HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

CAT FP

Triaxis Trust AG
Member of Cat Group

Redesign einer Banking-Software

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2011

Autor(en):	Sascha Bauer, Alexander Klee
Betreuer:	Prof. Hansjörg Huser
Projektpartner:	CAT Financial Products, Triaxis AG Zürich
Experte:	Prof. Hansjörg Huser
Gegenleser:	Prof. Peter Sommerlad

TABLE OF CONTENTS

1 ABSTRACT

Studienjahr	**FS 2011**
Titel der Studienarbeit	**StruktoManager2.0 - Redesign einer Banking-Applikation**
1.1.1.1 Examinator	**Prof. Hansjörg Huser**
1.1.1.2 Themengebiet	**Software**
1.1.1.3 Projektpartner	**Cat Financial Products (Triaxis Trust AG), Zürich**
1.1.1.4 Institut	**Institut für Software**

Seit drei Jahren benutzt und entwickelt die Firma „Cat Financial Products“ das Portfolio-Management-Tool „StruktoManager“, um die von ihnen gehandelten Finanzprodukte zu verwalten. Unser Auftrag bestand darin, eine komplette Neugestaltung der Softwarearchitektur, mithilfe des Net-Framework 4.0 und WPF, vorzunehmen um eine wirtschaftliche und effiziente Weiterentwicklung zu ermöglichen.

In Form einer multitaskingfähigen 2-Tier Architektur strebten wir eine möglichst hohe Entkopplung von UI und Business Layer an. Das Unity Framework setzt auf dependency injection und erlaubte uns Businessoperationen auf dem UI-Layer als Service anzubieten ohne Kenntnis der Implementierung der Service Wrapper. Unter anderem entstand daraus ein „change tracking“, welches das Persistieren komplexer „obejct trees“ unabhängig vom spezifischen Datentyp des root nodes ermöglicht.

Neben dem UI-Layer besteht die Architektur aus einem Service-, Business- und DataAccess-Layer. Die Aufgabe des Service-Layer liegt darin, die Serviceaufrufe der UI-Threads entgegenzunehmen, diese in Tasks zu verpacken und nach Bedarf parallel oder seriell auszuführen. Unser change tracking ermöglicht es, aus einem komplexen object tree heraus alle veränderten Attribute der abhängigen Entitäten zu finden. Diese können gezielt, nur mit dem Wissen über die gemeinsame Basisklasse (entity object), persistiert werden.

Durch die Entkopplung der Layer wurde die Weiterentwicklung stark erleichtert, da mehrere Personen an logisch unterteilten Bereichen arbeiten können. Dies wurde zu Beginn der Arbeit vom Auftraggeber als kritischer Punkt bezeichnet.

Projekt: StruktoManager 2.0

Technischer Bericht

2 EINLEITUNG UND ÜBERSICHT

2.1 ENTSTEHEUNG DES STRUKTOMANAGERS

Als die Entwicklungsarbeiten des StruktoManagers vor 3 Jahren begannen, wusste man noch nicht so genau, in welche Richtung sich das Programm im Detail entwickeln wird. Anfangs war lediglich von einem Mapping der Kunden auf die in Ihren Portfolios befindlichen Produkten die Rede. Der Head Financial Products, der das Projekt initiiert hatte, verfolgte das Ziel seinen Arbeitsalltag im Handel mit strukturierten Produkten zu vereinfachen. Was anfangs ein überschaubares Erfassungstool war, wuchs schnell zu einer komplexen Datenverarbeitungssoftware. Aus einem, die Produkte beschreibenden und übersichtbietenden Hilfsmittel wurde mit der Zeit immer mehr ein mächtiges Kalkulationsinstrument, das neben dem Zahlungsverkehr, Kommissionen, aktuelle Börsenkurse und verschieden Währungen, etc. handhaben konnte. Stetig wurden neue Funktionalitäten in das Programm eingefügt, die Entwicklung musste schnell vorangehen und es blieb wenig Zeit die wuchernde Architektur aufzuräumen.

2.1.1 INNOVATIONSDRUCK

Um den Betrieb nicht aufzuhalten nahm man sich nicht immer die Zeit, eine Architekturentscheidung gut zu überdenken. Oft wurde beim Design der Architektur der Weg des geringsten Widerstands gewählt, was oft auch bedeutet hat bestehenden Code zu kopieren und an einer anderen Stelle, leicht verändert, wieder einzufügen. Ein absolutes Horrorszenario wenn man an die Wartung der Applikation denkt, denn wenn sich ein Kopierter Code als fehlerhaft erweisen sollte, ist man eventuell gezwungen auch alle kopierten Codeteile anzupassen.

2.1.2 DEAD FUNCTIONALITY

Auch wurden Funktionen angedacht, die nie zur Verwendung kamen.

2.1.3 NOTWENDIGKEIT EINER NEUGESTALTUNG

Es handelte sich bei der Vorgängerversion unserer Entwicklung um ein Experiment im produktiven Betrieb, dessen Grundstruktur nun durch unsere Arbeit auf einen modernen und professionellen Level gebracht werden sollte.

2.1.4 BUSINESSTECHNISCHE PROBLEME

1. Die Arbeit der User sollte auch bei zeitintensiven Businessoperationen nicht behindert werden.
2. Mehrere Entwickler sollen an logisch voneinander getrennten Modulen (Services) weitere Businesslogik hinzufügen können. 3. Die 2-Tier Architektur soll in Zukunft möglichst einfach auf n-Tier umgestellt werden können.

2.2 MIGRATION DER FUNKTIONALITÄT

Die Funktionalität des StruktoManagers entstand in erster Linie nicht aus dem Programmcode, sondern durch die Verknüpfung tausender Datensätze einer sehr komplexen Datenbank. Schnell wurde uns klar, dass die Datenbank nicht nach freiem Ermessen neu gestaltet werden kann. sondern dass die bestehenden Daten übernommen werden müssen.

2.3 QUELLE DER FUNKTIONALITÄT

So entschieden wir uns die Datenbank per Skript automatisch neu anzuordnen.
Auch diverse externe Schnittstellen sind Bestandteile und es kann sein, dass irgendwann neue dazukommen. Bis anhin sind das ein Pdf-Generator, ein Webservice Zugang, um das Publizieren der Produkte auf einer Webseite zu ermöglichen und eine Bloomberg-COM-Schnittstelle, die den Zugang zum Datenmonopol von Bloomberg bereitstellt. Die Bloomberg Daten sind von zentraler Bedeutung für die Korrektheit der Berechnungen, sie liefern stets die aktuellen Börsenkurse der Produkte sowie deren Basiswerte.

Um all diese „Services“ mit einer möglichst einheitlichen Struktur unter einen Hut zu bringen entschieden wir uns ein multitaskingfähiges Serviceframework zu implementieren. Folgende Anforderungen wurden an das Serviceframework gestellt:

- Garantierte Funktionalität, unabhängig von Änderungen auf der Datenschicht.
- Komplette Entkopplung vom oberen Layer (UI) und dem DataAcces-Layer.
- Möglichkeit sowohl synchrone als auch asynchrone Serviceverhalten zu kapseln.

3 ERGEBNISSE

3.1 ARCHITEKTUR

3.1.1 2 TIER

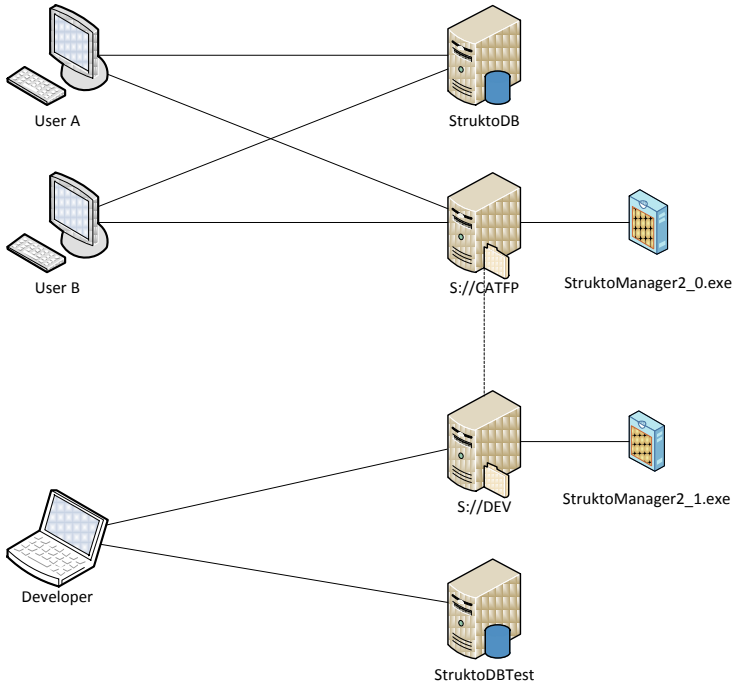


Abbildung 1 Neue 2 Tier Architektur mit Testumgebung

Die Architektur wurde gemäss obiger Abbildung beim Auftraggeber eingeführt. Vor dem Projekt wurde direkt auf der produktiven Datenbank StruktoDB entwickelt. Neu wurde eine TestDB sowie ein Developer-FileServer installiert. Die Ordnerstruktur des DEV-Shares entspricht derjenigen des CATFP-Shares. Dies ermöglicht eine sichere Weiterentwicklung der Software neben dem produktiven Betrieb.

3.2 DATENBANK

Eine Detailbeschreibung der vollzogenen Änderungen findet man im SoftwareArchitectureDocument.

3.2.1 EINSCHRÄNKUNG DER KOMPLEXITÄT

	StruktoManager1.x	StruktoManager2.0	StruktoManager2.x
Anzahl Tabellen	45	28	≈ 20
Grösste Tabelle (Produkt)	40 Zeilen	25 Zeilen (in EF)	≈ 20 (in EF)

Tabelle 1 Komplexitätsverminderung aufgrund neuer Tabellenstruktur

Im Vergleich zur alten Version müssen die Fremdschlüssel nicht mehr berücksichtigt werden, da diese vom EF gemanagt werden.

3.2.2 GEWÄHRLEISTUNG VON 1 ZU 1 BEZIEHUNGEN

Da es sich in der Datenbank oft nur um 1 zu 1 Beziehungen zwischen Tabellen handelt, mussten wir uns überlegen, wie wir diese im Entity Framework auch so auflösen zu können. Es wird erwartet, dass der Primary Key auch gleichzeitig der Fremdschlüssel für die in Beziehung stehende Tabelle ist. Dazu muss man in unserem Fall eine Haupttabelle definieren und alle anderen mit dieser Tabelle verknüpfen. Das wurde wie folgt gelöst:

Abbildung 2 One-to-one Constraints für EntityFramework

Nun ist in jeder „Untertabelle“ `tblProdukt_id` der Fremdschlüssel auf `tblProdukt` und gleichzeitig auch der Primary Key und somit Unique.

3.2.3 ENUM TABLE

Aus den typenbeschreibenden Tabellen, die in der Datenbank eliminiert wurden sind folgende Enum Typen entstanden (Abbildung links). Über das Settings File (Abbildung rechts) können diese im Programm identifiziert werden.

tblEnum_id	e_typ
1	eAnlageKlasse
101	eAuslöser
201	eBeobachtung
301	eCallDaten
401	eEventTyp
501	eGeschäftArt
601	eProdKlassifizierung
701	eProdStatus
801	eSchutzTyp
901	eSettlement
1001	eTerminArt
1101	eVerrechnungssteuer
1201	eWährungsRisiko
1301	eZeichnungsArt
1401	eCouponPeriode
1501	eBloombergTyp
1601	ePreisangabe
1701	eWährung
1801	eProdukteKat
1901	eProdukteTyp
2001	EFileTyp

Abbildung 3 Enum-Kategorien

eAnlageKlasse	string	Application	1-100
eAuslöser	string	Application	101-200
eBeobachtung	string	Application	201-300
eCallDaten	string	Application	301-400
eEventTyp	string	Application	401-500
eGeschäftArt	string	Application	501-600
eProdKlassifizie...	string	Application	601-700
eProdStatus	string	Application	701-800
eSchutzTyp	string	Application	801-900
eSettlement	string	Application	901-1000
eTerminArt	string	Application	1001-1100
eWährungsRisiko	string	Application	1201-1300
eZeichnungsArt	string	Application	1301-1400
eCouponPeriode	string	Application	1401-1500
eBloombergTyp	string	Application	1501-1600
ePreisangabe	string	Application	1601-1700
eWährung	string	Application	1701-1800
eProdukteKat	string	Application	1801-1900
eProdukteTyp	string	Application	1901-2000
eFileTyp	string	Application	2001-2100

Abbildung 4 Mapping der Kategorien in den Settings

3.2.4 SKRIPT

Abfolge	Skript	Beschreibung
1	migrateAll.sql	Alle untenstehenden Scripts werden der Reihe nach ausgeführt, wird diese Datei gestartet, so wird die Datenbankänderung vollzogen
2	procedure.sql	Erzeugt eine Prozedur, die für alle möglichen EnumTypen die aus der Datenbank eliminiert werden müssen, abarbeiten kann. Dabei mussten zuerst alle Referenzen auf die jeweiligen Tabellen entfernt werden, die Werte dann in eine temporäre Tabelle ausgelagert und schlussendlich mit den richtigen Id’s in die tblEnum eingefügt werden.

3	detailsCleaning.sql	Hier werden die einzelnen kleinen Fehler der Datenstruktur behoben
4	execProcedure.sql	Die Aufrufe der oben erzeugten Prozedur werden gehandhabt und teils mussten Constraints, die keinen Nutzen hatten entfernt werden.
5	resolve_valor_fk.sql	Löst die alten Constraints auf, die auf dem Valor basierten und erzeugt bei allen betroffenen Tabellen den Foreign und Primary Key tblProdukt_Id, es handelt sich wieder um einen komplexen Vorgang mit zwischenspeichern der Werte

Abbildung 5 DB-Migrations-Skripte

3.3 STRUKTO TASK MANAGER

Das Tasking besteht in erster Linie aus dem StruktoTaskManager. Dieser fungiert als zentraler Ansprechpunkt für sämtliche Serviceaufrufe. Der StruktoTaskManagers entkoppelt das UI vom Business- und DataAccessLayer. Die folgenden Abschnitte beschreiben die wichtigsten Methoden im Umgang mit dem StruktoTaskManager.

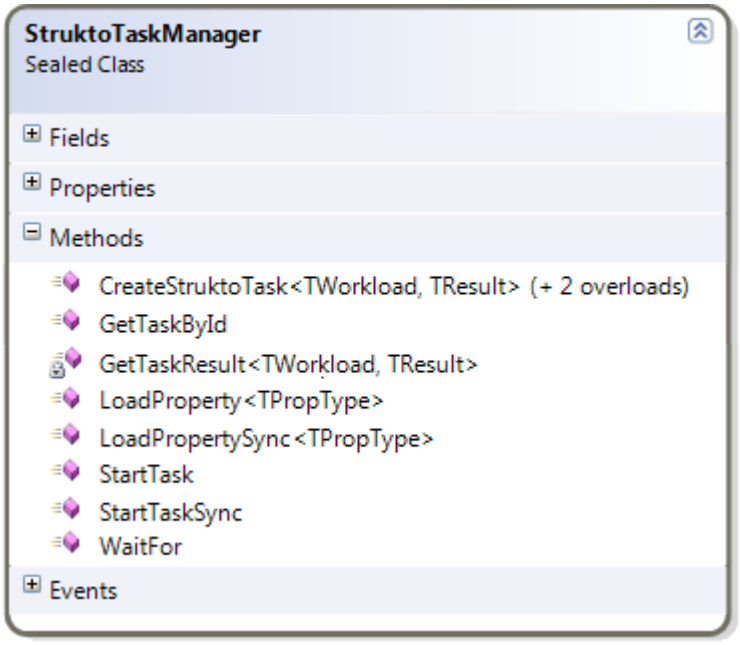


Abbildung 6 Wichtigste Methoden des StruktoTaskManagers

3.3.1 TASKS ERSTELLEN

Tasks können über die Methode CreateStruktoTask und beiden Overloads erstellt werden.

```
Private static void CreateTaskExamples()
{
        var manager = StruktoTaskManager.Default;
        var workload = "Hello World";

        // Create Tasks
/*1*/   var t1 = manager.CreateStruktoTask(workload,ServiceOperation.StringInStringOutOperation);
/*2*/   var t2 = manager.CreateStruktoTask<string, int>(workload,ServiceOperation.StringInIntOutOperation);
/*3*/   var t3 = manager.CreateStruktoTask(() => workload = "Goodbye");

        // Start Tasks
        manager.StartTask(t1);
/*4*/   manager.StartTask(t2);
        manager.StartTask(t3);
}
```

Listing 1 Erstellen eines Task

Obiges Listing zeigt wie die drei Methoden zu benutzen sind.

1. Erstellt einen Task der einen String entgegen, und als Resultat einen String zurückliefert.
2. Erstellt einen Task der einen String entgegennimmt und einen Integer zurückgibt.

3. Erstellt einen Task der eine Action ausführt.
4. Die Aufrufreihenfolge muss nicht der Ausführungsreihenfolge entsprechen.

Die Methode CreateStruktoTask und ihre Overloads wird im folgenden Listing vorgestellt.

```
private readonly object lockThis = new object();
public IStruktoTask CreateStruktoTask<TWorkload, TResult>(TWorkload workload, ServiceOperation serviceOper
ation)
{
/*1*/   lock (lockThis)
        {
/*2*/            var struktoTask = new StruktoTask<TWorkload, TResult>(
/*3*/                    new TaskDescription<TWorkload, TResult>(DoResolveService)
                       {
                           Workload = workload,
                           ServiceOperation = serviceOperation
                       });
/*4*/            taskPool.RegisterTask(struktoTask);
               return struktoTask;
        }
}

private StruktoServiceBase DoResolveService(ServiceOperation serviceOperation)
{
        try
        {
               return ServiceContainer.Resolve<StruktoServiceBase>(serviceOperation.ServiceName);
        }
        catch (ResolutionFailedException e) { ... }
}


public IStruktoTask CreateStruktoTask<TWorkload>(TWorkload workload, ServiceOperation serviceOperation)
{
        return CreateStruktoTask<TWorkload, TWorkload>(workload, serviceOperation);
}

public IStruktoTask CreateStruktoTask (Action action)
{
        return CreateStruktoTask(action, ServiceOperation.WorkSync);
}
```

Listing 2 CreateStruktoTask-Methode

1. Die Methode muss Thread Safe ablaufen und wird hier gelockt.
2. Der StruktoTask (siehe 3.4) wird erstellt, indem
3. eine TaskDescription instanziiert wird. (siehe 3.4.1)
4. Der Task wird auf dem TaskPool (siehe 3.3.6) registriert und anschliessend zurückgegeben.

3.3.2 LADEN VON TABELLEN

Eine häufige Operation ist das Laden von ganzen Tabellen aus der Datenbank. Der StruktoTaskManager stellt je eine synchrone und eine asynchrone Methode zur Verfügung um dies zu bewerkstelligen.

```
private ObservableCollection<BaseValue> BaseValues{ get; set; };
private void InitializeBaseValues()
{
   StruktoTaskManager.Default
        .LoadProperty<ObservableCollection<BaseValue>>(result => BaseValues = result); }
```

```
}
```

Listing 3 Beispiel eines LoadProperty-Aufrufs

Aus Listing 3 ist ersichtlich wie die LoadProperty-Methoden vom UI aus benutzt werden können. Als einziger Parameter wird eine Action<TPropertyType> erwartet, in der man das Verhalten nach Beendigung des Tasks definieren kann. In diesem Falle wird das Property BaseValues mit dem Resultat des Tasks gefüllt.

3.3.3 LOAD PROPERTY METHODE

Die LoadProperty-Methode sieht wie folgt aus.

```
public IStruktoTask LoadProperty<TPropType>(Action<TPropType> callback)
where TPropType : class
{
/*1*/   var localContext = InstatiateContextFromType<TPropType>();
/*2*/   Func<TPropType> loader =
                () => GetTaskResult<TPropType, TPropType>(localContext, ServiceOperation.Load);
/*3*/   var propertyLoaderTask = CreateWorkerTask(() => callback(loader.Invoke()));
/*4*/   StartTask(propertyLoaderTask);
        return propertyLoaderTask;
}
```

Listing 4 Details der LoadProperty-Methode

Hier geschieht folgendes:

1. Aus dem Template-Parameter TPropType wird eine Instanz generiert.
2. Der Loader wird initialisiert. Eine Funktion die einen Task startet und das Resultat zurückgibt. Im Falle der LoadProperty-Methode ist der Rückgabetyp des Loaders immer vom Typ TPropertyType.
3. Mit CreateWorkerTask() wird ein Task erstellt der den Loader ausführt und das Resultat dem Callback übergibt.
4. Der Task wird gestartet und zurückgegeben.

3.3.4 KOMPLEXE ARBEITSABLÄUFE SELBST MODELLIEREN

```
private static void GenerateReportAndPublishProduct(Product p)
{
        var manager = StruktoTaskManager.Default;
        var taskId = manager.StartOperation(p, ServiceOperation.Report);
        manager.WaitFor(taskId);

        // Do something with reported product...

        manager.StartOperation(p, ServiceOperation.Publish);
}
```

Listing 5 Einsatz von WaitFor

Der StruktoTaskManager (siehe 3.3) bietet mit WaitFor() eine Funktion an um auf einen bereits gestarteten Task zu warten. So können Arbeitsabläufe modelliert werden. WaitFor() verfügt über kein Timeout, da sämtliche Tasks vom User manuell abgebrochen werden können. Trotzdem ist bei der Verwendung von WaitFor Vorsicht geboten.

3.3.5 SERVICE CONTAINER

Dieser UnityContainer dient dem Taskmanager dazu seine Services zu verwalten.
An dieser Stelle sei auf die Dokumentation des Unity Application Block verwiesen.

3.3.6 TASKPOOL

Der TaskPool ist eine weitere Helferklasse des TaskManagers und verwaltet die erstellten Tasks. Zusätzlich versieht er die Tasks mit einem CancellationToken, über das der Task abgebrochen werden kann.

3.4 STRUKTO TASK

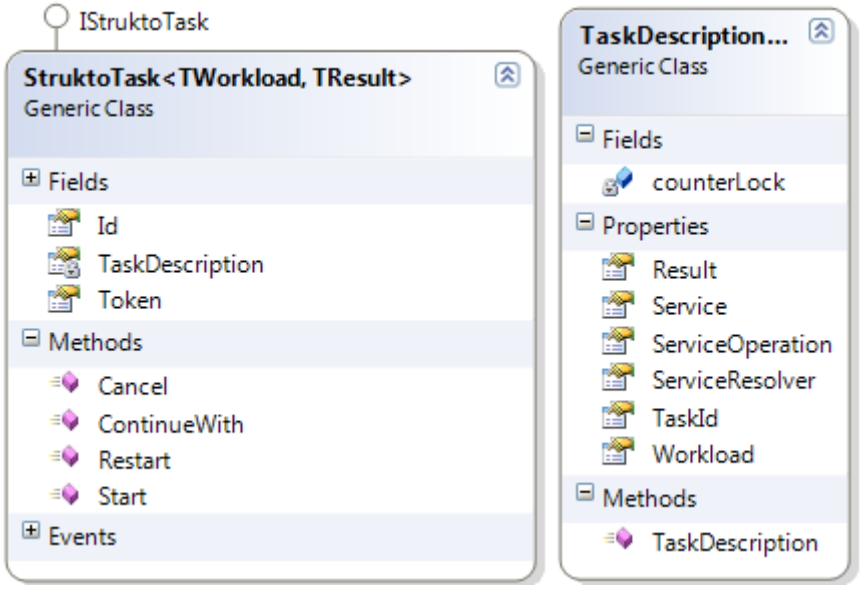


Abbildung 7 StruktoTask und TaskDescription

Ein StruktoTask besteht aus einer Instanz von StruktoTask und einer TaskDescription. Ohne die TaskDescription kann er dem TaskManager keine Auskunft darüber geben, was seine Aufgabe ist. Denn nur die TaskDescription weiss über den angesprochenen Service, die verlangte ServiceOperation und den dazugehörigen Workload. Die TaskDescription wird automatisch vom TaskManager beim Starten eines Tasks instanziiert.

