HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Ascentiv

Elements Model Driven

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2010

Autor:	Marco Bachmann
Betreuer:	Hansjörg Huser
Projektpartner:	Ascentiv AG, Zürich
Gegenleser:	Hans Rudin

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Kurzfassung der Studienarbeit

Abteilung	**Informatik**
Name des Studierenden	**Marco Bachmann**
Studienjahr	**HS 2010**
Titel der Studienarbeit	**Elements Model Driven**
Examinatorin / Examinator	**Hansjörg Huser**
Themengebiet	**Software**
Projektpartner	**Ascentiv AG, Zürich**
Institut	**Institut für vernetzte Systeme**

Elements Model Driven ist ein Erweiterung für das Microsoft Visual Studio 2010, um graphisch Domänen modellieren zu können. Anhand dieser Modelle werden Business Klassen, Datenbankdefinitionen und OR-Mapping generiert, welcher auf das Ascentiv Elements Framework aufbaut.

Ascentiv Elements bietet eine generische Lösung für Applikationen basierend auf einer Service Orientiented Architecture, wobei die Architekturschichten von Datenbank (MSSql), OR-Mapping (Entity Framework), Kommunikationsschnittstelle (WCF) bis hin zu den Business Objekten (C#) vom Framework übernommen werden.

Zur Implementierung des graphischen Editors wurde das Domainspecific Language Toolkit von Microsoft verwendet, welches dazu dient, domänenspezifische Sprachen zu entwerfen und umzusetzen. Aufgrund der Integration ins Visual Studios ist es transparent in die Projektstruktur integriert und erleichtert dem Entwickler das Arbeiten mit Ascentiv Elements.

Elements Model Driven unterstützt alle gängigen Objektabbildungen, wie zum Beispiel Klassen, Compositionen, Vererbungen, N zu N Relationen, Rekursionen, Mehrfachbeziehungen und Enumerationen. Unzählige Validierungen helfen dem Entwickler, Fehler vorzubeugen und gibt ihm zugleich jedoch volle Flexibilität durch eigene Erweiterungen. Damit die Übersicht in grossen Projekten nicht verloren, geht sind Relationen auch als Attribute darstellbar.

Mit Elements Model Driven und Ascentiv Elements sind auch grosse Projekte sehr effizient realisierbar.

Ascentiv AG
Binzstrasse 18
CH-8045 Zürich

T +41 (0)44 455 62 80
F +41 (0)44 455 60 69
info@ascentiv.ch
www.ascentiv.ch

Technischer Bericht

Elements Model Driven

Marco Bachmann
mbachmann@ascentiv.ch

Versionen

Datum	Version	Kommentar	Autor
08.10.10	0.1	Erstellt	mb
29.11.10	0.2	Struktur erstellt	mb
06.12.10	0.3	Kapitel gefüllt	mb
13.12.10	0.4	Schlussfolgerungen	Mb
20.12.10	0.5	Umstrukturierung gemäss Meeting	Mb
23.12.10	1.0	Release	mb

Inhaltsverzeichnis

1 Einleitung 3
1.1 Zweck des Dokuments 3
1.2 Glossar 3
2 Übersicht 4
2.1 Ausgangslage 4
2.1.1 Asentiv Elements 4
2.1.2 Abbildung 1 - Ascentiv Elements ÜberblickGraphisches Modellieren 4
2.2 Motivation 4
2.2.1 Graphisches Modellieren 4
2.2.2 Schnellere stabilere Entwicklung 5
2.2.3 Versionensprünge 5
2.3 Aufgabenstellung 5
3 Konzept 7
3.1 Meta Modell 7
3.2 Graphische Darstellung 7
3.3 Generierungsablauf 8
4 Technologien 9
4.1 VSVMSDK 9
4.1.1 Übersicht 9
4.1.2 T4 Template 9
4.1.3 Entwicklungsprozess 10
4.1.4 Sprache Modellieren 11
4.1.4.1 Model 11
4.1.4.2 Shapes 13
4.1.4.3 Mapping 13
4.1.5 Erweiterungsmöglichkeiten 14
4.1.5.1 Custom Code 14
4.1.5.2 Coherence Rules 15
4.1.5.3 Validation Rules 15
4.1.6 Deployment 16
4.1.6.1 Code Generator 16
4.2 Entity Framework 17
4.3 Ascentiv Elements 17
4.3.1 Übersicht 17
4.3.2 BO Facade 18
4.3.3 Konventionen 19
5 Arbeitsprozess mit ElementsMD 20
5.1 Graphisches Modellieren 20
5.2 Entwicklungsprozess bisher 20
5.3 Entwicklungsprozess mit ElementsMD 20

1 Einleitung

1.1 Zweck des Dokuments

Das Zielpublikum dieses Dokuments sind Software Architekten. Es beschreibt die Motivation und Grundkonzepte für Elements Model Driven und dessen verwendeten Technologien.

1.2 Glossar

Dsl	Domain Specific Language, eine Beschreibungssprache (in xml) um weitere Beschreibungssprachen zu definieren.
VSVMSDK	Visual Studio Visualization and Modeling SDK, die neue Bezeichnung für Dsl.
T4 Template	T4 ist eine Code Generierungssprache, welche auf „In place substitution" basiert.
Ascentiv Elements	Ein Framework der Firma Ascentiv, welches dazu dient, den Weg von der Datenbank auf dem Server bis hin zum Client zu vereinfachen.
C#	Eine Objektorientierte Programmiersprache von Microsoft.
Containment	Eine Relation zwischen zwei Objekten, welche eine Teil/Ganzes Beziehung beschreibt
Entity Framework	Ein objekt-relationaler Mapper für .Net
Edmx	Eine Beschreibung (in xml) für ein objekt-relationales Mapping im Entity Framework.
TTH	Table per Hierarchy. Eine Umsetzung für Generalisierung auf der Datenbank, wobei eine Tabelle für alle Klassen der Vererbungsstruktur erstellt wird und anhand eines bestimmten Attributs auf den effektiven Typ geschlossen werden kann.
TTP	Table per Type. Eine Umsetzung für Generalisierung auf der Datenbank, wobei jede Klasse in der Vererbungsstruktur eine eigene Tabelle erhält erhält und diese Tabellen über einen gemeinsamen Primärschlüssel verbunden sind.
BO	Business Objects sind Objekte basierend auf AscentivElements, welche eine Entität einer Domäne wiederspiegelt.
DTO	Data Transfer Objebts sind Objekte, die über eine Datenleitung geschickt werden können.
WCF	Windows Communication Foundation ist eine Service orientierte Übertragungsschnittstelle im Microsoft .Net Framework.
ElementsMD	Elements Model Driven
Elmd	Elements Model Driven Domänen Beschreibung
MS-PL	Microsoft Public License, Mirosofts Umsetzung der GPL (Open Source Lizenz)
CLR	Laufzeitumgebung für .Net Programme.
Attributierung	Eine Technik in .Net Sprachen um Typen statisch mit Metadaten zu versehen, welche über Reflection ausgewertet werden können.
Connection String	Ein Zeichenkette, welche alle Informationen enthält um mit einer Datenbank verbinden zu können.
Sql Compare File	Eine Datei die Informationen enthält, wie zwei Datenbanken zu vergleichen sind.
VSIX	Installationspaket für eine Visual Studio Erweiterung
DAL	Data Access Layer, die Programmschicht, welche die Datenbank anspricht.

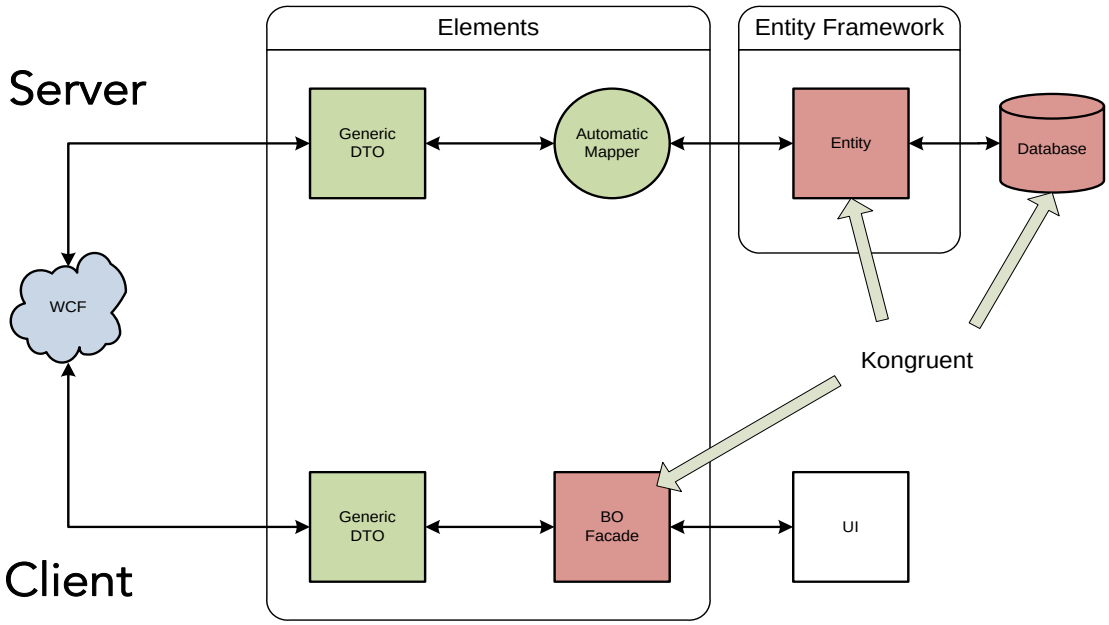
2 Übersicht

2.1 Ausgangslage

Für die Entwicklung von individuellen Geschäftslösungen besteht das Bedürfnis, abstrakte Geschäftsmodelle (Domänen Modell) mittels grafischen Tools effizient zu beschreiben. Aus diesen Geschäftsmodellen sollen unterschiedliche Artefakte bzw. Programm Code erzeugt werden können (Model Driven Approach).

2.1.1 Asentiv Elements

Viele Lösungen der Firma Ascentiv AG sind mit verteilten Systemen umgesetzt, die meisten haben eine Server Client Architektur mit Service Schnittstellen, eine sogenannte Service Oriented Architecture (SOA). Die Umsetzung einer solchen Lösung verläuft in vielen Architekturschichten immer gleich. Die Ascentiv AG hat hier eine Lösung geschaffen, Ascentiv Elements genannt, die neben einer generische CRUD Serviceschnittstelle weitere Automationen bietet, die sonst bei jedem Projekt implementiert werden müssten.
Als OR-Mapper wird das Microsoft Entity Framework verwendet, welches ein Metamodel namens Edmx benötigt. Anhand dieses Metamodels werden die Entity Framework Klassen erzeugt.
Durch Ascentiv Elements verkürzt sich die Time-To-Market enorm und die Software Qualität steigt, da einheitlichere Strukturen verwendet werden. Die grösste Arbeit die mit Ascentiv Elements anfällt, ist die präzise Abbildung der Domänen Struktur auf verschiedene Software Layer. Dabei müssen Datenbank, Edmx und die Business Object Facade kongruent sein.



2.1.2 Abbildung 1 - Ascentiv Elements ÜberblickGraphisches Modellieren

Ein graphisches Modellieren der Domäne muss mit externen Tools wie Enterprise Architect erledigt werden, da kein implementierungsnahes Tool in der Lage ist, eine Domäne sinnvoll zu repräsentieren. Somit muss die Struktur immer repliziert werden.

2.2 Motivation

2.2.1 Graphisches Modellieren

Als Software Entwickler möchte man sich nie wiederholen müssen (DRY-Prinzip). Dies gilt nicht nur für Code, sondern auch für die graphische Repräsentation. Um Domänen Abbilden zu können wird im Idealfall UML verwendet, welches sich weitgehend in der Software

Entwicklung durchgesetzt hat. Dabei unterscheiden wir das Forward- und das Backward-Engineering.
Vorteil des Forward-Engineering ist, dass man bei der Entwerfung der Domäne bereits graphische Darstellungen hat und daraus Code generiert werden kann Dies hat jedoch den Nachteil, dass wenn nun eine Änderung am Code erfolgt, das graphische Model von Hand angepasst werden muss.
Beim Backward-Engineering generiert man anhand des geschriebenen Codes die graphische Repräsentation. Das hat den Nachteil, dass man nicht beim Entwickeln der Domäne eine graphische Repräsentation besitzt, jedoch den Vorteil, dass der Code immer mit der Repräsentation übereinstimmt, ohne von Hand etwas zu modellieren.
Idealerweise soll eine Lösung entstehen, die beide Vorteile der beiden Verfahren in sich vereint. Man soll zuerst die Domäne modellieren können, die graphische Repräsentation jedoch soll auch im späteren Verlauf des Projekts immer mit dem Code übereinstimmen.

2.2.2 Schnellere stabilere Entwicklung

Bisher musste man für verschiedene Software Layer immer die gleiche Domänenstruktur replizieren. Wenn die Modelle nicht exakt übereinstimmten, gab dies Laufzeit Fehler. Dieser Aufwand und der daraus resultierende Anstieg an Entwicklungszeit sind bei jedem Projekt in jeder Phase möglich. Obschon Ascentiv Elements ein sehr effizientes Framework ist, ist dies immer noch ein Overhead, der sich automatisieren lässt. Dadurch sollen Projekte noch schneller und noch stabiler entwickelt werden können und somit die Time-To-Market nochmals verkürzt werden.

2.2.3 Versionensprünge

Wenn Ascentiv Elements oder der verwendete OR-Mapper bei einer neuen Version Breaking Changes beinhaltet, hat dies riesige Auswirkungen auf alle bestehenden Projekte. Besonders bei älteren Projekten zahlt sich die Versionserhöhung nicht immer aus und es muss ggf. von Elements einen Fork weitergezogen werden, wenn eine Änderung anfällt.
Ein Versionssprung soll nun vereinfacht werden, in dem man die Struktur eines Domänen Modells so abstrahiert, dass es stabiler ist als die Struktur von Ascentiv Elements oder des OR-Mappers. So kann ein Domänen Modell problemlos für eine neue Version von Ascentiv Elements generiert werden.

2.3 Aufgabenstellung

Ein Geschäftsmodell soll graphisch in einer UML ähnlichen Sprache abgebildet werden. Es soll jedoch nicht den vollen Abstraktionsgrad von UML Domänen Modellen besitzen, sondern auch technische (design) Repräsentationen zulassen.
Ein graphischer Editor soll mittels dem Microsoft Visual Studio Visualization and Modeling SDK (VSVMSDK) umgesetzt werden. Er soll wie andere UML Editoren per Drag'N'Drop Strukturen frei anordnen können.
Business Objekte sollen frei definierbare Eigenschaften (Attribute) enthalten können. Dabei sollen unterschiedliche Beziehungsarten, Kardinalitäten sowie Vererbung ausgedrückt werden können. Die unterschiedlichen Beziehungsarten sollen grafisch ersichtlich sein. Logik soll keine generiert werden.
Aus dem erstellten Geschäftsmodell sollen nun folgende Software Layer erstellt werden:

- Business Objekte in C#
- Datenbank Struktur in T-SQL
- OR-Mapping Metainformationen in Edmx

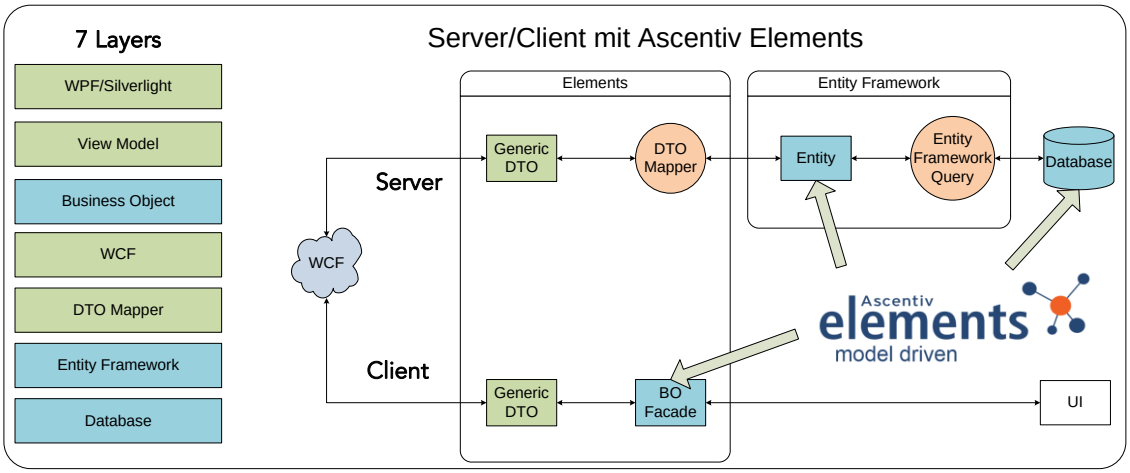


Abbildung 2 – Ascentiv Elements Integration

Die generierten Schichten sollen vom Entwickler ausserhalb des Frameworks erweitert werden können. Vererbungen sollen auf der Datenbank mittels Table Per Type oder Table Per Hierarchy erstellt werden.

3 Konzept

3.1 Meta Modell

Jedes Business Objekt in Ascentiv Elements muss einen Primary Key definieren. Nebst dem können Skalare Eigenschaften (Properties) erstellt werden. Skalare Properties können sowohl konkrete Werte, als auch Enumerationswerte darstellen. Enumerationen sind nur jene auswählbar, die auch in diesem Model definiert sind.
Jedes Business Objekt kann mehrere Relationen zu mehreren Klassen haben. Auch rekursive Relationen sind zulässig.Jede Relation hat eine Source- und eine Target-Kardinalität, sowie einen Source- und einen Target-Rollennamen. N zu N Relationen werden transparent abgebildet, d.h. ohne die auf der Datenbank vorhandene Zwischentabelle. Wenn eine N zu N Beziehung weitere Eigenschaften haben sollen, muss die Zwischenklasse explizit modelliert werden.
Eine Composition ist eine Relation und hat als Source Kardinalität immer 1 oder 0..1. Sie dient zur Beschreibung von Teilen und Ganzen und dient zur Definition von Lebenszyklen. Eine Assoziation ist ebenfalls eine Relation, welche keinen Einfluss auf den Lebenszyklus von Objekten hat und N zu N Relationen erlaubt.
Es kann immer nur von einer Klasse geerbt werden, die Tiefe der Vererbungsstruktur ist beliebeig. Erbende BO's erben auch den Primary Key der Basisklasse.
Eine Enumeration ist eine eigene Klasse, welche einen Enumerationswert beinhaltet. Jede Enumeration hat mehrere Enumerationswerte.

