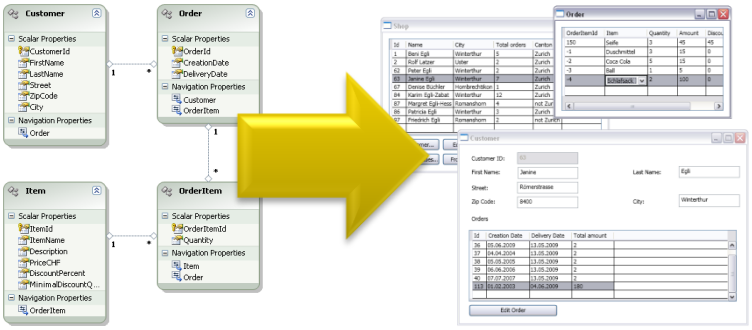
I Management Summary

Ausgangslage



Applikationen, die Daten visualisieren, zeigen meist nur einen gewünschten Ausschnitt der Daten. Daher gibt es das Konzept der *Views*, das einem die Definition solcher Ausschnitte erlaubt. Konventionelle Konzepte, wie z.B. das der Datenbank-Views, haben jedoch ihre Einschränkungen:

- Sollen die Ansichten editierbar sein, dann muss dies der Entwickler ausprogrammieren.
- Ändert sich der Inhalt der Datenquelle, dann sieht der Benutzer veraltete Daten.
- Ein Zusammenspiel der Views untereinander (z.B. Master-Detail-Ansicht) muss vom Entwickler ebenfalls umständlich programmiert werden.
- Für den Einsatz in einer verteilten Umgebung muss Service- und Client-Software geschrieben werden.

Mit Object Views wird dem Entwickler den Umgang mit Views leichter gemacht. Sie sind das moderne Werkzeug, um einfach und ohne grossen Aufwand dynamische und interaktive Ansichten zu erstellen.

Ziele

Mit Object Views sollen Views auf Data Object Graphen (DOG) realisiert werden können. Ein DOG besteht aus Datenobjekten und ihren Beziehungen zueinander, z.B. ein Kunde mitsamt seinen Bestellungen. Es ist dabei nicht relevant, woher dieser DOG geladen wird, sei dies nun eine Datenbank oder eine andere Datenquelle.

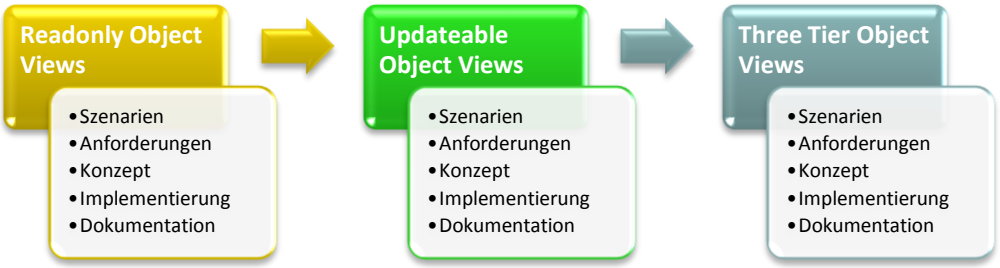
Die Ziele der Object Views sind:

- Sie lassen sich vom Programmierer direkt im Programmcode definieren.
- Sie aktualisieren sich bei Datenänderungen gegenseitig.
- Sie sind editierbar (Einfügen, Bearbeiten, Löschen).
- Sie bieten die Möglichkeit, Änderungen rückgängig zu machen und wiederherzustellen.
- Sie lassen sich auch in einer verteilten Umgebung einsetzen.

Projektbeteiligte

Person	Funktion
Benjamin Egli	Projektmitglied
Rolf Latzer	Projektmitglied
Hansjörg Huser	Betreuer & Verantwortlicher
Alain Schneble	Betreuer
Simon Gubler	Betreuer & Integration in pmMDA
S. Zettel	Experte
Markus Stolze	Gegenleser
bbv Software Services	Industriepartner

Vorgehen



Die Entwicklung des Frameworks wurde in drei Schritte aufgeteilt:

- **Readonly Object Views**: Views in einer Single oder Two Tier Umgebung, deren Datenänderungen nicht persistent gespeichert werden.
- **Updateable Object Views**: Views in einer Single oder Two Tier Umgebung, deren Datenänderungen persistent gespeichert werden können.
- **Three Tier Object Views:** Views in einer verteilten Umgebung.

Bei jedem der drei Schritte wurden folgende Themen erarbeitet:

- **Szenarien**: Die Szenarien dienen dazu, die Problemstellung zu verstehen und das gesamte Problem in Teilprobleme zu unterteilen.
- **Anforderungen**: Definition der funktionalen und nichtfunktionalen Anforderungen an das Object Views Framework.
- **Konzept**: Die Konzepte zeigen auf, wie die gestellten Anforderungen erfüllt werden.
- **Implementierung**: Programmierung, Unit Tests, Systemtests und Code Reviews.
- **Dokumentation**: Die Dokumentierung der durchgeführten Arbeiten und Beschreibung der Lösungen.

Natürlich wurde das Projekt von einem Projektmanagement, inkl. Zeit-, Risiko- und Qualitätsmanagement begleitet.

Ergebnisse

Das Projekt wurde erfolgreich durchgeführt. Es wurden 18 von 19 funktionalen Anforderungen erfüllt und sämtliche formulierten Ziele wurden erreicht:

Ziel	Ergebnis
Sie lassen sich vom Programmierer direkt im Programmcode definieren	Es müssen nicht für jede Applikation in der Datenbank Views erstellt werden, sondern diese können direkt im Programmcode definiert werden und sind dadurch direkt Bestandteil der Applikation.
Sie aktualisieren sich bei Datenänderungen gegenseitig	Werden in verschiedenen Object Views dieselben Felder von Datenobjekten angezeigt, so wird die Datenkonsistenz gewährleistet.
Sie sind editierbar (Einfügen, Bearbeiten, Löschen)	Man kann mit sehr wenig Aufwand sämtliche Änderungen in einer Datenquelle persistieren. Wird beispielsweise das Entity Framework eingesetzt, kann mit vier Zeilen Code jede beliebige Änderung gespeichert werden.
Sie bieten die Möglichkeit, Änderungen rückgängig zu machen und wiederherzustellen	Das Framework zeichnet auf, welche Änderungen an den Daten vollzogen werden. Diese Änderungen können sowohl rückgängig, als auch wiederhergestellt werden.
Sie lassen sich auch in einer verteilten Umgebung einsetzen	Es ist möglich, das Framework in einer Three Tier Umgebung einzusetzen. Dabei sind auf Seite des Clients die Object Views und auf Seite des Servers die Services für das Auslesen sowie das Persistieren der Daten.

Ausblick

Das Framework wird voraussichtlich beim Industriepartner in dessen pmMDA Framework integriert.

II Abstract

Aufgaben-stellung

Applikationen, die Daten visualisieren, zeigen meist nur einen gewünschten Ausschnitt der Daten. Daher gibt es das Konzept der *Views*, das einem die Definition solcher Ausschnitte erlaubt.

Konventionelle Konzepte, wie z.B. das der Datenbank-Views, haben jedoch ihre Einschränkungen. Solche Ansichten sind oft nur lesbar oder nur über Umwege editierbar. Ändert sich der Inhalt der Datenquelle, dann sieht der Benutzer veraltete Daten. Ein Zusammenspiel der Views untereinander (z.B. Master-Detail-Ansicht) muss vom Programmierer umständlich programmiert werden. Kurzum: Herkömmliche Views sind zu statisch und zu umständlich in der Handhabung.

Mit Object Views soll ein Framework entwickelt werden, das von Applikationen verwendet werden kann, um Ansichten zu erstellen. Die Object Views sollen die Nachteile konventioneller Views beheben, einen komfortableren Umgang ermöglichen und die folgend beschriebene Funktionalität anbieten.

Ziele

Mit Object Views sollen *Views auf Data Object Graphen (DOG)* realisiert werden können. Ein DOG besteht aus Datenobjekten und ihren Beziehungen zueinander, z.B. ein Kunde mitsamt seinen Bestellungen. Es ist dabei nicht relevant, woher dieser DOG geladen wird, sei dies nun eine Datenbank oder eine andere Datenquelle.

Die Object Views werden vom Programmierer im Programmcode *als Linq-Queries* definiert. Das Object Views Framework sorgt dafür, dass immer nur die *nötigsten Daten* geladen werden, dass die Ansichten *bearbeitet* werden können und dass bei Datenmanipulationen betroffene Views *aktualisiert* werden. Datenänderungen sollen auch *rückgängig gemacht* und *wiederhergestellt* werden können.

Das Framework soll zudem auch in einer *verteilten Umgebung* eingesetzt werden können.

Lösung

Das Framework wurde in drei Schritten entwickelt.

Mit den *Readonly Object Views* können Views auf DOGs in einer Single oder Two Tier Umgebung erstellt werden. Die Daten in den Views können zwar bearbeitet, die Änderungen können jedoch nicht persistiert werden. Wenn sich die Daten ändern, dann werden die Ansichten automatisch aktualisiert. Es ist die Aktualisierung sämtlicher denkbarer Fälle möglich (Projektionen, Selektionen, Verbunde, zusammengesetzte Werte, berechnete Werte und aggregierte Werte).

Mit den *Updateable Object Views* werden die Datenänderungen in den Views zusätzlich persistierbar und es wird eine Undo und Redo Funktionalität angeboten. Das Speichern kann in beliebigen Datenquellen erfolgen, solange dafür eine Implementierung ins Framework eingebunden ist. Aktuell unterstützt wird das Persistieren in einem Entity Framework Datenkontext und in einfachen Listen von Objekten.

Die *Three Tier Object Views* erlauben schliesslich auch den Einsatz der Views in einer verteilten Umgebung. Dazu wird *Interlinq* verwendet, eine Softwarekomponente, die das Lesen von Daten aus entfernten Datenquellen erlaubt. Das Schreiben der Daten ist mittels einem eigenen WCF-Service realisiert.

III Inhaltsverzeichnis

IV Abbildungsverzeichnis

1 Projektauftrag

Projektbezeichnung

Object Views

Aufraggeber

Der Auftrag für das Projekt kommt vom INS Institute for Networked Solutions. Anlass für diesen Auftrag ist das Projekt pmMDA, das von der Firma bbv Software Systems AG entwickelt wird.

Projektbeginn und -ende

Das Projekt dauert vom 16.02.2009 bis am 12.06.2009.

Projektteam

Das Projekt wird im Rahmen der Bachelorarbeit von Rolf Latzer und Benjamin Egli durchgeführt. Betreut werden sie von Hansjörg Huser und Alain Schneble.

Kurzbeschreibung und Ziele

Mit Object Views sollen Views auf Data Object Graphen (DOG) realisiert werden können. Es ist dabei nicht relevant, woher dieser DOG geladen wird, sei dies nun eine Datenbank, oder eine andere Datenquelle.

Diese Views werden vom Programmierer im Programmcode als Queries definiert. Das Object Views Framework sorgt dafür, dass immer nur die nötigsten Daten geladen werden, dass die Views bearbeitet werden können und dass bei Datenmanipulationen betroffene Views aktualisiert werden. Das Framework soll zudem auch in einer verteilten Umgebung eingesetzt werden können.

Beispiel:

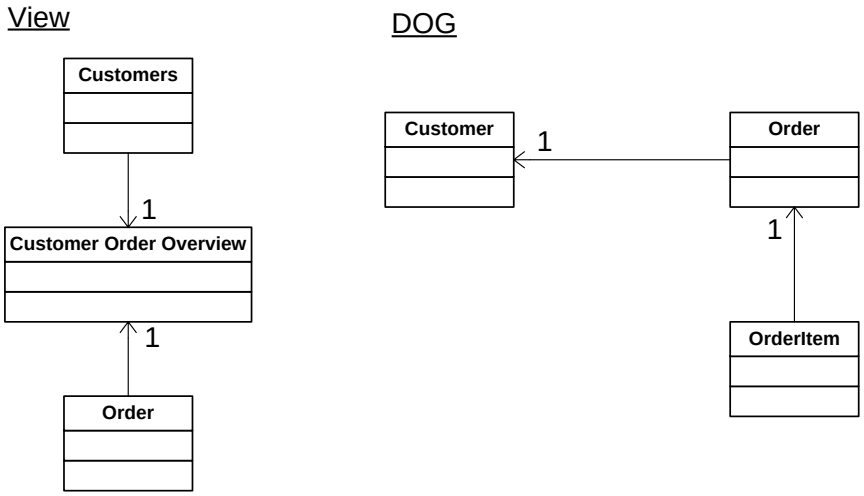


Abbildung 1 - Beispiel einer View auf einen Data Object Graphen

Es ist ersichtlich, dass wenn sich etwas an einem Kunden bzw. an einer Bestellung ändert, dies automatisch auch in der Übersicht geändert werden muss. Diese Abhängigkeit soll das Framework übernehmen.

Das Framework soll also eine Vereinfachung herbeiführen, welche dem Programmierer erlaubt, Views auf Daten mit Abhängigkeiten möglichst einfach und rasch zu realisieren.

Projektergebnisse

Die erwarteten Ziele des Projekts sind ein lauffähiges, getestetes System und eine Dokumentation des Frameworks. Zudem muss eine Beispielanwendung zur Verfügung stehen, welches die Möglichkeiten des Frameworks demonstriert.

Das Framework soll Funktionen zu verschiedenen Themenbereichen bieten:

- Readonly Object Views:
  - Views auf lokale Datenstrukturen
  - Views auf lokale Datenbank
  - Aggregationen wie z.B. Summen, Counts, usw.
- Updateable Object Views:
  - Bearbeitbare Views
  - Change Tracking
  - Konfliktsituationen lösen
  - Aktualisierung anderer Views
  - Dynamisches Nachladen der Businessobjekte (Lazy / Eager Loading)
- Three Tier Object Views:
  - Views in verteilten Systemen
  - Mit Dataservices
  - Mit Interlinq
- Randbedingungen:
  - Das Framework soll mit unterschiedlichen Technologien (Linq To Objects, Linq To Sql, Linq To Entities, etc.) eingesetzt werden können
  - Wenn möglich: Keine Anforderungen an DOG's (keine gemeinsame Basisklasse usw.)
  - Das Framework sollte einfach einzusetzen sein und keinen riesigen Konfigurationsaufwand voraussetzen

Annahmen und Beschränkungen

Das Framework wird zwar im Hinblick darauf entwickelt, im Projekt pmMDA eingesetzt werden zu können. Dennoch wird nicht auf konkrete Randbedingungen dieses Projekts eingegangen, sondern es wird ein Framework entwickelt, dass generell eingesetzt werden kann, wenn folgende Punkte gegeben sind:

- .Net: Das Framework kann nur in der .Net Laufzeitumgebung eingesetzt werden
- Linq und Queryable: Auf die Datenstruktur muss mit Linq Queries zugegriffen werden können
- Interlinq: In verteilten Systemen wird Interlinq zur Übertragung der Queries eingesetzt

2 Einführung

Mit diesem Kapitel wird ein erster Einblick in die Object Views gewährt. Es erklärt die Motivation für die Entwicklung dieser neuen Art von Ansichten.

2.1 Motivation

Applikationen, die Daten visualisieren, zeigen meist nur einen gewünschten Ausschnitt der Daten. Daher gibt es das Konzept der *Views*, das einem die Definition solcher Ausschnitte erlaubt.

Konventionelle Konzepte, wie z.B. das der Datenbank-Views, haben jedoch ihre Einschränkungen. Solche Ansichten sind oft nur lesbar oder nur über Umwege editierbar. Ändern sich im Hintergrund die Daten, dann sieht der Benutzer veraltete Daten. Ein Zusammenspiel der Views untereinander (z.B. Master-Detail-Ansicht) muss vom Programmierer umständlich programmiert werden. Kurzum: Herkömmliche Views sind zu statisch und zu umständlich in der Handhabung.

Mit Object Views werden solche Einschränkungen beseitigt. Im Vergleich zu konventionellen Views haben sie folgende Vorteile:

- Sie lassen sich vom Programmierer direkt im Programmcode definieren.
- Sie aktualisieren sich bei Datenänderungen gegenseitig.
- Sie sind editierbar (Einfügen, Bearbeiten, Löschen).
- Sie bieten die Möglichkeit, Änderungen rückgängig zu machen und wiederherzustellen.
- Sie lassen sich auch in einer verteilten Umgebung einsetzen.

Im folgenden Kapitel wird anhand einer Demoapplikation erläutert, in welchen Situationen Object Views eingesetzt werden und wie man sich damit Programmierarbeit ersparen kann.

2.2 Demoapplikation

2.2.1 Übersicht

Die Demoapplikation ist ein Shop mit Kunden (Customer), die Bestellungen (Order) tätigen. Die Bestellungen bestehen aus einer Anzahl von Bestellzeilen (OrderItem), die sich jeweils auf einen bestimmten Artikel (Item) beziehen.

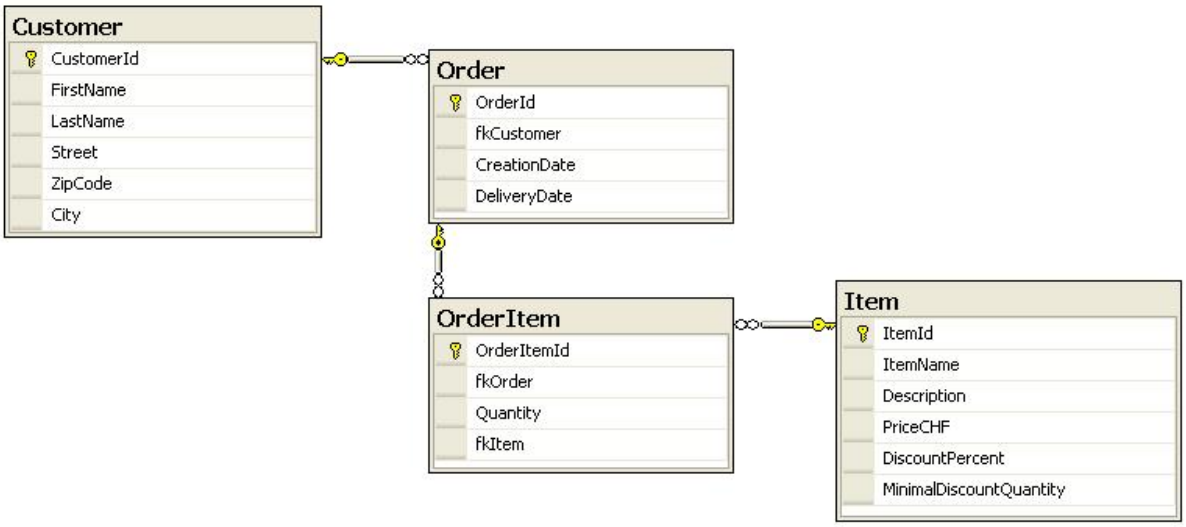


Abbildung 2 - Datenbankdiagramm Demoapplikation

2.2.2 Anwendungsfall 1

Als Erstes wollen wir eine View, die uns eine Liste der Kunden anzeigt, jedoch nicht sämtliche Daten der Kunden. Zusätzlich soll angezeigt werden, welcher Kunde wie viele Bestellungen getätigt hat. Zudem soll die View editierbar sein.

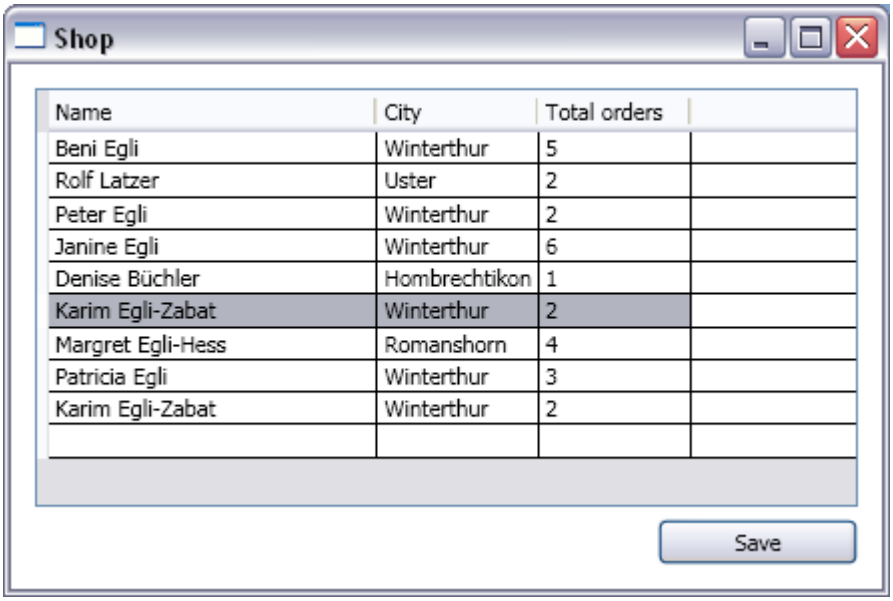


Abbildung 3 - Demoapplikation Screenshot Shop

Eine solche View könnte auch mit konventionellen Views erstellt werden. Die View editierbar zu machen, wäre jedoch mit Programmierarbeit verbunden. Zudem stellt sich die Frage nach dem Verhalten, wenn sich die Daten eines Kunden verändern. Um dies müsste sich der Programmierer ebenfalls kümmern.

Mit Object Views erstellt der Programmierer die View und dann ist die Sache für ihn erledigt. Aktualisierungen und Speichern der View übernimmt das Object Views Framework.

2.2.3 Anwendungsfall 2

Als zweiten Anwendungsfall betrachten wir die Master-Detail-Beziehung zwischen dem Kunden und seinen Bestellungen. Nehmen wir an, man kann sich für die Kunden in der obigen View ein Detail anzeigen lassen, das einem sämtliche Daten des Kunden und eine Übersicht über seine Bestellung präsentiert.

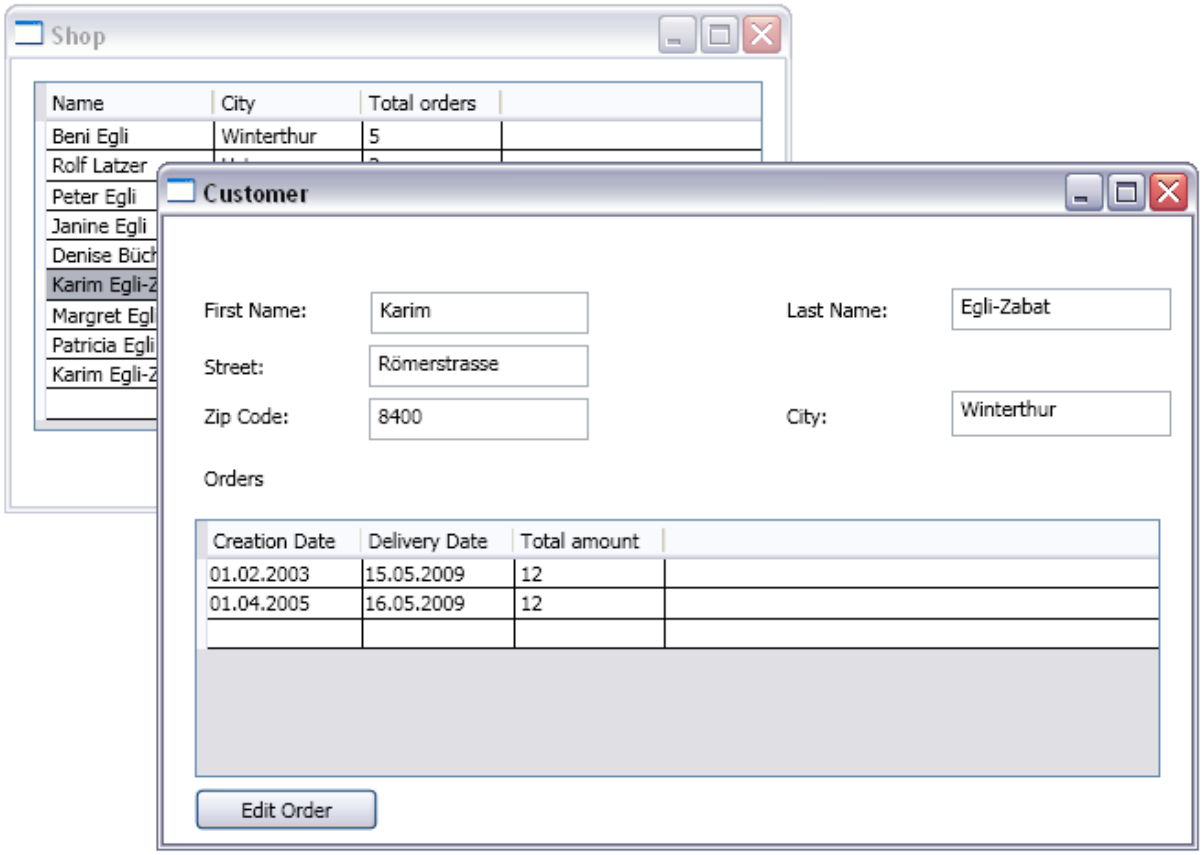


Abbildung 4 - Demoapplikation Screenshot Customer

Man stelle sich eine solche Ansicht als zwei Views vor: Eine View zur Darstellung der Daten des Kunden und eine View für die Bestellübersicht.

Was passiert nun, wenn man in einer dieser beiden Views Änderungen vornimmt? Was geschieht in der Kundenübersicht (der View aus Anwendungsfall 1), wenn man z.B. den Vornamen des Kunden editiert? Oder wenn man eine zusätzliche Bestellung erfasst? Die Anzahl Bestellungen müsste sich verändern in der anderen Ansicht.

Mit Object Views beeinflussen sich die Views gegenseitig. Aktualisierungen der Views geschehen automatisch. Dies umfasst sowohl simple Eins-zu-Eins-Felder (wie z.B. das Feld „City“), zusammengesetzte oder berechnete Felder (z.B. das Feld „Name“ in der Kundenübersicht, das sich aus Vorname und Nachname zusammensetzt), als auch aggregierte Felder (z.B. „Total orders“).

2.2.4 Anwendungsfall 3

Öffnet man das Detail für eine einzelne Bestellung, dann präsentiert einem eine View alle Bestellzeilen inkl. Betrag pro Zeile. Eine Bestellzeile bezieht sich dabei immer auf einen bestimmten Artikel, den man beim Erfassen einer neuen Bestellzeile aus einer Liste auswählen kann. Die View enthält also Daten aus mehreren Datenbanktabellen, nämlich aus *OrderItem* und *Item*.

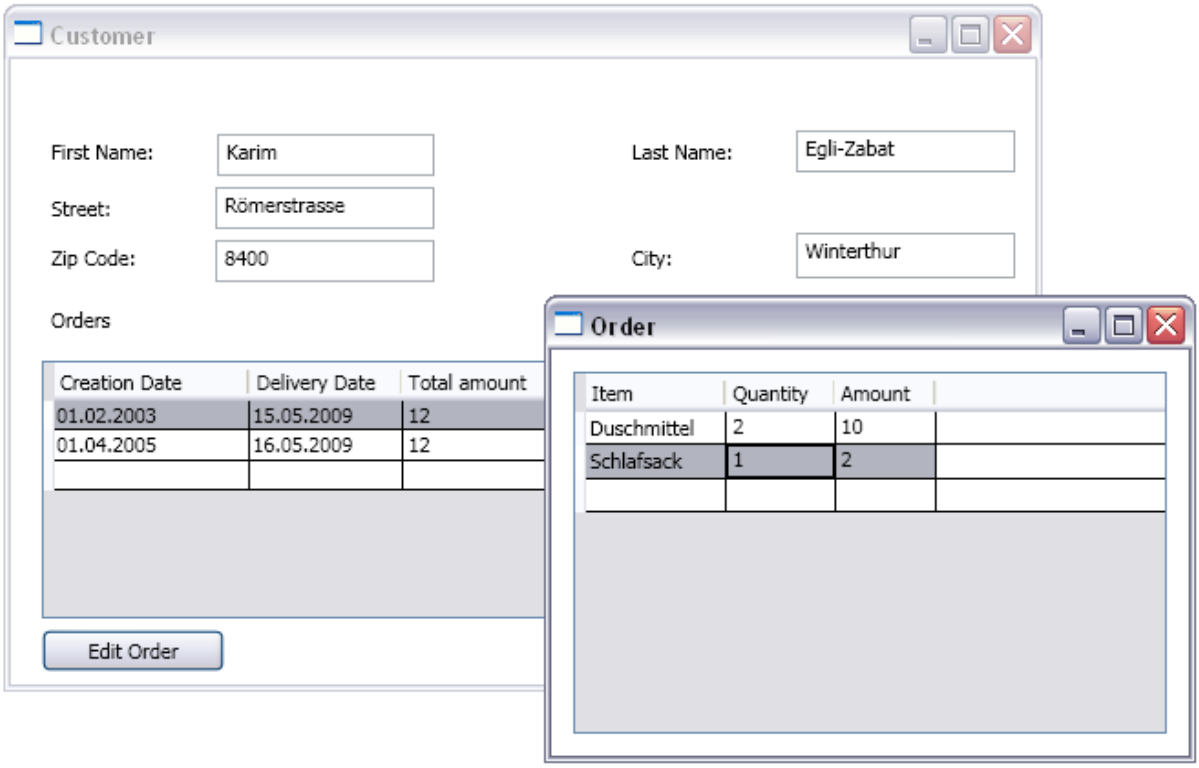


Abbildung 5 - Demoapplikation Screenshot Order

Wenn man also eine neue Bestellzeile erfasst, dann soll in der Datenbanktabelle *OrderItem* einen Eintrag hinzugefügt werden, in der Tabelle *Item* jedoch nicht. Und was geschieht, wenn man in der Bestellübersicht eine Bestellung löscht? Die Bestellzeilen müssen mitgelöscht werden, nicht jedoch die Artikel.

Dieses Verhalten entspricht der Businesslogik und muss vom Programmierer ausprogrammiert werden.

Object Views nehmen dem Programmierer diese Arbeit ebenfalls ab. Sie sind konfigurierbar, so dass der Programmierer gemäss der Businesslogik angeben kann, was erstellt resp. gelöscht werden soll und was nicht. Der Rest übernimmt das Framework.

2.2.5 Anwendungsfall 4

Betrachten wir eine beliebige View. Der Benutzer manipuliert Daten in der View, er fügt neue Zeilen ein, bearbeitet gewisse Werte, löscht bestehende Zeilen, etc. Schnell gerät er in eine Situation, in der er getätigte Änderungen rückgängig machen oder rückgängig gemachte Änderungen wiederherstellen will.

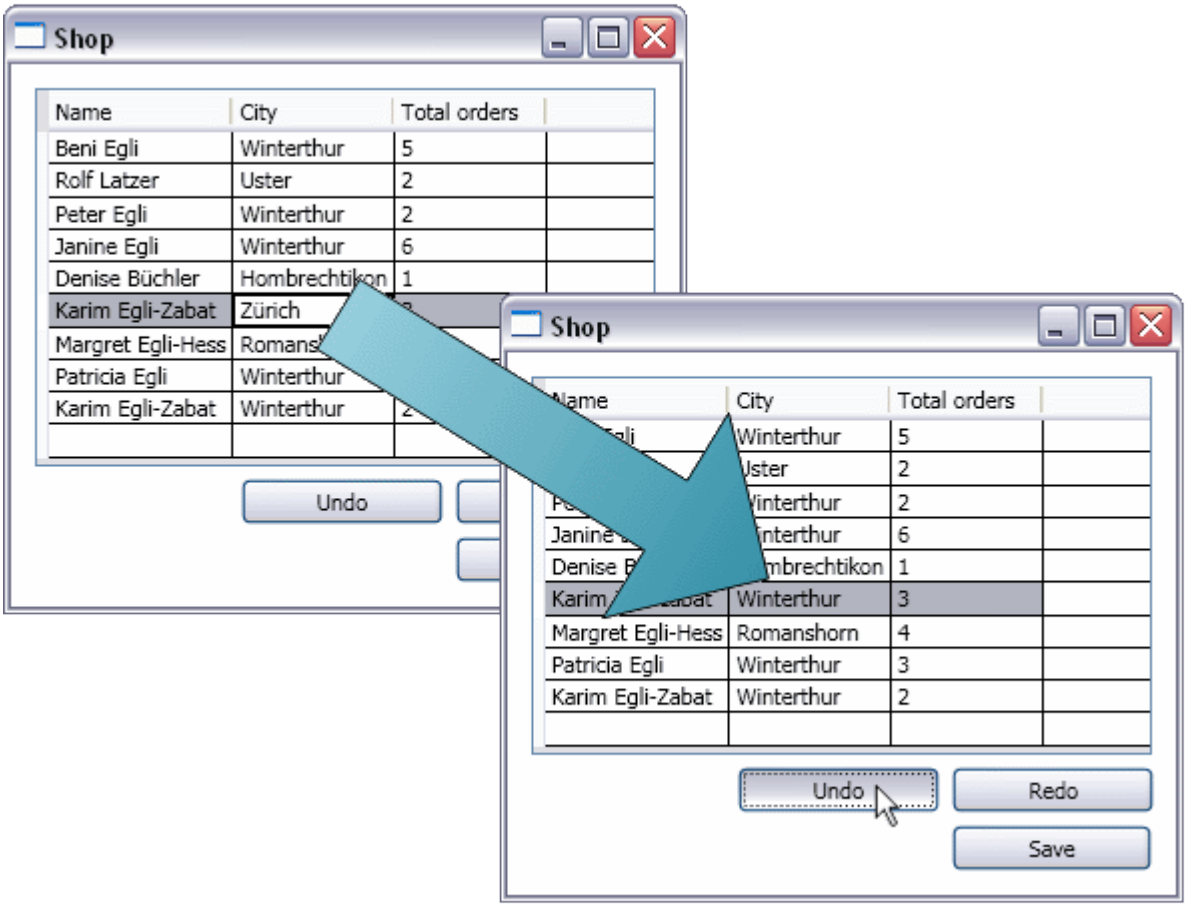


Abbildung 6 - Demoapplikation Undo/Redo

Eine solche mehrstufige Undo/Redo Funktionalität wird von den meisten modernen Applikationen angeboten. Auch diesen Programmieraufwand nehmen einem die Object Views ab, indem sie eine integrierte Lösung für das Undo und Redo zur Verfügung stellen.

2.2.6 Anwendungsfall 5

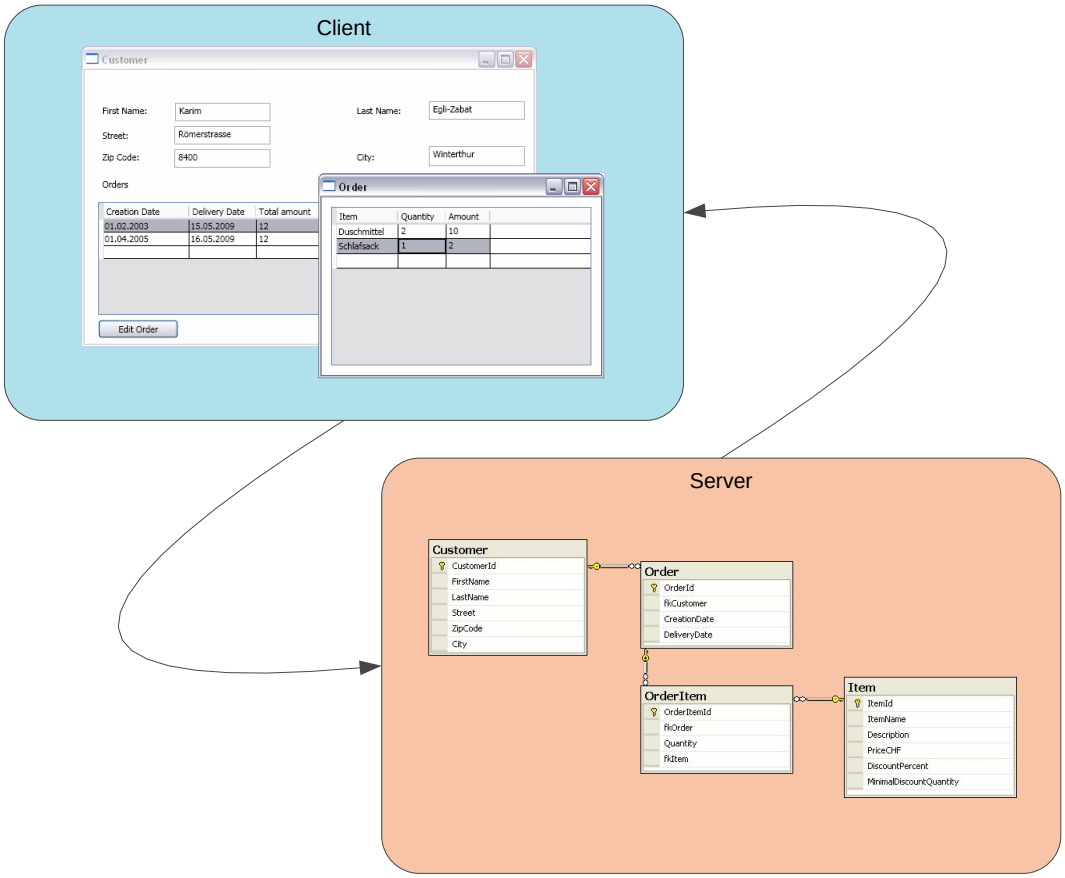


Abbildung 7 - Demoapplikation in verteilter Umgebung

Sie wollen alle eben beschriebenen Views auch in einer verteilten Umgebung (Three Tier) einsetzen können. Die Verwendung von konventionellen Views würde dann das mühselige Programmieren von Service- und Client-Software bedeuten, das Serialisieren von Daten, ev. das Erstellen von DTOs (Data Transfer Objects) und was sonst noch alles dazu gehört.

Mit Object Views ist auch dies kein Problem. Sie sind in einer Single, Two sowie Three Tier Umgebung immer gleich und ohne Einschränkungen einsetzbar.

3 Szenarien

In diesem Kapitel werden die Anwendungsfälle für die Object Views erläutert. Es liefert Antworten auf folgende Fragen: Welche Szenarien decken die Views ab? Was kann mit den Ansichten erreicht werden?

3.1 Readonly Object Views

3.1.1 Übersicht

Mit Readonly Object Views sind Views in einer Single oder Two Tier Umgebung gemeint, deren Datenänderungen nicht persistent gespeichert werden. D.h. die Object View kann zwar Daten anzeigen und die Daten können bearbeitet werden, die Änderungen werden jedoch nicht gespeichert.

Für diese Art von Object Views sind folgende Szenarien möglich:

- Projektion: Mit der View werden nur gewisse Felder der Data Objects angezeigt.
- Selektion: Mit der View wird eine Auswahl aus einer Data Object Collection angezeigt.
- Verbund: Die View setzt sich aus Data Objects verschiedener Klassen zusammen.
- Zusammengesetzte Werte: Die View enthält zusätzliche Felder, die sich aus anderen Feldern der View oder aus mehreren Feldern der Data Objects berechnen oder zusammensetzen.
- Aggregierte Werte: Es soll möglich sein z.B. Summen oder andere Aggregatsfunktionen zu nutzen.
- Aktualisierung: Wenn sich die Daten der Data Objects verändern, dann müssen sich die davon betroffenen Object Views aktualisieren.

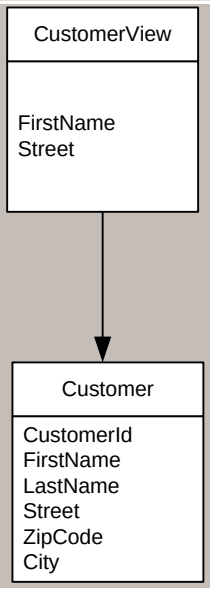
Alle Szenarien gelten sowohl für:

- Typisierte und untypisierte Views
- Single Tier (mit Listen) und Two Tier (mit Datenbanken)

3.1.2 Projektion

Szenario 1.1	Mit der View werden nur gewisse Felder der Data Objects angezeigt.
Beispiel	Das Beispiel zeigt in der View nur den Vornamen und die Strasse eines Kunden an und nicht die kompletten Informationen des Data Objects.

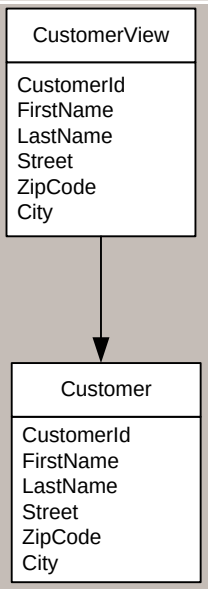
Query	`from customer in db.Customers`
	`select new CustomerView {FirstName = customer.FirstName,`
	`Street = customer.Street}`

Diagramm	

3.1.3 Selektion

Szenario 2.1	Mit der View wird eine Auswahl aus einer Data Object Collection angezeigt.
Beispiel	Es sollen in der View nur alle Kunden angezeigt werden, die in Zürich wohnen.

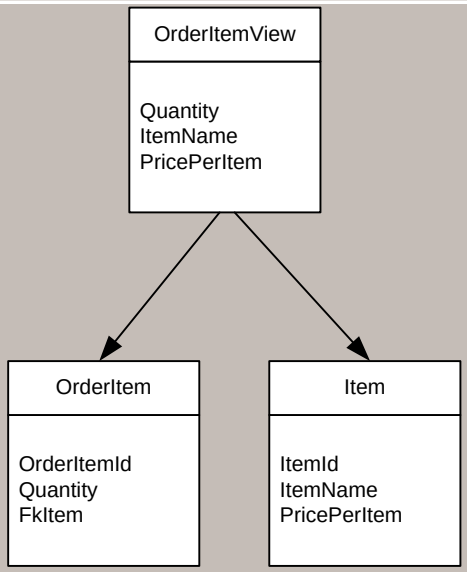
Query	`from customer in db.Customers`
	`where customer.City.Equals("Zürich")`
	`select new CustomerView {…}`

Diagramm	

3.1.4 Verbund

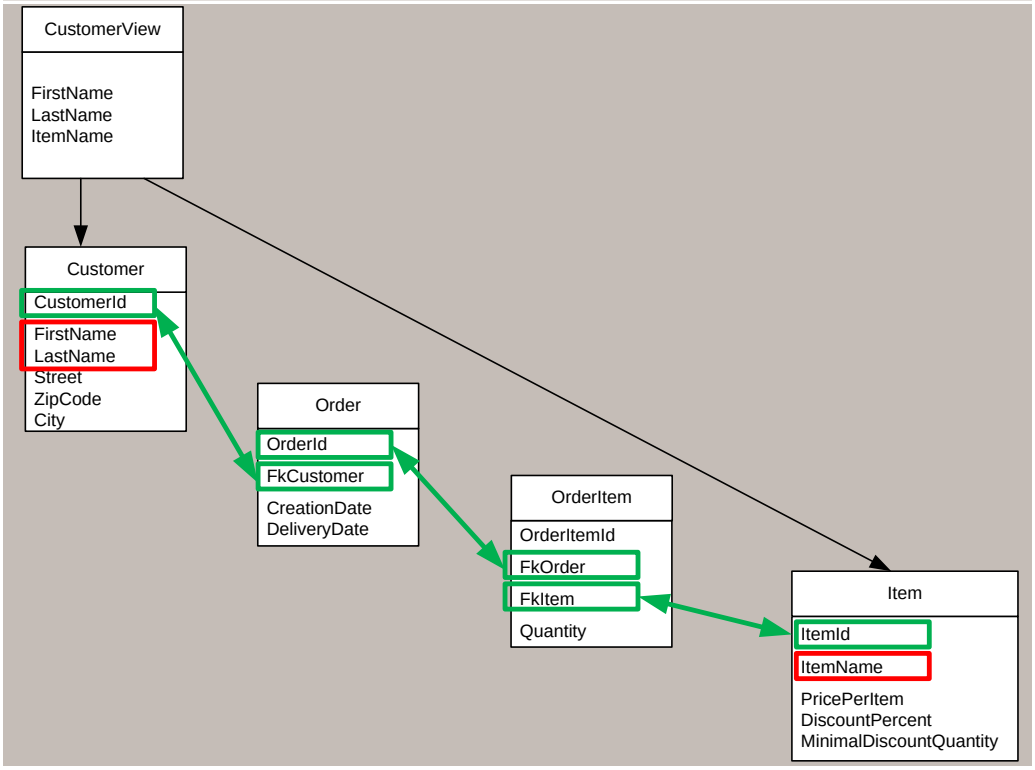
Szenario 3.1	Die View setzt sich aus Data Objects verschiedener Klassen zusammen.
Beispiel	Beispielsweise sollen die Anzahl der bestellten Produkte sowie der Produktname als auch der Preis pro Produkt angezeigt werden können.

