Vergleich von objektrelationalen Mappern

Florian Amstutz <florian@amstutz.nu>

Bachelorarbeit an der Zürcher Hochschule für Angewandte Wissenschaften

Eidesstattliche Erklärung

Hiermit bestätigt der Unterzeichnende, dass die Bachelorarbeit mit dem Thema "Vergleich von objektrelationalen Mappern" gemäss freigegebener Aufgabenstellung vom 8. Januar 2013 ohne jede fremde Hilfe im Rahmen des gültigen Reglements selbständig ausgeführt wurde.

Zürich, 6. Juni 2013

Florian Amstutz

Inhaltsverzeichnis

1 Einführung

Als einführendes Kapitel dieser Dokumentation wird die Bachelorarbeit als Projekt kurz vorgestellt und es werden die Rahmenbedingungen der Arbeit zusammengefasst niedergeschrieben. Weiter wird die Vorgehensweise für die Bearbeitung dieser Bachelorarbeit vorgestellt und erklärt. Am Ende dieses Kapitels wird eine Projektplanung erstellt und die wichtigsten Meilensteine festgelegt.

1.1 Management Summary

Objektorientierte Programmiersprachen kapseln Daten und Verhalten in Objekten, relationale Datenbank verwenden hingegen Tabellen. Um zwischen diesen grundverschiedenen Paradigmen übersetzen zu können werden objektrelationale Mapper (OR-Mapper) verwendet.

Heutige serviceorientierte Architekturen verwenden fast in jedem Fall einen Persistenzlayer welcher Objekte in eine Datenbank speichert und sie von dort liest. Diese Aufgabe wird meist nicht von der Applikation selber übernommen sondern durch einen OR-Mapper ausgeführt. Es gibt eine Reihe von Einflussfaktoren und Entscheidungskriterien die beim Entwurf und der Implementierung von Persistenzkomponenten beachtet werden müssen. Die Auswahl des passenden OR-Mappers für die jeweilige Architektur ist eine der zentralen Fragen, die zu Beginn des Implementierungszyklus einer Applikation beantwortet werden muss (nach [15]), denn oft ist der Persistenzlayer das Nadelöhr einer Enterprise Applikation (gemäss [2])

1.2 Über die Bachelorarbeit

Gemäss Reglement der ZHAW (siehe [16]) dient die Bachelorarbeit zum Nachweis der Fähigkeit ein Projekt auf ingenieurswissenschaftlich Art und Weise zu bearbeiten. Die Bachelorarbeit ist nach den allgemeingültigen Standards für technisch/wissenschaftliches Arbeiten verfasst.

Die Bachelorarbeit besteht aus einer konzeptionellen Arbeit und deren Umsetzung. Der Schwerpunkt liegt auf dem konzeptionellen Teil, in dem die theoretischen und methodischen Grundlagen einer Entwicklung oder eines Konzeptes ausgearbeitet und dargelegt werden. Im Umsetzungsteil erfolgt anschliessend die Beschreibung der Implementierung bzw. der Anwendung.

1.3 Ziel der Arbeit

Das Ziel dieser Arbeit besteht darin die Entscheidung zu vereinfachen, welchen OR-Mapper in welcher Art von Applikation eingesetzt werden soll. Die Entscheidungsgrundlage wird dabei auf Basis von Enterprise Pattern (siehe [10]) gegeben, welche heutzutage in jedem grösseren (Software-) System eine entscheidene Rolle spielen. Abschliessend wird pro Enterpise Pattern eine Empfehlung abgegeben werden, welcher frei verfügbare und Open-Source OR-Mapper im .NET-Umfeld eingesetzt werden soll. Zusätzlich zu diesen Vergleichen und der Empfehlungen wird eine serviceorientierte CRM-Applikation als Proof-of-Concept erstellt.

1.4 Vorgehensweise

Software lässt sich nach einer Vielzahl von Prozessen und Modellen entwickeln. Von iterativen Vorgehen wie Scrum über komplexe und vergleichsweise starre Modelle wie RUP bis hin zu klassischen, linearen Vorgehen wie dem Wasserfallmodell oder dem V-Modell. Nach [15] ist die Auswahl des Entwicklungsprozesses eine der kritischsten Entscheidungen, die man bei einem Softwareprojekt treffen muss. Häufig besitzen Unternehmungen bereits etablierte, auf sie zugeschnittene Entwicklungsmodelle, die mehr oder weniger gut zur Organisation der Unternehmung und dem jeweiligen Projekt passen. Ein ungünstig gewählter oder nicht vollständig eingeführter und gelebter Entwicklungsprozess ist nach [15] einer der Hauptgründe wieso Softwareprojekte mit Qualitätsmängeln, Budgetüberschreitungen oder zeitlichen Verzögerungen zu kämpfen haben.

Für dieses Projekt wurde das Wasserfallmodell als Entwicklungsprozess ausgewählt. Das Wasserfallmodell teilt die Softwareentwicklung meist in fünf verschiedene Phasen ein (siehe Abbildung 1.1). Dabei kann jeweils erst mit der nächsten Phase begonnen werden wenn die Lieferergebnisse und die Ergebnisdokumentation der vorhergehenden Phase fertiggestellt und abgenommen worden sind. Das Wasserfallmodell wurde ausgewählt, da dieses Projekt gut linear abgearbeitet werden kann und da der Betreuer als einziger externer Stakeholder des Projekts zu festdefinierten Zeitpunkten (namtlich dem Kick-Off und dem Design-Review) Einfluss auf das Projekt ausüben kann und danach keine andere Möglichkeit besitzt, im Projektverlauf zu intervenieren. Die grössten Nachteile des Wasserfallmodells sind nach [4] Abgrenzugsprobleme zwischen den einzelnen Phasen sowie die Schwierigkeit des Abschlusses einzelner Phasen da diese nur mit viel (zusätzlichem) Aufwand isoliert von anderen Phasen abgeschlossen werden können. Dadurch dass der Betreuer nur in der Anforderungs- und Konzeptphase Einfluss auf das Projekt nehmen kann und der Student alleinig den Abschluss der Phasen steuert und verifiziert sowie den Ablauf der Phasen innerhalb des Projekts steuert, können diese Nachteile umgangen werden.

Zu Beginn des Wasserfallmodells steht das Sammeln und Dokumentieren der Anforderungen (Requirements) im Vordergrund. Wenn die Anforderungen umfänglich und in hohem Detaillierungsgrad niedergeschrieben sind, werden diese vom Auftraggeber abgenommen und das Projekt geht in die Phase Design über. Die zu entwickelnde Software

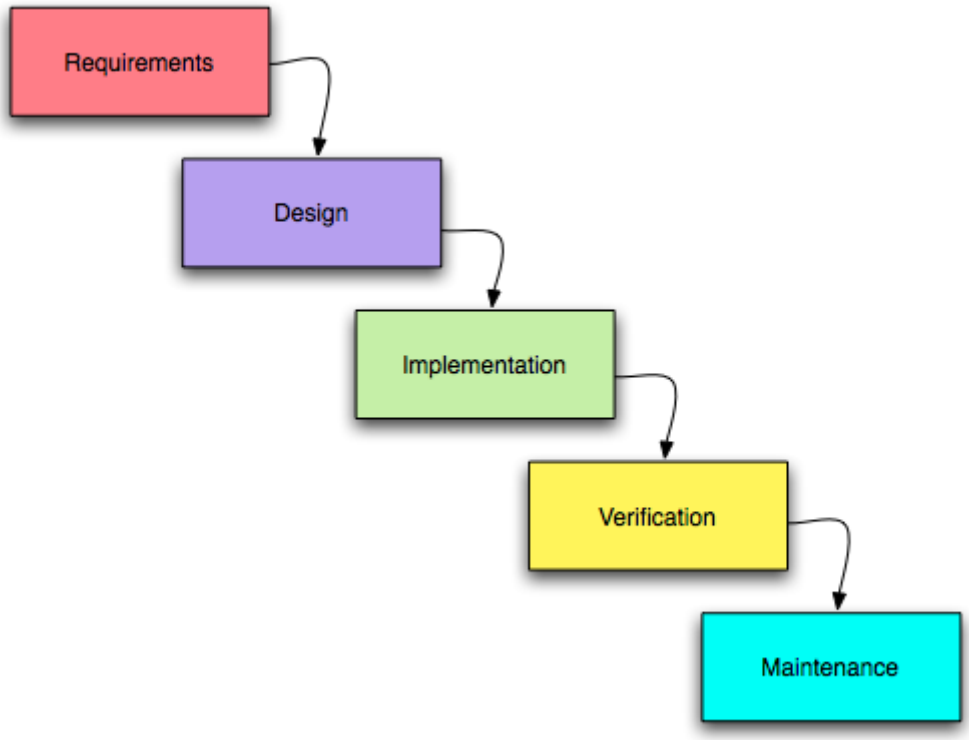

Abbildung 1.1: Wasserfallprozess nach [6]

Phase	Kapitelüberschrift	Siehe Seite
Requirements	Anforderungen	12
Design	Konzept und Architektur	35
Implementation	Implementierung	55
Verification	Verifikation	69

Tabelle 1.1: Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument

wird auf verschiedenen Ebenen von Softwarearchitekten designed und eine Blaupause wird erstellt, nach welcher sich die Softwareentwickler in der Implementationsphase zu halten haben. Das Design sollte einen Plan beinhalten, welcher die Implementierung der Anforderungen aufzeigt.

Wenn das Design fertiggestellt worden ist, wird dieses von den Entwicklern in Programmcode umgesetzt. Gegen Ende der Implementierungsphase werden die Softwarekomponenten verschiedener Teams integriert und als Gesamtsystem zum Einsatz gebracht. Nachdem die Implementierungs- und Integrationsphase abgeschlossen ist, wird das Softwareprodukt getestet und allfällige Fehler aus früheren Phasen werden zu diesem Zeitpunkt behoben. Danach wird das Softwareprodukt installiert und später in der Wartungsphase (Maintenance) um weitere Funktionalitäten erweitert beziehungsweise werden neu entdeckte Bugs behoben.