3.4.1 TASKDESCRIPTION

Wie erwähnt kapselt die TaskDescription die wichtigsten Informationen eines StruktoTasks.

```
class LockCounter { public static int Counter { get; set; } }

public class TaskDescription<TWorkload, TResult> {

        private readonly object counterLock = new object();

        public TaskDescription(Func<ServiceOperation,StruktoServiceBase> serviceResolver)
        {
            lock (counterLock) { TaskId = ++LockCounter.Counter; }
            ServiceResolver = serviceResolver;
        }

        public int TaskId { get; set; }
        public ServiceOperation ServiceOperation { get; set; }
        public TResult Result { get; set; }
        public TWorkload Workload { get; set; }


        public Func<ServiceOperation, StruktoServiceBase> ServiceResolver { get; set; }
        public StruktoServiceBase Service { get { return ServiceResolver(ServiceOperation); } }
}
```

Listing 6 Klasse TaskDescription mit ServiceResolver

Im obigen Listing ist der ServiceResolver und das Service-Property hervorzuheben. Das ServiceProperty stellt die Information über den angesprochenen Service bereit. Der dazu verwendete ServiceResolver muss vom StruktoTaskManager beim instantiieren der TaskDescription mitgegeben werden, weil nur er den ServiceContainer mit allen registrierten Services kennt. Siehe Kapitel 3.3.1

3.5 SERVICES / WRAPPER

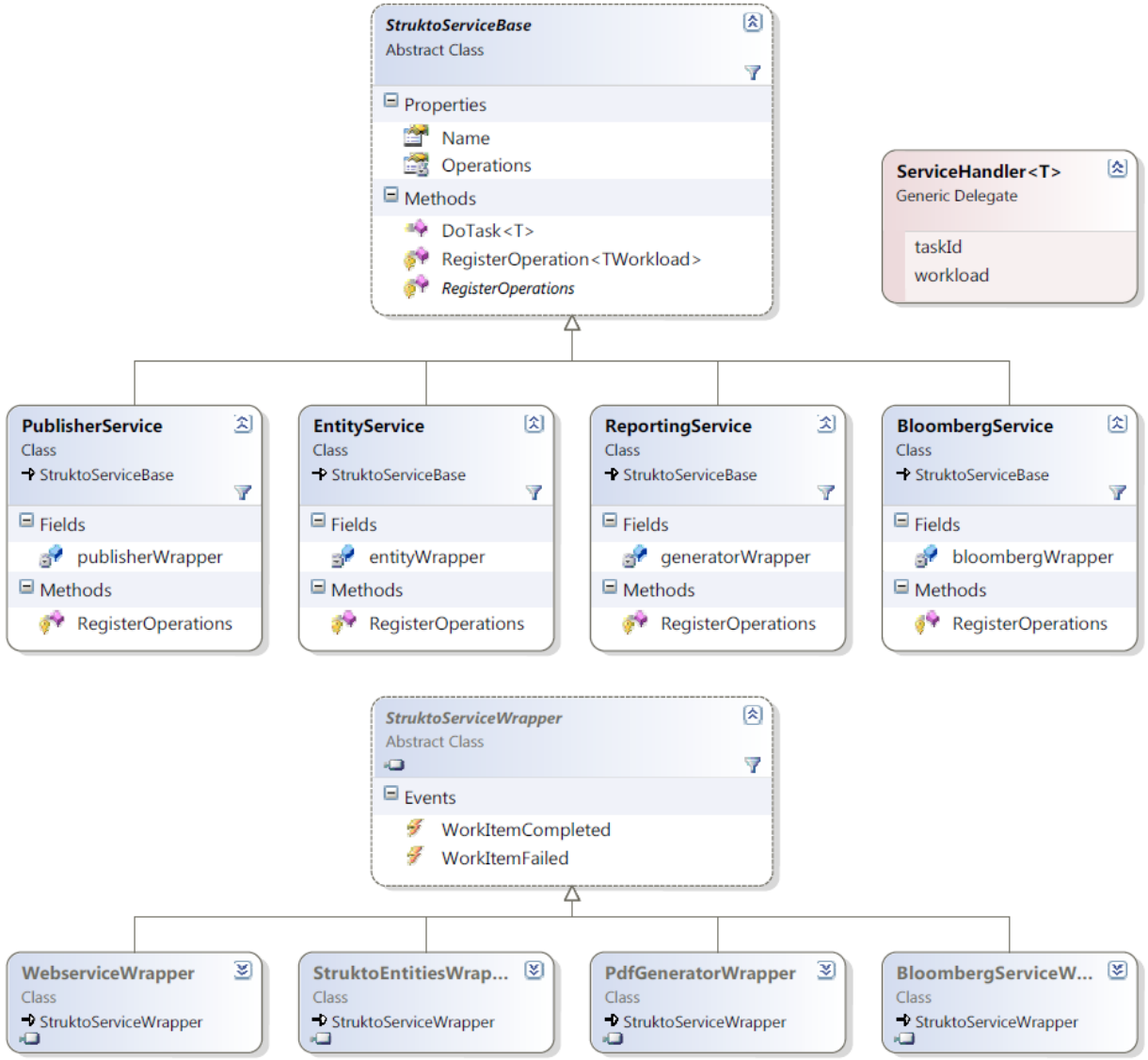


Abbildung 8 Best practice Klassenstruktur der Services

3.5.1 BASISKLASSE

Alle Services im StruktoManager haben die abstrakte Basisklasse *StruktoServiceBase*. Jeder Service ist für das Anbieten von Funktionalität selber verantwortlich. Die Basisklasse schreibt dem Service nicht vor wie er seine Funktionalität anbietet.

OPERATIONEN REGISTRIEREN

Sobald ein Service von *StruktoServiceBase* erbt kann er seine Funktionalität registrieren und anbieten. Da der UnityContainer „Operations“ private ist, müssen die Operationen über folgende Methode registriert werden.

```
protected void RegisterOperation<TWorkload>(ServiceOperation operation, ServiceHandler<TWorkload> handler)
{
        Operations.RegisterInstance(GetHandlerName<TWorkload>(operation), handler,
                new ContainerControlledLifetimeManager());
}
```

Die Methode erwartet als zweiten Parameter ein ServiceHandler-Delegate:

```
Public delegate void ServiceHandler<in T>(int 24asked, T workload);
```

Folgendes Listing zeigt wie für die Datentypen `Product` und `Event` die ServiceOperation `Publish` auf unterschiedliche Methoden des Wrappers registriert werden können.

```
protected override void RegisterOperations()
{
        RegisterOperation<Product>(ServiceOperation.Publish, publisherWrapper.DoPublishProduct);
        RegisterOperation<Event>(ServiceOperation.Publish, publisherWrapper.DoPublishEvent);
}
```

Die Methoden auf dem Wrapper sehen dann wie folgt aus.

```
public void DoPublishProduct(int taskId, Product p)
{
        //Do something here
}

public void DoPublishEvent(int taskId, Event e)
{
        //Do something here
}
```

DYNAMIC-DISPATCH PATTERN

Die DoTask-Methode auf *StruktoServiceBase* erwartet als einzigen Parameter ein IStruktoTask. Sie nutzt das Dynamic-Dispatch Pattern um vom Interface auf den konkreten Typ des übergebenen StruktoTasks zu schliessen. Im folgenden Listing wird gezeigt wie der Task im Service aufgenommen und von der Methode DoDynamicTask() in einen Task vom Typ StruktoTask<TWorkload, TResult> aufgelöst wird.

```
public void DoTask(IStruktoTask t)
{
        DoDynamicTask(t);
}

private void DoDynamicTask(dynamic t)
{
        ProcessTask(t);
}

private void ProcessTask<TWorkload, TResult>(StruktoTask<TWorkload, TResult> task) where TWorkload : class
```

```
{
        try
        {
                ResolveTask<TWorkload, TResult, TWorkload>(task);
        }
        catch (ResolutionFailedException)
        {
                HandleResolutionException(task);
        }
}
```

3.5.2 ENTITY SERVICE

PROBLEME

Ursprünglich verfolgten wir die Strategie mit Attached Entitities zu arbeiten, um das Lazy Loading auszukosten und möglichst einfaches Handling der Bearbeitung zu gewährleisten. Was wir aber zu diesem Zeitpunkt noch nicht berücksichtigten war, dass mehrere Bearbeitungen von Objekten innerhalb des Programmes gleichzeitig behandelt werden. Zb. Wenn zwei Produkte geöffnet sind und bei beiden Änderungen passieren soll es möglich sein, eines zu speichern und beim andern die Änderungen zu verwerfen. Dies wäre mit Attached Entities schwierig gewesen. Das EF bietet zwar eine Möglichkeit, alle Änderungen zu speichern (SaveChanges) aber das würde auch Objekte tangieren, die gerade im veränderten Zustand sind und später wieder zurückgesetzt werden sollten.

Also entschieden wir uns für die bestfunktionierende Variante, dass wir Detached Entities verwenden und einen Weg suchen, geänderte Daten mit den Original zu vergleichen; wir stiessen auf die Methode ApplyCurrentValues. Ausserdem musste es eine Möglichkeit geben, Objekte wieder im Originalzustand zu laden dafür bot sich die Methode GetObjectByEntityKey.

PERSISTIEREN VON DATEN

Da über alle Layers mit EntityOjbects gearbeitet wird suchten wir nach einer Möglichkeit, die CRUD Operationen auf dem Entity Framwork für alle Objekttypen anwenden zu können. Mittels des MetadataWorkspace vom EF und der Typeninformation werden die Parameterinformationen für den Entitätsunabhängigen Aufruf der AddObject Methode zusammengetragen.

```
public void AddItem(int taskId, EntityObject e)
{
        using (var saveContext = new StruktoEntities(GetConnectionString()))
        {
        var container = saveContext.MetadataWorkspace.GetEntityContainer(saveContext.DefaultContai
        nerName, DataSpace.CSpace);

        var entitySetName = (from meta in container.BaseEntitySets
                                where meta.ElementType.Name == e.GetType().Name
                                select meta.Name).First();
        saveContext.AddObject(entitySetName, e);
        saveContext.SaveChanges();
        }
}
```

Das Löschen der Objekte hat sich als weitaus einfacher herausgestellt.

```
public void DeleteItem(int taskId, EntityObject e)
{
    using (var saveContext = new StruktoEntities(GetConnectionString()))
    {
        saveContext.DeleteObject(e);
        saveContext.SaveChanges();
    }
}
```

Beim Update werden jeweils Änderungen mit dem Original aus der Datenbank verglichen und diese dann gespeichert. Da die Objekte mit all ihren abhängigen geladen sind vergleicht die Methode ApplyCurrentValues das eingehende Objekt mit dem gesamten entsprechenden „object tree“ aus der Datenbank. Die letzte Speicherung wird hierbei bevorzugt.

```
public void UpdateItem(int taskId, EntityObject e)
{
    using (var saveContext = new StruktoEntities(GetConnectionString()))
    {
        var container = saveContext.MetadataWorkspace.GetEntityContainer(saveContext.DefaultContai
        nerName, DataSpace.CSpace);
        var entitySetName = (from meta in container.BaseEntitySets
                             where meta.ElementType.Name == e.GetType().Name
                             select meta.Name).First();

        EntityObject o = (EntityObject)saveContext.GetObjectByKey(e.EntityKey);
        saveContext.ApplyCurrentValues(entitySetName, e);
        saveContext.SaveChanges();
    }
}
```

Abbildung 9 EntityService: UpdateItem

RELOAD DER DATEN

Objekte werden im Originalzustand an die Datenbank geliefert und als Callback Argumente dem aufrufenden Task übergeben

```
public void ReloadEntityObject(int taskId, EntityObject e)
{
    using (var context = new StruktoEntities(GetConnectionString()))
    {
        var container = context.MetadataWorkspace.GetEntityContainer(context.DefaultContainerName,
                                            DataSpace.CSpace);
        var entitySetName = (from meta in container.BaseEntitySets
                             where meta.ElementType.Name == e.GetType().Name
                             select meta.Name).First();

        EntityObject o = (EntityObject)context.GetObjectByKey(e.EntityKey);
        InvokeWorkItemCompleted(taskId, (EntityObject)o);
```

```
        }
    }
```

LADEN DER DATEN

Für das Laden müssen zwar noch individuelle Methoden erzeugt werden, da man das rechenintensive Abfragen der Datenbank gezielt nur auf benötigte Bereiche anwenden will.

```
    private void LoadCustomersData(StruktoEntities context)
    {
        customers = context.Customers
                .Include(e => e.Sales)
                .ToObservableCollection();
    }
```

3.5.3 CHANGE TRACKING

Jede Entität feuert zwar einen PropertyChanged Event bei einer Änderung, aber unser Ziel war es ja innerhalb eines geöffneten Tabs alle Änderungen festzustellen und dies auf dem Hauptobjekt des Baumes festzustellen. Folgendes Interface und dessen Implementation löste unsere Probleme. Eine Basisklasse mit der entsprechenden Funktionalität einzufügen, war nicht möglich, da die Klassen bereits von EntityObject erben.

```
public interface IDependantTracking
{
    event PropertyChangedEventHandler DependantChanged;
    void SetUpChangeTracking();
    bool IsChanged { get; set; }
}
```

SETUPCHANGETRACKING

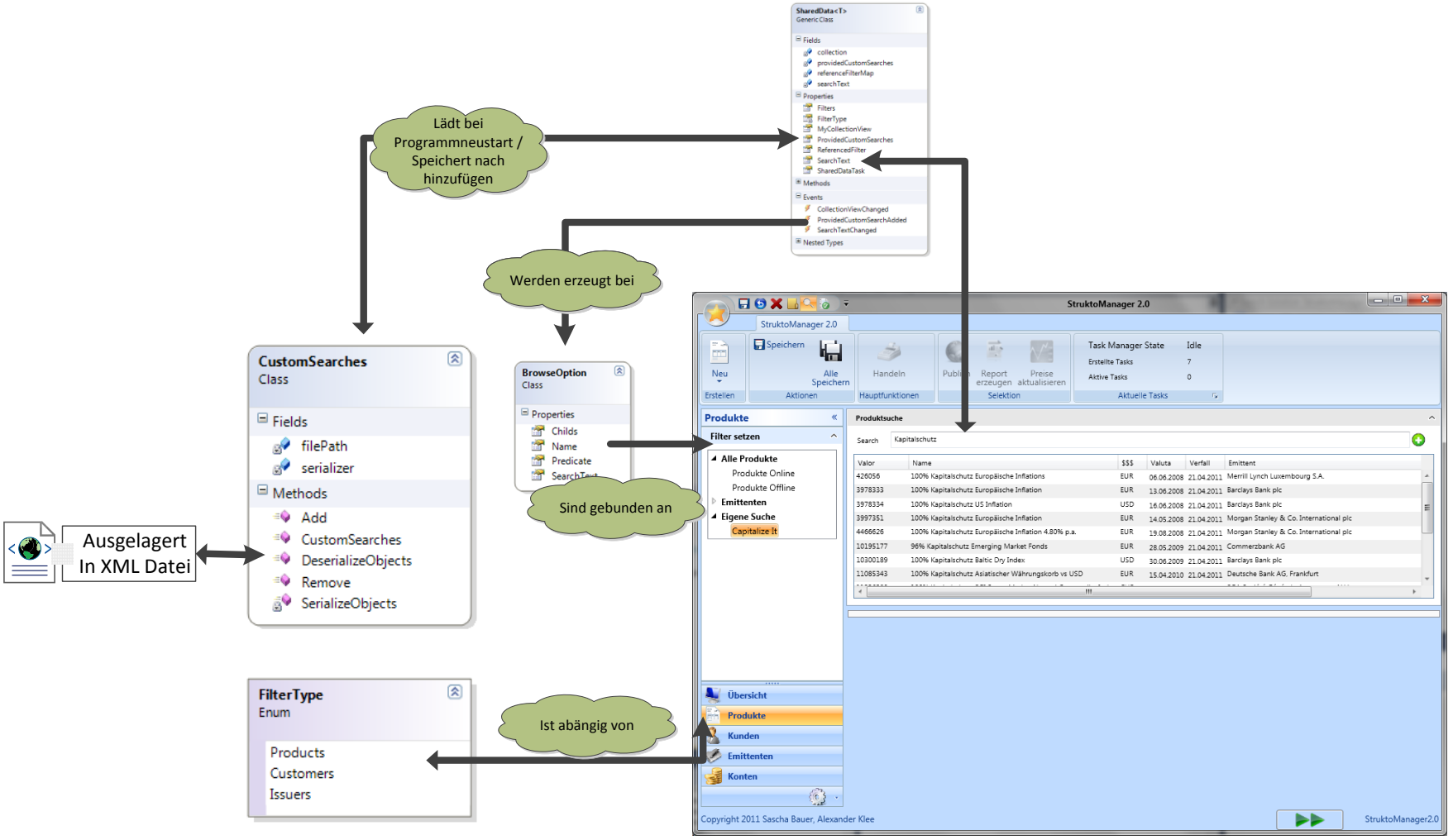
Das Hauptobjekt und alle abhängigen Objekte, deren Änderungen uns interessieren, müssen den PropertyChanged Event auf dem Hauptobjekt fangen.

```
    public void SetUpChangeTracking()
    {
        PropertyChanged += OnProductChanged;
        BaseInformation.PropertyChanged += OnProductChanged;
        foreach (var baseValue in BaseValues)
        {
            baseValue.PropertyChanged += OnProductChanged;
        }
    }
```

Innerhalb von OnProductChanged muss IsChanged auf true gesetzt werden und der DependantChanged Event gefeuert werden.

```
private void OnProductChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)
{
    if (DependantChanged == null) return;
    IsChanged = true;
    DependantChanged(this, e);
}
```

3.6 USERINTERFACE



3.6.1 MAINCONTROL

Abbildung 10 MainControl

Die gesamte Programmansicht ist abhängig von der gewählten Sektion im MainControl. Auch werden Commands im Ribbon Control abhängig der gewählten Sektion gebunden.

3.6.2 SHAREDDATA<T>

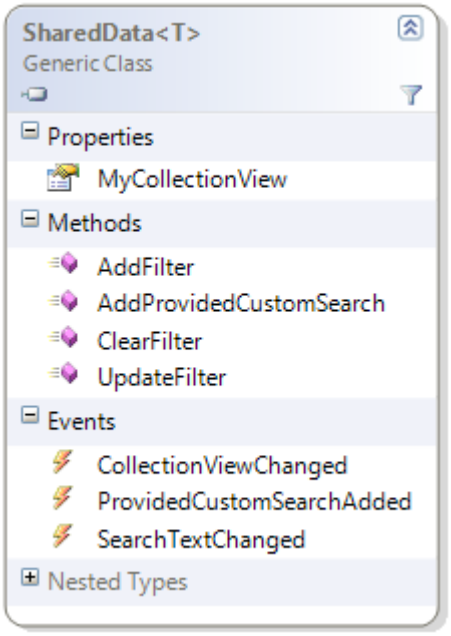


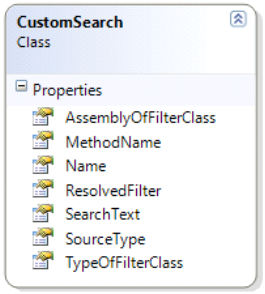
Abbildung 11 Wichtigste Properties, Methoden und Events der SharedData

Die SharedData-Klasse erstellt im Konstruktor einen Task. Dieser versucht den übergebenen Template-Parameter T von der Datenbank zu laden. Sobald der Task den Completed-Event wirft, wird das Task-Resultat in die SourceCollection von MyCollectionView geschrieben und Refresh aufgerufen. Die Controls welche MyCollectionView referenzieren erhalten somit die geladenen Daten unmittelbar.

3.6.3 BROWSECONTROL

ADD CUSTOM SEARCH

Um eine spezifische Such zu speichern muss einerseits der Suchtext und das Prädikat, welches durch das BrowseControl gesetzt wird gespeichert werden. Prädikate können nicht serialisiert werden deshalb blieb keine andere Wahl als die Information zu speichern, wo diese Prädikat Projekt auffindbar ist, um es mit Reflection wieder aufzulösen. Dabei werden die Parameter**: Name des Assembly, Name der Methode und Typ der Klasse** benötigt. Enpsrechend muss das zu serialisierende Objekt attributiert werden.




```
[Serializable]
public class CustomSearch {

    [XmlAttribute]
    public string SearchText { get; set; }
```

Dies kann wie folgt wieder mit Reflection aufgelöst werden.

```
[XmlIgnore]
public Predicate<object> ResolvedFilter
{
    get
    {
        try
        {
            \*1 Hier wird der Typ aufgelöst [typname, assembly]
            Type t = Type.GetType(string.Format("{0},{1}", TypeOfFilterClass, AssemblyOfFilterClas
                s));

            \*2 Hier wird die Methode aufgelöst
            MethodInfo mi = t.GetMethod(MethodName);

            \*3 Anhand einer MethodInfo lässt sich mit dieser statischen Methode wieder ein
                Predicate erzeugen
            return (Predicate<object>)Delegate.CreateDelegate(typeof(Predicate<object>), mi);
        }
        catch(Exception e)
        {
            return null;
        }
    }
}
            return null;
        }
    }
}
```

NEUE SUCHTYPEN HINZUFÜGEN

Beim Laden des entsprechenden BrowseModels muss dessen BrowseOptions Property geladen werden.

Ein Parent Ist immer ein Titel und beinhaltet bei der Suche alle Objekte, die Childs davon sind wieder BrowseOptions mit gesetztdem Predicate und oder SearchText

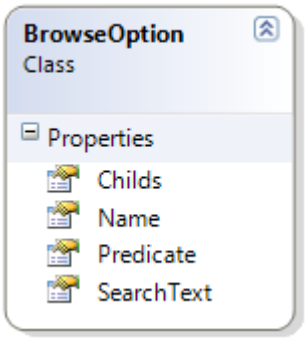


Abbildung 12 BrowseOption

3.6.4 SEARCHCONTROL

Die Sucheingabe des SearchControls kann die Datenansicht im DataGrid filtern. Ausserdem können gleich mehrere Tabs auf einmal Markierung setzen und mit Enter bestätigen oder nur eines mit Doppelklick geöffnet werden.

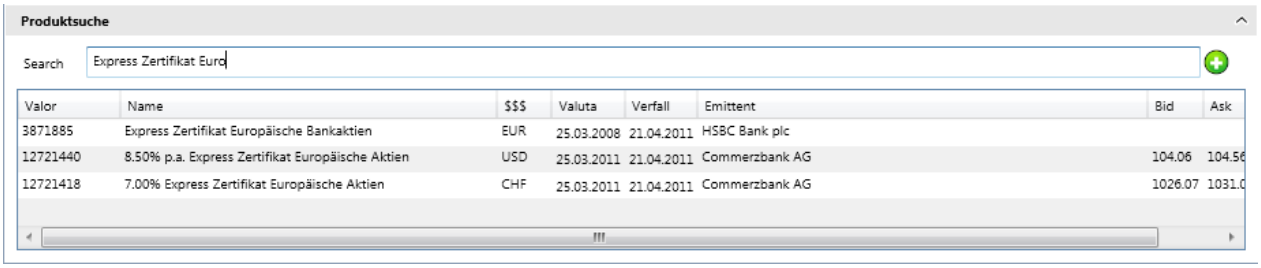


Abbildung 13 SearchControl der Produkte

Die eingegebene Suche lässt sich mit dem grünen Button rechts vom Eingabefeld speichern.

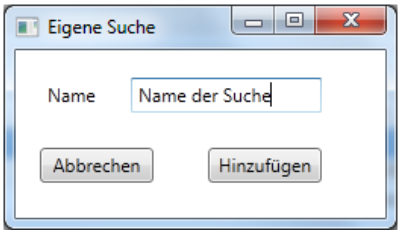


Abbildung 14 Eigene Suche hinzufügen

Anschliessend ist die Suche im BrowseControl eingebunden und Serialisiert.

3.6.5 DETAILSCONTROL

Repräsentiert die Detailansicht der Rohdaten dargestellt in verschiedenen Expander. Jeder Expander beinhaltet ein weiteres User Control.