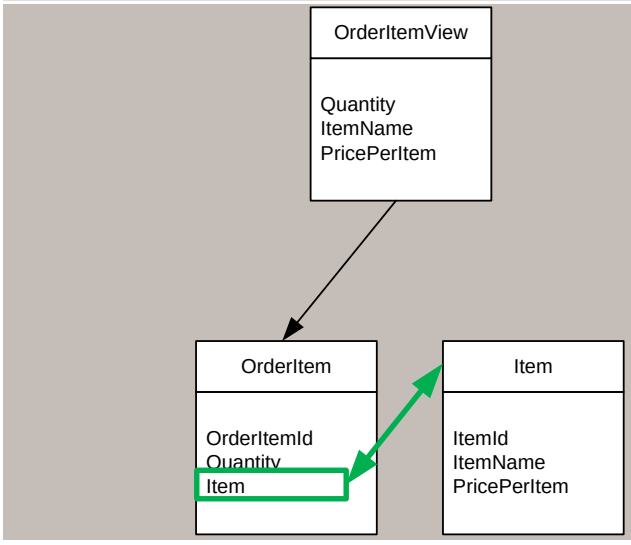
Query	`from orderItem in db.OrderItems`
	`join item in db.Items on item.ItemId equals orderItem.FkItem`
	`select new OrderItemView {Quantity = orderItem.Quantity,`
	`ItemName = item.ItemName, PricePerItem = item.PricePerItem}`

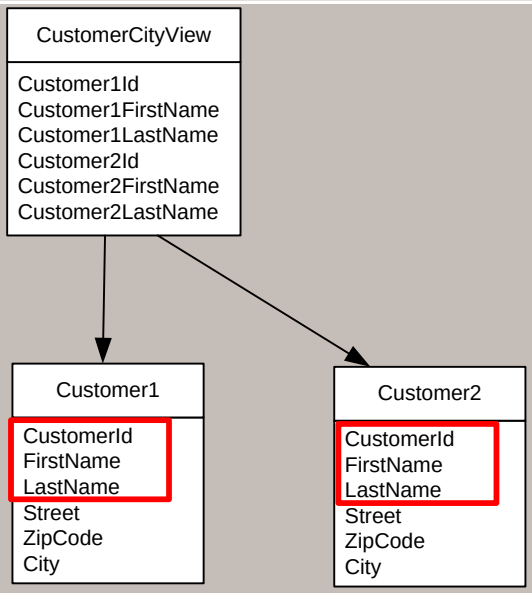
Diagramm	

Szenario 3.2	Die Tabellen sollen beliebig oft verknüpft werden können.
Beispiel	Einen Kunden mit allen bestellten Items anzeigen. Dazu muss über sämtliche DOG-Klassen iteriert werden.

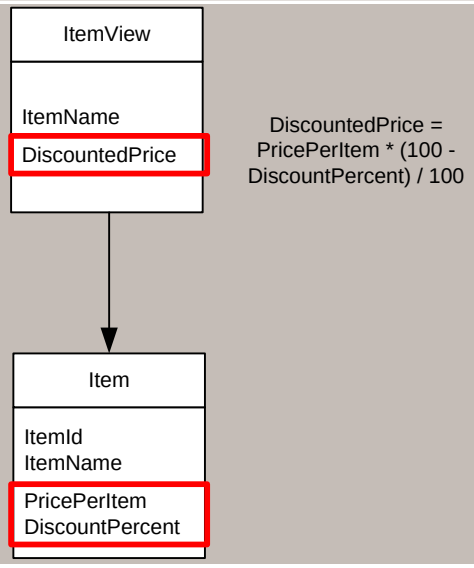
Query	`select customer in db.Customers`
	`join order in db.Orders on order.FkCustomer equals customer.CustomerId`
	`join orderItem in db.OrderItems on orderItem.FkOrder equals order.OrderId`
	`join item in db.Items on item.ItemId equals orderItem.FkItem`
	`select new CustomerView {…}`

Diagramm	

Szenario 3.3	Die Beziehungen des Data Object Graphen sind in den Data Object Klassen abgebildet.
Beispiel	Die Klasse OrderItem enthält ein Propety Item, über das direkt auf das Item Objekt zugegriffen werden kann.
Query	`from orderItem in db.OrderItems` `select new OrderItemView {Quantity = orderItem.Quantity, ItemName = orderItem.Item.ItemName, PricePerItem = orderItem.Item.PricePerItem}`
Diagramm	

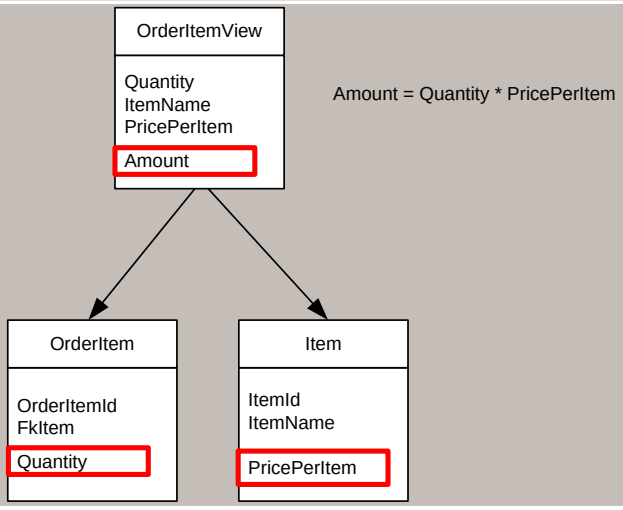
Szenario 3.4	Es soll möglich sein, dass mehrere Instanzen auf dieselbe Datentabelle, d.h. auf den gleichen Datentyp zugreifen können.
Beispiel	Es sollen zwei Kunden, die am selben Ort wohnen, angezeigt werden können.
Query	`from customer1 in db.Customers join customer2 in db.Customers on customer1.City equals customer2.City select new SameCityView { Customer1Id = customer1.Id, Customer1FirstName = customer1.FirstName, Customer1LastName = customer1.LastName, Customer2Id = customer2.Id, Customer2FirstName = customer2.FirstName, Customer2LastName = customer2.LastName };`
Diagramm	

3.1.5 Zusammengesetzte Werte

Szenario 4.1	Nicht jedes Mal kann ein Wert direkt auf einen Wert des Data Objects abgebildet werden, sondern muss durch einige mehrere Werte berechnet oder zusammengesetzt werden.
Beispiel	Es soll möglich sein, direkt den reduzierten Preis anzuzeigen.
Query	`from item in db.Items select new ItemView {ItemName = item.ItemName, DiscountedPrice = item.PricePerItem * (100 - item.DiscountPercent) / 100}`
Diagramm	

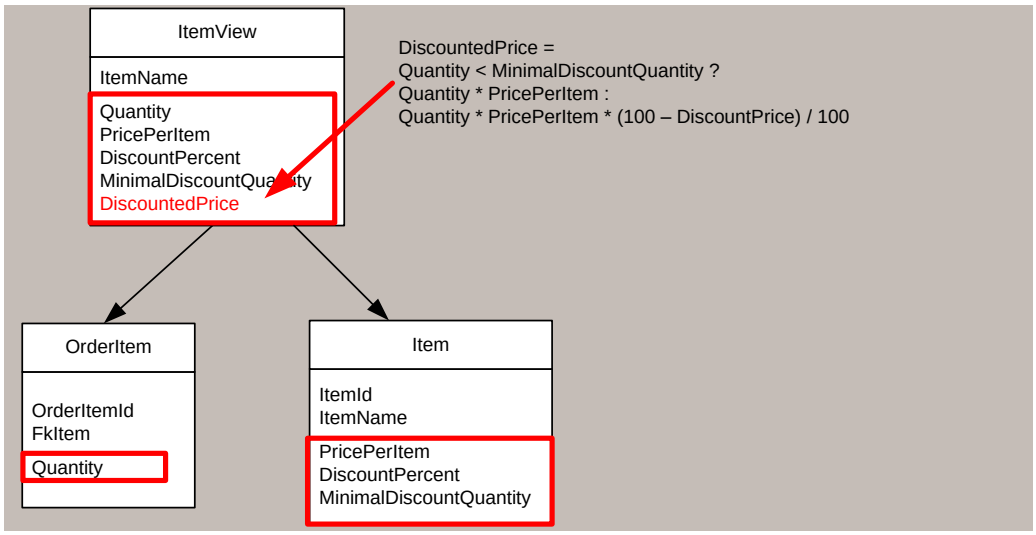
Szenario 4.2	**Die Berechnung kann auch Felder aus mehreren Data Object Klassen beinhalten. Dabei kann sich ein Object View Feld aus mehreren Data Object Feldern zusammensetzen, die aus unterschiedlichen Data Object Klassen stammen.**
Beispiel	Um den Preis eines bestellten Produktes zu berechnen, sind die bestellte Menge und der Preis pro Produkt zu berücksichtigen.

Query	`from orderItem in db.OrderItems`
	`join item in db.Items on item.ItemId equals orderItem.FkItem`
	`select new OrderItemView {Quantity = orderItem.Quantity,`
	`ItemName = item.ItemName, PricePerItem = item.PricePerItem,`
	`Amount = orderItem.Quantity * item.PricePerItem}`

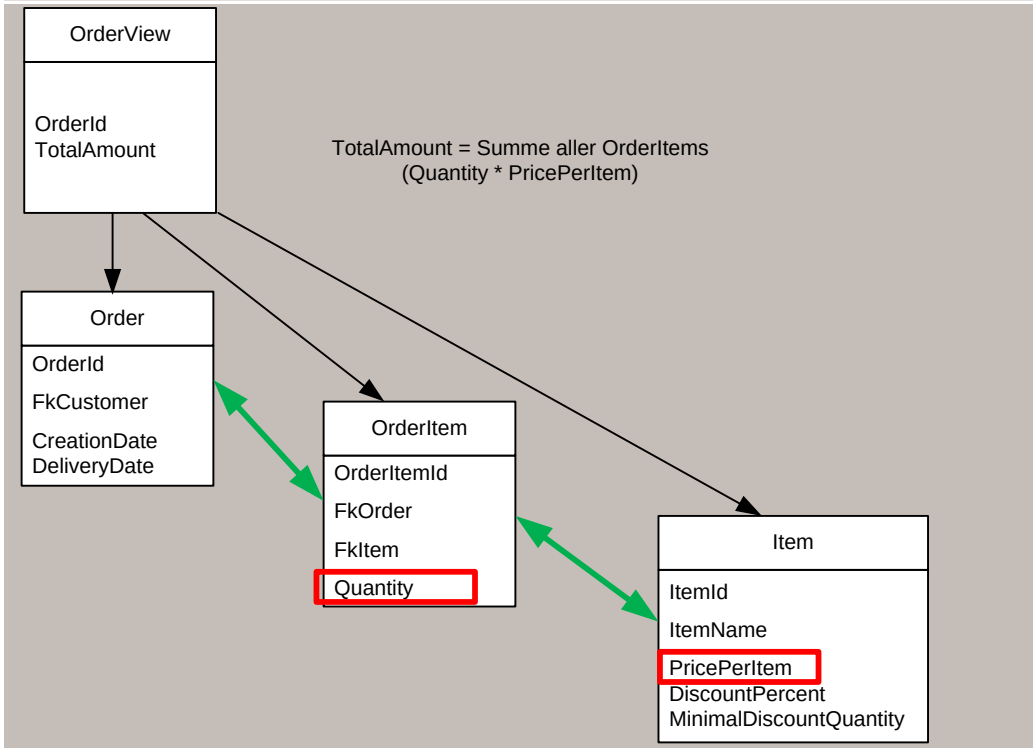
Diagramm	

Szenario 4.3	**Es sind auch zusätzliche Felder denkbar, welche sich nicht direkt auf die Data Objects beziehen, sondern die durch vorhandene Werte innerhalb der Object View berechnet werden.**
Beispiel	Es soll nur Rabatt gewährt werden, wenn eine bestimmte Mindestanzahl bestellt wurde. Eine solche Berechnung soll möglicherweise direkt im Abfragen des Properties (also in der get-Methode) durchgeführt werden können.

Programmcode	`public class ItemView {`
	`  …`
	`  double DiscountedPrice {`
	`  get {`
	`  return (Quantity < MinimalDiscountQuantity ?`
	`  Quantity * PricePerItem :`
	`  Quantity * PricePerItem *(100-DiscountPrice)/100)`
	`  }`
	`  }`
	`}`

Diagramm	

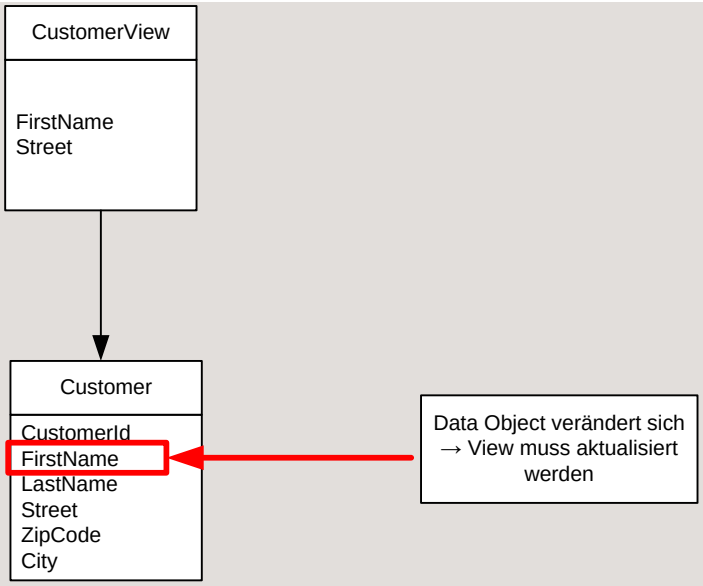
3.1.6 Aggregierte Werte

Szenario 5.1	Es soll möglich sein z.B. Summen oder andere Aggregatsfunktionen zu nutzen.
Beispiel	Es soll zu jeder Bestellung gleich der totale Preis angezeigt werden. Dafür muss eine Summe über alle bestellten Produkte gebildet werden und dabei die Anzahl mit dem Preis pro Produkt multipliziert werden.
Query	`from order in db.Orders join orderItem in db.OrderItems on order.OrderId equals orderItem.Order.OrderId into tempOrderItems from tempOrderItem in tempOrderItems.DefaultIfEmpty() group tempOrderItem by new {order.OrderId} into orderItemGroup select new OrderView {OrderId = orderItemGroup.Key.OrderId, TotalAmount = orderItemGroup.Sum(orderItem => orderItem != null ? orderItem.Quantity * orderItem.Item.PriceCHF : 0.0d)};`
Diagramm	

3.1.7 Aktualisierung

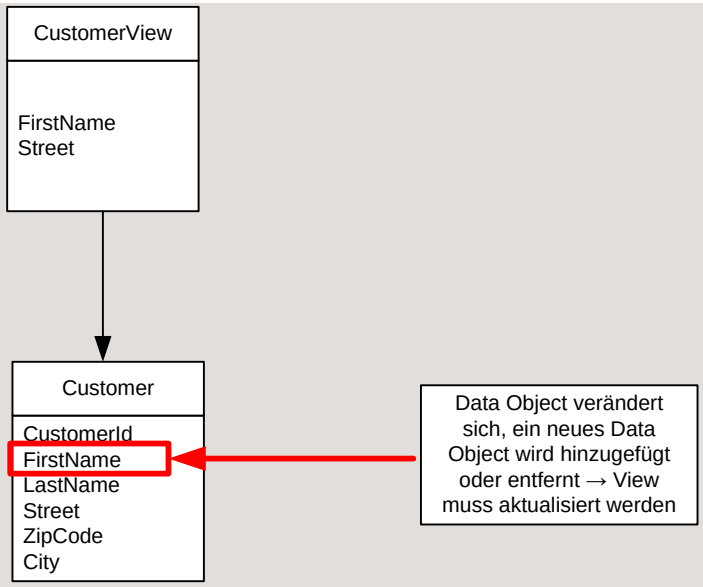
Szenario 6.1	**Wenn sich die Daten der Data Objects verändern, dann müssen sich die davon betroffenen Object Views aktualisieren. Änderungen können direkt auf dem Data Object durchgeführt werden. Ein Aktualisieren des Data Objects verursacht ein Aktualisieren der Object View.**
Beispiel	Wird in der Detailansicht (CustomerOrderView) der Vornamen eines Kunden verändert, so muss die Masteransicht (CustomerView) davon in Kenntnis gesetzt werden, dass der Vorname nun anders ist.

Query	`from customer in db.Customers`
	`select new CustomerView {FirstName = customer.FirstName,`
	`Street = customer.Street}`

Programmcode	`customerObject.FirstName = "Hans";`
Diagramm	

Szenario 6.2	**Wenn sich ein Data Object verändert, ein neues Data Object hinzugefügt oder entfernt wird, dann müssen sich Selektionen, die diese Änderung tangiert, aktualisieren.**
Beispiel	In der View werde alle Kunden mit Vornamen „Hans“ angezeigt. Wird ein neuer Kunde mit Vornamen „Hans“ hinzugefügt, dann muss dieser auch der View hinzugefügt werden.

Query	`from customer in db.Customers`
	`where customer.FirstName.Equals("Hans")`
	`select new CustomerView {FirstName = customer.FirstName,`
	`Street = customer.Street}`

Programmcode	`db.Customers.Add(new Customer {FirstName = "Hans", …};`
Diagramm	

3.2 Updateable Object Views

3.2.1 Übersicht

Mit Updateable Object Views sind Views in einer Single oder Two Tier Umgebung gemeint, deren Datenänderungen persistent gespeichert werden können. D.h. die Object View bietet eine Methode an, mit der die in den Views getätigten Änderungen in der zugrundeliegenden Datenquelle gespeichert werden können.

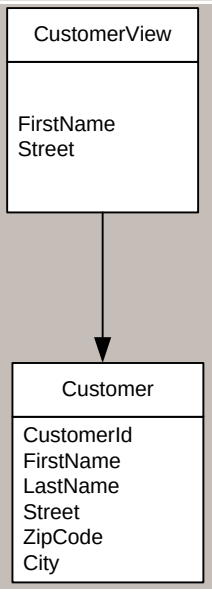
Für diese Art von Object Views sind folgende zusätzliche Szenarien möglich:

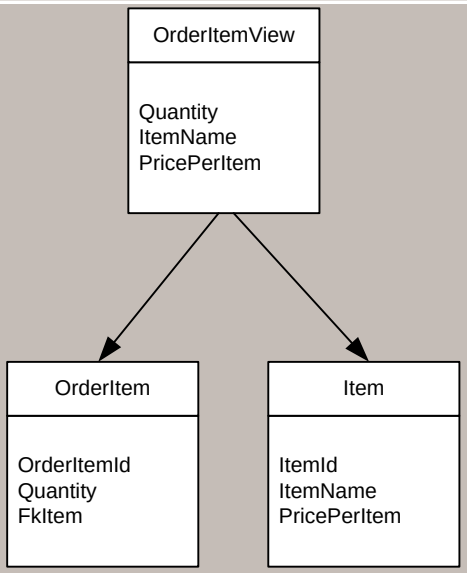
- Einfügen: Einfügen neuer Daten.
- Bearbeiten: Bearbeiten bereits bestehender Daten.
- Löschen: Löschen bestehender Daten.
- Undo und Redo: Änderungen rückgängig machen und wiederherstellen.

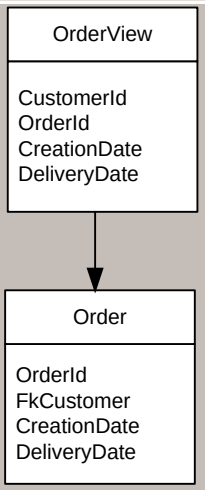
Alle Szenarien müssen für folgende Datenquellen möglich sein:

- Lokale Listen (Collections)
- Entity Framework Data Context

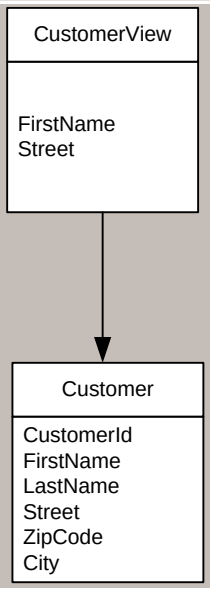
3.2.2 Einfügen

Szenario 7.1	Einfügen in eine Object View mit Bezug auf ein einzelnes Data Object.
Beispiel	In der Object View CustomerView wird eine neue Zeile eingefügt, womit in der Datenquelle ein neuer Kunde erfasst werden soll.
Programmcode	`objectView.Items.Add(…);` `objectView.CommitChanges();`
Diagramm	

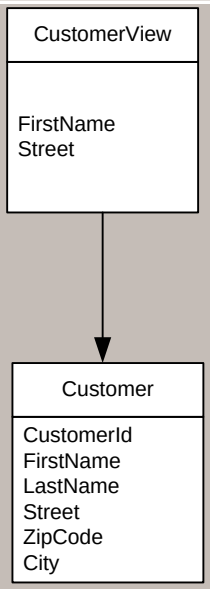
Szenario 7.2	Einfügen in eine Object View mit Bezug auf mehrere Data Objects.
Beispiel	In der Object View OrderItemView wird eine neue Zeile eingefügt, womit in der Datenquelle ein neues Order Item *und* ein neues Item erfasst werden sollen.
Programmcode	`objectView.Items.Add (…);` `objectView.CommitChanges();`
Diagramm	

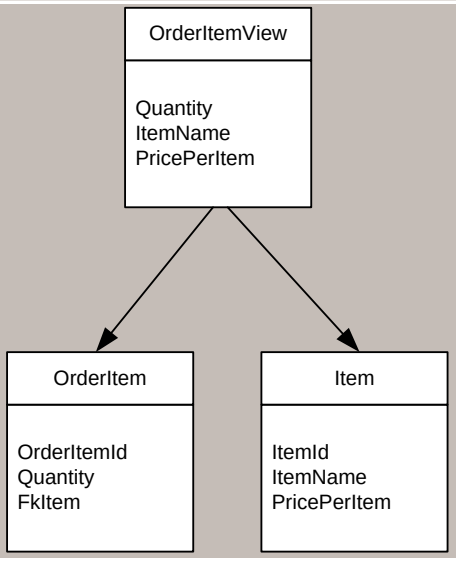
Szenario 7.3	Einfügen in eine Object View mit Bezug auf abhängige Data Objects.
Beispiel	In der Object View OrderView wird eine neue Zeile eingefügt, womit in der Datenquelle eine neue Bestellung erfasst werden soll, die dem via CustomerId selektierten Kunden zugewiesen werden soll.
Programmcode	`objectView.Items.Add (…);` `objectView.CommitChanges();`
Diagramm	

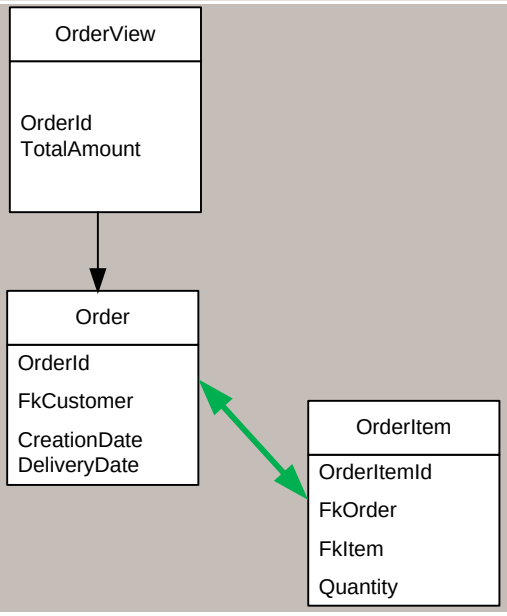
3.2.3 Bearbeiten

Szenario 8.1	Bearbeiten eines schreibbaren Feldes in einer Object View.
Beispiel	In der ObjectView CustomerView wird das Feld FirstName bearbeitet. Diese Änderung wird auf das Feld FirstName des entsprechenden Data Objects in der Datenquelle propagiert.
Programmcode	`objectView.Items[0].FirstName = …;` `objectView.CommitChanges();`
Diagramm	

3.2.4 Löschen

Szenario 9.1	Löschen in einer Object View mit Bezug auf ein einzelnes Data Object.
Beispiel	In der Object View CustomerView wird eine Zeile gelöscht, womit in der Datenquelle der entsprechende Kunde gelöscht wird.
Programmcode	`objectView.Items.Delete(…);` `objectView.CommitChanges();`
Diagramm	

Szenario 9.2	Löschen in einer Object View mit Bezug auf mehrere Data Objects.
Beispiel	In der Object View OrderItemView wird eine Zeile gelöscht. *Gemäss Businesslogik* soll in diesem Fall nun nur das Data Object OrderItem gelöscht werden. Das Data Object Item soll bestehen bleiben.
Programmcode	`objectView.Items.Delete(…);` `objectView.CommitChanges();`
Diagramm	

Szenario 9.3	Löschen in einer Object View mit Bezug auf ein oder mehrere Data Objects mit weiteren abhängigen Data Objects.
Beispiel	In der Object View OrderView wird eine Zeile gelöscht. *Gemäss Businesslogik* soll in diesem Fall nun nebst der Bestellung auch die Order Items gelöscht werden.
Programmcode	`objectView.Items.Delete(…);` `objectView.CommitChanges();`
Diagramm	

3.2.5 Undo und Redo

Szenario 10.1	Rückgängig machen von Änderungen.
Beispiel	In der Object View OrderItemView wird eine Zeile hinzugefügt. Solange die Änderung noch nicht in die Datenquelle propagiert wurde, soll sie vom Benutzer rückgängig gemacht werden können.
Programmcode	`objectView.Items[0].FirstName = …;` `objectView.UndoChanges();`

Szenario 10.2	Wiederherstellen von Änderungen.
Beispiel	Nachdem eine Änderung rückgängig gemacht wurde, soll sie vom Benutzer auch wiederhergestellt werden können, solange die Datenquelle nicht aktualisiert wurde.

Programmcode	`objectView.Items[0].FirstName = …;`
	`objectView.UndoChanges();`
	`objectView.RedoChanges();`

3.3 Three Tier Object Views

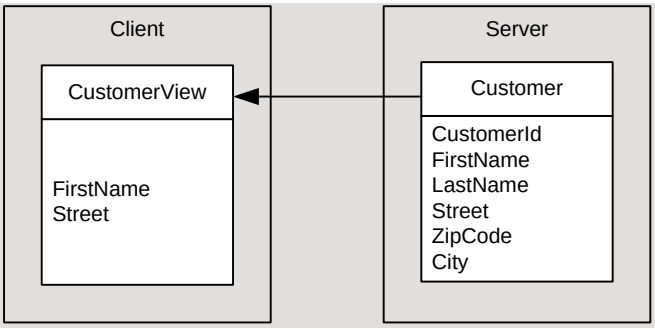
3.3.1 Übersicht

Mit Three Tier Object Views sind Views in einer verteilten (Three Tier) Umgebung gemeint. Die Datenquelle befindet sich auf einem Server, die Clients können Object Views auf diese Datenquelle erstellen und Datenänderungen in den Object Views können persistent gespeichert werden.

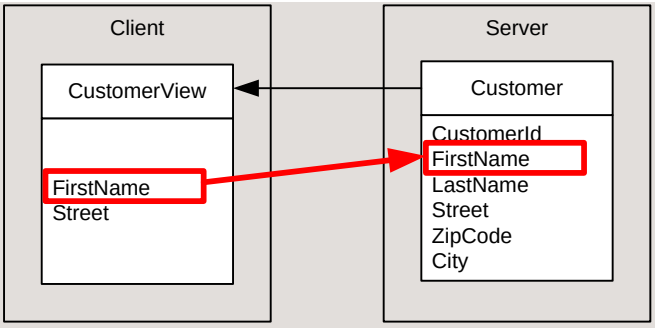
Für diese Art von Object Views sind folgende zusätzliche Szenarien möglich:

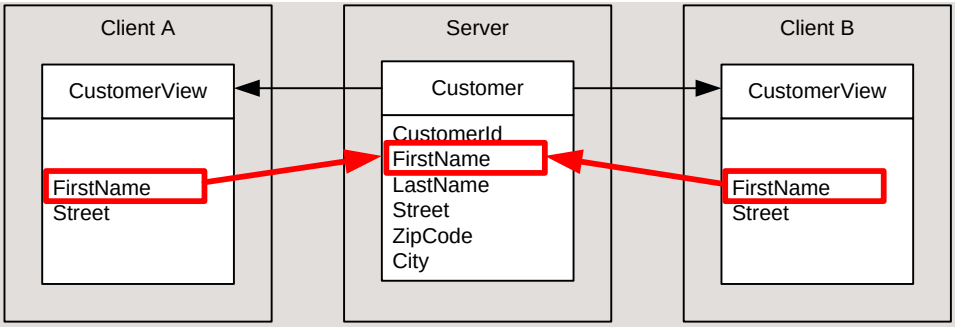
- Lesen: Es können Object Views erstellt werden auf entfernte Datenquellen.
- Schreiben: Es können via Object Views Daten in entfernten Datenquellen geändert werden.
- Aktualisierung: Die Object Views übernehmen die Änderungen, die in der entfernten Datenquelle stattfinden.

3.3.2 Lesen

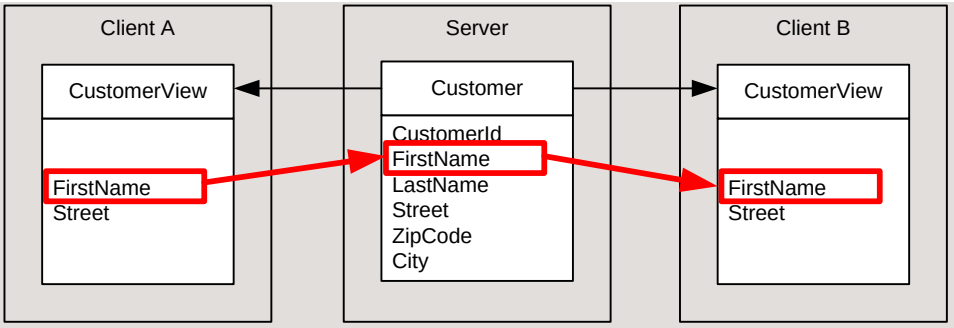
Szenario 11.1	Es können Object Views erstellt werden auf entfernte Datenquellen.
Beispiel	Die Shop-Datenbank befindet sich auf einem Server. Auf dem Client wird eine Object View CustomerView erstellt, der einen Auschnitt aus den Kundendaten in der Shop-Datenbank anzeigt.
Diagramm	

3.3.3 Schreiben

Szenario 12.1	Es können via Object Views Daten in entfernten Datenquellen eingefügt, bearbeitet und gelöscht werden.
Beispiel	Die Shop-Datenbank befindet sich auf einem Server. Auf dem Client wird eine Object View CustomerView erstellt. Nach Änderung des Vornamens eines Kunden in der Object View soll dies auf dem Server persistent gespeichert werden.
Diagramm	

Szenario 12.2	Konfliktsituation beim Schreiben von Daten in die entfernte Datenquelle.
Beispiel	Zwei Clients zeigen in je einer Object View Daten desselben Kunden an. Client A bearbeitet den Vornamen des Kunden und persistiert die Änderung auf dem Server. Auch der Client B bearbeitet den Vornamen des Kunden, wodurch beim Speichern eine Konfliktsituation auftritt.
Diagramm	

3.3.4 Aktualisierung

Szenario 13.1	Sobald in der entfernten Datenquelle eine Änderung stattfindet, übernehmen diese alle Clients automatisch.
Beispiel	Zwei Clients zeigen in je einer Object View Daten desselben Kunden an. Client A bearbeitet den Vornamen des Kunden und persistiert die Änderung auf dem Server. Da Client B denselben Kunden anzeigt, wird seine Anzeige des Kunden aktualisiert.
Diagramm	

4 Anforderungsspezifikation

Dieses Kapitel beschreibt die Anforderungen, die an das Object Views Framework gestellt werden. Sie sind abgeleitet von den Szenarien, die beschreiben, welche Anwendungsfälle abgedeckt werden sollen. Die funktionalen Anforderungen beziehen sich darum jeweils auf diese Szenarien.

4.1 Allgemeine Beschreibung

4.1.1 Produktperspektive

Mit Object Views sollen Views auf Data Object Graphen (DOG) realisiert werden können. Es ist dabei nicht relevant, woher dieser DOG geladen wird, sei dies nun eine Datenbank, oder eine andere Datenquelle.

Die Object Views werden vom Programmierer im Programmcode als Linq-Queries definiert. Das Object Views Framework sorgt dafür, dass immer nur die nötigsten Daten geladen werden, dass die Ansichten bearbeitet werden können und dass bei Datenmanipulationen betroffene Views aktualisiert werden. Datenänderungen sollen auch rückgängig gemacht und wiederhergestellt werden können. Das Framework soll zudem auch in einer verteilten Umgebung eingesetzt werden können.

4.2 Produktfunktionen

4.2.1.1 Readonly Object Views

- Es können Object Views aufgrund verschiedenster Queries (Projektionen, Selektionen, Verbunde, etc.) erstellt werden.
- Die Object Views müssen bearbeitbar sein, die Änderungen werden jedoch nicht in die Datenquelle zurückgeschrieben.
- Es können Object Views auf lokale Datenstrukturen erstellt werden (Single Tier Umgebung, z.B. mit Linq To Objects).
- Es können Object Views auf Datenbanken erstellt werden (Two Tier Umgebung, z.B. mit Linq To Entities).

4.2.1.2 Updateable Object Views

- In den Object Views getätigte Änderungen (Einfügen, Bearbeiten, Löschen) müssen auf Verlangen in die Datenquelle zurückgeschrieben werden.
- Als Datenquellen müssen sowohl lokale Listen, als auch ein Entity Framework Datenkontext unterstützt werden.

4.2.1.3 Three Tier Object Views

- Es können Object Views erstellt werden, die Daten aus entfernten Datenquellen anzeigen.
- Änderungen, die in den Object Views vorgenommen werden, müssen in die entfernte Datenquelle zurückgeschrieben werden können.
- Ändern sich die Daten in der entfernten Datenquelle, aufgrund von Manipulationen anderer Clients, dann werden die betroffenen Object Views aktualisiert.

4.2.2 Benutzermerkmale

Benutzerrolle	Applikationsentwickler
Beschreibung	Der Applikationsentwickler will das Object Views Framework nutzen, um sich die Arbeit im Umgang mit Views zu erleichtern. Die Person ist typischerweise ein Programmierer und im Umgang mit Computern ein sehr erfahrener Nutzer. Er weiss mit einer API umzugehen, möchte sich aber nicht zuerst noch gross in diese einlesen müssen, damit auch effektiv ein Zeitgewinn entsteht.

4.2.3 Einschränkungen

- Das Framework lässt sich nur in der .Net Laufzeitumgebung einsetzen.
- Auf die Datenquelle muss mit Linq Queries zugegriffen werden können.

4.2.4 Annahmen und Abhängigkeiten

- Aus dem Expression Tree bzw. aus den Linq Queries lassen sich zuverlässig alle Metadaten auslesen.
- In verteilten Systemen kann Interlinq zur Übertragung der Queries eingesetzt werden.

4.3 Spezifische Anforderungen

4.3.1 Funktionale Anforderungen

Die folgenden Anforderungen sind von den Szenarien abgeleitet. Für eine detaillierte Beschreibung der Szenarien siehe Kapitel Szenarien.

4.3.1.1 Readonly Object Views

Anforderung 1	Projektion
Beschreibung	Es können Object Views erstellt werden, mit denen nur gewisse Felder eines Datenobjekts angezeigt werden. Es sollen dafür keine anderen Felder des Datenobjekts geladen werden.
Szenarien	Szenario 1.1
Priorität	Hoch

Anforderung 2	Selektion
Beschreibung	Es können Object Views erstellt werden, mit denen eine Auswahl aus einer Liste von Datenobjekten angezeigt wird.
Szenarien	Szenario 2.1
Priorität	Hoch

Anforderung 3	Verbund
Beschreibung	Es können Object Views erstellt werden, die sich aus Datenobjekten verschiedener Klassen zusammensetzen.
Szenarien	Szenario 3.1 Szenario 3.2 Szenario 3.3 Szenario 3.4
Priorität	Hoch

Anforderung 4	Zusammengesetzte Werte
Beschreibung	Es können Object Views erstellt werden, die Felder enthalten, die sich aus mehreren Feldern der Datenobjekte berechnen oder zusammensetzen.
Szenarien	Szenario 4.1 Szenario 4.2 Szenario 4.3
Priorität	Hoch

Anforderung 5	Aggregierte Werte
Beschreibung	Es muss möglich sein, z.B. Summen oder andere Aggregatsfunktionen zu nutzen.
Szenarien	Szenario 5.1
Priorität	Hoch

Anforderung 6	Aktualisierung
Beschreibung	Wenn sich die Daten verändern, dann müssen sich *die davon betroffenen* Object Views aktualisieren. Die Object Views müssen sich ebenfalls aktualisieren, wenn neue Daten hinzugefügt oder entfernt werden.
Szenarien	Szenario 6.1 Szenario 6.2
Priorität	Hoch

Anforderung 7	Typisierte Object Views
Beschreibung	Es können Object Views erstellt werden, auf deren Daten typensicher via Properties zugegriffen werden kann.
Szenarien	Muss für alle Szenarien möglich sein.
Priorität	Hoch

Anforderung 8	Untypisierte Object Views
Beschreibung	Es können Object Views erstellt werden, die anonyme Objekte anzeigen.
Szenarien	Muss für alle Szenarien möglich sein.
Priorität	-
Bemerkung	Bei der Sitzung zu Meilenstein 2 wurde entschieden, auf untypisierte Object Views zu verzichten.

Anforderung 9	Single Tier Object Views
Beschreibung	Es können Object Views auf interne Datenstrukturen (z.B. Listen) erstellt werden.
Szenarien	Muss für alle Szenarien möglich sein.
Priorität	Hoch

Anforderung 10	Two Tier Object Views
Beschreibung	Es können Object Views auf Datenbanken, auf die direkten Zugriff besteht, erstellt werden. Dabei ist es nur möglich, Daten anzuzeigen, ein Persistieren von Datenänderungen ist nicht möglich.
Szenarien	Muss für alle Szenarien möglich sein.
Priorität	Hoch

4.3.1.2 Updateable Object Views

Anforderung 11	Einfügen
Beschreibung	Es können Daten in Object Views eingefügt werden, und diese Änderungen können auf Verlangen in die Datenquelle zurückgeschrieben werden.
Szenarien	Szenario 7.1 Szenario 7.2 Szenario 7.3
Priorität	Hoch

Anforderung 12	Bearbeiten
Beschreibung	Es können Daten in Object Views bearbeitet werden, und diese Änderungen können auf Verlangen in die Datenquelle zurückgeschrieben werden.
Szenarien	Szenario 8.1
Priorität	Hoch

Anforderung 13	Löschen
Beschreibung	Es können Daten in Object Views gelöscht werden, und diese Änderungen können auf Verlangen in die Datenquelle zurückgeschrieben werden.
Szenarien	Szenario 9.1 Szenario 9.2 Szenario 9.3
Priorität	Hoch

Anforderung 14	Undo und Redo
Beschreibung	Es können vorgenommene Änderungen rückgängig gemacht und wiederhergestellt werden, solange die Datenquelle nicht aktualisiert wurde.
Szenarien	Szenario 10.1 Szenario 10.2
Priorität	Mittel

Anforderung 15	Updateable Collections
Beschreibung	Es können lokale Listen als Datenquelle für Updateable Object Views eingesetzt werden.
Szenarien	Muss für alle Szenarien möglich sein.
Priorität	Mittel

Anforderung 16	Updateable Entity Framework Data Context
Beschreibung	Es kann ein Entity Framework Datenkontext als Datenquelle für Updateable Object Views eingesetzt werden.
Szenarien	Muss für alle Szenarien möglich sein.
Priorität	Hoch

4.3.1.3 Three Tier Object Views

Anforderung 17	Lesen von Remote Data Context
Beschreibung	Es können Object Views erstellt werden auf entfernte Datenquellen.
Szenarien	Szenario 11.1
Priorität	Mittel

Anforderung 18	Schreiben in Remote Data Context (Einfügen, Bearbeiten, Löschen)
Beschreibung	Es können via Object Views Daten in entfernten Datenquellen eingefügt, bearbeitet und gelöscht werden.
Szenarien	Szenario 12.1 Szenario 12.2
Priorität	Mittel

Anforderung 19	Übernehmen von Änderungen aus Remote Data Context
Beschreibung	Die Object Views übernehmen die Änderungen, die in der entfernten Datenquelle stattfinden.
Szenarien	Szenario 13.1
Priorität	Niedrig

4.3.2 Nichtfunktionale Anforderungen

4.3.2.1 Bedienbarkeit

Anforderung 20	Einfache Verwendung des Frameworks
Beschreibung	Die Verwendung des Frameworks muss intuitiv sein und darf kein aufwändiges Studium der Schnittstellen voraussetzen.
Priorität	Hoch

Anforderung 21	Einfache Einbindung des Frameworks
Beschreibung	Das Framework muss ohne Konfigurieraufwand eingesetzt werden können.
Priorität	Niedrig

4.3.2.2 Zuverlässigkeit

Anforderung 22	Hundertprozentige Verfügbarkeit
Beschreibung	Solange die Datenquelle verfügbar ist, muss das Framework ebenfalls verfügbar sein und die angeforderten Views liefern.
Priorität	Mittel

Anforderung 23	Absturzfreier Ablauf
Beschreibung	Exceptions, deren Ursache nicht beim Framework liegt, müssen von diesem nicht behandelt werden, sondern dürfen der ausführenden Applikation überlassen werden (z.B. wenn die Datenquelle nicht verfügbar ist). Exceptions, die innerhalb des Frameworks geworfen werden, dürfen nicht zum Absturz der aufrufenden Applikation führen.
Priorität	Hoch

4.3.2.3 Leistung

Anforderung 24	Minimale Datenübertragung
Beschreibung	Das Framework darf immer nur das nötigste an Daten zum Client übertragen.
Priorität	Hoch

Anforderung 25	Minimale Aktualisierung
Beschreibung	Beim Aktualisieren von Object Views muss darauf geachtet werden, dass nur jene Views aktualisiert werden, bei denen eine Aktualisierung auch nötig ist.
Priorität	Mittel

Anforderung 26	Minimaler Performanzverlust
Beschreibung	Durch den Einsatz von Object Views darf kein markanter Performanzverlust entstehen. Eine Abfrage darf maximal 10% länger dauern, als wenn sie ohne Object Views Framework ausgeführt würde.
Priorität	Hoch

Anforderung 27	Meistern von mindestens 100‘000 Datensätzen
Beschreibung	Das Framework muss mit mindestens 100‘000 Datensätzen umgehen können.
Priorität	Mittel

Anforderung 28	Meistern einer beliebigen Anzahl Abfragen
Beschreibung	Das Framework muss mit einer beliebig grossen Anzahl an Abfragen klar kommen. Es muss dafür sorgen, dass immer nur die noch benötigten Object Views im Speicher gehalten werden.
Priorität	Mittel

4.3.2.4 Wartbarkeit

Anforderung 29	Gute Wartbarkeit
Beschreibung	Das Framework muss so dokumentiert werden, dass es auch ein bei der Entwicklung nicht involvierter Programmierer warten kann.
Priorität	Mittel

4.3.2.5 Schnittstellen

Anforderung 30	Keine Anforderungen an Datenobjekte
Beschreibung	Es dürfen keine Anforderungen an die Klassen der Datenobjekte gestellt werden (z.B. dass diese von einer bestimmten Klasse abgeleitet sein müssen), *falls dies technisch möglich ist*.
Priorität	Mittel

Anforderung 31	Unterstützung aller Linq kompatiblen Datenquellen
Beschreibung	Das Framework muss auf alle Datenquellen angewandt werden können, die Linq Abfragen unterstützen und deren Datenstrukturen das Interface IQueryable implementieren.
Priorität	Mittel

5 Konzept

Dieses Kapitel beschreibt die Lösungsansätze, wie die an das Framework gestellten Anforderungen gelöst werden. Es werden vorerst lediglich die Grundkonzepte, die Grundideen, beschrieben. Eine detaillierte Dokumentation des resultierten Produkts findet sich im Kapitel Softwarearchitektur.