3.2 Graphische Darstellung

Business Objekte werden wie Klassen gemäss UML als Rechtecke dargestellt. Properties werden innerhalb einer Klasse aufgelistet. Relationen können wahlweise Verbindung zweier Business Objekte oder auch als Property dargestellt werden. Wenn eine Relation als Verbindung dargestellt wird, sollen Kardinalitäten sowie Rollennamen zu der Relation angezeigt werden.
Enumeration wird wie das Business Objekt als Rechteck dargestellt und ihre Enumerationswerte in ihr abgebildet. Um einem Enumerations-Property ein Enumerationstyp zuzuweisen wird das allgemeine Parametrierungsfenster (Property Window) verwendet. Weitere Eigenschaften zu den verschiedenen Objekten sind in einem allgemeinen Parametrierungsfenster zu finden.

3.3 Generierungsablauf

Der Generierungsprozess besteht aus sechs Schritten.

1. Anhand des Domänen Modells werden die Business Klassen in C# erzeugt.
2. Anhand des Domänen Modells wird das DDL Script in T-Sql generiert.
3. Das T-Sql Script wird auf einer MS Sql Instanz zur Ausführung gebracht und eine Reference DB wird erstellt.
4. Anhand der Reference DB wird über Reverse Engineering ein Edmx generiert.
5. Anhand des Domänen Modells wird das Edmx angepasst, da das Reverse Engineering aus der Datenbank nicht hinreichend ist.
6. Anhand des Edmx werden die Entity Klassen generiert.

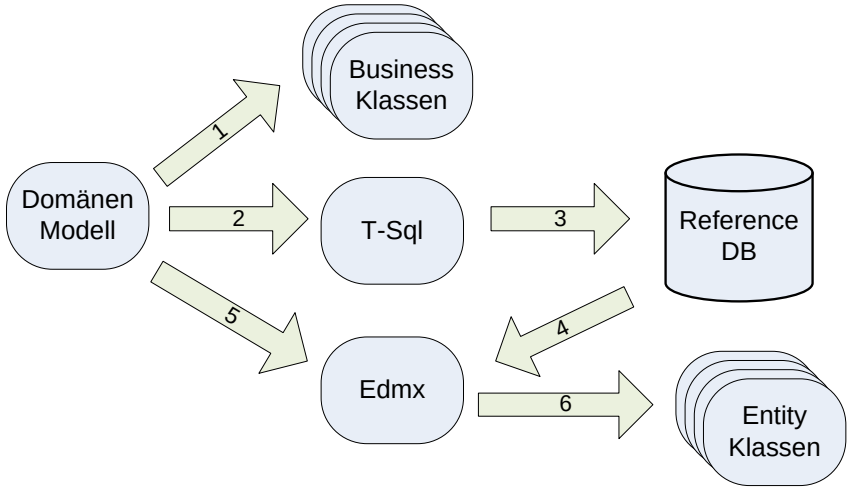


Abbildung 3 - Konzept des Generierungsprozesses

4 Technologien

4.1 VSVMSDK

4.1.1 Übersicht

VSVMSDK ist ein Framework zur Erweiterung des Visual Studios. Es dient dazu, Definitionen für eine graphische Repräsentation eines Models zu erstellen. Als bestes Beispiel dient dazu ein eigens erstellter UML Editor. Dabei übernimmt das Framework vorallem die graphische Repräsentation, sowie das Speichermodel. Die Regeldefinierung zum Model wird vom Entwickler definiert.

4.1.2 T4 Template

T4 ist eine Code Generierungssprache, welche auf „In place substitution" basiert. Es ist eine Mischung aus Textblöcken und Steuerlogik, welche über <# ... #> Statements aktiviert werden. Die Steuerlogik kann wahlweise in C# oder VB.Net geschrieben werden.

```
<table>
    <# for (int i = 1; i <= 10; i++)
       { #>
         <tr><td>Test name <#= i #> </td>
             <td>Test value <#= i * i #> </td> </tr>
    <# } #>
 </table>
```

Abbildung 4 - Beispiel eines T4 Templates für Html

T4 ist Bestandteil des Visual Studios und ist für das VSVMSDK von zentraler Bedeutung.

4.1.3 Entwicklungsprozess

Die Benutzung des VSVMSDK ist ein Bootstrapping Process. Selbst das erstellen der Sprachdefinition für die eigene Domänensprache ensteht bereits durch die selben Schritte wie später das Modellieren der eigenen Domäne.
VSVMSDK enthält die Modelling Assemblies, welche alle nötigen Basistypen für UI, Serializer und Model beinhaltet, sowie die nötigen Projekt Templates. Es ist jedoch nicht nur ein Sammlung von Libraries, sondern das Kernelement ist das Projekt Template. Denn dieses beinhaltet alle T4 Templates um die eigene Dsl Sprache zu erzeugen. Ohne diese Templates wäre dieses Framework deutlich weniger mächtig.
Man definiert in einer Xml Datei die Sprachstruktur, welche der Endbenutzer zur Verfügung hat. Der Code und das Deployment Paket werden nun durch die Templates anhand dieser Xml Datei generiert. Es ist somit bereits dieselbe Architektur, die man benutzt, um eine weitere Dsl Sprache zu entwerfen.

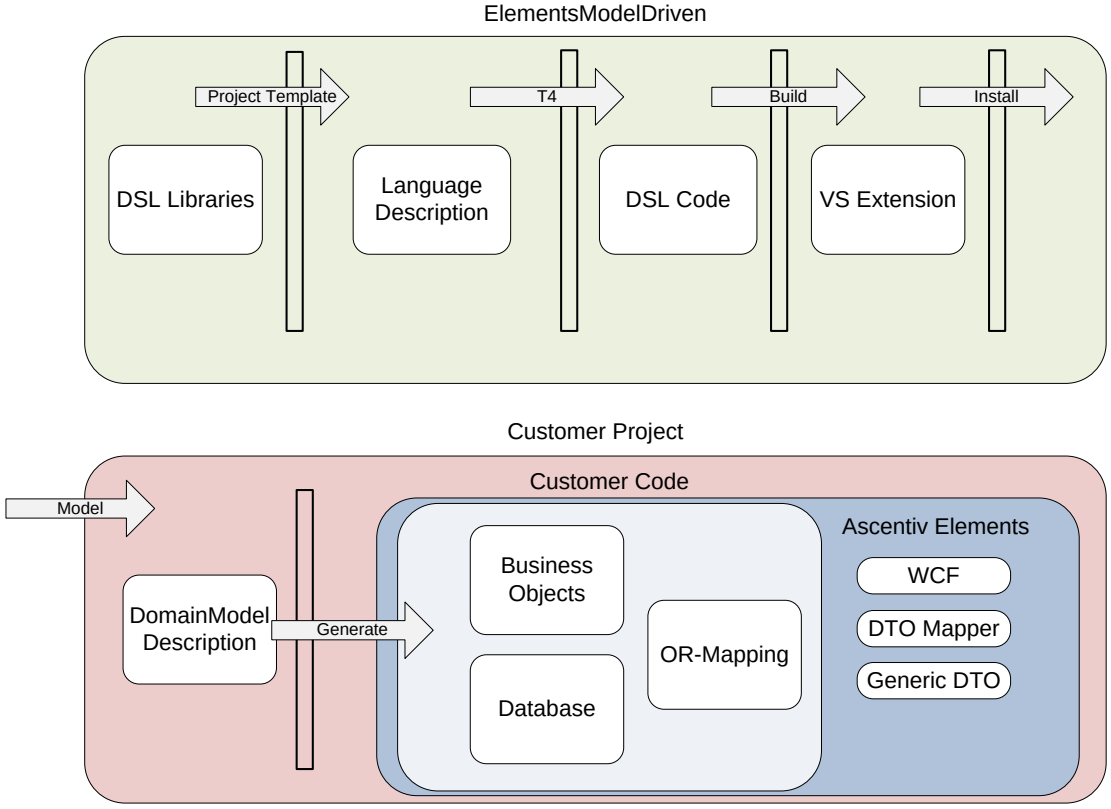


Abbildung 5 : Bootstrapping Process

Die folgende Grafik von Microsoft zeigt deutlich, was Library (violett), was generierter Code (grün) und was selbst geschrieben (blau) ist.

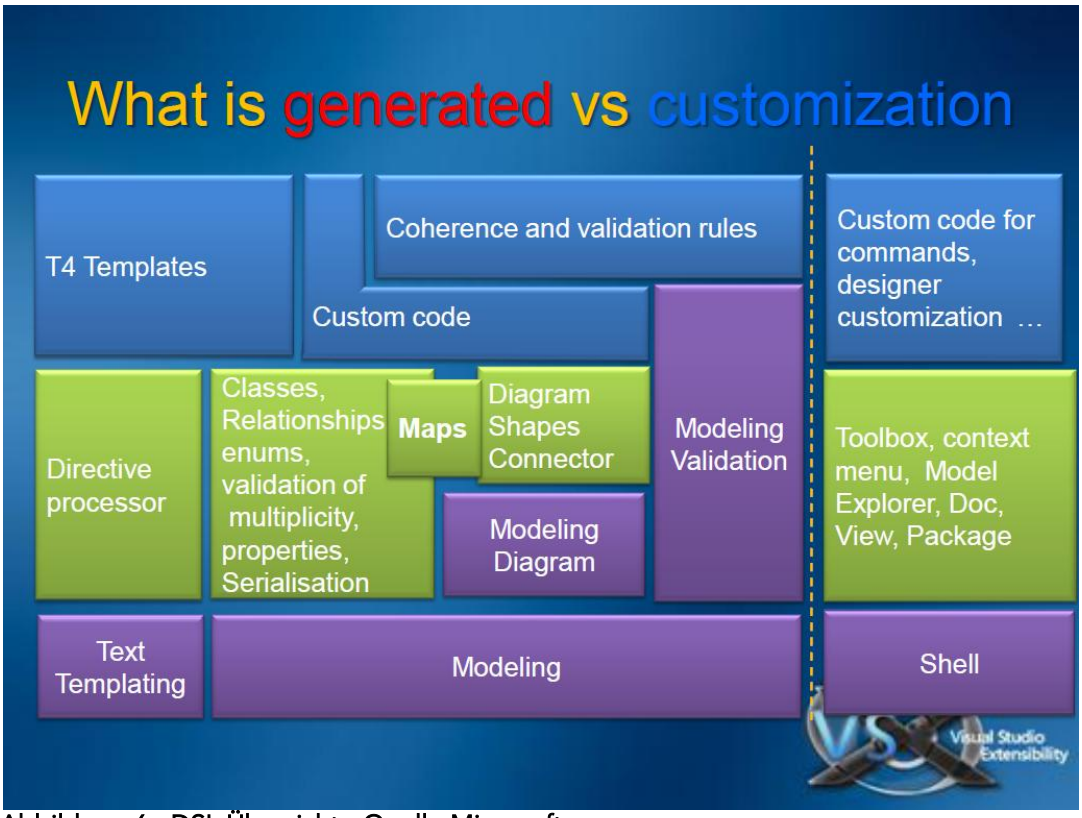


Abbildung 6 : DSL Übersicht - Quelle Microsoft

„Custom Code", sowie „Coherence and validation rules" handeln nur davon, die generierten Klassen zu erweitern.

4.1.4 Sprache Modellieren

Um eine Sprache definieren zu können gibt es eine breite Tool-Unterstützung. Das wichtigste Element dabei ist der Dsl Designer. Im Designer kann man graphisch die Regeln und Abbildungen der Sprache definieren.

4.1.4.1 Model

Im ersten Schritt definiert man das Model, welches persistiert werden soll. Dabei kann man statische Properties hinzufügen, welche die Objekte immer haben sollen.
Es gibt zwei Möglichkeiten Relationen zwischen Klassen darstellen zu können. Entweder geschieht dies über eine EmbeddingRelationship, dann werden in der Persistenz diese Unterobjekte direkt in das übergeordnete Element gespeichert oder über eine ReferenceRelationship, welche dann nur noch eine Referenz im übergeordneten Element hält. Es muss jedoch für jedes Objekt ein EmbeddingRelationship Weg zum Root Knoten vorhanden sein.

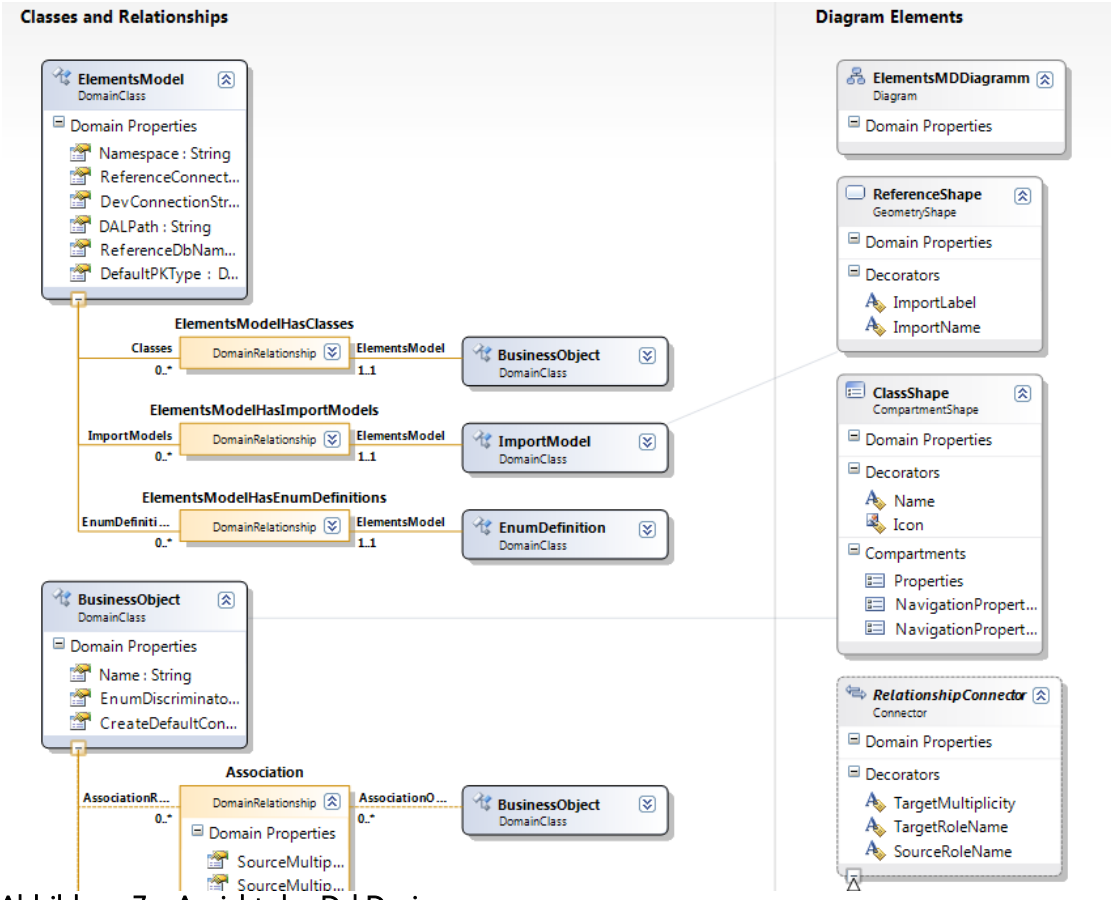


Abbildung 7 – Ansicht des Dsl Designers

Im Dsl definierte Properties sind statisch. D.h. sie werden statisch zum Objekt generiert. Um dynamische Properties einführen zu wollen, muss ein spezifisches DomainClass Objekt modelliert werden, welches das Property darstellt und dieses über eine Relation ins Model einbinden.

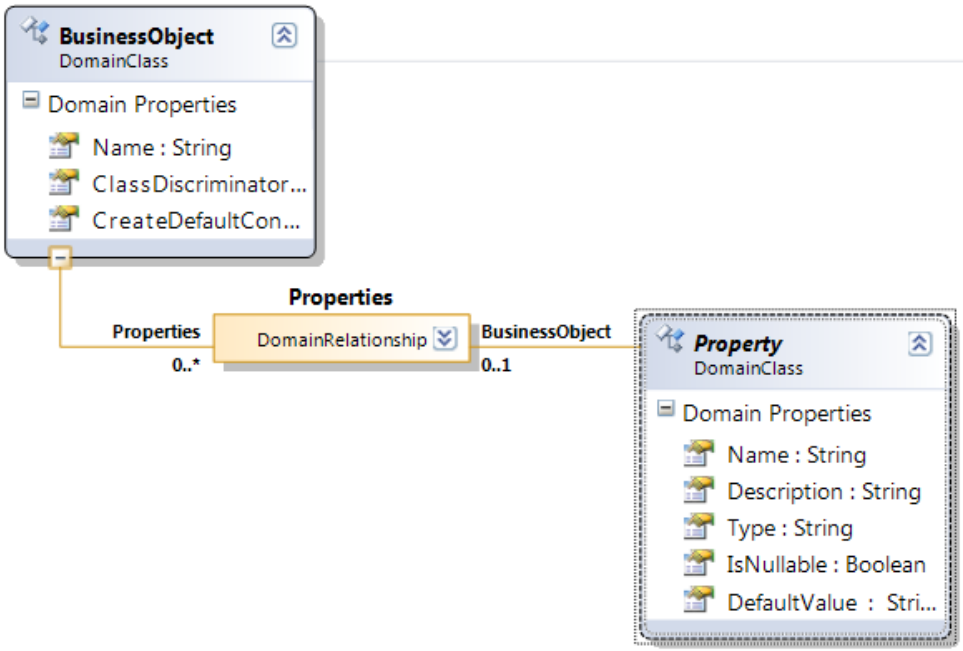


Abbildung 8 - Beispiel für dynamische Properties

Es gibt hier auch die Möglichkeit der Vererbung, welche auf alle Elemente angewendet werden kann.