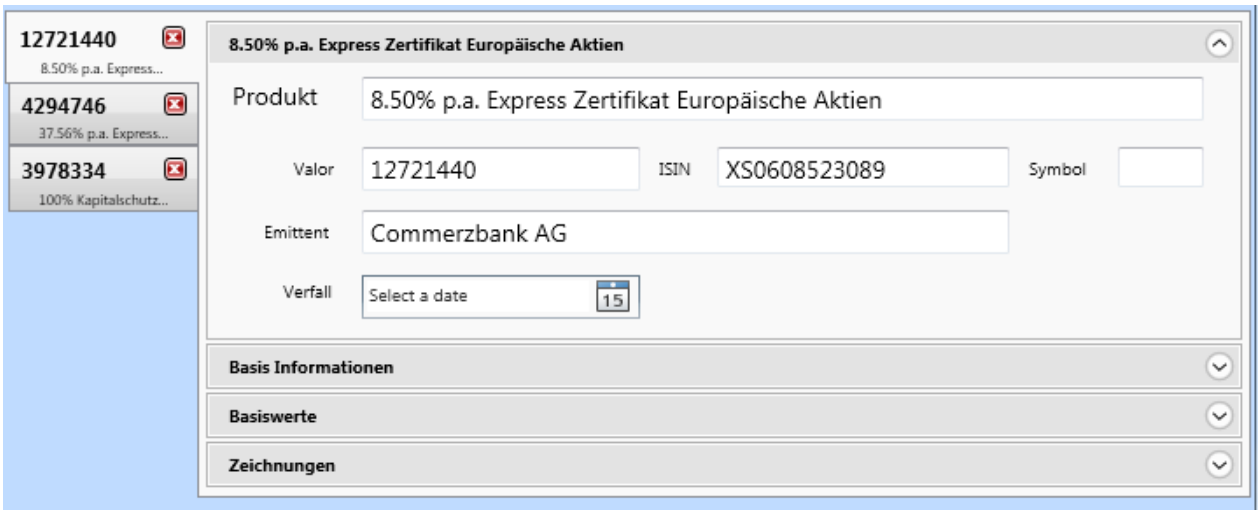


Abbildung 15 DetailsControl der Produkte

Es können mehrere Tabs gleichzeitig geöffnet sein, jedes Tab für sich stellt fest, ob daran etwas geändert wurde oder nicht. Wird es geschlossen und es ist geändert, hat der Benutzer die Chance zu speichern oder die Änderungen zu verwerfen. Genaueres siehe Software Architecture Document.

3.7 NICHT ERREICHTE PUNKTE

Aus Zeitgründen gab es einige Punkte die in dieser Arbeit nicht umgesetzt werden konnten. Folgende Punkte fehlen

3.7.1 VALIDIERUNG

Die komplette und fehlerfreie Anbindung der User Interfaces ist zum jetzigen Entwicklungsstand nicht gewährleistet. Wir kamen nicht dazu zu testen, ob auch alle Felder in der Datenbank korrekt gespeichert werden.

Die Implementierung der Validierung von Usereingaben musste im Zuge dessen ebenfalls gestrichen werden, da ohne Tests keine Garantie auf eine korrekte Implementation bestand. Wichtig war, dass alle Möglichkeiten für das CRUD bereitstehen und umgesetzt werden können. Wir haben den Fokus auf die Struktur der Applikation gelegt damit die IT-Mitarbeiter der Bank wissen, wie neue Entitäten ins Programm eingebaut werden können.

3.7.2 BUCHHALTUNGSLOGIK

Die Buchhaltungslogik wurde nicht implementiert.

3.7.3 KONTEN

Geplant waren neben den Produkten und Kunden auch die Kontenverwaltung umzusetzen. Aufgrund der fehlenden Buchhaltungslogik musste dieser Punkt gestrichen werden.

3.7.4 TEST IM GESCHÄFTSUMFELD

Diese Tests hätten aufgrund der unfertigen Applikation keinen Sinn gemacht.

4 SCHLUSSFOLGERUNGEN

4.1 TASKING

4.1.1 TASK.FACTORY.STARTNEW VS. STRUKTOTASK

.Net stellt mit dem Tasking-Framework bereits ein sehr mächtiges Werkzeug für asynchrone Datenverarbeitung zur Verfügung. Wenn möglich sollte also direkt ein Task aus der .Net-Library verwendet werden um asynchrone Aufrufe zu kapseln. Sobald jedoch ein simples Interface für das Ansprechen komplexer Funktionalität gebraucht wird, eignet sich das StruktoTasking hervorragend.

4.1.2 ANWENDUNGSBEISPIEL

REGISTRIERUNG EINES SERVICES ZUR LAUFZEIT

```
private class MyOwnService : StruktoServiceBase
{
     public ServiceOperation MyOperation =
          new ServiceOperation<MyOwnService>("MyOperation", ServiceMode.Async);

     protected override void RegisterOperations()
     {
          RegisterOperation<Action>(MyOperation, MyHandler);
     }

     private void MyHandler(int taskId, Action myAction)
     {
          myAction.Invoke();
          InvokeWorkItemCompleted(new ServiceEventArgs(taskId));
     }
}

private void RegisterAtRuntime()
{
     StruktoTaskManager.Default.RegisterService<MyOwnService>();
     StruktoTaskManager.Default.StartOperation(
           () => Console.WriteLine("I did it!"),
           MyOwnService.MyOperation
     );
}
```

4.1.3 MANUELLE REGISTRIERUNG DER SERVICES

Soll der Default-TaskManager um einen neuen Service - der bereits beim Start des TaskManagers registriert wird – erweitert werden, so muss man den Service in der RegisterServices() Methode „manuell“ registrieren. Am Beispiel unseres obigen Services sieht das wie folgt aus.

```
private void RegisterServices()
{
        services = new UnityContainer();
        RegisterService<EntityService>();
        RegisterService<ReportingService>();
        RegisterService<PublisherService>();
```

```
        RegisterService<BloombergService>();
        RegisterService<MyOwnService>();
}
```

4.1.4 AUSBLICK: AUTOMATISCHE REGISTRIERUNG

Reflection bietet mit Attribut-Tags eine Möglichkeit um Methoden und KJassen mit Meta-Informationen zu versehen. In Zukunft könnten die ServiceKlassen sowie die Wrapper-Methoden mit einem entsprechenden Attribut versehen werden. Darüber wäre der StruktoTaskManager in der Lage sämtliche Services automatisch zu registrieren:

```
// Für ServiceKlassen
[IsStruktoService(typeof(StruktoEntitiesWrapper))]
public class EntityService : StruktoServiceBase
{

// Für Wrapper-Methoden
[IsWrapperMethod("AddItem", typeof(EntityObject))]
public void AddItem(int taskId, EntityObject e)
{
```

4.1.5 HINZUFÜGEN EINER NEUEN SERVICEOPERATION

DIE KLASSE SERVICEOPERATION UND SERVICEOPERATION<T>

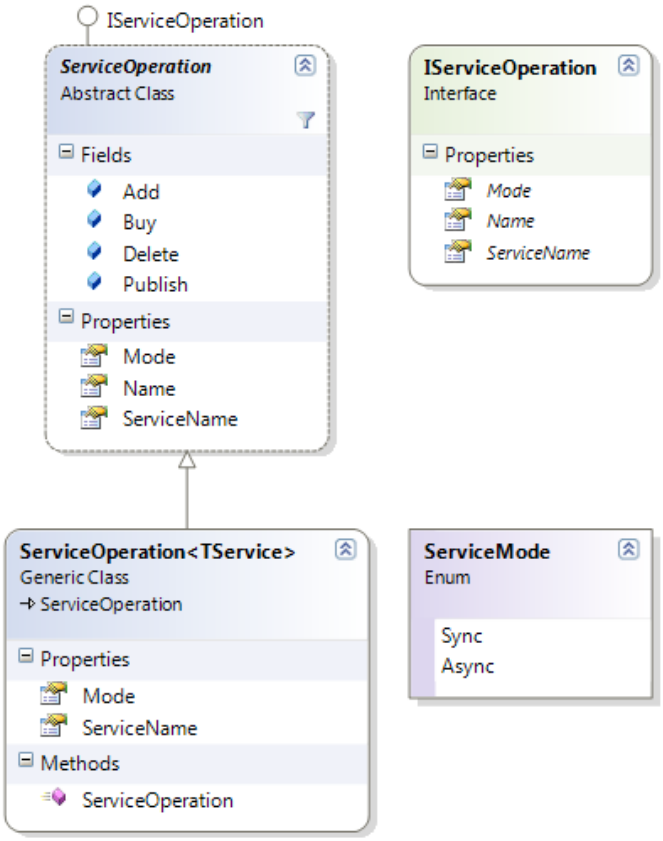


Abbildung 16 ServiceOperations

Die Klasse ServiceOperation bietet alle ServiceOperationen (Add, Buy, Delete, Publish, etc.) als Felder an.

DEFINITION EINER OPERATION

```
public abstract class ServiceOperation : IServiceOperation
{
        public static readonly ServiceOperation Add =
                new ServiceOperation<EntityService>("Add", ServiceMode.Sync);
        ...
}
```

REGISTRIERUNG DER OPERATION

```
public  class EntityService : StruktoServiceBase
{
        ...

        protected override void RegisterOperations()
        {
                RegisterOperation<EntityObject>(ServiceOperation.Add, entityWrapper.AddItem);
                ...
```

```
        }
}
```

4.1.6 ERWEITERN EINER BESTEHENDEN SERVICEOPERATION

ERWEITERN != NEUE OPERATION

Der Vorteil von ServiceOperationen ist, dass auf eine ServiceOperation mehrere DatenTypen registriert werden können. Der OperationContainer löst die Wrapper-Methode nach WorkloadTyp UND ServiceOperation des StruktoTasks auf. Es muss also nicht für jeden neuen Datentyp für den man auf einem Service eine Methode anbieten will eine neue ServiceOperation definiert werden. Das Registrieren der ServiceOperation auf den neuen Datentyp reicht dafür aus.

REGISTRIERUNG EINES NEUEN DATENTYPS FÜR EINE OPERATION

```
public  class EntityService : StruktoServiceBase
{
        ...

        protected override void RegisterOperations()
        {
                RegisterOperation<EntityObject>(ServiceOperation.Add, entityWrapper.AddItem);
                RegisterOperation<SomeOtherObject>(ServiceOperation.Add, entityWrapper.AddOtherItem);
                ...
        }
}
```

4.1.7 HÄUFIGSTER FEHLER BEIM AUFLÖSEN EINER SERVICEOPERATION

Das Auflösen von Kindklassen bei registrierter Basisklasse funktioniert nicht implizit. Beim auflösen des OperationHandlers wird vom UnityContainer eine ResolutionException geworfen, weil die Kindklasse nicht registriert wurde.

Will man nun aber nicht für alle Kind-Klassen eine eigene Operation registrieren sondern eine „Basisfunktion“ für alle abgeleiteten Klassen erstellen muss man die Methode HandleResolutionException der StruktoServiceBase wie folgt erweitern.

```
private void ProcessTask<TWorkload, TResult>(StruktoTask<TWorkload, TResult> task) where TWorkload : class
{
        try
        {
                ResolveTask<TWorkload, TResult, TWorkload>(task); //This won’t work for childclasses
        }
        catch (ResolutionFailedException)
        {
                HandleResolutionException(task);
        }
}

private void HandleResolutionException<TWorkload, TResult>(StruktoTask<TWorkload, TResult> task)
{
        try
        {
                ...
```

```
                if (GetType().Equals(typeof(YourBaseClass)))
                {
                        ResolveTask<TWorkload, TResult, YourBaseClass>(task);
                        return;
                }
                ...
        }
        ...
}
```

4.2 NEUER MAINCONTROL BUTTON: EVENTS

Folgendes Kapitel zeigt wie eine neue Datensicht für den Bereich „Events“ eingefügt werden kann.

Abbildung 17 Neue MainControl Button "Event"

4.2.1 VORBEREITUNGEN

MAINVIEW ERWEITERN

Wir erweitern die MainView damit sie das obige Aussehen erhält

```
<odc:OutlookSection x:Name="eventSection" Header="Events"
                    Click="StruktoSectionEvents_Click" Image="../_Images/Basic/Events_32.png">
        <odc:OdcExpander Header="Filter setzen"
                         Margin="0" Background="White" ShowEllipse="False" IsExpanded="True">
                <StackPanel>
                        <EventManagement:EventBrowseControl />
                </StackPanel>
        </odc:OdcExpander>
</odc:OutlookSection>
```

Im Code-behind der MainControl.xaml fügen wir den EventsClickHandler ein:

```
 private static readonly EventContentControl eventContentControl = new EventContentControl();

        private void StruktoSectionEvents_Click(object sender, RoutedEventArgs e)
        {
            OpenEventsMainView();
        }

        private void OpenEventsMainView()
        {
            ChooseMainViewType(
                eventContentControl,
                Enums.MainUserViewType.Events,
                true,
```

```
                MainControlLocator.MainViewModelStatic.CreateEventCommand);
        }
```

Das CreateEventCommand muss noch im MainViewModel.cs erstellt werden.

```
        public ICommand CreateEventCommand
        {
            get
            {
                createEventCommand =
                    new RelayCommand(
                        () => Messenger.Default.Send(
                                new OpenCreateWindowMessage(Enums.MainUserViewType.Events)));
                return createEventCommand;
            }
        }
```

NEUER FILTER TYP

Da wir die Events später in der BrowseControl filtern wollen, erweitern wir den FilterType- und MainUserViewType-Enum um einen entsprechenden Eintrag.

```
Namespace StruktoManager.UserInterface.Common.DataSharing
{
    public enum FilterType
    {
        Products,
        Customers,
        Issuers,
        Events  //Insert this to identify the EventFilter
    }
}
```

```
namespace StruktoManager.UserInterface.Common
{
    public class Enums
    {
        public enum MainUserViewType
        {
            Overview,
            ApplicationSettings,
            Products,
            Customers,
            Accounts,
            Issuers,
            Events //Insert this to identify define the ViewType
        }
    }
}
```

EVENT-TYP UM IDEPENDANTTRACKING ERWEITERN

Als nächstes erstellen wir im Package „Database“ des StruktoManager.Artefacts.Local Projekts eine neue Klasse „Event“.

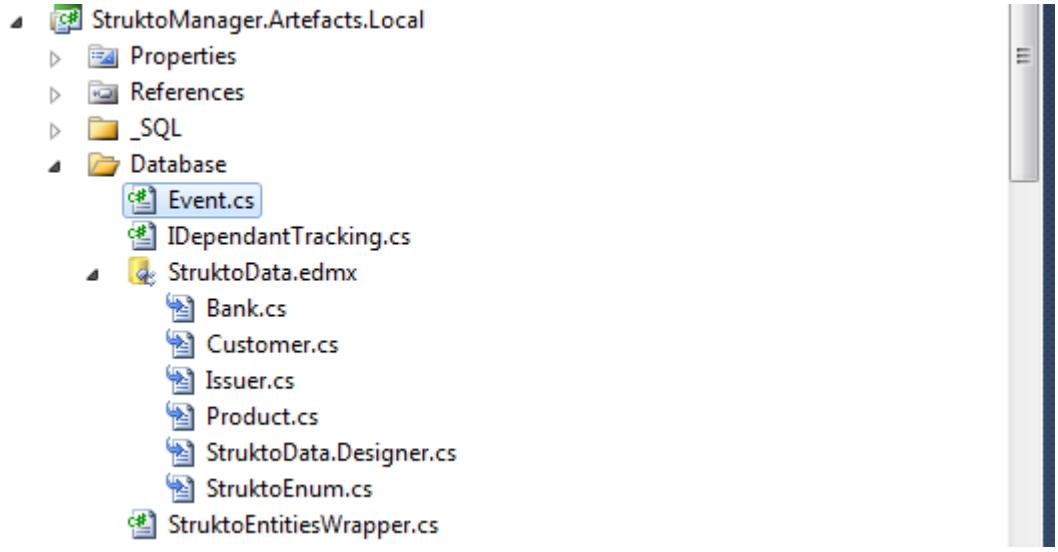


Abbildung 18 Database-Package vorher

Nun öffnen wir das StruktoManager.Artefacts.Local.csproj im Editor und ändern die Zeile

```
<Compile Include="Database\Event.cs" />
```

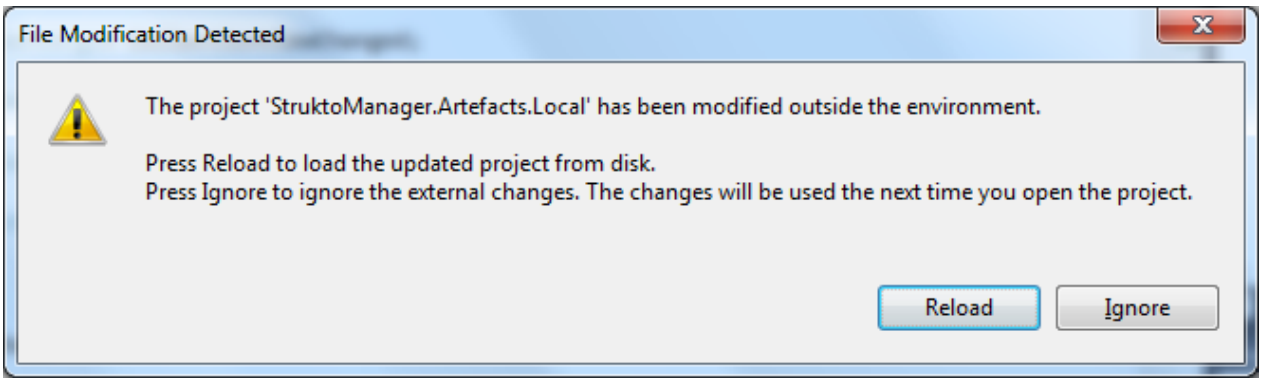
Abbildung 19 .csproj vorher

wie folgt:

```
<Compile Include="Database\Event.cs">
  <DependentUpon>StruktoData.edmx</DependentUpon>
</Compile>
```

Abbildung 20 .csproj nachher

Nach dem Speichern wechseln wir zurück ins VisualStudio und bestätigen folgende Meldung mit „Reload“.



Anschliessen sieht das Projekt so aus:

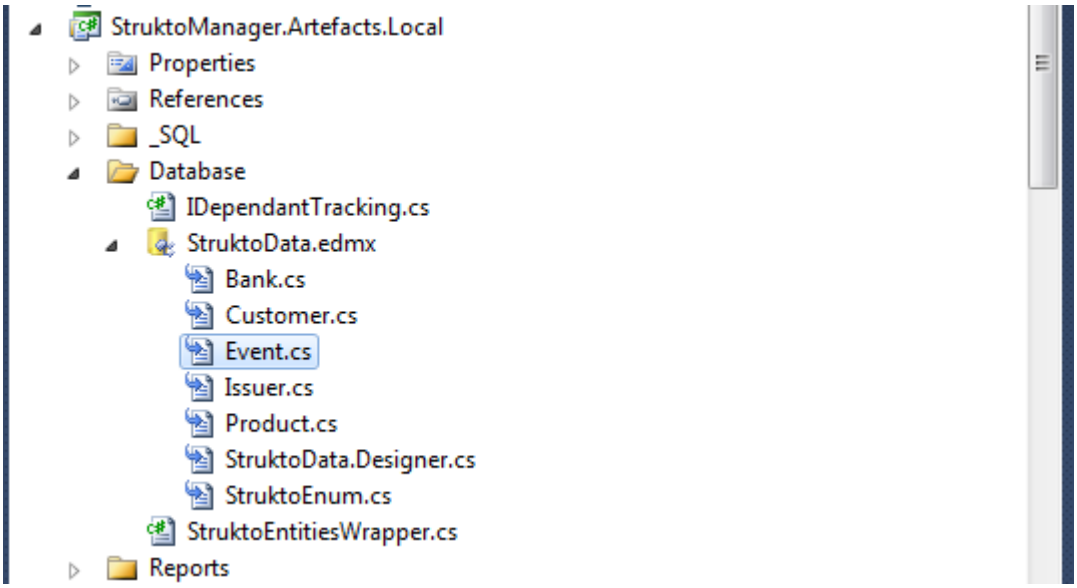


Abbildung 21 Database-Package nachher

Bevor wir mit dem erstellen der Model-Klassen beginnen können muss die Klasse „Event“ das IDependantTracking Interface implementieren:

```
namespace StruktoManager.Artefacts.Local.Database {
    partial class Event : IDependantTracking
    {
        public event PropertyChangedEventHandler DependantChanged;
        public void SetUpChangeTracking()
        {
            throw new NotImplementedException();
        }

        public bool IsChanged
        {
            get { throw new NotImplementedException(); }
            set { throw new NotImplementedException(); }
        }
    }
}
```

EVENTS AUF ENTITYSERVICEWRAPPER LADEN

Da wir alle Events von der Datenbank laden wollen, müssen wir den EntityServiceWrapper um die gewünschte Funktionalität erweitern.

```
        private ObservableCollection<Event> events = new ObservableCollection<Event>();

        private void LoadEventsData(StruktoEntities context)
        {
            events = context.Events
                .Include(e => e.Product)
                .Include(e=>e.EventTyp)
```

```
                .ToObservableCollection();
        }

        private void LoadAllData()
        {
            using (var context = new StruktoEntities(GetConnectionString()))
            {
                LoadProductsData(context);
                LoadIssuers(context);
                LoadEnums(context);
                LoadCustomersData(context);
                LoadBanksData(context);
                LoadEventsData(context);
            }
        }

        public void LoadEvents(int taskid, ObservableCollection<Event> t)
        {
            InvokeWorkItemCompleted(taskid, events);
        }
```

EVENT AUF ENTITYSERVICE REGISTRIEREN

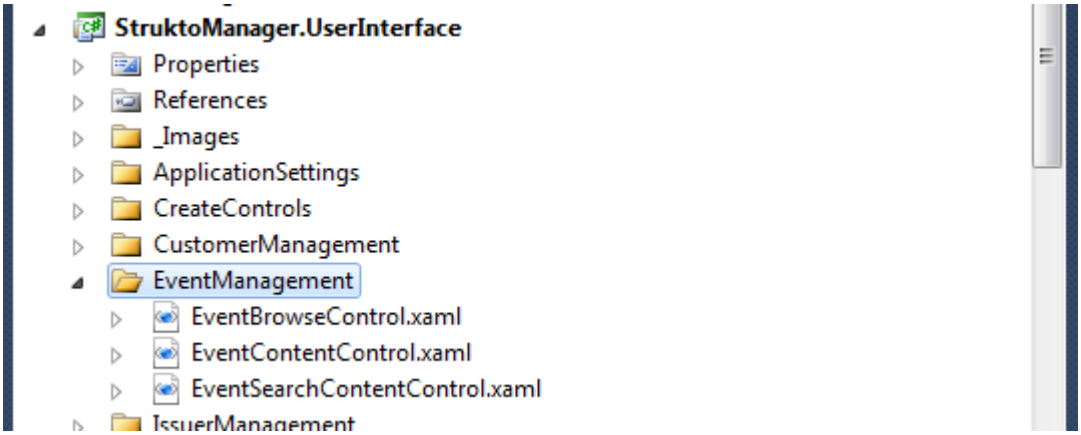
Damit unser EntityService die Events laden kann, registrieren wir die Funktionalität in seiner RegisterOperations-Methode.

```
Public  class EntityService : StruktoServiceBase
{
        public EntityService() : base(EntityWrapper) { }

        protected override void RegisterOperations()
        {
                //...
                RegisterOperation<ObservableCollection<Event>>(
                        ServiceOperation.Load,
                        EntityWrapper.LoadEvents);
        }
    }
```

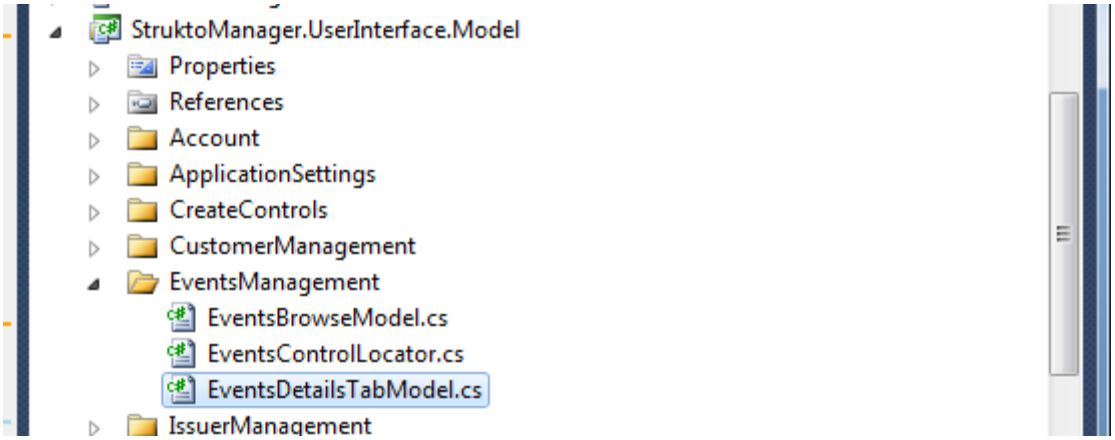
4.2.2 ERSTELLEN DER BENÖTIGTEN VIEWS

Wir erstellen 3 leere WPF-UserControls.