5.1 Readonly Object Views

5.1.1 Grundkonzept

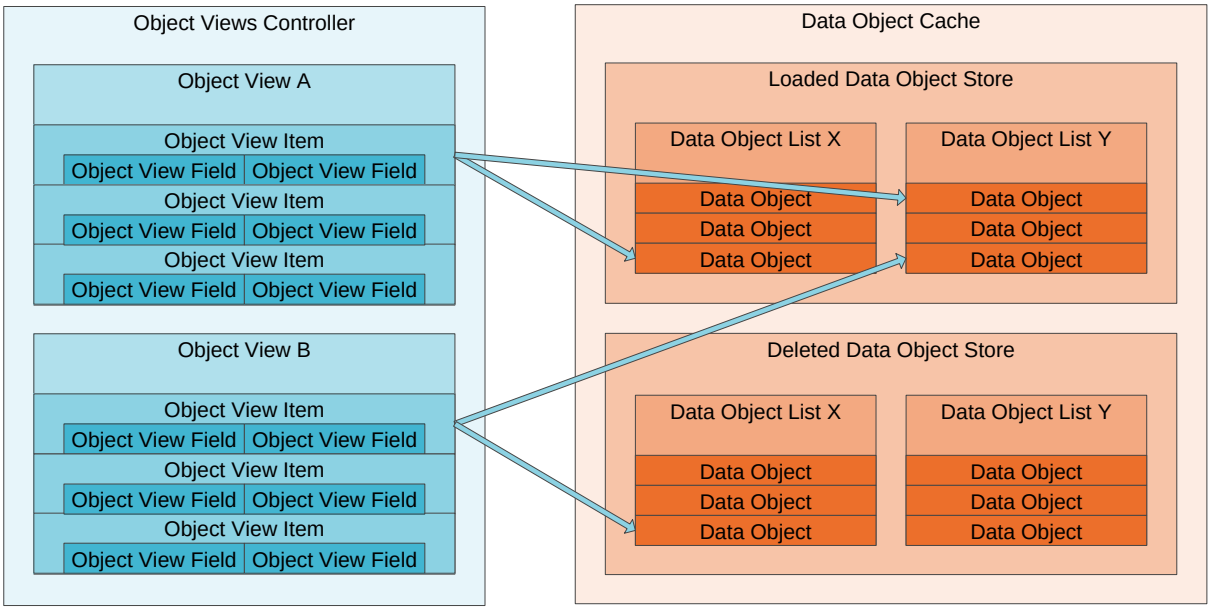


Abbildung 8 - Konzept für Readonly Object Views

5.1.1.1 Object Views Controller

Für die Erstellung einer Object View auf einen Data Object Graphen benötigt es eine zentrale Instanz, die Queries entgegennimmt und daraus ein View Objekt erstellt. Diese Instanz nennen wir den *Object Views Controller*. Dieser Controller ist die Schnittstelle für den Verwender des Frameworks.

5.1.1.2 Object View

Eine *Object View* wird mittels dem Object Views Controller erstellt, indem dem Controller eine Query gesendet wird. Diese Object View wird dann dazu verwendet, die durch die Query angeforderten Daten z.B. im User Interface anzuzeigen.

Die Daten sind in der Object View als Liste enthalten, da Select Statements Listen zurückgeben. Die einzelnen Positionen in diesen Listen werden *Object View Items* genannt. Die einzelnen Felder innerhalb der Object View Items werden durch sogenannte *Object View Fields* repräsentiert.

5.1.1.3 Data Object Cache

Sobald eine Query abgesetzt wird, dann speichert der Object View Controller alle von der Datenquelle abgeholten Daten im sogenannten *Data Object Cache*. Der Data Object Cache enthält also Data Objects, die ganz oder teilweise mit Werten abgefüllt sind.

Die Object View Items in den Object Views merken sich jeweils, auf welche Data Objects im Data Object Cache sie sich beziehen. Wenn nun Daten in einer Object View geändert werden, dann aktualisiert das Framework die Data Objects, auf die sich das betroffene Object View Item bezieht.

Damit die Data Objects vom Data Object Cache identifiziert werden können, muss für jedes Data Object, das geladen wird, eine eindeutige Id vorhanden sein. Dies bedingt, dass die Id jedes einzelnen Data Objects in der Query vorhanden sein muss.

5.1.1.4 Metamodell

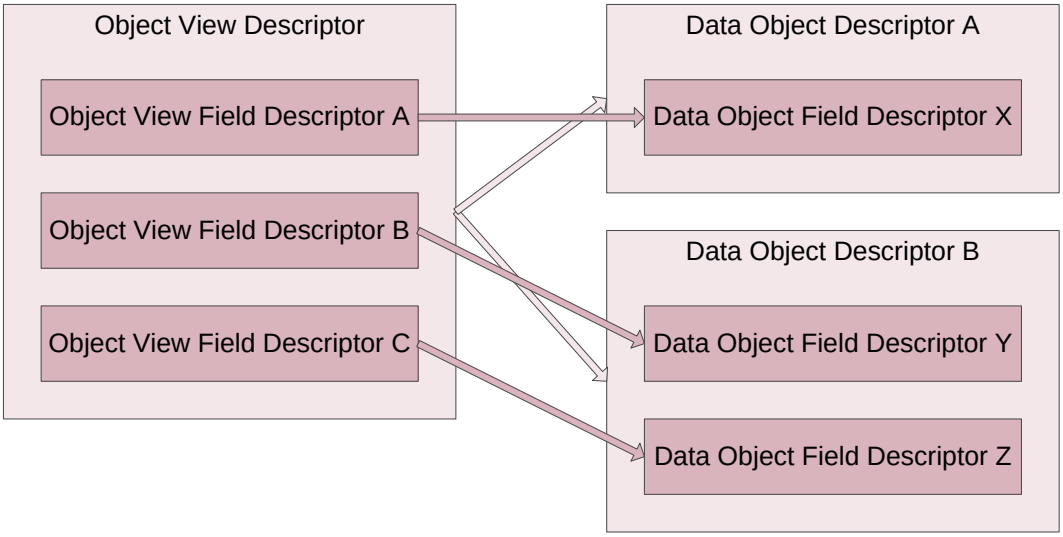


Abbildung 9 - Konzept für Metamodell

Für jede Object View, die erstellt wird, wird ein *Metamodell* angelegt. Das Metamodell beschreibt, wie die Object View aussieht, d.h. aus welchen Feldern sie besteht und auf was für Data Objects und auf welche Felder der Data Objects sie sich bezieht.

Das Metamodell wird angelegt, um auf Datenänderungen in den Object Views und in den Data Objects reagieren zu können. Ändert sich z.B. ein Wert in einem Data Object, dann wissen die Object Views dank dem Metamodell, ob sie davon betroffen sind und welche Felder im Object View aktualisiert werden müssen.

Das Metamodell wird aufgrund des Expression Trees der abgesetzten Query erstellt. Die Query wird dazu mittels dem Visitor Pattern geparst.

5.1.2 Konzepte für Anforderungen

5.1.2.1 Projektion

Konzept 1.1	Projektion
Problem	Bei einer Projektion wird nur einen Teil der Data Objects geladen. Dies muss beim Speichern der geladenen Data Objects im Data Object Cache berücksichtigt werden.
Beispiel	`from c in db.Customer select new CustomerView {CustomerId = c.CustomerId, FirstName = c.FirstName, LastName = c.LastName};`
Lösung	Für jedes im Cache abgelegte Data Object merkt sich das Framework, welche Felder des Data Objects bereits geladen wurden. Wenn nun eine neue Object View erstellt wird, die Data Objects enthält, die bereits im Data Object Cache vorhanden sind, dann werden diese um die neu dazu kommenden Daten erweitert. Mit obiger Query wird im Data Object Cache eine gewisse Anzahl von Customer Objekte angelegt, deren Attribute CustomerId, FirstName und LastName jeweils abgefüllt sind.

5.1.2.2 Selektion

Object Views mit einer Selektion müssen erst dann speziell behandelt werden, wenn im Data Cache Daten hinzugefügt, geändert oder entfernt werden, die diese Selektion betreffen. Diese Problematik wird im Kapitel Aktualisierung, im Konzept 6.3 beschrieben.

5.1.2.3 Verbund

Konzept 3.1	Verbund
Problem	Queries mit Joins liefern im Normalfall mehrere Data Objects verschiedener Typen, die in einer Beziehung zueinander stehen. Um die Data Objects im Cache ablegen zu können, müssen diese Typen ermittelt werden.

Beispiel	`from orderItem in db.OrderItems`
	`join item in db.Items on item.ItemId equals orderItem.FkItem`
	`select …`
	
	Der Verbund kann auch über ein Property des Persistenzframeworks realisiert werden (z.B. beim Entity Framework über das Property *Item* der Klasse *OrderItem*):
	`from orderItem in db.OrderItems`
	`select new { orderItem.Item.ItemName, …}`

Lösung	Die Typen der Data Objects werden mittels dem Visitor Pattern aus dem Query ermittelt und im Metamodell gespeichert. Die Beziehungen der Data Objects werden in den Deskriptoren abgebildet und in den Data Objects direkt hergestellt.

Konzept 3.2	Verbund von Data Objects des gleichen Typs
Problem	Bei Verbunden speziell zu berücksichtigen ist der Fall, dass zwei oder mehrere Data Objects *des gleichen Typs* miteinander gejoint werden. Das heisst ein Object View Item verweist auf zwei oder mehrere Data Objects des gleichen Typs und muss diese dennoch unterscheiden können.
Beispiel	`from customer1 in db.Customers join customer2 in db.Customers on …`
Lösung	Ein Object View Item merkt sich nebst der Referenz zu einem Data Object auch den *Namen*, mit dem das Data Object in der Query identifiziert wird. Bei obigem Beispiel werden im Data Object Cache zwei Customer Data Objects angelegt, wobei sich das Object View Item die Verknüpfung zu diesen Objekten unter den in der Query angegebenen Namen „customer1“ und „customer2“ merkt.

5.1.2.4 Zusammengesetzte Werte

Konzept 4.1	Zusammengesetzte Werte
Problem	Zusammengesetzte oder berechnete Werte sind readonly, da die Zusammensetzung oder Berechnung im Normalfall nicht umkehrbar ist. Es muss also dafür gesorgt werden, dass diese Werte vom Benutzer nicht bearbeitet werden können.
Beispiel	`from customer in db.Customers select new {Name = customer.FirstName + „ „ + customer.LastName};`
Lösung	Der Verwender des Frameworks muss im GUI dafür sorgen, dass zusammengesetzte Felder nicht bearbeitet werden können. Falls er dies nicht tut, so werden Änderungen in zusammengesetzten Feldern vom Object Views Framework ignoriert.

5.1.2.5 Aggregierte Werte

Konzept 5.1	Aggregierte Werte
Problem	Aggregierte Werte sind wie zusammengesetzte Werte readonly. Es muss also dafür gesorgt werden, dass diese Werte vom Benutzer nicht bearbeitet werden können.
Beispiel	`from order in db.Orders` `select new OrderView {OrderId = order.OrderId, TotalAmount = order.OrderItem.Sum(orderItem => (double)(orderItem.Quantity*orderItem.Item.PriceCHF))}`
Lösung	Der Verwender des Frameworks muss im GUI dafür sorgen, dass aggregierte Felder nicht bearbeitet werden können. Falls er dies nicht tut, so werden Änderungen in aggregierten Feldern vom Object Views Framework ignoriert.

5.1.2.6 Aktualisierung

Konzept 6.1	Aktualisierung nach Änderung vorhandener Daten
Problem	Wenn Daten in einer Object View geändert werden, dann müssen andere Object Views, die dieselben Daten anzeigen, aktualisiert werden.
Beispiel	Eine Object View A listet alle Kunden auf. Mit einer zweiten Object View B kann ein einzelner Kunde bearbeitet werden. Bei einer solchen Bearbeitung eines Kunden mittels der Object View B muss die Object View A aktualisiert werden.
Lösung	Zu diesem Zweck wird für jedes Feld in der Object View ein Event Handler registriert. Ändern sich nun die Daten in einem Feld, dann wird dieser Event Handler aufgerufen, welcher die mit der Object View verbundenen Data Objects anpasst. Zudem registriert jedes Object View Feld für jedes Data Object Feld, von dem es abhängt, einen Event Handler. Wenn sich nun der Wert im Data Objects ändert, dann werden alle Event Handler aufgerufen, die den Event abonniert haben. Dadurch können sich alle betroffenen Object Views aktualisieren.

Konzept 6.2	Aktualisierung nach Einfügen oder Löschen
Problem	Wenn einer Object View eine Zeile hinzugefügt oder entfernt wird, dann müssen im Data Object Cache Data Objects hinzugefügt oder entfernt werden. Zudem müssen sich dann jene Object Views aktualisieren, die von dieser Änderung im Data Object Cache betroffen sind.
Beispiel	Eine Object View A zeigt alle Kunden und die Anzahl Bestellungen pro Kunde. Object View B listet alle Bestellungen eines Kunden auf. Wenn in der Object View B eine Bestellung hinzugefügt wird, dann muss sich die Object View A aktualisieren.
Lösung	Jedes Hinzufügen und jedes Löschen erzeugt einen Create-Command bzw. einen Delete-Command. Eine Object View bezieht die Daten immer von einem oder mehreren Data Objects. Auf den Create- und Delete-Commands kann sich nun eine Object View mit einem Event Handler registrieren, damit sie informiert wird, wenn ein Data Object eines bestimmten Typs (z.B. Customer) hinzukommt. Die Object View macht bei einem solchen Event einen Refresh. Solche Aktualisierungen laufen immer nur lokal ab, d.h. es erfolgt kein Zugriff auf die Datenquelle.

Konzept 6.3	Aktualisierung von Selektionen
Problem	Queries mit Selektionen schränken die Menge der im Object View enthaltenen Daten durch Bedingungen ein. Wenn sich nun Daten im Data Object Cache verändern, die diese Bedingungen erfüllen, dann muss die Object View aktualisiert werden.

Beispiel	`from customer in db.Customers`
	`where customer.City.Equals("Zürich")`
	`select …`

Lösung	Für jede Veränderung auf einem Data Object wird ein Update-Command erstellt. Auf diesem Command kann sich eine Object View wiederum registrieren. Eine Registration erfolgt immer auf: • Typ des Data Object (z.B. Customer) • Name des Feldes (z.B. City) • Event Handler über welchen das Update erfolgen soll (z.B. Event Handler, welcher Refresh ausführt) Dadurch wird der Event nur ausgelöst, wenn sich wirklich das Feld eines bestimmten Data Objects verändert hat. Die Object View muss dann wiederum einen Refresh ausführen, indem das Query erneut auf den lokalen Cache abgesetzt wird.

Konzept 6.4	Aktualisierung zusammengesetzter Werte
Problem	Bei zusammengesetzten oder berechneten Werten ergibt sich die Situation, dass die Object View einen Wert enthält, der aus Werten der Data Objects berechnet oder zusammengesetzt wurde, diese Werte in den Data Objects jedoch nicht zwingend gespeichert sind.

	Dies ist so, weil das Query nur die berechneten Werte (im folgenden Beispiel DiscountedPrice) zurückliefert und nicht die Werte, aus denen sich diese berechneten Werte berechnen. Da zusammengesetzte Werte readonly sind, stellt dies soweit kein Problem dar. Wenn nun aber eine zweite Object View einen dieser Ursprungs-Werte lädt und sich dieser ändert, dann muss der berechnete Wert in der ersten Object View aktualisiert werden. Diese Aktualisierung des berechneten Wertes stellt jedoch dann ein Problem dar, wenn nicht alle Werte, aus denen er sich zusammensetzt, im Data Object Cache vorhanden sind. Im folgenden Beispiel wird im Data Object Cache zwar ein Item Objekt angelegt, die Felder PricePerItem und DiscountPercent der Objekte sind jedoch *nicht* abgefüllt.
Beispiel	`from item in db.Items select new ItemView {ItemName = item.ItemName, DiscountedPrice = item.PricePerItem * (100 – item.DiscountPercent) / 100}`
Lösung	Bei den zusammengesetzten Feldern wird zur Aktualisierung die Berechnung nochmals durchgeführt. Um das Feld „*DiscountedPrice*“ neu zu berechnen wird folgendes Statement nochmals abgesetzt: `item.PricePerItem * (100 – item.DiscountPercent) / 100}` Dazu müssen sämtliche Felder in den Datenobjekten gesetzt sein, damit diese Berechnung lokal ausgeführt werden kann. In unserem Fall müssen zwei Felder gesetzt, d.h. mit Werten abgefüllt sein: • PricePerItem im Datenobjekt Item • DiscountPercent im Datenobjekt Item Beim Erstellen einer Object View, die zusammengesetzte Werte enthält, wird sichergestellt, dass alle benötigten Daten geladen werden. Diese Überprüfung wird von einem sog. *Query Validator* vorgenommen. Dieser überprüft die Linq Query und stellt sicher, dass alle zur Berechnung benötigten Werte in der selben Query mitgeladen werden.

Konzept 6.5	**Aktualisierung aggregierter Werte**
Problem	Bei aggregierten Werten stellt sich ein sehr ähnliches Problem, wie bei den zusammengesetzten Werten. Wenn sich ein Wert ändert, der in einem aggregierten Wert enthalten ist, oder ein Wert hinzukommt oder entfernt wird, dann muss der aggregierte Wert aktualisiert werden.
Beispiel	`from customer in dataContext.Customers` `select new CustomerView {CustomerId = customer.CustomerId, OrderCount = customer.Order.Count}`

Lösung	Damit die Aktualisierung wie bei den anderen Object View Felder durchgeführt werden kann, wird das Feld jeweils einzeln aktualisiert. Dazu muss das Query modifiziert werden:
	
	`from customer in localContext.Customers`
	`where customer.CustomerId.Equals(2)`
	`select new CustomerView {OrderCount = customer.Order.Count}`
	
	Das Where Statement wird der vorhandenen Expression hinzugefügt und es wird nur der nötige Wert (OrderCount) abgeholt. Durch das Where Statement wird sichergestellt, dass nur der Wert für den bestimmten Kunden abgeholt wird. Dieser Wert wird aus dem bereits abgefüllten Object View Item ausgelesen und so gesetzt.
	
	Das Framework setzt voraus, dass die benötigten Daten (hier alle Orders) im lokalen Kontext geladen wurden. Dies muss vom Verwender des Frameworks durch ein zusätzliches Query gewährleistet werden:
	`from customer in _dataContext.Customers`
	`join order in _dataContext.Orders on customer.CustomerId equals order.Customer.CustomerId`
	`select new OrderView {OrderId = order.OrderId, CustomerId = customer.CustomerId}`

5.1.2.7 Typisierte Object Views

Konzept 7.1	Typisierte Object Views
Problem	Bei typisierten Object Views soll der Verwender des Frameworks mittels Properties und typensicher auf die Werte in der View zugreifen können.
Beispiel	`var customerFirstName = customerObjectView.ObjectViewItems[0].FirstName;`

Lösung	Für eine typisierte Object View muss der Verwender des Frameworks eine View-Klasse schreiben, die beschreibt, wie die Object View auszusehen hat.
	
	Beispiel einer solchen View-Klasse:
	`public class CustomerView:ObjectViewItem { public int CustomerId { get; set; } public string FirstName { get; set; } public string LastName { get; set; } }`
	
	Beispiel einer Query für eine typisierte Object View:
	`from c in db.Customer select new CustomerView {CustomerId = c.CustomerId, FirstName = c.FirstName, LastName = c.LastName};`

5.1.2.8 Untypisierte Object Views

Konzept 8.1	Untypisierte Object Views
Problem	Es können auch nicht typisierte Object Views erstellt werden, für die keine eigene View-Klasse zur Verfügung gestellt werden muss. Von einer Query zurückgegeben werden anonyme Objekte, die *nicht bearbeitbar* sind. Das bedeutet, dass die untypisierten Object Views nicht auf dieselbe Art und Weise wie die typisierten Object Views behandelt werden können.
Beispiel	`from c in db.Customer select new {CustomerId = c.CustomerId, FirstName = c.FirstName, LastName = c.LastName};`
Lösung	-
Bemerkung	Bei der Sitzung zu Meilenstein 2 wurde entschieden, auf untypisierte Object Views zu verzichten.

5.1.2.9 Single Tier und Two Tier Umgebung

Da bei Readonly Object Views nur Daten aus den Datenquellen geladen und *nicht zurückgeschrieben* werden, ergibt sich beim Konzept keinen Unterschied zwischen einer Single Tier und Two Tier Umgebung. Von *wo* die Daten geladen werden, wird vom Verwender des Frameworks in den Queries definiert. Für das Framework macht es keinen Unterschied, ob dies nun einfache Collections oder Daten im Data Context eines Persistenz-Frameworks sind.

5.2 Updateable Object Views

5.2.1 Grundkonzept

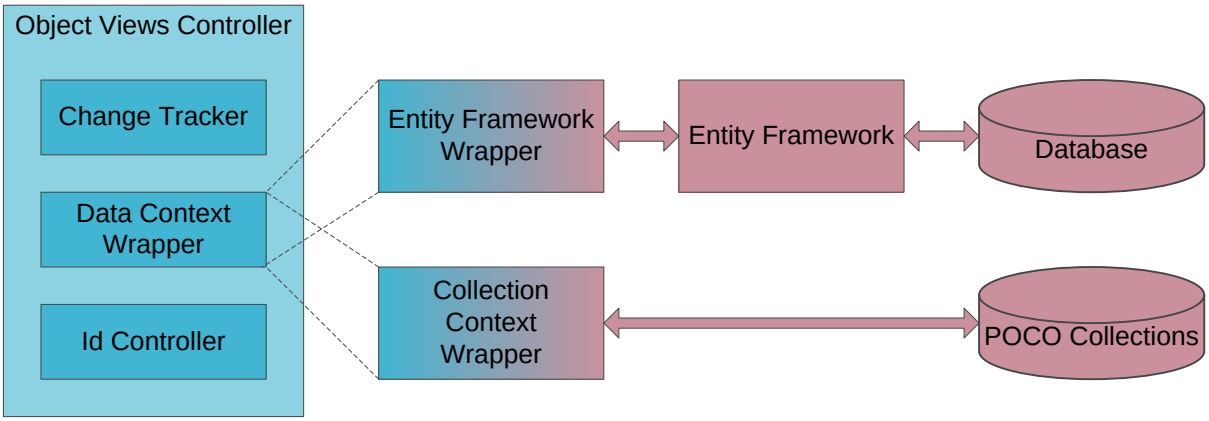


Abbildung 10 - Konzept für Updateable Object Views

5.2.1.1 Change Tracker

Der Object Views Controller enthält einen *Change Tracker*, der sich jegliche Änderungen merkt, die an Data Objects vorgenommen werden. Mögliche Änderungen sind:

- Einfügen eines neuen Data Objects
- Bearbeiten eines einzelnen Feldes eines Data Objects
- Löschen eines Data Objects

Der Change Tracker merkt sich die Änderungen mittels Commands, d.h. für jede Änderung wird ein Command-Objekt erstellt und aufgelistet. Grundsätzlich werden die Commands erstellt, wie sie der Kunde erfasst. Werden jedoch in einer View mehrere Data Objects gleichzeitig erstellt, wird aufgrund der Beziehungen zwischen den Data Objects ermittelt, in welcher Reihenfolge die Data Objects erstellt werden müssen. Kann beispielsweise eine Bestellposition und ein Artikel gleichzeitig eingefügt werden, so wird zuerst der Artikel eingefügt und erst anschliessend die Bestellposition (ansonsten wären Fremdschlüsselbeziehungen verletzt).

5.2.1.2 Data Context Wrapper

Um die vorgenommenen Änderungen persistent machen zu können, treten sog. *Data Context Wrapper* in Kraft. Für jeden Typ von Data Context (z.B. ein Entity Framework Data Context oder ein Linq To Sql Data Context) muss ein solcher Data Context Wrapper geschrieben werden. Das Object Views Framework stellt bereits einen Data Context Wrapper für das Entity Framework zur Verfügung.

Der Object Views Controller sendet die Änderungen in Form von Commands an den Data Context Wrapper, der verwendet werden soll. Dieser ist dann dafür verantwortlich, dass die Änderungen persistent gemacht werden.

5.2.1.3 Id Controller

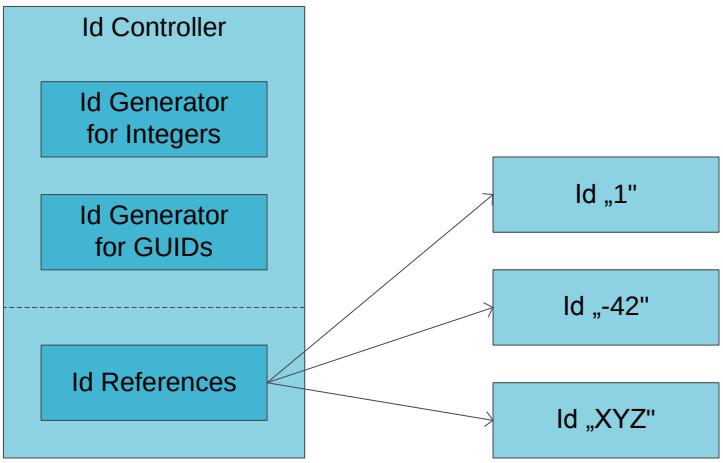


Abbildung 11 - Konzept für Id Controller

Werden neue Data Objects erstellt, dann muss diesen jeweils eine neue, eindeutige Id vergeben werden, die in der Datenbank noch nicht existieren darf. Dazu ist der *Id Controller*, resp. seine *Id Generators* zuständig. Es können Id Generatoren für beliebige Typen von Ids registriert werden. Eine Implementation für Integer-Ids wird vom Object Views Framework bereits angeboten. In diesem Falle werden für die neuen Data Objects Minus-Integer-Zahlen vergeben. Beim Persistieren der Änderungen werden diese Minus-Ids dann vom Data Context Wrapper durch die in der Datenquelle vergebene Id ersetzt.

Des Weiteren hält der Id Controller eine Liste mit Referenzen auf alle Data Object Ids, die in den Object Views verwendet werden. Dieses Konzept verhindert, dass bei einer Änderung der Id (was z.B. beim Speichern eines Data Objects in einer Datenbank geschehen kann), das Object Views Framework mit falschen Ids arbeitet.

5.2.2 Konzepte für Anforderungen

5.2.2.1 Einfügen

Konzept 9.1	Einfügen in eine Object View mit Bezug auf ein einzelnes Data Object
Problem	In einer Object View, welche nur auf ein Data Object verweist, wird eine neue Zeile eingefügt. Diese neue Zeile soll in der Datenquelle persistiert werden.
Beispiel	In der Object View CustomerView wird eine neue Zeile eingefügt, womit in der Datenquelle ein neuer Kunde erfasst werden soll.
Lösung	Das Framework merkt, wenn in der Object View einen Eintrag hinzukommt, erstellt dazu das Data Object und kreiert den Command um einen Datensatz zu persistieren. Dieser muss mit einem Commit bestätigt werden. Beim Commit werden alle ausstehenden Commands einem Data Context Wrapper weitergeleitet, der die Persistierung dann vornimmt. Dieses Verhalten kann vom Verwender des Frameworks überschrieben werden, indem er einen Event Handler in der Object View registriert, welcher das Verhalten beim Einfügen bestimmt.

Konzept 9.2	Einfügen in eine Object View mit Bezug auf mehrere Data Objects
Problem	Eine Object View kann auf mehrere verschiedene Typen von Data Objects verweisen. Dazu muss die Beziehung zwischen den verschiedenen Data Objects ebenfalls abgebildet werden und sämtliche Objekte in der Datenquelle persistiert werden.
Beispiel	In der Object View OrderItemView wird eine neue Zeile eingefügt, womit in der Datenquelle ein neues Order Item *und* ein neues Item erfasst werden sollen.
Lösung	Das Framework legt alle benötigten Data Objects an und stellt die Beziehung her. Für jedes angelegte Data Object wird ein Command erstellt und persistiert, sobald es mit Commit bestätigt wird. Welche Data Objects gespeichert werden sollen (d.h. ob das OrderItem und das Item gespeichert werden sollen oder nur das OrderItem) ist Teil der Businesslogik und kann vom Verwender des Frameworks in der Object View mit einem Attribut angegeben werden. Wird eine Beziehung zu einem Data Object hergestellt, wird dieses zuerst gespeichert bzw. der Create Command zuerst erstellt. Damit ist die Reihenfolge der Speicherung sichergestellt und es sollten keine Abhängigkeiten in der Datenbank verletzt werden.

Konzept 9.3	Einfügen in eine Object View mit Bezug auf abhängige Data Objects
Problem	Eine Abhängigkeit kann beispielsweise durch eine Selektion entstehen. Sobald die Selektion auf ein einzelnes Feld greift, soll diese Abhängigkeit ebenfalls gesetzt werden.
Beispiel	In der Object View OrderView wird eine neue Zeile eingefügt, womit in der Datenquelle eine neue Bestellung erfasst werden soll, die dem via CustomerId selektierten Kunden zugewiesen werden soll.
Lösung	Das Framework erkennt die Selektion und fügt beim Data Object ebenfalls die Beziehung zum anderen Data Object hinzu, indem es das Data Object erstellt und das Feld (z.B. CustomerId) setzt.

5.2.2.2 Bearbeiten

Konzept 10.1	Bearbeiten eines schreibbaren Feldes in einer Object View
Beispiel	Wird eine Object View bearbeitet, muss die entsprechende Änderung in der Datenquelle persistiert werden können.
Beispiel	In der ObjectView CustomerView wird das Feld FirstName bearbeitet. Diese Änderung wird auf das Feld FirstName des entsprechenden Data Objects in der Datenquelle propagiert.
Lösung	Die Änderung auf dem Object View Field wird durch das Framework festgestellt und in das Data Object geschrieben. Zudem merkt sich das Framework diese Änderung, damit bei einem Commit die Änderung in die Datenquelle persistiert werden kann.

5.2.2.3 Löschen

Konzept 11.1	Löschen in einer Object View mit Bezug auf ein einzelnes Data Object
Problem	Wird in einer Object View eine Zeile gelöscht, so soll dieses Löschen auch auf die Datenquelle propagiert werden können.
Beispiel	In der Object View CustomerView wird eine Zeile gelöscht, worauf nach dem Commit der entsprechende Kunde in der Datenquelle gelöscht wird.
Lösung	Sobald ein Eintrag in der Object View gelöscht wird, merkt sich dies das Framework und löscht den Eintrag aus der Datenquelle, sobald mit einem Commit bestätigt wird. Um sich die gelöschten Data Objects zu merken, wird das Data Object im Data Object Cache vom Loaded Data Object Store in den Deleted Data Object Store verschoben. D.h. das Objekt ist zwar immer noch im Cache vorhanden, wird jedoch nicht mehr in den Object Views angezeigt. Dieses Verhalten kann vom Verwender des Frameworks überschrieben werden, indem er einen Event Handler in der Object View registriert, welcher das Verhalten beim Löschen bestimmt.

Konzept 11.2	Löschen in einer Object View mit Bezug auf mehrere Data Objects
Problem	Verweist die Object View auf mehrere Typen von Data Objects, so ist unklar, welche Data Objects tatsächlich aus der Datenquelle gelöscht werden sollen.
Beispiel	In der Object View OrderItemView wird eine Zeile gelöscht. *Gemäss Businesslogik* soll in diesem Fall nun nur das Data Object OrderItem gelöscht werden. Das Data Object Item soll bestehen bleiben.
Lösung	Der Verwender des Frameworks kann pro Object View definieren, welche Typen von Data Objects gelöscht werden sollen. Wird nun ein Eintrag in der Object View gelöscht, merkt sich das Framework nur diejenigen Data Objects, welche schliesslich gelöscht werden sollen. Im obigen Beispiel wird in der Object View OrderItemView angegeben, dass nur Data Objects vom Typ OrderItem gelöscht werden sollen. Deshalb werden vom Framework auch nur diejenigen Data Objects zum Löschen vermerkt, welche den Typ OrderItem haben.

Konzept 11.3	Löschen in einer Object View mit Bezug auf ein oder mehrere Data Objects mit weiteren abhängigen Data Objects
Problem	Wird ein Data Object gelöscht, so stellt sich die Frage, ob untergeordnete Data Objects ebenfalls gelöscht werden sollen oder nicht.
Beispiel	In der Object View CustomerView wird eine Zeile gelöscht. *Gemäss Businesslogik* sollen in diesem Fall nun nebst dem Kunden auch dessen Bestellungen gelöscht werden.
Lösung	Der Verwender des Frameworks kann pro Data Object definieren, welche untergeordneten Data Objects ebenfalls gelöscht werden sollen. Im obigen Beispiel wird für das Data Object Customer angegeben, dass die Orders ebenfalls gelöscht werden sollen. Beim Data Object Order wird wiederum angegeben, dass die Order Items ebenfalls gelöscht werden sollen.

5.2.2.4 Undo und Redo

Konzept 12.1	Rückgängig machen von Änderungen
Problem	Änderungen an einer Object View sollen rückgängig gemacht werden können, solange sie noch nicht in der Datenquelle persistiert worden sind.
Beispiel	In der Object View OrderItemView wird eine Zeile hinzugefügt. Solange die Änderung noch nicht in die Datenquelle propagiert wurde, soll sie vom Benutzer rückgängig gemacht werden können.
Lösung	Das Framework merkt sich jede Änderung, welche an der Object View durchgeführt wurde, d.h. jede Benutzerinteraktion. Diese Änderungen können durch einen entsprechenden Undo Befehl rückgängig gemacht werden.

Konzept 12.2	Wiederherstellen von Änderungen
Problem	Nachdem eine Änderung rückgängig gemacht wurde, soll sie vom Benutzer auch wiederhergestellt werden können, solange die Datenquelle nicht aktualisiert wurde.
Beispiel	Nachdem eine Zeile in der Object View OrderItemView hinzugefügt und dies Aktion dann rückgängig gemacht wurde, soll sie wiederhergestellt werden können.
Lösung	Nachdem das Framework eine Aktion rückgängig gemacht hat, merkt es sie sich diese Aktion nach wie vor. Dadurch kann die Benutzeraktion auch nach einem Undo wiederhergestellt werden.

5.2.2.5 Updateable Collections

Werden als Datenquelle für die Object Views Collections von Plain Old C# Objects verwendet, dann kommt der *Collection Context Wrapper* zum Einsatz. Mit diesem Wrapper werden Änderungen an Daten, die über die Object Views vorgenommen werden, an die Objekte in den Collections weitergeleitet.

5.2.2.6 Updateable Entity Framework Data Context

Analog zum Collection Context Wrapper bietet das Object Views Framework den *Entity Framework Wrapper*, um Änderungen in einem Entity Framework Data Context persistent zu machen.

5.3 Three Tier Object Views

5.3.1 Grundkonzept

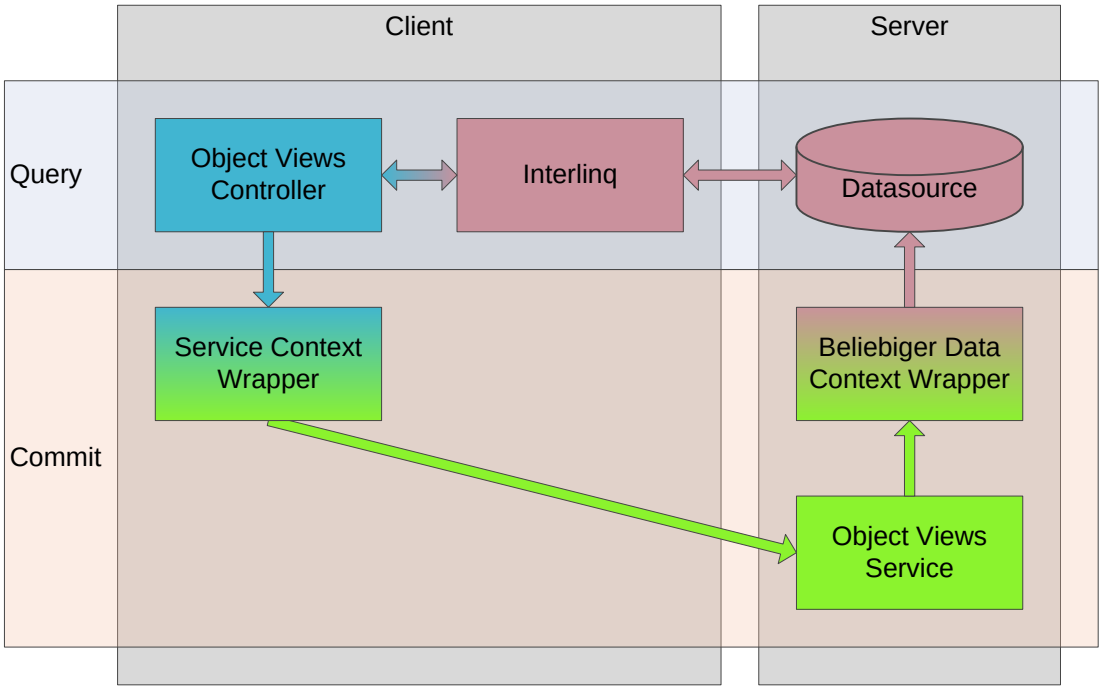


Abbildung 12 - Konzept für Three Tier Object Views

5.3.1.1 Interlinq

Interlinq ist eine Softwarekomponente, die Linq-Queries an entfernte Data Kontexte in einer Three Tier Umgebung weiterleitet. Das Object Views Framework verwendet Interlinq, damit auf dem Client weiterhin mit Linq Queries gearbeitet werden kann, wie in einer Single oder Two Tier Umgebung.

5.3.1.2 Service Context Wrapper

Zum Persistieren von Datenänderungen werden sog. Data Context Wrapper benutzt, wie im Konzept für Updateable Object Views beschrieben. Der Service Context Wrapper ist ein solcher Data Context Wrapper. Er nimmt die Änderungen im Data Context jedoch nicht selber vor, sondern leitet sie dem Object Views Service weiter.

5.3.1.3 Object Views Service

Der Object Views Service läuft auf dem Server innerhalb einer Hostanwendung, die vom Verwender des Frameworks erstellt wird. Er nimmt vom Service Context Wrapper Datenänderungen entgegen und leitet sie seinerseits einem beliebigen Data Context Wrapper weiter. Was das genau für ein Wrapper ist, das wird vom Programmierer der Hostanwendung festgelegt. Das hängt natürlich davon ab, mit was für einem Persistenz-Framework gearbeitet werden will.

5.3.2 Konzepte für Anforderungen

5.3.2.1 Lesen von Remote Data Context

Konzept 13.1	Lesen von Remote Data Context
Problem	Beim Lesen aus der Datenquelle in einer Three Tier Umgebung stellt sich das Problem, dass sich der Data Context auf dem Server befindet und die Queries zum Server übermittelt werden müssen und dieser dann die gewünschten Daten aus seinem Data Context zurückliefert. Linq Queries sind nicht serialisierbar.
Beispiel	Die Datenbank, sowie ein Service, befinden sich auf dem Server. Der Service hält einen Data Context zur Datenbank. Ein Client A verbindet sich mit dem Service und möchte folgenden Query in Richtung des Service absetzen: `from c in db.Customer select new {CustomerId = c.CustomerId, FirstName = c.FirstName, LastName = c.LastName};`
Lösung	Die Lösung für dieses Problem bietet sich mit Interlinq an. Interlinq ist eine Softwarekomponente, die die Serialisierung und Übertragung von Queries und Daten in einer Three Tier Umgebung übernimmt.