Auch die Struktur dieses Projekts und der dazugehörigen Dokumentation hält sich an den Wasserfallprozess nach [6] (siehe Tabelle 1.1). Die Phase Maintenance wird dabei ausgelassen, da sich die innerhalb des Projekts entwickelte Applikation nach Abschluss der Verifizierungsphase noch im Protoypenstadium befinden wird und nicht dem Reifegrad einer Applikation entspricht, die in die Wartung übergehen kann.

1.5 Projektplanung

Nach Reglement der ZHAW (siehe [16]) muss die Bachelorarbeit sechs Monate nach Freigabe beendet sein. Um die Planbarkeit der Arbeit zu erhöhen wurde das Projekt bei Projektstart in einzelne Phasen unterteilt, welche auf die vorgängig bekannten Termine als Meilensteine enden.

Ziel dieser Phasenplanung ist es, möglichst frühzeitig im Verlauf der Bachelorarbeit Teile der erwarteten Resultate sowie der Dokumentation fertiggestellt zu haben, so dass das Risiko von Qualitätseinbussen der Abgaberesultate durch überhastiges fertigstellen, gering ist. Ausserdem sinkt das Wahrscheinlichkeit, dass nicht alle erwarteten Resultate der Arbeit geliefert werden können, da die Lieferergebnisse bei jeweiligem Phasenende schon in abgabefertiger Qualität vorliegen.

1.5.1 Phasenplanung

Nach dem Wasserfallmodell wird das Projekt in einzelne Phasen eingeteilt, die zu einem bestimmten Zeitpunkt mit vordefinierten Endergebnissenen enden. Bei Erreichen

Phase	Start	Ende	Endet in Meilenstein
Themenevaluation	02.10.2012	16.11.2012	
Erstellung der Aufgabenstellung	17.11.2012	07.01.2013	M1
Erfassen der Anforderungen	08.01.2013	30.01.2013	M2
Erstellen der Marktübersicht	31.01.2013	02.03.2013	
Auswahl und Vergleich ausgewählter OR-Mapper	03.03.2013	28.03.2013	
Implementierung Proof of Concept	29.03.2013	22.05.2013	M3
Erstellung Produktempfehlung	23.05.2013	01.06.2013	
Finalisierung Dokumentation	02.06.2013	18.06.2013	M4
Erarbeiten der Präsentation	18.06.2013	01.07.2013	M5

Tabelle 1.2: Phasenplan

des Endzeitpunkts und bei Lieferung aller Endergebnisse geht das Projekt in die nächste Phase über. Die Phasen dieser Semesterarbeit wurden so modelliert, dass ihr Endzeitpunkt möglichst mit dem Erreichen eines Meilensteins zusammenfällt (siehe Tabelle 1.2). Das heisst, dass bei Erreichen des Meilensteins die vorhergehende Phase zwingend abgeschlossen sein muss.

Vor dem eigentlichen Projektstart werden geeigeneten Themen für die Semesterarbeit evaluiert, ein Betreuer gesucht sowie ein Projektantrag in EBS erfasst. Diese Phase endet mit dem Kick-Off-Meeting zwischen Betreuer und Student sowie der formalen Freigabe der Semesterarbeit durch die Studiengangsleitung. Das Erheben und Dokumentieren der Anforderungen ist die erste Phase des eigentlichen Projekts und mündet in der Konzepterarbeitung auf Basis der Anforderungen. Ist das Konzept vollständig abgeschlossen, findet das Design-Review statt, bei welchem der Betreuer das Konzept begutachtet und allfällige Anpassungen daran vorschlägt. Nach Fertigstellung des Konzepts folgt die Implementierungsphase und Testphase, nach denen die Arbeit abgegeben werden muss. In der letzten Phase des Projekts wird die Präsentation erarbeitet, die an der Schlusspräsentation (Meilenstein M5) vorgetragen wird.

1.5.2 Meilensteine

Ein Meilenstein ist ein Ereignis von besonderer Bedeutung und stellt ein (Zwischen-) Ziel innerhalb eines Projekts dar. Meilensteine werden typischerweise von Personen oder Organisationen ausserhalb des Projekts vorgegeben und passen mit den im vorhergehenden Kapitel definierten Phasenenden überein.

Die Meilensteine des Projekts sind in der Tabelle 1.3 ersichtlich.

Bezeichner	Meilenstein	Datum
M1	Freigabe der Arbeit	07.01.2013
M2	Kick-Off	30.01.2013
M3	Design-Review	22.05.2013
M4	Abgabe der Arbeit	18.06.2013
M5	Schlusspräsentation	02.07.2013

Tabelle 1.3: Meilensteine

2 Anforderungen

In der Phasenplanung (siehe Kapitel 1.5.1 auf Seite 9) wurde festgelegt, dass in der ersten Projektphase die Anforderungen erhoben werden. Dazu wird zu Beginn dieses Kapitels der Begriff Anforderung definiert und es wird auf verschiedene Arten von Anforderungen näher eingegangen. Anschliessend werden die Anforderungen an den Persistenzlayer einer serviceorientierten Architektur festgehalten sowie die Anforderungen der betrachteten Enterprise Pattern erhoben. In einem letzten Schritt wird der Systemkontext der CRM-Applikation beschrieben und eingegrenzt sowie die konkreten Anforderungen an die CRM-Applikation als Use-Cases modelliert und spezifiziert. Der Abschluss dieses Kapitels wie auch der Anforderungsphase selbst bildet das Entwerfen der für den Benutzer sichtbaren Dialogfenster.

2.1 Was sind Anforderungen?

Die erste Phase des Wasserfallmodells beschäftigt sich mit den Anforderungen an das zu entwickelnde Softwareprodukt. Damit das Entwicklungsprodukt zum Erfolg geführt werden kann, muss zunächst bekannt sein, was die Anforderungen an das System sind und diese müssen geeignet dokumentiert sein. Nach [7] wird eine Anforderung wie folgt definiert:

Anforderung Eine Anforderung ist:

1. Eine Bedingung oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems zur Erreichung eines Ziels benötigt wird.
2. Eine Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder andere, formell vorgegebene Dokumente zu erfüllen.
3. Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft gemäss 1. oder 2.

—

Die Anforderungen an das im Rahmen dieser Semesterarbeit zu entwickelnde System werden in Use-Case-Diagrammen modellhaft dargestellt und als Use-Case-Spezifikationen ausformuliert. Auf eine natürlichsprachige Dokumentation der Anforderungen wird verzichtet, da die modellierten Anforderungen innerhalb der Use-Case-Diagramme verständlich genug sind und zu den Use-Case-Diagrammen noch detaillierte Use-Case-Spezifikationen vorhanden sind, welche die gesamte Anforderung abdecken.

2.1.1 Arten von Anforderungen

Nach [8] unterscheidet man im Allgemeinen zwischen drei Arten von Anforderungen:

- Funktionale Anforderungen legen die Funktionalität fest, die das geplante System zur Verfügung stellen soll. Sie werden typischerweise in Funktions-, Verhaltens- und Strukturanforderungen unterteilt.
- Qualitätsanforderungen legen gewünschte Qualitäten des zu entwickelnden Systems fest und beeinflussen häufig in grösserem Umfang als die funktionalen Anforderungen die Gestalt der Systemarchitektur. Typischerweise beziehen sich Qualitätsanforderungen auf die Performance, die Verfügbarkeit, die Zuverlässigkeit, die Skalierbarkeit oder die Portabilität des betrachteten Systems. Anforderungen dieses Typs werden häufig auch der Klasse "nicht funktionaler Anforderungen" zugeordnet.
- Randbedingungen[1] können von den Projektbeteiligten nicht beeinflusst werden. Randbedingungen können sich sowohl auf das betrachtete System beziehen (z.B. "Das System soll über Webservices mit Aussensysteme kommunizieren") als auch auf den Entwicklungsprozess des Systems (z.B. "Das System soll bis spätestens Mitte 2013 am Markt verfügbar sein"). Randbedingungen werden, im Gegensatz zu funktionalen Anforderungen und Qualitätsanforderungen, nicht umgesetzt, sondern schränken die Umsetzungsmöglichkeiten, das heisst den Lösungsraum im Entwicklungsprozess ein.

In diesem Dokument werden der Einfachheit halber nur funktionale Anforderungen beschrieben. Qualitätsanforderungen betreffend der Leistungsfähigkeit oder Wartbarkeit des Systems werden nicht erhoben, da diese für den Prototypen nur eingeschränkt relevant sind und innerhalb dieses Projekts darauf keinen Wert gelegt werden soll.

Rahmenbedingungen werden nicht implizit festgehalten sondern ergeben sich einerseits aus [16] sowohl der technischen Umgebung des Systems. Technische Rahmenbedingungen die das Projekt beeinflussen werden im Konzept beziehungsweise in der Implementierung berücksichtigt und entsprechend hervorgehoben.

2.2 Anforderungen Persistenzlayer

Dieses Kapitel enthält die Anforderungen an den Persistenzlayer einer serviceorientierten Architektur und wird auch in den Anforderungen der CRM-Applikation (siehe Kapitel 2.4 auf Seite 15) berücksichtigt. Anforderungen werden normalerweise aus Aktorensicht definiert und sind nicht auf einen Layer einer Applikation ausgelegt. Damit jedoch später eine Vergleichsmöglichkeit verschiedener OR-Mapper gegeben werden kann müssen diese im Rahmen dieser Bachelorarbeit jedoch auf diese Art und Weise niedergeschrieben sein. Anstelle einer modellbasierten Beschreibung der Anforderungen werden diese natürlichsprachig dokumentiert. Dazu wird die Satzschablone nach [8] verwendet und die

[1] auch Rahmenbedingungen genannt

Bezeichner	Anforderung
P-F-1	Das System muss CRUD-Operationen pro persistierbarer Entität zur Verfügung stellen.
P-F-2	Das System muss eine einzelne Instanz einer Entität auf Basis seiner ID zurückgeben könne
P-F-3	Das System muss alle Instanzen einer Entität zurückgeben können.
P-F-4	Das System muss eine oder mehrer Instanzen einer Entität auf Basis von Suchkriterien zurü
P-F-5	Das System muss eine einzelne Instanz persistieren können.
P-F-6	Das System muss mehrere Instanzen gleichzeitig persistieren können.
P-F-7	Das System muss eine einzelne Instanz löschen können.
P-F-8	Das System muss mehrere Instanzen gleichzeitig löschen können.
P-F-9	Das System muss dem Administrator die Möglichkeit bieten den Ablauf sämtlicher Operatio

Tabelle 2.1: Funktionale Anforderungen an den Persistenzlayer

Bezeichner	Anforderung
P-Q-1	Das System muss fähig sein bei sämtlichen Operationen ACID zu garantieren.
P-Q-2	Das System muss die Datenbankmanagementssysteme Microsoft SQL Server und Oracle Da
P-Q-3	Das System muss dem Entwickler die Möglichkeit bieten die externe Schnittstelle des Syste
P-Q-4	Das System soll keine Logik auf die Datenbank auslagern.
P-Q-5	Das System muss bestimmte Performancewerte erfüllen[2].