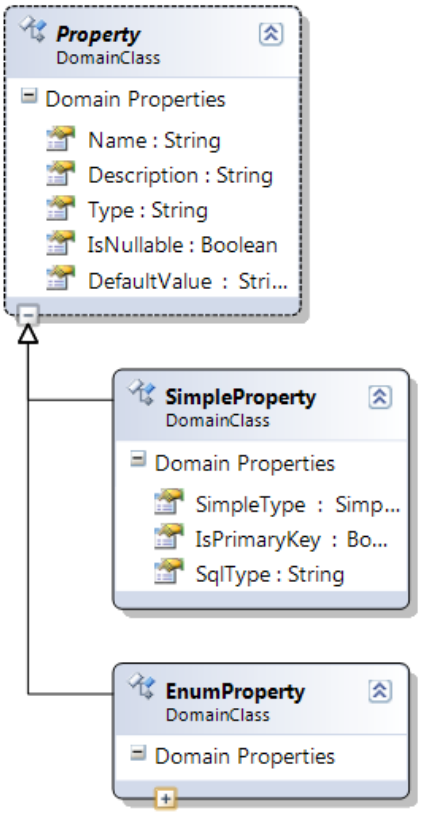


Abbildung 9 - Beispiel einer Vererbung in Dsl Designer

Alles was nicht über den Designer modelliert werden kann, kann sehr gut später erweitert werden. Dabei gibt es eine breite Unterstüzung bei der Code Generierung, später dazu mehr.

4.1.4.2 Shapes

Um das Model nun graphisch abbilden zu können, definiert man Shapes. Oftmals entsprechen die Shapes exakt dem Model, die Darstellung kann jedoch auch abweichen. Der einzige notwendige Shape ist das Diagramm, welches die Rolle des Root Knoten hat. Shapes werden ebenfalls persistiert, werden jedoch in ein *.layout Datei ausgelagert und sind für das Model nicht von Bedeutung. Die zwei wichtigsten Shapes zur Abbildung sind dabei der CompartmentShape und der ConnectorShape. Der CompartmentShape ist eine Objekt/Klassenabbildung, welche eine Liste von Elementen, zum Beispiel Properties, in sich darstellen kann. Der ConnectorShape entspricht der Abildung einer Relation.
Um die graphische Darstellung von Eigenschaften, z.B. statische Properties wie Name, auf einen Shape steuern zu können, gibt es Decorators. Diese müssen dem Shape hinzugefügt, mit einem Model Property gemapped und dann einer Position zugeteilt werden. Die Positionen sind vom Shape statisch vorgegeben und können nicht erweitert werden. Die Möglichkeit der Vererbung steht auch hier zur Verfügung.
Um einen Shape erweitern zu können, zum Beispiel eine weitere Position hinzufügen oder Darstellungseigenschaften weitreichend verändern zu können gibt es keine einfache Möglichkeiten als dies im Modell der Fall ist. Man erhält die Möglichkeit eigene Shapes zu definieren, dies ist jedoch sehr mühselig und nur unter grossem zeitlichem Aufwand möglich. Eine Erweiterung über WPF ähnliches Template Verfahren wäre sehr hilfreich.
Viele Eingeschaften von Shapes sind jedoch genügend parametrierbar, sodass man eine Sprache definieren kann, ohne auf eigene Shapes zurückgreifen zu müssen.

4.1.4.3 Mapping

Das Mapping verbindet nun das Model mit den Shapes. Es bestimmt, wie das Model worin abgebildet wird. Dies kann ebenfalls über den Dsl Designer modelliert werden. Es gibt dabei ein eigenes Visual Studio Window.

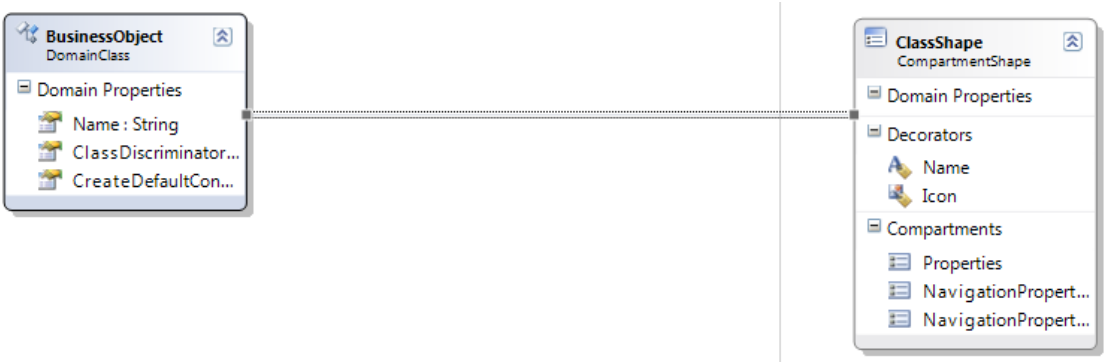


Abbildung 10 - Graphische Abbildung des Mappings im Dsl Designer

Decorator Properties können direkt auf Properties der Domänen Klasse abgebildet werden.

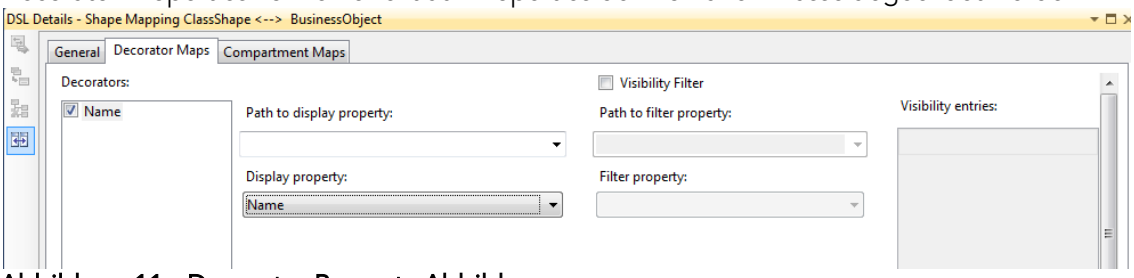


Abbildung 11 - Decorator Property Abbilden

Compartments können anhand eines Pfades auf eine ganze Sammlung von Elementen abgebildet werden. Dies ist zum Beipiel für Properties sinnvoll, um diese direkt in der Klasse darstellen zu können.

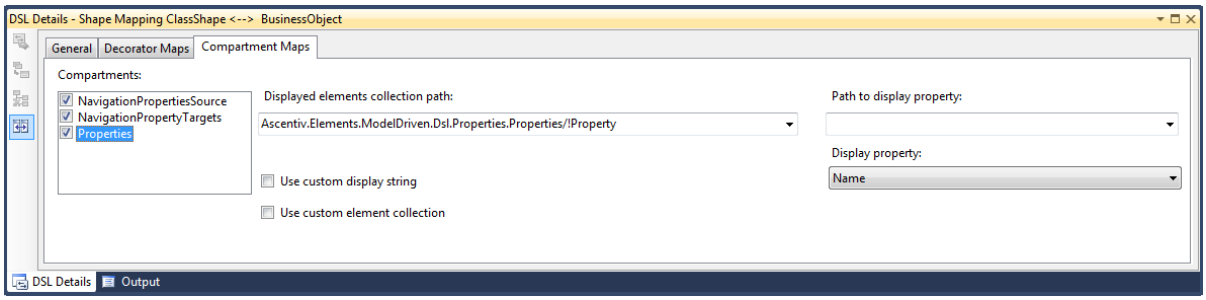


Abbildung 12 - Beispiel eines Compartment-Mappings

4.1.5 Erweiterungsmöglichkeiten

4.1.5.1 Custom Code

Custom Code erweitert die generierten Modell Klassen (DomainClasses, Relationships, Shapes etc.) über partial classes, ähnlich wie dies bei der WinForms Code-Generierung der Fall ist. Bei partial classes kann dieselbe Klasse in mehreren Dateien deklariert werden, worauf der Inhalt der Klassen zusammengefasst wird. Über die Dsl Sprachdefinition kann man dann definieren, was generiert werden soll und was in einer partial class explizit implementiert werden muss. Es gibt auch die Möglichkeit von „Double Derived". Dabei wird neben dem üblichen Basistyp aus der Dsl Library ein generierter Basistyp eingefügt. Dies ermöglicht das überschreiben sämtlicher virtueller Methoden, welche sonst im generierten Code bereits überschrieben werden.

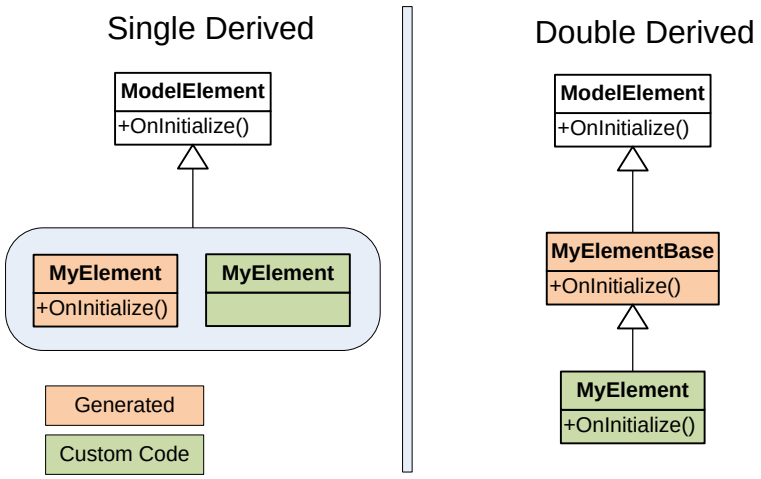


Abbildung 13 - Double Derived Struktur

Die Möglichkeiten zur Anpassung des Codes sind so weitreichend, dass ich mir kein Szenario vorstellen kann, indem man nicht den Dsl Editor benutzen kann um die Sprache zu definieren.

4.1.5.2 Coherence Rules

Nebst dem Custom Code gibt es noch eine weitere, syntaktisch sehr schöne Möglichkeit, Logik in das generierte Model zu bringen. Wenn zum Beispiel Standard-Werte initialisiert werden müssen, welche vom Kontext abhängig sind und somit nicht statisch definiert werden können, kann dies über eine AddRule erfolgen.
Hierfür erzeugt man eine Klasse, die von „AddRule" (aus der Dsl Lib) erbt und die Dummy Methode OnElementAdded überschreibt. Um diese Rule nun einem bestimmten Typen zuzuweisen attributiert man nun diese Klasse mit „RuleOn" und dem entsprechenden Typ als Parameter. Diese AddRule Klasse muss nun noch bei der Model-Initialisierung als Coherence Type angegeben werden.
Neben AddRule existiert auch ChangeRule, welche aufgerufen wird, wenn eine Eigenschaft geändert hat, sowie DeleteRule, wenn ein Element gelöscht wird.

```
[RuleOn(typeof(Generalization))]
public class GeneralizationAddRule : AddRule
{
  public override void ElementAdded(ElementAddedEventArgs e)
  {
    var generalization = e.ModelElement as Generalization;
    generalization.UpdateInheritanceType();
  }
}
```

Abbildung 14 - AddRule Beispiel

4.1.5.3 Validation Rules

Da der Benutzer später bei der Modellierung sehr viele Freiheiten beistzt, ist ein wichtiger Aspekt für die Generierung einer Dsl Sprache ist die Validierung, . Einige Regeln bezüglich der Verknüpfung von Elementen kann bereits im Dsl Editor konfiguriert werden. So zum Beispiel dass nur einfache Vererbung möglich ist und dass jeder Klassenname einzigartig sein muss.
In komplexeren Fällen jedoch muss eigene Validierung helfen, das Model konsistent zu halten. Es gibt dabei zwei Fälle, wie die Validierung sich auswirkt. Entweder wird über eine MessageBox gemeldet, dass ein Wert nicht gültig ist und der Stand vor der Aktion wird wiederherstellt, oder es wird beim Speichern oder Kompilieren eine Compile Error in der Error List erscheinen.
Eine MessageBox wird erzeugt indem man in der entsprechenden Rule eine Exception wirft.

Einen Compile Error erzeugt man, indem man eine ValidationMethod einführt.
Eine ValidationMethod kann überall in den Model Klassen eingefügt werden, die Sichtbarkeit der Methode spielt dabei keine Rolle. Einzig wichtig ist die passende Signatur und eine entsprechende Attributierung. Attributiert wird mit „ValidationMethod“ und als Parameter die verschiedenen Zeitpunkte, in welchen validiert werden soll (beim Laden, Speichern etc.). Als Parameter erhält man den ValidationContext, in welchem man Validierungsfehler eintragen kann.

```
[ValidationMethod(ValidationCategories.Save | ValidationCategories.Open)]
private void CheckEmptyName(ValidationContext context)
{
  if (string.IsNullOrWhiteSpace(this.Name))
    context.LogError("Property name must not be empty", ValidationCodes.Elmd002Mis
singPropertyName.ToString(), this);
  var regex = new Regex(CSharpNameRegex);
  if (!regex.Match(this.Name).Success)
    context.LogError("Property name is not in correct format", ValidationCodes.Elm
d016FalsePropertyName.ToString(), this);
}
```

Abbildung 15 - ValidationMethod Beispiel

Mit der bereits beschriebenen „Double Derived“ Methode sind alle Coherence und Validation Rules ebenfalls möglich, jedoch verliert man viel Abstraktion und die Implementierung wird einiges komplexer.

4.1.6 Deployment

Wenn ein VSVMSDK Projekt erstellt wurde, ist ein VSIX Deployment bereits eingerichtet. Dabei wird bei jedem Build eine VSIX Datei erstellt. Wenn ein Visual Studio Projekt ein VSIX Paket erstellt, steht im Property Window zu den Files ein weiteres Flag, „Include in VSIX“. Mit diesem Flag kann man steuern, welche Dateien in das VSIX gepackt werden soll. Generierte Assemblies hingegen werden wiederum über eine Manifestdatei angegeben.
Das VSIX Deployment hat den Vorteil der Einfachheit und dass es in eine experimentelle Instanz eines Visual Studios deployed werden kann. Somit ist es eine gute Entwicklungsform. Will man die Dsl Sprache jedoch auslieferen, ist dies ungeeignet, da es nicht genug mächtig ist. Auf http://msdn.microsoft.com/en-us/library/dd393694.aspx gibt es eine gute Übersicht über die verschiedenen Möglichkeiten des Deployments und desssen Fähigkeiten.
Dabei gibt es in VSIX zwei Dinge die nicht erledigt werden können, jedoch sehr erwünscht wären. Man will mit der Dsl Sprache auch gerne die XSD Datei der Sprache ausliefern. Dies erlaubt die Xml Validierung der eigenen Domänensprache. Diese Datei muss in den Visual Studio Installationspfad eingefügt werden.
Eine weitere Eigenschaft ist die Registrierung des neu erstellten Dateityps mit einem Icon und der korrekten Registrierung zum Visual Studio. Da dies nicht im Visual Studio registriert werden muss, sondern im Windows Explorer, genügt VSIX auch hier nicht.
Deswegen schlägt Microsoft die Kombination von VSIX und MSI vor, denn MSI ergänzt alle fehlenden Funktionalitäten. Dabei wird VSIX nur noch zur Entwicklung verwendet und zur Auslieferung wird ein MSI Paket erstellt, welches neben dem VSIX Deployment noch die genannten Aufgaben übernimmt.

4.1.6.1 Code Generator

Ein VSVMSDK Projekt beinhaltet noch keinen Code Generator. Dieser muss zuerst noch erstellt und mit deployed werden. Dazu erstellt man im Dsl Package File ein pkgdef Datei mit den nötigen Einträgen zum Code Generator und verlinkt diese Datei dann im vsixmanifest, welches die Beschreibung zum VSIX Paket enthält.
Dazu benötigt man eine Ableitung der TemplatedCodeGenerator Klasse im VSIX Projekt. Diese muss mit einer InteropServices Guid versehen werden, damit diese im pkgdef File referenziert werden kann. Von dieser abgeleiteten Klasse wird nun bei jedem Speichern des Editors die Funktion „GenerateCode“ aufgerufen. Deswegen wird diese Funktion nun

überschrieben und ein wenig abgeändert. Das T4 File, welches zur Code Generierung benötigt wird, wird nun aus den Assembly Ressourcen geladen und zur Ausführung gebracht.

```
[global::System.Runtime.InteropServices.Guid("9799A402-F0A3-48ee-B103-
2223E74553A4")]
public class CodeGenerator : TemplatedCodeGenerator
{
  protected override byte[] GenerateCode(string inputFileName, string inputFileCon
tent)
  {
    // Replace the supplied file contents with the template we want to run
    inputFileContent = ASCIIEncoding.UTF8.GetString(CodeGenerationResource.Templat
e);

    // Substitute the name of the current model file into the template.
    FileInfo fi = new FileInfo(inputFileName);
    inputFileContent = inputFileContent.Replace("LineCreate-
Behavior.sm", fi.Name);
    inputFileContent = inputFileContent.Replace("namespace ShapesTest", "namespace
 " + FileNamespace);

    // Now just delegate the rest of the work to the base class
    byte[] data; data = base.GenerateCode(inputFileName, inputFileContent);
    byte[] ascii = new byte[data.Length -
 3]; Array.Copy(data, 3, ascii, 0, data.Length - 3);
    return ascii;
  }
}
```

Abbildung 16 – Beispiel eines CodeGenerators aus den VSVMSDK Labs

4.2 Entity Framework

Das Entity Framework ist ein Objekt Relationaler Mapper (ORM) für .Net, welcher ein relationales Datenbank Modell in ein Objekt Modell laden kann, ohne selbst Sql Code schreiben zu müssen.
Damit dies funktioniert, muss dem Framework die Struktur der Datenbank, die Struktur des Objekt Models und das Mapping zwischen diesen Strukturen bekannt sein. Dieses Metamodell wird in einem Edmx File abgelegt. Jedes Edmx besteht aus vier Teilen:

- StorageModel: Beschreibt das Datenbank Schema
- ConceptualModel: Beschreibt, welche Entitäten wie generiert werden. Dabei werden Name, Typen, Vererbung, Assoziationen, Kardinalitäten etc. für das Objekt Model bestimmt.
- Mapping: Welche Eigenschaft auf dem Conceptual Model gehört zu welcher Eigenschaft auf dem Storage Model.
- Layout: Designer darstellungsspezifische Eigenschaften. Optional.