4.2.3 ERSTELLEN DER BENÖTIGTEN MODELS

Danach werden die ViewModels erstellt.



Die drei Klassen brauchen folgende Signaturen

```
namespace StruktoManager.UserInterface.Model.EventsManagement
{
        public class EventsBrowseModel : BrowseModelBase<Event> {
                public EventsBrowseModel(SharedData<Event> sharedData) : base(sharedData) { ... }
        }

        public class EventsControlLocator : SharedDataControlLocator<Event> {
                public EventsControlLocator() : base(FilterType.Events) { ... }
        }

        public class EventDetailsTabModel : DetailsTabModelBase<Event> {
                public EventDetailsTabModel(Event myEvent) : base(myEvent) { ... }
        }
}
```

4.2.4 VORBEREITUNGEN FÜR DAS EINBINDEN DER USERCONTROLS

Bevor wir mit dem einbinden der Controls beginnen können, müssen wird den EventsControlLocator im App.xaml als statische Ressource anbieten.

```
<!—Namespace einbinden -->
xmlns:EventsManagement="clr-
namespace:StruktoManager.UserInterface.Model.EventsManagement;assembly=StruktoManager.UserInterface.Model"

<!—Statische resource definieren -->
<Application.Resources>
        <EventsManagement:EventsControlLocator x:Key="EventsControlLocator“ />
```

```
...
```

4.2.5 EINBINDEN DER BROWSECONTROL

Jetzt können wir auf dem EventsControlLocator ein MVVMLocatorProperty vom Typ EventsBrowseModel. Dieses werden wir an den DataContext der BrowseControl binden.

```
        private static EventsBrowseModel eventsBrowseModel;

        public static EventsBrowseModel EventsBrowseModelStatic
        {
            get
            {
                if (eventsBrowseModel == null)
                {
                    CreateEventsBrowseModel();
                }

                return eventsBrowseModel;
            }
        }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Performance",
            "CA1822:MarkMembersAsStatic",
            Justification = "This non-static member is needed for data binding purposes.")]
        public EventsBrowseModel EventsBrowseModel
        {
            get
            {
                return EventsBrowseModelStatic;
            }
        }

        public static void ClearEventsBrowseModel()
        {
            eventsBrowseModel.Cleanup();
            eventsBrowseModel = null;
        }

        public static void CreateEventsBrowseModel()
        {
            if (eventsBrowseModel == null)
            {
                eventsBrowseModel = new EventsBrowseModel(SharedData);
            }
        }

        public static void Cleanup()
        {
            ClearEventsBrowseModel();
        }
```

Wichtig: Der Konstruktor erwartet eine SharedData<Event>. Diese bietet unser EventsControlLocator über das Property SharedData an.

Das EventBrowseControl.xaml erhält nun seinen DataContext und den entsprechenden Inhalt:

```
<UserControl x:Class="StruktoManager.UserInterface.EventManagement.EventBrowseControl"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:i="clr-
namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity" mc:Ignorable="d"
             DataContext="{Binding EventBrowseModel, Source={StaticResource EventsControlLocator}}"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <StackPanel>
            <StackPanel.Resources>
                <HierarchicalDataTemplate x:Key="ChildTemplate" >
                    <TextBlock Text="{Binding Path=Name}" />
                </HierarchicalDataTemplate>
                <HierarchicalDataTemplate x:Key="NameTemplate"
                            ItemsSource="{Binding Path=Childs}"
                            ItemTemplate="{StaticResource ChildTemplate}">
                    <TextBlock Text="{Binding Path=Name}" FontWeight="Bold" />
                </HierarchicalDataTemplate>
            </StackPanel.Resources>
            <TreeView x:Name="treeView" ItemsSource="{Binding BrowseOptions}" ItemTemplate="{StaticResourc
e NameTemplate}">
                <i:Interaction.Triggers>
                    <i:EventTrigger EventName="SelectedItemChanged">
                        <i:InvokeCommandAction Command="{Binding SelectedNodeChangedCommand}" CommandParam
eter="{Binding SelectedItem, ElementName=treeView, Mode=OneWay}"/>
                    </i:EventTrigger>
                </i:Interaction.Triggers>
            </TreeView>
        </StackPanel>
    </Grid>
</UserControl>
```

4.2.6 EINBINDEN DER SEARCHCONTROL

Für die Suche erhält der EventControlLocator ein weiteres LocatorProperty vom Typ SearchControlModel<Event>.

Wieder braucht der Konstruktor des SearchControlModels unsere SharedData Property vom EventsControlLocator.

```
        public static void CreateSearchControlModel()
        {
            if (searchControlModel == null)
            {
                searchControlModel = new SearchControlModel<Event>(SharedData);
            }
        }
```

4.2.7 EINBINDEN DER DETAILSCONTROL

Ein letztes LocatorProperty wird für die DetailsControl benötigt. Der Datentyp der EventDetailsControl ist

```
DetailsControlModel<Event,EventDetailsTabModel>
```

Somit ist das Einbinden der UserControls beendet. Die Bindings auf das MainControl funktionieren und die leeren UserControls könne mit Inhalt gefüllt werden.

Projekt: StruktoManager 2.0

Projektplan

5 EINFÜHRUNG

5.1 ZWECK

Dieses Dokument beschreibt den Prozess der Entwicklung des StruktoManagers2.0.

5.2 GÜLTIGKEITSBEREICH

Dieses Projekt dient als Grundlage für die Semesterarbeit „StruktoManager2.0“ und hat Gültigkeit über die gesamte Projektdauer.

5.3 DEFINITIONEN UND ABKÜRZUNGEN

siehe doc/05_PMQM/Glossar.docx

5.4 REFERENZEN

ProjektWiki :	http://sa.tweak.ch
Projektphasen:	http://sa.tweak.ch/Projektphasen
Bloomberg API :	http://sa.tweak.ch/Bloomberg

5.5 ÜBERSICHT

Nachfolgend wird der Auftraggeber der Arbeit sowie die zu überarbeitende Software „StruktoManager1.x“ vorgestellt und auf die Organisationsstruktur des Teams während dieser SA eingegangen. Im Kapitel „Management Abläufe“ findet sich eine detaillierte Iterationsplanung inkl. Meilensteinen des Projekts. Im Abschnitt „RisikoManagement“ findet sich die Risiko-Analyse mit dem Risikovermeidungsstrategien. Anschliessend folgt eine Auflistung der bisher geplanten Arbeitspaketen und der verwendeten Infrastruktur. Abschliessend werden die Qualitätsmassnahmen beschrieben, welche die höchstmögliche Softwarequalität für den Auftraggeber sicherstellen soll.

6 PROJEKTÜBERSICHT

6.1 AUFTRAGGEBER

Cat Group AG (Holding)

- Triaxis Trust AG
    - Cat Financial Products
        - **Giuliano Glocker**
        - Patrick Walker
        - Julia Savina

6.2 EINFÜHRUNG

Als Spezialist im Bereich Strukturierter Produkte und massgeschneiderten Anlagelösungen bietet Cat Financial Products (CatFP) unabhängige Beratung, Brokerage- und Asset Management Dienstleistungen

an. Zur Verwaltung der strukturierten Produkte setzt CatFP seit 2008 den StruktoManager ein. Die Software wurde eigens für diesen Zweck von der IT-Abteilung der Triaxis Trust AG entwickelt.

In den vergangenen drei Jahren hat sich der Funktionsumfang der Software stark erweitert und mit ihm hat auch die Komplexität zugenommen.

Damit der StruktoManager auch in Zukunft wart- und erweiterbar bleibt, bat CatFP die HSR den StruktoManager zu analysieren. Neben der Überarbeitung der Architektur soll die bestehende WinForms-Technologie durch WPF ersetzt werden. Um die gleiche Funktionalität wie in der bestehenden Software sicherzustellen, soll eine umfangreiche Testumgebung entwickelt werden.

6.3 STRUKTO MANAGER 1.X

Siehe doc/02_AnalyseUndDesign/StruktoManager/IstAnalyse.docx

6.4 PROJEKTZIELE

Die Projektziele lassen sich in folgende Kategorien unterteilen:

Ziel	Beschreib
Technologie	Die eingesetzten Technologien sollen auf den neuesten Stand gebracht werden. (.Net-Framework 4.0)
Architektur	Es soll eine Desktop-Applikation entworfen werden deren Architektur die Schichten klar voneinander trennt. Desweiteren sollte die Architektur so geplant werden, dass daraus leicht eine Client/Server Architektur gemacht werden kann.
Module	Die Business-Logik soll in kleine Module gegliedert werden. Die Module sollen über Schnittstellen Informationen untereinander austauschen können.
Testumgebung	Als Nebenprodukt der Arbeit wird eine funktionierende Testumgebung entwickelt.

Tabelle 2 Projektziele

6.5 FEATURE LISTE

Feature	Beschreib	Art
Produkte erfassen	Es soll weiterhin möglich sein, sämtliche Details eines Produkts zu erfassen.	Required
Kunden erfassen	Um ein Produkt handeln zu können, muss es erst im Primärmarkt veräussert (gezeichnet) werden.	Required
Asynchrones UI	Das UserInterface darf bei keiner Operation blockieren.	Required

Trading	Primär- und Sekundärmarkttrades müssen wie anhin funktionieren. Im BusinessCase von Cat Financial Products ist genau beschrieben wie das Trading zu funktionieren hat.	*Optional*
Veröffentlichung	Ein gezeichnetes Produkt kann im Sekundärmarkt veröffentlicht werden und wird so für potentielle Käufer sichtbar.	*Optional*
Handel während Lebenszeit	Sobald ein Produkt gezeichnet wurde, kann im Sekundärmarkt damit gehandelt werden.	*Optional*
Basis-Beobachtung	Basiswerte eines Produktes sollen beobachtet werden können.	*Optional*
Barrieren-Beobachtung	Auf Produkten können Barrieren erfasst werden. Sobald diese über- /unterschritten werden soll der Benutzer benachrichtigt werden.	*Optional*
Bloomberg-Felder	In den Produktdetails sollen einzelne Felder mit der Bloomberg-API verbunden werden können. Diese Felder werden vom StruktoManager2.0 automatisch geupdated.	*Optional*
Synchronisation der Produktevents ins Outlook	Events die auf den Produkten erfasst wurden, erscheinen automatisch auch im Outlook des Benutzers.	*Optional*
Upload-Report via Email	Nach erfolgtem Report Upload schickt der StruktoManager2.0 dem Benutzer ein Mail mit einer Übersicht des letzten Updload-Prozesses. Somit werden die Benutzer schnellstmöglich über etwaige Fehler informiert.	*Optional*
Sprache einstellen	Die Applikation soll Internationalisierung unterstützen. Die Sprache der Oberfläche soll zur Laufzeit gewechselt werden können.	*Optional*

Tabelle 3 Featureliste

7 PROJEKTORGANISATION

7.1 PROJEKTTEAM

Alexander Klee	Sascha Bauer

Tabelle 4 Projektteam

7.2 EXTERNE SCHNITTSTELLEN

Wer	Organisation	Was
Hansjörg Huser	HSR	Projektbetreuer
Giuliano Glocker	CatFP	Auftraggeber
Patrick Walker	CatFP	Projektmanager „StruktoManager2.0“
Julia Savina	CatFP	Benutzerin

Tabelle 5 Externe Schnittstellen

8 MANAGEMENT ABLÄUFE

8.1 AUFWANDSCHÄTZUNG

Die Semesterarbeit dauert vom 21.02.2011 bis 03.06.2011. Jedes Mitglied arbeitet pro Woche 18 Stunden. Sollte es Probleme bei der Umsetzung der Ziele geben, werden die Requirements, in Absprache mit dem Auftraggeber, gekürzt.

8.2 ITERATIONEN UND MEILENSTEINE

Iteration	Endet am	Resultat

Inception 1	27.02.2011	**Projektplanung**: • Die Anforderungen an die neue Software wurden mit dem Auftraggeber abgeklärt und ein Architekturentwurf ausgearbeitet. • Zeitplan ist erstellt • Entwicklungsumgebung auf Arbeitsrechnern installiert
Elaboration 1 **MS1: Projektsetup**	06.03.2011	**Projektplanung**: • Die Projektplanung wurde mit dem Betreuer verifiziert. **Projekt-Dokumentation**: • Dokumentvorlagen erstellt • Konfigurationsverwaltung eingerichtet • Domain Model begonnen • Analyse StruktoManager1.x begonnen • Sequenzdiagramme begonnen • Qualitätsmassnahmen definiert • Festlegung der Technologien/Architektur **Testumgebung**: • StruktoManager1.x läuft in Testumgebung **Architekturprototyp:** Projekttemplate eingerichtet
Elaboration 2 **MS2: Architekturprototyp**	26.03.2011	**Projekt-Dokumentation**: • Domainmodel abgeschlossen • Klassendiagramm abgeschlossen • Sequenzdiagramme abgeschlossen **Architekturprototyp:** Die Kernfunktionalität wurde in die Architektur integriert. **Prototyptests:** UnitTests untersuchen das Verhalten des Architekturprototyps.
Construction 1	03.04.2011	**Software Architektur Dokument:** Die letzten Korrekturen am SAD sind im Prototyp umgesetzt. Beta-Phase der Software beginnt. **Businesskomponenten:** Die Kernkomponenten des Business Layers sind implementiert und kommunizieren durch alle Schichten hindurch. **Tasking Tests:** Das korrekte Verhalten der Tasks wird mit UnitTests überprüft und sichergestellt.
Construction 2 **MS3: Business Regeln**	21.04.2011	**Businesskomponenten:** Das erstellen von Kunden und Produkten funktioniert.
Construction 3 **MS4: Release Candidate**	13.05.2011	**Release Candidate:** Die Infrastruktur für den Release Candidate wird beim Auftraggeber vorbereitet.
Transition 1	22.05.2011	**Release Candidate:** Der Release Candidate wird beim Kunden übergeben und erfüllt alle Anforderungen.

Projektabschluss **MS5: Abgabe**	03.06.2011	**Abgabe:** Die Dokumentation der Semesterarbeit wird dem Betreuer übergeben. **StruktoManager2.0:** Mit dem Abschluss des Projekts, wird dem Auftraggeber das Projekt Repository übergeben.

Tabelle 6 Projektplan

8.2.1 ARBEITSPAKETE

Siehe doc/05_PMQM/ProjektCockpit.xlsx

9 RISIKOMANAGEMENT

9.1 RISIKOANALYSE

KM **Kosten der Massnahmen (h)**
S_max Maximaler Schaden (h)
S_gem **Gewichteter Schaden (h)**
P Eintrittswahrscheinlichkeit

Id	Risiko	Auswirkung	Massnahme	KM	S_max	P	S_gew	Prio
R05	Implementierung der Businesslogik wird unterschätzt	Die Applikation kann nicht in vollem Umfang implementiert werden.	Features müssen gestrichen werden	-	-	70%	-	-
R04	Migration der Daten kann nicht komplett umgesetzt werden.	Die Applikation kann nicht in vollem Umfang implementiert werden..	Implementierung der Datenbank auf wichtigste Tabellen beschränken.	-	-	70%	-	-
R01	Komplexität der eingesetzten Technologien wird unterschätzt.	Das Projekt gerät in Zeitverzug	Projekt nach Pflicht-Zielen und optionalen Zielen ausrichten, dass bei unerwartet hohem Aufwand optionale Ziele gestrichen werden können	2	48	15%	7.2	Mittel
R06	Aufwand für das Redesign der Datenbank wird unterschätzt. Das Migrations-Script erreicht einen zu hohen Komplexitätsgrad	Das Projekt gerät in Zeitverzug	Erst eine Liste erstellen mit allen gewünschten Änderungen, zu komplexe Änderungen abstreichen	1	24	30%	7.2	Mittel
R03	Projekt tangiert Bereiche der Infrastruktur und Prozesse von Geschäftspartner die fest bestehen und nicht geändert werden können (z.B. Webhosting)	Die Arbeit kann nicht vollständig eine Eigenentwicklung sein, sondern muss angepasst werden. Kann nicht von Grund auf neu Konzipiert werden (auch wenn optimaler wäre)	Systemgrenzen müssen klar definiert sein. Rollen der Stakeholder müssen in Projekt involviert werden.	2	24	20%	4.8	Mittel
R02	Bestehende Infrastruktur der Bank unterstützt eingesetzte Technologien nicht	Projekt kann nicht erwartungsgemäss ausgerollt werden	Infrastruktur(Server) bei Zeiten analysieren und Austausch/Update Massnahmen beizeiten einleiten	4	8	35%	2.8	Mittel

Tabelle 7 Risikoanalyse

10 INFRASTRUKTUR

10.1 ENTWICKLUNGSUMGEBUNG

Was	Womit
StruktoManager2.0	Visual Studio 2010, WPF-Application, Entity Framework, (PRISM Framework)
STRUKTO Datenbank	SQL Server 2008
Versionisierung	SVN (siehe 7.4 Backup)
Dokumentation	Dropbox0.7.110

Tabelle 8 Entwicklungsumgebung

10.2 REQUIREMENTS / ISSUE TRACKING

Requirements und Issues fliessen im Verlaufe des Projektes in unser Zeitmanagement Instrument, die Exceldatei ZeitManagement.xlsx. Dort werden alle Arbeitspakete erfasst, der Aufwand geschätzt und man hat eine stetige Übersicht über den Projektstatus und die Planungseffizienz. Ausserdem dient dieses Dokument dazu einzelne Zeiteinträge pro Person und Arbeitspaket zu erstellen.

10.3 KOMMUNIKATION

Bevorzugte Kommunikation ist der direkte Dialog. Weiter benützen wir für Studenten gängige Kommunikationsmittel wie E-Mail, Skype, Facebook, Mobiltelefone und das ProjektWiki.

10.4 BACKUP

Source Code und Datenbankscripts (somit auch der Zustand der Datenbank) werden nach jeder Arbeitseinheit ans SVN Repo „COMMITTED“.

Zusätzlich liegt der Source der letzten Iteration in der Dropbox, welche ebenfalls über ein Versionierungsfeature verfügt.

Lokal werden grundsätzlich keine Daten gehalten. Somit ist ein komplettes Backup unseres Projektes sichergestellt.

11 QUALITÄTSMASSNAHMEN

11.1 DOKUMENTATION

Unsere Dokumentation wird von jedem Teilnehmer nachgeführt und stimmt immer mit der aktuellen Situation überein. In erster Linie gilt die Dokumentation, dann der Code.

Es werden im Code bei komplexen Fragmenten Kommentare angebracht. Alle Klassenschnittstellen werden mit „triple slash“ im Code dokumentiert.

11.2 SITZUNGSWESEN

Im Sitzungswesen haben sich folgende Dokumente bewährt.

Dokument	Zweck
Traktanden	Dienen als Wegweiser in der Sitzung und sind ein gutes Instrument zur pragmatischen Sitzungsführung / -Vorbereitung
Protokoll	Während den Sitzungen werden die wichtigsten Punkte aufgeschrieben. Daraus lassen sich später die Beschlüsse ableiten.
Beschlüsse	Die Beschlussliste kann aus den Traktanden und dem Sitzungsprotokoll extrahiert werden. Die Beschlüsse werden in neue bzw. vorhandene Arbeitspakete integriert.

Tabelle 9 Sitzungsdokumente

Sitzungen mit dem Projektbetreuer finden stets im Wochenrhythmus statt. Sitzungen mit CatFP werden individuell, je nach Projektstatus vereinbart.

Die Sitzungsprotokolle werden dem Betreuer per Dropbox zur Verfügung gestellt.

11.3 PROJEKT- UND ZEITPLAN AKTUALISIEREN

Die Arbeitszeiterfassung geschieht über

Bis Sonntagabend müssen jeweils alle geleisteten Arbeitszeiten eingetragen sein.

11.4 TODO-LISTEN

Im ZeitManagement.xlsx kann immer herausgelesen werden, was es zu tun gibt.

11.5 KNOW-HOW SHARING

Technologie –Wissen soll stets auf dem Projektwiki platziert werden, damit sich Auftraggeber und Projektmitarbeiter stets informieren können.

11.6 REVIEWS

11.6.1 DOKUMENTREVIEW

Nach Abschluss jeder Iteration werden die Dokumente resultierend aus den Arbeitspaketen gereviewt. Anschliessend wird dies in der Dokument-Änderungsgeschichte eingetragen.

11.6.2 CODE REVIEW

Werden gemäss CodeReview.docx durchgeführt. Die Projektmitglieder reviewen jeweils den Code des andern. Nach jeder Iteration sobald Code geschrieben wurde. Daraus resultiert jeweils ein kurzer Bericht gemäss Checkliste im Word Dokument

11.7 VERSIONSVERWALTUNGSSYSTEM

Der Source Code des Projekte, sowie Datenbankscripts werden auf einem SVN-Server gespeichert. Prinzipiell gelten die folgenden Regeln: Wird ein komplexes Problem von nur einer Person bearbeitet, so wird ein eigener Branch dafür erstellt. Ansonsten wird auf demselben Branch gearbeitet. Sobald ein Feature fertig implementiert ist oder einen stabilen Stand erreicht hat, soll auf einem neuen Branch gearbeitet werden.

Das Repository befindet sich unter folgendem Pfad: svn://svns.hsr.ch/StruktoManager

11.8 TESTS

11.8.1 UNIT-TESTS

Während der Entwicklung werden, wo sinnvoll, Unit Tests durchgeführt. Nach jeder Iteration müssen alle Tests erfolgreich durchlaufen. Es soll eine Code Coverage von mindestens 80% erzielt werden.

11.8.2 SYSTEM-TESTS

Auf Basis der UseCases werden, in der Transition Phase, die System-Tests durchgeführt und protokolliert. Diese überprüfen ob die Software den funktionalen Anforderungen entspricht. Diese Tests sind BlackBox-Tests und sollen 100% der Use Cases abdecken.

11.8.3 BUSINESS-TESTS

Die Business-Tests bilden den Kern der System-Tests. Sie überprüfen hauptsächlich die programmierte Business-Logik der Trading Use Cases. Da dies für den Auftraggeber kritische Anforderungen sind, wird den Business-Tests besondere Beachtung geschenkt.

11.8.4 USABILITY-TESTS

Nachdem in der Construction 2 der RC1 veröffentlicht wurde, werden Usability-Tests bei Cat Financial Products durchgeführt. Diese Tests geben uns wertvollen Feedback in Bezug auf das neue GUI-Design. Da sich auch die Menüführung leicht ändern wird, geben diese Tests den Mitarbeitern von Cat Financial Products auch die Möglichkeit, sich bereits an die neuen Arbeitsabläufe zu gewöhnen.

Projekt: StruktoManager 2.0

Anforderungsspezifikation

12 EINFÜHRUNG

12.1 ZWECK

Da wir eine bestehende Applikation überarbeiten, setzt sich die Anforderungsspezifikation an die bestehende Software. Die Anforderungen werden individuell im Gespräch mit den Partnern aus der Cat Financial Group ausgehandelt. Dieses Dokument hält lediglich wichtiges Know How fest, das während diesen gesprächen entstanden ist.