Abbildung 2.1: Qualitätsanforderungen an den Persistenzlayer

Anforderungen werden funktionalen Anforderungen, Qualitätsanforderungen und Randbedingungen aufgeteilt.

Die Anforderungen stammen aus eigenen Erfahrungen mit serviceorientierten Architekturen sowie [2], [3] und [5].

2.2.1 Funktionale Anforderungen

Tabelle (2.1) enthält die funktionalen Anforderungen in natürlichsprachiger Form an den Persistenzlayer einer serviceorientierten Architektur. Der Begriff "das System" stehet dabei für den Persistenzlayer als eigenständiges System auf welches über einen Webservice zugegriffen werden kann.

Bezeichner	Anforderung
P-R-1	Das System muss alle Operationen als SOAP-basierter Webservice anbieten.
P-R-2	Das System muss dem Entwickler die Möglichkeit bieten ohne Wissen über die Datenbank c
P-R-3	Das System muss dem Entwickler für sämtliche Operationen ein Resultat entweder in Form
P-R-4	Das System muss dem Entwickler die Möglichkeit bieten durch ein anderes System (anderen

Abbildung 2.2: Randbedingungen an den Persistenzlayer

Bezeichner	Anforderung
T-1	Das System muss alle Datenbankoperationen durch einen dünnen Datenbankwrapper direkt
T-2	Das System muss alle Operationen atomar ausführen können.

Tabelle 2.2: Anforderungen Transaction Script

2.2.2 Qualitätsanforderungen

2.2.3 Randbedingungen

2.3 Anforderungen Enterprise Pattern

2.3.1 Transaction Script

2.3.2 Domain Model

2.3.3 Table Module

2.3.4 Active Record

2.4 Anforderungen CRM-Applikation

2.4.1 Systemkontext

Als erster Schritt in der Erhebung und Dokumentierung der Anforderung wird der Systemkontext ermittelt. Es wird eine Sollperspektive eingenommen, das heisst, es wird eine Annahme getroffen, wie sich das geplante System in die Realität integrieren wird. Hierdurch wird der Realitätsausschnitt identifiziert, der das System und damit potenziell

Bezeichner	Anforderung
D-1	Das System muss eine minimale Kopplung des Domain Models an andere Layer ermöglichen
D-2	Das System muss die Möglichkeiten einer Spezifikation bieten, mit welcher der Teil eines zu
D-3	Das System muss den Entiäten sowohl Informationen sowie Verhalten zuweisen.
D-4	Das System soll Design Patterns verwenden um ein reiches Domain Model zu erstellen.
D-5	Das System muss das Domain Model von der Datenbank vollständig entkoppeln.

Tabelle 2.3: Anforderungen Domain Model

Bezeichner	Anforderung
T-1	Das System muss die Objektstruktur analog der Datenbankstruktur abbilden.
T-2	Das System muss eine einzelne Instanz zur Verfügung stellen, welche alle Geschäftslogik alle
T-3	Das System muss eine einfache Domänenlogik verwenden.

Tabelle 2.4: Anforderungen Table Module

Bezeichner	Anforderung
A-1	Das System muss die Objektstruktur analog der Datenbankstruktur abbilden.
A-2	Das System muss keine Konversion von Objektdatentypen und Datentypen der Datenbankt
A-3	Das System muss eine einfache Domänenlogik verwenden.

Tabelle 2.5: Anforderungen Active Record

auch dessen Anforderungen beeinflusst. Um die Anforderungen an das geplante System korrekt und vollständig spezifizieren zu können, ist es notwendig, die Beziehung zwischen den einzelnen materiellen und immateriellen Aspekten im Systemkontext und dem geplanten System exakt zu definieren. Der für die Anforderungen des Systems relevante Ausschnitt der Realität wird als Systemkontext bezeichnet (nach [8]).

Der Ursprung der Anforderungen des Systems liegt im Systemkontext des geplanten Systems. Aus diesem Grund wird der Systemkontext vor Erhebung und Dokumentierung der Anforderungen festgelegt. Der Systemkontext des Systems CRM-Applikation ist in Abbildung 2.3 als Modell dargestellt.

Die Benutzer als Stakeholder an das System erfassen Kunden und Kontakte und befinden sich innerhalb des Systemkontexts, da sie direkt mit dem System interagieren. Die Prozesse als geben Rahmenbedingungen vor, wie neue Kontakte oder Kunden erfasst werden können. Diese sind in der Geschäftslogik spezifiziert und integragieren über Wofklows mit dem System. Der Administrator verwaltet die Benutzer des Systems.

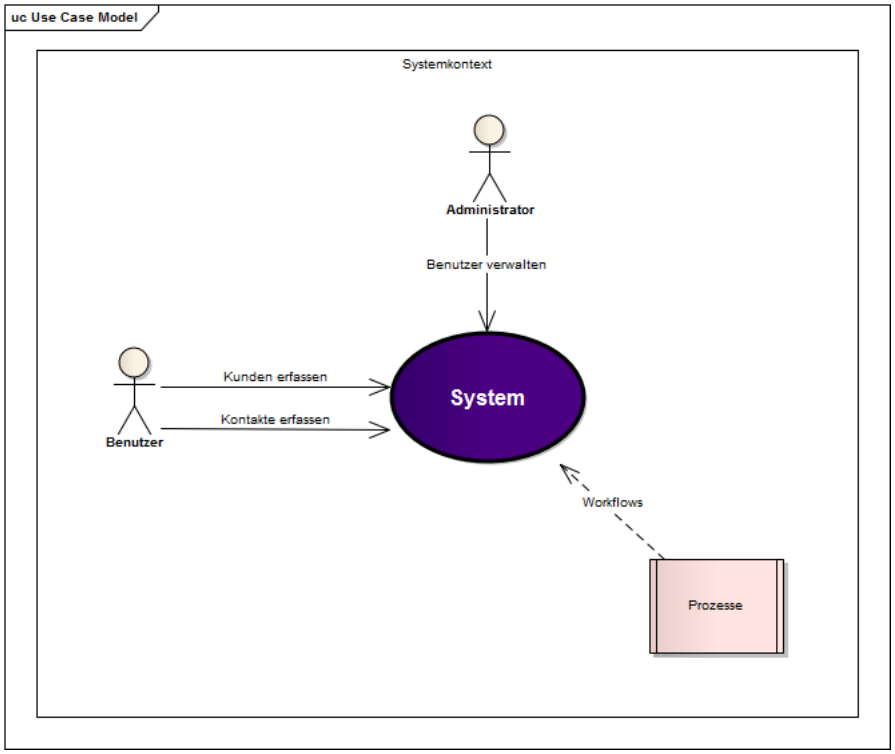

Abbildung 2.3: Systemkontext

2.4.2 Use-Case-Spezifikationen

Nach [8] zeigen Use-Case-Diagamme die aus einer externen Nutzungssicht wesentlichen Funktionalitäten des betrachteten Systems sowie spezifische Beziehungen der einzelnen Funktionalitäten untereinander beziehungsweise zu Aspekten in der Umgebung des Systems. Abgesehen vom Namen des Use-Cases und dessen Beziehungen dokumentieren Use-Case-Diagramme keinerlei weitere Informationen über die einzelnen Use-Cases, wie z.B. die Systematik der Interaktion eines Use-Case mit Akteuren in der Umgebung. Diese Informationen werden unter Verwendung einer geeigneten Schablone zusätzlich zum Use-Case-Diagramm textuell als Use-Case-Spezifikation festgehalten.

Alle funktionalen Anforderungen (siehe Kapitel 2.1.1 auf Seite 13) werden als Use-Cases modelliert und spezifiziert[3]. Als Quellen der Anforderungen dienen der Betreuer, das Reglement der ZHAW betreffend der Semesterarbeit sowie der Student in der Rolle des Benutzers des Systems. Zusätzlich zu den Use-Cases und den dazugehörigen Use-Case-Spezifikationen wird vorgängig in Prosatext der Anwendungsfall beschrieben. Aus Gründen der Übersichtlichkeit und der limitierten Gesamtfunktionaliät des Systems stel-

[3] Die verwendete Schablone stammt aus [8] und dient zur zweckmässigen Strukturierung von Typen von Informationen, die einen Use-Case betreffen. Die vorgeschlagenen Abschnitte der Schablone Autor, Quelle, Verantwortlicher und Qualität werden ausgelassen, da sie im Rahmen dieses Projekts keinen zusätzlichen Nutzen bringen.