Anhand des ConceptualModels wird der .Net Code der Entitäten generiert. Das Entity Framework unterstützt das graphische Modellieren, ‚Reverse Engineering' von Datenbanken, Kaskadierung, N zu N Relationen und Vererbung.
Weiete Informationen unter http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx

4.3 Ascentiv Elements

4.3.1 Übersicht

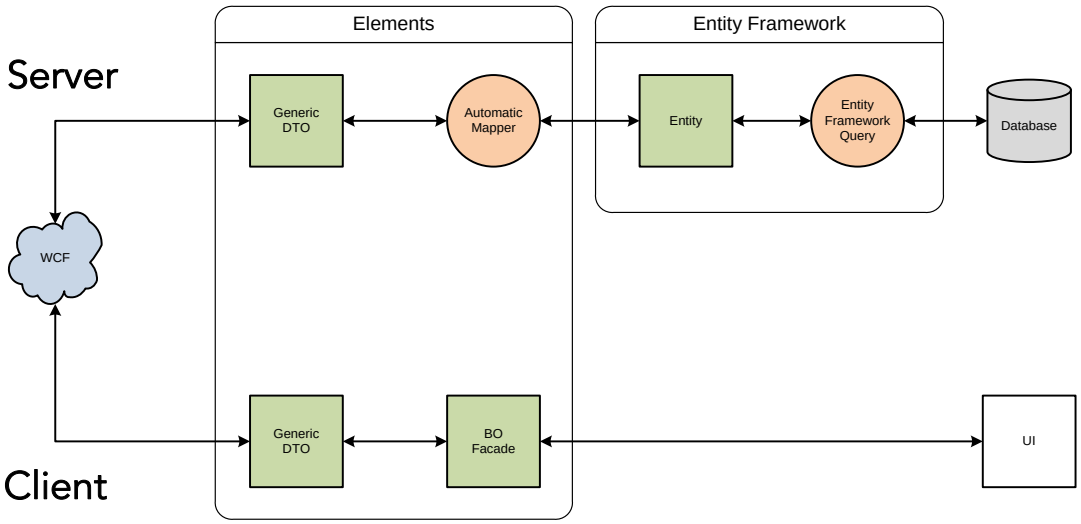
Ascentiv Elements ist ein Framework der Firma Ascentiv, welches die Kommunikation und Datenspeicherung in einer Server Client Applikation vereinfacht und generalisiert. Der Weg, den Daten von Datenbank bis hin zum User Interface zurücklegen ist bei sehr vielen Projekten immer gleich. Ohne Ascentiv Elements:

- definiert man die Datenbank

- generiert das EF Mapping
- mapped EF Entities zu DTO's
- erstellt für jedes Objekt eine passende Kommunikationsschnittstelle
- generiert auf der Client Seite einen entsprechenden Proxy
- erstellt für jede Domänen Entität eine BO Facade
- mapped die empfangenen DTO auf eine BO Facade
- bindet die BO Facade an ein User Interface

Ascentiv Elements bietet nun eine generische Kommunikations Schnittstelle, welche den Weg von Server zu Client so vereinfacht, dass wir im Client nur eine BO Facade anfordern müssen und der Weg von der Datenbank bis hin zur BO Facade wird automatisch zurückgelegt.
Mit Ascentiv Elements:

- definiert man die Datenbank
- generiert das EF Mapping
- erstellt für jede Domänen Entität eine BO Facade
- bindet die BO Facade an ein User Interface



4.3.2 BO Facade

Eine BO Facade erbt von "BusinessObject" und muss einer Entität im Entity Framework entsprechen. Das heisst, alle Eigenschaften müssen gleich benannt sein, Kardinalitäten müssen gleich sein und der Entitäten Name muss über eine bestimmte Konvention gleich sein. Eine BO Facade benötigt einen parameterlosen Konstruktor und einen, der einen „GenericDTO" entgegen nimmt. Eine Eigenschaft auf der Facade besteht aus einem statischen Feld, welches das Property beschreibt und einem Property, welches anhand dieser Property Definition Werte schreibt oder setzt. Dies stellt sicher, dass alle Werte in den unterliegenden DTO geschrieben oder von dort gelesen werden.

```
[CustomBusinessObject(IsCachedOnClient = true)]
public class Contact: BusinessObject
{
  public Contact()
  {}

  public Contact(GenericDTO dto): base(dto)
  {}

  private static readonly StructPropertyDef<Contact, Guid> _keyPropDef =
    new StructPropertyDef<Contact, Guid>(bo => bo.ContactKey, EPropertyKind.PrimaryKey);
  public Guid? ContactKey { get { return GetValue(_keyPropDef); } set { SetValue(_keyPropDef, value); } }

  private static readonly PropertyDef<Contact, String> _firstNamePropDef =
    new PropertyDef<Contact, String>(bo => bo.FirstName) { VisibleKeyInfo = new VisibleKeyInfo() { Index = 0, Postfix = " " } };
  public String FirstName { get { return GetValue(_firstNamePropDef); } set { SetValue(_firstNamePropDef, value); } }

  private static readonly PropertyDef<Contact, String> _lastNamePropDef =
    new PropertyDef<Contact, String>(bo => bo.LastName) {VisibleKeyInfo = new VisibleKeyInfo(){Index = 1} };
  public String LastName { get { return GetValue(_lastNamePropDef); } set { SetValue(_lastNamePropDef, value); } }
}
```

Abbildung 17 - Beispiel für ein BusinessObject

4.3.3 Konventionen

Ascentiv Elements enthält Konventionen, die eingehalten werden müssen um die Lauffähigkeit zu garantieren. Wenn ein Projekt mit Ascentiv Elements kompiliert bedeutet dies nicht, dass es auch läuft.

- Ein BO Typ muss im EF einen entsprechenden Entity Typ haben, der für den Typ Namen die Konvention ‚<BOName>Entity'und für den Entity Set Typ Namen die Konvention ‚<BOName>EntitySet' einhält.
- Properties müssen sowohl im BO sowie im EF gleich benannt sein.
- Die Kardinalitäten im EF müssen mit denen der BO's übereinstimmen.
- Ein BO muss einen Primärschlüssel definieren.
- Will man einen eigenen Accessor für das EF definieren, muss dieser eine Namenskonvention einhalten, von einem bestimmten Typ ableiten und in einem per Metadaten spezifizierten Pfad liegen.
- Assemblies, die BO's enthalten, müssen beim Applikationsstart über die Metadaten registriert werden.

5 Arbeitsprozess mit ElementsMD

5.1 Graphisches Modellieren

Um eine Domäne in der Umsetzung graphisch darzustellen, gibt es bei der Benutzung von Ascentiv Elements mehrere Varianten. Man könnte sich zum Beispiel das Datenbank Diagramm anschauen. Dieses ist jedoch sehr technisch und Kardinalitäten sind nicht vollständig rekonstruierbar. Weiter könnte man den Edmx Designer benützen. Die Übersichtlichkeit leidet jedoch an den ungerichteten Relationen, an fehlenden Möglichkeiten in der Vererbung und an fehlenden Enumerationen. Die letzte Möglichkeit wäre der Standard Klassen Diagramm Designer des Visual Studios. Dieser bildet jedoch neben den Domäneneigenschaften noch zahlreiche technische Details an, welcher die Domäne schnell unübersichtlich erscheinen lässt. Somit ebenfalls ungeeignet, um evt. mit einem Kunden darüber sprechen zu können.

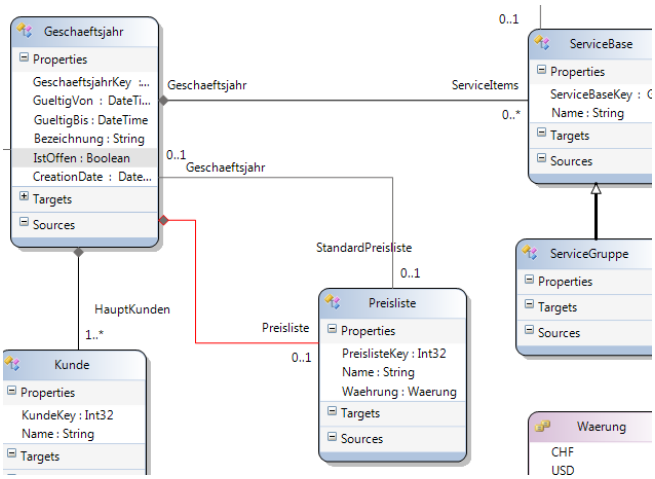


Abbildung 18 - Domäne als Diagramm

Somit benötigt es eine graphische Darstellung der Domäne, die nicht überladen ist, nur die einfachsten technischen Details Preis gibt, gerichtete Relationen erlaubt, Enumerationen unterstützt und dies alles mit der Unterstützung für Ascentiv Elements.

5.2 Entwicklungsprozess bisher

Der Benutzer des Ascentiv Elements Frameworks ist immer wieder mit der Aufgabe konfrontiert, das Business Model zu ändern. Dies ist vorallem in den frühen Phasen, aber auch weiterhin im späteren Verlauf eines Projekts der Fall. Die Prozedur ist immer gleich und leicht fehleranfällig:

- Verändern der Datenbank: hier entstehen schon die ersten Stolpersteine, da teilweise Tabellen zuerst gelöscht und dann wieder erstellt werden müssen. Aufgrund der Abhängigkeiten der Tabellen kann dies recht aufwendig werden.
- Verändern des OR-Mapping: die Edmx Datei des EntityFrameworks muss auf das neue DB Schema aktualisiert werden. Hier entstehen am meisten Probleme, da EF nicht sehr gut mit allen Änderungen klar kommt und auch sonst seine Eigenheiten aufweist. Mehr dazu in **Error! Reference source not found. Error! Reference source not found.**.
- Verändern der BusinessObjects: jede Änderung muss auscodiert werden, um die gleichen Eigenschaften wie die des Edmx zu haben. Hier entstehen leicht Schreibfehler, Eigenschaften gehen vergessen oder nicht mehr existierende Altlasten bleiben erhalten.

Schnell sieht man, dass das Prozedere immer gleich ist und immer die gleichen Informationen in den verschiedenen Technologien umgesetzt werden. Und hier setzt Elements Model Driven an. Eine Änderung am Model wird graphisch einmal vorgenommen und alle weiteren Schritte, die vorher von Hand ausgeführt werden mussten und dem Entwickler Zeit und Nerven kosteten, sind bereits ausgeführt.

5.3 Entwicklungsprozess mit ElementsMD

Es arbeiten mehrere Entwickler über eine Source Control an einem Projekt. Alle haben die Möglichkeit, das Model zu verändern. Es gibt eine gemeinsame Entwicklerdatenbank, gegen welche entwickelt wird und jeder Entwickler hat eine lokale Instanz eines Datenbank Services. Developer 1 will nun ein Attribut der Entität Person hinzufügen. Er öffnet das Elmd (die

Domänenbeschreibung) und fügt beim BusinessObject Person eine Property hinzu. Er speichert.
Nun wird auf seinem Rechner eine neue Referenzdatenbank erstellt, daraus wird das Edmx erzeugt und angepasst und der Code wird geschrieben. Will er seine Änderungen gleich auch auf der Developer Database sehen, dann öffnet er ein von Elements MD bereitgestelltes Sql Compare File, welches ihm gleich anzeigt, welche Änderungen an der Datenbank übertragen werden. Alternativ dazu kann er zuerst gegen seine Referenzdatenbank implementieren und später seine Änderungen in die Developer Database eingeben. Dies kann sinnvoll sein, wenn seine Änderungen längere Entwicklungszeit in Anspruch nehmen. Denn eine veränderte Entwicklerdatenbank ohne die veränderten BO's und das Edmx wird zur Laufzeit Fehler erzeugen und das Entwickeln für alle anderen Entwicklern erschweren.
Egal welche Datenbank er gewählt hat, er kann jetzt gleich sein UI um dieses Attribut erweitern und Testen. Wenn die gewünschten Änderungen abgeschlossen sind, speichert er sein Elmd in der Source Control ab, wo andere Entwickler dieses wiederum mit denselben Schritten verändern können.

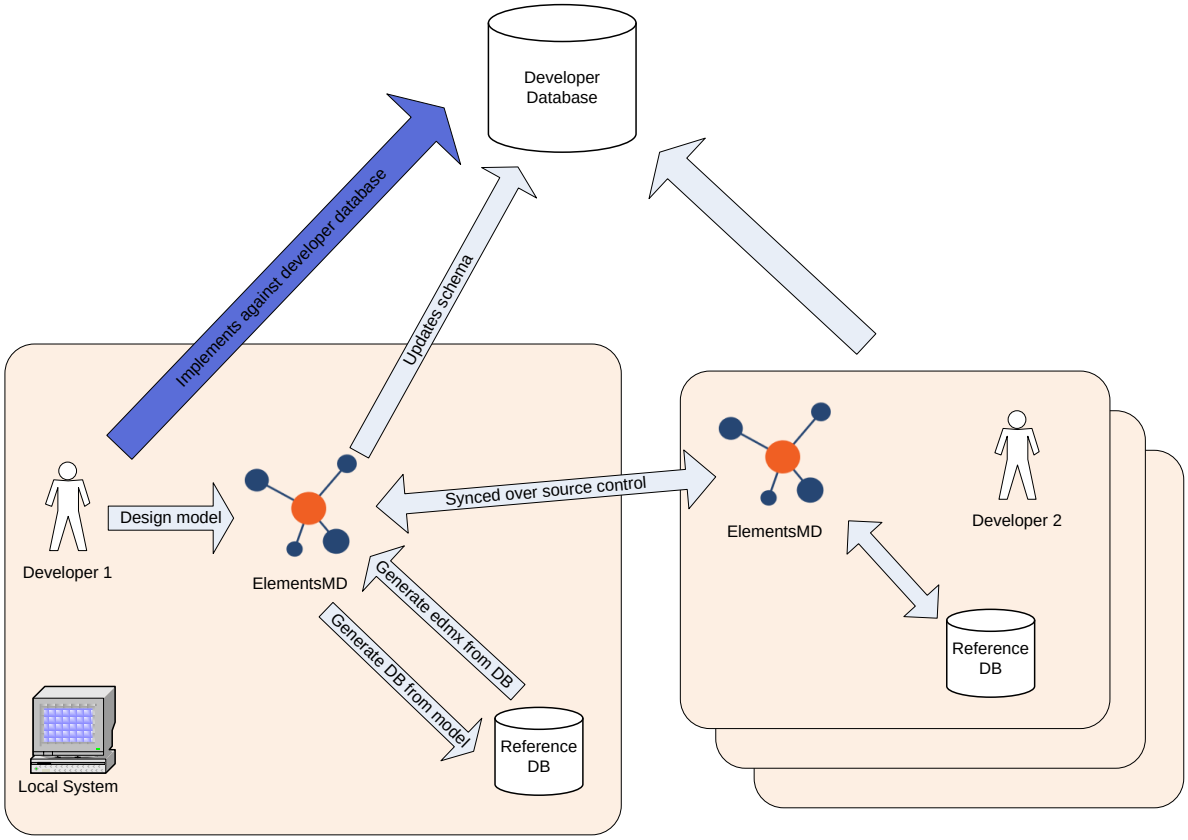


Abbildung 19 - Projekt Arbeitsablauf

Wenn sich das Projekt in einer späteren Phase befindet und eine erste Version des Projekts bereits in Betrieb ist, können anhand des Sql Compare Files die nötigen Änderungen am Model in ein Sql Script gepackt werden um mit der nächsten Auslieferung auf den produktiven Server geladen werden.

Ascentiv AG
Binzstrasse 18
CH-8045 Zürich

T +41 (0)44 455 62 80
F +41 (0)44 455 60 69
info@ascentiv.ch
www.ascentiv.ch

Design

Elements Model Driven

Marco Bachmann
mbachmann@ascentiv.ch

Versionen

Datum	Version	Kommentar	Autor
08.10.10	0.1	Erstellt	mb
29.11.10	0.2	Struktur erstellt	mb
06.12.10	0.3	Kapitel gefüllt	mb
13.12.10	0.4	Schlussfolgerungen	mb
23.12.10	1.0	Release	mb

Inhaltsverzeichnis

1 Einleitung 3
1.1 Zweck des Dokuments 3
1.2 Glossar 3
2 Tool Architektur 4
2.1 Übersicht der Komponenten 4
2.1.1 Dsl 4
2.1.2 DslPackage 4
2.1.3 ModelBusAdapter 4
2.1.4 EdmGen2 4
2.1.5 LinqToEdmx 4
2.2 Dsl Sprachdefinition 5
2.2.1 ElementsModel 5
2.2.2 BusinessObject 6
2.2.3 EnumDefinition 7
2.2.4 EnumEntry 7
2.2.5 Property 7
2.2.6 SimpleProperty 8
2.2.7 EnumProperty 8
2.2.8 Relationship 9
2.2.9 Composition 9
2.2.10 Association 10
2.3 Generierungsprozess 10
2.3.1 Template Manager 12
2.3.2 Code Generator 12
2.3.3 Sql Generator 12
2.3.4 DbManager 12
2.3.5 Edmx Generator 12
3 Test Projekt – ServiceCms 16
3.1 Zweck 16
3.2 Projektstruktur 16
3.3 Die Domäne 18
3.3.1 Abgedeckte Fälle 19
3.3.2 Nicht abgedeckte Fälle 19
3.4 Strategie 20
4 Schlussfolgerungen 21
4.1 Erreichte Ziele 21
4.1.1 Modellierung 21
4.1.2 Generierung 21
4.2 Fehlende Funktionalität 22
4.2.1 Fehlende Ascentiv Elements Unterstützung 22
4.2.2 Fehlende Funktionalität 23

1 Einleitung

1.1 Zweck des Dokuments

Das Zielpublikum dieses Dokuments sind Software Architekten. Dieses Dokument beschreibt die Umsetzungen, Anwendungsfälle und das Test Projekt von Elements Model Driven. Es ist ein Design Dokument für das ganze Projekt und basiert auf den Konzepten des technischen Berichts.

