatement(
                   disposerId, accountNumber);
28         }
```

```
29
30          public void AccountBalancing()
31          {
32        ..........
33              var balance = TransactionWrapper.AccountBalancing(disposerId
                    , accountNumber, currency);
34          }
35      }
36  }
```

- Wrapper 'TransactionModul':

```
1   using tw = Components.Wrapper.Own.InternalTransactionWrapper;
2
3   namespace Components.Wrapper.Own
4   {
5       public static class TransactionWrapper
6       {
7           public static void PayOut(int disposerId, int accountNumber,
                double amount, OwnCurrency currency)
8           {
9              tw.PayOut(disposerId, accountNumber, amount, currency));
10          }
11
12          public static void PayIn(int disposerId, int accountNumber,
                double amount, OwnCurrency currency)
13          {
14              tw.PayIn(disposerId, accountNumber, amount, currency));
15          }
16
17          public static void Transfer(int disposerId, int
                fromAccountNumber, int toAccountNumber, double amount,
18              OwnCurrency currency)
19          {
20              tw.Transfer(disposerId, fromAccountNumber, toAccountNumber,
                    amount,currency));
21          }
22
23          public static IEnumerable<Transaction> AccountStatement(int
                disposerId, int accountNumber)
24          {
25              ..........
26        tw.AccountStatement(disposerId, accountNumber, transactions, out
              numOfEntries));
27
28              return transactions.Select(Transaction.FromStruct).ToList();
```

```
29          }
30
31          public static double AccountBalancing(int disposerId, int
                accountNumber, OwnCurrency currency)
32          {
33          ..........
34              tw.AccountBalancing(disposerId, accountNumber, currency, out
                    balance));
35              return balance;
36          }
37      }
38  }
```

- Internal Wrapper 'TransactionModul':

```
1   namespace Components.Wrapper.Own
2   {
3       internal static class InternalTransactionWrapper
4       {
5           [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
6           internal static extern int PayOut(int disposerId, int
                accountNumber, double amount, OwnCurrency ownCurrency);
7
8           [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
9           internal static extern int PayIn(int disposerId, int
                accountNumber, double amount, OwnCurrency currency);
10
11          [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
12          internal static extern int Transfer(int disposerId, int
                fromAccountNumber, int toAccountNumber, double amount,
                OwnCurrency currency);
13
14          [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
15          internal static extern int AccountStatement(int disposerId, int
                accountNumber, [In,Out] TransactionStruct[] data, out int
                numberOfEntries);
16
17          [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
18          internal static extern int AccountBalancing(int disposerId, int
                accountNumber, OwnCurrency currency, out double balance);
19      }
```

```
20 }
```

1.2.4 Interface Currency-Translation-Service

```
1  namespace Components.Contracts.Services
2  {
3      public interface ICurrencyTranslationService
4      {
5          void SetCurrencyToEuroFactor();
6          void GetCurrencyToEuroFactor();
7          void TranslateToEuro();
8          void TranslateFromEuro();
9      }
10 }
```

- Implementierung Service, ' CurrancyTranslationService':

```
1  namespace Components.Service.Own
2  {
3      [Export(typeof(ICurrencyTranslationService))]
4      public class CurrencyTranslationService :
           ICurrencyTranslationService
5      {
6          public void SetCurrencyToEuroFactor()
7          {
8          ..........
9              CurrencyTranslationWrapper.SetCurrencyToEuroFactor(currency,
                   factor);
10         }
11
12         public void GetCurrencyToEuroFactor()
13         {
14         ..........
15             var factor = CurrencyTranslationWrapper.
                   GetCurrencyToEuroFactor(currency);
16         }
17
18         public void TranslateToEuro()
19         {
20         ..........
21             var result = CurrencyTranslationWrapper.TranslateToEuro(
                   currency, amount);
22         }
23
24         public void TranslateFromEuro()
25         {
26         ..........
```

```
27                var result = CurrencyTranslationWrapper.TranslateFromEuro(
                      currency, amount);
28            }
29        }
30    }
```

- Wrapper 'CurrancyTranslationWrapper':