13 ANFORDERUNGEN

Die bestehenden Funktionalitäten des StruktoManagers ist im Dokument *doc/02_AnalyseUndDesign/IstAnalyse.docx* beschrieben und wird durch folgende Anforderungen erweitert.

Anforderung	Beschreib
F01: Non-Blocking UI	Alle Funktionen die aus dem GUI heraus aufgerufen werden, müssen Non-Blocking sein. *[…] in the .NET Framework 4, tasks are the preferred API for writing multi-threaded, asynchronous, and parallel code. (MSDN1)*
F02: Asynchrone Services pausieren / unterbrechen	Alle asynchronen Services (Operationen die der StruktoManager für den Benutzer ausführt) müssen auf dem GUI ersichtlich und pausier- / abbrechbar sein. *The System.Threading.Tasks.Task […] classes support cancellation through the use of cancellation tokens, which are new in the .NET Framework 4.(MSDN2)* Die Services sind: - Publishing (Veröffentlichen von Produktinformationen), - Reporting (Generieren von Performance Reports) - Trading (Handeln von Produkten) - Requesting (Abfragen von Bloomberg-Daten)
F03: User durch Validierung und Formatierung seiner	

Eingaben unterstützen

Tabelle 10 Anforderungen

14 BUSINESS CASES

14.1 HANDELN IM PRIMÄRMARKT

Nr.	Arbeitsschritt	Einfluss auf StuktoManager
1	Cat FP stellt eine neue Produkt Idee zusammen	Kein Einfluss
2	Cat FP kontaktiert verschiedene Banken (Emittenten), um die besten Konditionen für sein Produkt zu erlangen	Kein Einfluss
3	Cat FP kontaktiert seine Kunden mit den erlangten Konditionen (Preis, Coupon usw.), um möglichst viele Kundengelder für die Lancierung eines Produktes zusammen zu tragen (Zeichnungsperiode).	Kein Einfluss
4a (Normalfall)	Cat FP lanciert das Produkt bei dem Emittenten mit dem besten Preis/Service (Anfangsfixierung)	1 Tag nach dem Anfangsfixierungsdatum wird das Produkt im StruktoManager erfasst, da zu diesem Zeitpunkt die Valorennummer (Identifikation des Produktes) vorhanden ist. Es wird also ein neues Produkt über die Valorennummer im StruktoManager erfasst.
4b (Spezial-fall)	(Dies würde dazu dienen, um das Produkt mit den angestrebten Konditionen bereits auf der Homepage zu veröffentlichen, um so weitere Kunden(gelder) anzuziehen)	Cat FP erfasst das Produkt vor dem Anfangsfixierungsdatum im StruktoManager. In diesem Fall soll das Produkt anhand einer fiktiven (temporären) Valorennummer erfasst werden. Die fiktive Valorennummer soll mit der richtigen Valorennummer überschrieben werden, sobald diese vorhanden ist. Vorsicht: Die Valorennummer gilt bislang als Identifikation in vielen Tabellen der Datenbank.
5	Kauf des Produktes, um es anschliessend an den Kunden zu verkaufen (Die Verkaufsmenge wurde bereits zuvor mit den Kunden abgemacht. Cat FP kauft somit das fixierte Produkt vom Emittenten um es anschliessend den Kunden zu verkaufen. Cat FP schlägt auf den Verkaufspreis seinen	Cat FP erfasst den Kauf des Produktes im StruktoManager. Der Bestand des Produktes wird erhöht und der Preis wird erfasst.

	Komissionsanteil drauf).	
6	Verkauf des Produktes im Primärmarkt.	Cat FP erfasst die Verkäufe des Produktes im StruktoManager. Der Gesamtbetrag des Verkaufes ist schon bei der Fixierung des Produktes bekannt. Die Details der Verkäufe (Gegenpartei usw.) werden jedoch erst ca. 2 Tage nach dem Fixierungsdatum bekannt. Der Bestand des Produktes nimmt ab.
7	Die Periode nach dem Anfangsfixierungsdatum aber vor dem Valuta Datum nennt man Graumarkt. Ein Verkauf im Graumarkt wird ganz normal als Verkauf im Primärmarkt im StruktoManager erfasst. Dies ist der Fall, wenn Cat FP nach der Anfangsfixierung nochmals vom Produkt dazu kauft (normalerweise zu einem anderen Preis).	Cat FP erfasst den erneuten Kauf im StruktoManager. Der Bestand des Produktes wird wieder erhöht sowie der neue Preis (wichtig für die spätere Berechnung) erfasst. Ebenfalls werden die erneuten Verkäufe erfasst (berechnet anhand des neuen Preises).
8	Das Produkt beginnt nach dem Valuta Datum (ca. 2 Wochen nach Anfangsfixierung) zu leben. Das Produkt befindet sich nun im Sekundärmarkt.	Sämtliche Verkäufe in dieser Phase werden als Sekundärmarkt Trades erfasst. Die Berechnungen im Sekundärmarkt sind identisch zu den Berechnungen im Primärmarkt, jedoch benötigen sie andere Zusatzinformationen zu den Verkäufen.
9	Rückzahlungs-Datum = Ablauf des Produktes, Produkt wird zurückbezahlt	Kein Einfluss

Tabelle 11Handeln im Primärmarkt

Auf weitere BusinessCases wurde aufgrund der hohen Komplexität der weiteren BusinessCases verzichtet.

Projekt: StruktoManager 2.0

Software Architecture Document

15 DATENBANK ANALYSE

15.1 ANALYSE IST

Beschreibt die Ist Situation der aktuellen Datenbank. Zeigt als Resultat einer ausführlichen Analyse alle Unschönheiten auf, wie und in welchem Umfang diese behoben werden.

15.1.1 MÄNGELLISTE GROB

STATISCHE TABELLEN

Folgende Tabellen verwalten statische Werte, sie werden wie Enum Werte eingesetzt. Ausserdem enthalten diese zum Teil Einträge mit `'n/A'` oder `''` leerem String. Es muss zwischen verschiedenen Datenkonstruktionen unterschieden werden (mit 3 Spalten, mit 2 Spalten, referenziert von mehreren andern Tabellen, einmalig referenziert oder gar nicht referenzierte).

```
-- TBLANLAGEKLASSE
-- TBLAUSLÖSER
-- TBLBEOBACHTUNG
-- TBLCALL_DATEN
-- TBLEVENT_TYP
-- TBLGESCHÄFT_ART
-- TBLPROKLASSIFIZIERUNG
-- TBLPROSTATUS
-- TBLSCHUTZTYP
-- TBLSETTLEMENT
-- TBLTERMIN_ART
-- TBLVERRECHNUNGSSTEUER
-- TBLWÄHRUNGSRISIKO
-- TBLZEICHNUNGS_ART
-- TBLCOUPON_PERIODE
-- TBLBLOOMBERG_TYP
-- TBLPREISANGABE
-- TBLPRODUKTE_KAT
-- TBLPRODUKTE_TYP
-- TBLWÄHRUNGEN
```

1 ZU 1 BEZIEHUNGEN

Diverse Tabllen enthalten identisch Anzahl Rows, daraus lässt sich schliessen, dass es sich um eins zu eins Beziehungen handelt. Um dies auch im Entity Framework einfach umzusetzen müssen diese Beziehungen auch in der Datenbank so abbgebildet werden. Ausserdem hat man so eine höhere Gewährleistung auf auf Korrektheit der gespeicherten Daten.

Betroffene Tabellen:

- tblBasis_informationen
- tblZeichnungen
- tblEmission
- tblGenerelle_informationen
- tblProdukt
- tblStruktur

TABELLE KUNDEN

Eine Adresse befindet sich im

ALLGEMEINE SCHWACHSTELLEN

- Nur gerade die Primary Keys sind als NOT NULL Attribute definiert
- TBLACCOUNTS_AUSGANG
  - sek Markt ID ist immer -1 wenn auf nichts referenziert wird
    Ausserdem ist die Beziehung zu tblWährungen nicht definiert
  - ✓ bemerkung, leere Einträge sollten NULL sein
  - ✓ status sollte vom Datentyp Bit sein
- TBLACCOUNTS_AUSGANG_BETRÄGE Die Beziehung zu tblWährungen ist nicht definiert Werte der Spalte tblWährungen_id verweisen aber darauf
- TBLACCOUNTS_EINGANG
  - sek Markt ID ist immer -1 wenn auf nichts referenziert wird
  - ✓ Ausserdem ist die Beziehung zu tblWährungen nicht definiert
  - ✓ bemerkung, leere Einträge sollten NULL sein
  - Es wird über eine Spalte definiert, auf welche Tabelle eine andere Spalte referenzieren soll → am besten einfach zwei Felder erzeugen
- TBLACCOUNTS_EINGANG_BETRÄGE
  - ✓ Die Beziehung zu tblWährungen ist nicht definiert Werte der Spalte tblWährungen_id verweisen aber darauf
- tblBanken
  - Spalten r_moody, r_sp, r_fitch stellen ein Rating dar, wird nicht überall gemacht, hier wäre Potential für eine neue Tabelle
  - enthält ausserdem ein unnötiges `'n/A'` Feld
  - ✓ name_kurz, leere Einträge, sollten NULL sein
  - es gibt einen leeren Eintrag
- tblBarrieren
  - ✓ manuell, wurde als char(1) definiert, sollte bit sein
- tblBasis_informationen
  - ✓ produkt_online, produkt_in_zeichnung, indikative_bedingungen, sollte bit sein
  - ✓ emissionstyp, bloomberg_ticker, produkt_valor, wo leer NULL
- tblBasis_werte
  - ✓ anfangswert enthält nutzloses `'n/A'` Daten, setze auf NULL
- tblEmission

- ✓ schlussfixierung_existiert, sollte bit sein

- tblEmittentenliste
  - ✓ hasSp, hasMoody, hasFitch, Werte umkehren, sind falsch, anschliessend zu [bit] machen
  - ✓ Ausserdem sind alle Felder die mit ticker… beginnen zu NULLEN
- tblEvents
  - ✓ beschreibung, perioden_name → wo leer NULL
- tblGenerelle_informationen
  - ✓ deutschland, ***sekundärmarkt***, einkommenssteuer, tblVerrechnungssteuer_id, tblEu_zins_id → wo leer NULL
- tblGraumarkt
  - ✓ existiert sollte [bit] sein
- tblKauf
  - ✓ bemerkung wo leer, NULL
- tblKennzahlen
  - ✓ isin, symbol, börsenplatz wo leer, NULL
- tblKunden
  - ✓ alle Felder die leere strings enthalten → NULL
  - ✓ ausserdem ist ein Feld PLZ_Wohnort definiert, wird auf zwei Felder verteilt
- tblKunden
  - ✓ enthält nicht bei jedem Kunden eine Adresse, am besten man erstellt eine Tabelle tblKundenAdresse und verlinkt die betroffenen Kunden damit
  - ✓ alle Felder die leere strings enthalten → NULL
  - ✓ ausserdem ist ein Feld PLZ / Wohnort definiert sollte auf zwei Felder geteilt werden
  - ✓ vertrieb, sollte bit sein
- tblProdukt
  - ✓ preis_manuell sollte bit sein
  - ✓ preis_manuell hat kuriose Werte, sie werden wie folgt geändert (0 =1, NULL = 0)
- tblStruktur
  - ✓ coupon_garantiert, coupon_bedingt, text_bed_coupon, floater, floaterBlmbT, text_floater, min_coupon, max_coupon, text_coupon, text_frühz_rückz, cap, max_rückzahlung, text_max_rückz, schutz, min_rückzahlung, partizipation, partizipation_text, discount, max_rendite, bonus_level
  - ✓ folgende Werte sollten als [bit] gespeichert sein: callable, frühzeitige_rückzahlung
- tblVerkauf
  - ✓ ertrag_vertrieb_kunden_id und Werte ändern (-1 = NULL)
  - ✓ sollte [bit] sein: retro_zahlung
  - ✓ bemerkung, leere Felder NULL
- tblZeichnungen_sekundärmarkt
  - ✓ ertrag_vertrieb_kunden_id:, Werte ändern (-1 = NULL)
  - ✓ settlement, leere Felder NULL

15.2 DATENBANKANALYSE RESULTATE

15.2.1 STATISCHE TABELLEN ZU ENUM TABELLE

Eine Tabelle soll entstehen, die alle diese statischen Werte verwaltet. Ausserdem sollen auch alle referenzierenden Tabellen mittels Constraint verbunden sein zur tblEnum, um die Konsistenz zu gewährleisten. Die Einträge welche `'n/A'` oder `''` als Wert enthalten sollen gelöscht werden und die Felder die darauf referenzierten NULL enthalten.

Die EnumTypen werden so benannt oder ähnlich wie vorher die Tabellenbezeichnung war. Ausserdem müssen sie mit einem kleingeschriebenen `'e'` beginnen, z.B. `'eAuslöser'`

Die Spalten, die auf Enum-Werte referenzieren, sollen so umbenannt werden, dass sie entweder gleichnamig mit dem Enumtyp sind oder wenn dieser mehrmals in derselben Tablle verwendet wird muss der Enumtyp zumindest im Namen der Spalte vorkommen.

VORHER tblWährungen

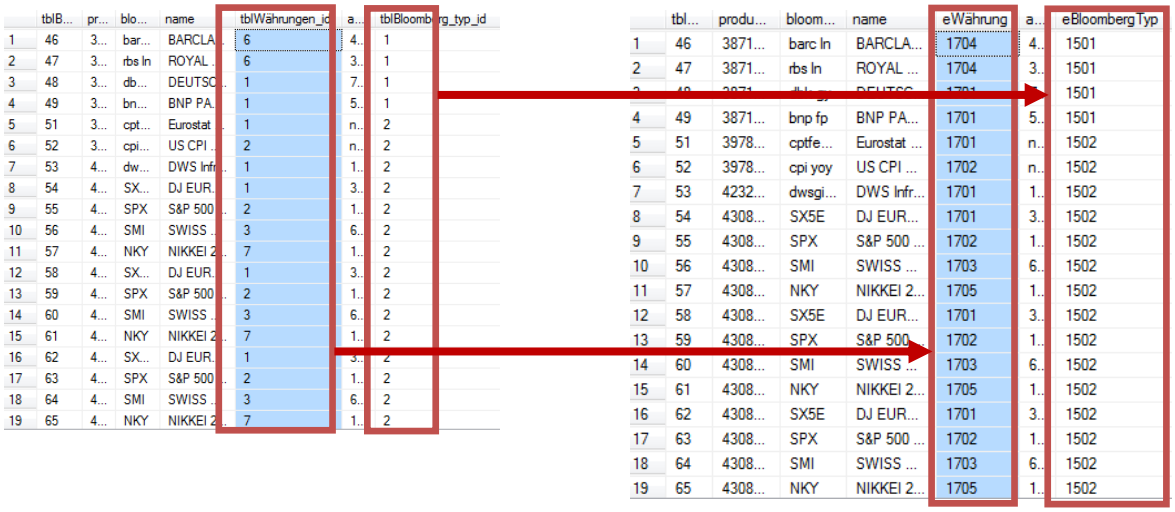
	tblWährungen_id	name
1	0	n/a
2	1	EUR
3	2	USD
4	3	CHF
5	6	GBP
6	7	JPY
7	8	HKD
8	9	SEK
9	10	CAD
10	11	NOK

NACHHER tblEnum

1504	eBloombergTyp	CRNCY	NULL	NULL
1505	eBloombergTyp	CORP	NULL	NULL
1601	ePreisangabe	Nominal	NULL	NULL
1602	ePreisangabe	Stück	NULL	NULL
1701	eWährung	EUR	NULL	NULL
1702	eWährung	USD	NULL	NULL
1703	eWährung	CHF	NULL	NULL
1704	eWährung	GBP	NULL	NULL
1705	eWährung	JPY	NULL	NULL
1706	eWährung	HKD	NULL	NULL
1707	eWährung	SEK	NULL	NULL
1708	eWährung	CAD	NULL	NULL
1709	eWährung	NOK	NULL	NULL
1801	eProdukteKat	Kapitalschutz	NULL	11
1802	eProdukteKat	Renditeoptimie...	NULL	12
1803	eProdukteKat	Partizipation	NULL	13
1804	eProdukteKat	Hebel	NULL	0

VORHER tblBasis_werte

NACHHER tblBasis_werte



	tblB...	pr...	blo...	name	tblWährungen_id	a...	tblBloomberg_typ_id
1	46	3...	bar...	BARCLA...	6	4...	1
2	47	3...	rbs ln	ROYAL ...	6	3...	1
3	48	3...	db...	DEUTSC...	1	7...	1
4	49	3...	bn...	BNP PA...	1	5...	1
5	51	3...	cpt...	Eurostat ...	1	n...	2
6	52	3...	cpi...	US CPI ...	2	n...	2
7	53	4...	dw...	DWS Infr...	1	1...	2
8	54	4...	SX...	DJ EUR...	1	3...	2
9	55	4...	SPX	S&P 500 ...	2	1...	2
10	56	4...	SMI	SWISS ...	3	6...	2
11	57	4...	NKY	NIKKEI 2...	7	1...	2
12	58	4...	SX...	DJ EUR...	1	3...	2
13	59	4...	SPX	S&P 500 ...	2	1...	2
14	60	4...	SMI	SWISS ...	3	6...	2
15	61	4...	NKY	NIKKEI 2...	7	1...	2
16	62	4...	SX...	DJ EUR...	1	3...	2
17	63	4...	SPX	S&P 500 ...	2	1...	2
18	64	4...	SMI	SWISS ...	3	6...	2
19	65	4...	NKY	NIKKEI 2...	7	1...	2

	tbl...	produ...	bloom...	name	eWährung	a...	eBloombergTyp
1	46	3871...	barc ln	BARCLA...	1704	4...	1501
2	47	3871...	rbs ln	ROYAL ...	1704	3...	1501
3	48	3871...	db...	DEUTSC...	1701	7...	1501
4	49	3871...	bnp fp	BNP PA...	1701	5...	1501
5	51	3978...	cptfe...	Eurostat ...	1701	n...	1502
6	52	3978...	cpi yoy	US CPI ...	1702	n...	1502
7	53	4232...	dwsgi...	DWS Infr...	1701	1...	1502
8	54	4308...	SX5E	DJ EUR...	1701	3...	1502
9	55	4308...	SPX	S&P 500 ...	1702	1...	1502
10	56	4308...	SMI	SWISS ...	1703	6...	1502
11	57	4308...	NKY	NIKKEI 2...	1705	1...	1502
12	58	4308...	SX5E	DJ EUR...	1701	3...	1502
13	59	4308...	SPX	S&P 500 ...	1702	1...	1502
14	60	4308...	SMI	SWISS ...	1703	6...	1502
15	61	4308...	NKY	NIKKEI 2...	1705	1...	1502
16	62	4308...	SX5E	DJ EUR...	1701	3...	1502
17	63	4308...	SPX	S&P 500 ...	1702	1...	1502
18	64	4308...	SMI	SWISS ...	1703	6...	1502
19	65	4308...	NKY	NIKKEI 2...	1705	1...	1502

15.2.2 NULL NOT NULL AUF ATTRIBUTSEBENE

Wir haben beschlossen, dass dies auch noch in einem späteren Stadium erledigt werden kann. Es wird lediglich dort wo Bit-Datentypen gesetzt werden auch gleichzeitig das NOT NULL Attribut gesetzt.

15.2.3 GEWÄHRLEISTUNG VON 1 ZU 1 BEZIEHUNGEN

Um 1 zu 1 Beziehungen im Entity Framework auch so auflösen zu können, wird erwaret, dass der Primary Key auch gleichzeitig der Fremdschlüssel für die in Beziehung stehende Tabelle ist. Dazu muss man in unserem Fall eine Haupttabelle definieren und alle anderen mit dieser Tabelle verknüpfen.

16 TECHNOLOGIE ANALYSE

16.1 MVVM FRAMEWORK

16.1.1 VORGEHENSWEISE

Für diesen Punkt kamen entweder PRISM, das meist für Businessapplikationen eigesetzt wird oder das einfach gehaltene MVVMLight in Frage. die Vor- und Nachteile der einzelnen Frameworks einander gegenüberzustellen. Um dies objektiv zu betrachten werden wir die Anforderungen an die Software und deren Datenverarbeitungsbedarf möglichst berücksichtigen. Es werden folgende Faktoren analysiert um das am besten geeignete Framework zu bestimmen.

Zu berücksichtigen	Beschreib
Komplexitätsgrad	Jedes Framework weist eine bestimmte Komplexität auf. Ist diese höher als der Unterstützungsgrad sollte das Framework nicht verwendet werden.
Vorkenntnisse	Wie gut kennen wir das Framework? Eignet sich das Framework eine Desktop-Applikation zu erstellen?
Support / Ressourcen	Wo und in welchem Masse stehen Support, Ressourcen und Beispiele zur Verfügung?
Vor- / Nachteile	Was wären die grössten Vor- und Nachteile wenn dieses Framework eingesetzt werden würde?

Tabelle 12 Analysekriterien der Technologieanalyse

16.1.2 ABSTRAHIERUNG DER VISION

Aus der bestehenden Applikation und den Vorstellungen des Auftraggebers gehen folgende Anforderungen hervor.

16.1.3 MVVM-ARCHITEKTUR

Die bestehende WinForms-Applikation mit Code-Behind, soll durch eine moderne MVVM-Architektur ersetzt werden. Damit wird die Business-Logik von der View getrennt, was die Wartbarkeit des Codes dramatisch verbessert.

MODULARER AUFBAU DES BUSINESSLAYERS

Der Business-Layer soll viele kleine Module enthalten, die voneinander unabhängige Aufgaben erledigen. Die Module sollen eine klare Schichtentrennung zwischen Businesslogik und GUI herstellen.

VERKNÜPFUNG VON BUSINESSLOGIK-MODULEN

Module sollen verknüpfbar sein. Das heisst, ein Event-Anzeige-Modul soll mit einem Produkt-Container-Modul verknüpft werden können. Das Resultat ist ein Modul das für jedes Produkt im Container den Event anzeigt.

WIEDERVERWENDBARE GUI-KOMPONENTEN

Ein Produkt kann mehrere Basiswerte haben. Diese Basiswerte sollen sowohl auf der Produktübersicht als auch auf der Basiswerte-Übersicht angezeigt werden können.

16.1.4 VERGLEICH DER FRAMEWORKS

STUDIUM DER FRAMEWORKS

Detaillierte Beschreibungen der Frameworks:

- PRISM: http://compositewpf.codeplex.com/
- MVVM Light: http://www.galasoft.ch/mvvm/getstarted/

UNTERSTÜTZUNG DER ANFORDERUNGEN

Anforderung	PRISM	MVVMLight
MVVM-Architektur	geringe Unterstützung	sehr grosse Unterstützung
Modularer Aufbau	sehr grosse Unterstützung	mittelmässige Unterstützung
Verknüpfung Businesslogik	grosse Unterstützung	mittelmässige Unterstützung
Wiederverwendung GUI	implizit	geringe Unterstützung
Komplexitätsgrad	Hoch	gering

Tabelle 13 Anforderungsunterstützung der Frameworks

In obiger Tabelle fällt auf, dass PRISM die geforderte Funktionalität optimal unterstützen kann. Die Stärken von PRISM liegen in den beiden Kernanforderungen „Modularer Aufbau“ und „Verknüpfung der Businesslogik“. Desweiteren kann PRISM mit einer impliziten „Wiederverwendung von GUI-Komponenten“ punkten.

ALLGEMEIN

Kennzahl	PRISM	MVVMLight
Komplexität	sehr hoch	mittel

Vorkenntnisse	keine	vorhanden
Support / Ressourcen	sehr ausführlich	ausführlich
Vorteile	Löst viele Designprobleme die sich aus den Anforderungen ergeben.	Erlaubt einen schnellen und pragmatischen Aufbau der Architektur.
Nachteile	Sehr grosser Lernaufwand.	Keine Unterstützung bei der Umsetzung der Businesslogik.