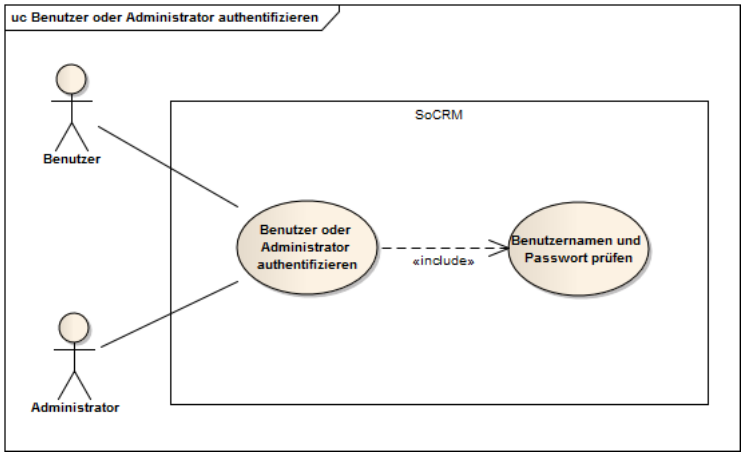

Abbildung 2.4: Use-Case Benutzer oder Administrator authentifizieren

len diese Use-Cases die primären Anforderungen an das zu entwickelnde Softwaresystem dar.

2.4.2.1 Benutzer oder Administrator authentifizieren

Ein Benutzer oder Administrator möchte mit dem System interagieren und authentifiziert sich dazu mit seinem Benutzernamen und Passwort beim System. Das System prüft den eingegebenen Benutzernamen und das Passwort und der Benutzer oder Administrator wird authentifziert falls diese dem System bekannt sind. Er kann nun weitere Aktionen mit dem System tätigen.

Abschnitt	Inhalt
Bezeichner	UC1
Name	Benutzer oder Administrator authentifizieren
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: mittel
Kritikalität	Hoch
Beschreibung	Der Benutzer oder Administrator authentifiziert sich beim System.
Auslösendes Ereignis	Der Benutzer oder Administrator möchte mit dem System interagieren.
Akteure	Benutzer, Administrator
Vorbedingung	Der Benutzer oder Administrator ist nicht schon beim System authentifiziert.
Nachbedingung	Der Benutzer oder Administrator ist authentifziert und kann mit dem System interagieren.
Ergebnis	Eine Authentifikation wird ausgestellt.
Hauptszenario	1. Der Benutzer oder Administrator gibt seinen Benutzernamen und Passwort ein. 2. Der Benutzer oder Administrator erfragt die Authentifizierung vom System. 3. Das System erstellt eine Authentifikation und übergibt dieses dem Benutzer oder Administrator.
Alternativszenarien	2a. Der Benutzername oder das Passwort ist inkorrekt. 2a1. Der Benutzer oder Administrator wird aufgefordert seinen Benutzernamen oder Passwort zu überprüfen und erneut einzugeben.
Ausnahmeszenarien	Auslösendes Ereignis: Der Benutzer oder Administrator kann keine Verbindung zum System herstellen.

Tabelle 2.6: Use-Case-Spezifikation Benutzer oder Administrator authentifizieren

2.4.2.2 Kunden anzeigen

Der Benutzer ist beim System authentifiziert und zeigt aller Kunden an.

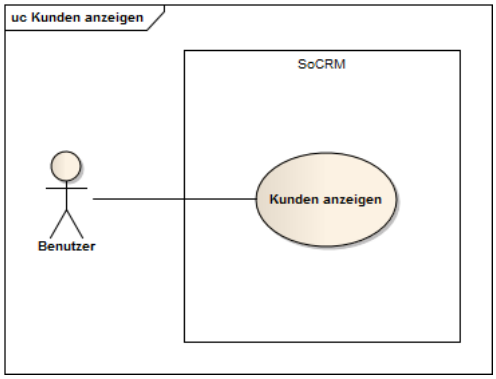

Abbildung 2.5: Use-Case Kunden anzeigen

Abschnitt	Inhalt
Bezeichner	UC2
Name	Kunden anzeigen
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Hoch
Beschreibung	Der Benutzer zeigt eine Liste aller Kunden an.
Auslösendes Ereignis	Der Benutzer möchte eine Liste aller Kunden anzeigen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann für einen angezeigten Kunden einen Kontakt erfassen oder den Kunden löschen.
Ergebnis	Eine Liste mit Kunden wird angezeigt.
Hauptszenario	1. Der Benutzer gibt den Vor- oder Nachnamen des zu suchenden Kunden ein. 2. Der Benutzer gibt den Namen des Arbeitgebers des zu suchenden Kunden ein. 3. Das System zeigt dem Benutzer die passenden Kunden an.
Alternativszenarien	1a. Der Benutzer gibt keinen Namen ein. 2a. Der Benutzer gibt keinen Firmennamen ein.
Ausnahmeszenarien	Keine

Tabelle 2.7: Use-Case-Spezifikation Kunden anzeigen

2.4.2.3 Kunde hinzufügen

Der Benutzer fügt dem System einen neuen Kunden hinzu. Jeder Kunde hat einen Arbeitgeber, ist dieser dem System noch nicht hinzugefügt so kann er auch beim Hinzufügen des Kunden erstellt werden. Zusätzlich kann jedem Kunden eine oder mehrere E-Mail-

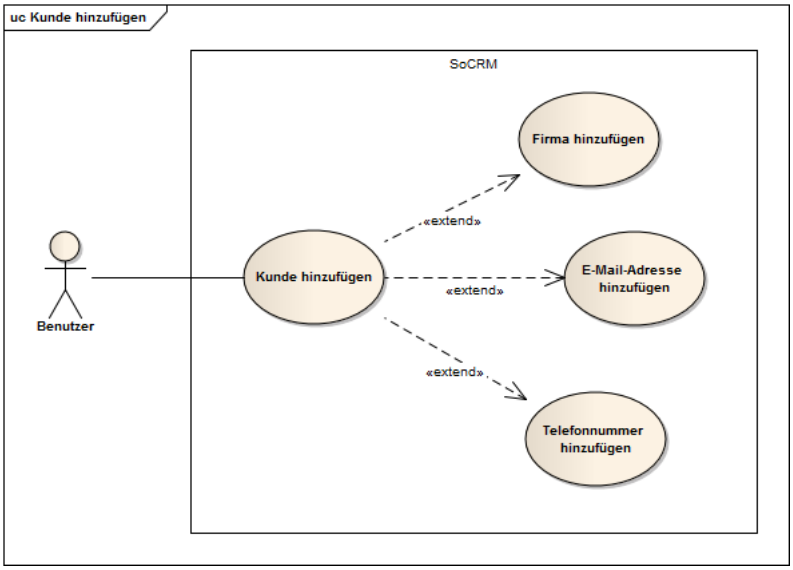

Abbildung 2.6: Use-Case Kunde hinzufügen

Adressen sowie Telefonnummern hinzugefügt werden.

Abschnitt	Inhalt
Bezeichner	UC3
Name	Kunde hinzufügen
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Mittel
Beschreibung	Der Benutzer fügt dem System einen neuen Kunden hinzu.
Auslösendes Ereignis	Der Benutzer möchte dem System einen neuen Kunden hinzufügen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann für den neu hinzugefügten Kunden Kontakte erfassen.
Ergebnis	Der hinzugefügte Kunde wird angezeigt.
Hauptszenario	1. Der Benutzer gibt den Namen sowie Adresse des Kunden ein. 2. Der Benutzer wählt den Arbeitgeber des neuen Kunden aus. 3. Der Benutzer fügt eine oder mehrere E-Mail-Adressen dem Kunden hinzu. 4. Der Benutzer fügt eine oder mehrere Telefonnummern dem Kunden hinzu. 5. Der Benutzer speichert den Kunden und dieser wird ihm angezeigt.
Alternativszenarien	2a. Der Arbeitgeber existiert nicht. Der Benutzer kann diesen als Firma erfassen (siehe Kapitel 2.4.2.6 auf Seite 24).
Ausnahmeszenarien	Auslösenden Ereignis: Der Kunde existiert schon. Der Benutzer wird aufgefordert zu überprüfen ob die Daten fehlerhaft eingetragen wurden oder ob dies ein Duplikat darstellt.

Tabelle 2.8: Use-Case-Spezifikation Kunde hinzufügen

2.4.2.4 Kunde löschen

Der Anwendungsfall "Kunde löschen" wird ausgeführt wenn ein Benutzer einen im System erfassten Kunden löschen möchte. Besitzt dieser Kunde Kontakte so werden diese mit dem Kunden zusammen gelöscht.

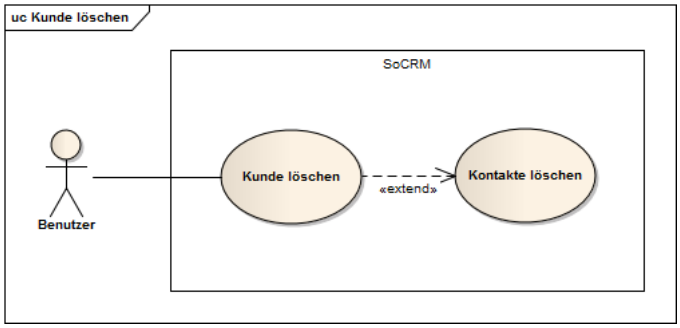

Abbildung 2.7: Use-Case Kunde löschen

Abschnitt	Inhalt
Bezeichner	UC4
Name	Kunde löschen
Priorität	Wichtigkeit für Systemerfolg: mittel Technologisches Risiko: niedrig
Kritikalität	Hoch
Beschreibung	Der Benutzer löscht einen Kunden aus dem System.
Auslösendes Ereignis	Der Benutzer möchte einen Kunden aus dem System löschen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann für den gelöschten Kunden keine Kontakte mehr erfassen und findet diesen nicht mehr (siehe Kapitel 2.4.2.2 auf Seite 19).
Ergebnis	Die Löschung wird bestätigt.
Hauptszenario	1. Der Benutzer wählt den zu löschenden Kunden aus. 2. Der Kunde wird vom System gelöscht.
Alternativszenarien	1a. Falls der Kunde Kontakte besitzt wird der Benutzer darauf hingewiesen dass auch die zum Kunden gehörigen Kontakte gelöscht werden.
Ausnahmeszenarien	Keine.

Tabelle 2.9: Use-Case-Spezifikation Kunde löschen

2.4.2.5 Firmen anzeigen

Der Benutzer ist beim System authentifiziert und zeigt alle Firmen an.