```
1   using tw = Components.Wrapper.Own.InternalCurrencyTranslationWrapper;
2   namespace Components.Wrapper.Own
3   {
4       public static class CurrencyTranslationWrapper
5       {
6           public static void SetCurrencyToEuroFactor(OwnCurrency currency,
                double factor)
7           {
8               SaveApiCaller.ExecuteCall(() => tw.SetCurrencyToEuroFactor(
                    currency, factor));
9           }
10
11          public static double GetCurrencyToEuroFactor(OwnCurrency
                currency)
12          {
13              double factor = 0;
14              SaveApiCaller.ExecuteCall(() => tw.GetCurrencyToEuroFactor(
                    currency, out factor));
15              return factor;
16          }
17
18          public static double TranslateToEuro(OwnCurrency currency,
                double amount)
19          {
20              double result = 0;
21              SaveApiCaller.ExecuteCall(() => tw.TranslateToEuro(currency,
                     amount, out result));
22              return result;
23          }
24
25          public static double TranslateFromEuro(OwnCurrency currency,
                double amount)
26          {
27              double result = 0;
28              SaveApiCaller.ExecuteCall(() => tw.TranslateFromEuro(
                    currency, amount, out result));
29              return result;
30          }
```

```
31      }
32  }
```

- Internal Wrapper 'CurrancyTranslationWrapper':

```
1  namespace Components.Wrapper.Own
2  {
3      internal static class InternalCurrencyTranslationWrapper
4      {
5          [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
6          internal static extern int SetCurrencyToEuroFactor(OwnCurrency
               ownCurrency, double factor);
7
8          [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
9          internal static extern int GetCurrencyToEuroFactor(OwnCurrency
               ownCurrency, out double factor);
10
11         [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
12         internal static extern int TranslateToEuro(OwnCurrency
               ownCurrency, double amount, out double result);
13
14         [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
15         internal static extern int TranslateFromEuro(OwnCurrency
               ownCurrency, double amount, out double result);
16     }
17 }
```

1.2.5 Interface Persistence-Service

```
1  namespace Components.Contracts.Services
2  {
3      public interface IPersistenceService
4      {
5          void Save();
6          void Load();
7      }
8  }
```

- Implementierung Service, 'Persistance Service':

```
1  namespace Components.Service.Own
2  {
3      [Export(typeof(IPersistenceService))]
```

```
4       public class PersistenceService : IPersistenceService
5       {
6           public void Save()
7           {
8               PersistenceWrapper.Store();
9           }
10
11          public void Load()
12          {
13              PersistenceWrapper.Load();
14          }
15      }
```

- Wrapper 'PersistanceModul':

```
1   using pw = Components.Wrapper.Own.InternalPersistenceWrapper;
2
3   namespace Components.Wrapper.Own
4   {
5       public static class PersistenceWrapper
6       {
7           public static void Load()
8           {
9              pw.Load;
10          }
11
12          public static void Store()
13          {
14              pw.Stor);
15          }
16      }
17  }
```

- Internal Wrapper 'PersistanceModul':

```
1   namespace Components.Wrapper.Own
2   {
3       public static class InternalPersistenceWrapper
4       {
5           [DllImport(Common.DllNames.OwnPersistenceModule,
                CallingConvention = CallingConvention.Cdecl)]
6           public static extern int Load();
7
8           [DllImport(Common.DllNames.OwnPersistenceModule,
                CallingConvention = CallingConvention.Cdecl)]
9           public static extern int Store();
```