5.3.2.2 Schreiben in Remote Data Context (Einfügen, Bearbeiten, Löschen)

Konzept 14.1	Schreiben in Remote Data Context (Einfügen, Bearbeiten, Löschen)
Problem	Beim Schreiben stellt sich ein ähnliches Problem wie beim Lesen aus einem entfernten Data Context. Die Objekte können lokal beim Client zwar verändert werden, es gibt jedoch keinen Data Context, in dem man die Änderung persistent machen könnte.
Beispiel	Selbiges Szenario wie beim Lesen. Der Client verändert nun den Nachnamen des Customers und möchte die Änderung persistent machen.
Lösung	Auf dem Server wird ein zusätzlicher Service angeboten, der einen Data Context zur Datenbank hält und Änderungen vom Client entgegennehmen kann.

Konzept 14.2	Konfliktlösung beim Schreiben
Problem	Beim Schreiben in eine entfernte Datenquelle kann die Situation auftreten, dass ein anderer Client dieselben Daten bereits bearbeitet hat.
Beispiel	Client A sowie auch Client B laden die Customer View. Beide bearbeiten den Vornamen desselben Kunden. Client A speichert seine Änderungen, danach speichert auch Client B. Dies führt zu einer Konfliktsituation.
Lösung	Diese Konfliktlösung ist Teil der Data Context Wrapper. Eine Konfliktsituation muss von der entsprechenden Implementierung gehandhabt werden. Dabei wird vom Object Views Framework der alte aus der Datenquelle gelesene Wert übermittelt sowie der neue vom Benutzer eingegebene Wert. Der bereits implementierte Entity Framework Wrapper schreibt die Änderung eines Data Object Feldes nur in die Datenbank, wenn dieses Feld seit dem Lesen der Daten nicht verändert wurde.

5.3.2.3 Übernehmen von Änderungen aus Remote Data Context

Konzept 15.1	Übernehmen der geänderten Data Object Id nach Einfügen
Problem	Neu erstellten Data Objects werden auf dem Client vorerst lokale Ids (z.B. Minuswerte) vergeben, um sie eindeutig identifizieren zu können. Beim Speichern im entfernten Data Context wird den Data Objects von der Datenquelle dann eine neue Id vergeben. Damit die Clients nicht weiterhin mit den lokalen Ids arbeiten, müssen sie über diese Änderung informiert werden.
Beispiel	Auf dem Client wird in der Customer View eine neue Zeile eingefügt, was dazu führt, dass ein neues Customer Data Object mit der Id -1 erstellt wird. Beim Commit wird dem neuen Customer von der Datenquelle die Id 42 vergeben.
Lösung	Der Service, der vom Client Änderungen entgegennimmt, kann dem Client selbst Änderungen zurücksenden. In diesem Fall sendet er dem Client zurück, dass sich die Id vom neu erstellten Data Object geändert hat.

Konzept 15.2	Übernehmen der Änderungen anderer Clients
Problem	Arbeiten mehrere Clients gleichzeitig auf derselben Datenquelle, dann sehen alle anderen Clients veraltete Daten, sobald ein Client seine Änderungen persistiert.
Beispiel	Client A sowie auch Client B laden die Customer View. Client A bearbeitet den Vornamen eines Kunden und speichert seine Änderungen. Client B sieht jedoch noch immer den alten Vornamen.
Lösung	Das Übernehmen der im entfernten Data Context vorgenommenen Änderungen wurde im Object Views Framework *mit Einschränkungen vorbereitet*. Das Konzept sieht vor, dass die Data Context Wrapper nach dem Commit eigener Änderungen die Änderungen der anderen Clients zurückgeben. Diese Funktionalität ist in den bereitgestellten Wrappern jedoch noch nicht implementiert.

6 Softwarearchitektur

In diesem Kapitel wird die Architektur des entwickelten Produkts beschrieben. Es soll damit Aufschluss darüber geben, wie die einzelnen Konzepte konkret umgesetzt wurden. Zuerst werden allgemeine architektonische Entscheidungen erläutert, die grössere Teile der Architektur betreffen. Danach werden unterschiedliche Sichten auf die Struktur der Software angeboten.

6.1 Architektonische Entscheidungen

Unit of Work

Werden in Object Views vom Benutzer Daten manipuliert, dann kommt das Unit of Work Pattern gemäss Fowler [1] zum Einsatz. Alle Änderungen (Einfügen, Bearbeiten, Löschen) werden als *Commands* in einem *Change Tracker* abgelegt. Sobald die Änderungen persistiert werden sollen, werden die noch nicht persistierten Commands zu einem *Change Set* zusammengefasst. Dieses Change Set wird dann in einer Transaktion Richtung Datenquelle propagiert.

Identity Field

Wird eine Query ausgeführt, dann werden die geladenen Daten in *Data Objects* in einem lokalen *Cache* abgelegt. Zur Identifizierung dieser Data Objects wird das Identity Field Pattern von Fowler [1] eingesetzt.

Das Object Views Framework hat die Anforderung, dass für jedes Data Object, das mit einer Query geladen wird, die *Id* aus der Datenbank mitgeladen wird. Das Framework überprüft alle Queries, ob dies auch wirklich gemacht wird, ansonsten wird eine Exception geworfen, um den Verwender des Frameworks darauf aufmerksam zu machen.

Diese Id wird dazu benutzt, die Data Objects im lokalen Cache zu identifizieren. Werden vom Benutzer in der Object View neue Zeilen hinzugefügt, wodurch neue Data Objects erstellt werden, dann vergibt das Framework diesen eine *lokale Id*. Solche lokalen Ids werden von sog. *Id Generators* generiert. Vom Id Generator für Integer-Ids werden z.B. Minus-Integer-Zahlen als lokale Ids vergeben.

Beim Speichern der neuen Data Objects wird diesen von der Datenquelle eine neue (definitive) Id vergeben. Diese neue Id wird dem Framework zurückgereicht, damit es die Ids der lokalen Data Objects angleichen kann.

Optimistic Offline Lock

Im Umgang mit Konkurrenzsituationen bei Object Views in einer Two oder Three Tier Umgebung wird die Strategie Optimistic Offline Lock [1] angewandt. Das Object Views Framework erlaubt es mehreren Benutzern, gleichzeitig dieselben Daten zu laden und zu manipulieren. D.h. die Daten werden nicht gesperrt.

Beim Persistieren der Änderungen wird überprüft, ob die lokalen Daten auf dem Client noch aktuell sind. Dies wird gemacht, indem bei jedem Client immer der ursprüngliche Wert aus der Datenquelle und der aktuelle Wert bekannt sind.

Bei einer Konfliktsituation (z.B. wenn zwei Clients gleichzeitig dieselben Daten bearbeitet haben), dann wird im Moment nur eine Lösungsstrategie angeboten: Derjenige der die Änderung zuerst persistiert, der gewinnt. Es ist natürlich denkbar, noch andere Strategien zu implementieren und dem Benutzer als Auswahl anzubieten.

Lazy Load

Verwendet man das Object Views Framework, dann kommt Lazy Loading [1] zum Einsatz. Bei Object Views werden immer nur die nötigsten Daten geladen, in den meisten Fällen werden nicht einmal die ganzen Objekte geladen. Abhängige Objekte werden nur dann geladen, wenn dies in der Query auch so definiert ist.

Command Pattern

Das im Object Views Framework integrierte Change Tracking arbeitet mit dem Command Pattern [2]. Jegliche vom Benutzer vorgenommenen Datenmanipulationen werden als *Commands* gespeichert. Solche Commands lassen sich ausführen und es kann auch ein Undo resp. Redo durchgeführt werden. Mit dem Change Tracking wird also gleichzeitig auch ein Undo/Redo Mechanismus angeboten.

Visitor Pattern

Um die Metadaten für eine Object View zu erstellen, wird die Query analysiert, aufgrund der die View entstehen soll. Für diese Analyse, dieses Parsing, wird das Visitor Pattern [2] angewandt. Dies hat sich so ergeben, da Microsoft eine Basisklasse anbietet für das Parsen von Queries, resp. von deren *Expression Trees*. Wie die Klasse aussieht und wie sie zu verwenden ist, ist in der MSDN dokumentiert [3].

Wrapper

Damit das Object Views Framework unabhängig ist vom Persistenz-Framework, kommen das Adapter und Strategy Pattern [2] zum Einsatz. Es sorgen nämlich sog. *Data Context Wrapper* dafür, dass beliebige Persistenz-Frameworks eingesetzt werden können. Dem Object Views Framework wird mitgeteilt, welcher Wrapper zum Einsatz kommt. Dieser hält selbst eine Instanz eines Data Contexts (z.B. ein Entity Framework Data Context). Will nun das Object Views Framework Datenänderungen persistieren, dann reicht es die Änderungen einfach dem Wrapper. Dieser leitet sie dann dem Data Context weiter.

Diese Architektur kann auch in einer Three Tier Umgebung angewandt werden. Das Object Views Framework verwendet in dieser Situation als Wrapper einen WCF-Client, der die Datenänderungen dem Service weiterleitet. Der Service selbst hat wiederum einen Wrapper, der dann die Weiterleitung der Datenänderungen an den Data Context übernimmt.

6.2 Logische Sicht

6.2.1 Übersicht

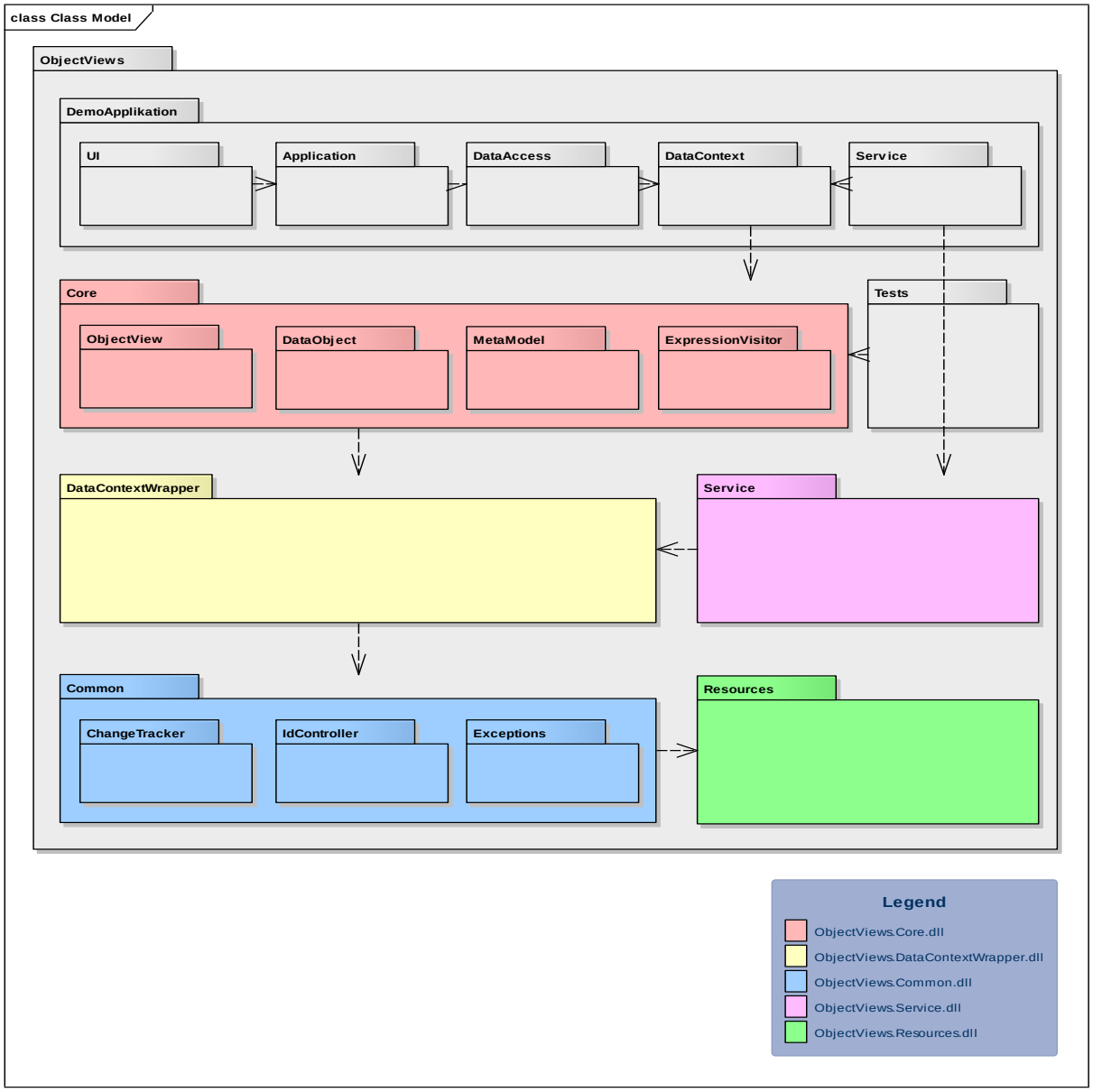


Abbildung 13 - Paketdiagramm

Dies ist eine Übersicht über alle relevanten im Object Views Framework enthaltenen Assemblies und Pakete resp. Namespaces. Sie werden im Folgenden detailliert erklärt.

Für die Verwendung des Frameworks notwendig sind die Pakete ObjectViews.Core, ObjectViews.Service, ObjectViews.DataContextWrapper, ObjectViews.Common und ObjectViews.Resources.

ObjectViews.DemoApplikation und ObjectViews.Tests müssen von Verwendern des Frameworks nicht eingebunden werden.

6.2.2 ObjectViews.Core

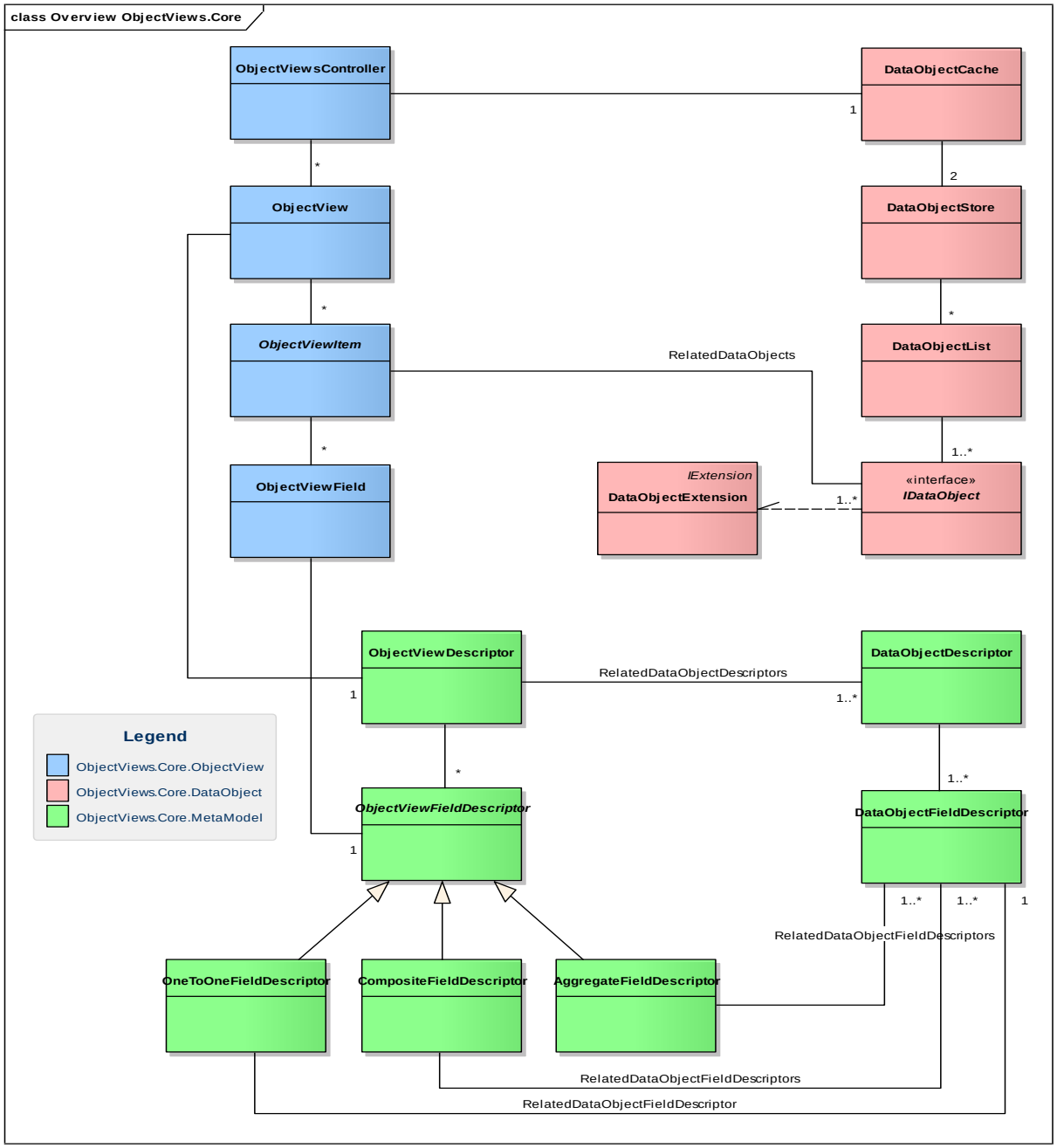


Abbildung 14 - Klassendiagramm ObjectViews.Core

Das Paket ObjectViews.Core enthält die Kern-Klassen des Object Views Frameworks. Es ist in folgende Unterpakete aufgeteilt: ObjectView, DataObject, MetaModel und ExpressionVisitor. Diese Unterpakete werden in den folgenden Kapiteln detailliert beschrieben.

Das obige Klassendiagramm zeigt das Zusammenspiel zwischen den Object Views, den Data Objects und dem Metamodell. Zwischen den drei Paketen bestehen folgende wichtige Beziehungen:

- Der *ObjectViewsController* hält eine Instanz des *DataObjectCaches.*
- Eine einzelne Zeile in einer Object View zeigt Daten von einem oder mehreren Data Object an (*RelatedDataObjects*).
- Was in der Object View alles angezeigt wird (welche Felder von welchen Data Objects), wird mit dem Metamodell beschrieben → Jede *ObjectView*-Instanz besitzt eine *ObjectViewDescriptor*-Instanz, die wiederum weitere Deskriptoren enthält.

6.2.2.1 ObjectViews.Core.ObjectView

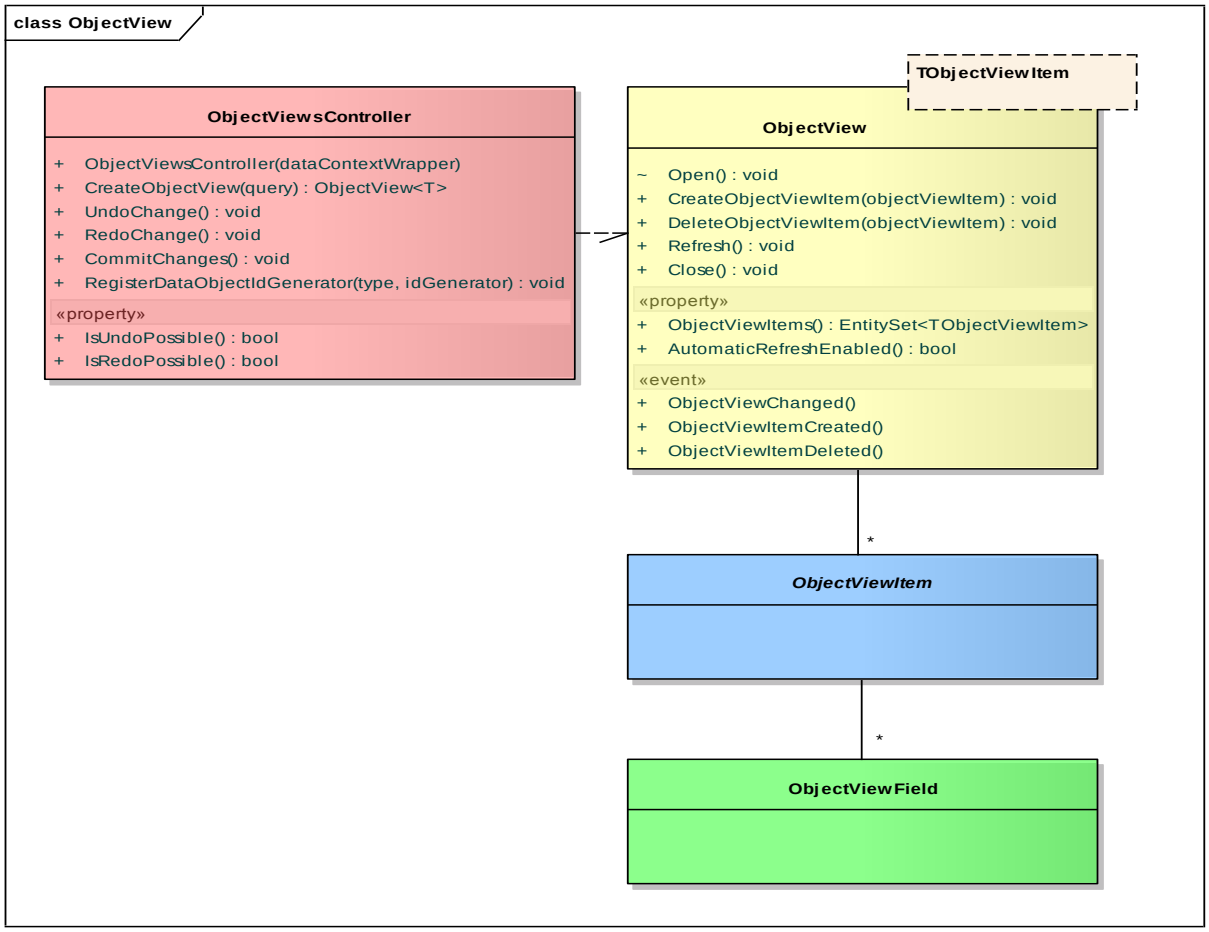


Abbildung 15 - Klassendiagramm ObjectViews.Core.ObjectView

Die Klasse *ObjectViewsController* ist die hauptsächliche Schnittstelle zum Verwender des Frameworks. Mit dieser können Object Views erstellt und Änderungen innerhalb der Object Views rückgängig gemacht, wiederhergestellt und gespeichert werden.

Ein Objekt der Klasse *ObjectView* repräsentiert eine spezifische View, die Zeilen enthält (*ObjectViewItem*), welche wiederum aus mehreren Feldern bestehen (*ObjectViewField*). Die Klasse *ObjectView* bietet dem Verwender des Frameworks zusätzliche Methoden und Events.

Beispiel

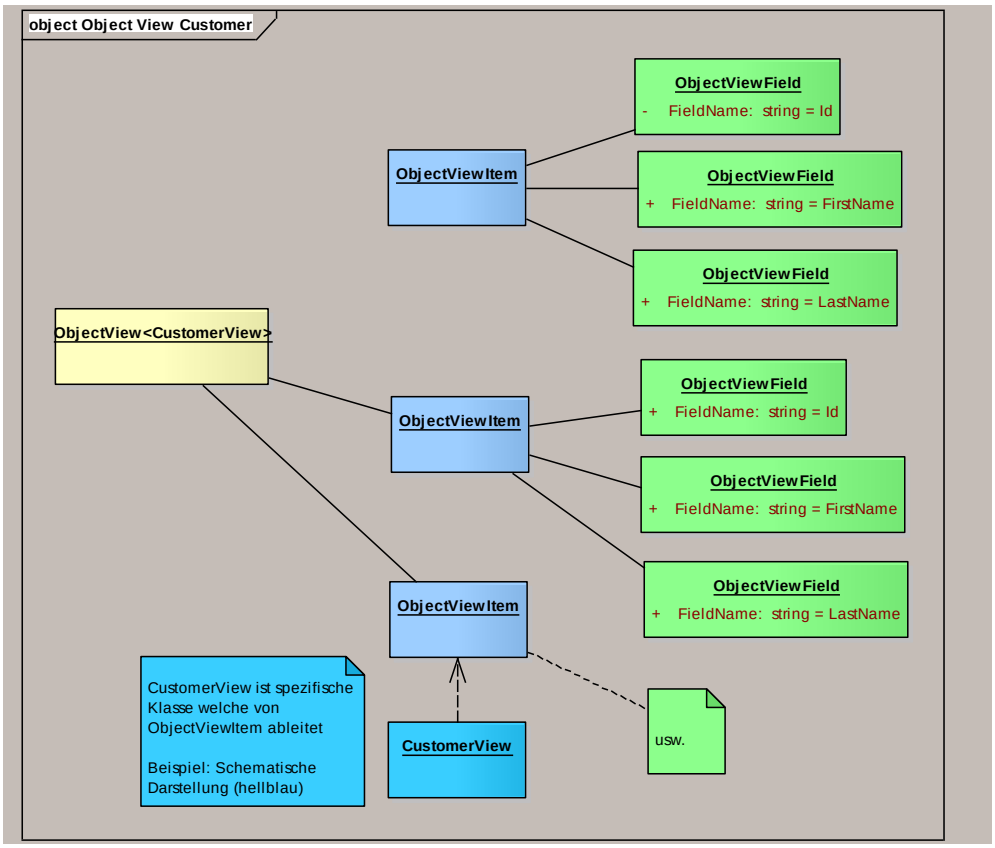
Tabelle „Customer“

Id	Vorname	Nachname
1	Rolf	Latzer
3	Beni	Egli
4	Hans	Mustermann

Mit folgendem Programmcode

```
from customer in context.Customers
select new CustomerView { CustomerId = customer.CustomerId,
FirstName = customer.FirstName, LastName = customer.LastName
};
```

Ergibt folgendes Objektdiagramm



6.2.2.2 ObjectViews.Core.DataObject

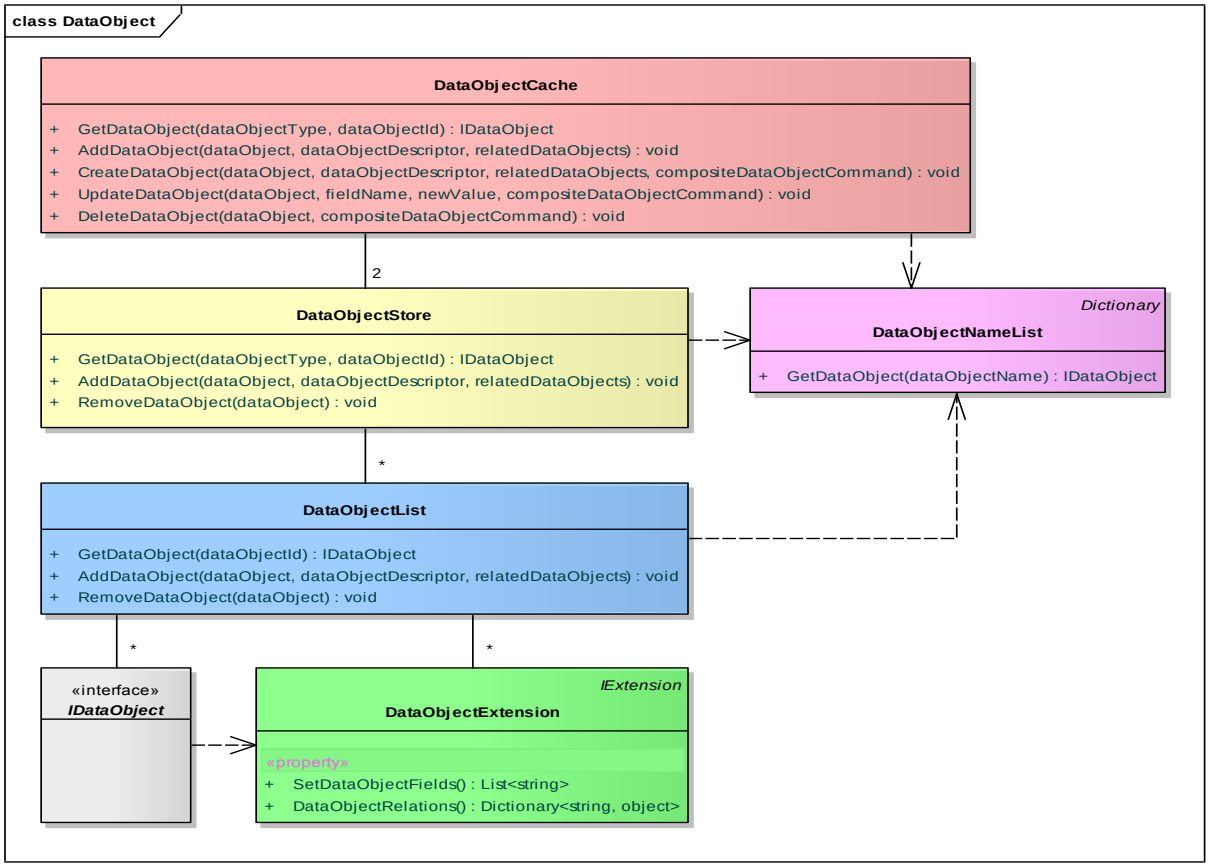


Abbildung 16 - Klassendiagramm ObjectViews.Core.DataObject

Der Object Views Controller hält eine Instanz der Klasse *DataObjectCache*, welche die Struktur aufspannt, wo die Data Objects beim Client lokal gespeichert werden. Der Data Object Cache ist also eine Sammlung von teilweise oder ganz abgefüllten Data Objects. Dieser Cache erlaubt es dem Framework, Queries lokal auf dem Client auszuführen. Dadurch können Object Views lokal aktualisiert werden.

Die Klasse *DataObjectStore* dient dazu, gelöschte Data Objects von den nicht gelöschten Data Objects zu unterscheiden. Daher hält der Data Object Cache zwei Instanzen dieser Klasse (einen LoadedDataObjectStore und einen DeletedDataObjectStore).

Die Data Objects werden in einem Objekt der Klasse *DataObjectList* aufgelistet. Es gibt für jeden Typ von Data Objects eine eigene Data Object Liste. Die Data Objects innerhalb der Liste werden über ihre Ids identifiziert.

Objekte der Klasse *DataObjectExtension* enthalten zusätzliche Informationen zu den Data Objects. Jedes sich im Cache befindliche Data Object besitzt also ein solche Extension-Objekt, in dem das Framework benötigte Informationen zum Data Object speichert.

Die Klasse *DataObjectNameList* dient dazu, beliebige Listen von Data Objects zu halten, in welchen die Data Objects über einen bestimmten Namen identifiziert werden. Solche Listen werden z.B. dazu verwendet, um alle zu einem Object View Item gehörenden Data Objects in einer Liste zu halten, wobei die Data Objects über den Namen identifiziert werden, der vom Verwender des Frameworks im Query definiert wird.

6.2.2.2.1 ObjectViews.Core.DataObject.Commands

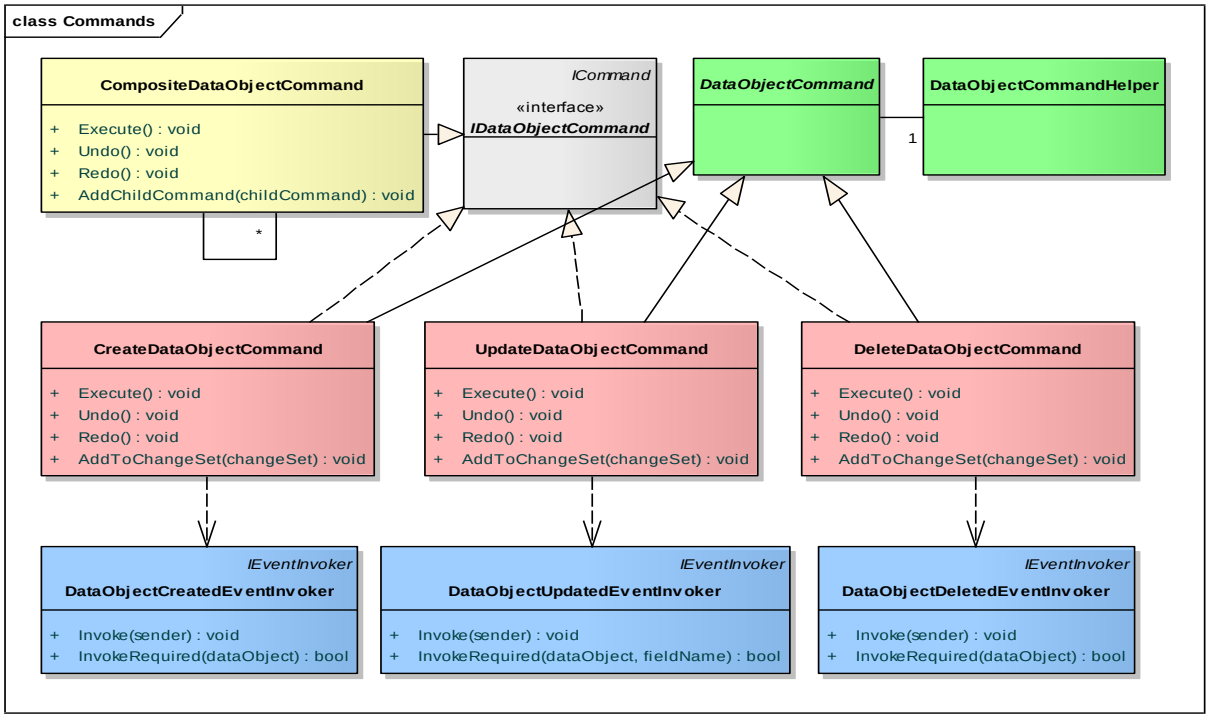


Abbildung 17 - Klassendiagramm ObjectViews.Core.DataObject.Commands

Da das Object Views Framework keinerlei Abhängigkeiten zu einem bestimmten Persistenz-Framework haben soll, sind auch alle CRUD Operationen völlig unabhängig gehalten. Damit jedoch dennoch die einzelnen CRUD Operationen abgebildet werden können, haben wir für Create, Update und Delete die Klassen *CreateDataObjectCommand*, *UpdateDataObjectCommand* und *DeleteDataObjectCommand* erstellt.

Solche Command-Objekte sind also Operationen (Create, Update, Delete), die auf einem bestimmten Data Object ausgeführt werden. Mit der Auflistung solcher Commands kann aufgezeichnet werden, auf welchen Data Objects Änderungen gemacht wurden. Dies dient einerseits dem Change Tracking, als auch dem Durchführen der Undo/Redo Operationen.

Die Klasse *CompositeDataObjectCommand* kann mehrere solcher Commands zusammenfassen. Wenn z.B. in einer Object View eine neue Zeile eingefügt wird, dann werden dadurch unter Umständen mehrere Create

Commands abgesetzt. Soll nun ein Undo gemacht werden, dann will man alle diese Create Commands auf einmal rückgängig machen und nicht nur einen davon.

Die *EventInvoker*-Klassen sind im Prinzip Event Handler, jedoch mit der Erweiterung, dass der Event Handler nur unter bestimmten Umständen aufgerufen wird. Diese Bedingung wird in der Methode *InvokeRequired* entschieden.

Die zwei Klassen *DataObjectCommand* und *DataObjectCommandHelper* bieten Methoden an, die von allen anderen Command-Klassen verwendet werden. Gewisse Methoden der Helper-Klasse werden auch noch an anderen Orten verwendet, daher die Aufteilung auf zwei Klassen.

6.2.2.2.2 ObjectViews.Core.DataObject.IdController

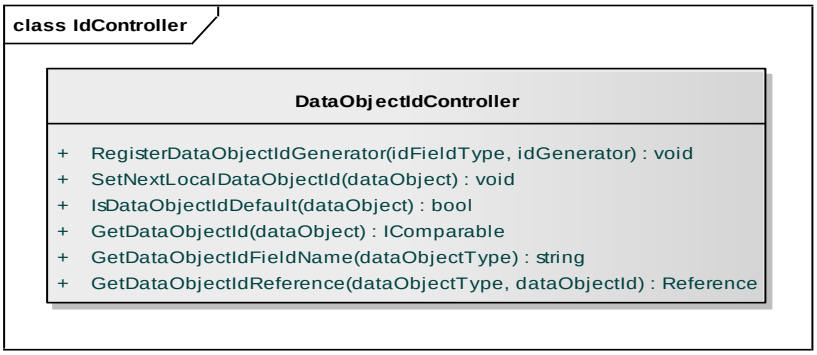


Abbildung 18 - Klassendiagramm ObjectViews.Core.DataObject.IdController

Der *DataObjectIdController* übernimmt verschiedene Aufgaben im Zusammenhang mit den Ids der Data Objects. Er bietet z.B. Methoden an für den Umgang mit lokalen Ids, die neuen Data Objects vergeben werden, bevor sie in der Datenquelle persistiert werden. Des Weiteren kann man mittels dieser Klasse z.B. auch ganz einfach nur die Id eines Data Objects ermitteln, was im Framework natürlich an vielen Orten verwendet wird.

6.2.2.3 ObjectViews.Core.MetaModel

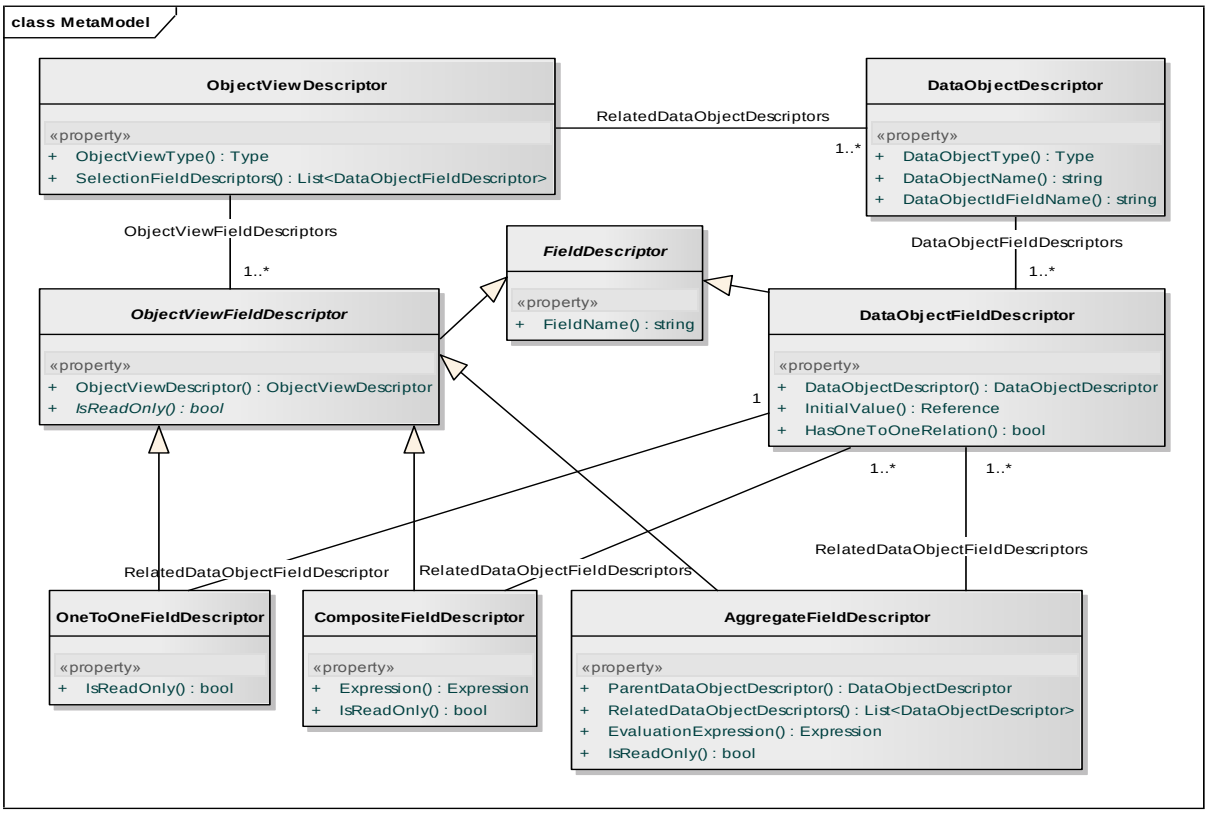


Abbildung 19 - Klassendiagramm ObjectViews.Core.MetaModel

Das Metamodell beschreibt, wie eine spezifische Object View aussieht, d.h., aus was für Data Objects sie besteht und welche Felder dieser Data Objects angezeigt werden. Jede Object View besitzt also ein Objekt der Klasse *ObjectViewDescriptor*, welches wiederum weitere Deskriptor-Objekte gemäss Diagramm enthält.

6.2.2.4 ObjectViews.Core.ExpressionVisitor

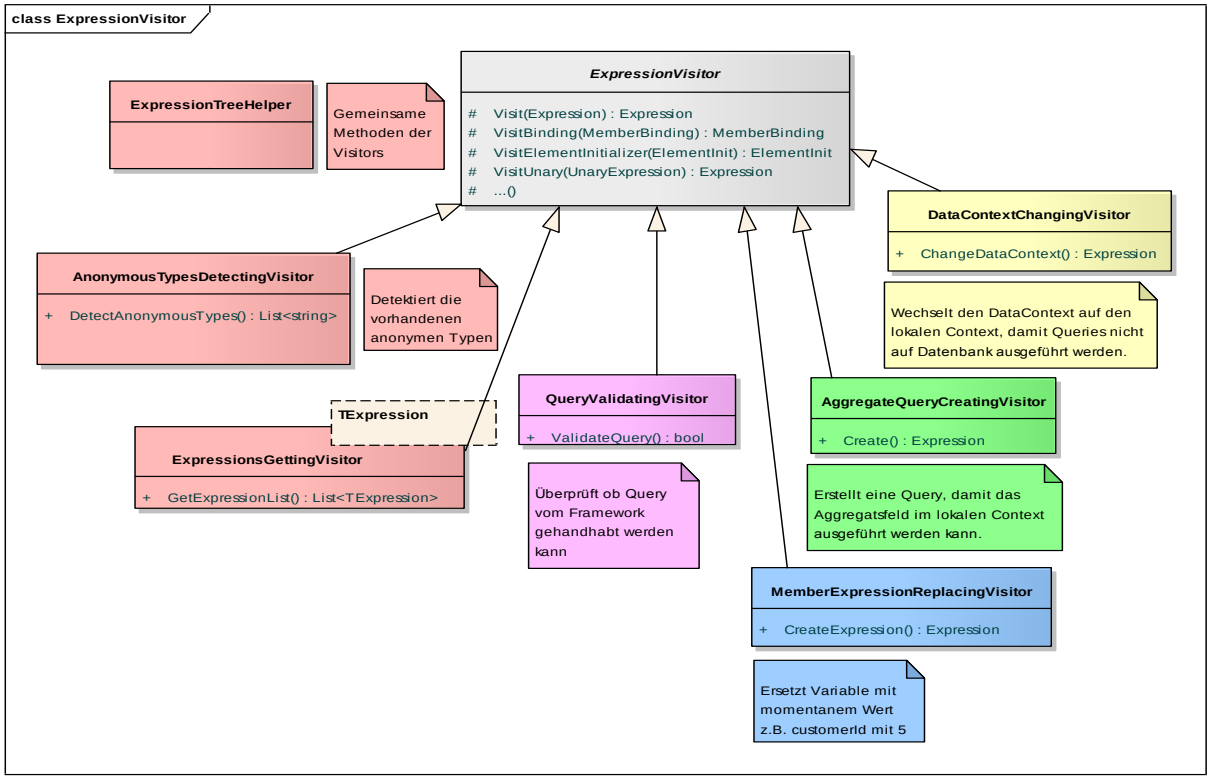


Abbildung 20 - Klassendiagramm ObjectViews.Core.ExpressionVisitor

Die Hilfsklassen stellen Funktionalität zur Verfügung, welche von mehreren Visitors genutzt werden:

- Gemeinsame Methoden werden im *ExpressionTreeHelper* abgelegt.
- Ein Query generiert implizit anonyme Typen, welche mit dem *AnonymousTypesDetectingVisitor* ermittelt werden können.
- Der *ExpressionGettingVisitor* holt eine Liste mit allen Expression eines bestimmten Expression-Typs (z.B. MemberExpression). Es kann zudem ein Prädikat angegeben werden, welches erfüllt sein muss, damit die Expression in der Liste erscheint.