1.2 Glossar

Dsl	Domain Specific Language, eine Beschreibungssprache (in xml) um weitere Beschreibungssprachen zu definieren.
VSVMSDK	Visual Studio Visualization and Modeling SDK, die neue Bezeichnung für Dsl.
T4 Template	T4 ist eine Code Generierungssprache, welche auf „In place substitution" basiert.
Ascentiv Elements	Ein Framework der Firma Ascentiv, welches dazu dient, den Weg von der Datenbank auf dem Server bis hin zum Client zu vereinfachen.
C#	Eine Objektorientierte Programmiersprache von Microsoft.
Containment	Eine Relation zwischen zwei Objekten, welche eine Teil/Ganzes Beziehung beschreibt
Entity Framework	Ein objekt-relationaler Mapper für .Net
Edmx	Eine Beschreibung (in xml) für ein objekt-relationales Mapping im Entity Framework.
TTH	Table per Hierarchy. Eine Umsetzung für Generalisierung auf der Datenbank, wobei eine Tabelle für alle Klassen der Vererbungsstruktur erstellt wird und anhand eines bestimmten Attributs auf den effektiven Typ geschlossen werden kann.
TTP	Table per Type. Eine Umsetzung für Generalisierung auf der Datenbank, wobei jede Klasse in der Vererbungsstruktur eine eigene Tabelle erhält erhält und diese Tabellen über einen gemeinsamen Primärschlüssel verbunden sind.
BO	Business Objects sind Objekte basierend auf AscentivElements, welche eine Entität einer Domäne wiederspiegelt.
DTO	Data Transfer Objebts sind Objekte, die über eine Datenleitung geschickt werden können.
WCF	Windows Communication Foundation ist eine Service orientierte Übertragungsschnittstelle im Microsoft .Net Framework.
ElementsMD	Elements Model Driven
Elmd	Elements Model Driven Domänen Beschreibung
MS-PL	Microsoft Public License, Mirosofts Umsetzung der GPL (Open Source Lizenz)
CLR	Laufzeitumgebung für .Net Programme.
Attributierung	Eine Technik in .Net Sprachen um Typen statisch mit Metadaten zu versehen, welche über Reflection ausgewertet werden können.
Connection String	Ein Zeichenkette, welche alle Informationen enthält um mit einer Datenbank verbinden zu können.
Sql Compare File	Eine Datei die Informationen enthält, wie zwei Datenbanken zu vergleichen sind.
VSIX	Installationspaket für eine Visual Studio Erweiterung
DAL	Data Access Layer, die Programmschicht, welche die Datenbank anspricht.

2 Tool Architektur

2.1 Übersicht der Komponenten

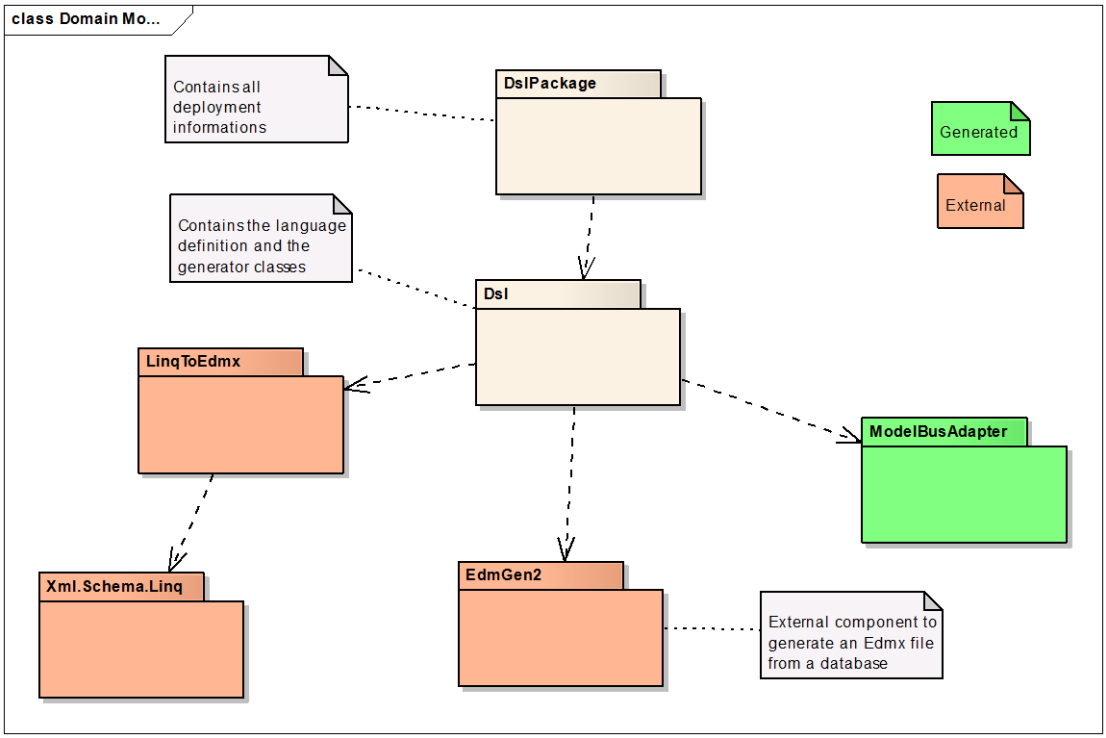


Abbildung 1 - Komponenten Übersicht

2.1.1 Dsl

Dieses Paket beinhaltet:

- Domänen Definition (Dsl Datei)
- Generierter Dsl Code
- Custom Code
- Validierung
- Coherence
- Elements Code Generator Klassen
- Editor Icons

2.1.2 DslPackage

In diesem Paket sind alle Informationen für das Deployment und das T4 Template, aus welchem der Code generiert wird.

2.1.3 ModelBusAdapter

Der ModelBusAdapter ist für spätere Funktionen wie die Modularisierung von zentraler Bedeutung. Er wird jedoch generiert anhand der Dsl Sprachdefinition und ist im Moment noch ungenutzt.

2.1.4 EdmGen2

Ist eine Drittkomponente, welche erweiterte Möglichkeiten (siehe 2.3.5 Edmx Generator) zur Edmx Generierung anhand einer Datenbank ermöglicht als die originale EdmGen.exe. Diese Komponente steht unter MS-PL und ist als Source Code und nicht als Assembly in das Projekt eingebunden.

2.1.5 LinqToEdmx

Um die Edmx Datei auf abstraktere Art und Weise zu verändern wird LinqToEdmx verwendet, ein Codeplex Projekt (zu finden auf linqtoedmx.codeplex.com). Es stellt eine typisierte Benutzung von LinqToXsd dar, welches ebenfalls auf Codeplex zu finden ist. Beide Projekte sind als Assemblies in das Projekt eingebunden und stehen unter MS-PL.

```
var edmx = Edmx.Load(@"Northwind.edmx");

                            // From the mappings
var mappingToConceptualJoin = from entityTypeMapping in edmx.GetItems<EntityTypeMapping>()
                            // From the conceptual model
                            from entityType in edmx.GetItems<EntityType>()
                            // Get the namespace of the conceptual model entities
                            let entityNamespace = edmx.GetItems<ConceptualSchema>().First().Namespace
                            // Join the two models by the fully qualified type name
                            where entityTypeMapping.TypeName == entityNamespace + "." + entityType.Name
                            select new
                                    {
                                      // The type name from the conceptual model
                                      entityType.Name,
                                      // The fully qualified type name from the mapping model
                                      entityTypeMapping.TypeName
                                    };
```

Abbildung 2 - Beispiel einer LinqToEdmx Query

2.2 Dsl Sprachdefinition

Die Sprachdefinition eines VSVMSDK Projekts ist in einer dsl Datei abgespeichert. Diese Definition wiederspiegelt nun die Struktur und Abbildung eines Diagrams. Um nun in Elements Model Driven eine Domäne zu modellieren benutzen wir nun eine eigene Dateiendung elmd. Diese elmd Datei beinhaltet ein konkretes Domänenmodel, welches anhand der Sprache strukturiert wurde, welche im dsl definiert ist.

2.2.1 ElementsModel

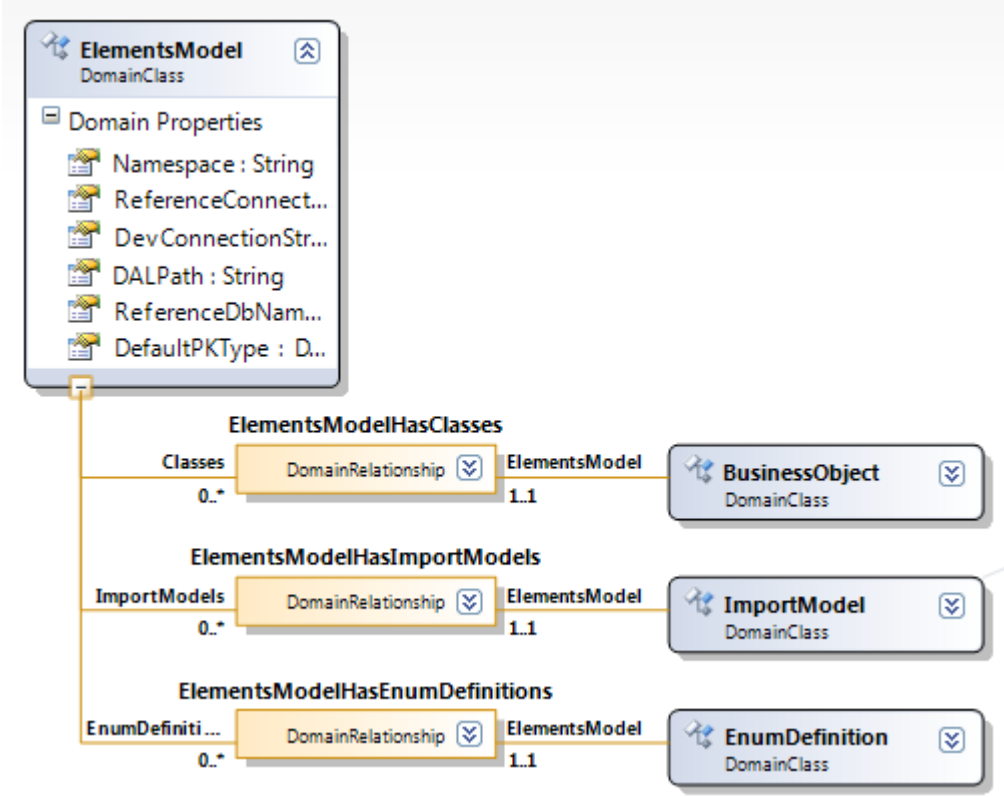


Abbildung 3 - ElementsModel Struktur

Das ElementsModel ist der Hauptknoten des Diagramms. Ihm unterliegt jedes weitere Model.
Properties:

- Namespace: Der Namespace in welcher der Code generiert wird.

- ReferenceConnectionString: Beinhaltet den Connection String zu einer Sql Server Instanz (und **nicht** zu einer Datenbank). Die Referenzdatenbank dient dazu, den Sql Code zu überprüfen und das Edmx zu erzeugen.
- DevConnectionString: Beinhaltet einen Connection String auf die Entwickler Datenbank. Dieser wird benutzt um das Sql Compare File fertig zu stellen.
- DALPath: Da der Data Access Layer typischerweise nicht bei den Business Objekten liegt, müssen jene Dateien, welche zur Datenbank gehören (sql, scmp, edmx etc.), nicht an der Stelle der elmd Datei liegen. Dafür gibt man diesen Pfad explizit an. Der Pfad ist folgendermassen aufgebaut: „<ProjektName>:\<OptionalerPfad>\"
  Dabei kann „OptionalerPfad" vorhanden sein oder nicht. Der Pfad muss dabei als Ordnerunterteilung einen Backslash verwenden. Beispiel eines DALPath: „ServiceCms.Dal:\DB\".
  Ist der dieser Pfad nicht gesetzt wird eine Warnung ausgegeben und die Datenbankdateien werden unter dem elmd gespeichert.
- ReferenceDbName: Der Name der Referenzdatenbank, welche erstellt werden soll.

Relationen:

- Classes: Alle Klassen in diesem Diagramm
- ImportModels: Alle Importe von anderen Diagrammen, nicht implementiert.
- EnumDefinitions: Alle EnumDefinitionen

2.2.2 BusinessObject

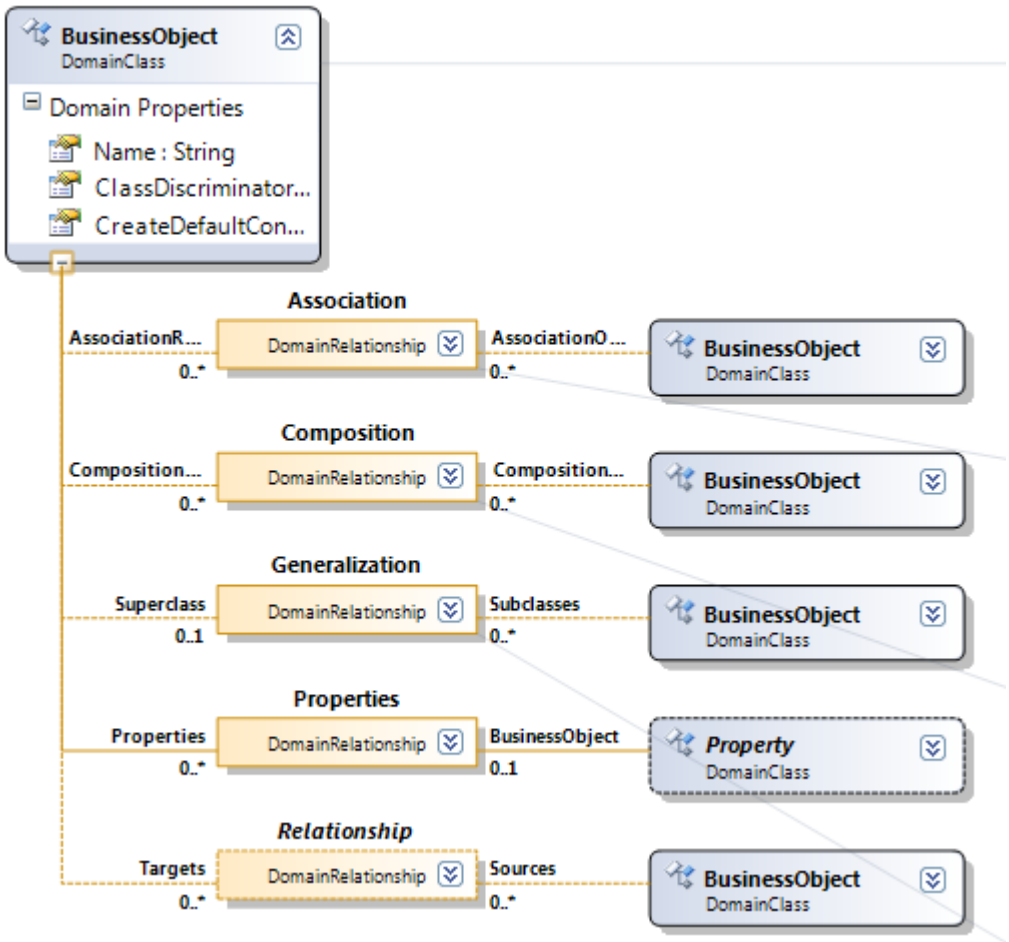


Abbildung 4 - BusinessObject Struktur

Die BusinessObject Domänenklasse beschreibt eine Domänenklasse, welche auf eine AscentivElements BusinessObject abgebildet wird.
Properties:

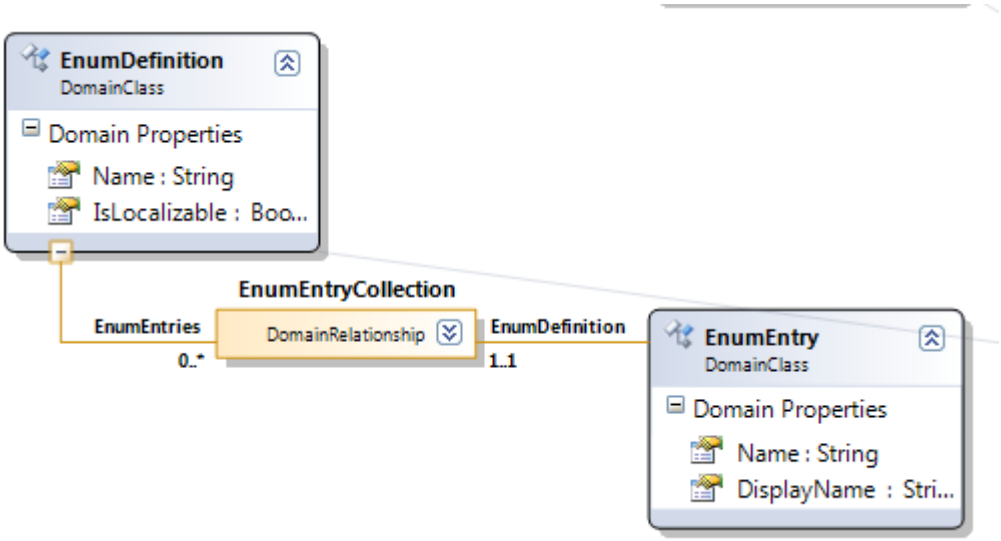
- Name: Der einzigartige Name der Klasse

- ClassDiscriminator: Ein einzigartiger Diskriminator, welcher benutzt wird im Falle einer Table Per Hierarchy Vererbung, um die Klasse zu identifizieren.
- CreateDefaultConstructor: Sollen die Standard Konstruktoren des BO's erstellt werden oder muss dies der Entwickler über Custom Code tun.