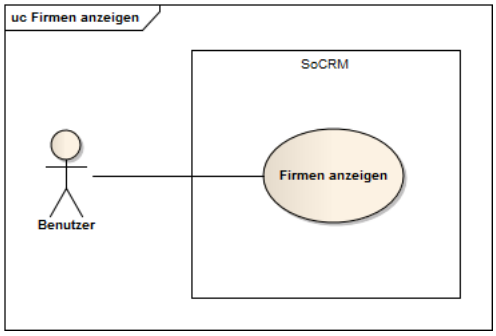

Abbildung 2.8: Use-Case Firmen anzeigen

Abschnitt	Inhalt
Bezeichner	UC5
Name	Firmen anzeigen
Priorität	Wichtigkeit für Systemerfolg: mittel Technologisches Risiko: niedrig
Kritikalität	Mittel
Beschreibung	Der Benutzer zeigt eine Liste aller Firmen an.
Auslösendes Ereignis	Der Benutzer möchte eine Liste aller Firmen anzeigen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann für eine angezeigte Firma einen Kunden hinzufügen oder die Firma löschen.
Ergebnis	Eine Liste mit Firmen wird angezeigt.
Hauptszenario	1. Der Benutzer gibt den Namen der zu suchenden Firma ein. 2. Der Benutzer gibt das Land der zu suchenden Firma ein. 3. Das System zeigt dem Benutzer die passenden Firmen an.
Alternativszenarien	1a. Der Benutzer gibt keinen Namen ein. 2a. Der Benutzer gibt kein Land an.
Ausnahmeszenarien	Keine.

Tabelle 2.10: Use-Case-Spezifikation Firmen anzeigen

2.4.2.6 Firma hinzufügen

Der Benutzer möchte eine neue Firma dem System hinzufügen. Kontakte können nur auf Mitarbeitern von Firmen erfasst werden.

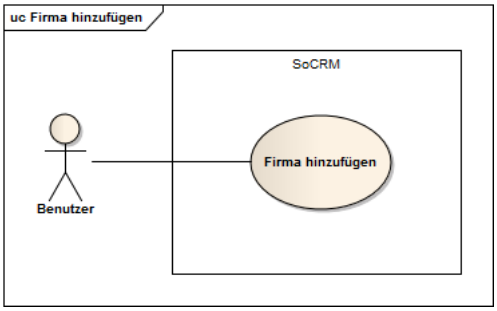

Abbildung 2.9: Use-Case Firma hinzufügen

Abschnitt	Inhalt
Bezeichner	UC6
Name	Firma hinzufügen
Priorität	Wichtigkeit für Systemerfolg: mittel Technologisches Risiko: niedrig
Kritikalität	Mittel
Beschreibung	Der Benutzer fügt eine neue Firma hinzu.
Auslösendes Ereignis	Der Benutzer möchte eine neue Firma hinzufügen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann der neu hinzugefügten Firma Kunden (Mitarbeiter) hinzufügen.
Ergebnis	Die hinzugefügte Firma wird angezeigt.
Hauptszenario	1. Der Benutzer gibt den Namen sowie Adresse der Firma ein. 2. Der Benutzer speichert die Firma und diese wird ihm angezeigt.
Alternativszenarien	1a. Die Firma existiert bereits. Der Benutzer wird aufgefordert zu überprüfen ob die Daten fehlerhaft eingetragen wurden oder ob dies ein Duplikat darstellt.
Ausnahmeszenarien	Keine

Tabelle 2.11: Use-Case-Spezifikation Firma hinzufügen

2.4.2.7 Firma löschen

Wenn der Benutzer eine Firma löschen möchte, wird dieser Anwendungsfall ausgeführt. Besitzt diese Firma Angestellte so werden diese mit der Firma zusammen gelöscht. Besitzen dei Angestellte Kontakte so werden die Kontakte mit dem Angestellten zusammen gelöscht.

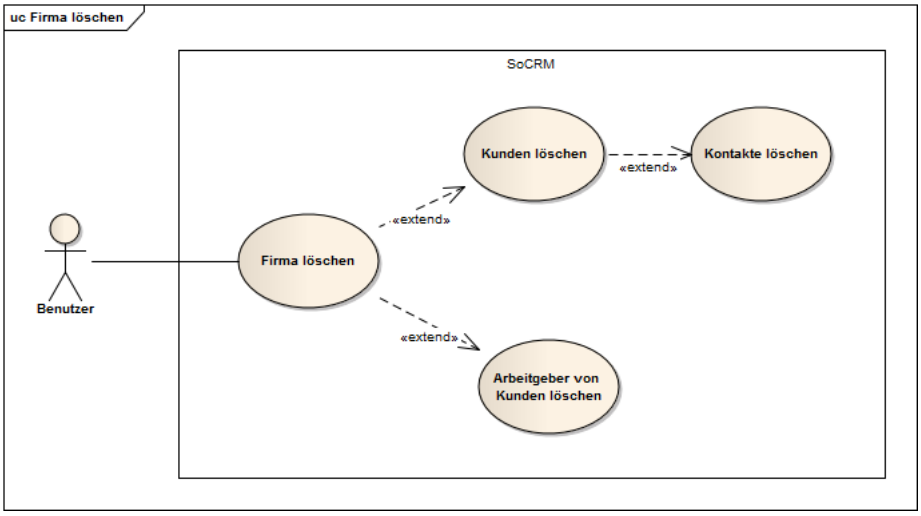

Abbildung 2.10: Use-Case Firma löschen

Abschnitt	Inhalt
Bezeichner	UC7
Name	Firma löschen
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: mittel
Kritikalität	Mittel
Beschreibung	Der Benutzer löscht eine Firma.
Auslösendes Ereignis	Der Benutzer möchte eine Firma löschen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann der gelöschten Firma keine Mitarbeiter mehr hinzufügen und findet diese auch nicht mehr (siehe Kapitel 2.4.2.5 auf Seite 23).
Ergebnis	Die Löschung wird bestätigt.
Hauptszenario	1. Der Benutzer wählt die zu löschende Firma aus. 2. Die Firma wird vom System gelöscht.
Alternativszenarien	1a. Falls die Firma Angestellte besitzt (Kunden) wird der Benutzer darauf hingewiesen dass auch die zur Firma gehörigen Angestellten gelöscht werden. 1b. Falls der Angestellte Kontakte besitzt wird der Benutzer darauf hingewiesen dass auch die zum Angestellten gehörigen Kontakte gelöscht werden.
Ausnahmeszenarien	Keine.

Tabelle 2.12: Use-Case-Spezifikation Firma löschen

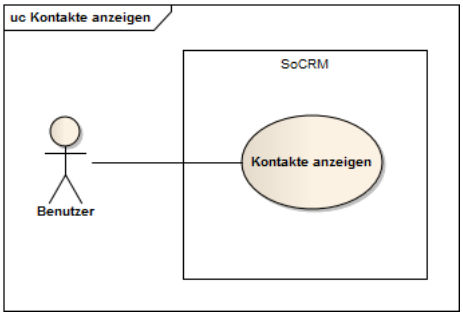

Abbildung 2.11: Use-Case Kontakte anzeigen

2.4.2.8 Kontakte anzeigen

Abschnitt	Inhalt
Bezeichner	UC8
Name	Kontakte anzeigen
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Niedrig
Beschreibung	Der Benutzer zeigt eine Liste aller Kontakte an.
Auslösendes Ereignis	Der Benutzer möchte eine Liste aller Kontakte anzeigen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann einen angezeigten Kontakt detaillierter betrachten oder den Kontakt löschen.
Ergebnis	Eine Liste mit Kontakten wird angezeigt.
Hauptszenario	1. Der Benutzer gibt den Vor- oder Nachnamen des Kunden an, dessen Kontakte er suche möchte. 2. Der Benutzer wählt das Kontaktmedium aus, über welches der Kontakt mit dem Kunden zustandegekommen ist.
Alternativszenarien	1a. Der Benutzer gibt keine Namen ein. 2a. Der Benutzer wählt kein Kontaktmedium aus.
Ausnahmeszenarien	Keine.

Tabelle 2.13: Use-Case-Spezifikation Kontakte anzeigen

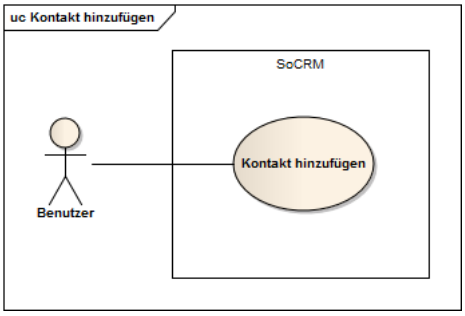

Abbildung 2.12: Use-Case Kontakt hinzufügen

2.4.2.9 Kontakt hinzufügen

Abschnitt	Inhalt
Bezeichner	UC9
Name	Kontakt hinzufügen
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Niedrig
Beschreibung	Der Benutzer erfasst einen neuen Kontakt für einen Kunden.
Auslösendes Ereignis	Der Benutzer möchte einen neuen Kontakt für einen Kunden erfassen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Kontakt ist dem System hinzugefügt.
Ergebnis	Der hinzugefügte Kontakt wird angezeigt.
Hauptszenario	1. Der Benutzer wählt den Kontaktzeitpunkt, das Kontaktmedium und den Inhalt des Kontakts aus. 2. Der Benutzer speichert den Kontakt und dieser wird ihm angezeigt.
Alternativszenarien	Keine.
Ausnahmeszenarien	Keine.

Tabelle 2.14: Use-Case-Spezifikation Kontakt hinzufügen.