Mit dem *QueryValidatingVisitor* wird überprüft, ob das Query die vom Framework geforderten Einschränkungen erfüllt:

- Implementieren alle Data Objects unser Interface *IDataObject?*
- Werden alle Ids von den verwendeten Data Objects ebenfalls abgeholt?
- Werden nicht mehr als zwei Data Objects in einem Statement referenziert? (z.B. solche Statements sind nicht erlaubt: *orderItem.Order.Customer*)
- Beinhaltet das Where-Statement nur eine binäre Abfrage?
- Usw.

Der *MemberExpressionReplacingVisitor* ersetzt eine Variable mit einem konstanten Wert. Dies ist nötig, damit zusammengesetzte Felder evaluiert werden können.

Beispiel	
Feld CustomerName wird wie folgt evaluiert	`customer.FirstName + „ „ + customer.LastName;`
Und muss nun durch konkrete Werte ersetzt werden	`„Beni“ + „ „ + „Egli“`

Nach der Evaluation wird dies zu	CustomerName = Beni Egli

Damit das Aggregatsfeld im lokalen Kontext ausgeführt werden kann, muss die ursprüngliche Aggregatsexpression verwendet werden und zudem mit einem Where-Statement erweitert werden. Dies ist insofern nötig, damit das Aggregatsfeld nur für den aktuellen Datensatz ausgeführt wird. Der *AggregateQueryCreatingVisitor* erstellt deshalb dynamisch im Expression Tree dieses Where-Statement.

Beispiel

Tabelle „Customer“

Id	Vorname	Nachname	Anzahl Bestellungen
1	Rolf	Latzer	1
2	Beni	Egli	2

Ablauf für Aggregatsfeld

Die Aggregatsquery sieht wie folgt aus:

```
from customer in _shopDataContext.Customers
select new CustomerView { OrderCount =
customer.Order.Count()}
```

und wird nun mit AggregateQueryCreatingVisitor zu:

```
from customer in _shopDataContext.Customers
where customer.CustomerId.Equals(1)
select new CustomerView { OrderCount =
customer.Order.Count()}
```

Dadurch wird nur die Anzahl Orders für den Kunden “Rolf Latzer” abgerufen und nicht für alle Kunden ausgewertet.

Der *DataContextChangingVisitor* ersetzt sämtliche Datenkontexte in einem Query durch den lokalen Kontext aus unserem Framework.

6.2.2.4.1 ObjectViews.Core.ExpressionVisitor.DescriptorsGeneratingVisitor

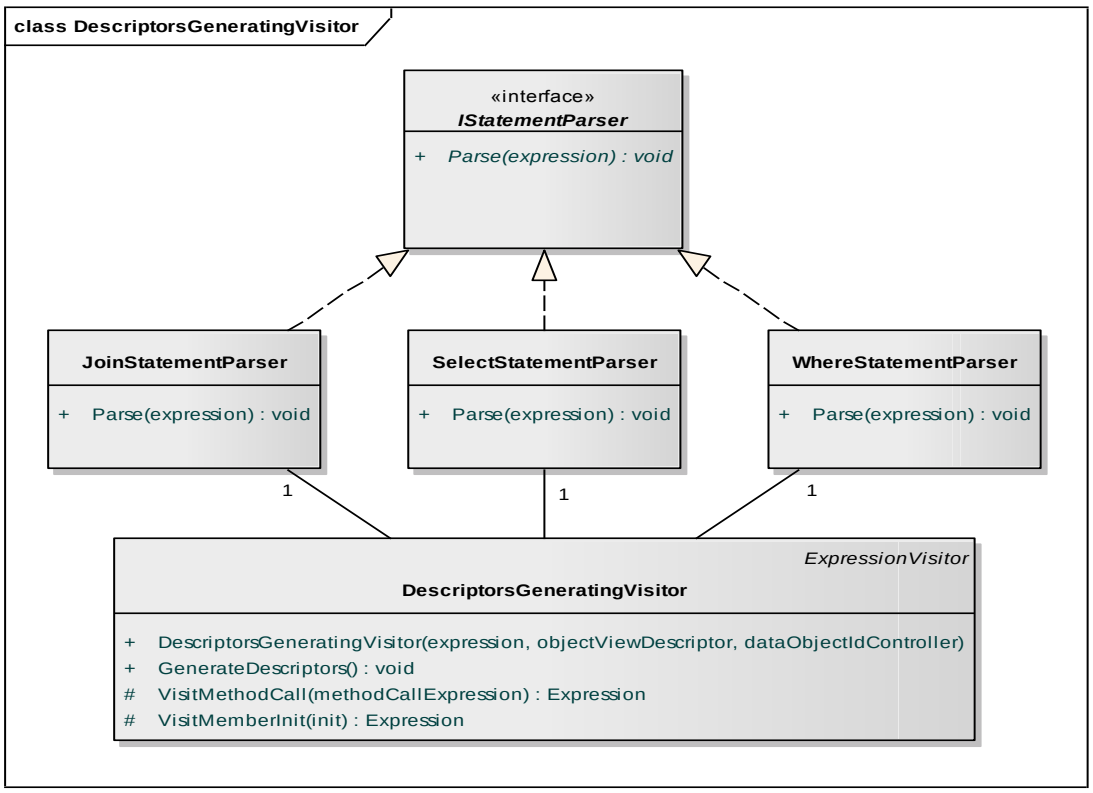


Abbildung 21 - Klassendiagramm ObjectViews.Core.ExpressionVisitor.DescriptorsGeneratingVisitor

Die Funktionalität soll mittels folgendem Linq Query erläutert werden:

```
from customer in _shopDataContext.Customers
join order in _shopDataContext.Orders on customer equals order.Customer
where customer.CustomerId.Equals(customerId)
select new OrderView {OrderId = order.OrderId,
                      CustomerId = customer.CustomerId,
                      CreationDate = order.CreationDate,
                      TotalAmount = order.OrderItem.Sum(orderItem =>
                      (double)(orderItem.Quantity*orderItem.Item.PriceCHF))
}
```

Nachfolgend wird anhand der Klassen / Interfaces beschrieben welche Informationen aus dem obenstehenden Query geparst werden:

Klasse / Interface	Beschreibung
DescriptorsGeneratingVisitor	Besucht nach Aufruf der Methode *GenerateDescriptors()* den ganzen Expression Tree mittels dem Visitor Pattern. Dabei delegiert er Teile des Expression Trees an den entsprechenden Statement Parser.
IStatementParser	Interface für die Parser
JoinStatementParser	Dieser Parser stellt die Beziehungen zwischen den Data Objects her. Im obigen Beispiel wird dabei eine Beziehung zwischen Order.Customer → Customer hergestellt. Somit ist über die Deskriptoren bekannt, dass über das Feld Customer im Data Object Order das Data Object Customer referenziert wird.

WhereStatementParser	Mit dem Where Statement wird die Selektion eingeschränkt. Auch hier wird wiederum geparst auf welche Data Objects bzw. welche Felder der Data Objects die Einschränkung stattfindet.
	Dadurch können gleich zwei Dinge erreicht werden:
	• Veränderungen können detektiert werden
	Beispiel:
	`where customer.City == „Zürich"`
	Es können nun Kunden welche den Ort auf Zürich ändern hinzukommen oder wegfallen.
	• Neue im Object View erstellte Items erhalten gleich die richtige Zuweisung
	Beispiel: In der Order View
	`where customer.CustomerId.Equals(3)`
	Nun wird jeder neu erstellten Order im Object View gleich die Referenz auf den Kunden mit der Id 3 zugewiesen.

SelectStatementParser	Beim Parsen des Select Statements werden Deskriptoren für alle Object View-Felder erstellt: • OneToOneFields (z.B. OrderId, CustomerId, CreationDate) • CompositeFields (kein Composite Field im Beispiel vorhanden) • AggregateFields (z.B. TotalAmount) Hiermit ist immer bekannt welche Object View Felder auf welche Data Object Felder verweisen und welche Funktionen (Aggregatsfunktionen, Funktionen auf zusammengesetzte Felder) zur Evaluation angewandt werden.

6.2.3 ObjectViews.DataContextWrapper

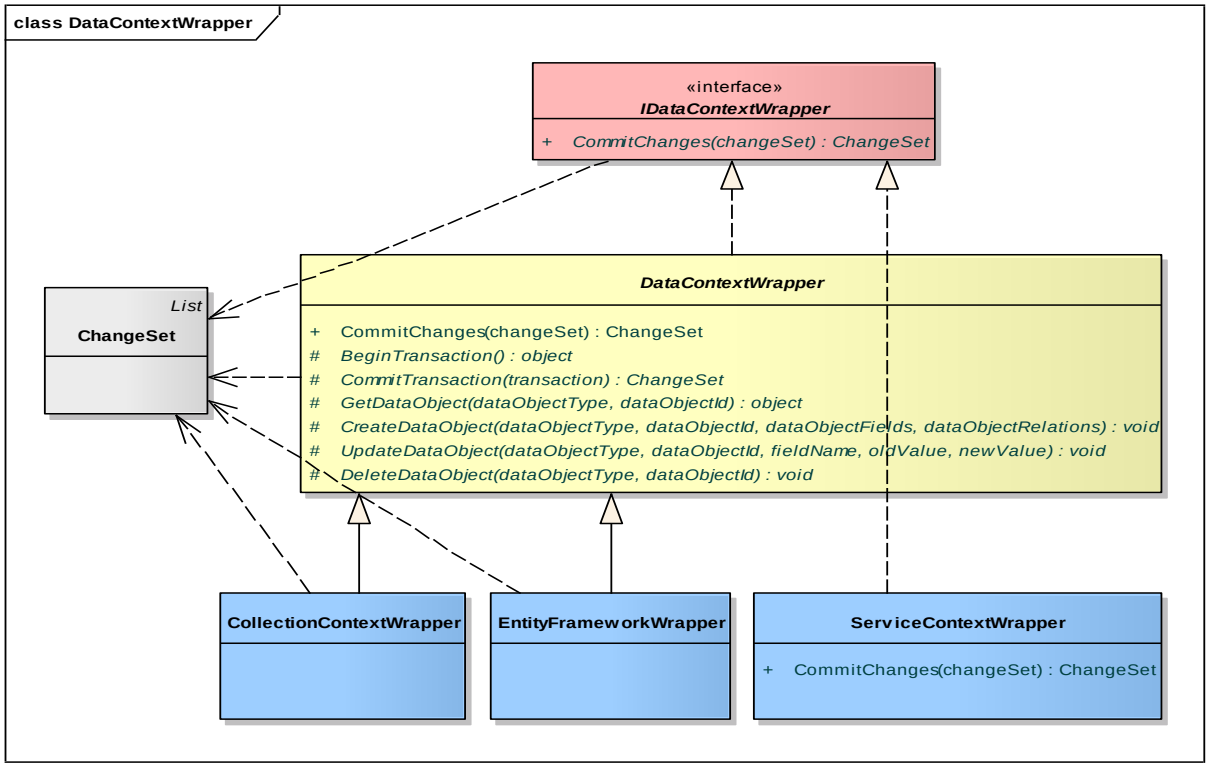


Abbildung 22 - Klassendiagramm ObjectViews.DataContextWrapper

Da jedes Persistenz-Framework seine Eigenheiten hat, wie Data Objects persistiert werden, wurde auch dieser Teil abstrahiert. Soll ein Persistenz-Framework unterstützt werden, muss für dieses einen entsprechenden Wrapper implementiert werden (abgeleitet von *IDataContextWrapper*).

Die Basisklasse *DataContextWrapper* kann aber muss nicht für die Entwicklung eines Wrappers verwendet werden. Sie bietet bereits eine Implementierung der Methode CommitChanges und verlangt eine Implementierung der abstrakten Methoden.

Die Wrapper *CollectionContextWrapper, EntityFrameworkWrapper und ServiceContextWrapper* wurden bereits von uns implementiert. Diese übernehmen folgende Aufgaben:

- CollectionContextWrapper: Persistierung in Listen von Plain Old C# Objects.
- EntityFrameworkWrapper: Persistierung in einem Entity Framework Data Context.
- ServiceContextWrapper: Weiterleitung des Change Sets an einen Service.

6.2.4 ObjectViews.Service

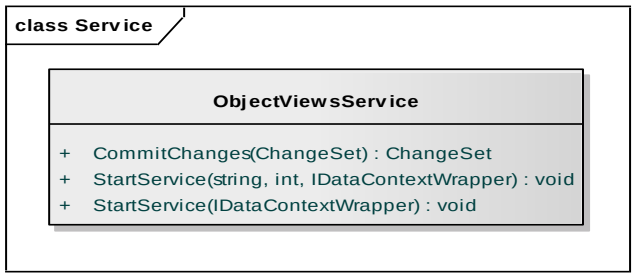


Abbildung 23 - Klassendiagramm ObjectViews.Service

Der ObjectViewsService bietet einen WCF-Dienst an, um die Änderungen an den Data Objects zu persistieren. Dazu muss der Dienst mit einem Data Context Wrapper gestartet werden. Der Wrapper definiert, welches Persistenz-Framework (EntityFramework, o.ä.) verwendet werden soll.

6.2.5 ObjectViews.Common

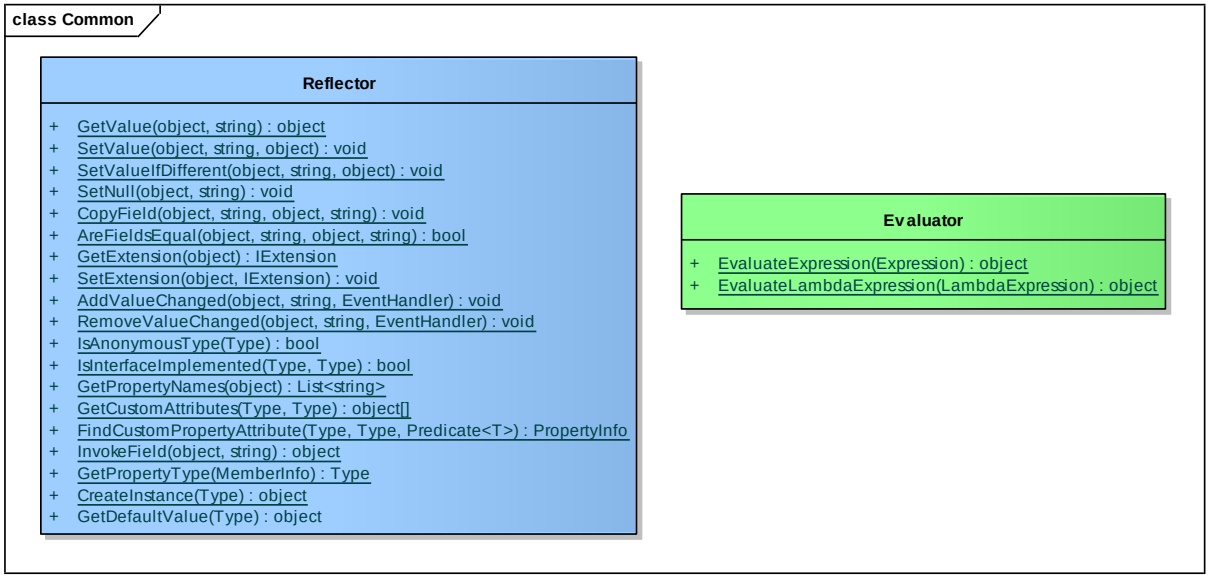


Abbildung 24 - Klassendiagramm ObjectViews.Common

Der *Reflector* abstrahiert sämtliche verwendete Reflection Methoden.

Mit dem *Evaluator* können Linq-Expressions ausgewertet werden.

6.2.5.1 ObjectViews.Common.ChangeTracker

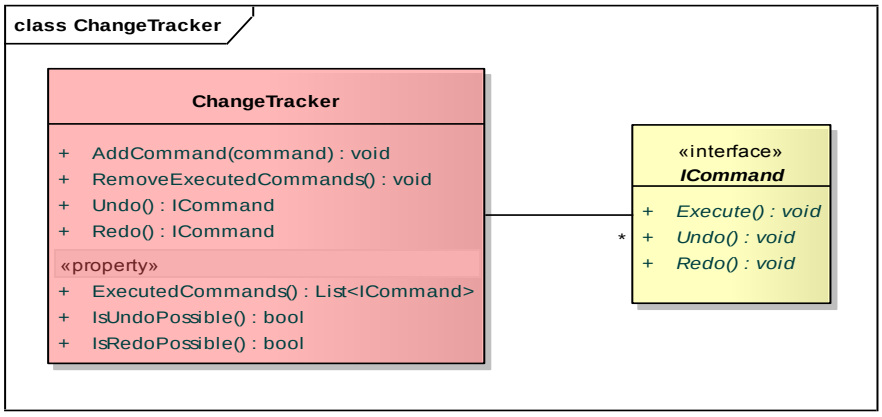


Abbildung 25 - Klassendiagramm ObjectViews.Common.ChangeTracker

Um sämtliche Änderungen in einem Change Tracker aufzuzeigen, wird das Command Konzept angewandt. Jeder Command (abgeleitet von *ICommand*) wird dem *Change Tracker* übergeben und wird dort in einer Liste (*ExecutedCommands*) gespeichert. Jeder Command wird ausgeführt bevor er dem Change Tracker übergeben wird.

6.2.5.2 ObjectViews.Common.IdGenerator

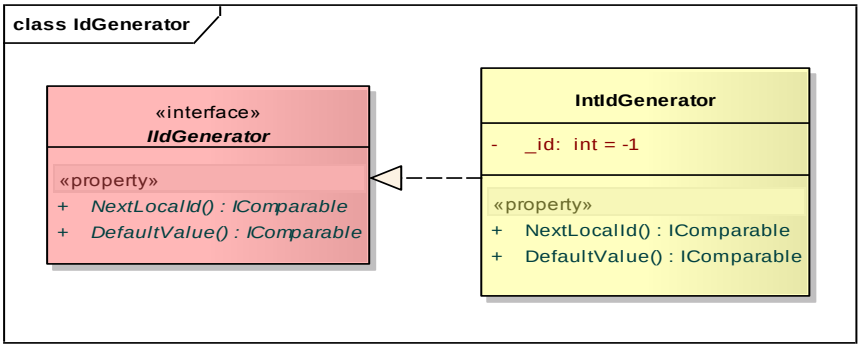


Abbildung 26 - Klassendiagramm ObjectViews.Common.IdGenerator

Sämtliche Datenobjekte werden über eine Id identifiziert. Normalerweise vergibt die Datenbank diese Id, sobald ein Data Object in der Datenbank hinzugefügt wird. Das Framework benötigt jedoch bereits vor der Persistierung in der Datenbank eine eindeutige Id. Für die Generierung solcher lokalen, temporären Ids sind sog. *IdGenerators* zuständig. Im Falle eines Integers werden jeweils negative Ids generiert, welche wiederum eindeutig sind. Somit erhält das erste neue Data Object die Id -1, das nächste die Id -2 usw.

Das Framework enthält bisher nur den *IntIdGenerator*. Es können aber durchwegs auch andere IdGenerators programmiert und verwendet werden und so z.B. mit GUIDs gearbeitet werden.

6.3 Use Case Sicht

6.3.1 Erstellen einer Object View

6.3.1.1 Übersicht

Dieses Kapitel erläutert den Ablauf, welcher zur Erstellung einer Object View (d.h. Erstellen der Object View, Laden der Object View Item, Aufbau der Metadaten, Registrieren der Event Handler) nötig sind.
Es gibt grob den folgenden Ablauf (Nummern entsprechen denjenigen im Sequenzdiagramm):

1. **CreateObjectView**: Aufruf des Frameworks mit Übergabe des entsprechenden Linq-Queries.
2. **Query ausführen**: Linq Query auf der Datenbank ausführen und Datensätze in Collection speichern. Bei dieser Collection werden zudem gleich die Events registriert, wenn ein Data Object hinzugefügt oder gelöscht wird.
3. **Generieren der Metadaten**: Analysieren des Linq Queries und erstellen der Metadaten.
4. **Erstellen der Data Objects**: Das Data Object wird dem lokalen Data Object Cache im Memory hinzugefügt. Die Id und der Typ des Data Objects dient dabei als Identifikation.
5. **Registrieren der Event Handler auf dem ObjectViewField**: Jedem Feld (z.B. dem Feld City) wird mittels dem TypeDescriptor ein EventHandler registriert. Dieser wird aufgerufen sobald sich der Wert des Feldes ändert.
6. **Refresh auf lokalem Context**: Das Query wird nochmals auf dem lokalen Data Context (Data Object Cache) ausgeführt. Dadurch wird sichergestellt, dass die Daten aktuell sind. Wird lokal z.B. die Stadt von Zürich auf Uster geändert, so wird anschliessend in der Applikation die Stadt Uster angezeigt, obwohl in der Datenbank noch die Stadt Zürich steht.
7. **Registrieren der Event Handler**: Hier werden noch zwei Arten von EventHandler registriert:
   - Hinzufügen, Löschen eines in Beziehung stehenden Objektes.
   - Hinzufügen, Verändern, Löschen der Objekte, welche die Selektion einschränken.

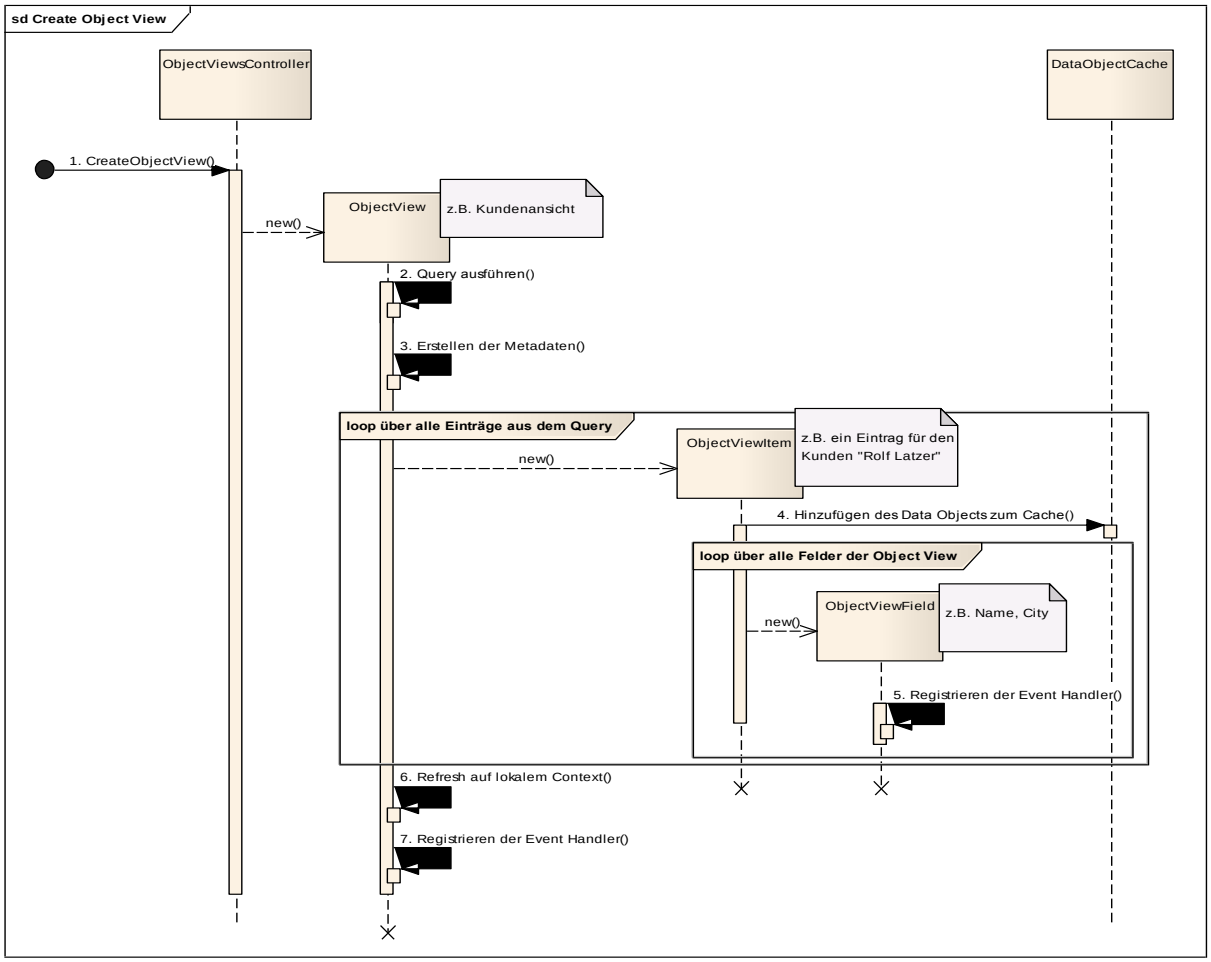


Abbildung 27 - Sequenzdiagramm "Erstellen einer ObjectView"

6.3.1.2 Generieren der Metadaten

Hier wird auf ein Sequenzdiagramm verzichtet. Im Wesentlichen werden drei Parser aufgerufen:

- JoinStatementParser
- WhereStatementParser
- SelectStatementParser

Diese drei Parser erstellen aufgrund der Expression Trees für die entsprechenden Statements die Metadaten.

6.3.1.3 Erstellen der Data Objects

Beim Ausführen einer Query werden die Daten verschiedenster Data Objects zurückgeliefert. Das Object Views Framework erstellt im Data Object Cache eigene Data Objects. Dabei wird für jedes Data Object zwingend eine Id benötigt, um sie zu identifizieren. So kann entschieden werden, ob sich ein Data Object bereits im Cache befindet (dank einer früheren Query), oder ob es ein Neues ist. Daten von bereits vorhandenen Data Objects werden ergänzt, wobei die bereits gesetzten Felder im Cache immer Vorrang haben. D.h., wenn ein Feld bereits gesetzt ist, so wird dieses Feld nicht nochmals gesetzt, weil es in der Applikation bereits verändert worden sein könnte.

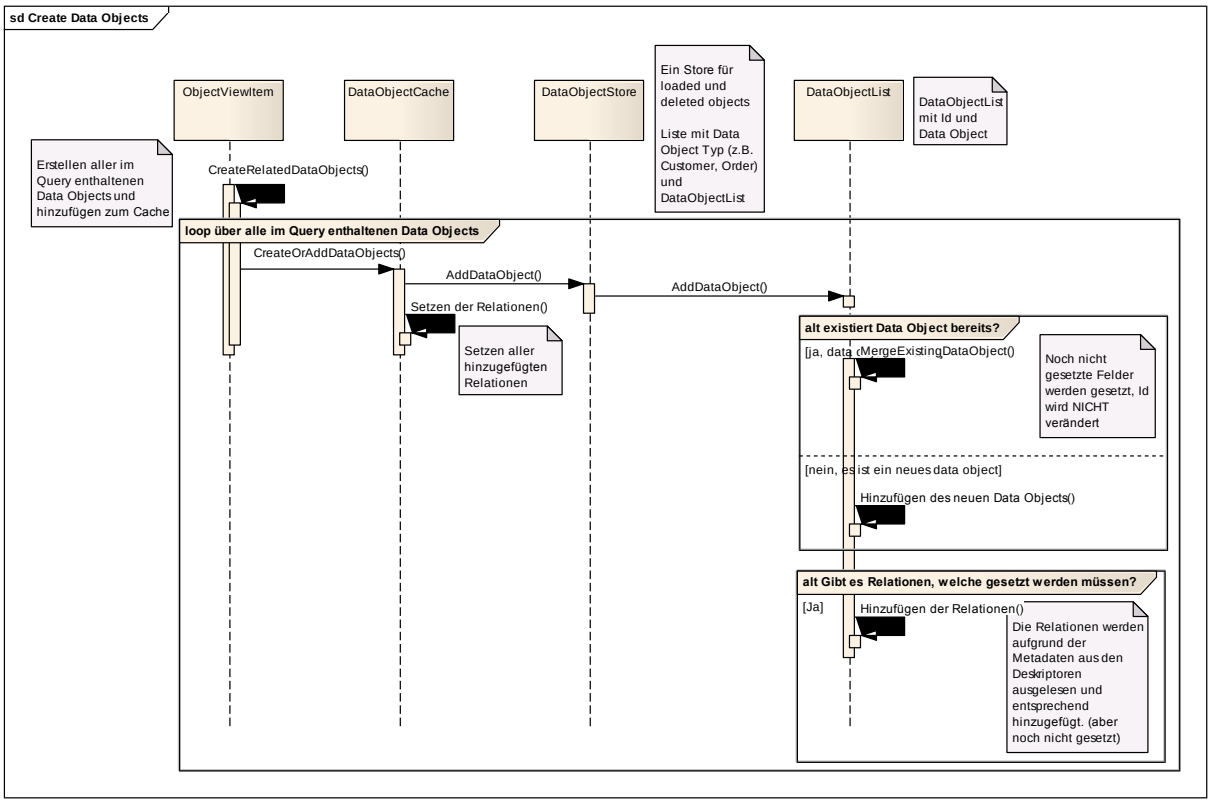


Abbildung 28 - Sequenzdiagramm "Erstellen eines Data Objects"

6.3.2 Reagieren auf Datenänderungen

6.3.2.1 Registrieren der Event Handler

Es gibt zwei Arten von Datenänderungen:

Einfügen und Löschen	Das Einfügen und Löschen wird durch eine Collection vollzogen, welche einen Event wirft sobald ein Objekt hinzugefügt oder gelöscht wurde. Unser Framework registriert sich auf diese Änderungen.
Bearbeiten	Sobald das Feld eines Data Objects bearbeitet wird, wird ein Event geworfen. Diesen Event kann man abfangen, indem man ihn über den TypeDescriptor abonniert. Der Event wird jedoch nur geworfen, wenn das Feld des Data Objects dann auch über den TypeDescriptor gesetzt wird.

6.3.2.2 Ablauf in den Event Handler

Im Sequenzdiagramm wird exemplarisch dargestellt, welche Events geworfen werden, wenn durch den Benutzer einen neuen Eintrag im ObjectView erstellt wird. Beim Löschen eines Eintrages oder beim Bearbeiten eines Feldes ist der Ablauf sehr ähnlich.

Wichtig ist, dass die Events gekapselt geworfen werden (siehe AddPendingEvent() und RaisePendingEvents()). D.h. jede Object View wird immer nur einmal aufgefordert, einen Refresh durchzuführen. Es wird sichergestellt, dass das Target und die Methode nur einmal aufgerufen werden. Das Target ist eine ObjectView Klasse (z.B. CustomerView) und die Methode entspricht der Delegate-Methode (z.B. Refresh).

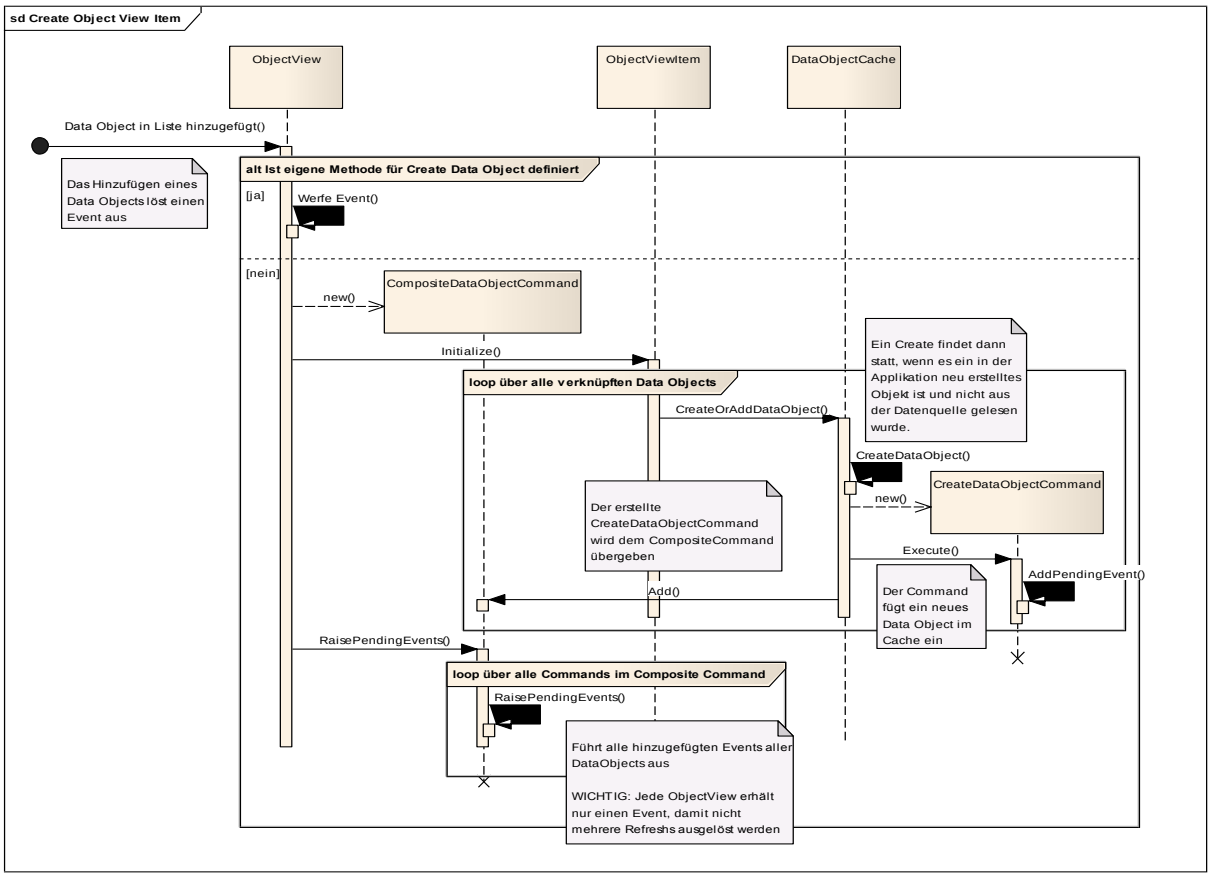


Abbildung 29 - Sequenzdiagramm "Erstellen eines Object View Item"

6.3.3 Rückgängig machen und wiederherstellen

Durch das Konzept mit den Commands ist das Undo und Redo relativ simpel. Es muss der entsprechende Command aus dem ChangeTracker geholt werden und auf diesem die Undo/Redo-Aktion durchgeführt werden.

Wichtig ist, dass durch den CompositeCommand mehrere Commands rückgängig gemacht resp. wiederhergestellt werden können. Werden z.B. beim Hinzufügen einer neuen Zeile in einer Object View mehrere Data Objects erstellt, so sollen diese beim Undo *alle* wieder gelöscht werden.

Der ChangeTracker hält jedoch nicht zwingend nur Commands vom Typ CompositeCommand. Er kann mit beliebigen Commands vom Typ ICommand umgehen.

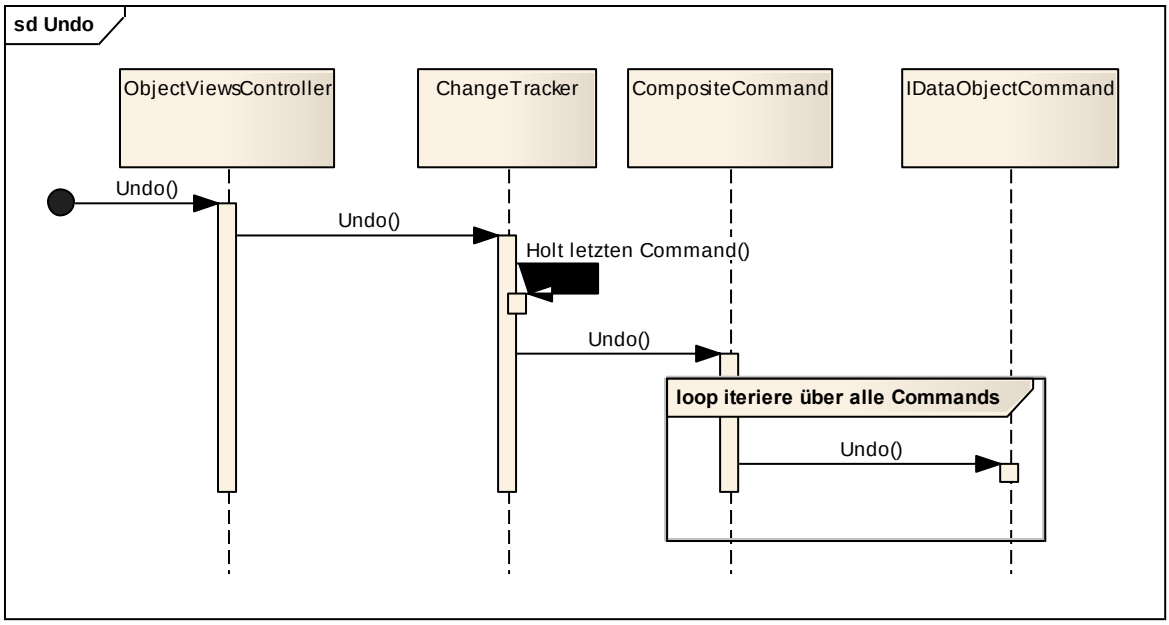


Abbildung 30 - Sequenzdiagramm „Undo“

6.3.4 Datenänderungen persistieren

Sollen Datenänderung in einer Datenquelle, wie beispielsweise einer Datenbank, persistent gemacht werden, muss ein ChangeSet erstellt werden. Das ChangeSet enthält alle DataObjectCommands (Create, Update, Delete) welche die Änderungen repräsentieren, die durch den Benutzer in der Applikation vollzogen wurden.

Der DataContextWrapper (im Sequenzdiagramm wird dies am Beispiel des EntityFrameworkWrappers gezeigt) enthält schliesslich die Logik, wie auf das Persistenz-Framework zugegriffen wird. Der EntityFrameworkWrapper kann durch einen anderen Wrapper ersetzt werden, er muss beim Erstellen des ObjectViewsController angegeben werden.

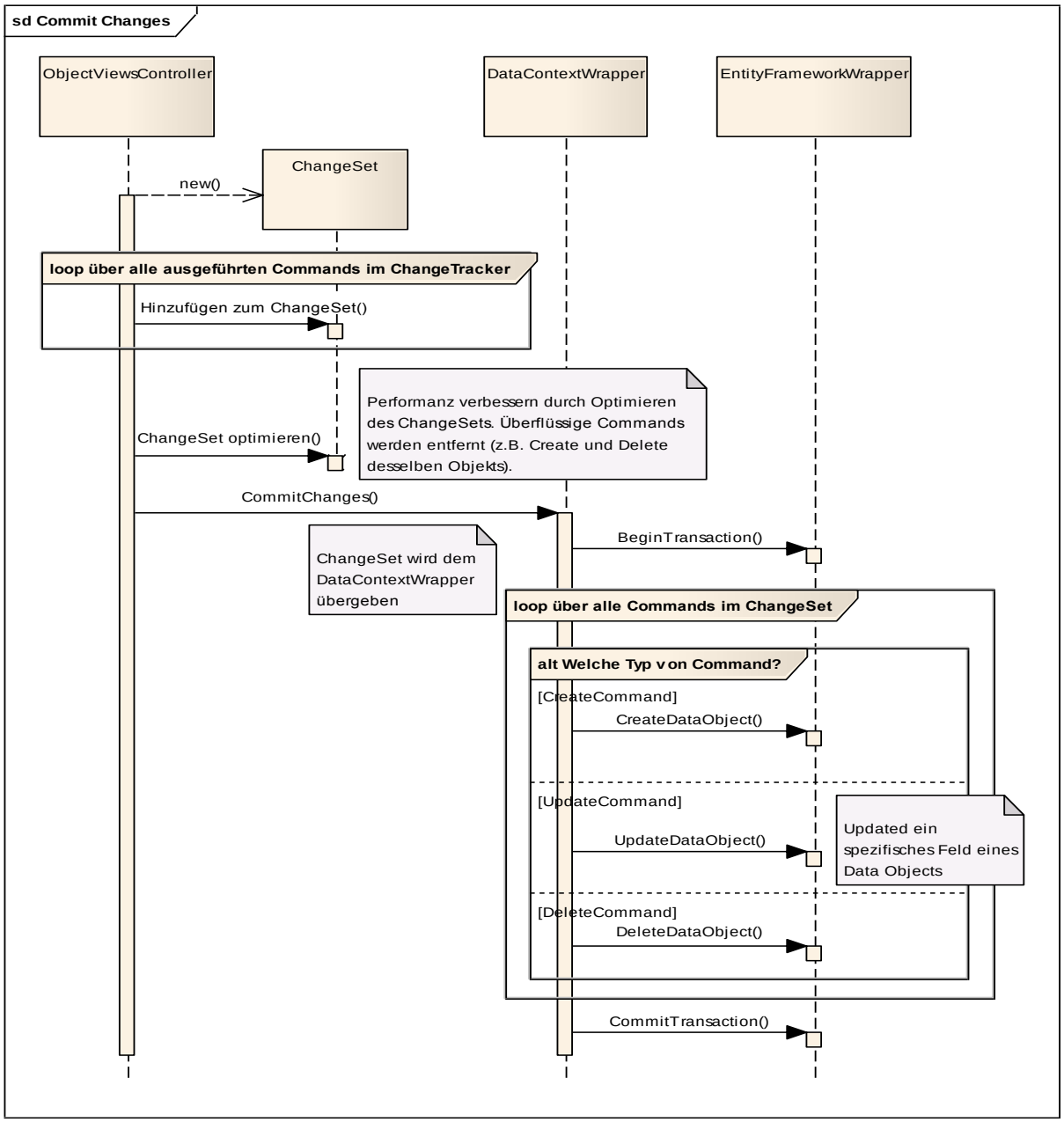


Abbildung 31 - Sequenzdiagramm „Datenänderungen persistieren“

6.4 Verteilungssicht

Das folgende Verteilungsdiagramm zeigt die Verteilung der beteiligten Komponenten für die Three Tier Object Views.