Tabelle 14 Allgemeine Kennzahlen

16.1.5 FAZIT

Abschliessend können folgende Aussagen gemacht werden:

1. MVVM Light ist ein Architektur-Helfer der dem Programmierer den Umgang mit MVVM-Architekturen erleichtern kann.
2. PRISM ein Business-Applikationen-Helfer der dem Programmierer dabei hilft, komplexe Business-Applikationen zu erstellen.
3. Das PRISM Framework erfüllt die Anforderungen an die neue Applikation am vollwertigsten.
4. PRISM nimmt dem Programmierer sehr viel Designarbeit ab. Die Komplexität des Frameworks ist jedoch vergleichsweise hoch.

Die Frage die wir uns stellen müssen ist:
Sind wir bereit den Mehraufwand zu leisten, uns mit dem PRISM Framework auseinanderzusetzen?

Trotz optimaler Unterstützung von PRISM haben wir uns dennoch dagegen entschieden. Mit MVVM Light lässt sich vorerst sehr gut arbeiten, falls wir an allen Ecken und Kanten damit anstossen können wir immernoch umsatteln.

16.2 TASKING

In der bestehenden Applikation werden für verschiedene Objekte (z.B. Produkt, Event. Kauf) teilweise kombiniert Services aufgerufen. Dabei ist es unser Ziel, für ein Objekt möglichst die gegebenen Funktionalitäten anzubieten, ohne dabei leere Interfaces oder nichtfunktionale Superklassen zu verwenden.

16.2.1 UMSETZUNGSANALYSE

UNITY FRAMEWORK

Implementieren aller ausführenden Methoden in den ServiceWrappern. Diese Methoden werden dann beim Aufruf des Konstruktors der Service-Klassen im UnityContainer registriert. Bei Verwendung kann anhand des Typs des Objekts und der Instruktion in Form eines Strings die betreffende Methode aufgerufen werden.

REFLECTION

Implementieren aller ausführenden Methoden in den unteren Schichten. Wenn ein Task erstellt wird und dieser Weitergereicht werden soll, kann die ausführende Methode anhand des Methodennamen und der Zielklasse bestimmt werden, ausserdem kann sogar anhand des Typs die dafür vorgesehene überladene Methode gewählt werden, wenn mehrere Typen akzeptiert werden.

WEITERES VORGEHENS

Es werden Prototypen erstellt, die die oben beschriebenen Technologien umsetzen anschliessend soll entschieden werden, welche der beiden für unser Anwendungsfall besser geeignet ist.

16.2.2 VOR- UND NACHTEILE

Unity	Reflection
Vorteile	
• Inversion of Control: Es ist so möglich unterschiedlichste Services und Methoden zur Verfügung zu stellen, diese können mittels eines Strings wieder aufgelöst werden • Mit Dependency Injection Funktionalität, die vom Unity gegeben ist, können wir Objekte bei Bedarf erzeugen und müssen uns nicht um den Aufruf für das spezifische Objekt kümmern. Die Kopplung ist somit direkt in der ausführenden Klasse.	• Bietet nahezu unbeschränkte Möglichkeiten, jegliche Funktionalitäten können ohne darüber Bescheid zu wissen ermittelt werden.
Nachteile	
• Es entsteht eine Kopplung in Form von strings die registriert werden und mit denen man wieder die Referenz der Methoden erhält, diese Kopplung lässt sich jedoch einfach in einer einzigen Klasse im Common Layer realisieren.	• Man muss aber wissen, welche Funktionen jeweils zum Einsatz kommen sollen. Man braucht ein bestimmtes Wissen über eine Klasse und es entsteht auch eine nicht kontrollierbare Kopplung • Man kann nicht flexibel den Source Code anpassen, da eventuell darauf mit Reflection zugegriffen wird. Der Kompiler kann aber keine Fehler ermitteln • Man hat keine Garantie, dass eine Methode für ein Objekt auch existiert.

	Auch keine Übersicht, welche Methoden überhaupt benötigt werden

Tabelle 15 Vor- und Nachteile Unity / DI

16.2.3 ENTSCHEID

Für unsere Applikation eignet sich der UnityContainer, da er die Funktionalität die wir mit dem Reflection-Ansatz verfolgen würden bereits anbietet. Er stellt eine sauber programmierte Schnittestelle, die zur Kompilierzeit bestimmen kann, ob eine korrekte Ausführung möglich ist. Auch ist das Exception Handling bereits implementiert. Es ist professionell diese Möglichkeit einzusetzen.

17 ARCHITEKTUR ÜBERSICHT

17.1 SERVICE ORIENTEDARCHITEKTUR

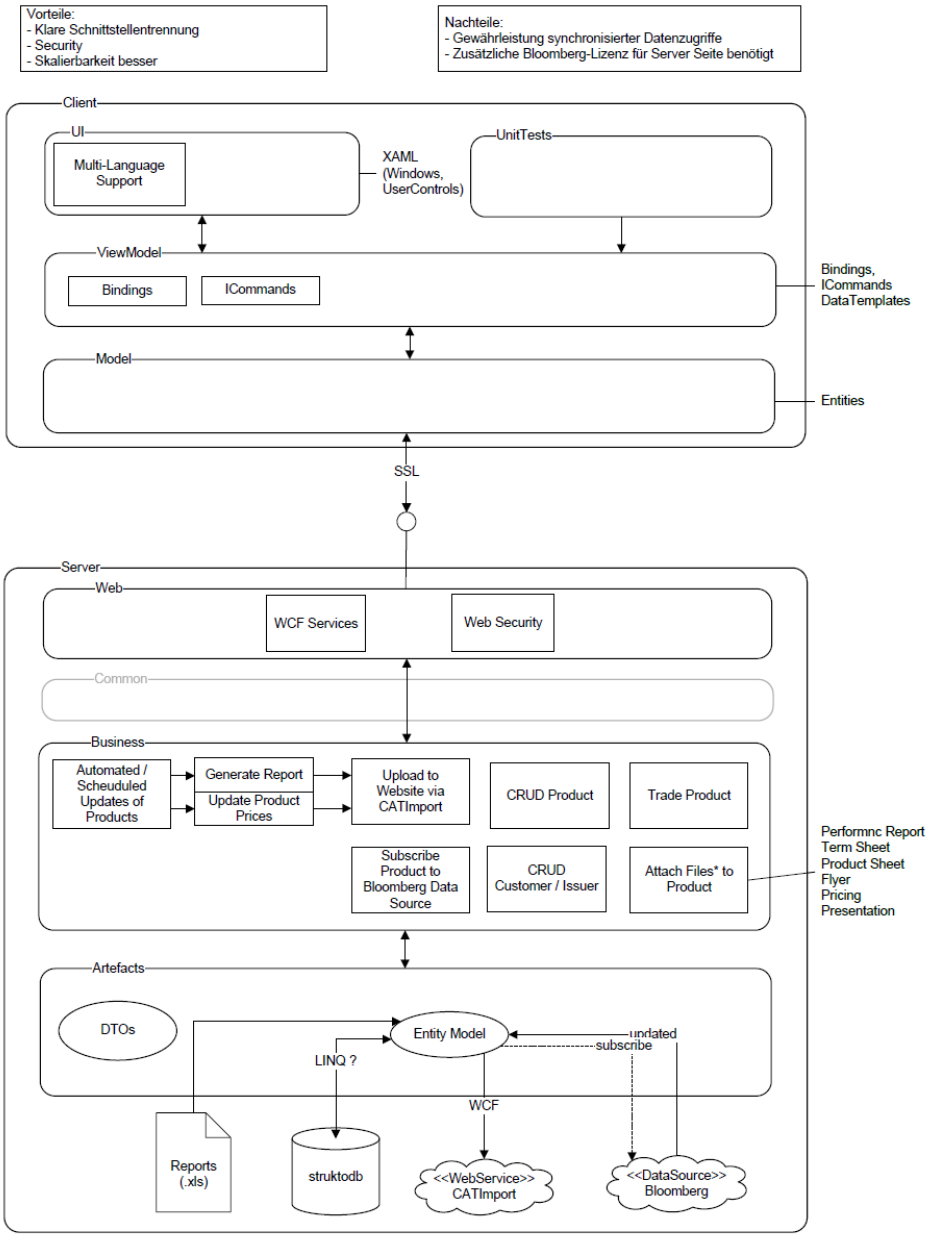


Abbildung 22 Architekturübersicht SOA

17.2 EINFACHE WPF MVVM ARCHITEKTUR (WIRD UMGESETZT)

Vorteile:
- Weniger Technologien (kein WCF) als SOA
- Keine zusätzliche Bloomberg Lizenz benötigt

Nachteile:
- Erschwerte Synchronisation zw. Applikationsinstanzen
- Erweiterbarkeit eingeschränkt => Erweiterung auf bspw. Silverlight Client, HTML5

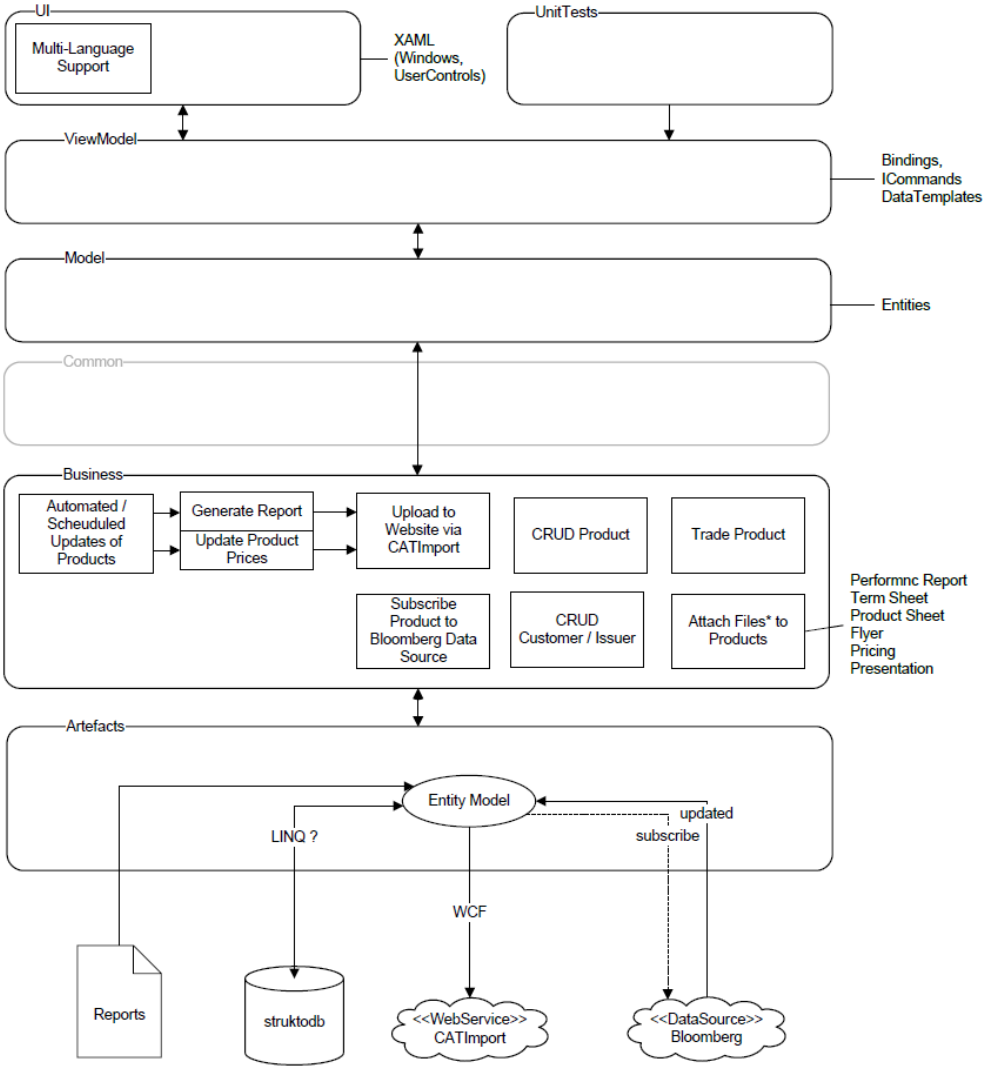


Abbildung 23 Architekturübersicht 2Tier

17.3 BENUTZTE LIBRARIES ZU FUNKTIONALITÄT

Die Funktionalität des StruktoManager 2.0 ist in verschiedene Schichten aufgeteilt. Ausserdem wurden diverse externe Libraries und Services verwendet:

Externe Libraries

- Odyssey
- Unity 2.0
- MVVM Light WPF4 + extras
- Bloomberg API
- Unity 2.0
- Log4Net

Externe Services

- CatWebservice.Import

Wie diese im Kontext der Applikation zum Einsatz kommen wird in nachfolgender Grafik illustriert.

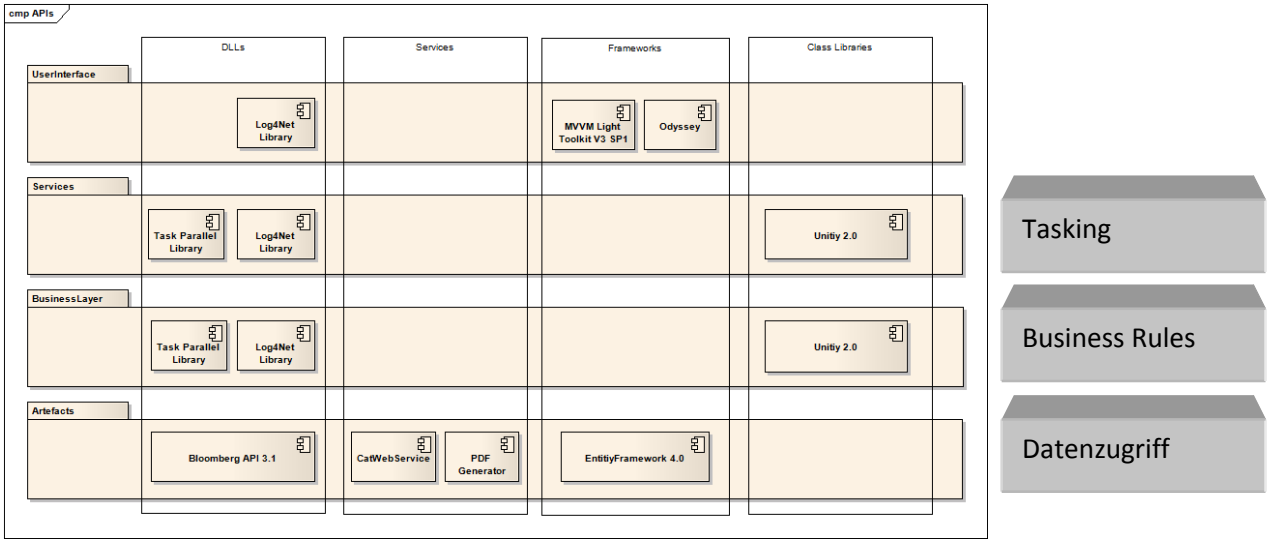


Abbildung 24 Benutzte Libraries

18 DOMAIN MODEL

Folgendes DomainModel stellt die Business Objekte dar und ist der geplante Stand für die Finale Entwicklung. Dieses wird sich sehr wahrscheinlich bei der Weiterentwicklung noch ändern.

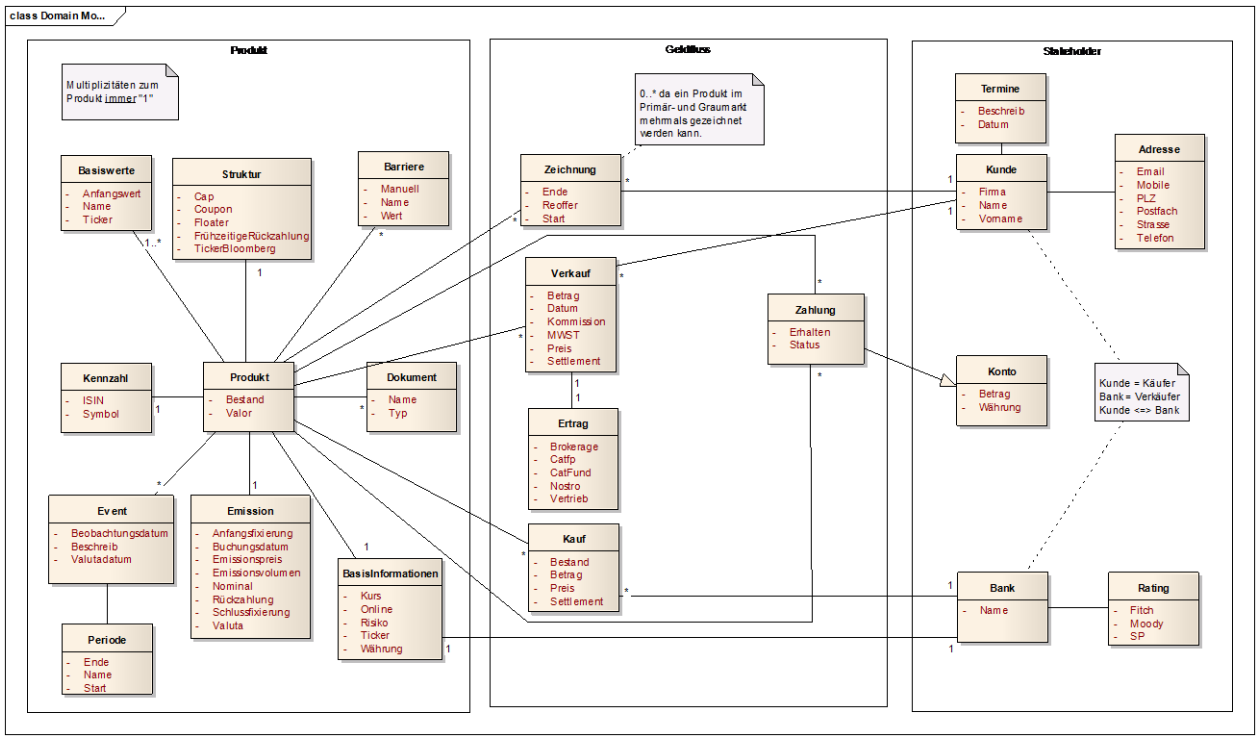
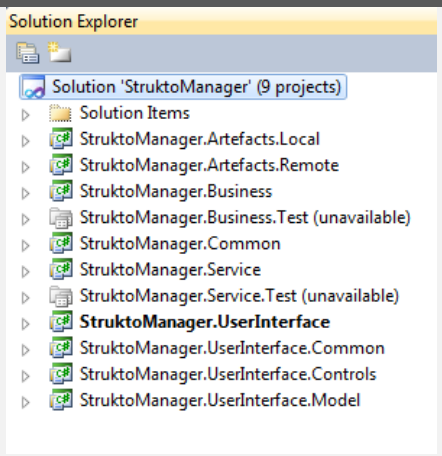
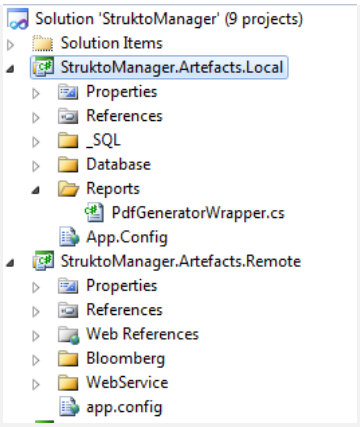


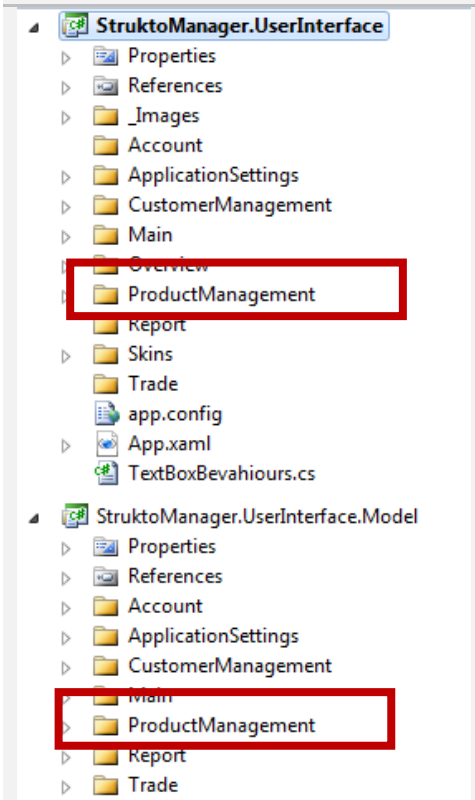
Abbildung 25 Domainmodel

19 PROJEKTSTRUKTUR / DESIGN PAKETE

Bild	Beschreibung
	Die Projektstruktur wurde gemäss Definition der Architektur abgebildet. Für jedes Package bzw. Layer existiert ein eignes Projekt. Diese Struktur repräsentiert auch gleich die Design Pakete des Projektes.



Innerhalb der Projekte wird pro Funktionsgruppe ein Ordner erstellt. Wird diese Funktionalität über mehrere Layer abgearbeitet, so werden identisch benannte Ordner über verschiedene Layer erstellt



ViewModel und Userinterface werden jeweils in einer symmetrischen Ordnerstruktur verwaltet. Dabei gilt auch, dass pro Funktionsgebiet ein Überordner erstellt wird und werden Controls von einem Control inkludiert so wird ein Unterordner erstellt.

Themengebiete werden definiert durch Navigation unten Links im UI

Oder über die Losgelösten Funktionen zuoberst im User Interface

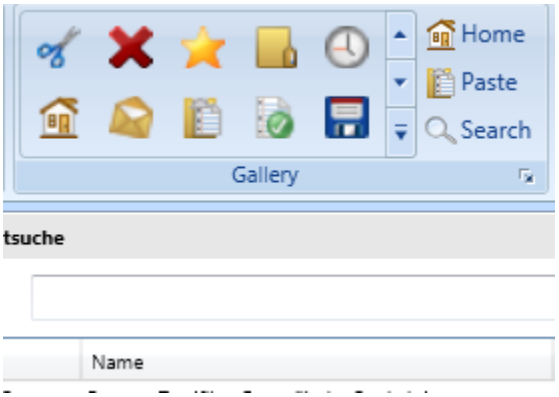


Abbildung 26 Projektstruktur

19.1 ZUSÄTZLICHE FUNKTIONALITÄT

19.1.1 LOGGING

Für das Logging wird Log4Net eingesetzt. Dabei werden alle Exceptions festgehalten und alle wichtigen Aktionen im Programm, wie das Handeln mit den Wertpapieren und die StruktoTasks, welche an die Services delegiert werden.

19.1.2 ASYNC LOADING

Ziel der Software ist es, Daten asynchron zu laden. Während der Zeit in der in einem Control Daten geladen werden, soll ein Lade-Layer über dieses gelegt werden. Das Benutzerinterface reagiert also immernoch auf Useraktionen.

19.1.3 SWITCHING ZWISCHEN FENSTERN

Wenn der Anwender während der Erfassung eines Produktes unterbrochen wird, soll es dem Anwender möglich sein, nach einer anderen Aktion wieder beim ursprünglichen Fenster weiterzuarbeiten. Man nehme an ein Kunde meldet sich per Telefon und möchte sich über sein Konto erkundigen, dann kann man direkt zum Bereich “Konten” wechseln, die Aktion dort vornehmen und nach dem Telefonat wieder auf Produkte klicken, um bei demselben Stand weiterzuarbeiten wo er unterbrochen wurde. Solche User Interfaces sollen als Singleton implementiert werden.

19.1.4 SHARED DATA

Filter die im “BrowseControl“ gesetzt werden sollen auch Anwendung im „SearchControl“ finden. Um dies zu ermöglichen soll die Applikation so ausgelegt werden, dass mit Objekten von derselben Instanz auf beiden Controls gearbeitet wird. Diese Datenstruktur ist Generic und somit für beliebige Datentypen, die in ähnlicher Weise angezeigt werden, einsetzbar.

20 UMSETZUNG DES TASKING

Das Tasking bildet das zentrale Element der Applikation. Die folgenden Klassen sind für das Verständnis essentiell.