Relationen:

- Association: Alle Relationen zwischen BO's, die nur referenzieren aber nicht integrieren (kein Containment).
- Composition: Alle Relationen zwischen BO's, die eine Komposition (Containment) darstellen.
- Generalization: Vererbungen zwischen verschiedenen BO's
- Properties: Eigenschaften eines BO's.
- Relationship: Alle Relationen zwischen BO's. Fasst Composition und Association zusammen.

2.2.3 EnumDefinition



<
Abbildung 5 - EnumDefinition Struktur

Die EnumDefinition stellt eine Enumeration aus Elementen dar. Diese Enumerationen können als Property Type ausgewählt werden.

Properties:

- Name: Der einzigartige Name im Model.
- IsLocalizable: Soll die Enumeration als lokalisierbar erstellt werden. Dies ist ein AscentivElements Feature.

Relationen:

- EnumEntryCollection: Alle Einträge einer Enumeration.

2.2.4 EnumEntry

Ein Enumerationseintrag einer EnumDefinition.

Properties:

- Name: Der Name eines Enumeintrages, der innerhalb der Enumdefinition einzigartig sein muss.
- DisplayName: Wenn die EnumDefinition nicht lokalisierbar ist, gibt diese Eigenschaft den Anzeige Name wieder. Dies ist ein Ascentiv Elements Feature.

2.2.5 Property

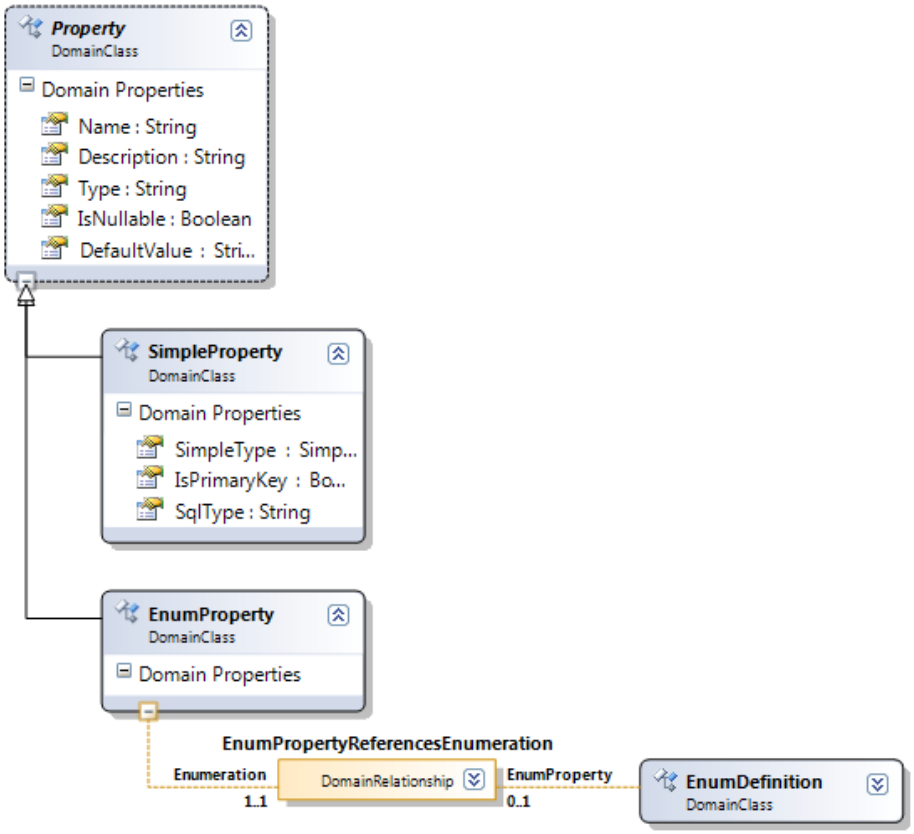


Abbildung 6 - Property Struktur

Property ist eine Überklasse für alle Eigenschaften eines BO's.
Properties:

- Name: Der Name des Properties, der innerhalb eines BO's einzigartig sein muss.
- Description: Die Beschreibung für ein Property. Diese Eigenschaft ist nicht implementiert.
- Type: Der Property Type als String
- IsNullable: Kann die Eingeschaft nicht gesetzt sein oder muss sie immer existieren.
- DefaultValue: Ein Standard Wert als String. Dieser Wert wird nur für die Datenbank benutzt.

2.2.6 SimpleProperty

Ein SimpleProperty ist eine Skalare Eigenschaft für ein BO.
Properties:

- SimpleType: Eine vorgegebene Enumeration von skalaren Datentypen. Dies beinhaltet zum Beispiel Zahlenwerte, Zeichenketten oder Wahr/Falsch Werte.
- IsPrimaryKey: Beschreibt dieses Property die Einzigartigkeit des BO's. Wenn dies gesetzt ist, muss der SimpleType entweder eine GUID oder ein Int32 sein.
- SqlType: Der Datentyp, der auf der Datenbank benutzt werden soll. Wird automatisch anhand des SimpleType gesetzt, kann jedoch selbst gändert werden. Dies erlaubt die Parametrierung von Sql Typen wie zum Beispiel „nvarchar(50)". Die Eingaben werden streng überprüft und im Fehlerfall zurückgesetzt. Erlaubte parametrierbare Typen sind anhand dieser Regular Expressions:
  ^(char|varchar|text|nvarchar|ntext|nchar)\(([0-9]+|max)\)$
  ^float\([1-5]?[0-9]\)$
  ^(decimal|numeric)\([1-5]?[0-9]\)$
  ^(datetime2|datetime)$

2.2.7 EnumProperty

Ein EnumProperty ist eine Eigenschaft, welche eine EnumDefinition als Typ haben kann. Der DefaultValue der Basisklasse kann einem EnumEntry entsprechen.

Relationen:

- EnumDefinition: Eine im Model vorhandene EnumDefinition.

2.2.8 Relationship

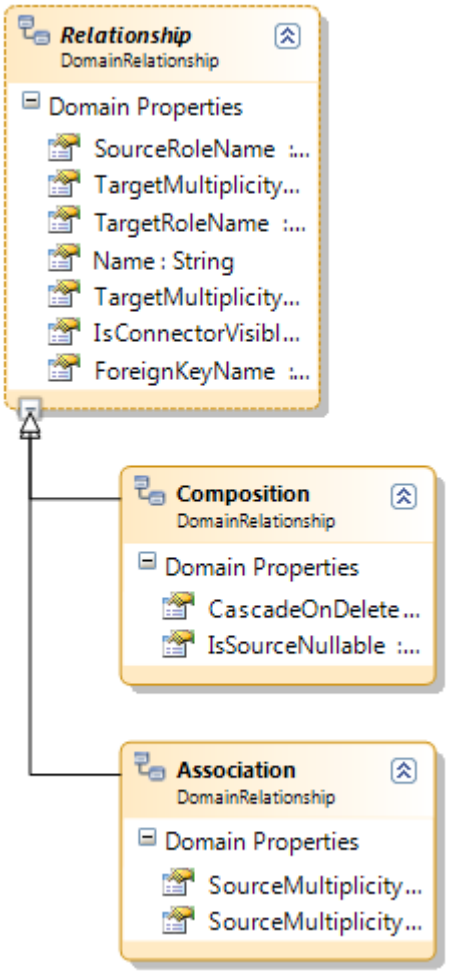


Abbildung 7 - Relationship Struktur

Eine Relationhip ist eine Überklasse für alle Relationen zwischen BO's.
Properties:

- SourceRoleName: Der Eigenschaftsname auf der Source Seite. Dies wird der Name des NavigationProperty auf der Target Seite.
- TargetMultiplicity: Die Kardinalität auf der Target Seite.
- TargetRoleName: Der Eigenschaftsname auf der Target Seite. Dies wird der Name des NavigationProperty auf der Source Seite.
- Name: Der im Model einzigartige Name der Relation.
- TargetMultiplicityDisplay: Die graphische Repräsentation der TargetMultiplicity. Dies erlaubt Eingaben als Strings der Form 0..1, 1, 0..*, 1..*
- IsConnectorVisible: Soll die Relation als graphische Verbindung dargestellt werden oder soll die Verbindung über ein NavigationProperty sichtbar sein.
- ForeignKeyName: Der FK Name auf der Datenbank. Pro BO muss der FK Name einzigartig sein.

2.2.9 Composition

Composition repräsentiert eine Teil/Ganzes Beziehung und enthält Ascentiv Elements spezifische Semantik. Die SourceMultiplicity ist dabei immer 0 oder 0..1.
Properties:

- CascadeOnDelete: Soll die Relation auf der Datenbank ein CascadeOnDelete enthalten

- IsSourceNullable: Soll die Composition Beziehung aufgelockert werden und ein Null als FK Wert erlauben.

2.2.10 Association

Association repräsentiert eine Referenzbeziehung zwischen BO's.
Properties:

- SourceMultiplicity: Die Kardinalität auf der Source Seite.
- SourceMultiplicityDisplay: Die graphische Repräsentation der SourceMultiplicity als String. Erlaubte Eingaben sind 0..1, 1, 0..*, 1..*

2.3 Generierungsprozess

Der Generierungsprozess startet in der DslPackage Komponente. Dort gibt es eine registrierte Klasse CsharpCodeGenerator, welche bei jedem Speichern aufgerufen wird. Diese Klasse wird benötigt um die T4 Generierung aufzurufen. Es wird dort das Template aus den Resourcen geladen, ein paar Substitutionen ausgeführt und dann der Code generiert.
Das T4 Template ist aus zwei Gründen sehr schlank. Einerseits ist die Entwicklung im T4 sehr fehleranfällig und untypisiert. Es geht lange um Code zu schreiben und noch länger um ihn zu warten. Andererseits wäre viel zu viel Code im Template und viel zu wenige statische Statements. T4 ist vorallem dann geeignet, wenn nur weniger Teile ersetzt und nicht ganze Strukturen aus dem Boden gestampft werden müssen.

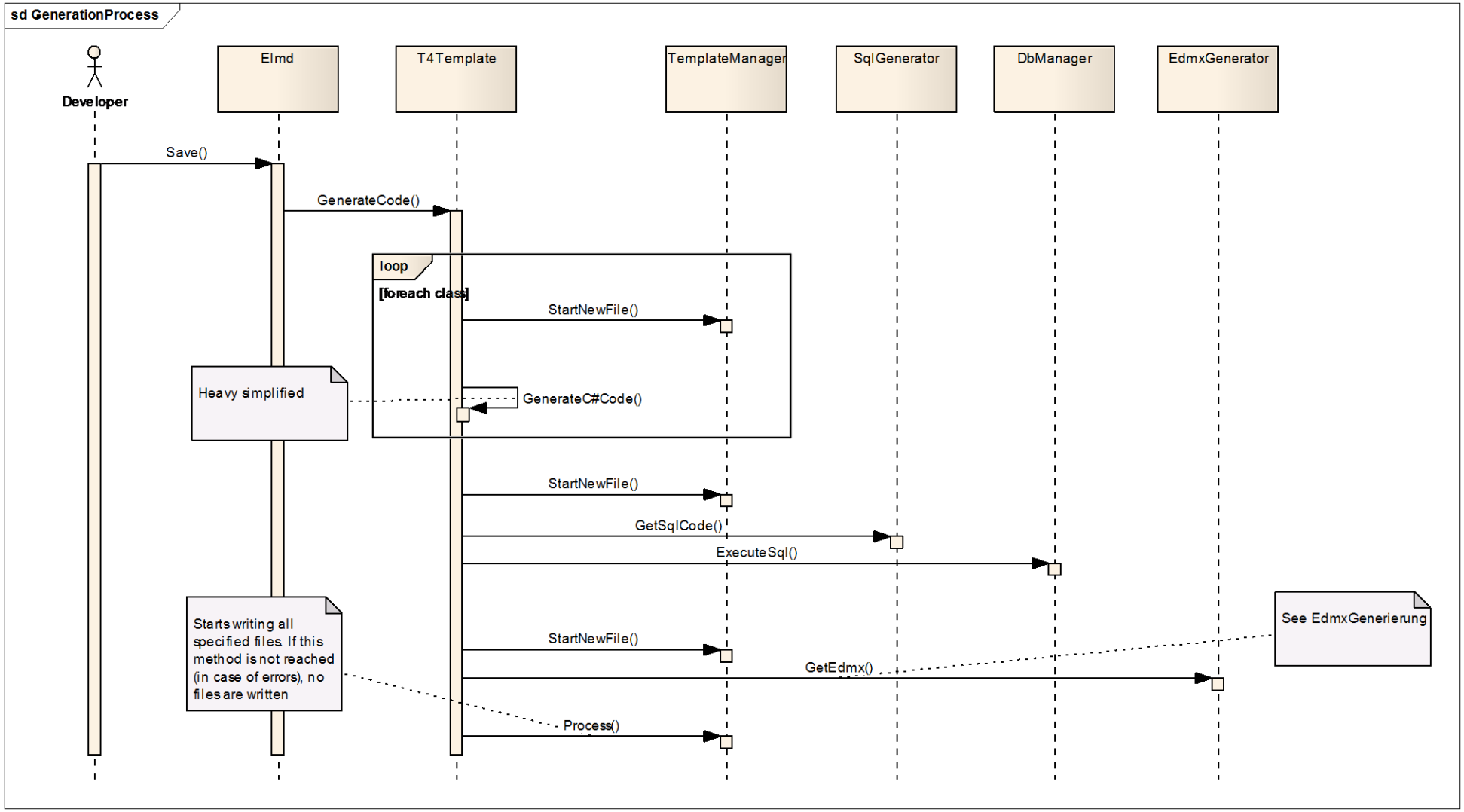


Abbildung 8 - Übersicht Generierungs Prozess

Die Code Generierung wird gestartet, wenn der Benutzer das Elmd speichert. Dies wird vom Visual Studio gesteuert. Das T4 Template wird gestartet und der TemplateManager initialisiert. Für jede Klasse wird eine Datei erstellt und der C# Code eingefügt. Danach wird der Sql Code erzeugt und über den DbManager zur Ausführung gebracht. Und anhand der erstellten Datenbank wird darauf das Edmx generiert und angepasst.

2.3.1 Template Manager

Um aus einem T4 Code Generator mehrere Outputs zu generieren, muss dies ausprogrammiert werden. Im Netz wurde ich fündig nach einer Lösung, die sich Template Manager nennt. Diese Lösung ist frei verfügbar zu finden unter http://damieng.com/blog/2009/11/06/multiple-outputs-from-t4-made-easy-revisited
Dieser Code stellt jedoch nur die Basis dar, da noch weit mehr Funktionalität benötigt wird. Der Code wurde erweitert um einen Pfad, an den der Output hingelangt. Dieser Projekt Pfad wird programmatisch aufgelöst und Items dem Projekt hinzugefügt.
Der Projekt Pfad wird angegeben durch <ProjektName>:\<Pfad>\<FileName>.<Endung>.
Eine weitere zusätzliche Funktionalität ist das automatische Entfernen von nicht mehr generierten Dateien. Source Control wird unterstützt indem Checkouts ausgeführt werden falls benötigt.
Um im T4 eine neue Datei zu Starten muss StartNewFile aufgerufen werden und wenn dieses geschlossen werden soll muss EndBlock aufgerufen werden. Wenn alle Blöcke geschrieben wurden, muss auf dem TemplateManager Process aufgerufen werden und erst dann werden alle Dateien geschrieben.

2.3.2 Code Generator

Der Code Generator „ElementsClassGenerator" beinhaltet alle Methoden um eine BusinessObject Klasse zu generieren. Der Namespace und die Klassendeklaration hingegen befinden sich auf dem Template.
ElementsClassGenerator ist mit partial classes in mehrere Dateien aufgeteilt. Aufgrund der Einfachheit der Klassen ist eine strengere Strukturierung nicht von Nöten.

2.3.3 Sql Generator

Der SqlGenerator hat eine Methode GetSqlCode, die als einzige public ist. Dies erzeugt Sql Code der:

1. die alte Referenzdatenbank löscht.
2. die neue Datenbank erzeugt.
3. jede Tabelle mit allen Attributen erstellt.
4. jedes Constraint einfügt.

Der Sql Code wird, falls gegeben, an den DAL Path gespeichert.

2.3.4 DbManager

Der DbManager führt Sql Code aus. Dabei teilt er das Sql Script nach „GO" Statements auf und führt Teil für Teil aus. Wenn ein Statements fehl schlägt wird eine AbortException geworfen und die ganze Generierung wird abgebrochen. Erst wenn alle Statements erfolgreich sind, wird die Generierung weiter geführt.
Der DbManager führt nicht nur den Code aus, der ihm als Parameter mitgegeben wird, sondern auch Sql Extensions. Die Sql Extensions sind dazu da um weiteren Sql Code ins Model einfliessen zulassen. Dazu wird der DAL Path verwendet. In diesem Pfad muss ein Unterordner „Extensions" existieren und in diesem werden alle „*.sql" Dateien nach dem eigentlichen Sql Code zusätzlich ausgeführt.
Mit dieser Methode können zum Beispiel Triggers oder StoredProcedures eingefügt werden.
Der DbManager enthält neben der ExecuteSql Methode eine Ping Methode, welche einen ConnectionString auf seine Gültigkeit überprüft. Dieser wird von der Model Validierung benutzt um den Start der Generierung abzubrechen, wenn die Referenzdatenbank nicht erreicht werden kann.