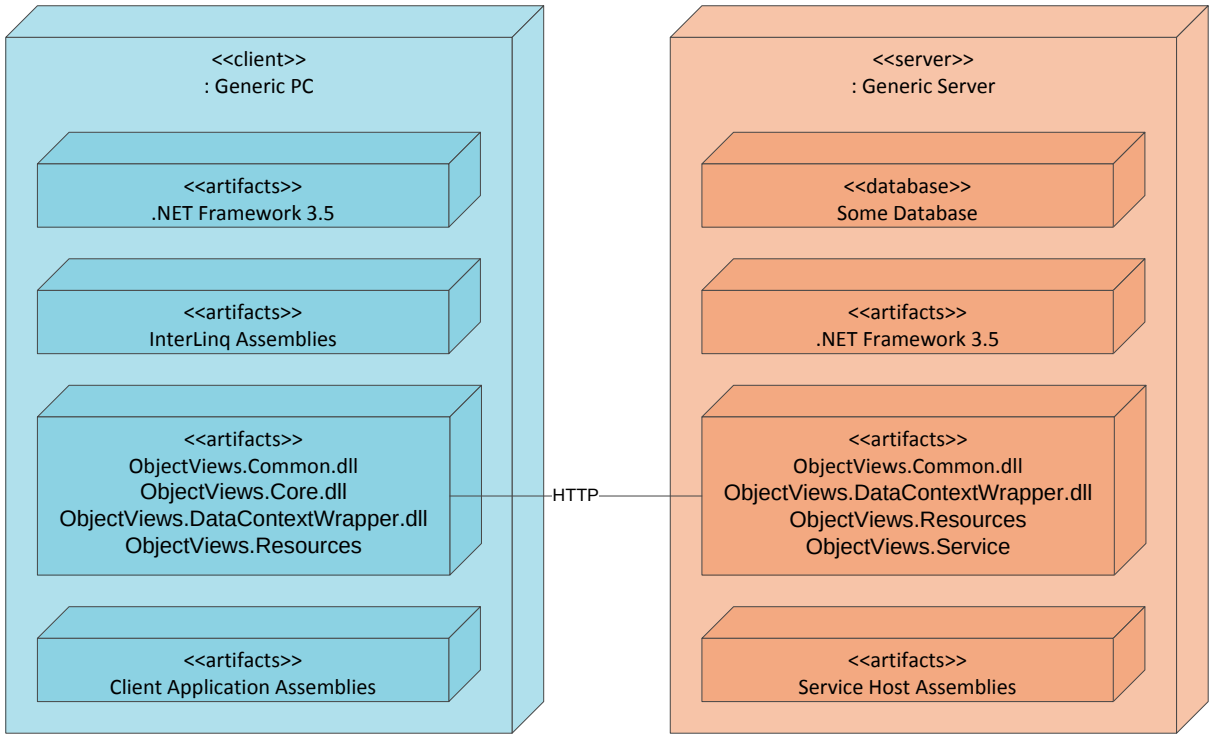


Abbildung 32 - Verteilungsdiagramm

Die Verbindung zwischen dem Client und dem Server wird von den Komponenten ObjectViews.DataContextWrapper.dll (Klasse ServiceContextWrapper) und ObjectViews.Service.dll (Klasse ObjectViewsService) hergestellt.

Vom Verwender des Frameworks entwickelt werden müssen die Client Application Assemblies und die Service Host Assemblies.

7 Resultat

Dieses Kapitel beschreibt das Resultat des Projekts. Es zeigt auf, wie das entwickelte Produkt verwendet werden kann, welche Anforderungen dieses erfüllt und welche Probleme noch bestehen.

7.1 Verwendung

7.1.1 Übersicht

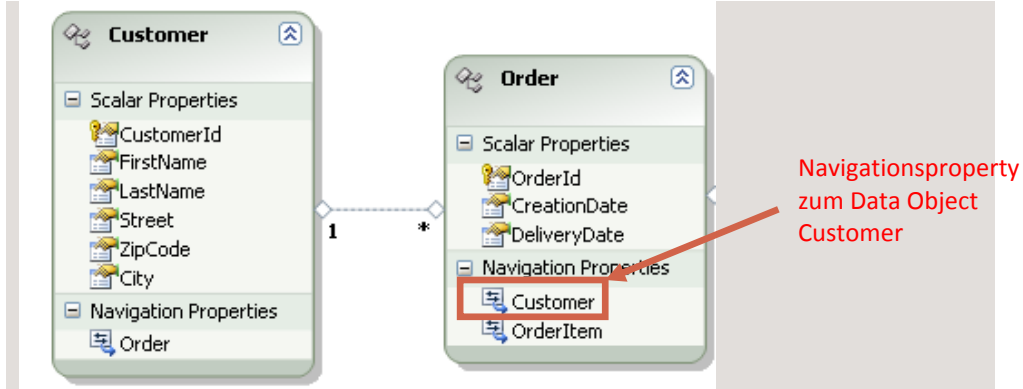
Die Verwendung des Frameworks wird aufgrund eines kompletten Beispiels erläutert:

Beispiel

Bemerkungen zum Beispiel

Das Beispiel verwendet:
- ADO.Net Entity Framework als Persistenzframework.
- Das WPF DataGrid zur Darstellung der Daten.

Klassendiagramm für Data Objects



Erweiterung an den Data Objects

- Mit dem Attribut `[DataObject]` wird das Feld zur Identifikation des Data Objects angegeben.
- Die `[DeletionStrategy]` definiert wie sich das Framework bei Löschen eines Data Objects zu verhalten hat. ChildDataObjectFieldName entspricht dem oben markierten Navigationsproperty.

```
using ObjectViews.Common;
using ObjectViews.Core.DataObject;
using ObjectViews.Core.DataObject.Attributes;

namespace ObjectViews.DemoApplication.DataContext
{
        [DataObject(DataObjectIdFieldName = "CustomerId")]
        [DeletionStrategy(ChildDataObjectType = typeof (Order),
                    ChildDataObjectFieldName = "Customer",
                    DeletionStrategy = DeletionStrategy.CascadeDeletion)]
       public partial class Customer:IDataObject {}
}
```

Erstellen der Object View Item Klasse

- Die Klasse muss zwingend von der Basisklasse `ObjectViewItem` abgeleitet werden.
- Die Klasse enthält ein Property für jedes Feld, das in der View angezeigt werden soll.
- Mit dem `[AllowDeletion]` Attribut wird angegeben, welcher Typ von Data Objects beim Löschen einer Zeile aus der View aus der Datenquelle entfernt werden soll.

```
using ObjectViews.Core.ObjectView;
using ObjectViews.Core.ObjectView.Attributes;
using ObjectViews.DemoApplication.DataContext;

namespace ObjectViews.DemoApplication.DataAccess
{
        [AllowDeletion(DataObjectType = typeof (Customer))]
        public class CustomerView:ObjectViewItem
        {
                public int CustomerId { get; set; }
                public string Name { get; set; }
```

```
                public string FirstName { get; set; }
                public string LastName { get; set; }
                public int ZipCode { get; set; }
                public string City { get; set; }
                public int OrderCount { get; set; }
        }
}
```

Erstellen des Object View Controllers

- Demo Object Views Controller muss ein Data Context Wrapper übergeben werden.
- Im Beispiel wird mit der bereits implementierte EntityFrameworkWrapper verwendet, da der Datenkontext ein Entity Framework Data Context ist.

```
public class BaseController
{
        protected IDataContextWrapper _dataContextWrapper;
        protected ObjectViewsController  objectViewsController;
        protected ObjectViewsEntities  dataContext;

        public BaseController()
        {
                _dataContext = new ObjectViewsEntities();
                 dataContextWrapper =
                        new EntityFrameworkWrapper( dataContext);
                 objectViewsController =
                        new ObjectViewsController(_dataContextWrapper);
        }
}
```

Absetzen eins Queries / Erstellen einer Object View

- Die verwendete und vorher erstellte View Klasse `CustomerView` muss zwingend von `ObjectViewItem` abgeleitet sein. Es können hier nur Views mit dem Basistyp `ObjectViewItem` verwendet werden.
- Für das Linq Query gelten bestimmte Einschränkungen:
  - Das Feld mit der Id des Data Objects muss zwingend abgeholt werden.
  - Sämtliche Beziehungen müssen zwingend über Joins abgebildet werden.
  - Alle Felder von zusammengesetzten/berechneten Feldern müssen einzeln abgeholt werden.

```
objectViewsController.CreateObjectView(
        from customer in  shopDataContext.Customers
        select new CustomerView
        {
                CustomerId = customer.CustomerId,
                Name = customer.FirstName + " " + customer.LastName,
                FirstName = customer.FirstName,
                LastName = customer.LastName,
                ZipCode = customer.ZipCode,
                City = customer.City
        });
```

Zuweisen der Object View zu einer GUI-Komponente

- Die Zuweisung der Daten erfolgt über das Property `ObjectViewItems`.
- Das Property `AutomaticRefreshEnabled` erlaubt das automatische Aktualisieren der Object View. Wird beispielweise eine Zeile aus einer anderen Object View gelöscht, so muss das Query lokal nochmals ausgeführt werden. Bei vielen Daten sollte dieses Property auf `false` gesetzt, da sonst die Applikation zu langsam wird.
- Über `ObjectViewChanged` kann bei Änderungen auf einer Object View eigener Code ausgeführt werden.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
        _objectViewOrderItems =
                        _shopController.CreateOrderItemView(_orderId);
        gridOrderItems.ItemsSource =  objectViewOrderItems.ObjectViewItems;

        _objectViewItems = _shopController.CreateItemView();
        _objectViewItems.AutomaticRefreshEnabled = true;
        _objectViewItems.ObjectViewChanged += ObjectViewChanged;
        dataGridColumnItem.ItemsSource = _objectViewItems.ObjectViewItems;
}

private void ObjectViewChanged(object sender, EventArgs e)
{
```

Zuweisung an WPF DataGrid

	`dataGridColumnItem.ItemsSource = null;`
	`dataGridColumnItem.ItemsSource = _objectViewItems.ObjectViewItems;`
	`}`

Speichern von Benutzer-änderungen	• Mit `CommitChanges()` werden sämtliche Änderungen an den Object Views gespeichert. Es kann nicht nur eine einzelne Object View gespeichert werden, sondern immer gleich sämtliche Änderungen. `objectViewsController.CommitChanges();`

Rückgängig machen und Wiederherstellen	• Rückgängig machen und Wiederherstellen wird direkt vom Framework zur Verfügung gestellt. Es können sämtliche Datenänderungen rückgängig gemacht und wiederhergestellt werden, solange sie nicht persistiert wurden.
	
	`_objectViewsController.UndoChange();`
	bzw.
	`_objectViewsController.RedoChange();`

Schliessen der Object View	• Damit die Object View wieder sauber vom Controller abgehängt werden kann, muss sie geschlossen werden.
	
	`private void Window_Closed(object sender, EventArgs e)`
	`{`
	`_objectView.Close();`
	`}`

7.1.2 Erweiterte Themen

7.1.2.1 Erstellen einer Object View auf Object Views

Das Erstellen einer Object View mit anderen Object Views als Datenquelle ist möglich. Es muss dabei jedoch beachtet werden, dass zwei Data Context Wrapper und somit auch zwei Object Views Controller zum Einsatz kommen. Der erste Data Context Wrapper ist immer vom Typ *CollectionContextWrapper* und übernimmt die Persistierung der Datenänderungen in den Object Views, die als Datenquelle dienen. Der zweite Data Context Wrapper nimmt dann die Persistierung in der eigentlichen Datenquelle (im folgenden Beispiel ein Entity Framework Data Context) vor.

Beispiel	

Code	`// create controller for datasource object views`
	`var efw = new EntityFrameworkWrapper(db);`
	`var objectViewsController1 = new ObjectViewsController(efw);`
	
	`// create datasource object view`
	`var objectView = objectViewsController1.CreateObjectView(from customer in db.Customer select new CustomerView {…});`
	
	`// create controller for object views on object views`
	`var ccw = new CollectionContextWrapper(…);`
	`var objectViewsController2 = new ObjectViewsController(ccw);`
	
	`// create object view on object view`
	`var objectViewOnObjectView = objectViewsController2.CreateObjectView(from customerView in objectView.ObjectViewItems.AsQueryable() select new CustomerFirstNameView {…});`
	
	`// …`
	
	`// commit changes`
	`objectViewsController2.CommitChanges();`
	`objectViewsController1.CommitChanges();`

7.1.2.2 Verwendung eigener Methoden für Einfügen und Löschen

Beim Einfügen und Löschen in eine Object View kann das standardmässige Verhalten überschrieben werden, indem man sich an den folgenden Events anhängt:

- ObjectViewItemCreated
- ObjectViewItemDeleted

Hängt man sich an diesen Event, kann der eigene Code ausgeführt werden und es wird kein Standardverhalten ausgeführt. Soll dieses Standardverhalten nach dem Durchführen der Methode dennoch ausgeführt werden, können die folgenden Methoden aufgerufen werden:

- CreateObjectViewItem(ObjectViewItem)
- DeleteObjectViewItem(ObjectViewItem)

Das Standardverhalten fügt das neu erstellte Data Object dem lokalen Cache hinzu bzw. kennzeichnet es als gelöscht.

Beispiel

Beschreibung

Das Beispiel zeigt, wie eigener Code ausgeführt werden kann und anschliessend das Standardverhalten ausgeführt werden kann

Code

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
        _objectView = _shopController.CreateCustomerView();
        gridCustomers.ItemsSource =  objectView.ObjectViewItems;

        _objectView.ObjectViewItemCreated += _objectView_ObjectViewItemCreated;
}

private void  objectView_ObjectViewItemCreated(object sender,
                                               ObjectViewItemCreatedEventArgs e)
{
    // do something

    // execute default behaviour
      objectView.CreateObjectViewItem(e.ObjectViewItem);
}
```

Überschreibt das Standardverhalten

Standardverhalten muss explizit aufgerufen werden

7.1.2.3 Verwendung von eigenem Data Context Wrapper

Möchte man einen eigenen Data Context Wrapper erstellen, muss man das Interface *IDataContextWrapper* implementieren.

Interface IDataContextWrapper

Code

```
public interface IDataContextWrapper
{
        ChangeSet CommitChanges(ChangeSet changeSet);
}
```

Beschreibung

Die Methode CommitChanges soll sämtliche Änderungen an den Daten in der Datenquelle persistieren. Als Parameter wird ein ChangeSet übergeben, welches eine Liste ist mit drei versteckten Typen (nested Types) in der Klasse ChangeSet:

- ChangeSet.CreateDataObject: Für jedes neu hinzugefügte Data Object
- ChangeSet.UpdateDataObject: Für jedes veränderte Property eines Data Objects
- ChangeSet.DeleteDataObject: Für jedes gelöschte Data Object

Klasse DataContextWrapper

Code

```
public abstract class DataContextWrapper:IDataContextWrapper
{
        protected abstract object BeginTransaction();
        protected abstract ChangeSet CommitTransaction(object transaction);
        protected abstract object GetDataObject(Type dataObjectType,
                                                IComparable dataObjectId);
        protected abstract void CreateDataObject(object dataObject);
        protected abstract void UpdateDataObject(object dataObject,
                                                 string fieldName,
                                                 object oldValue,
                                                 object newValue);
        protected abstract void DeleteDataObject(object dataObject);
}
```

Beschreibung

Es kann von dieser abstrakten Klasse geerbt werden. Dadurch kann der Wrapper vereinfacht implementiert werden, weil dadurch nicht selbst überprüft werden muss, welche Nested Class verwendet wurde, sondern man direkt alle Attribute der Nested Class als Methodenparameter erhält.

7.1.2.4 Verwendung von eigenem Id Generator

Die Id-Generatoren werden benötigt um lokal eindeutige Ids zu generieren. Werden lokal neue Data Objects erstellt, müssen diese ebenfalls eine eindeutige Identifikation haben, bis sie in der Datenquelle persistiert sind und eine entsprechende Datenbank-Id erhalten. Dies ist nötig, da im Object Views Framework alle Data Objects über eine eindeutige Id identifiziert werden.

Zur Zeit ist nur ein solcher Id Generator für Integer-Ids erstellt worden. Möchte man beispielsweise GUIDs verwenden, müsste ein eigener Id Generator implementiert werden.

Beispiel	
Beschreibung	Um den eigenen Id Generator verwenden zu können, müssen zwei Schritte vollzogen werden: • Implementierung des Interfaces *ObjectViews.Common.IdGenerator.IIdGenerator* • Registrieren beim ObjectViewsController

Code für Implementierung des IdGenerators	`public class IntIdGenerator:IIdGenerator`
	`{`
	`  private int _id = -1;`
	
	`  #region IIdGenerator Members`
	`  public IComparable NextLocalId`
	`  {`
	`  get`
	`  {`
	`  lock (this) {`
	`  return id--;`
	`  }`
	`  }`
	`  }`
	
	`  public IComparable DefaultValue`
	`  {`
	`  get { return 0; }`
	`  }`
	`  #endregion`
	`}`

Code für das Registrieren beim Object Views Controller	`_objectViewsController.RegisterDataObjectIdGenerator(`
	`  typeof (int),`
	`  new IntIdGenerator());`

7.1.2.5 Einsatz in einer verteilten Umgebung

Die Übersicht zeigt das Zusammenspiel der einzelnen Komponenten:

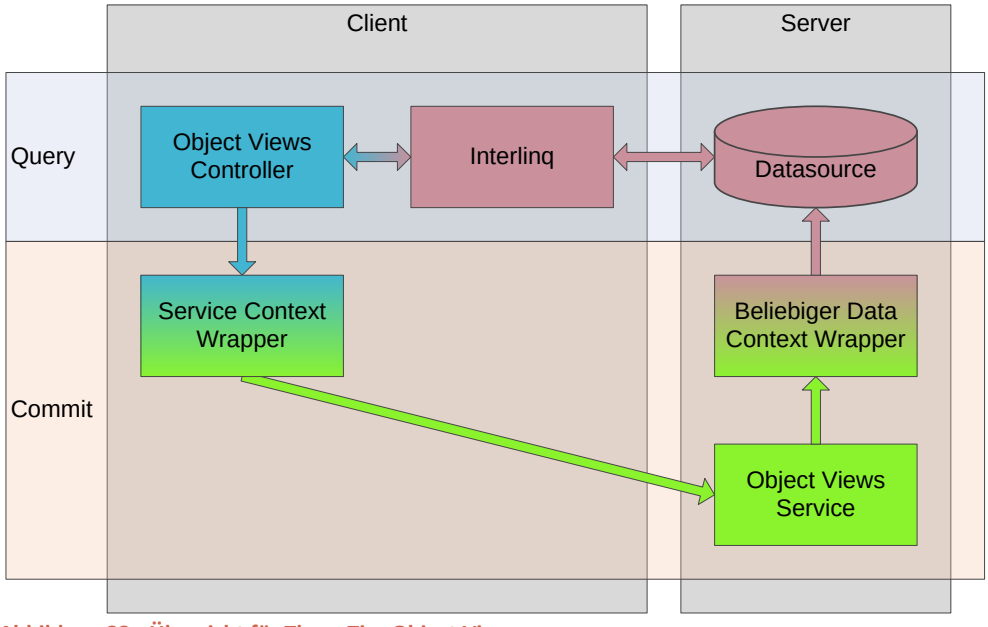


Abbildung 33 - Übersicht für Three Tier Object Views

Beispiel	
Beschreibung	Um in einer verteilten Umgebung Object Views zu nutzen, müssen drei Schritte vollzogen werden: 1. Interlinq Service starten 2. Eigener WCF-Service für das Persistieren der Daten erstellen 3. Erstellter WCF-Service und Interlinq DataContext aus der Client-Applikation nutzen
Interlinq Service starten	Durch Interlinq wird das Linq-Query direkt auf dem Server ausgeführt und so können Daten vom Server gelesen werden. Für weitergehende Informationen wird auf die Projektseite von Interlinq verwiesen [4]. Damit Interlinq in Zusammenhang mit Object Views als Datenquelle verwendet werden kann, muss ein eigener Service geschrieben werden: `public class Program { static void Main() { IQueryHandler queryHandler = new EntityFrameworkQueryHandler(new ObjectViewsEntities()); // Publish the IQueryHandler by InterLINQ over WCF // using the default connection (TCP-Binding, Binary, // tcp://localhost:7890/InterLinqService) Console.WriteLine("Publishing service..."); using (ServerQueryWcfHandler serverQueryHandler = new ServerQueryWcfHandler(queryHandler)) { serverQueryHandler.Start(); Console.WriteLine("Server is started and running."); Console.WriteLine(); // Wait for user input Console.WriteLine("Press [Enter] to quit."); Console.ReadLine(); // Close the service Console.WriteLine("Closing service..."); } Console.WriteLine("Bye"); } }`
Eigener WCF-Service für das Persistieren der Daten erstellen	• Um den Service zu erstellen, muss Adresse und Port angegeben werden, auf welchen der Service gebunden werden soll. • Zudem muss angegeben mit welchem `IDataContextWrapper` die Änderungen gespeichert werden soll. Der Service leitet die erhaltenen Änderungen direkt an diesen Wrapper weiter. `namespace ObjectViews.DemoApplication.Service { internal class Program { private static void Main(string[] args) { new ObjectViewsService().StartService("localhost", 7777, new EntityFrameworkWrapper(new ObjectViewsEntities())); } } }`
Erstellter WCF-Service und Interlinq DataContext aus der Client-Applikation nutzen	• Der Service kann genutzt werden, indem beim Erstellen des `ObjectViewsController` der `ServiceContextWrapper` übergeben wird. Dadurch werden die Änderungen immer an den WCF-Service gesendet und dort persistiert. • Interlinq kann verwendet werden, indem der Data Context auf Interlinq umgestellt wird. `public BaseController() { ClientQueryWcfHandler queryHandler = new ClientQueryWcfHandler(); queryHandler.Connect(); _shopDataContext = new InterLinqDataContext(queryHandler); _dataContextWrapper = new ServiceContextWrapper("localhost", 7777); objectViewsController = new ObjectViewsController(dataContextWrapper); }`

7.2 Erfüllung der Anforderungen

Diese Übersicht zeigt, welche Anforderungen der Anforderungsspezifikation erreicht wurden.

7.2.1 Funktionale Anforderungen

7.2.1.1 Readonly Object Views

Id	Anforderung	Erfüllt	Bemerkungen
1	Projektion	Ja	-
2	Selektion	Ja	-
3	Verbund	Ja	Der Verbund muss jedoch immer zwingend mit einem Join-Operator definiert werden.
4	Zusammengesetze Werte	Ja	Es müssen jedoch die einzelnen Werte des zusammengesetzten Teils ebenfalls aus der Datenquelle geladen werden.
5	Aggregierte Werte	Ja	Funktioniert, sofern die benötigten Aggregatsobjekte ebenfalls im lokalen Cache zugänglich sind.
6	Aktualisierung	Ja	-
7	Typisierte Object Views	Ja	-
8	Untypisierte Object Views	-	Konnte nach Entscheid von H. Huser weggelassen werden.
9	Single Tier Object Views	Ja	-
10	Two Tier Object Views	Ja	-

7.2.1.2 Updateable Object Views

Id	Anforderung	Erfüllt	Bemerkungen
11	Einfügen	Ja	-
12	Bearbeiten	Ja	-
13	Löschen	Ja	-
14	Undo / Redo	Ja	Beliebig oft möglich Undo/Redo durchzuführen.
15	Updateable Collections	Ja	-
16	Updateable Entity Framework Data Context	Ja	-

7.2.1.3 Three Tier Object Views

Id	Anforderung	Erfüllt	Bemerkungen
17	Lesen von Remote Data Context	Ja	Dies wurde erreicht unter Verwendung von Interlinq.
18	Schreiben in Remote Data Context (Einfügen, Bearbeiten, Löschen)	Ja	Erreicht mittels eines eigenen WCF-Services.
19	Übernehmen von Änderungen aus Remote Data Context	Nein	Das Übernehmen der im entfernten Datenkontext vorgenommenen Änderungen wurde im Object Views Framework vorbereitet. Das Konzept sieht vor, dass die Data Context Wrapper nach dem Commit eigener Änderungen die Änderungen der anderen Clients zurückgeben. Diese Funktionalität ist in den bereitgestellten Wrappern jedoch noch nicht implementiert.

7.2.2 Nichtfunktionale Anforderungen

7.2.2.1 Bedienbarkeit

Id	Anforderung	Erfüllt	Bemerkungen
20	Einfache Verwendung des Frameworks	Ja	Die Verwendung des Object Views Frameworks ist sehr einfach und intuitiv, da die Schnittstellen sehr minimal gehalten wurden.
21	Einfache Einbindung des Frameworks	Ja	Das Object Views Framework ist ohne Konfigurieraufwand einzusetzen. Man muss einzig die Assemblies einbinden.

7.2.2.2 Zuverlässigkeit

Id	Anforderung	Erfüllt	Bemerkungen
22	Hundertprozentige Verfügbarkeit	Nein	Es gibt Queries, mit denen das Framework nicht umgehen kann. Die meisten Fälle solcher Queries werden vom Query Validator erkannt. Andere führen leider noch zu einer Exception. Es war uns nicht möglich, sämtliche möglichen Fälle zu testen.
23	Absturzfreier Ablauf	Nein	Wie erwähnt, können spezielle Queries zu Exceptions führen. In den meisten Fällen muss die Query dann einfach umformuliert werden.

7.2.2.3 Leistung

Id	Anforderung	Erfüllt	Bemerkungen
24	Minimale Datenübertragung	Ja	Das Object Views Framework überträgt immer nur die Daten, die mit einer Query angefordert werden. Zur Aktualisierung der Views wird die Query im lokalen Cache durchgeführt, wodurch keine Datenübertragung mehr stattfindet.
25	Minimale Aktualisierung	Ja	In den Object Views wird immer nur das nötigste aktualisiert. Sogar Aggregatsfelder können sich einzeln aktualisieren, ohne Einbezug der gesamten Query.
26	Minimaler Performanzverlust	Ja	Ein Unit Test bestätigt das Erreichen dieses Ziels. Eine Query ist weniger als 10% langsamer, wenn sie mit dem Object Views Framework durchgeführt wird.
27	Meistern von mindestens 100‘000 Datensätzen	Ja	Das Object Views Framework kann im Prinzip mit beliebig vielen Datensätzen umgehen, solange der Arbeitsspeicher reicht.
28	Meistern einer beliebigen Anzahl Abfragen	Ja	Ist gewährleistet, solange der Arbeitsspeicher reicht.

7.2.2.4 Wartbarkeit

Id	Anforderung	Erfüllt	Bemerkungen
29	Gute Wartbarkeit	Ja	Das Framework ist gut strukturiert und dokumentiert.

7.2.2.5 Schnittstellen

Id	Anforderung	Erfüllt	Bemerkungen
30	Keine Anforderungen an Datenobjekte	Nein	Die Klassen der Datenobjekte müssen erweitert werden. Sie müssen einerseits ein Marker Interface implementieren und andererseit attributiert werden, damit das Framework weiss, welches das Id Feld der Klasse ist.
31	Unterstützung aller Linq kompatiblen Datenquellen	Ja	-

7.3 Offene Probleme

Problem 1	Löschen im Object View
Beschreibung	Bei einer Object View kann attributiert werden, welche Data Objects gelöscht werden sollen, wenn eine Zeile in der Object View gelöscht wird. Wenn nun aber *nichts* gelöscht werden soll, dann verschwindet die Zeile trotzdem aus der Object View. Der Anwendungprogrammierer muss in solchen Fällen selbst dafür sorgen, dass das Löschen nicht möglich ist.
Lösungsvorschlag	Momentan wird eine Exception geworfen, wenn in einer Object View gelöscht werden kann, dabei aber keine Data Objects gelöscht werden. Wenn anderes Verhalten gewünscht ist, so kann dies an der Stelle im Programmcode, die mit „Problem 1“ gekennzeichnet ist, implementiert werden.

Problem 2	Mehrere Verschachtelungen im Query
Beschreibung	Bei den Joins, Wheres und aggregierten Feldern machen mehrstufige Verschachtelungen wie *orderItem.Order.Customer* Probleme. Dieses Problem stellt sich beim Aufbauen der Metadaten. Es kann nicht mehr aufgelöst werden, dass sowohl eine Beziehung von orderItem zu Order als auch von Order zu Customer besteht.
Lösungsvorschlag	Dieses Problem kann einfach umgangen werden, indem auf solche Syntax in den Queries verzichtet wird. Anstelle von Verschachtelungen können immer auch Joins verwendet werden. Somit ist das Problem gelöst.

Problem 3	Komplexe Where-Statements
Beschreibung	Es können keine Where Statements mit AND und/oder OR geparst werden. Das Problem liegt hier bei der Herstellung der Beziehungen zwischen den Data Objects: Beispiel: `where customer.City == „Zürich“ || customer.City == „Uster“ || customer.CustomerId == 5` In einer solchen Ansicht kann keine Aussage mehr gemacht werden, ob eine neue hinzugefügte Bestellung wirklich zum Kunden mit der Id 5 gehört oder nicht doch zu einem Kunden welcher in Zürich oder Uster wohnt.
Lösungsvorschlag	Das Problem kann durch Erweitern des Object Views Frameworks gelöst werden. Die entsprechenden Stellen sind im Programmcode mit „Problem 3“ gekennzeichnet.

Problem 4	Aggregation über mehrere Hierarchiestufen
Beschreibung	Eine Aggregation über mehrere Hierarchiestufen macht Probleme, da kein Bezug zwischen den Parent- und Child-Data-Objects hergestellt werden kann. Beispiel: `order.OrderItem.Sum(orderItem => (double)(orderItem.Quantity*orderItem.Item.PriceCHF))` Wird nun der Preis des Items geändert, dann wird das aggregierte Feld nicht aktualisiert, da der Zusammenhang zwischen den beteiligten Items und der Order View (wo sich ja das aggregierte Feld befindet) nicht bekannt ist.
Lösungsvorschlag	Das Problem kann durch Erweitern des Object Views Frameworks gelöst werden. Die entsprechenden Stellen sind im Programmcode mit „Problem 4“ gekennzeichnet.

Problem 5	Beziehungen ändern
Beschreibung	Wird bei einem bestehenden OrderItem das Item von Item A auf Item B geändert, dann gibt das massiv Probleme für unser Framework. Unser Framework versucht dann, die Id des Items A zu ändern (und zwar auf den Wert der Id von Item B), was natürlich zu Problemen führt.

Lösungsvorschlag	Wenn die Query korrekt formuliert wird, dann können Beziehungen geändert werden.
	
	Beispiel für eine Query, die ein Ändern der Beziehung zulässt:
	`from orderItem in db.Orders`
	`join item in db.Items on orderItem.Item equals item`
	`select new OrderItemView {`
	`OrderItemId = orderItem.OrderItemId,`
	`ItemId = orderItem.ItemId,` `…});`
	
	Beispiel für eine Query, die ein Ändern der Beziehung *nicht* zulässt:
	`from orderItem in db.Orders`
	`join item in db.Items on orderItem.Item equals item`
	`select new OrderItemView {`
	`OrderItemId = orderItem.OrderItemId,`
	`ItemId = item.ItemId,` `…});`
	
	Mit dem Entity Framework ist es momentan noch nicht möglich, die Query korrekt zu formulieren. Mit der nächsten Version (.NET Framework 4.0) wird dies jedoch möglich sein.

Problem 6	Nicht erfüllte Constraints
Beschreibung	Fügt man in der Customer View eine neue Zeile ein, dann wird im lokalen Cache ein neues Customer Objekt erstellt, die Constraints für dieses Objekt werden jedoch (noch) nicht geprüft. Das kann zu Problemen führen. Wenn man z.B. eine Query absetzt mit LastName = customer.LastName.ToUpper() und der LastName ist null (was gegen die Constraints ist), dann gibt es beim Ausführen der Query eine Exception.
Lösungsvorschlag	Das Problem kann durch Erweitern des Object Views Frameworks gelöst werden. Die entsprechenden Stellen sind im Programmcode mit „Problem 6“ gekennzeichnet.

Problem 7	Konfliktsituationen
Beschreibung	Wie wird bei Konfliksituationen in der Datenbank reagiert? Im Moment wird nur *eine* Lösungsstrategie angeboten (Erster gewinnt). Bearbeiten also zwei Clients gleichzeitig dieselben Daten, dann wird dessen Änderung übernommen, der sie zuerst persistiert.
Lösungsvorschlag	Das Problem kann durch Erweitern des Object Views Frameworks gelöst werden. Die entsprechenden Stellen sind im Programmcode mit „Problem 7“ gekennzeichnet.

Problem 8	Selektion ohne konstanten Wert

Beschreibung	Wird eine Selektion erstellt ohne konstante Werte, so findet keine automatische Aktualisierung statt.
	
	Beispiel: Es werden alle Kunden mit der selben Stadt angezeigt:
	`from cust1 in _shopDataContext.Customers`
	`join cust2 in _shopDataContext.Customers`
	`  on customer1.City equals customer2.City`
	`where !cust1.CustomerId.Equals(cust2.CustomerId)`
	`select new CustomerSameCityView { … }`
	
	Nun findet keine automatische Aktualisierung statt, wenn beispielsweise die Stadt des Kunden A auf dieselbe Stadt des Kunden B geändert wird.

Lösungsvorschlag	Eine mögliche Lösung wäre, bei *jeder* Veränderung des betroffenen Data Object Feldes (im Beispiel das Feld „City“) eine Aktualisierung der Object View durchzuführen. Betrachtet man jedoch auch den Performanz-Aspekt, so ist schnell ersichtlich, dass mit vielen Daten dies nur schwer performant gehandhabt werden kann.

Problem 9	Linq Methode Skip & Take
Beschreibung	Die Methode Skip(5) überspringt die ersten 5 Datensätze und Take(7) nimmt die folgenden 7 Datensätzen aus dem Kontext. Dies stellt für das Framework ein Problem dar, sobald das Query lokal ausgeführt wird. Dann werden ebenfalls die ersten 5 Datensätze übersprungen, was natürlich nicht der Sinn und Zweck ist.
Lösungsvorschlag	Lösungsvorschlag 1: Wird auf den Data Context zugegriffen, wird Skip und Take weggelassen und erst auf dem lokalen Data Object Cache wieder verwendet. Dadurch werden alle Datensätze zwar aus der Datenbank in den lokalen Cache geladen, jedoch in der Object View nur die gewünschten Einträge angezeigt. Lösungsvorschlag 2: Skip und Take werden nur auf den Data Context angewandt und nicht auf den lokalen Data Object Cache. Dadurch kann ein Paging realisiert werden, indem nur die angeforderten Datensätze aus der Datenquelle geladen werden. In der Anwender-Applikation werden dann jedoch alle geladenen Datensätze angezeigt. Sind beispielsweise zuvor bereits einige Datensätze geladen worden, werden diese Datensätze ebenfalls in der GUI-Applikation angezeigt.

8 Literaturverzeichnis

Nummer	Verweis
1	Martin Fowler Patterns of Enterprise Application Architecture. Verlag Addison-Wesley. 3. Auflage, März 2003.
2	SourceMaking Dokumentation der wichtigsten Design Patterns. http://sourcemaking.com/design_patterns Abruf am 03.06.2009.
3	MSDN How to: Implement an Expression Tree Visitor. http://msdn.microsoft.com/en-us/library/bb882521.aspx Abruf am 21.02.2009
4	Manuel Bauer, Pascal Schäfer InterLinq Dokumentation. http://www.codeplex.com/interlinq Abruf am 10.06.2009.

9 Glossar

Begriff	Beschreibung
Change Tracker	Ein Change Tracker zeichnet sämtliche Benutzeränderungen auf. Im Zusammenhang mit dieser Arbeit wurde der ChangeTracker verwendet um die Datenänderungen eines Endanwenders aufzuzeichnen.
Data Context Wrapper	Ein Data Context Wrapper persistiert sämtliche Datenänderungen in einer Datenquelle.
Data Object	Ein Data Object wiederspiegelt das Objekt einer Entität (normalerweise einer Tabelle in einer Datenbank).
Data Object Cache	Ein Data Object Cache hält geladene Data Objects, damit sie beim nächsten Zugriff sofort zur Verfügung stehen und nicht erneut aus der Datenquelle geladen werden müssen.
Data Object Graph (DOG)	Ein Data Object Graph besteht aus Data Objects und ihren Beziehungen untereinander. Ein Beispiel für einen solchen Graphen ist z.B. ein Kunde mitsamt seinen Bestellungen.
Deleted Data Object Store	Eine Untermenge des Data Object Cache. Dieser Store hält die gelöschten Data Objects. Diese müssen noch zur Verfügung stehen, damit sie beim Commit ebenfalls aus der Datenquelle gelöscht werden können, dürfen aber in den Object Views nicht mehr erscheinen.
Expression Tree	Jede Linq Query kann als Expression Tree abgebildet werden. Ein Expression Tree stellt eine baumartige Struktur der Aufrufe dar, welche in Linq formuliert werden. Das Framework nutzt den Expression Tree, um die Metadaten aus den Linq Queries zu erstellen.
Framework	Ein Framework ist eine Bibliothek von Softwarekomponenten, welche die Entwicklung von Programmen erleichtern soll.
Id Controller	Der Id Controller übernimmt Aufgaben rund um die Ids der Data Objects.
Id Generator	Ein Id Generator generiert eindeutige lokale Ids. Diese Id wird benötigt, damit ein neu erstelltes Data Object im Data Object Cache identifiziert werden kann, bevor es in der Datenbank gespeichert wird.
Linq Query	Eine Linq Query ist eine Datenabfrage, formuliert in der Linq Syntax.
Loaded Data Object Store	Eine Untermenge des Data Object Cache. Dieser Store hält alle geladenen Data Objects. Sämtliche Linq Queries, welche lokal ausgeführt werden, werden auf diesem Store ausgeführt.
Object View	Die Object View ist das Pendant zur Datenbank-View. Diese Object View wird im Gegensatz zur Datenbank-View jedoch direkt im Programmcode definiert.
Object View Item	Eine Zeile innerhalb einer Object View wird als Object View Item bezeichnet.
Object Views Controller	Der Object Views Controller ist das Kernstück des Object Views Frameworks. Über diesen werden neue Object Views erstellt, Änderungen persistiert, sowie Undo und Redo durchgeführt, etc.
Readonly Object Views	Mit Readonly Object Views sind Views gemeint, welche sich untereinander aktualisieren, aber noch keine Datenänderungen in einer Datenquelle persistieren können. Es wird also nur die lokale Datenkonsistenz gewährleistet.
Three Tier Object Views	Mittels Three Tier Object Views ist es möglich, Object Views ebenfalls in einer verteilten Umgebung einzusetzen.
Updateable Object Views	Die Updateable Object Views erlauben es, Datenänderungen in einer Datenquelle zu persistieren.

10 Anhang

10.1 Projektplan

10.1.1 Projektorganisation

10.1.1.1 Projektteam

Person	Email	Rolle	Aufgaben
Rolf Latzer	rlatzer@hsr.ch	Projektleitung	Qualitätscontrolling, Sitzungsprotokolle
Benjamin Egli	begli@hsr.ch	Leiter Entwicklung	Projektcontrolling

10.1.1.2 Betreuer

Person	Email	Rolle
Hansjörg Huser	hhuser@hsr.ch	Projektbetreuer
Stefan Zettel		Externer Experte
Alain Schneble	a1schneb@hsr.ch	Ansprechperson bei technischen Fragen

10.1.2 Zeitmanagement

10.1.2.1 Zeitplan

Das Projekt dauert 14 Wochen à 20 Stunden plus 2 Wochen à 45 Stunden Arbeitsaufwand pro Person. Insgesamt ergibt dies 740 Stunden, die auf die einzelnen Arbeitspakete verteilt werden können.

Vom 9.4.2009 bis 15.4.2009 sind die Frühlingsferien, die nicht als Arbeitszeit angerechnet werden.

Der *detaillierte* Zeitplan wird in Excel erfasst. Es wird dort zu Beginn jeder Iteration eingetragen, wer welche Arbeiten zu erledigen hat. Am Ende der Iteration werden die Ist-Arbeitszeiten aus der Zeiterfassung ebenfalls in diesem Zeitplan eingetragen, was einen Soll/Ist-Vergleich ermöglicht. Ausgewertet werden die Daten in einem Projektcockpit, welches dem Betreuerstab zeigen soll, welche Arbeiten angegangen wurden und wie der aktuelle Stand ist.