```
10          }
11  }
```

1.3 Namespace Übersicht

Components
Basisnamespace

Components.Wrapper
Alle C/C++ Dll Wrapper

Components.Wrapper.Own
Wrapper für eigene Dlls

Components.Wrapper.Foreign
Wrapper für Fremde Dlls

Components.Common
Gemeinsam genutzte Daten/Klassen/Enumerationen/Strings
Inputparser: Statische Klasse genutzt um Benutzereingaben entgegen zu nehmen, und teilweise zu validieren.
ExceptionFactory: Statische Klasse die Fehlercodes aus den eigenen C/C++ Dlls in Exceptions umwandelt
SaveApiCaller: Statische klasse, die die Aufrufe auf die Internen Wrapper in try/catch Blöcke umhüllt und aus Fehlercodes mithilfe der ExceptionFactory Exceptions generiert.

Components.Common.Exceptions
Gemeinsam genutzte Exceptions

Components.Contracts
Allgemeine Interfaces

Components.Contracts.Services
Interfaces der Services

Components.Service
Basisnamespace für die Service Implementierungen

Components.Service.Own
Implementierung der Services die die eigenen C/C++ Dlls verwenden

Components.Service.Foreign
Implementierung der Services die die fremden C/C++ Dlls verwenden

1.4 Zusammenfassung und Kurzbeschreibung von allen Komponenten aus dem Debug-Order

bin/Debug/AccountManagement.dll
Die C++ Dll, der anderen Gruppe, zuständig für das Account Management

bin/Debug/AccountModule.dll
Die C++ Dll, zuständig für das Account Management

bin/Debug/Components.Common.dll
dotNet Assembly wo gemeinsam genutzte Funktionen enthalten sind.

bin/Debug/Components.Contracts.dll
dotNet Assembly welche die Interfaces zu den Services (eigene und Fremde) enthält

bin/Debug/Components.Service.Foreign.AccountService.dll
Service Implementierung des fremden AccountSerivce

bin/Debug/Components.Service.Foreign.CurrencyTranslationService.dll
Service Implementierung des fremden Währungsumrechners -¿ wurde von der anderen Gruppe nicht implementiert

bin/Debug/Components.Service.Foreign.CustomerService.dll
Service Implementierung des fremden Kundenmanagements

bin/Debug/Components.Service.Foreign.PersistenceService.dll
Service Implementierung der fremden Persistence

bin/Debug/Components.Service.Foreign.TransactionService.dll
Service Implementierung des fremden Überweisungsservices

bin/Debug/Components.Service.Own.AccountService.dll
Service Implementierung des eigenen AccountSerivce

bin/Debug/Components.Service.Own.CurrencyTranslationService.dll
Service Implementierung des eigenen Währungsumrechners

bin/Debug/Components.Service.Own.CustomerService.dll
Service Implementierung des Kundenmanagements

bin/Debug/Components.Service.Own.PersistenceService.dll
Service Implementierung der Persistence

bin/Debug/Components.Service.Own.TransactionService.dll
Service Implementierung des Überweisungsservices

bin/Debug/Components.Wrapper.Foreign.AccountWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Foreign.CustomerWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Foreign.PersistenceWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Foreign.TransactionWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Own.AccountWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.CurrencyTranslationWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.CustomerWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.PersistenceWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.TransactionWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/CurrencyTranslationModule.dll
Die C++ Dll, zuständig für die Währungsumrechnung

bin/Debug/CustomerManagement.dll
Die C++ Dll, der anderen Gruppe, zuständig für das Kundenmanagement

bin/Debug/CustomerModule.dll
Die C++ Dll, zuständig für das Kundenmanagement

bin/Debug/DataAccessLayer.dll
Die C++ Dll, der anderen Gruppe, zuständig für die Persistence

bin/Debug/PersistenceModule.dll
Die C++ Dll, zuständig für die Persistence

bin/Debug/Shared.dll
Die C++ Komponente wo gemeinsam genutzte Funktionen implementiert sind.

bin/Debug/TransactionManagement.dll
Die C++ Dll, der anderen Gruppe, zuständig für die Überweisungen

bin/Debug/TransactionModule.dll
Die C++ Dll, zuständig für die Überweisungen

---

2 TODO

(Alle TODO's erledigt. Topic bleibt trotzdem, falls es weitere TODO's dazukommen)

3 Offene Punkte

Die Fremde DLL's konnten nicht endgültig implementiert werden auf Grund von fehlenden Funktionalitäten aus der Laborübung 1.
Status bis zum 01.11.2016 betreffend der Implementierung der Fremd-DLL:
CustomerManagement:
- CreateCustomer() funktionsfähig
- GetCustomerById() nicht funktionsfähig
- updateCustomer() nicht funktionsfähig
- DeleteCustomer() nicht funktionsfähig
Fazit: CustomerManagement.dll zu 3/4 nicht funktionsfähig

AccountManagement:
- createAccount() funktionsfähig
- updateAccount() nicht funktionsfähig
- addDisposer() nicht funktionsfähig
- closeAccount() funktionsfähig
- getAccountById() nicht funktionsfähig
- GetAccountsByCustomerId() nicht funktionsfähig
Fazit: AccountManagement.dll zu 2/3 nicht funktionsfähig

TransactionManagement:
- auszahlung() funktionsfähig
- einzahlung() funktionsfähig
- calcSaldo() nicht funktionsfähig
- ueberweisung() funktionsfähig
Fazit: TransactionManagement.dll zu 3/4 funktionsfähig, jedoch keine Möglichkeit an die Transaktionen oder den Kontostand zu kommen.

4 Anhang

A) Komponenten Übersicht

B) Dokumentation inkl. Komponentenübersicht

C) Die gesamte Solution

D) Video