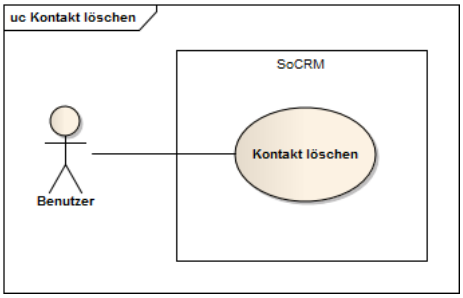

Abbildung 2.13: Use-Case Kontakt löschen

2.4.2.10 Kontakt löschen

Abschnitt	Inhalt
Bezeichner	UC10
Name	Kontakt löschen
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: mittel
Kritikalität	Mittel
Beschreibung	Der Benutzer löscht einen Kontakt.
Auslösendes Ereignis	Der Benutzer möchte einen Kontakt löschen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist beim System authentifiziert.
Nachbedingung	Der Benutzer kann den Kontakt nicht mehr finden (siehe Kapitel 2.4.2.8 auf Seite 27).
Ergebnis	Die Löschung wird bestätigt.
Hauptszenario	1. Der Benutzer wählt den zu löschenden Kontakt aus. 2. Der Kontakt wird vom System gelöscht.
Alternativszenarien	Keine.
Ausnahmeszenarien	Keine.

Tabelle 2.15: Use-Case-Spezifikation Kontakt löschen

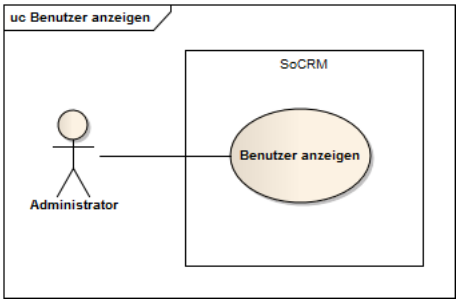

Abbildung 2.14: Use-Case Benutzer anzeigen

2.4.2.11 Benutzer anzeigen

Abschnitt	Inhalt
Bezeichner	UC11
Name	Benutzer anzeigen
Priorität	Wichtigkeit für Systemerfolg: mittel Technologisches Risiko: niedrig
Kritikalität	Niedrig
Beschreibung	Der Administrator zeigt eine Liste aller Benutzer an.
Auslösendes Ereignis	Der Administrator möchte eine Liste aller Benutzer anzeigen.
Akteure	Administrator
Vorbedingung	Der Administrator ist beim System authentifiziert.
Nachbedingung	Der Administrator kann einen angezeigten Benutzer detaillierter betrachten oder den Benutzer löschen.
Ergebnis	Eine Liste mit Benutzern wird angezeigt.
Hauptszenario	1. Der Administrator gibt den Benutzernamen des Benutzers an, den er suchen möchte. 2. Der Administrator wählt die Rolle aus, die die zu suchenden Benutzer innehaben.
Alternativszenarien	1a. Der Administrator gibt keinen Benutzernamen ein. 2a. Der Administrator wählt keine Rolle aus.
Ausnahmeszenarien	Keine.

Tabelle 2.16: Use-Case-Spezifikation Benutzer anzeigen

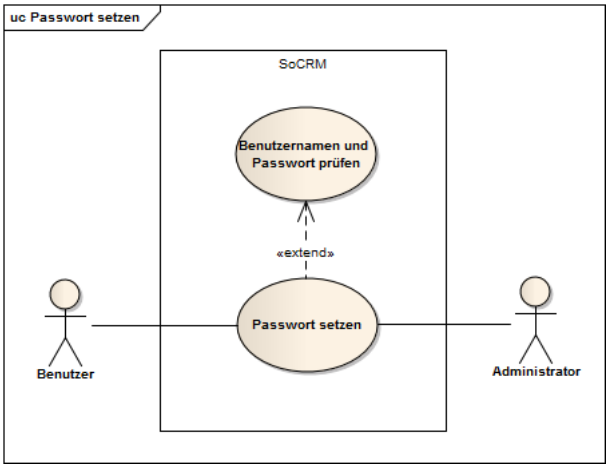

Abbildung 2.15: Use-Case Passwort setzen

2.4.2.12 Passwort setzen

Abschnitt	Inhalt
Bezeichner	UC12
Name	Passwort setzen
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: niedrig
Kritikalität	Mittel
Beschreibung	Der Benutzer setzt sein eigenes Passwort. Der Administrator setzt sein eigenes oder dasjenigen jedes anderen Benutzers.
Auslösendes Ereignis	Der Benutzer möchte sein eigenes Passwort setzen. Der Administrator möchten sein eigenes oder dasjenigen jedes anderen Benutzers setzen.
Akteure	Benutzer, Administrator
Vorbedingung	Der Benutzer oder Administrator ist beim System authentifiziert.
Nachbedingung	Der Benutzer oder Administrator kann sich mit seinem neuen Passwort beim System authentifizieren.
Ergebnis	Eine Bestätigung wird angezeigt.
Hauptszenario	1. Der Benutzer gibt sein altes Passwort sowie das neue Passwort ein. 2. Das System setzt das neue Passwort.
Alternativszenarien	1a. Der Administrator gibt das neue Passwort eines Benutzers ein. 2a. Das alte Passwort des Benutzers ist falsch, der Benutzer wird aufgefordert das alte Passwort erneut einzugeben.
Ausnahmeszenarien	Keine.

Tabelle 2.17: Use-Case-Spezifikation Passwort setzen

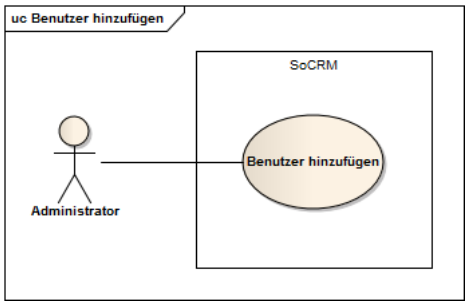

Abbildung 2.16: Use-Case Benutzer hinzufügen

2.4.2.13 Benutzer hinzufügen

Abschnitt	Inhalt
Bezeichner	UC13
Name	Benutzer hinzufügen
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: niedrig
Kritikalität	Niedrig
Beschreibung	Der Administrator erfasst einen neuen Benutzer.
Auslösendes Ereignis	Der Administrator möchte einen neuen Benutzer erfassen.
Akteure	Administrator
Vorbedingung	Der Administrator ist beim System authentifiziert.
Nachbedingung	Der Benutzer ist dem System hinzugefügt und kann sich beim System authentifizieren.
Ergebnis	Der hinzugefügte Benutzer wird angezeigt.
Hauptszenario	1. Der Administrator gibt den Benutzernamen und das Passwort das neuen Benutzers ein und wählt dess Rolle. 2. Der Administrator speichert den Benutzer und dieser wird ihm angezeigt.
Alternativszenarien	Keine.
Ausnahmeszenarien	1a. Der Benutzer existiert bereits. Der Administrator wird aufgefordert zu überprüfen ob die Daten fehlerhaft eingetragen wurden oder ob dies ein Duplikat darstellt.

Tabelle 2.18: Use-Case-Spezifikation Benutzer hinzufügen.

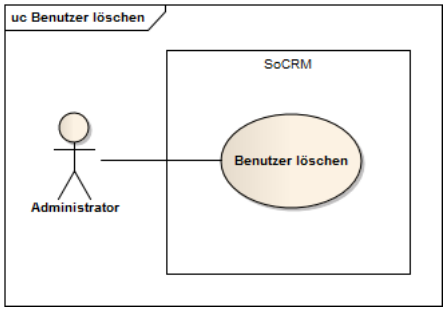

Abbildung 2.17: Use-Case Benutzer löschen

2.4.2.14 Benutzer löschen

Abschnitt	Inhalt
Bezeichner	UC14
Name	Benutzer löschen
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: hoch
Kritikalität	Hoch
Beschreibung	Der Administrator löscht einen Benutzer.
Auslösendes Ereignis	Der Administrator möchte einen Benutzer löschen.
Akteure	Administrator
Vorbedingung	Der Administrator ist beim System authentifiziert.
Nachbedingung	Der Administrator kann den Benutzer nicht mehr finden (siehe Kapitel 2.4.2.11 auf Seite 30). Der gelöschte Benutzer kann sich nicht mehr beim System authentifizieren (siehe Kapitel 2.4.2.1 auf Seite 18).
Ergebnis	Die Löschung wird bestätigt.
Hauptszenario	1. Der Administrator wählt den zu löschenden Benutzer aus. 2. Der Benutzer wird vom System gelöscht.
Alternativszenarien	Keine.
Ausnahmeszenarien	2a. Der Benutzer ist gegenwärtig auf dem System eingeloggt. Der Administrator wird gefragt, ob das System die Authentifizierung löschen soll.

Tabelle 2.19: Use-Case-Spezifikation Benutzer löschen

3 Konzept und Architektur

Die Konzeptphase[1] des Wasserfallmodells behandelt die Entwicklung eines vollständigen und umfassenden Lösungskonzepts auf Basis der dokumentierten Anforderungen (nach [1]). Als Grundlage für das Konzept wird in einem ersten Schritt die Toolchain erfasst, welche weitere (technisch) Rahmenbedingungen für das Konzept und die Archtektur des Systems vorgibt. Danach wird das System aus der Bausteinperspektive betrachtet, es wird von der Komponentenebene bis zur Klassenebene das System modelliert und die Systemarchitektur festgelegt. Als weitere Sicht wird die Laufzeitsicht und die Verteilungssicht des Systems beleuchtet und spezifiziert.

3.1 Marktübersicht

Die Marktübersicht ist an [13] angelehnt.