Klasse	Beschreib

StruktoTaskManager	Erstellt und verwaltet sämtliche Tasks die das UI starten will.
StruktoTask	Kapselt die Informationen über die auszuführenden Arbeiten und gibt Auskunft über den Stand der Arbeit.
TaskDescription	Kapselt die Arbeits-Informationen eines StruktoTasks und kann dem Task mitteilen auf welchem Service er sein Arbeitspaket platzieren kann.
StruktoService	Kann Tasks verarbeiten indem er die Informationen, die der StruktoTask bereitstellt, analysiert und der entsprechenden Wrappermethode weiterleitet.
ServiceWrapper	Diese Klassen enthalten die eigentliche Funktionalität der Applikation.
ServiceOperation	siehe folgender Abschnitt

Tabelle 16 Wichtigste Klassen des Taskings

Eine weitere wichtige Klasse ist die ServiceOperation. Sie beschreibt eine bestimmte Methode auf einem ServiceWrapper. Der Task nutzt die ServiceOperation um dem Service mitzuteilen für welche Methode auf dem ServiceWrapper er etwas zu tun hat.

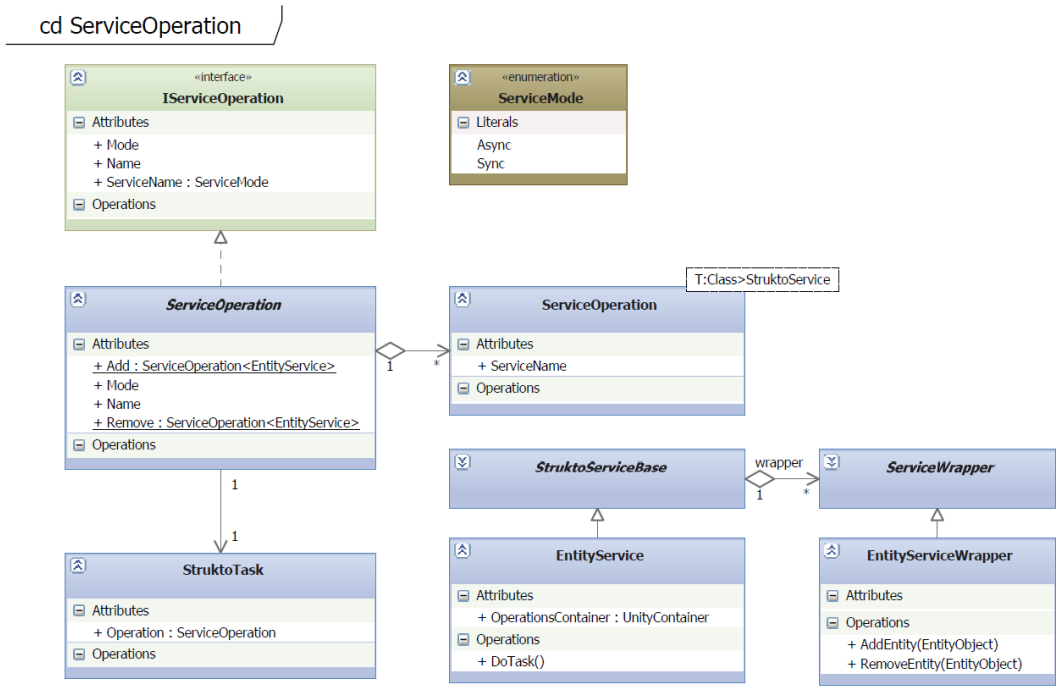


Abbildung 27 Klassenstruktur ServiceOperation

Die Properties der ServiceOperation haben folgende Bedeutung:

Property	Beschreib
Mode	Beschreibt ob die Operation auf dem Service synchron oder asynchron ablaufen soll.
Name	Identifiziert die ServiceOperation und muss, pro Service, eindeutig sein.
ServiceName	Gibt den Namen des Services zurück für den die Service Operation definiert wurde.

Tabelle 17 Properties der ServiceOperation

Folgende Abbildung verdeutlicht das Zusammenspiel der oben beschrieben Klassen. In den folgenden Kapiteln wird genauer auf diesen Ablauf eingegangen.

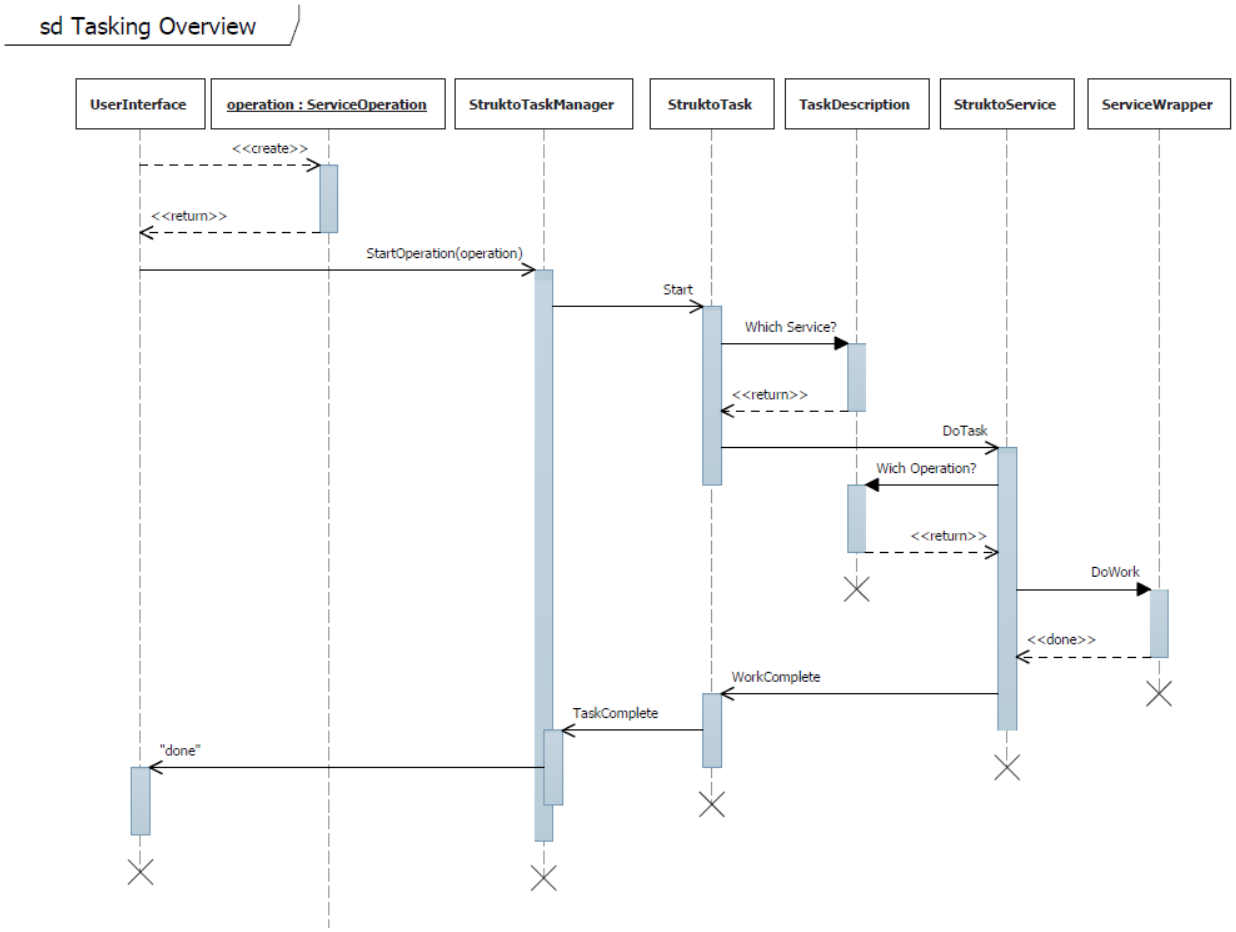


Abbildung 28 Übersicht Tasking

20.1.1 STARTEN EINES TASKS

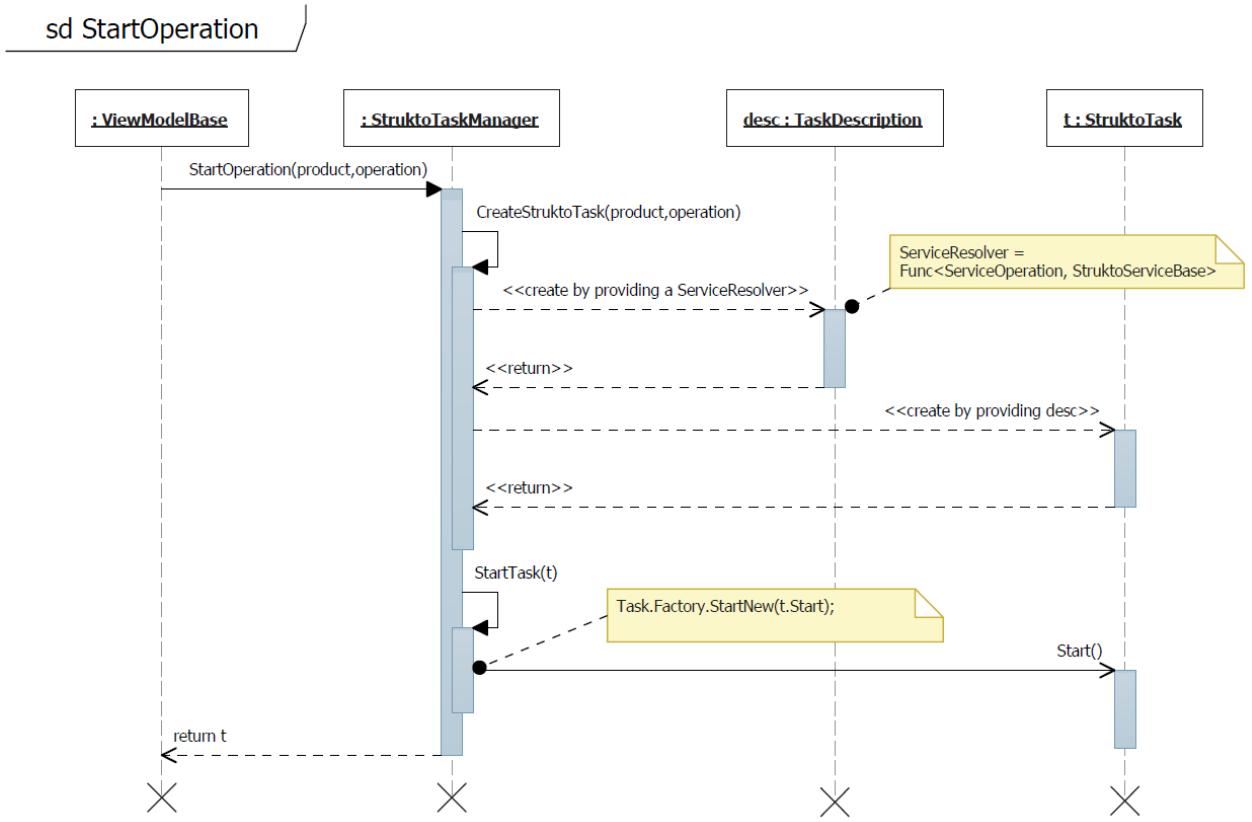


Abbildung 29 Ablauf des Starts einer Operation

Obige Abbildung zeigt was alles beim Start eines Tasks geschieht. Nachdem das ViewModel die auszuführende Operation an den StruktoTaskManager übergeben hat, wird eine TaskDescription erstellt. Die TaskDescription erhält vom StruktoTaskManager eine Funktion die es ihr ermöglicht aus einer ServiceOperation den richtigen Service herauszufinden. Anschliessend wird der Task instanziert und mithilfe des Task-Frameworks von .Net asynchron gestartet. Das UI erhält den gestarteten Task und kann sich so jederzeit über den Stand der Arbeit informieren.

20.1.2 AUSFÜHREN EINES TASKS

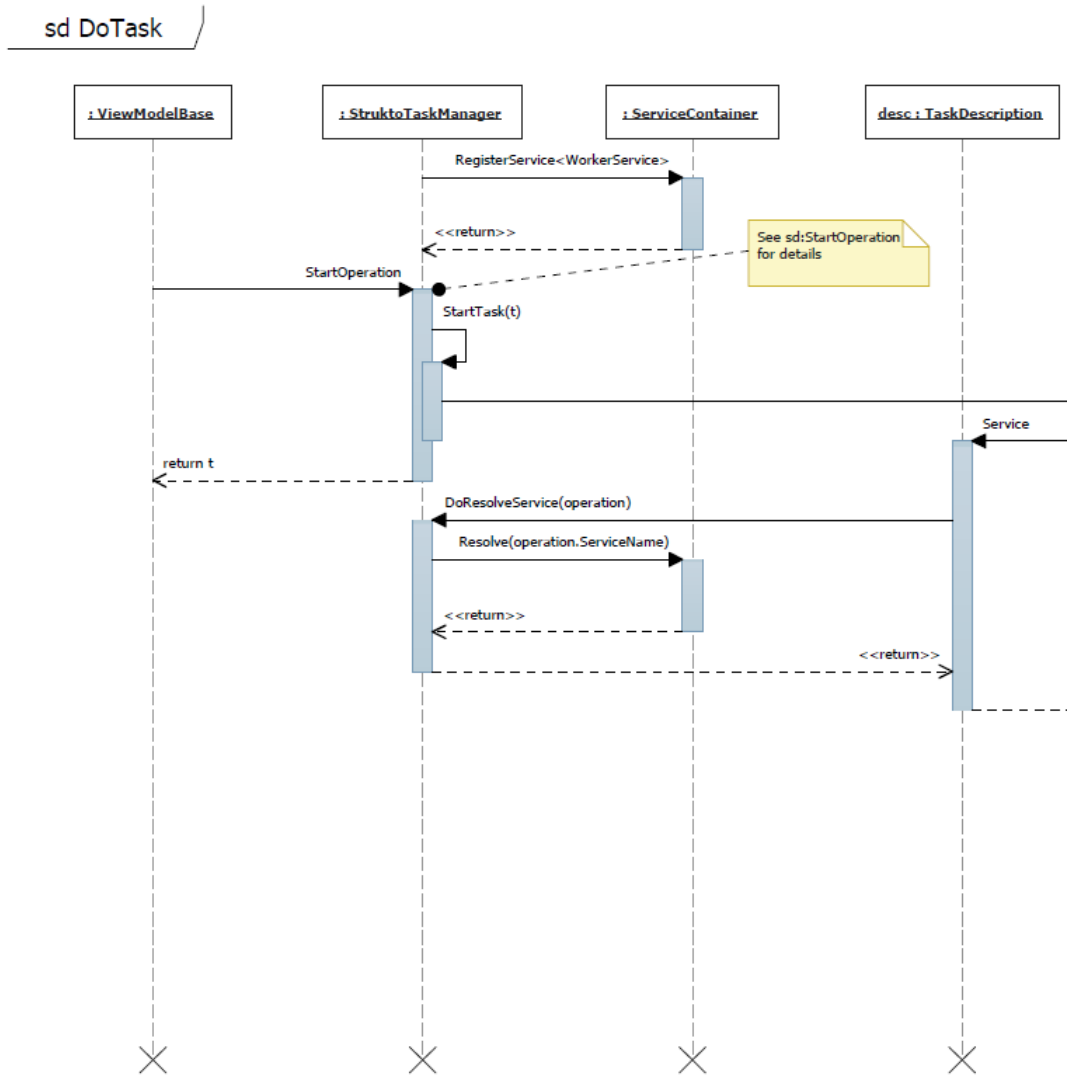


Abbildung 30 Registrierung eines Service und ResolveService-Funktionalität

Im Diagramm wird verdeutlicht wie der StruktoTaskManager einen Service registrieren kann. (siehe nächste Abbildung für die Funktionalität der WorkerService-Klasse). Weiter ist zu sehen wie die TaskDescription des StruktoTasks den angeforderten Service über die ServiceOperation auflösen kann. Die Funktion liegt auf dem StruktoTaskManager, aber die TaskDescription kapselt diese auf ihrem Service-Property.

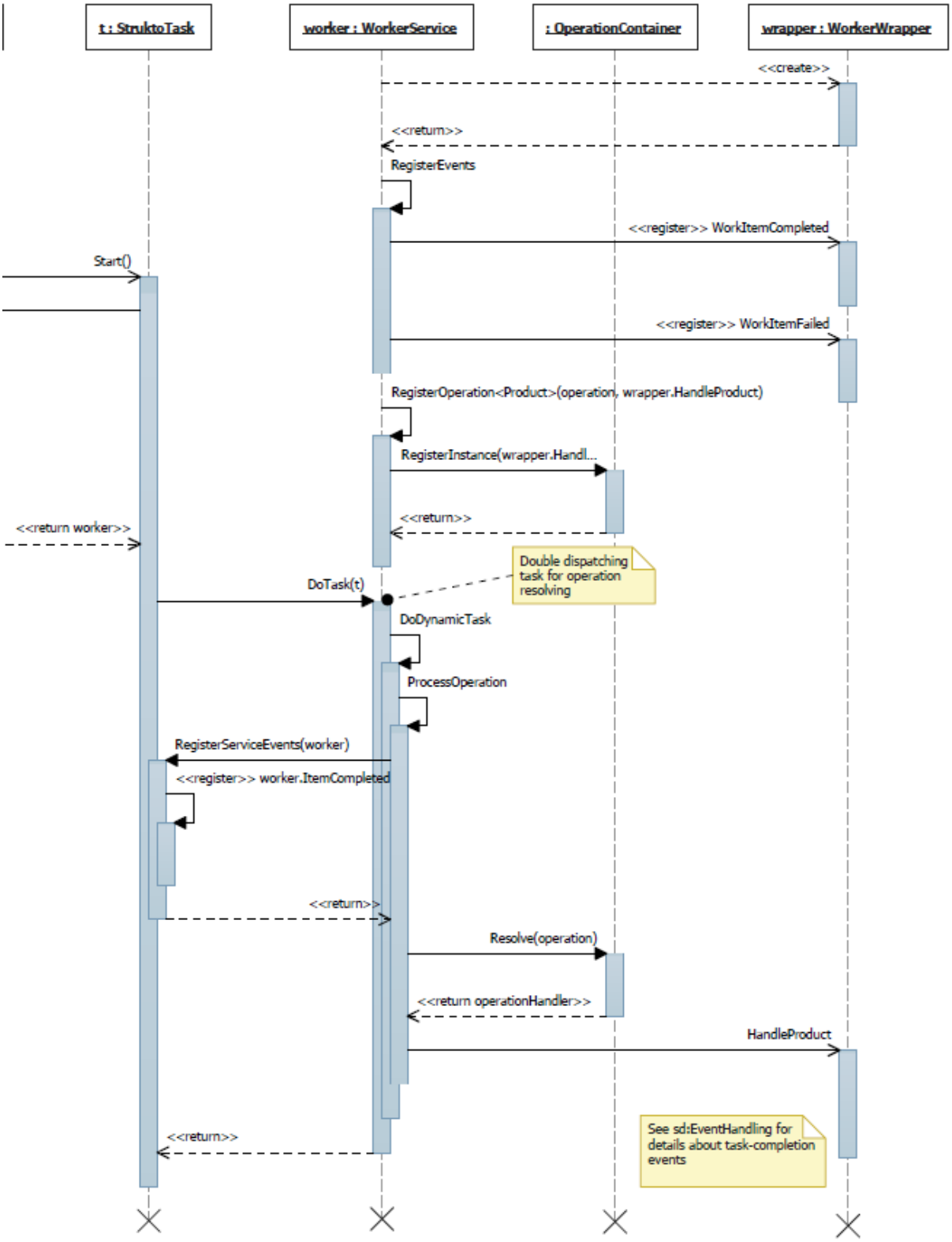


Abbildung 31 Registrierung einer Operation und DoTask-Funktionalität

Der Worker Service registriert die Funktionalität des Wrappers auf dem OperationContainer und in der ProcessOperation-Methode kann diese wieder aufgelöst werden. Weiter ist die DoTask-Methode zu beachten. Hier wird eine Variante des Double-Dispatch Pattern [ddp] verwendet um den übergebene Task (der als Interface übergeben wird) wieder in ein typisiertes Objekt zu verwandeln. Dies ist nötig, da nur auf einem typisierten Objekt der korrekte OperationHandler-Name aufgelöst werden kann.

20.1.3 EVENTHANDLING

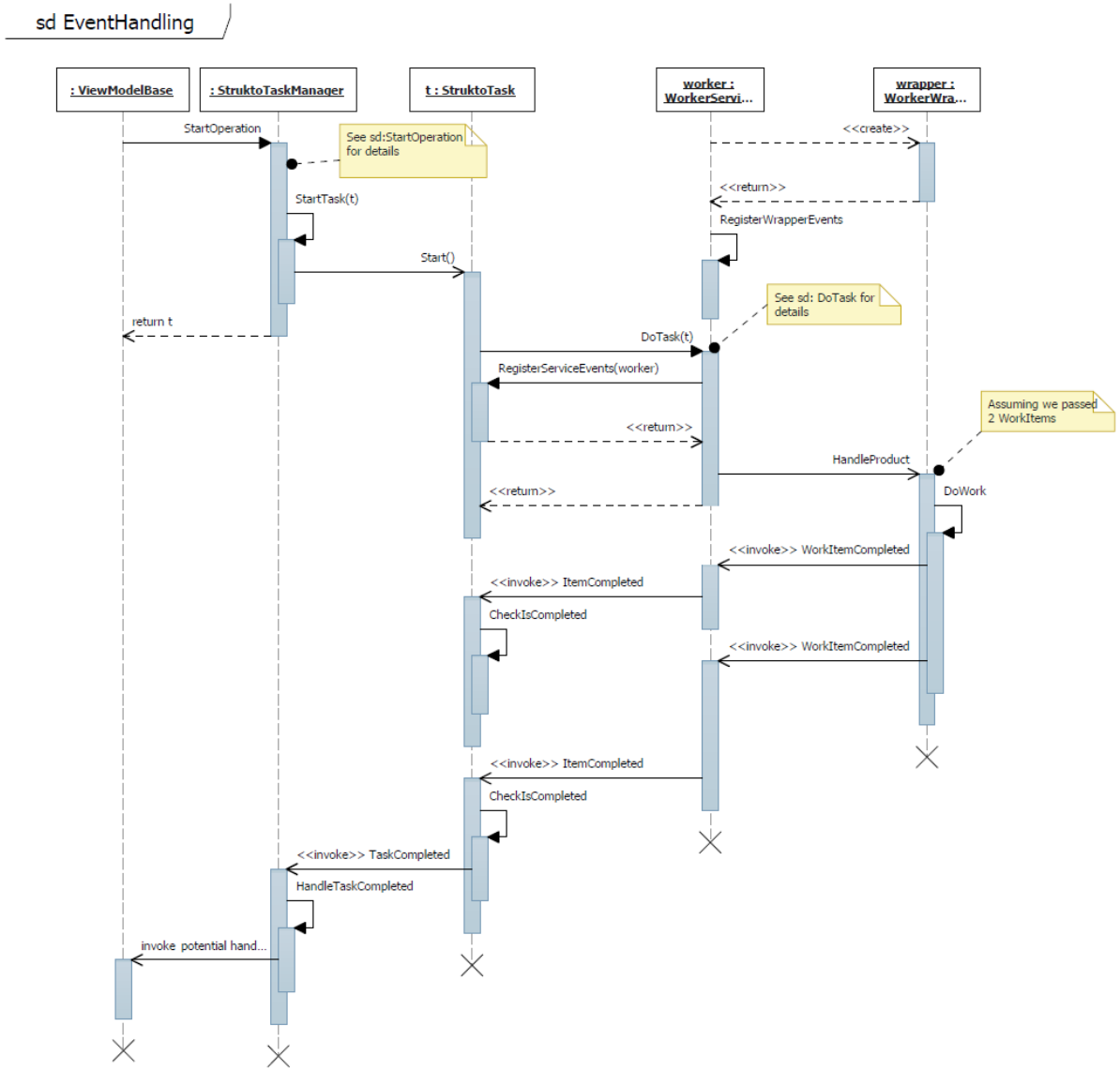


Abbildung 32 Ablauf des Event Handling

Hier werden die wichtigsten Messages eines erfolgreichen Tasks illustriert. Zu beachten ist, dass hier zwei WorkItems dem Wrapper übergeben wurden. Nachdem der Task vom Service über die erfolgreiche Abarbeitung der beiden Workitems informiert wurde, meldet er dies dem StruktoTaskManager. Dieser löst aus, dass alle potentiellen Interessenten in der HandleTaskCompleted-Methode informiert werden.