Das folgende Gantt-Diagramm gibt einen *groben* Überblick, wann welche Arbeiten getätigt werden:

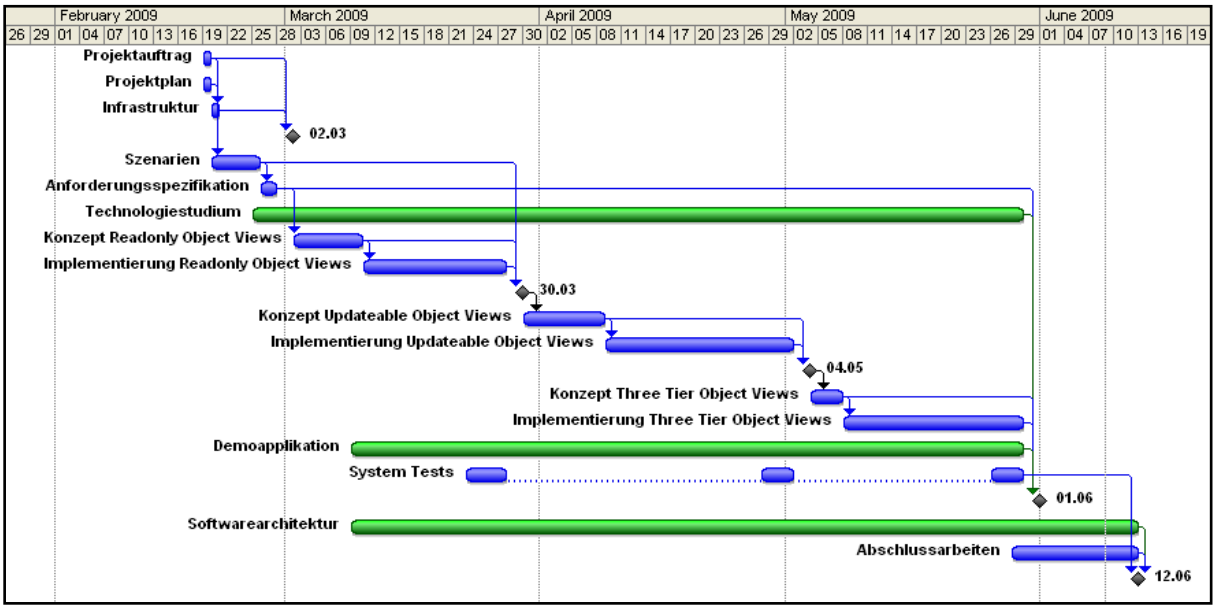


Abbildung 34 - Zeitplan

10.1.2.2 Iterationen

Phase	Iteration	Von – Bis
Inception	1. Iteration	16.02.2009 – 01.03.2009
Elaboration & Contruction	2. Iteration	02.03.2009 – 15.03.2009
	3. Iteration	16.03.2009 – 29.03.2009
	4. Iteration (inkl. Ferien)	30.03.2009 – 19.04.2009
	5. Iteration	20.04.2009 – 03.05.2009
	6. Iteration	04.05.2009 – 17.05.2009
	7. Iteration	18.05.2009 – 31.05.2009
Transition	8. Iteration	01.06.2009 – 12.06.2009

10.1.2.3 Meilensteine

Meilenstein 1	**Ende Vorbereitung**
Datum	02.03.2009
Beschreibung	Projektauftrag vom Auftraggeber abgenommen, Projektplan erstellt, Arbeitsplätze vorbereitet.
Abgeschlossene Arbeitspakete	Projektauftrag, Projektplan, Infrastruktur

Meilenstein 2	**Fertigstellung Readonly Object Views**
Datum	30.03.2009
Beschreibung	Protoypen eines Frameworks für Object Views *ohne* Propagation der Datenänderungen in einer *Single oder Two Tier Umgebung* fertiggestellt, inkl. Demoapplikation.
Abgeschlossene Arbeitspakete	Konzept Readonly Object Views, Implementierung Readonly Object Views

Meilenstein 3	**Fertigstellung Updateable Object Views**
Datum	04.05.2009
Beschreibung	Protoypen eines Frameworks für Object Views *mit* Propagation der Datenänderungen in einer *Single oder Two Tier Umgebung* fertiggestellt, inkl. Demoapplikation.
Abgeschlossene Arbeitspakete	Konzept Updateable Object Views, Implementierung Updateable Object Views

Meilenstein 4	**Fertigstellung Three Tier Object Views**
Datum	01.06.2009
Beschreibung	Protoypen eines Frameworks für Object Views *mit* Propagation der Datenänderungen in einer *Three Tier Umgebung* fertiggestellt, inkl. Demoapplikation.
Abgeschlossene Arbeitspakete	Szenarien, Anforderungsspezifikation, Technologiestudium, Konzept Three Tier Object Views, Implementierung Three Tier Object Views, Demoapplikation, Unit Tests, Code Reviews

Meilenstein 5	**Projektende**
Datum	12.06.2009
Beschreibung	Framework für Object Views fertiggestellt, inkl. Demoapplikation und Dokumentation, Projekt abgeschlossen.
Abgeschlossene Arbeitspakete	Softwarearchitektur, System Tests, Qualitätscontrolling, Projektcontrolling, Projektdokumentation, Abschlussarbeiten, Teamsitzungen, Sitzungen mit Betreuer

10.1.3 Risikomanagement

Risiko Bewertungen								
Risk ID	Risiko	Auswirkung	Massnahme	Kosten der Massnahmen in Stunden	Max. Schaden in Stunden	Wahrschein-lichkeit des Eintreffens	Gewichteter Schaden in Stunden	Priorität
R01	Einarbeitung in die neuen Technologien (Entity Framework, WPF, ...) erweist sich als sehr aufwendig	Die Arbeit mit den Technologien führt zur Verzögerung des gesamten Projektes.	Technologiestudium einplanen, grosse Probleme mit dem Betreuer besprechen	80	200	60%	120	Mittel
R02	Das erstellte Framework erweist sich als zu komplex	Das Framework ist nur mit sehr viel Konfigurations-aufwand nutzbar.	Regelmässige Sitzungen mit Betreuer, eigener Prototyp soll Einfachheit demonstrieren	32	150	30%	45	Hoch
R03	Der Aufwand der Anforderungen ist schwierig abzuschätzen	Es können nicht alle Anforderungen implementiert werden	Anforderungen priorisieren, zuerst nur die wichtigsten Anforderungen implementieren, ev. Meilenstein "MS04 - Three Tier Object Views" weglassen	10	100	20%	20	Mittel
R04	Das Framework ist zu fehleranfällig.	Das Framework wird nicht genutzt, da sonst die Applikation Fehler enthält	Viele Unit-Tests erstellten. Zudem mit Systemtests die Qualität weiter erhöhen	50	200	10%	20	Hoch
R05	Das Framework kann mit grossen Datenmengen nicht umgehen.	Es kann nur in Projekten mit wenigen Daten eingesetzt werden.	Lasttests durchführen, Framework testen mit sehr viel Daten.	40	150	70%	105	Hoch
R06	Änderungen können nur schwer durchgeführt werden	Das Framework ist nicht erweiterbar und kann bei spezieller Nutzung nicht verwendet werden.	Spezielle Schnittstellen zur Verfügung stellen, in Design Erweiterbarkeit berücksichtigen	20	300	20%	60	Tief
	Total Kosten in Arbeitspaketen enthalten			232				
	Total Rückstellungen				1'100		370	

10.1.4 Arbeitspakete

10.1.4.1 Tabellenerklärung

(1)	(2)	(3)
Beschreibung	(4)	
Abhängigkeit	(5)	

1) Name des Arbeitspakets.
2) Verantwortlichkeit: Dieses Feld zeigt, wer die Hauptverantwortung für dieses Arbeitspaket besitzt. Das heisst nicht, dass derjenige das Arbeitspaket alleine umsetzt, sondern nur, dass diese Person die Umsetzung kontrolliert. Es arbeiten grundsätzlich alle im Team an allen Arbeitspaketen mit.
3) Geplante Aufwandsschätzung.
4) Kurze Beschreibung des Arbeitspaketes.
5) In diesem Feld werden allfällige Abhängigkeiten zu anderen Arbeitspaketen festgehalten.

10.1.4.2 Projektmanagement

Projektauftrag	Latzer	8h
Beschreibung	Verfassen des Projektauftrags.	
Abhängigkeiten		

Projektplan	Egli	8h
Beschreibung	Verfassen des Projektplans.	
Abhängigkeiten		

10.1.4.3 Anforderungen

Szenarien	Latzer	32h
Beschreibung	Erarbeitung und Dokumentation aller möglichen Szenarien für den Einsatz von Object Views.	
Abhängigkeiten	Projektauftrag	

Anforderungsspezifikation	Latzer	16h
Beschreibung	Definieren der funktionalen und nichtfunktionalen Anforderungen.	
Abhängigkeiten	Szenarien	

10.1.4.4 Analyse

Technologiestudium	Latzer	32h
Beschreibung	Bestehende Implementation und Technologien analysieren und studieren.	
Abhängigkeiten		

Konzept Readonly Object Views	Latzer	32h
Beschreibung	Erarbeitung und Dokumentation des Konzepts für die Readonly Object Views.	
Abhängigkeiten	Anforderungsspezifikation	

Konzept Updateable Object Views	Latzer	32h
Beschreibung	Erarbeitung und Dokumentation des Konzepts für die Updateable Object Views.	
Abhängigkeiten	Konzept Readonly Object Views	

Konzept Three Tier Object Views	Latzer	32h
Beschreibung	Erarbeitung und Dokumentation des Konzepts für die Three Tier Object Views.	
Abhängigkeiten	Konzept Updateable Object Views	

10.1.4.5 Design

Softwarearchitektur	Egli	32h
Beschreibung	Erarbeitung und Dokumentation der Softwarearchitektur.	
Abhängigkeiten	Konzept Readonly Object Views	

10.1.4.6 Implementierung

Implementierung Readonly Object Views	Egli	72h
Beschreibung	Prototyping und Programmierung der Readonly Object Views.	
Abhängigkeiten	Konzept Readonly Object Views	

Implementierung Updateable Object Views	Egli	56h
Beschreibung	Prototyping und Programmierung der Updateable Object Views.	
Abhängigkeiten	Konzept Updateable Object Views	

Implementierung Three Tier Object Views	Egli	56h
Beschreibung	Prototyping und Programmierung der Object Views in einer Three Tier Umgebung.	
Abhängigkeiten	Konzept Three Tier Object Views	

Demoapplikation	Egli	32h
Beschreibung	Programmierung einer Demoapplikation.	
Abhängigkeiten		

10.1.4.7 Qualitätsmanagement

Unit Tests	Egli	64h
Beschreibung	Schreiben und Durchführen von Unit Tests.	
Abhängigkeiten		

System Tests	Latzer	16h
Beschreibung	Definieren und Durchführen von Systemtests aufgrund der Anforderungsspezifikation.	
Abhängigkeiten	Implementierung Readonly Object Views	

Code Reviews	Egli	32h
Beschreibung	Gegenlesen von Code anderer Projektbeteiligter.	
Abhängigkeiten		

Qualitätscontrolling	Latzer	8h
Beschreibung	Kontrolle der Qualitätsmassnahmen und Einleiten von Gegenmassnahmen.	
Abhängigkeiten		

Projektcontrolling	Latzer	8h
Beschreibung	Kontrolle der Stand der Arbeiten und Einleiten von Gegenmassnahmen.	
Abhängigkeiten		

10.1.4.8 Dokumentation

Projektdokumentation	Egli	8h
Beschreibung	Erstellen von Vorlagen, Protokollen, Zeitplan, etc.	
Abhängigkeiten		

Abschlussarbeiten	Latzer	92h
Beschreibung	Verfassen des Hauptdokuments, Management Summary, Zusammenführen der einzelnen Dokumente, Binden, etc.	
Abhängigkeiten		

10.1.4.9 Sitzungen

Teamsitzungen	Egli	32h
Beschreibung	Besprechungen innerhalb des Teams.	
Abhängigkeiten		

Sitzungen mit Betreuer	Latzer	32h
Beschreibung	Besprechungen mit dem Betreuer.	
Abhängigkeiten		

10.1.4.10 Infrastruktur

Infrastruktur	Latzer	8h
Beschreibung	Einrichten der Arbeitsplätze, Softwareinstallation, etc.	
Abhängigkeiten		

10.1.5 Infrastruktur

10.1.5.1 Hardware

Für das Projekt wird hauptsächlich mit unseren privaten Notebooks gearbeitet. Falls diese ausfallen, könnte auf die PCs in den Labors zurückgegriffen werden.

10.1.5.2 Software

Software	Hersteller	Version	Verwendungszweck	Quelle	Lizenzen
SQL Server Express	Microsoft	2008	Datenbanksystem	www.microsoft.com	Gratislizenz
Visual Studio	Microsoft	2008	Entwicklungsumgebung	ELMS-Server	Schulungslizenz
.NET Framework	Microsoft	3.5 SP1	Laufzeit Umgebung	www.microsoft.com	-
BugTracking	Nerves	1.0	Fehlerverwaltung	clientsdev.nerves.ch	Tool eines Kollegen
Enterprise Architect	Sparx Systems	7	Modellierung	Auf PCs der HSR installiert	-
Tortoise SVN	Open Source	1.5.3	Versionenmanagement	tortoisesvn.tigris.org	Open Source
Toolkit	Microsoft	Januar 2009	WPF Controls (DataGrid)	http://www.codeplex.com/wpf/Release/ProjectReleases.aspx?ReleaseId=22567	Open Source

10.1.5.3 Räumlichkeiten

Es werden hauptsächlich die im Labor zugeteilten Arbeitsplätze verwendet. Ansonsten wird von zuhause aus gearbeitet.

10.1.6 Qualitätsmanagement

10.1.6.1 Tests

10.1.6.1.1 Unit Tests

Für das Testen der C#-Programmteile wird die im Visual Studio integrierte Unit-Test-Umgebung verwendet. Diese Tests werden während dem Fertigstellen jedes Programmstücks durchgeführt. Der Entwickler des jeweiligen Programmteils ist verantwortlich für die Durchführung des Tests. Ein Codestück darf erst im SVN eingecheckt werden wenn der Unit-Test erfolgreich ausgeführt worden ist.

10.1.6.1.2 System Tests

Ab der ersten lauffähigen Version werden Systemtests durchgeführt. In Testspezifikationen wird der Testablauf festgelegt und protokolliert. Die Systemtests werden in diesem Projekt auf den Prototyp gemacht, welches die Verwendung des Frameworks demonstriert.

10.1.6.2 Code Reviews

Das Gegenlesen von Code anderer Teammitglieder wird vom Leiter Entwicklung koordiniert, darf aber auch spontan und unangekündigt vorgenommen werden. Bei jedem Code Review wird ein Protokoll erstellt, das vom Autor des Codes angesehen und abgearbeitet werden muss.

10.1.6.3 Qualitätscontrolling

Beim Qualitätscontrolling wird regelmässig kontrolliert, ob die angeordneten Qualitätsmassnahmen (z.B. Tests, Code Reviews) auch wirklich durchgeführt werden, ob die Arbeit den definierten Vorgaben (z.B. Coderichtlinien) entspricht und ob die definierten Anforderungen erfüllt werden. Bei Abweichungen zu den Vorgaben sind Gegenmassnahmen anzuordnen.

10.1.6.4 Projektcontrolling

Beim Projektcontrolling wird regelmässig überprüft, ob das Projekt im Zeitplan liegt, ob die Arbeitspakete korrekt abgearbeitet werden, ob alle Abgaben rechtzeitig getätigt werden und ob die bei einem Meilenstein definierten Ziele erreicht werden. Wenn Probleme gefunden werden, müssen Massnahmen getroffen werden. Konkrete Massnahmen sind zum Beispiel die Einschränkung der Funktionalität des Produktes oder die Erhöhung der Arbeitszeit.

10.1.6.5 Sitzungen

10.1.6.5.1 Teamsitzungen

Es findet wöchentlich eine Teamsitzung statt, bei der Ergebnisse präsentiert werden und geprüft wird, ob die gesetzten Ziele erreicht worden sind. Dabei wird auch der Arbeitsplan für die nächste Woche gemacht. Teamsitzungen werden bei Bedarf natürlich auch spontan abgehalten.

10.1.6.5.2 Sitzungen mit Betreuer

Eine Sitzung mit dem Betreuer findet wöchentlich am Dienstag von 13:00 – 14:00 Uhr statt. Dabei werden die Resultate gezeigt und das weitere Vorgehen besprochen.

10.1.6.6 Vorgaben / Standards

10.1.6.6.1 Codierrichtlinien

Die Codierrichtlinien werden vom Leiter Entwicklung festgelegt und in einem Dokument festgehalten. Alle Programmierer haben sich selbstständig an diese Richtlinien zu halten.

10.1.6.6.2 Vorlagen

Alle Dokumente basieren auf einer Vorlage. Diese Vorlage wird verwendet, damit ein einheitliches Gesamtbild der Projektdokumentation entsteht.

10.1.6.7 Bug-Tracking

Gefundene Fehler werden im Bug-Tracking-System von Nerves erfasst. Der Fehler wird beschrieben und einem Verantwortlichen zugewiesen.

10.1.6.8 Backup

Auf dem SVN-Server von Nerves wird von den Projektdaten täglich ein Backup durchgeführt. Somit ist die stetige Verfügbarkeit sichergestellt.

10.2 Technologiestudium

10.2.1 Expression Trees

10.2.1.1 Einführung

Mit dem Expression Tree kann eine Linq Query bzw. eine Lambda Funktion auseinandergenommen werden kann.

Durch den Expression Tree ist ersichtlich wie .NET eine Funktion auswertet.
Zur Veranschaulichung wird folgendes Beispiel genommen:

```
Expression<Func<int, bool>> filter = n => (n * 3) < 5;
```

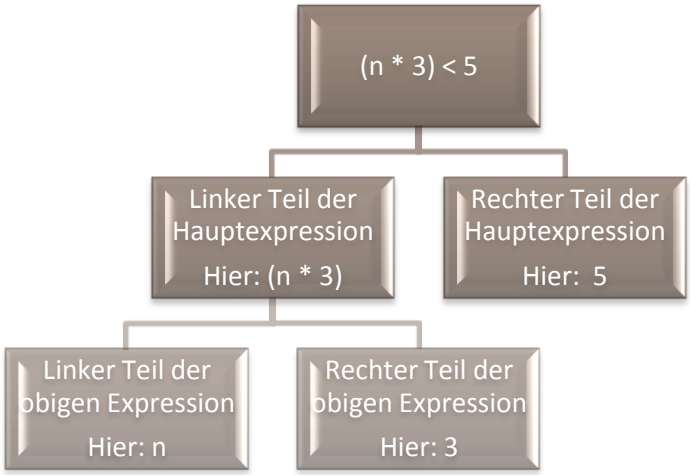


Abbildung 35 - Expression Tree

Beispielcode um den Expression Tree auseinander zu nehmen:

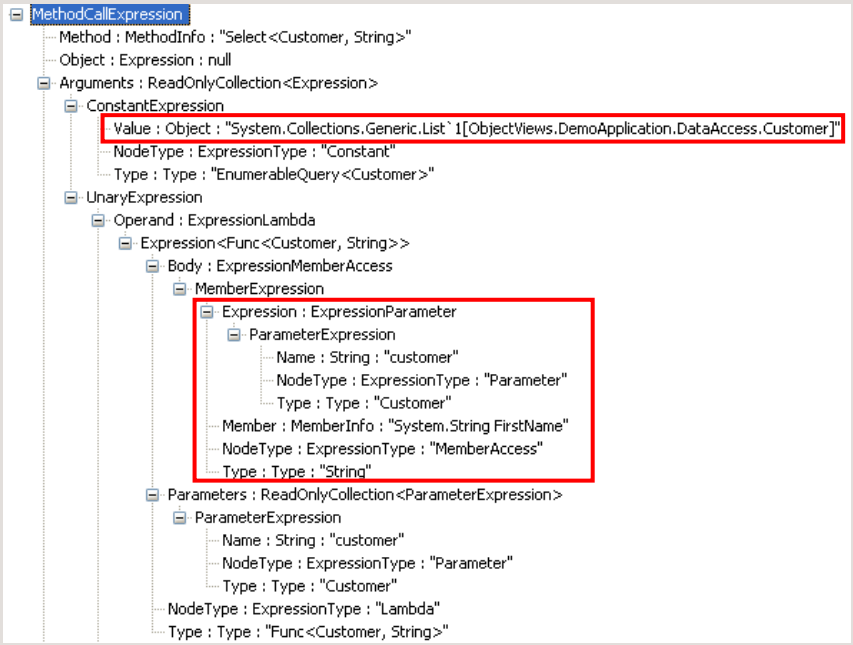
```
BinaryExpression lessThan = (BinaryExpression)filter.Body;
ConstantExpression right = (ConstantExpression)lessThan.Right;
BinaryExpression multiply = (BinaryExpression)lessThan.Left;
ParameterExpression param = filter.Parameters[0];

Func<int, bool> f = filter.Compile();

Console.WriteLine("f(1) = {0}", f(1));
Console.WriteLine("f(2) = {0}", f(2));
```

Um den Expression Tree mit Linq zu nutzen, muss die Datenquelle die Schnittstelle IQueryable implementieren. Der Compiler wird dadurch veranlasst, einen Expression Tree zu erstellen. Auf diesen kann mittels query.Expression zugegriffen werden.

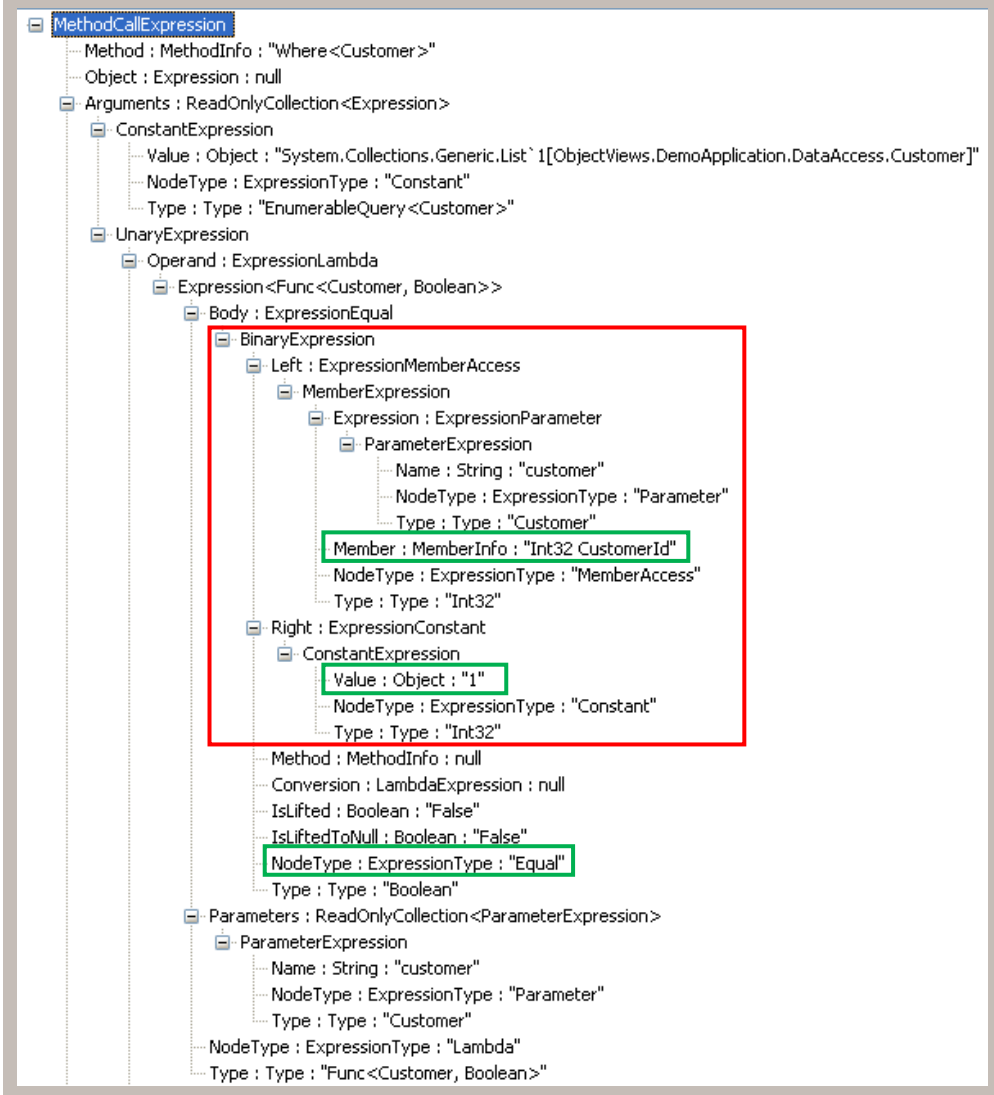
10.2.1.2 Projektion

Einleitung	Nachfolgende wird der Aufbau eines Expression Trees anhand einer Linq Query erklärt, welcher eine Projektion macht.
Beispiel	Linq Query um sämtliche Kunden aus der Datenbank zu holen
Query	`var query = from customer in _dataContext.CustomerList` `select customer.FirstName;`
Expression Tree	
Erläuterungen	**Method Call Expression** Die Method Call Expression bildet einen Methodenaufruf ab, d.h. hier wird der Methodenaufruf der Linq Query abgebildet. **Constant Expression** Repräsentiert einen konstanten Wert. Als konstante Werte sowohl Datenquellen (bzw. die Listen der Datenquellen) betrachtet als auch Werte, die beispielsweise in der Where Klausel gesetzt wurden. Value: Eine Constant Expression hat zudem das Property **Value**. Dieses Property erlaubt den Zugriff auf die zugrunde liegenden Daten (z.B. Zugriff auf die Liste mit den Kundendaten). Beispiel: Das oben gezeigte Beispiel hat als Datenquelle eine Kundenliste. Dies wird im Expression Tree als Constant Expression abgebildet. **Member Expression** Eine Member Expression bildet den Zugriff auf ein Property einer Klasse ab. Beispiel: Im Beispiel ist der Zugriff auf das Property FirstName der Data Object Klasse Customer als Member Expression im Baum.

10.2.1.3 Selektion

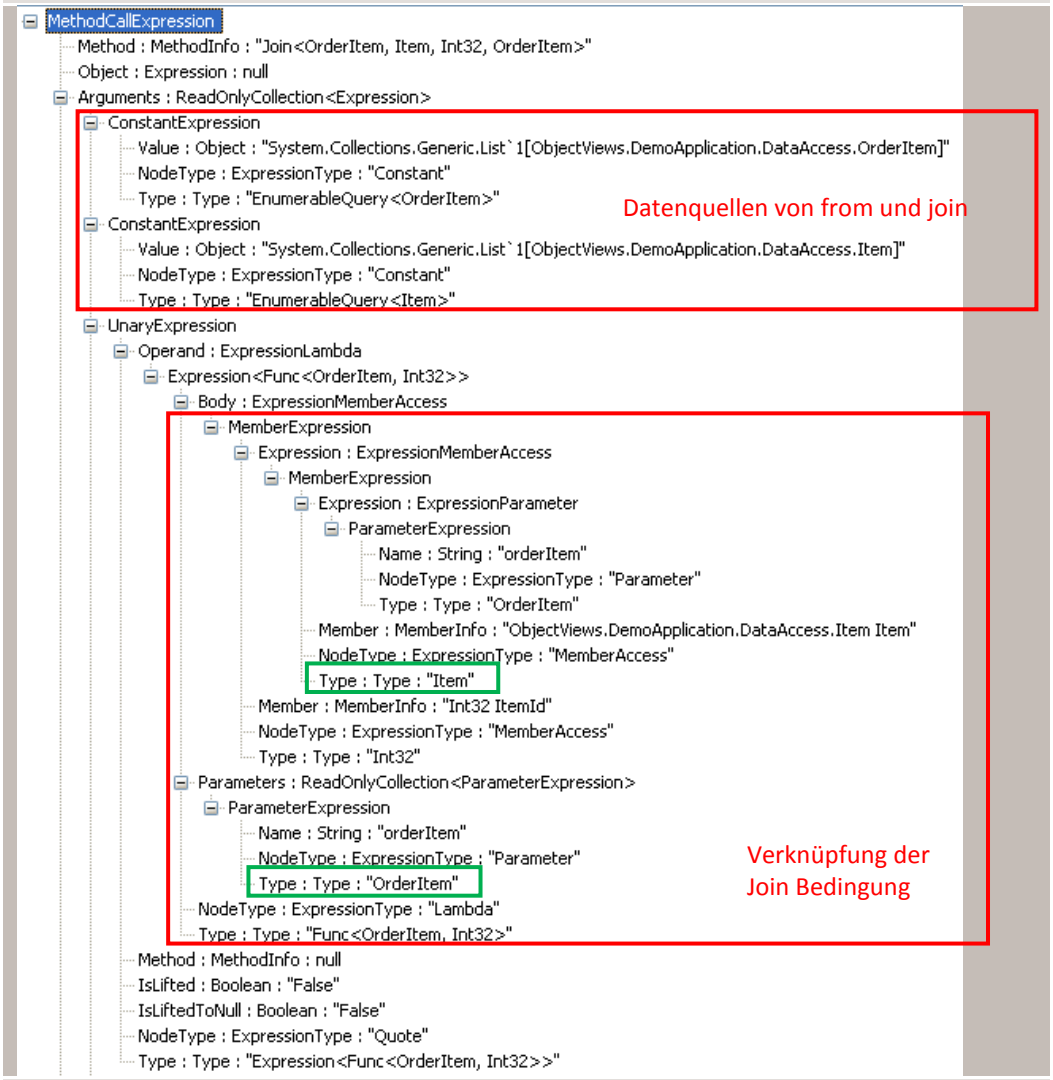
Einleitung	Um eine Selektion zu realisieren wird genau wie in SQL ein where Statement verwendet.

Query	`var query = from customer in _dataContext.CustomerList`
	`where customer.CustomerId == 1`
	`select customer;`

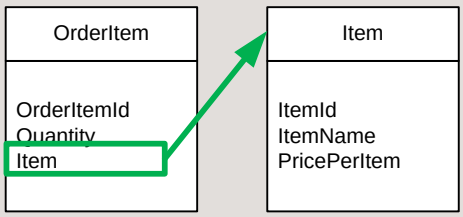
Expression Tree	
Erläuterungen	**Binary Expression** Die Binary Expression bildet das WHERE Statement ab. Diese muss natürlich True liefern, damit der Datensatz in der Selektion berücksichtigt wird. Die wichtigsten Punkte in einer Binary Expression sind: • **Left**: Linker Teil des Where Statements (hier der Verweis auf das Property CustomerId von der Data Class Customer. • **Right**: Rechter Teil des Where Statements (hier: der Wert 1) • **NodeType**: Welcher binäre Vergleich soll gemacht werden? (hier: auf Gleichheit überprüfen → also Equal)

10.2.1.4 Verbund

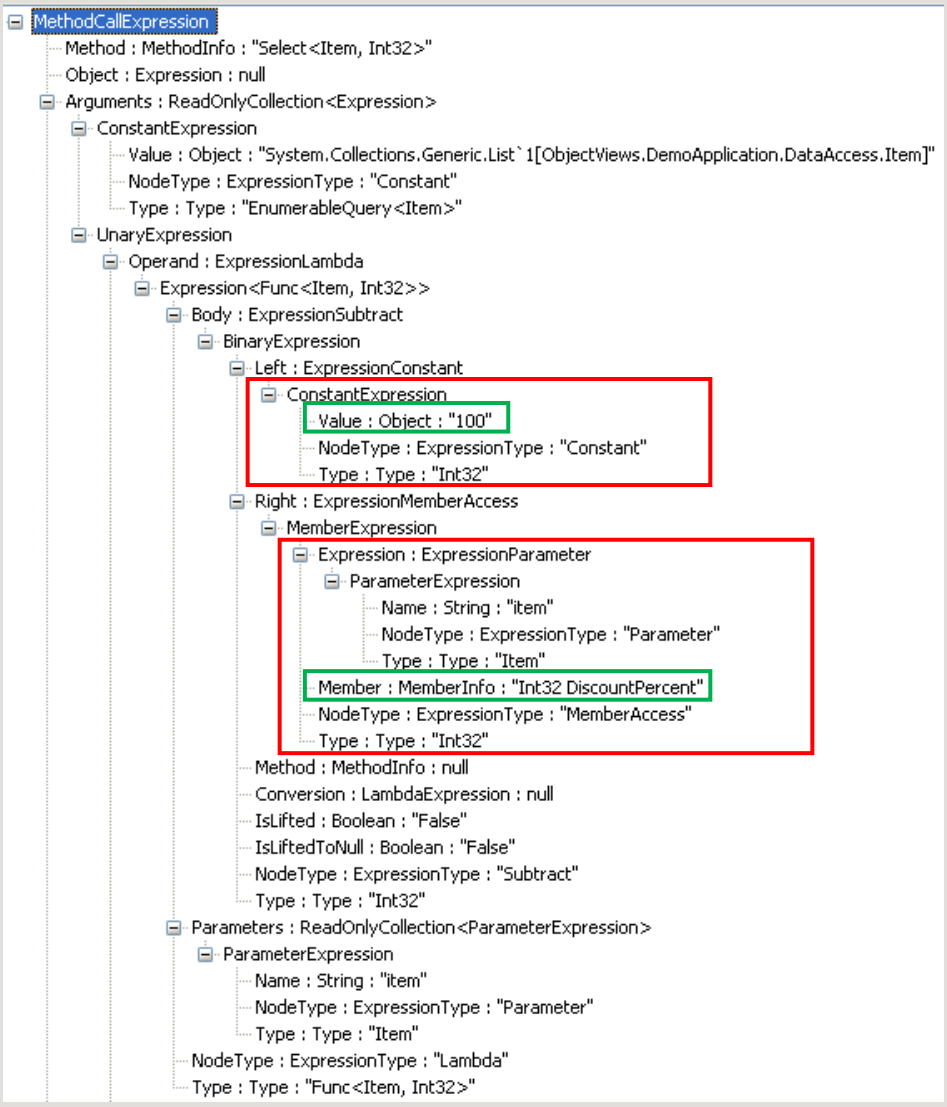
10.2.1.4.1 Verbund ohne DB-Constraints

Einleitung	Um Daten aus mehreren Data Objects abzufragen, muss eine Join Abfrage formuliert werden. Wiederum ist die Syntax ähnlich wie in SQL.
Query	`var query = from orderItem in _dataContext.OrderItemList join item in _dataContext.ItemList on orderItem.Item.ItemId equals item.ItemId select orderItem;`
Expression Tree	
Erläuterungen	**Constant Expression** Eine Join Abfrage ist an mehreren Orten im Expression Tree verteilt: Datenquelle: Die Datenquelle des Join Operators wird gleich zu Beginn über eine Constant Expression angegeben. **Unary Expression** Verknüpfung: Die Bedingung unter welcher die Daten verknüpft werden, wird über eine Unary Expression definiert. Es besagt über welche Attribute die Verknüpfung stattfindet. *Beispiel:* `on orderItem.Item.ItemId equals item.ItemId` Die Daten werden über die ItemId verknüpft. Diese müssen nicht zwingend dieselben Member-Namen besitzen. Dies ist im gezeigten Beispiel nur der Fall weil die Daten bereits auf Datenbankebene verknüpft wurden und der Designer die bereits automatisch umgesetzt hat (anstelle der Spalte mit dem Fremdschlüssel)

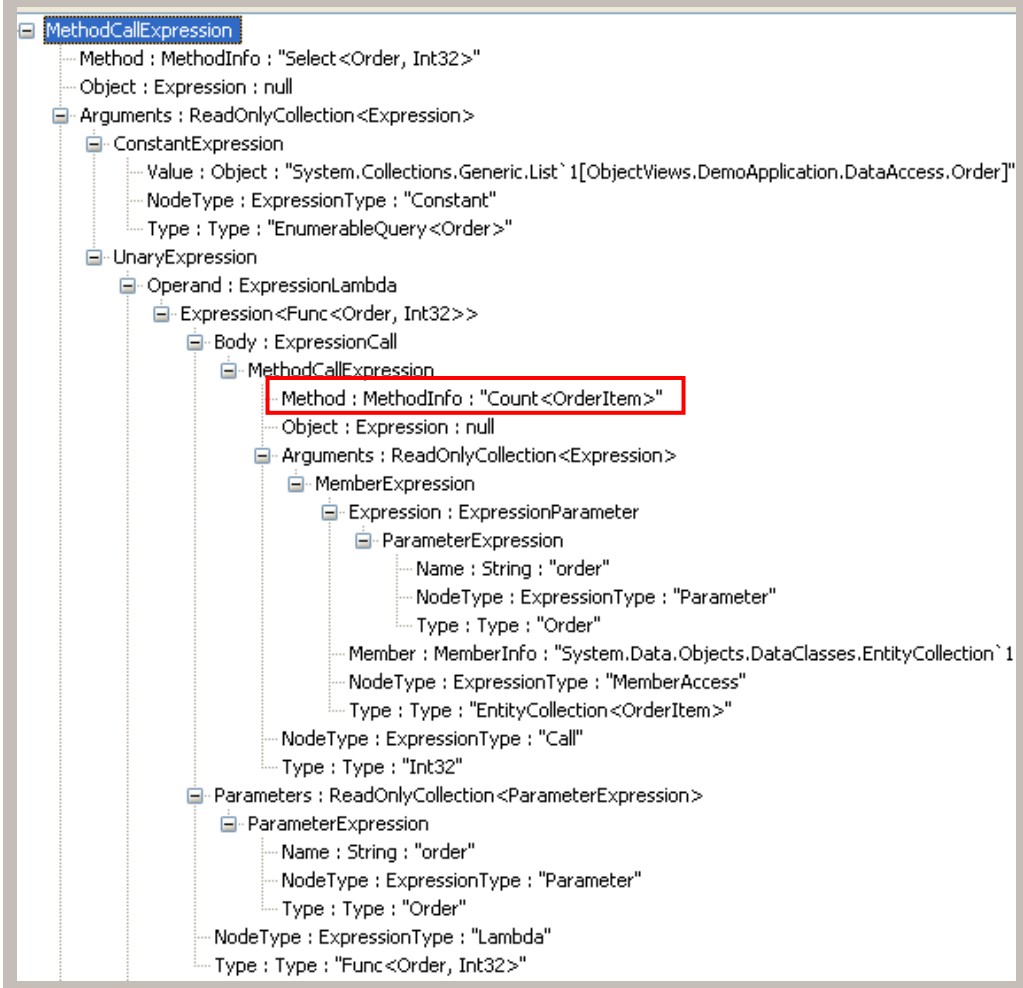
10.2.1.4.2 Verbund mit DB-Constraints

Einleitung	Wird die Verbindung bereits auf Datenbank-Ebene erstellt, so kann beispielweise Linq To Entities diese Verbindung abbilden. Der Zugriff erfolgt anschliessend über ein Property innerhalb der Klasse.
Beispiel	Beziehung zwischen Item und OrderItem. 
Query	`var query = from orderItem in _dataContext.OrderItemList` `select orderItem.Item;`
Expression Tree	MethodCallExpression Method : MethodInfo : "Select<OrderItem, Item>" Object : Expression : null Arguments : ReadOnlyCollection<Expression> ConstantExpression Value : Object : "System.Collections.Generic.List`1[ObjectViews.DemoApplication.DataAccess.OrderItem]" NodeType : ExpressionType : "Constant" Type : Type : "EnumerableQuery<OrderItem>" UnaryExpression Operand : ExpressionLambda Expression<Func<OrderItem, Item>> Body : ExpressionMemberAccess MemberExpression Expression : ExpressionParameter ParameterExpression Name : String : "orderItem" NodeType : ExpressionType : "Parameter" Type : Type : "OrderItem" Member : MemberInfo : "ObjectViews.DemoApplication.DataAccess.Item Item" NodeType : ExpressionType : "MemberAccess" Type : Type : "Item" Parameters : ReadOnlyCollection<ParameterExpression> ParameterExpression Name : String : "orderItem" NodeType : ExpressionType : "Parameter" Type : Type : "OrderItem" NodeType : ExpressionType : "Lambda" Type : Type : "Func<OrderItem, Item>" Method : MethodInfo : null IsLifted : Boolean : "False" IsLiftedToNull : Boolean : "False" NodeType : ExpressionType : "Quote" Type : Type : "Expression<Func<OrderItem, Item>>" NodeType : ExpressionType : "Call" Type : Type : "IQueryable<Item>"
Erläuterungen	**Member Expression** Die Member Expression greift ganz normal auf ein Property der Klasse OrderItem zu. Es wird jedoch anstelle eines primitiven Datentyps ein ganzes Data Object (Klasse Item) zurückgegeben.

10.2.1.5 Berechnete Werte

Einleitung	Eine weitere Möglichkeit von Linq ist es, dass Werte gleich im Query berechnet werden können. Diese werden dann ausgewertet im Rückgabewert zurückgegeben und muss nur einmal durchgeführt werden.
Beispiel	Es wird berechnet wieder Prozent der totale Preis noch ist. Wird 20% Rabatt gewährt, so ist der Rückgabewert 80.
Query	`var query = from item in _dataContext.ItemList` `select 100 - item.DiscountPercent;`
Expression Tree	
Erläuterungen	**Expression Subtract** Eine Subtraktion wird auf eine Expression Substract im Expression Tree abgebildet. Es beinhaltet die zwingenden Elemente Left und Right. Left ist dabei der linke Operand und Right der rechte Operand einer Berechnung. **Constant Expression** Der linke Operand ist eine Constant Expression, weil dessen Wert konstant 100 ist. Dieser Wert ändert sich während dem ganzen Ablauf der Query nie. **Member Expression** Da sich der rechte Operand immer auf ein Property des Data Objects Item bezieht und sich immer nur auf eine Instanz bezieht, ist dies eine Member Expression. Diese definiert auf welches Property sich der rechte Operand bezieht.

10.2.1.6 Aggregierte Werte

Einleitung	Linq unterstützt ebenfalls diverse Aggregatsfunktionen wie z.B. Count, Sum, usw. Dies kann über ein Grouping oder über das Entity Framework gemacht werden. Das Entity Framework kann ebenfalls 1:m Beziehungen abbilden und darauf sind ebenfalls Aggregatsfunktionen möglich.
Query	`var query = from order in _dataContext.OrderList` `select order.OrderItem.Count();`
Expression Tree	
Erläuterungen	**Method Call Expression** Diese Expression enthält das Property *Method*, wo ersichtlich ist, welche Aggregatsfunktion angewandt wurde. Zudem sind auch sämtliche Parameter, die der Funktion übergeben wurde in der Expression enthalten.

10.2.1.7 Visitor Pattern

Das Visitor Pattern ermöglicht sämtliche Knoten im Expression Tree zu besuchen ohne selbst jedes Mal zu überprüfen, ob weitere Knoten vorhanden sind.

Klassen:

- Die Klasse ExpressionVisitor stammt aus einem Beispiel der MSDN und konnte 1:1 verwendet werden.
- DependencyDetectingVisitor ist eine Spezialisierung von ExpressionVisitor und erlaubt das Auslesen spezifischer Informationen aus dem Expression Tree. Dazu müssen in der abgeleiteten Klasse die entsprechenden Visit Methoden (z.B. VisitConstantExpression()) überschrieben werden.

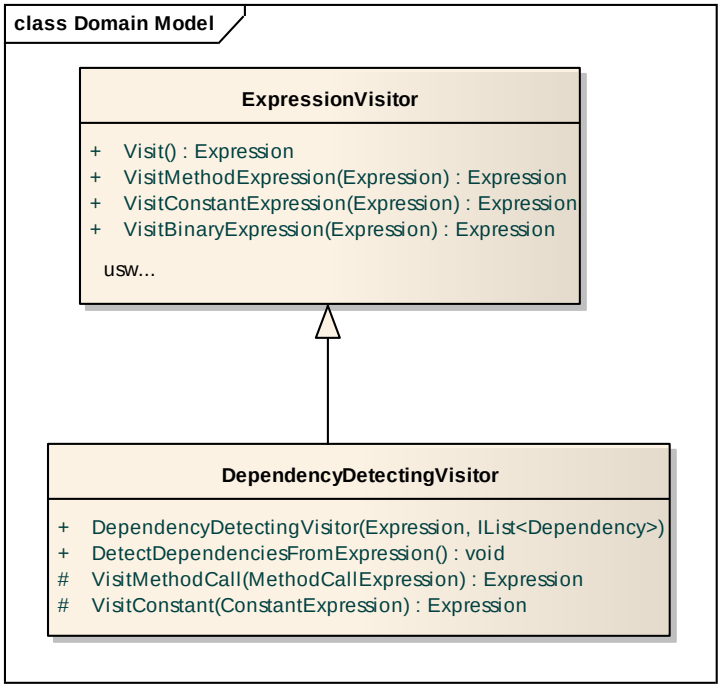


Abbildung 36 - Visitor Pattern

Soll nun die Information herausgelesen werden, welche Listen in die Linq Abfrage involviert waren, kann über die Constant Expression diese Information herausgelesen werden.

10.2.2 TypeDescriptor

10.2.2.1 Einführung

Um Datenänderungen in den Object Views und in den darunterliegenden Data Objects mitzubekommen, benötigen wir einen möglichst generellen Mechanismus, der uns mitteilt, welches Objekt und welches Attribut des Objekts sich verändert hat.