[1] auch Designphase genannt

Kriterium	Entity Framework	NHibernate	LINQ-to-SQL	OpenAccess ORM	mybatis.NET	Business Logic Toolkit	SubSonic	Dapper	nHydrate	Signum Framework
Anbieter	Microsoft	Open Source	Microsoft	Telerik	Open Source	Open Source	Open Source	Open Source	Open Source	Open Source
Website	http://msdn.microsoft.com/en-us/data/ef.aspx	http://nhforge.org/	http://msdn.microsoft.com/en-us/library/bb386976.aspx	http://www.telerik.com/products/orm.aspx	https://code.google.com/p/mybatisnet/	http://bltoolkit.net/index.htm	http://subsonicproject.com/	https://code.google.com/p/dapper-dot-net/	http://nhydrate.codeplex.com/	http://www.signumframework.com/
Zeitpunkt der ersten Version	August 2005	Oktober 2005	November 2007	November 2009	Oktober 2004	2010	Juli 2009	März 2011	September 2009	March 2009
Aktuelle und untersuchte Version	5.0.0	3.3.3	1.0	2013.1.219.3	3.2.2	4.1.20	3.0.0.4	8b405c70e67b	5.1	2.0
Lizenz	Apache License 2.0	GNU Lesser General Public License	Microsoft .NET Framework Redistribuable EULA	Telerik EULA	Apache License 2.0	MIT License	Mozilla Public License 1.1	Apache License 2.0	MIT License	GNU Library General Public License 2.1
Typ	Komplett	Komplett	Leichtgewichtig	Komplett	Komplett	Leichtgewichtig	Leichtgewichtig	Leichtgewichtig	Komplett	Leichtgewichtig
Anforderungen										
Unterstützte Datenbankmanagementsysteme	SQL Server, SQLCE, verschiedene andere via 3rd-Party-Data Provider	SQL Server, Oracle, Access, Firebird, PostgreSQL, DB2 UDB, MySQL, SQLite	SQL Server, SQLCE	SQL Server, SQLCE, MySQL, Firebird, Advantage Database Server, SQL Anywhere, Oracle, VistaDB, SQLIte, PostgreSQL, Maria DB	SQL Server, Oracle, MySQL	Access, DB2, Informix, Fdp, MySQL, ODP, PostgreSQL, SQLCE, SQLite, SQL Server, Sybase Ase	SQL Server, MySQL, PostgreSQL, SQLite, SQLCE, VistaDB, Oracle	SQLIte, SQLCE, Firebird, MySQL, Oracle, SQL Server	SQLServer, Oracle	SQL Server
Minimale .NET-Version	4.0	1.0	3.5	3.5	3.0	2.0	3.5	3.5	3.5	3.5
Unterstützt Mono	Ja	Ja	Teilweise portiert	Nein	Nein	Ja	Nein	Ja	Nein	Nein
Tooling										
Projektvorlagen für Visual Studio	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Ja
Elementvorlagen für Visual Studio	ADO.NET Entity Data Model, EF 5.x DbContext Generator	Nein	LINQ-to-SQL-Klassen	Telerik OpenAccess Class Library, Telerik OpenAccess Dynamic Data Application, Telerik OpenAccess Fluent Library, Telerik OpenAccess Web Application	Nein	Nein	Nein	Nein	Ja	Ja
Extension für Visual Studio	Ja	Ja	Ja	Ja	Nein	Nein	Nein	Nein	Ja	Nein
Designer	Ja	Nein	Ja	Ja	Nein	Nein	Nein	Nein	Ja	Nein
Assistenten	Entity Data Model Wizard	Nein	Nein	Telerik OpenAccess Domain Model Wizard, Telerik OpenAccess Fluent Model Wizard	Nein	Nein	Nein	Nein	Nein	Nein
Weitere Tools	Nein	Nein	Nein	OpenAccess Profiler	Nein	Nein	Nein	Nein	Nein	Nein
Kommandozeilentools	EdmGen.exe	Nein	SQLMetal.exe	Vschema.exe, Venhance.exe	Nein	Nein	Nein	Nein	Nein	Nein
Mapping-Techniken										
Benutzte interne Datenzugriffs-API	ADO.NET	JDBC	ADO.NET	ADO.NET	ADO.NET	ADO.NET	ADO.NET	ADO.NET	ADO.NET	ADO.NET
Verwendete Techniken	Reflection, Codegenerierung, Codegenerierung zur Laufzeit	Reflection, Codegenerierung zur Laufzeit	Reflection, Codegenerierung zur Laufzeit	Enhancing bei Kompilierung, Codegenerierung, Codegenerierung zur Laufzeit	Reflection,Codegenerierung zur Laufzeit	Reflection, Codegenerierung zur Laufzeit	Codegenerierung	Codegenerierung zur Laufzeit	Codegenerierung, Reflection, Codegenierung zur Laufzeit	Codegenerierung
Forward Mapping (Model-First)	Ja	Nein	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Reverse Mapping (Schema-First)	Ja	Ja	Ja	Ja	Ja	Nein	Ja	Nein	Ja	Nein
Datenmigrationen	Ja	Ja	Nein	Ja	Nein	Nein	Ja	Nein	Nein	Nein
Business Objects										
POCO Support	Ja	Ja	Nein	Nein	Nein	Ja	Ja	Ja	Nein	Nein
Erfordern spezifische Basisklasse	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Ja	Ja
Erfordern Annotation	Nein	Nein	Ja	Nein	Nein	Ja	Nein	Nein	Ja	Ja
Erfordern parameterlosen Konstruktor	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Müssen abstrakt sein	Nein	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Nein	Nein
Erzeugte Business Objects unterstützen Datenbindung	INotifyPropertyChanging, INotifyPropertyChanged, IListSource	-	INotifyPropertyChanging, INotifyPropertyChanged	INotifyPropertyChanging, INotifyPropertyChanged	-	-	-	-	-	INotifyPropertyChanging, INotifyPropertyChanged
Erzeugte Business Obejcts sind als [Serializable] gekennzeichnet	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Ja
Erzeugte Business Objects unterstützen WCF Contracts	Ja	Nein	Ja	Ja	Nein	Nein	Nein	Nein	Nein	Nein
Erzeuge Business Objects sind partielle Klassen	Ja	Ja	Ja	Ja	Nein	Nein	Nein	Nein	Ja	Nein
Mapping-Arten										

Persistiert Felder	Ja	Nein	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Persistiert Eigenschaften	Ja	Ja	Nein	Ja	Nein	Ja	Ja	Ja	Nein	Ja
Unterstützung für nullbasierte Werte	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
1 Klasse -> 1 Tabelle	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
1 Klasse -> n Tabellen	Ja	ja	Nein	Ja	Ja	Ja	Nein	Ja	Ja	Ja
n Klassen -> 1 Tabelle	Ja	Ja	Nein	Ja	Nein	Ja	Nein	Ja	Nein	ja
n Klassen -> m Tabellen	Ja	Ja	Nein	Ja	Nein	Ja	Nein	Ja	Nein	ja
Unterstützung für Collection-Typen	EntityCollection	IList, ISet, IDictionary, ICollection	EntitySet	IList, IDictionary, ICollection	IList, IDictionary, ICollection	IList, IDictionary, ICollection	IEnumerable	IList, IDictionary, ICollection	IList, IDictionary, ICollection	IList, IDictionary, ICollection
Codebasiertes Mapping über Annotationen	Ja	optional, via Mapping-Attribute	Ja	Nein	Nein	Nein	Nein	Nein	Nein	Ja
XML-basiertes Mapping/Mapping-Datei	Ja	Ja	Ja	Ja	Ja	Nein	Ja	Nein	Ja	Nein
Mapping zur Kompilierzeit	Nein	Nein	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Nein
Zusammengesetzte Schlüssel	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Ja	Ja	Ja
Referenz über den Schlüssel	Ja	Ja	Nein	Ja	Nein	Nein	Nein	Nein	Ja	Ja
Abfragen										
Containername	ObjectContext	Session	DataContext	ObjectContainer	SqlSession	Db	Db	IDbConnection	Entity	Database
Abfragesprache	LINQ, eSQL	HQL, SQL	LINQ, SQL	LINQ, SQL	SQL	LINQ, SQL	LINQ, SQL	LINQ, SQL	LINQ, SQL	LINQ
Parameterisierung der erzeugten SQL-Abfragen	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Nein
Kompilierte Abfragen	Ja	Ja	Ja	Ja	Nein	Nein	Ja	Ja	Nein	Nein
Lazy Loading	Ja	Ja	ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Eager Loading	Ja	Ja	ja	Ja	Nein	Ja	Ja	Ja	Ja	Ja
Datenaktualisierung										
Speichern von geänderten Daten	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Änderungen bei einzelnen Objekten verfolgen	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Änderungen bei Collections verfolgen	Ja	Ja	Ja	Ja	Nein	Ja	Ja	Ja	Nein	Ja
Unterstützung von Stored Procedures	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Ja
Unterstützung von Transaktionen	ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Ja
Pessimistisches Sperren (bei Transaktionen)	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Ja
Optimistisches Sperren	Ja	Ja	Ja	Ja	Ja	Nein	Ja	Nein	Nein	Ja
Konflikterkennung	Ja	Ja	Ja	Ja	Nein	Nein	Ja	Nein	Nein	Ja
Änderungen bei unverbunden Objekten verfolgen	Ja	Ja	Nein	Ja	Nein	Ja	Ja	Nein	Nein	Ja
Cache										
First Level Cache (Session Level)	Ja	Ja	Ja	Ja	Ja	Nein	Ja	Ja	Ja	Ja
Second Level Cache (Arbeitsspeicher/DB/...)	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Ja
Verteiltes Caching	Nein	NCache (3rd-Party)	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein
Monitoring										
Logging/Tracing	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Ja	Ja
Windows-Leistungsindikatoren	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Nein

3.2 Produkteauswahl

Das Ziel der Produkteauswahl ist es, aus der Marktübersicht drei OR-Mapper zu identifizieren, die genauer betrachtet werden sollen. Dazu wurde eine nicht repräsentative Umfrage unter 30 erfahrenen .NET-Entwicklern durchgeführt, welche Verbreitung, Erfahrungswerte und Beliebtheit bei Entwicklern messen sollte.

3.2.1 Umfrage

Die folgenden Fragen wurden in der Umfrage gestellt:

1. Welche der folgenden .NET OR-Mapper kennst du?
2. Welche davon hast du schon in einem Projekt eingesetzt?
3. Mit welchem Produkt hast du die besten Erfahrungen gemacht?
4. Welches Produkt schätzt du am verbreitesten ein?
5. Für ein privates Projekt, welchen ORM würdest du einsetzen?

Um die Umfrage einzugrenzen wurden nur die folgenden, in der Marktübersicht (siehe Kapitel 3.1 auf Seite 35) betrachteten, Produkte zur Auswahl zugelassen:

- Entity Framework
- OpenAccess ORM
- BLToolkit
- Dapper
- MyBatis.NET
- Linq-2-SQL
- NHibernate
- SubSonic
- Signum Framework

3.2.1.1 Rücklauf

Von den 30 angeschriebenen Entwicklern haben 25 geantwortet.