2.3.5 Edmx Generator

Das Edmx wird anhand der Datenbank generiert („reverse engineered") Dazu wird die Komponente EdmGen2 verwendet. Es ist eine quelloffene verbesserte Implementierung des EdmGen von Microsoft, welches keine direkte Edmx Generierung erlaubt, sondern nur einzelne Teile, welche dann zusammengefügt werden müssten. Um nicht in Lizenzprobleme zu enden ist dieser Code in einem eigenen Assembly.

Das Reverse Engineering genügt jedoch nicht, da viele Informationen aus dem DB Schema nicht ausgelesen werden können. Das Storage Model ist zwar nach der Generierung vollständig, das Conceptual Model sowie das Mapping jedoch müssen angepasst werden.

Anzupassen sind folgende Punkte:

- Die Namen der Entitäten im Conceptual Model müssen das Suffix „Entity" bekommen. Das korrespondierende EntitySet muss das Suffix „EntitySet" erhalten. (ClassNameEdmxProcessor)
- Die NavigationProperties im Conceptual Model müssen umbenannt werden, damit sie den Rollen Namen im Dsl entsprechen. (NavigationPropertyEdmxProcessor)
- Assoziationen im Conceptual Model müssen, wenn eine 1:1 Kardinalität besteht, dies explizit angeben, da anhand der Datenstruktur eine 1:n Kardinalität angenommen wird. (AssociationEdmxProcessor)
- Vererbungen müssen komplett erstellt werden. Das bedeutet, dass im TPH Entitäten und alle Mappings erstellt werden müssen. Im Falle von TPT müssen alle Mappings geändert und die Vererbungsstruktur eingeflochten werden. (InheritanceEdmxProcessor)

Um diese Aufgaben klar zu trennen werden Processors benutzt.

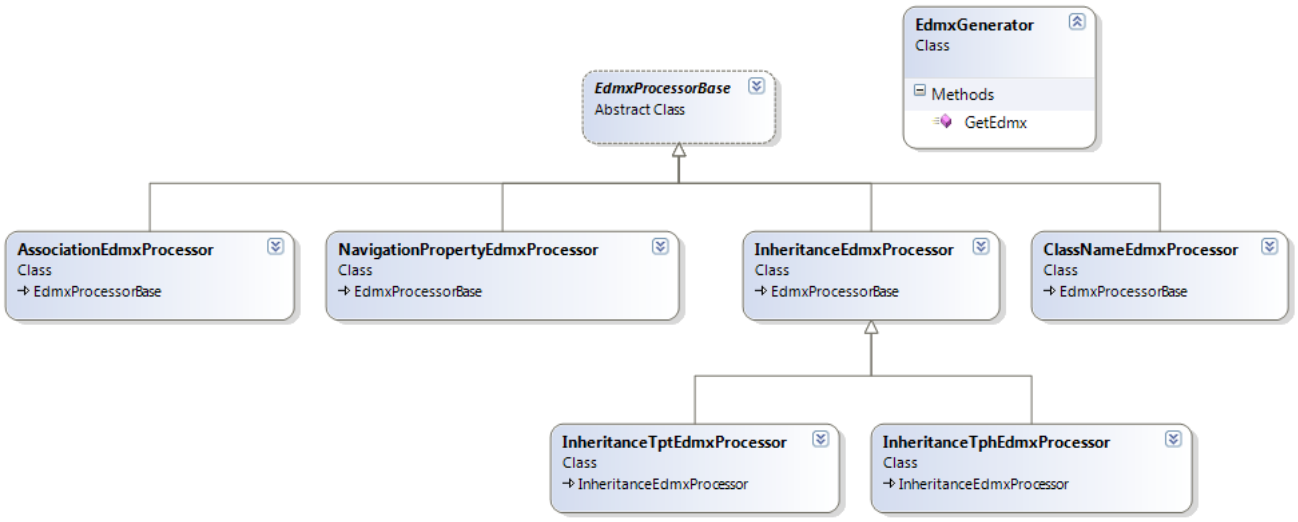


Abbildung 9 - Edmx Processor Übersicht

Der Ablauf dieser Processors ist in den folgenden Diagrammen abgebildet:

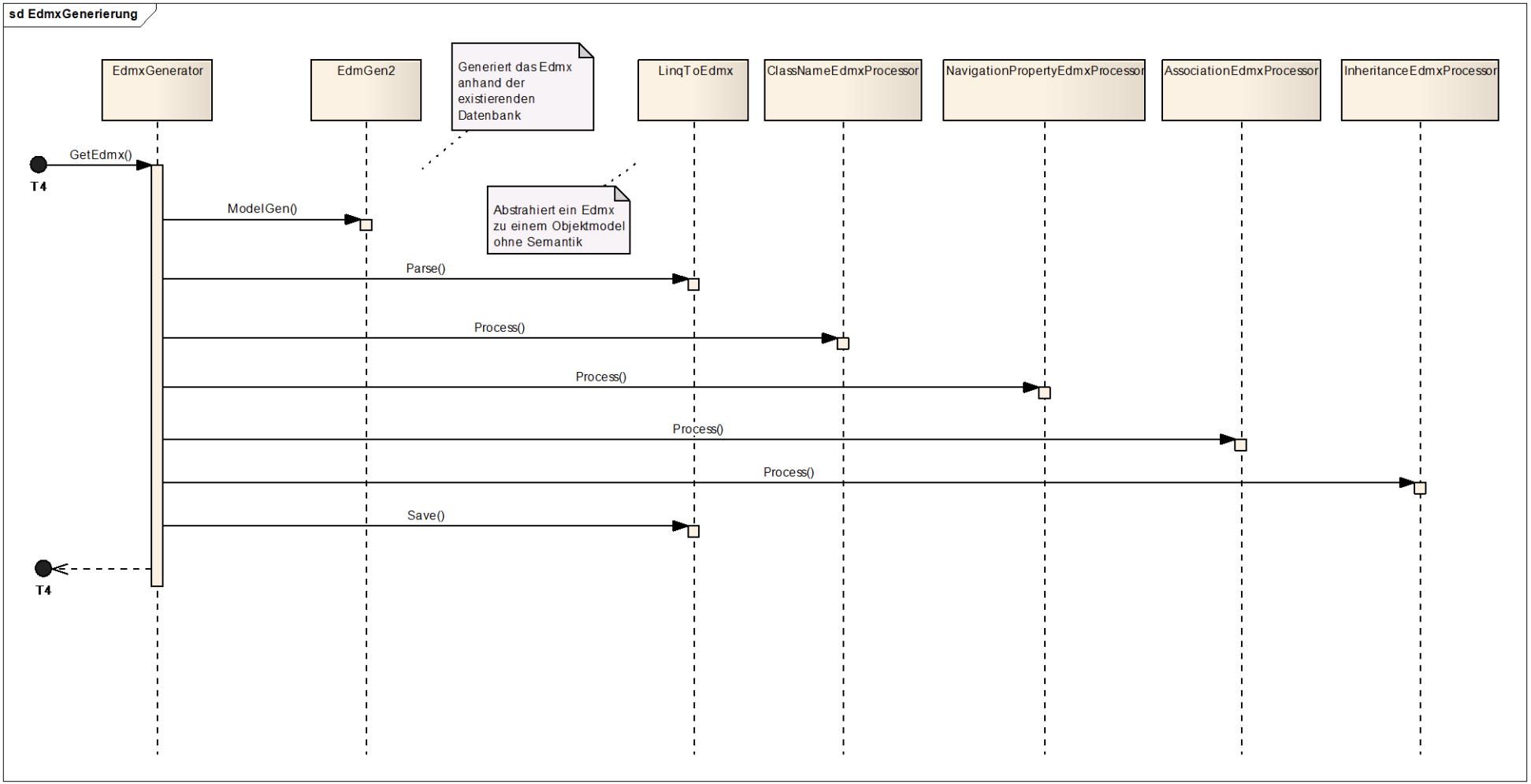


Abbildung 10 - Edmx Generierung

Der Edmx Generator erhält als Parameter die Verbindungsinformationen und das Modell. Zuerst wird über ReverseEngineering über die EdmGen2 Komponente ein Edmx erzeugt. Anhand dieser Datei lädt LinqToEdmx seinen Kontext. Für jede Aufgabe existiert ein Processor. Wenn die Aufgabe nochmals unterteilt werden kann werden dort weitere Processor benutzt.

Der Vererbungs Processor ist in sich noch weiter in Processor aufgeteilt:

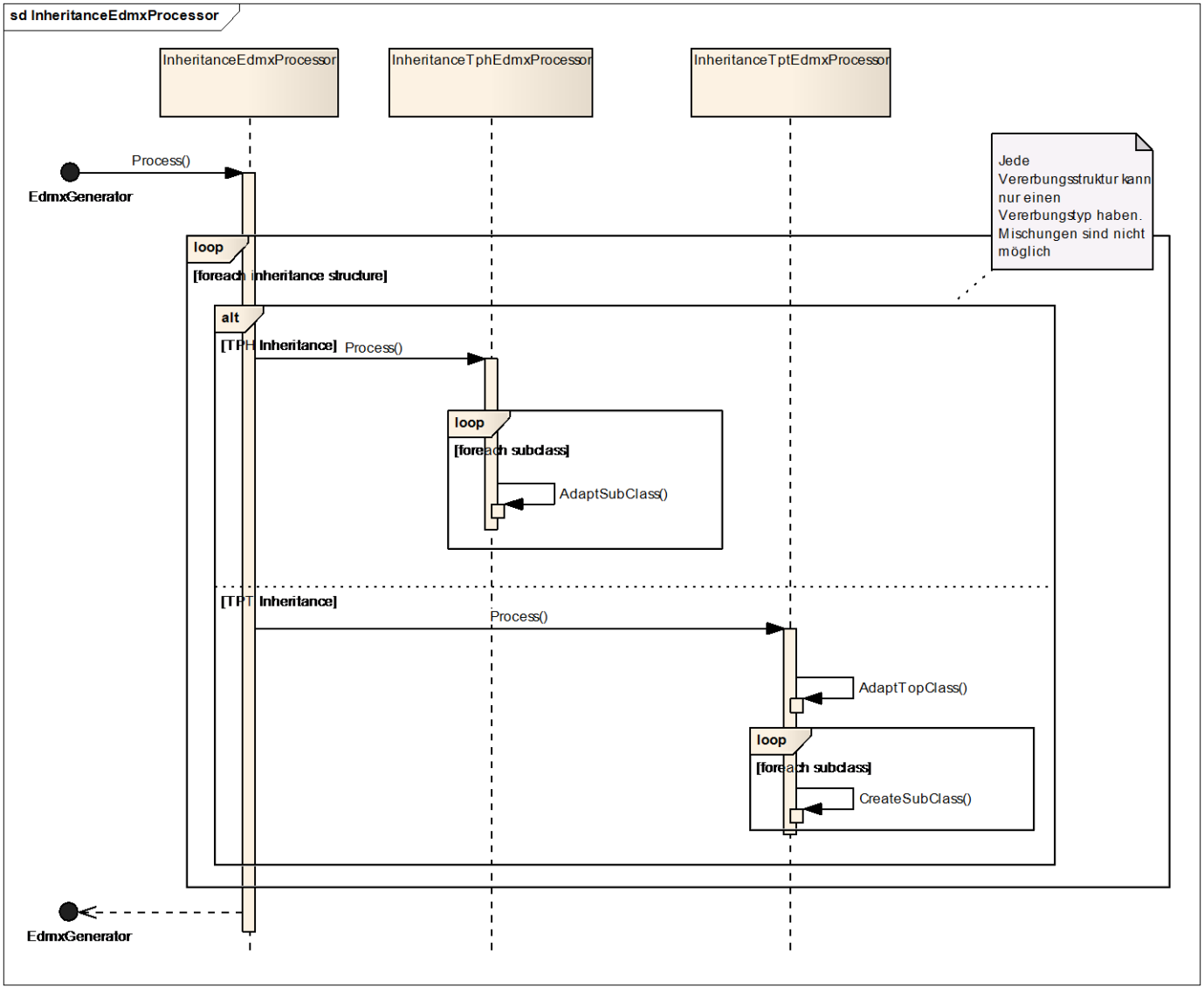


Abbildung 11 - Vererbungs Processor

Jede Verbung gehört zu einer Vererbungsstruktur. Und pro Vererbungsstruktur gibt es nur einen Vererbungstyp. Im Fall der TPH Vererbung müssen nur die Subklassen angepasst werden. Bei TPT Vererbung hingegen muss die höchste Klasse angepasst werden und alle Sub Klassen neu erstellt werden.

3 Test Projekt – ServiceCms

3.1 Zweck

Unit Tests für Elements MD sind sehr umständlich, da es sich um eine Visual Studio Erweiterung handelt. Der Aufwand wäre gross und der Nutzen beschränkt, da der generierte Output kaum verifizierbar ist. Ein besserer Ansatz ist ein System Test mit Ascentiv Elements, denn wenn dieses beim speichern und laden von Daten keine Fehler meldet funktioniert das System.
Das Projekt ServiceCms soll eine Referenz Benutzung von Ascentiv Elements mit Elements Model Driven darstellen. Es gilt zugleich als Testprojekt für einen Systemtest. Dieses Projekt deckt ein breites Spektrum der Funktionalität ab und testet zugleich die Integration mit Ascentiv Elements.
Das Projekt bildet ein Customer Service Relationship Tool ab. Es besteht aus einem in IIS gehosteten Server und einem Silverlight Client.

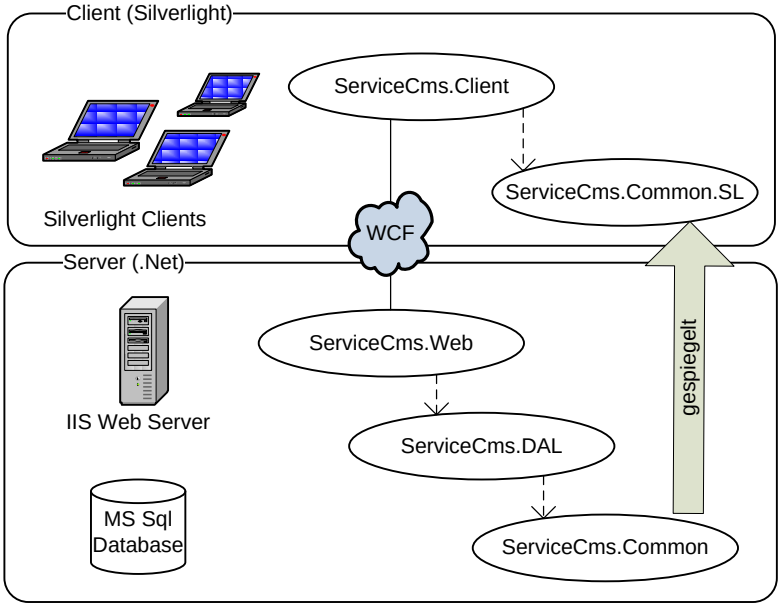


Abbildung 12 - ServiceCms Overview

3.2 Projektstruktur

Das Projekt besteht aus folgenden Komponenten:

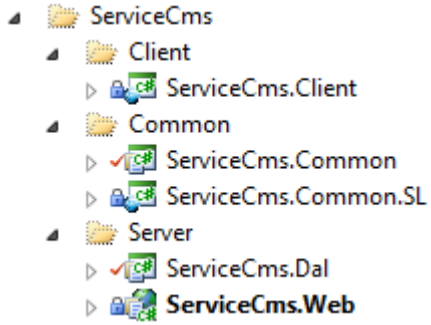


Abbildung 13 – Projektstruktur ServiceCms

ServiceCms.Web: Das IIS Server Projekt. Es beinhaltet den Ascentiv Elements CRUD Service und das Hosting für den Silverlight Client.
ServiceCms.Dal: Der Data Access Layer auf dem Server. Er übernimmt alle Datenbank Zugänge, beinhaltet das Edmx, EF Accessoren, alle Sql Scripts und das Sql Compare File.

ServiceCms.Common: Die Business Objekte und die gemeinsame Business Logik. In dieses Projekt gehört das Elmd File. Dies ist die .Net Version der BusinessObjects.
ServiceCms. Common.SL: Dieses Projekt ist gespiegelt von seinem .Net Pendant und beinhaltet die gleichen Business Objekte. Das Spiegeln kann von Hand geschehen, in diesem Fall aber mit dem ProjektLinker der CompositeApplicationLibrary (http://msdn.microsoft.com/en-us/library/ff921108(PandP.20).aspx) bewerkstelligt. Das Linking zu Silverlight ist nötig, da der Code zwar für beide Technologien kompiliert, der Code aber nicht Clr kompatibel ist.
ServiceCms.Client: Der Silverlight Client. Er beinhaltet das UI, lädt, speichert und editiert BusinessObjects.