Von allen Object View und Data Object Klassen zu verlangen, das Interface INotifyPropertyChanged zu implementieren, ist mit Programmieraufwand für den Verwender des Frameworks verbunden und bei untypisierten Object Views sowieso nicht anzuwenden, da für diese Object Views ja keine eigene Klasse geschrieben wird.

Wir haben auch mit *Aspects* experimentiert, diese haben aber auch das Problem, dass sie bei untypisierten Object Views nicht greifen können.

Mit dem Type Descriptor kann man ein Objekt via Reflections zur Laufzeit verändern. Der Type Descriptor bietet sogar die Möglichkeit, sich pro Objekt bei einem Event anzumelden, das einem über Änderungen im Objekt informiert.

10.2.2.2 Property Changed Notification

Mit folgendem Code kann man einen Event Handler registrieren, der Aufgerufen wird, wenn sich ein Property eines Objekts verändert:

```
var propertyDescriptor = TypeDescriptor.GetProperties(obj)[propertyName];
propertyDescriptor.AddValueChanged(obj, PropertyChanged);

private void PropertyChanged(object sender, EventArgs eventArgs) {
      var changedObject = sender;
      var propertyChangedEventArgs = (PropertyChangedEventArgs)eventArgs;
      var propertyName = propertyChangedEventArgs.PropertyName;
}
```

Diese Lösung hat zwei Hacken: Erstens funktioniert dieses Event Handling nur dann in jedem Fall, wenn das Objekt vom Typ INotifyPropertyChanged ist, und zweitens hat man im Event Handler nicht immer die Information welches *Property* des Objekts sich verändert hat.

Das erste Problem zeigt sich bei folgendem Code:

```
obj.GetType().GetProperty(propertyName).SetValue(obj, value, null);
// oder
obj.PropertyName = value;
```

Wenn das Property eines Objekt auf eine dieser beiden Arten verändert wird und das Objekt *nicht* vom Typ INotifyPropertyChanged ist, dann wir obig registrierter Event Handler *nicht* aufgerufen. Dies kann jedoch gelöst werden, indem einfach wiederum der Type Descriptor verwendet wird, um das Property zu setzen:

```
var propertyDescriptor = TypeDescriptor.GetProperties(obj)[propertyName];
propertyDescriptor.SetValue(obj, value);
```

Das zweite Problem zeigt sich dadurch, dass es beim Casten der Event Argumente eine Exception gibt. Die übergebenen Event Argumente sind nämlich nur dann vom Typ PropertyChangedEventArgs, wenn das Objekt wiederum vom Typ INotifyPropertyChanged ist. Von diesem Umstand können wir ja aber genau nicht ausgehen. Es ist also nicht immer möglich, den Namen des Propertys, das sich geändert hat, zu ermitteln.

10.3 Systemtest

10.3.1 Testdaten

10.3.1.1 Customer

Id	First Name	Last Name	Steet	Zip Code	City
1	Beni	Egli	Juchstrasse 10	8700	Winterthur
2	Rolf	Latzer	Rütiweg 15	8610	Uster
3	Hans	Mustermann	Seestrasse 45	8610	Uster

10.3.1.2 Item

Id	Item Name	Description	Price Per Product	DiscountPercent	Minimal Discount Quantity
1	Seife	Beste Seife	15	20%	100 items
2	Duschmittel	AXE	5	10%	50 items
4	Coca Cola	Besser als Pepsi	3	15	200 items

10.3.1.3 Order

Id	Customer (fkCustomer)	Creation date	Delivery date
1	Beni Egli (1)	16.02.2009	19.02.2009
3	Beni Egli (1)	18.02.2009	20.02.2009
5	Rolf Latzer (2)	17.02.2009	03.03.2009

10.3.1.4 OrderItem

Id	fkOrder	Quantity	Item (fkItem)
1	1	4	Seife (1)
2	1	20	Coca Cola (4)
3	3	435	Coca Cola (4)
4	5	20	Seife (1)

10.3.2 Funktionale Tests

10.3.2.1 Übersicht

10.3.2.1.1 Readonly Object Views

Id	Bezeichnung	Letztes Testdatum	Resultat
1.1	Projektion	31.05.2009	OK
2.1	Selektion	31.05.2009	OK
3.1	Verbund	31.05.2009	OK
3.2	Instanzen auf gleiche Data Object Typen	31.05.2009	OK
4.1	Zusammengesetzte Werte	31.05.2009	OK
5.1	Aggregierte Werte	31.05.2009	OK
6.1	Aktualisierung bei 1:1 Beziehungen mit einem Data Object	31.05.2009	OK
6.2	Aktualisierung bei 1:1 Beziehungen mit mehreren Data Objects	31.05.2009	OK
6.3	Aktualisierung bei Selektion	31.05.2009	OK/NOK
6.4	Aktualisierung zusammengesetzter Werte	31.05.2009	OK
6.5	Aktualisierung aggregierter Werte	31.05.2009	OK
7.1	Typisierte Object Views	31.05.2009	OK
8.1	Untypisierte Object Views	31.05.2009	Kein Testszenario mehr
9.1	Single Tier Object Views	31.05.2009	OK
10.1	Two Tier Object Views	31.05.2009	OK

10.3.2.1.2 Updateable Object Views

Id	Bezeichnung	Datum	Resultat
11.1	**Einfügen:** Hinzufügen von Daten wenn View nur auf einen Typ von Data Objects verweist	31.05.2009	OK
11.2	**Einfügen:** Hinzufügen von Daten wenn View auf mehrere Typen von Data Objects verweist	05.06.2009	OK
11.3	**Einfügen:** Hinzufügen von weiteren Daten nach erstem Speichern von Änderungen	31.05.2009	OK
11.4	**Einfügen:** Hinzufügen von weiteren Daten nach erstem Speichern von Änderungen	31.05.2009	OK
11.5	**Einfügen:** Hinzufügen von Daten und aktualisieren von anderen Object Views	31.05.2009	OK
12.1	**Bearbeiten:** Bearbeiten einer Object View	31.05.2009	OK
13.1	**Löschen:** Löschen in einer Object View mit Bezug auf ein einzelnes Data Object.	31.05.2009	OK
13.2	**Löschen:** Löschen in einer Object View mit Bezug auf mehrere Data Objects	31.05.2009	OK
14.1	**Undo und Redo:** Rückgängig machen von Änderungen.	31.05.2009	OK
14.2	**Undo und Redo:** Rückgängig machen von Änderungen und anschliessendes Speichern	31.05.2009	OK
14.3	**Undo und Redo:** Wiederherstellen von Änderungen	31.05.2009	OK
14.4	**Undo und Redo:** Wiederherstellen von Änderungen und anschliessendes Speichern	31.05.2009	OK
14.5	**Undo und Redo:** Redo oder Undo nicht möglich	31.05.2009	OK
15.1	**Updateable Collections**	31.05.2009	OK
15.2	**Updateable Entity Framework Data Context**	31.05.2009	OK

10.3.2.1.3 Three Tier Object Views

Id	Bezeichnung	Datum	Resultat
17.1	Lesen von Remote Data Context	09.06.2009	OK
18.1	Schreiben in Remote Data Context - Einfügen	28.05.2009	OK
18.2	Schreiben in Remote Data Context - Bearbeiten	31.05.2009	OK
18.3	Schreiben in Remote Data Context - Löschen	31.05.2009	OK
19.1	Übernehmen von Änderungen aus Remote Data Context – Neu erstellte ID's	29.05.2009	OK
19.2	Übernehmen von Änderungen aus Remote Data Context – Alle Datenänderungen	31.05.2009	NOK

10.3.2.2 Readonly Object Views

10.3.2.2.1 Projektion

Systemtest 1.1	Projektion
Beschreibung	Es können Object Views erstellt werden, mit denen nur gewisse Felder eines Data Objects angezeigt werden. Es sollen dafür keine anderen Felder des Data Objects geladen werden.
Erwartetes Resultat	Die Customer View enthält nur die Id, den Namen (Vor- und Nachnamen) und die Stadt. Diese Ansicht soll möglich sein, ohne dass weitere Daten des Kunden geladen werden.
Resultat	OK
Kommentar	25.03.2009 & 10.05.2009: Die Ansicht ist möglich und es werden keine weiteren Daten geladen.
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.2 Selektion

Systemtest 2.1	Selektion
Beschreibung	Es können Object Views erstellt werden, mit denen eine Auswahl aus einer Data Object Collection angezeigt werden. Diese Einschränkung wird mittels eines Where-Statements gemacht.
Erwartetes Resultat	Es erscheinen nur die Daten, die der Selektion (d.h. dem Where Statement) entsprechen. Änderungen sind entsprechend möglich
Resultat	OK
Kommentar	25.03.2009 & 10.05.2009: Es werden nur die Daten angezeigt und ein Update ist möglich.
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.3 Verbund

Systemtest 3.1	Verbund
Beschreibung	Es können Object Views erstellt werden, die sich aus Data Objects verschiedener Klassen zusammensetzen. Dieser Verbund wird mittels des *join* Schlüsselworts in Linq gemacht.
Erwartetes Resultat	Es können Properties der Data Klasse „OrderItem“ und „Item“ angezeigt werden. Die Data Objects werden mittels join miteinander verknüpft.
Resultat	OK
Kommentar	-
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

Systemtest 3.2	Instanzen auf gleichen Data Object Typen
Beschreibung	Es können Daten von mehreren Instanzen gleicher Datentypen bezogen werden.
Erwartetes Resultat	Der Vergleich ob zwei Kunden in derselben Stadt funktioniert einwandfrei. Wohnen zwei Kunden in derselben Stadt sind sie in dieser Ansicht enthalten.
Resultat	OK
Kommentar	Der Vergleich funktioniert. Wohnen zwei Kunden in der gleichen Stadt so werden diese entsprechend angezeigt.
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.4 Zusammengesetzte Werte

Systemtest 4.1	Zusammengesetzte Werte
Beschreibung	Es muss möglich sein, dass man zusammengesetzte Werte definieren kann und das Framework die Werte entsprechend lädt.
Erwartetes Resultat	Der Name des Kunden in der Hauptansicht setzt sich auf dem Vor- und Nachnamen zusammen. Ein Query soll wie folgt möglich sein: … `select new { CustomerName = customer.FirstName + „“ + customer.LastName, … }`
Resultat	OK
Kommentar	25.03.2009 & 10.05.2009: Es funktioniert und der zusammengesetzte Wert wird angezeigt.
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.5 Aggregierte Werte

Systemtest 5.1	Aggregierte Werte
Beschreibung	Es muss möglich sein, z.B. Summen oder andere Aggregatsfunktionen zu nutzen.
Erwartetes Resultat	Es wird der totale Preis (welcher sich aus der bestellten Menge und dem Preis pro Stück zusammensetzt) in der Bestellübersicht angezeigt.
Resultat	OK
Kommentar	26.03.2009 & 10.05.2009: Der totale Preis wird richtig angezeigt
Datum des Tests	26.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.6 Aktualisierung

Systemtest 6.1	Aktualisierung bei 1:1 Beziehungen mit einem Data Object
Beschreibung	Wird ein Feld in einem Object View geändert, so müssen sämtliche andere Object Views, welche das Feld ebenfalls anzeigen, den aktualisierten Wert anzeigen.
Erwartetes Resultat	Wird in der Detailansicht des Kunden der Nachname „Egli“ auf „Meier“ geändert, so muss in der Gesamtübersicht ebenfalls der geänderte Name erscheinen.
Resultat	OK
Kommentar	25.03.2009 & 10.05.2009: Der Nachname wird auch in der Übersicht auf „Meier“ geändert.
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

Systemtest 6.2	Aktualisierung bei 1:1 Beziehungen mit mehreren Data Objects
Beschreibung	Es können Felder aus Data Objects die mit einem join verbunden sind aktualisiert werden.
Erwartetes Resultat	Wird das Item in der Item List von „Seife“ auf „Duschmittel“ geändert, so erscheint in der OrderItemList (mit OrderItems und Items) ebenfalls der aktualisierte Namen.
Resultat	OK
Kommentar	25.03.2009 & 10.05.2009: Der Name des Items wurde geändert
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

Systemtest 6.3	Aktualisierung bei Selektion
Beschreibung	Werden Daten geändert und entsprechen sie der Selektion (d.h. dem Where-Statement) so müssen sie ebenfalls in der Ansicht erscheinen.
Erwartetes Resultat	Der Ort von Beni Egli kann auf Uster geändert werden und anschliessend erscheinen die Mappings zwischen Rolf Latzer und Beni Egli in der „CustomerSameCityView“.
Resultat	25.03.2009: NOK 10.05.2009: OK/NOK
Kommentar	25.03.2009: Die Mappings erscheinen nicht, wenn sie nachträglich in der Demoapplikation geändert wurden. 10.05.2009 & 31.05.2009: Wenn der Vergleich keinen statischen Wert enthält (z.B. einen String oder eine Zahl), ist eine automatische Aktualisierung noch nicht möglich. Wird jedoch die Einschränkung beispielsweise auf *Stadt == „Zürich“* gemacht, so findet die Aktualisierung statt.
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

Systemtest 6.4	Aktualisierung zusammengesetzter Werte
Beschreibung	Sind alle Werte geladen um ein zusammengesetztes Feld zu berechnen, so soll diese Berechnung direkt durch das Framework vollzogen werden und nicht erneut durch ein Absetzen eines Linq Query s.
Erwartetes Resultat	Der Name des Kunden in der Hauptansicht setzt sich auf dem Vor- und Nachnamen zusammen. Wird der Nachname in der Detailansicht eines Kunden von ‚*Egli*‘ auf ‚*Müller*‘ geändert, so ist der Name anschliessend in der Kundenübersicht auf ‚*Beni Müller*‘ geändert.
Resultat	OK
Kommentar	-
Datum des Tests	25.03.2009, 10.05.2009, 31.05.2009

Systemtest 6.5	Aktualisierung aggregierter Werte
Beschreibung	Wenn Daten hinzugefügt, gelöscht oder geändert werden, sollen sich die Aggregationsfelder entsprechend aktualisieren.
Erwartetes Resultat	Wird ein Bestellitem hinzugefügt, gelöscht oder verändert, so aktualisiert sich der totale Preis.
Resultat	26.03.2009: NOK 10.05.2009: OK
Kommentar	26.03.2009: Es wird nichts aktualisiert, die Aggregation funktioniert nur zu Beginn. Muss abgewartet werden, bis Antwort von H.Huser und/oder A.Schneble. 10.05.2009 & 31.05.2009: Aktualisierung ist nun möglich, wenn die Werte geladen sind.
Datum des Tests	26.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.7 Typisierte Object Views

Systemtest 7.1	Typisierte Object Views
Beschreibung	Es können Object Views erstellt werden, auf deren Daten typensicher via Properties zugegriffen werden kann.
Erwartetes Resultat	Für die Übersicht über alle Kunden wurde die Klasse CustomerView erstellt. Diese wird beim Ändern des Namens ebenfalls aktualisiert und sie aktualisiert die anderen Ansichten bzw. die Datenobjekte.
Resultat	OK
Kommentar	-
Datum des Tests	26.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.8 Untypisierte Object Views

Systemtest 8.1	Untypisierte Views
Beschreibung	Es können Object Views erstellt werden, die anonyme Objekte anzeigen.
Erwartetes Resultat	Sämtliche Felder der Object View werden richtig gesetzt und entsprechend abgefüllt.
Resultat	26.03.2009: NOK 10.05.2009 & 31.05.2009: Kein Testszenario mehr
Kommentar	26.03.2009: Wurde noch nicht implementiert. 10.05.2009: Kann gemäss Antwort von H. Huser / A. Schneble weggelassen werden.
Datum des Tests	26.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.9 Single Tier Object Views

Systemtest 9.1	Single Tier Object Views
Beschreibung	Es können Object Views auf interne Datenstrukturen (z.B. Listen) erstellt werden.
Erwartetes Resultat	Wird die Klasse InMemoryDb verwendet, wird auf Listen im Speicher zugegriffen. Die Daten müssen in der ganzen Applikation zugänglich sein und verändert werden können.
Resultat	OK
Kommentar	-
Datum des Tests	26.03.2009, 10.05.2009, 31.05.2009

10.3.2.2.10 Two Tier Object Views

Systemtest 10.1	Two Tier Object Views
Beschreibung	Es können Object Views auf Datenbanken, auf die direkten Zugriff besteht, erstellt werden.
Erwartetes Resultat	Alle Daten sind in der GUI Applikation zugänglich, können intern verändert werden, werden jedoch noch nicht auf die Datenbank geschrieben.
Resultat	OK
Kommentar	-
Datum des Tests	26.03.2009, 10.05.2009, 31.05.2009

10.3.2.3 Updateable Object Views

10.3.2.3.1 Einfügen

Systemtest 11.1	Hinzufügen von Daten wenn View nur auf einen Typ von Data Objects verweist
Beschreibung	In der Kundenübersicht wird ein neuer Kunde hinzugefügt indem eine neue Zeile erstellt wird. Die Daten des Kunden werden wie folgt eingetragen: Vorname: Heinz Nachname: Müller Strasse: Bahnhofstrasse 35 PLZ/Ort: 8032 Zürich
Erwartetes Resultat	Der Kunde erscheint in der Object View mit einer negativen Id. Sobald auf Save gedrückt wird, erscheint er ebenfalls in der Datenbank.
Resultat	OK
Kommentar	09.05.2009: Datensatz erscheint in der Datenbank mit automatisch generierter Id. 31.05.2009: Generierte Id erscheint nun auch in der Demoapplikation
Datum des Tests	09.05.2009, 31.05.2009

Systemtest 11.2	Hinzufügen von Daten wenn View auf mehrere Typen von Data Objects verweist
Beschreibung	In der OrderItem-Ansicht wird eine neue Bestellposition hinzugefügt. Dabei wird sowohl ein neue Bestellposition eingefügt als auch gleich ein neues Produkt. Es soll folgende Bestellposition eingefügt werden: Anzahl: 30 Produkt: Swatch-Uhr Preis pro Produkt: 50CHF
Erwartetes Resultat	Produkt erscheint in der Ansicht, es wird sofort das Zwischentotal ausgerechnet und auch in der abhängigen Order-Ansicht wird sofort der totale Betrag aktualisiert.
Resultat	09.05.2009: NOK 12.05.2009: OK 31.05.2009: NOK 05.06.2009: OK
Kommentar	09.05.2009: Das Zwischentotal in der OrderItem-Ansicht aktualisiert sich nicht mehr 31.05.2009: Folgender Ablauf schlägt fehl: 1. Neues Item (Produkt) hinzufügen 2. Neue Order zu bestehendem Kunden hinzufügen 3. Neues OrderItem mit Verweis auf neues Item 4. Speichern ➔ Erstellt den Kunden in der Datenbank doppelt
Datum des Tests	09.05.2009, 12.05.2009, 31.05.2009

Systemtest 11.3	Hinzufügen von weiteren Daten nach erstem Speichern von Änderungen
Beschreibung	Folgender Testablauf wird durchgeführt: 1. Neue Order hinzufügen 2. Neues OrderItem (inkl. neuem Produkt) hinzufügen 3. Speichern 4. Hinzufügen von einem weiterem OrderItem 5. Speichern
Erwartetes Resultat	Die Order ist in der Datenbank, zeigt das aktuelle Total an und alle OrderItems sind gespeichert und zeigen das Zwischentotal.
Resultat	12.05.2009: NOK 31.05.2009: OK
Kommentar	12.05.2009: Stürzt ab mit folgender Exception: *Cannot insert the value NULL into column 'ItemName', table 'ObjectViews.dbo.Item'; column does not allow nulls. INSERT fails. The statement has been terminated.*
Datum des Tests	12.05.2009, 31.05.2009

Systemtest 11.4	Hinzufügen von weiteren Daten nach erstem Speichern von Änderungen
Beschreibung	Folgender Testablauf wird durchgeführt: 1. Neue Order hinzufügen 2. Speichern 3. Neuer Kunde hinzufügen 4. Speichern
Erwartetes Resultat	Sowohl die Order als auch der Kunde ist in der Datenbank persistiert.
Resultat	OK
Kommentar	-
Datum des Tests	12.05.2009, 31.05.2009

Systemtest 11.5	Hinzufügen von Daten und aktualisieren von anderen Object Views
Beschreibung	Aggregatsfelder müssen ebenfalls aktualisiert werden, wenn ein Datensatz hinzukommt.
Erwartetes Resultat	Die Anzahl Bestellungen in der Customer View aktualisiert sich sobald eine Order hinzukommt.
Resultat	OK
Kommentar	-
Datum des Tests	09.05.2009, 31.05.2009

10.3.2.3.2 Bearbeiten

Systemtest 12.1	Bearbeiten einer Object View
Beschreibung	Wird ein Object View bearbeitet, muss die entsprechende Änderung ebenfalls auf der Datenquelle persistiert werden.
Erwartetes Resultat	Wird der Vorname des Kunden „Rolf Latzer“ auf „Rolfo“ geändert, so ist in der Datenbank der Vornamen ebenfalls entsprechend geändert.
Resultat	OK
Kommentar	-
Datum des Tests	09.05.2009, 31.05.2009

10.3.2.3.3 Löschen

Systemtest 13.1	Löschen in einer Object View mit Bezug auf ein einzelnes Data Object
Beschreibung	Sobald aus der Object View ein Eintrag gelöscht wird, soll dieses Löschen ebenfalls in der Datenquelle erfolgen.
Erwartetes Resultat	Wird ein Item aus der Item-View gelöscht und auf Save gedrückt, so ist das Item aus der Datenbank gelöscht.
Resultat	09.05.2009: NOK 12.05.2009 & 31.05.2009: OK
Kommentar	09.05.2009: • Schlägt bereits beim Entfernen aus dem DataGrid fehl • Exception tritt auf, wenn dadurch Abhängigkeiten in der Datenbank verletzt werden → Verhalten muss so sein
Datum des Tests	09.05.2009, 12.05.2009, 31.05.2009

Systemtest 13.2	Löschen in einer Object View mit Bezug auf mehrere Data Objects
Beschreibung	Verweist die Object View auf mehrere Typen von Data Objects, so ist unklar welche Data Objects tatsächlich aus der Datenquelle gelöscht werden sollen. Diese Typen, welche gelöscht werden können über Attribute angegeben werden. Es soll nun eine Bestellposition gelöscht werden. Dabei soll nur das OrderItem und nicht auch das Item (d.h. das eigentliche Produkt) gelöscht werden.
Erwartetes Resultat	Sobald etwas aus der OrderItem-View gelöscht wird, ist nur das OrderItem gelöscht und nicht ebenfalls das Item. Zudem wird in das Gesamttotal in der Order-View aktualisiert.
Resultat	OK
Kommentar	09.05.2009: • Schlägt bereits beim Entfernen aus dem DataGrid fehl 12.05.2009: Gesamttotal in der OrderView wird nach Entfernen eines OrderItems nicht aktualisiert. 31.05.2009: Funktioniert
Datum des Tests	09.05.2009, 12.05.2009, 31.05.2009

10.3.2.3.4 Undo / Redo

Systemtest 14.1	Rückgängig machen von Änderungen
Beschreibung	Änderungen an einer Object View sollen rückgängig gemacht werden können solange sie noch nicht auf der Datenquelle persistiert worden sind. Wird der Name von „Rolf“ auf „Rolfo“ geändert, muss mit dem Knopf „Undo“ der Name wieder zurückgeändert werden. Dasselbe gilt für einen neuen Kunden oder einen gelöschten Kunden.
Erwartetes Resultat	Für alle Aktionen gilt, dass sie wieder rückgängig gemacht werden können. Dies ist bis zu einer beliebigen Hierarchiestufe möglich.
Resultat	09.05.2009: NOK 12.05.2009 & 31.05.2009: OK
Kommentar	09.05.2009: Entfernen eines Datensatzes nicht möglich
Datum des Tests	09.05.2009, 12.05.2009, 31.05.2009

Systemtest 14.2	Rückgängig machen von Änderungen und anschliessendes Speichern
Beschreibung	Wird der Name von „Rolf“ auf „Rolfo“ geändert werden und mit einem Undo rückgängig gemacht, soll nichts gespeichert werden.
Erwartetes Resultat	Der Name in der Applikation und in der Datenbank ist noch immer „Rolf“
Resultat	OK
Kommentar	-
Datum des Tests	12.05.2009, 31.05.2009

Systemtest 14.3	Wiederherstellen von Änderungen
Beschreibung	Nachdem eine Änderung rückgängig gemacht wurde, soll sie vom Benutzer auch wiederhergestellt werden können, solange die Datenquelle nicht aktualisiert wurde.
Erwartetes Resultat	Wird der Name von „Rolfo“ auf „Rolf“ rückgängig gemacht, muss mit dem Knopf „Redo“ der Name wiederhergestellt werden können. Dasselbe gilt für einen neuen Kunden oder einen gelöschten Kunden.
Resultat	09.05.2009: NOK 12.05.2009 & 31.05.2009: OK
Kommentar	09.05.2009: Entfernen eines Datensatzes nicht möglich
Datum des Tests	09.05.2009, 31.05.2009

Systemtest 14.4	Wiederherstellen von Änderungen und anschliessendes Speichern
Beschreibung	Nachdem eine Änderung rückgängig gemacht wurde, soll sie vom Benutzer auch wiederhergestellt werden können, solange die Datenquelle nicht aktualisiert wurde. Ein Speichern der wiederhergestellten Aktion soll ebenfalls möglich sein.
Erwartetes Resultat	Der Name „Rolfo“ wird sowohl in der Applikation als auch in der Datenbank angezeigt.
Resultat	OK
Kommentar	-
Datum des Tests	12.05.2009, 31.05.2009

Systemtest 14.5	Redo oder Undo nicht möglich
Beschreibung	Fall 1: Ist noch keine Aktion getätigt worden, kann weder Undo oder Redo getätigt werden. Fall 2: Sind alle Aktionen rückgängig gemacht worden, kann kein Undo mehr gemacht werden Fall 3: Sind alle Aktionen wiederhergestellt, kann kein Redo mehr gemacht werden.
Erwartetes Resultat	Es wird in allen Fällen eine Exception geworfen, sobald ein Undo oder Redo nicht mehr möglich ist.
Resultat	OK
Kommentar	-
Datum des Tests	12.05.2009, 31.05.2009

10.3.2.3.5 Updateable Collections

Systemtest 15.1	Verwaltung von Plain Old C# Objects
Beschreibung	Sollen keine Datenbanken oder andere persistente Dateistrukturen eingesetzt werden, kann die Verwaltung ebenfalls im Speicher stattfinden.
Erwartetes Resultat	Es kann analog zur Datenbank mit Listen gearbeitet werden. Dazu wird lediglich der dazugehörige *CollectionContextWrapper* benötigt.
Resultat	OK
Kommentar	-
Datum des Tests	10.05.2009, 31.05.2009

10.3.2.3.6 Updateable Entity Framework Data Context

Systemtest 16.1	Daten mittels dem Entity Framework und dem entsprechenden Wrapper persistent machen
Beschreibung	Sämtliche CRUD Operationen können auf eine Datenbank ausgeführt werden.
Erwartetes Resultat	Mittels dem dazugehörigen *EntityFrameworkWrapper* können sämtliche Änderungen (Insert, Update, Delete) persistent gemacht werden.
Resultat	OK
Kommentar	10.05.2009: Änderungen erscheinen auf der Datebank (wurde via SQL Management Studio überprüft)
Datum des Tests	10.05.2009, 31.05.2009

10.3.2.4 Three Tier Object Views

10.3.2.4.1 Lesen von Remote Data Context

Systemtest 17.1	Lesen von Remote Data Context
Beschreibung	Es ist möglich mittels Interlinq Daten aus einem Remote Data Context zu holen. Es wird keine Datenbank auf dem Client geholt
Erwartetes Resultat	Es werden die Daten der entfernten Umgebung angezeigt.
Resultat	OK
Kommentar	-
Datum des Tests	09.06.2009

10.3.2.4.2 Schreiben in Remote Data Context (Einfügen, Bearbeiten, Löschen)

Systemtest 18.1	Schreiben in Remote Data Context - Einfügen
Beschreibung	Der Benutzer kann über einen WCF-Service neue Data Objects persistieren.
Erwartetes Resultat	Nachdem der Benutzer neue Data Objects in der Demoapplikation eingefügt hat und das Speichern ausgelöst hat, sind die neuen Datensätze auf der entfernten Datenbank vorhanden.
Resultat	OK
Kommentar	28.05.2009: Läuft auch in einer verteilten Umgebung mittels basicHttpBinding.
Datum des Tests	28.05.2009

Systemtest 18.2	Schreiben in Remote Data Context - Bearbeiten
Beschreibung	Der Benutzer kann über einen WCF-Service Daten dauerhaft ändern.
Erwartetes Resultat	Nachdem der Benutzer Änderungen in der Applikation durchgeführt hat, können diese an den Server gesendet und entsprechend in der Datenbank gespeichert werden.
Resultat	OK
Kommentar	-
Datum des Tests	28.05.2009

Systemtest 18.3	Schreiben in Remote Data Context - Löschen
Beschreibung	Der Benutzer kann über einen WCF-Service Data Objects aus der Datenbank löschen.
Erwartetes Resultat	Nachdem der Benutzer Object View Items gelöscht hat, kann mit einer entsprechenden Speicheraktion die Daten auf der entfernten Datenbank gelöscht werden.
Resultat	OK
Kommentar	-
Datum des Tests	28.05.2009

10.3.2.4.3 Übernehmen von Änderungen aus Remote Data Context

Systemtest 19.1	Übernehmen von Änderungen aus Remote Data Context – Neu erstellte ID's
Beschreibung	In der Demoapplikation erscheinen die neu eingefügten Daten mit der richtigen Id.
Erwartetes Resultat	Der Benutzer erhält ein aktuelles Set seiner Daten zurück. (d.h. er erhält die Datenbankid für alle erstellten Data Objects)
Resultat	OK
Kommentar	29.05.2009: Die Id erhält der Benutzer zurück
Datum des Tests	29.05.2009

Systemtest 19.2	Übernehmen von Änderungen aus Remote Data Context – Alle Datenänderungen
Beschreibung	Der Stand der Daten ist bei allen Benutzern aktuell.
Erwartetes Resultat	Alle Datenänderungen eines Benutzers werden bei allen anderen Benutzern verteilt.
Resultat	NOK
Kommentar	29.05.2009: Wurde noch nicht implementiert.
Datum des Tests	29.05.2009

10.3.3 Nichtfunktionale Tests

10.3.3.1 Übersicht

Id	Bezeichnung	Datum	Resultat
20.1	Konfigurieraufwand	31.05.2009	OK
21.1	Exception	31.05.2009	NOK
22.1	Aktualisierung	31.05.2009	OK
22.2	Performanz	27.03.2009	OK
22.3	Viele Daten	31.05.2009	OK
24.1	Schnittstellen	31.05.2009	OK
24.2	Keine Anforderung an Data Objects	31.05.2009	NOK

10.3.3.2 Bedienbarkeit

Systemtest 20.1	Konfigurieraufwand
Beschreibung	Das Framework muss ohne Konfigurieraufwand eingesetzt werden können.
Erwartetes Resultat	Es existieren keine Konfigurationsdateien, sondern das Framework kann direkt über einen Befehl benutzt werden.
Resultat	OK
Kommentar	31.05.2009: Es werden keine Konfigurationsdateien benötigt. Sämtliche Einstellungen können über Attribute vollzogen werden, deshalb sollte es möglich das Framework rasch einzusetzen.
Datum des Tests	26.03.2009, 31.05.2009

10.3.3.3 Zuverlässigkeit

Systemtest 21.1	Exceptions
Beschreibung	Exceptions werden weitergereicht, wenn sie nicht vom Framework geworfen werden und Exceptions innerhalb des Frameworks beeinträchtigen nicht die Applikation ausserhalb.
Erwartetes Resultat	Keine Exception vom Framework, die zum Programmabsturz der Demoapplikation führt. Ist die Datenbankverbindung nicht vorhanden, wird die Exception der Demoapplikation weitergereicht.
Resultat	26.03.2009: OK 31.05.2009: NOK
Kommentar	31.05.2009: Es gibt Queries, mit denen das Framework nicht umgehen kann. Die meisten Fälle solcher Queries werden vom Query Validator erkannt. Andere führen leider noch zu einer Exception.
Datum des Tests	26.03.2009, 31.05.2009

10.3.3.4 Leistung

Systemtest 22.1	Aktualisierung
Beschreibung	Es dürfen nur diejenigen Object View Items aktualisiert werden, die von einer Änderung betroffen sind.
Erwartetes Resultat	Wird der Vorname eines Kunden in der CustomerEditView (Detailansicht) geändert, so wird nur derselbe Kunde in der CustomerView (Hauptansicht mit allen Kunden) aktualisiert.
Resultat	OK
Kommentar	26.03.2009: Es wird immer direkt das entsprechende Object View Item aktualisiert (mit dem TypeDescriptor). Deshalb ist die Aktualisierung sehr performant. 31.05.2009: Mit den Commands und dem Change Tracker werden ebenfalls nur diejenigen Änderungen an die Datenbank weitergereicht, welche die Änderung wirklich betrifft.
Datum des Tests	26.03.2009, 31.05.2009

Systemtest 22.2	Performanz
Beschreibung	Es darf durch den Einsatz unseres Frameworks zu keinem Performanceverlust kommen.
Erwartetes Resultat	Die Queries dauern mit dem Framework maximal 10% länger als wenn sie direkt abgesetzt werden.
Resultat	OK
Kommentar	Es wurde ein Unit Test geschrieben, um zu zeigen, dass das Framework genügend gut optimiert wurde und keine zeitlichen Einbussen durch die Nutzung des Frameworks entstehen.
Datum des Tests	26.03.2009, 31.05.2009

Systemtest 22.3	Viele Daten
Beschreibung	Das Framework soll mit mehr als 100‘000 Datensätzen problemlos umgehen können.
Erwartetes Resultat	Sobald die Daten geladen sind, sollen die Daten performant verändert werden können.
Resultat	OK
Kommentar	Es dauert lediglich etwas bis WPF die Daten anzeigt. Sind die Daten geladen, kann effizient die Änderungen propagiert werden.
Datum des Tests	26.03.2009, 31.05.2009

10.3.3.5 Schnittstellen

Systemtest 24.1	Schnittstellen
Beschreibung	Das Framework soll zwei Schnittstellen anbieten. Eines für die typisierten und eines für die untypisierten Views.
Erwartetes Resultat	Es gibt im Framework zwei Methoden um die typisierten bzw. nicht typisierten Views nutzen zu können.
Resultat	OK
Kommentar	26.03.2009: Es gibt zwei Methoden, jedoch sind die untypisierten Views noch nicht implementiert. 31.05.2009: Untypisierte Views sind kein Thema mehr
Datum des Tests	26.03.2009, 31.05.2009

Systemtest 24.2	Keine Anforderung an Data Objects
Beschreibung	Die Data Objects sollen direkt verwendet werden können, ohne das diese in ihrem ursprünglichen Zustand verändert worden sind.
Erwartetes Resultat	Data Objects können ohne Anpassung genutzt werden.
Resultat	NOK
Kommentar	31.05.2009: Es muss das Interface IDataObject implementiert werden
Datum des Tests	31.05.2009

10.4 Erfahrungsberichte

10.4.1 Benjamin Egli

10.4.1.1 Projektverlauf

Als uns beim Kickoff-Meeting erstmals die Idee der Object Views erklärt wurde, konnte ich mir noch nicht sehr gut vorstellen, was da auf uns zukommen wird. Glücklicherweise wurde uns empfohlen, uns mittels einer Demoapplikation in die Thematik einzuarbeiten, was sehr gut funktioniert hat.

Trotzdem war die Analyse und der Aufbau eines solchen Frameworks, für uns beide, Rolf und mich, etwas Neues. Dies hatte zur Folge hatte, dass wir nicht immer direkt zur Lösung kamen, sondern oft noch einmal einen Schritt zurückgehen mussten, um einen neuen Anlauf zu holen. Dies hat uns vor allem zu Beginn viel Zeit gekostet, aber im Verlauf des Projekts lief es dann immer runder und wir konnten sehr effizient arbeiten.

Was es natürlich immer mal wieder gab, waren isolierte knifflige Probleme, aber ohne die wäre das Programmiererleben ja langweilig. Die Analyse der Abfragen war teilweise eine mühsame Sache und auch das Entity Framework hat uns so einige Steine in den Weg gelegt, aber schliesslich konnten wir für alles eine Lösung finden.

Zu Beginn des Projekts hatte ich nicht gedacht, dass wir das Framework soweit zu Ende bringen können, dass es auch in einer verteilten Umgebung eingesetzt werden kann. Doch lief es nach dem etwas harzigen Start dann sehr rund und wir haben das Projekt tatsächlich zu Ende gebracht. Natürlich kann man endlos an einer Software programmieren, wie wir alle wissen. Das Object Views Framework hat aus meiner Sicht jedoch einen Stand erreicht, der eine erste Integration im produktiven Umfeld und einen Praxistest erlaubt.

10.4.1.2 Zusammenarbeit

Mit Rolf habe ich schon öfters zusammengearbeitet und war mir darum sicher, einen engagierten und kompetenten Teamkollegen an der Seite zu haben. Die Arbeitsteilung hat sich quasi intuitiv ergeben. Rolf hat sich mehr um die einzelnen Technologien gekümmert (Expression Trees, Entity Framework, Services), während dem ich mehr mit der Architektur des Object Views Framework beschäftigt war. Bei Problemen haben wir uns gegenseitig ausgeholfen, das hat sehr gut geklappt.

Die Betreuung empfand ich als sehr angenehm. Man hat uns viel Freiheiten gelassen, und wenn Probleme auftauchten, dann hatten wir stets einen Ansprechpartner. Was mir besonders gefiel, war, dass Simon Gubler schon sehr früh einbezogen wurde, um die Integration der Object Views im pmMDA zu vollziehen. Dadurch hatten wir schon früh ein Feedback, inwiefern unsere Arbeit einsetzbar ist und was noch geändert werden muss.

10.4.1.3 Fazit

Ich bin sehr zufrieden mit unserer Bachelorarbeit. Wir haben einiges mehr erreicht, als ich zu Beginn gedacht habe. Das erreichte Resultat empfinde ich als sehr gelungen. Natürlich gibt es noch offene Punkte und natürlich muss sich das Framework zuerst noch im Praxistest beweisen. Aber ich war noch nie an einem Projekt beteiligt, bei dem es *nicht* so war.

Wie gesagt war die Entwicklung eines solchen Frameworks Neuland für mich, daher war die Arbeit sehr interessant und es macht mich auch ein wenig Stolz, dass wir eine solch komplexe Aufgabe gemeistert haben. Wenn das Framework nun auch wirklich zum Einsatz kommt bei der bbv, dann bin ich rundum zufrieden.

10.4.2 Rolf Latzer

10.4.2.1 Projektverlauf

Die ursprüngliche Idee ein modulares Framework für Webapplikationen zu erstellen, wurde beim Start des Projektes nochmals ziemlich abgeändert. Auf einmal war die Rede von Object Views. Wir erhielten ein bereits vorbereitetes Papier wo die Idee etwas beschrieben war.
Da stand man nach einer Woche mit einer Idee und wusste zu Beginn eigentlich selbst nicht genau, was uns nun erwartet. Erst mit dem Erstellen einer Demoapplikation und dem Aufzeichnen der Szenarien wurde uns langsam etwas klarer, was eigentlich erwartet wurde. Speziell am Projekt war, dass eigentlich keine wirklichen Anforderungen an das Framework vorhanden waren, sondern diese mussten wir selbst aus den Szenarien ableiten.

Oftmals habe ich mich gefragt, ob wir am Ende wirklich ein brauchbares Framework haben oder ob unsere Idee an technischen Schwierigkeiten scheitert. Diese technischen Schwierigkeiten tauchten auch immer wieder auf. Oftmals mussten wir erstellten Code nochmals komplett überarbeiten oder die Struktur völlig ändern. Beispielsweise benötigten wir doch mehrere Anläufe bis wir eine saubere und vollständige Struktur der Metadaten hatten. Eine Änderung der Metadatenstrukturen hatte beispielsweise auch wieder eine Änderung an den Visitoren (welche die Metadaten aufbauen) zur Folge. Das Verändern der Visitoren hatte wiederum Auswirkungen auf die Unit Tests. So kam es, dass Änderungen manchmal etwas schwerwiegend waren, aber dennoch nötig um die Qualität des Frameworks zu gewährleisten.
Manchmal war es auch nicht ganz einfach mit der Komplexität klarzukommen. Alleine das Parsen der Expression Trees hatte bereits viele Tücken. So werden beispielsweise durch das Verschachteln der verschiedenen Statements (from, join, where, select, usw.) implizit anonyme Typen erstellt, welche wir für die Metadaten wieder entfernen mussten. Ebenfalls sind die Expression Trees beispielweise für Linq To Objects und Linq To Entities unterschiedlich.

Ich lernte besonders die Unit Tests als effektives Werkzeug einzusetzen. Unser Framework eignete sich dafür auch sehr gut, da man die Resultate sehr einfach mit Unit Tests überprüfen kann. Schliesslich half uns dies sehr, wenn wir nach einer Änderung die Unit Tests laufen lassen konnten und wir sahen, wo die Probleme lagen.

Dass wir kurz vor Schluss des Projektes unser Framework noch der Firma bbv vorstellen konnten, stimmte mich zudem positiv, dass die Arbeit möglicherweise sogar „überleben“ könnte und nicht nur in einer Schublade laden könnte.

10.4.2.2 Zusammenarbeit

Die Zusammenarbeit mit Beni Egli klappte gut, da wir uns auch bereits aus dem Verlaufe des Studiums kannten. Etwas speziell war für mich zu Beginn, dass Beni doch fast zehn Jahre mehr Programmiererfahrung hat und deshalb bei Programmierproblemen und auch in der Architektur oftmals eine bessere Lösung präsentieren konnte als ich. Ich denke jedoch für mich war dies eine grosse Chance und ich habe viel von seiner Erfahrung profitiert.

Die Sitzungen waren meist sehr hilfreich. Zum einen konnte wir jeweils den aktuellen Stand präsentieren und zeigen, dass wir nicht die ganze Woche faul herumgesessen sind und zum anderen konnten wir jeweils immer gleich offene Fragen diskutieren und klären.

10.4.2.3 Fazit

Ich kann ein sehr erfolgreiches Fazit ziehen, da wir fast alle Anforderungen, die wir uns selbst gestellt hatten, erreichen konnten. Meines Erachtens haben wir ein ziemlich innovatives Framework erschaffen, welches jetzt vielleicht bei der bbv sogar zum Einsatz kommen könnte.

10.5 Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.

Rapperswil, den 10.06.2009 ……
Benjamin Egli

Rapperswil, den 10.06.2009 ……
Rolf Latzer