21 USER INTERFACE

21.1 DESIGN

Das Userinterface baut auf den frei zur Verfügung stehenden Odissey Controls auf, eine Entwicklung von Microsoft. Anhand unseres Prototypen und der daraus gewonnenen Erkenntnis folgt eine Beschreibung, wie wir das Userinterface umsetzen.

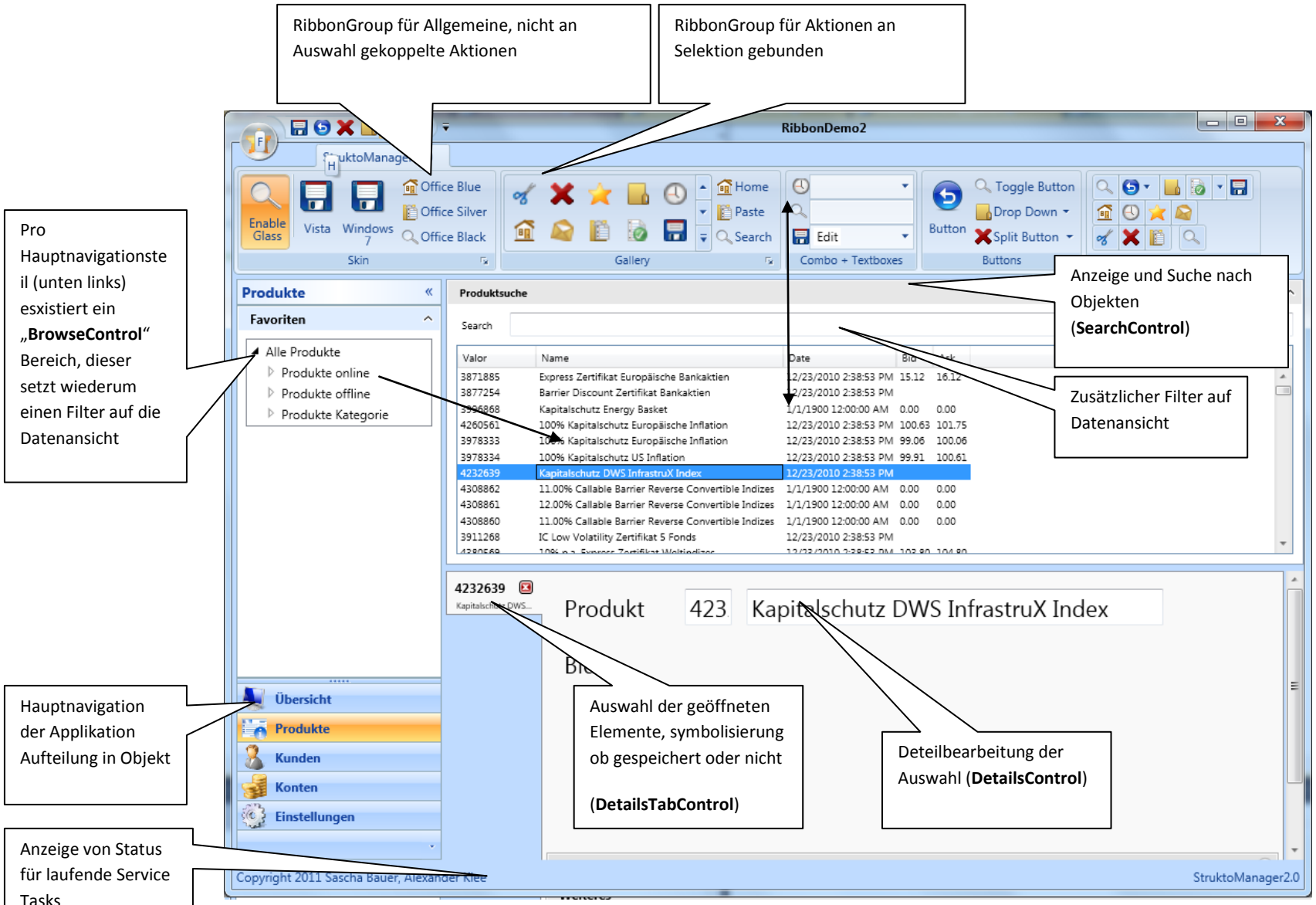


Abbildung 33 Übersicht aller Controls

21.2 EDITIERMODUS

Innerhalb eines TabControls müssen Änderungen unterschiedlicher Entitäten festgestellt werden können und dem Benutzer kommuniziert werden. Darauf soll mit einer Benutzerrückfrage reagiert werden, ob beim Schliessen die Änderungen gespeichert werden sollen oder nicht. (Siehe Messaging).

Ein roter Punkt erscheint, sobald ein geänderter Status herrscht. Dazu wurde jedes IDependantTracking implementierende Objekt stellt ein IsChanged Property zur Verfügung. Dieses lässt sich einfach an die Benutzeroberfläche binden.

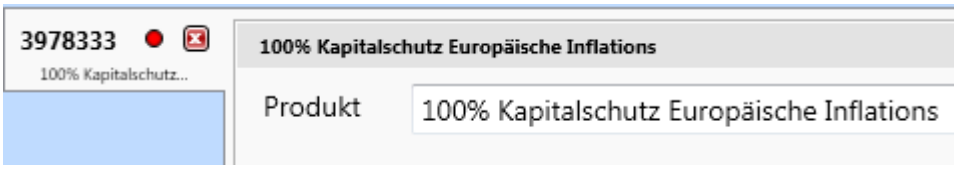


Abbildung 34 IsChanged-Punkt

21.3 EINSATZ VON MVVM LIGHT

Für die Umsetzung von MVVM wird die Fremdentwicklung MVVM Light verwendet. Dieses bietet eine Möglichkeit an, die ViewModel zu verwalten in sogenannten ViewModelLocator. Ausserdem wird die Klasse RelayCommand<T> verwendet. Für Delegationen vom ViewModel an das UI wird weiter Messaging eingesetzt.

21.3.1 MESSAGING

Messaging dient in unserem Falle in erster Linie dazu Fenster zu öffnen und zu schliessen. Desweiteren werden damit verbundene Entscheidungen und die dazu benötigten Parameter so dem ViewModel oder der View mitgeteilt.

Das wohl komplizierteste Konstrukt, das damit gelöst wurde, ist wenn der User ein Tab schliessen will, soll eine Meldung erscheinen, ob er die Änderungen speichern möchte.

Abbildung 35 Messaging

21.4 VIEM MODEL ZUGEHÖRIGKEIT

Untenstehende Grafik illustriert den Einsatz der verschiedenen Viewmodel. In diesem Fall ist das Set der ViewModel für ein Produkt dargestellt.

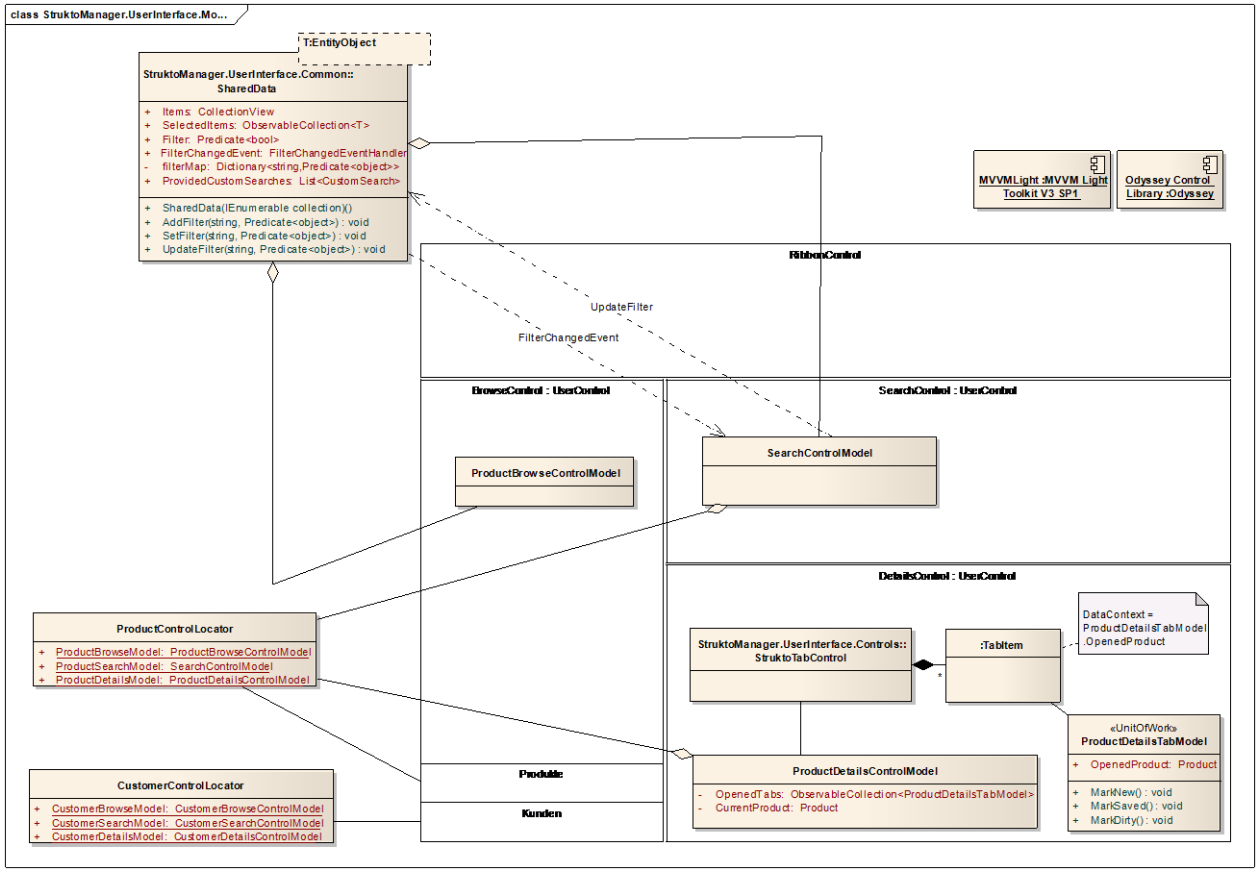


Abbildung 36 Lokalisierung der Modelklassen

Das Schwergewicht an Programmierlogik findet sich aber in den jeweiligen Basisklassen und den generischen Klassen. Alle typenspezifischen Models werden von diesen abgeleitet. Durch die generische Programmierung kann immer wieder dieselbe Logik verwendet werden, sofern kein abweichendes Userinterface erstellt wird. Dazu gilt es noch zu erwähnen, dass ContextBasedModel parallel zum Change Tracking die Abhängigen eines User Controls innerhalb des TabControls an den behandelten Datenkontext bindet.

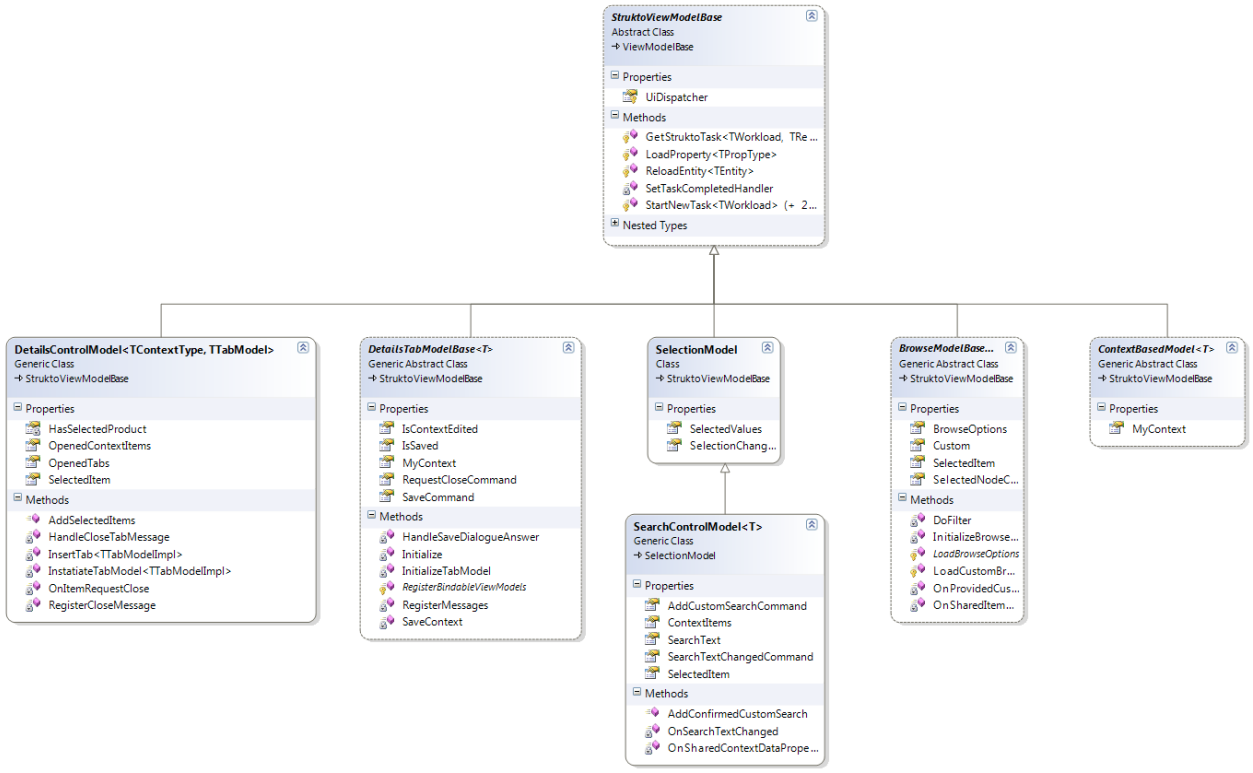


Abbildung 37 BaseKlassen der Models

22 DATENANBINDUNG

22.1 ENTITY FRAMEWORK

In der Hälfte des Projektes entschieden wir uns vom Entity Framework 4.0 auf 4.1 zu wechseln für die Anbindung der Daten. Es bot den Vorteil, dass die Include() Methode nicht mit einem String die abhängigen Entitäten geladen werden, sondern per Delegate und somit vom Kompiler interpretierbar. Dies verringert die Fehleranfälligkeit, lässt Änderungen per Refactoring zu und ausserdem kann man ohne die Strings nachzuschlagen programmieren.

Entity Model → siehe in Visual Studio StruktoManager.Artefacts.Local.Database StruktoData.edmx

22.2 CHANGE TRACKING

Das ChangeTracking der Entitäten wurde um folgendes Konstrukt erweitert. In unserem Fall wurde dies bereits für die eingesetzten Datentypen eingebaut, es handelt sich um eine einfache aber effektive Methode abhängige Entitäten bewusst von einem Hauptobjekt (von dem die anderen mitgeladen wurden) aus zu verfolgen, ob auf diesen Modifikationen passiert sind.

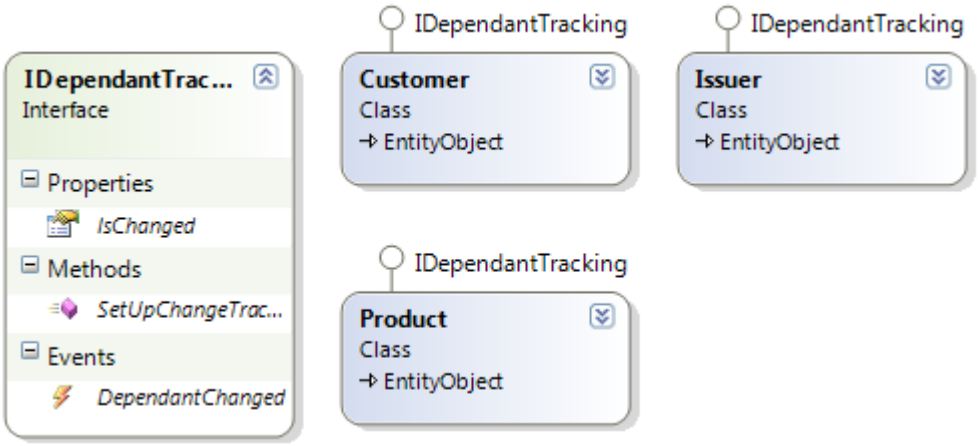


Abbildung 38 IDependantTracking

Projekt: StruktoManager 2.0

Glossar

23 GLOSSAR

23.1 FINANZTERMINOLOGIE

Strukturiertes Finanzprodukt
Ein Anlageprodukt, das durch die Kombination mehrerer Basisfinanzprodukte entsteht, von denen eines ein Derivat sein kann.

Zeichnung
Siehe Börsengang

Coupon
Der Abschnitt eines Wertpapiers, der zur Einlösung eines Gewinnanteils (Dividendenschein) oder Zinses (Zinsschein) berechtigt. Ein Coupon von 6% besagt, dass zum jeweiligen Zinstermin 6% des Nominalwerts als Zins gezahlt wird.

Coupon-Periode
Bezeichnet die Zeitspanne in der ein Bestimmter Gewinnanteil auf dem Produkt gewährt wird.

Valuta
Hiermit wird der Tag der Belastung des Käufer-Kontos, bzw. der Gutschrift auf dem Verkäufer-Konto bezeichnet. Ein Produkt kann am Montag gehandelt werden und am Dienstag erfolgt die Belastung des Kontos. Früher wurden Vermögenswerte gegen Lieferung der Papiere übermittelt. Heute werden die Papiere lediglich vom Käufer auf die Verkäufer umgeschrieben, was dem eigentlichen Handel entspricht.

Primärmarkt
Als Primärmarkt wird die Geschäftsbeziehung zwischen Ersteller eines Produkts und dem Erstkäufer des Produktes bezeichnet. Nachdem ein Produkt vom Ersteller gezeichnet wurde, kann dieses Produkt vom Erstkäufer im Sekundärmarkt gehandelt werden.

Sekundärmarkt
Der Sekundärmarkt bezeichnet die Geschäftsbeziehung zwischen Käufer und Verkäufer eines Produkts. (Keine Erstveräusserung)

Emittent
Bezeichnet diejenige Partei die ein Produkt zeichnen lässt. Meistens ist dies eine Bank.

Basiswert
Ein strukturiertes Produkt besteht aus mehreren Basisfinanzprodukten, welche als Basiswerte bezeichnet werden.

Börsengang
Das Erstmalige Angebot eines Wertpapier auf dem Markt, von diesem Zeitpunkt an kann damit gehandelt werden

IPO
Initial Public Offering (siehe Börsengang)

Zertifikat
…

Nominalwert
(auch Nennwert genannt) Gibt an, welchen „gesetzlichen Wert“ ein Zahlungsmittel hat. Normalerweise hat eine Anleihe einen festen Nennwert und ihr aktueller Kurs wird relativ zum Nennwert angegeben. Ein Produkt mit Nominalwert CHF 100 zum Kurs von 110% kostet folglich CHF 110. Der Nennwert berechnet sich aus dem Grundkapital der Aktiengesellschaft pro emittierte Aktie.

Kurs
Der Kurs eines Wertpapiers zeigt das Verhältnis zwischen Angebot und Nachfrage. Der Kurs wird so angesetzt, damit sich aus der Gesamtheit aller „Orders“, am meisten erfüllen lassen.

Order
Ein Ankauf- bzw. Verkaufswunsch. Aus dem Verhältnis der Ankaufs- und Verkaufswünsche wird der Kurs eines Produkts bestimmt.

Ertrag Nostro
Bezeichnet den erzielten Ertrag der Cat FP eines Trades nach allen Abzügen (Retro-Betrag, Vertrieb, etc.)

Retro-Betrag
Entgelt für den Kunden

Vertrieb
Vermittlung eines Kunden. Die Vermittlung ist oft mit einem Entgelt verbunden, welches dem Vermittler gezahlt wird.

Übertrag
Bezeichnet den Vorgang bei welchem ein Produkt intern, von einem Kunden auf einen anderen, übertragen wird.

Kaufkurs
[%] Kurs für den Ankauf eines Produkts.

Verkaufskurs
[%] Kurs für den Verkauf eines Produkts.

Anfangsfixierungsdatum
…

Primärmarkttrade
Ein Trade vor dem Anfangsfixierungsdatum eines Produkts.

Sekundärmarkttrade
Ein Trade nach dem Anfangsvalutadatum eines Produkts.

Graumarkttransaktionen
Ein Trade vor dem Anfangsvalutadatum eines Produkts.

Anfangsvalutadatum
…

Zeichnungsperiode

Anfangsfixierung
Bei Lancierung eines Produktes beim Emittenten.

ISIN
International Securies Identifiaction Number, internationale eindeutige Identifikation eines Börsenproduktes

Valor

Tabelle 18 Glossar Finanzterminologie

23.2 STRUKTO MANAGER

Strukto Manager
Software zur Verwaltung von Strukturierten Finanzprodukten

STRUKTO

Die Datenbank des StruktoManagers.

CatWebService

Dieser Webservice ermöglicht das Veröffentlichen eines Produktes auf www.catfp.ch. Der Service bietet mehrere Methoden an, die es erlauben, kundenrelevante Produktdaten an den Webserver zu übermitteln.

Barriere

Der StruktoManager erlaubt die Definition mehrerer Barrieren pro Produkt. Eine Barriere kann Upper- bzw. Lower-Bound sein. Sobald der Kurs eine Barriere überschreitet geschieht WAS?

Produkt LifeCycle

1. Produktidee
2. Aushandeln der besten Konditionen mit Emittenten.
3. Sammeln von Kundengeldern für Lancierung des Produktes.
4. Produkt befindet sich zwischen Lancierung und Anfangsfixierung im Primärmarkt.
5. Anfangsfixierung
6. Produkt befindet sich zwischen Anfangsfixierung und Valutadatum im Graumarkt.
7. Lebenszeit des Produktes ab Valutadatum. Ca. 2 Wochen nach Anfangsfixierung
8. Handel nach Valuta Datum (Sekundärmarkt)
9. Ablauf des Produktes. Produkt wird zurückbezahlt. (Rückzahlungsdatum)

StruktoManager1.x

Bezeichnet explizit jene Version des StruktoManagers, die vor und während der Semesterarbeit, im produktiven Betrieb von Cat Financial Products, im Einsatz war.

SturktoManager2.0

Bezeichnet explizit jene Version des StruktoManagers, die im Laufe der Semesterarbeit erstellt wird.

Tabelle 19 Glossar StruktoManager

23.3 ORGANISATIONEN UND PERSONEN

Cat Financial Products

Auftraggeber der Studienarbeit. Abteilung für strukturierte Produkte der Triaxis Trust AG.

Cat FP

Siehe Cat Financial Products

www.catfp.ch

Webseite von Cat FP. Jeder Kunde kann auf dieser Seite sein persönliches Portfolio, mit den veröffentlichten Produkten des StruktoManagers, verwalten.

Triaxis Trust AG

Teilunternehmung der Cat Group AG. Besteht aus mehreren Abteilungen von deren eine Cat Financial Products ist.

Cat Group AG

Holdinggesellschaft der Triaxis Trust AG.

Giuliano Glocker

Head Financial Products der Cat Financial Products.

Patrick Walker

Erster Ansprechpartner für diese SA bei Fragen die den „alten“ StruktoManager betreffen. Projektmanager des CatFP-internen Projekts „Erneuerung des StruktoManagers“.

Julia Savina

Assistentin G. Glocker.

LivingTech

Web-Hosting Firma. Hostet www.catfp.ch, und stellt den Webservice „CatWebService“ zur Verfügung.

Tabelle 20 Glossar Organisationen und Personen

24 ABBILDUNGSVERZEICHNIS

25 TABELLENVERZEICHNIS

26 QUELLENVERZEICHNIS

MSDN1 MSDN, http://msdn.microsoft.com/en-us/library/dd537609.aspx (Stand: März 2011)]

MSDN2 MSDN, http://msdn.microsoft.com/en-us/library/dd997396.aspx (Stand: März 2011)]