3.3 Die Domäne

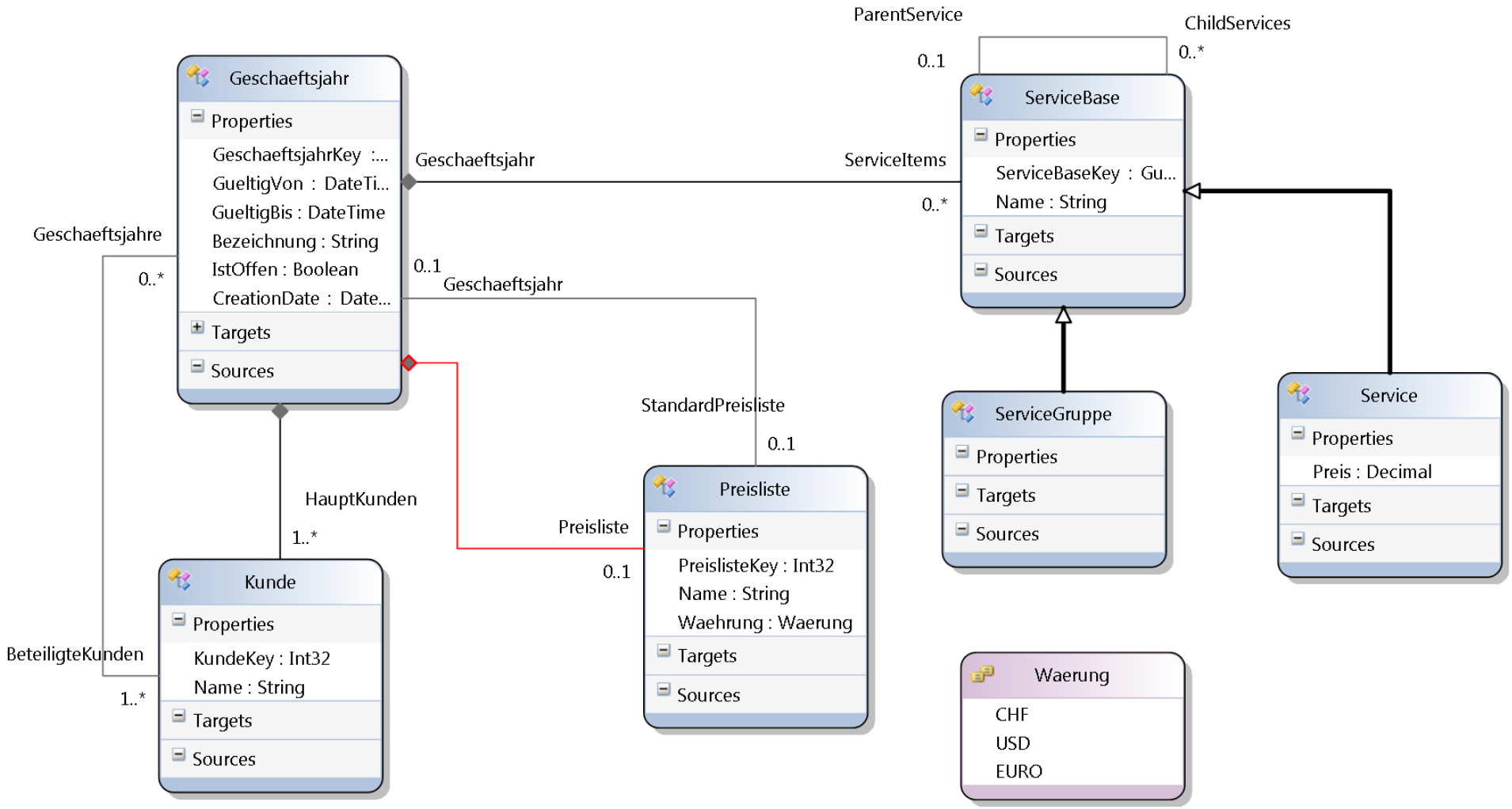


Abbildung 14 - Elmd von ServiceCms

3.3.1 Abgedeckte Fälle

In diesem Model haben wir folgende Fälle abgedeckt:

N zu N: Zwischen Geschaeftsjahr und Kunde.
Composition mit Cascade: Zwischen Geschaeftsjahr und Preisliste
Composition mit 1:0..1: Zwischen Geschaeftsjahr und Preisliste
Enumeration: Waerung
Enumeration Property: Preisliste.Waerung
Vererbung: ServiceBase, ServiceGruppe, Service
Rekursion: ServiceBase
Composition mit 1:N: Zwischen Geschaeftsjahr und Kunde
Doppelte Relationships zwischen Typen: Zwischen Geschaeftsjahr und Preisliste
Int als Key: PreistlisteKey Property
N zu N mit Guid und Int als Keys: Zwischen Geschaeftsjahr und Kunde.
Partial Class Erweiterung: Geschaeftsjahr ist erweitert worden um transiente Properties.

3.3.2 Nicht abgedeckte Fälle

Enum in Vererbung: Ein Enumerationsproperty in einer Vererbung
Verschiedene Vererbungen: Mehrere Vererbungsarten in einem Diagram
Relationship zu Vererbung: Erbende Klassen mit Relationen auf andere Klassen.

3.4 Strategie

Das Projekt muss nicht nur kompilieren, sondern muss nach jedem Generierungsprozess auch lauffähig sein. Dazu startet man den Silverlight Client, fügt alle möglichen Objekte (Geschäftsjahre, Kunden, Services etc.) hinzu und speichert die Daten. Wenn das Speichern erfolgreich war kommt keine Fehlermeldung. Als nächstes wird verifiziert, dass die Daten auch korrekt geladen werden können. Dazu startet man die Applikation neu (über Refresh im Browser). Sind die Daten wieder sichtbar, ist die volle Funktionsfähigkeit sichergestellt.

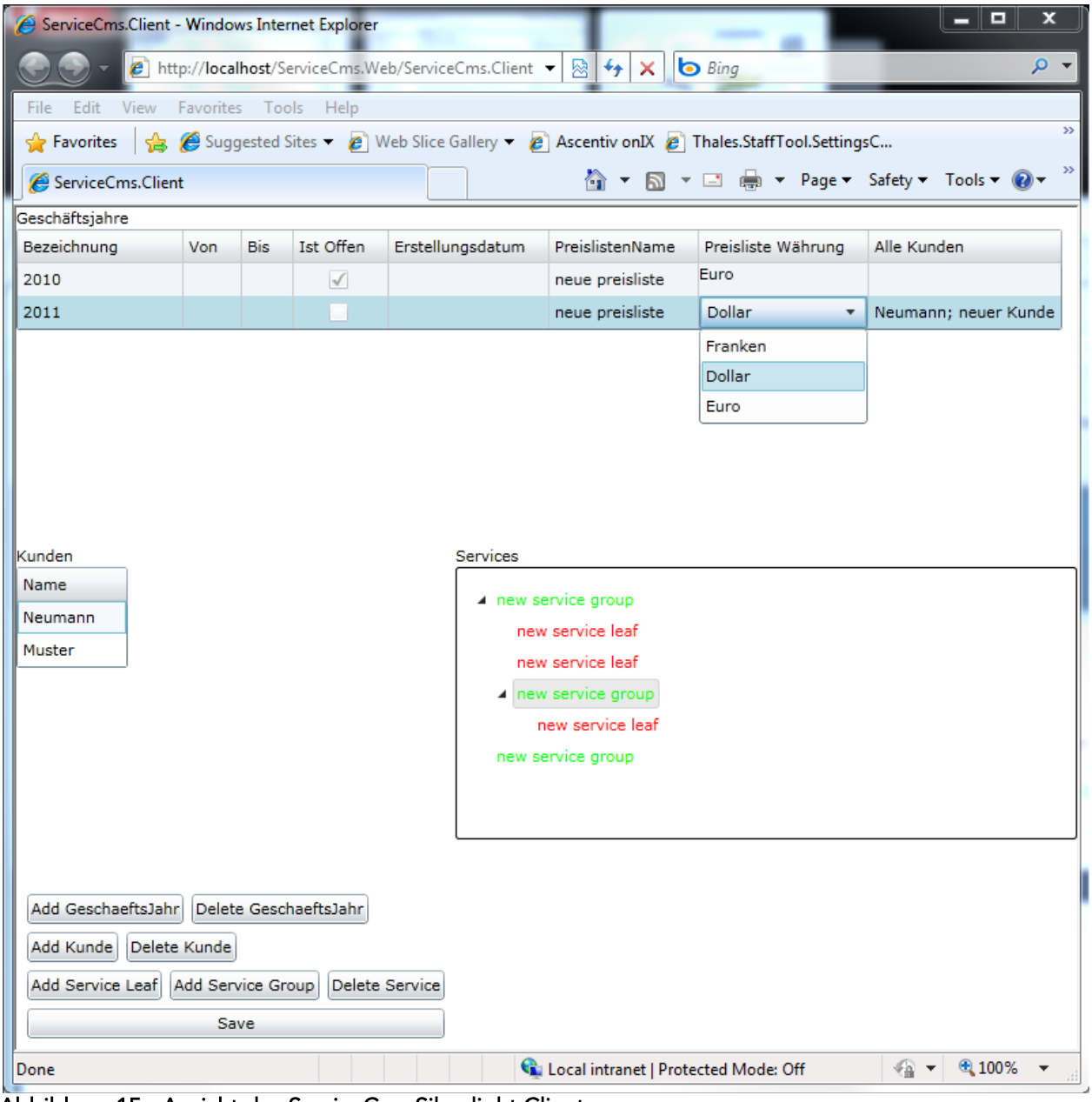


Abbildung 15 - Ansicht des ServiceCms Silverlight Clients

4 Schlussfolgerungen

4.1 Erreichte Ziele

Der Entwickler eines Projekts mit Ascentiv Elements kann Elements MD verwenden, ohne dabei an technische Grenzen zu stossen. Mit den Sql Extensions und Partial Classes kann soweit alles erweitert werden, was noch nicht unterstützt wird. Natürlich ist das fernere Ziel von Elements MD die volle Unterstützung von Ascentiv Elements. Wichtig ist jedoch, dass man sich bei einem Projekt nicht entscheiden muss zwischen Elements MD oder Ascentiv Elements Funktionalität. Denn nur wenn dieses Tool auch konsequent genutzt wird, wird es sich weiterentwickeln und nur wenn es sich weiterenwickelt, wird es besser und effizienter.
Der service-orientierte Ansatz wird in diesem generischen Model Driven Ansatz jedoch nur für ein Daten CRUD verwendet. Höherwertige Abstraktionen müssen weiterhin explizit implementiert werden. Dies ist jedoch auch wünschenswert, da dies die Minderheit der Fälle darstellt und hauptsächlich zur Performance Optimierung benutzt wird.
Leider geht bei der Benutzung von Ascentiv Elements die Heterogenität der SOA ein wenig verloren, da ohne Ascentiv Elements auf der Client Seite die Schnittstelle kaum zu gebrauchen ist. Und auch Elements MD kann hier nur Abhilfe schaffen, wenn die Technologie ins Visual Studio integrierbar ist. Microsoft geht mit den RIA Services dem rein service-orientierten Weg und lässt sich die Schnittstelle anhand von Modellen generieren. Und in Kombination mit Microsoft LightSwitch, welches sich noch in der Beta Phase befindet, kommt ein weiteres Kernstück des Model Driven Ansatz hinzu. LightSwitch geht sogar noch einen Schritt weiter und generiert auch ein Datengetriebenes UI dazu. Man sieht, dass die Bewegung auch von Microsofts Seite sich in dieselbe Richtung wie Ascentiv Elements mit Elements Model Driven. Dieser Ansatz ist definitiv zukunftsträchtig.

4.1.1 Modellierung

Es sind alle gängigen Modellierungsarten unterstützt. Es können Klassen, Enumerationen, Vererbungen, Assoziationen und Kompositionen abgebildet werden, ohne dabei Einschränkungen machen zu müssen. N zu N Relationen, Rekursive Relationen mehrfach auf die gleiche Klasse, beliebig gerichtete Relationen; Die Flexibilität genügt jeder Domäne.
Graphisch enthält der Editor viele Hilfen zur Übersicht, zum Beispiel zum Lifecycle von Elementen, gerichtete Relationen oder optionale Darstellung von Relationen. All diese simplen Funktionen helfen dem Entwickler bei der Analyse und Entwicklung einer grossen komplexen Domäne.

4.1.2 Generierung

Die Generierung eines Models nimmt dem Entwickler nicht nur einfache Arbeit ab, sondern auch komplexere Arbeiten, zum Beispiel am Edmx, die nur unter mühseliger und komplexer Handarbeit erledigt werden müssten. Das Implementieren von Vererbungen war bisher im Edmx immer umständlich und mit einigem Know How verbunden, diese Arbeit wird dem Entwickler nun abgenommen. Auch muss der Entwickler nie mehr damit kämpfen, die Namen von Typen und Attributen kongruent zu halten. Er kann damit rechnen, dass wenn das Model geändert wurde und der Code kompiliert, keine Laufzeitfehler von Ascentiv Elements auftreten.
Die Generierung ist so konzipiert, dass sie mühelos erweitert werden kann. Verallgemeinerungen wurden so gut als möglich umgesetzt und neuer Funktionalität steht nichts im Weg.

4.2 Fehlende Funktionalität

4.2.1 Fehlende Ascentiv Elements Unterstützung

Feature	Priorität (1-3)	Beschreibung	Konsequenzen	Aufwandschätzung
BO Cache	1	Eine Attributierung auf Klassen und Properties um diese als Cached zu markieren.	Das Dsl Model muss erweitert werden um ein Property, das als Editor einen Abhängigkeitsbaum darstellt.	Quickfix 8h, Komplett 20h
Validierung	2	Eine Typen Validierung auf BO Ebene, um z.B String Überlänge zu detektieren	Komplexere Interpretation des Sql Types sowie erweiterte Code Generierung.	String Länge 6h, Numerische Fälle 12h
Visualisierung	2	BO's können in unabhängigen Views betrachtet und eingeschränkt verändert werden.	Einarbeitung in den ModelBus, Einführen einer neuen Dateiendung, altes Model ggf erweitern, neues Model erzeugen.	30-40h
Visual Keys	1	Die Code Generierung um Visual Keys erweitern, welche die Sortierung der geladenen Daten anhand eines Properties erlaubt und die Darstellung eines Objekts definiert.	Das Dsl Model muss erweitert um mehrere Properties erweitert und die Code Generierung ein wenig angepasst werden.	Quickfix 8h, Komplett 20h
Navigation Property Header verstecken	3	Der Header der Navigation Property Compartments soll nur gezeigt werden, wenn auch Navigation Properties dargestellt werden.	Anpassung an der Shape Klassen, evt. weitere Coherence Rules.	8h
Transient Propeties	3	Transient Propeties sollen im Designer erstellt werden können.	Das Dsl Model muss erweitert werten um einen weiteren Property Typ, der jedoch nur in der Code Generierung berücksichtig wird.	4h
Transient Business Objects	3	Transiente Business Objects, welche nie persistiert werden, können im Designer modelliert werden, woraus dann ebenfalls der Code generiert wird.	Das Dsl Model muss erweitert werden um einen weiteren Klassen Typ, der nur im Code Generator berücksichtigt wird.	6h
Modularisierung	2	BO's können von anderen elmd Diagrammen referenziert werden und somit ebenfalls generiert werden.	Einarbeitung in den ModelBus, zusammenführen von Modellen on demand, leichte Anpassungen an die gesammte Model Generierung, UI Konzepte zur Referenzierung erarbeiten	30-40h
FileStore	1	Der FileStore ist die BO Unterstützung für persistente Dateien	Einen neuen Property Typ in das Dsl Model einbringen, die Code Generierung leicht anpassen, spezielle Sql Code Generierung einfügen und weitere serverseitige Aufgaben	20h

			erledigen für den FileStore	
Localization	1	Lokalisierbaren Text auf BO Ebene Unterstützen.	Speziellen Property Typ im Dsl Model einfügen, die Code Generierung anpassen, die Sql Generierung erweitern.	8h
Metadaten und Mapping erzeugen	3	Metadaten und das Mapping von DTO zu EF wird Momentan über Reflection ausgeführt. Dies könnte nun ebenfalls ein Teil der Generierung werden und somit statisch erledigt.	Ascentiv Elements muss angepasst werden, weiterer Code müsste für die serverseite erzeugt werden, jedoch keine Änderungen am Dsl Model	20h (ohne Elements Anpassung)
Property Dependency	3	Die Ascentiv Elements PropertyDependency Attributierung soll im Designer modelliert werden können.	Neben Erweiterungen im Dsl Model muss ein Property Editor eingefügt werden, der andere Properties selektieren kann.	16h
Property Reihenfolge	2	Die Reihenfolge der Properties einer Klassen soll beliebig änderbar sein.	Evt. leichte Dsl Model Erweiterungen, weiterer Dsl Code und Visual Studio Integration für Commands	8h
Generic Operations	2	Operationen, welche über eine generische WCF Schnittstelle aufgerufen werden können, sollen modelliert und generiert werden können.	Es müsste eine Swimlane eingeführt werden, um die Operationen graphisch vom Model zu trennen. Das Model selbst müsste erweitert werden, sowie geringfügig die Code Generierung. Die Operationen müssten aber zusätzlich auf dem Server mit der entsprechenden Methode verlinkt werden.	14h
Documentation	3	Tripple Slash Kommentare einfügen	Leichte Modeländerung und leichte Änderung an Code Generator	2h

4.2.2 Fehlende Funktionalität

Feature	Priorität (1-3)	Beschreibung	Konsequenzen	Aufwandschätzung
Auto Layout	2	Das Diagram soll nach Wunsch automatisch gelayoutet werden.	Visual Studio Integration für Commands	4h
Explizite Validierung	2	Nach Wunsch und ohne speichern validieren können.	Visual Studio Integration für Commands	4h
Optionales Auto Generate	2	In diesem Stand werden bei jedem Speichern und teilweise beim Window wechsel die Generierung ausgelöst. Im sehr frühen, oder auch im späten Verlauf des Projekts möchte man vielleicht das Model speichern,	Leichte Dsl Model Erweiterungen und Visual Studio Integration für Commands	8h

		jedoch die Generierung verzögern. So könnte über eine Einstellung das automatische Generieren abgeschaltet werden.		
Solution Template	3	Ein Solution Template für eine Silverlight oder Wpf Client/Server Lösung, bei der die ganze Elements Struktur eingerichtet ist, die richtigen Assemblies gespiegelt werden und ein Elmd existiert, um bereits mit dem Prototyp beginnen zu können.	Ein Solution Template erstellen und mit Deployen.	8h
EF Connection String	2	Ein bereits angepasster EntityFramework ConnectionString, der nur noch kopiert werden muss.	Dsl Code verändern	3h
Copy Paste	1	Copy Paste von Properties und Klassen funktioniert nur teilweise.	Dsl Model konfigurieren für Copy Paste	4h
Foreign Keys in EF	2	Foreign Keys sollen im EF optional als Property erstellt werden.	EdmGen2 erweitern um Parameter	3h