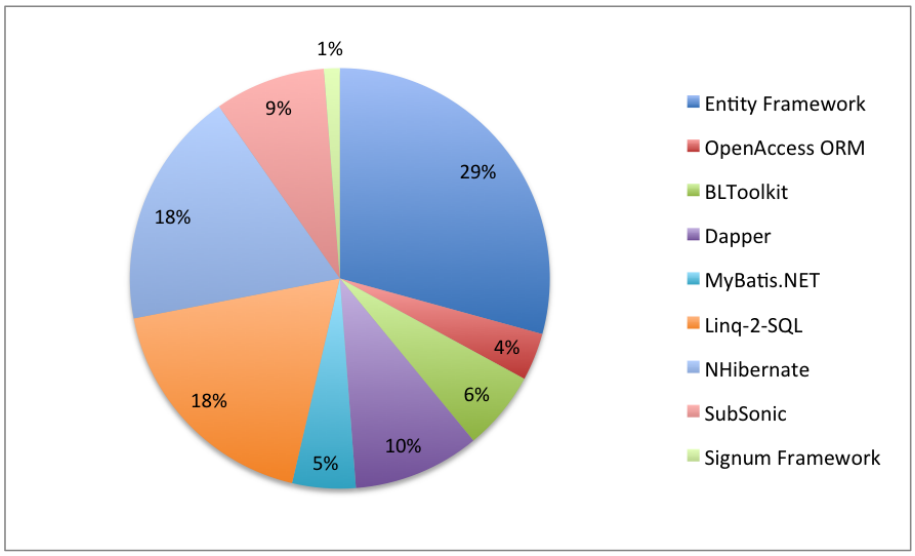

Abbildung 3.1: Umfrageresultate Welche der folgenden .NET OR-Mapper kennst du?

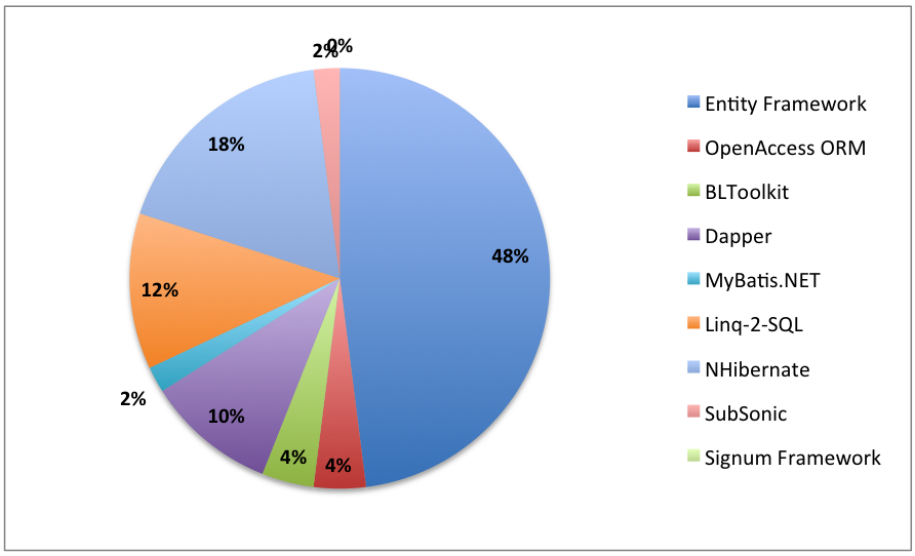

Abbildung 3.2: Umfrageresultate Welche davon hast du schon in einem Projekt eingesetzt?

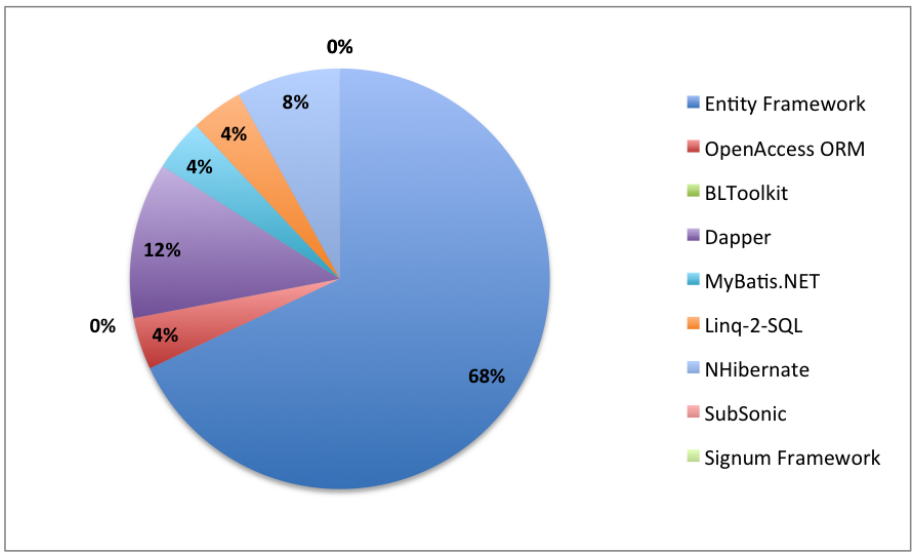

Abbildung 3.3: Umfrageresultate Mit welchem Produkt hast du die besten Erfahrungen gemacht?

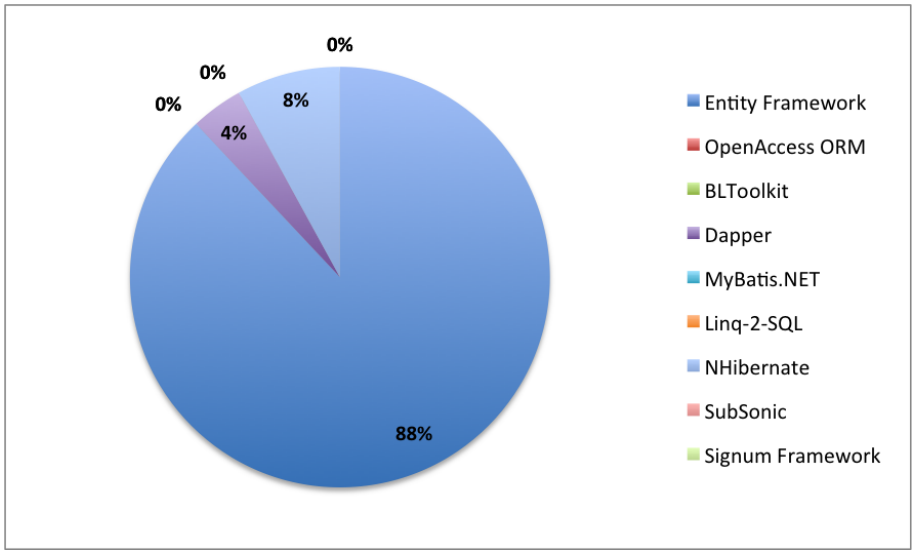

Abbildung 3.4: Umfrageresultate Welches Produkt schätzt du am verbreitesten ein?

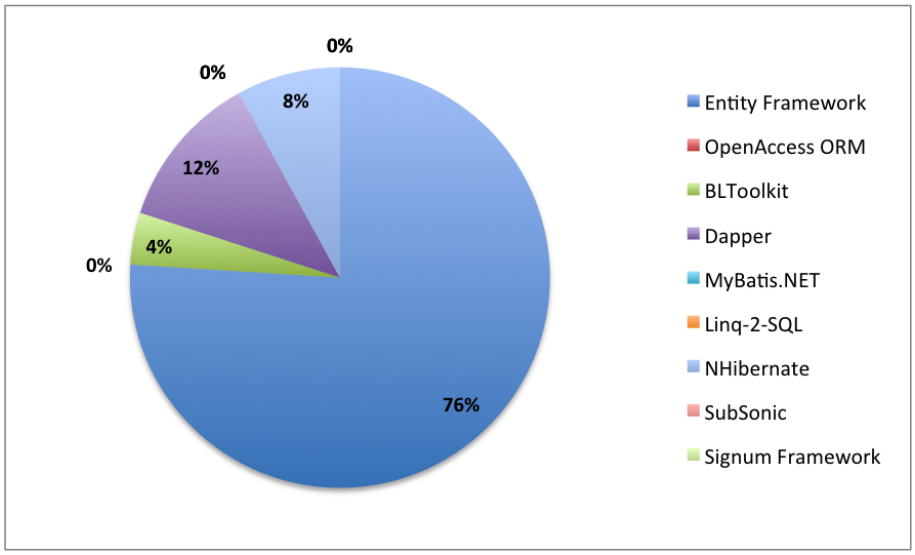

Abbildung 3.5: Umfrageresultate Für ein privates Projekt, welchen ORM würdest du einsetzen?

3.2.1.2 Welche der folgenden .NET OR-Mapper kennst du?

3.2.1.3 Welche davon hast du schon in einem Projekt eingesetzt?

3.2.1.4 Mit welchem Produkt hast du die besten Erfahrungen gemacht?

3.2.1.5 Welches Produkt schätzt du am verbreitesten ein?

3.2.1.6 Für ein privates Projekt, welchen ORM würdest du einsetzen?

3.2.1.7 Resultat

Das Resultat der Umfrage war wenig überraschend. Die weite Verbreitung und den guten Ruf des Entity Frameworks war anzunehmen aufgrund des Marketings von Microsoft. Für die weiteren Schritte werden nun somit folgende drei Produkte genauer betrachtet:

- Entity Framework
- Dapper
- NHibernate

3.3 Vergleich ausgewählter Produkte

Der Vergleich der drei in der Umfage (siehe Kapitel 3.2.1 auf Seite 38) ausgewählten Produkte Entity Framework, Dapper und NHibernate basiert auf den Kriterien Benutzerfreundlichkeit, Plattformunterstützung, Performance, Toolunterstützung, Persistenzlayerpatternsupport, Modelgenerierung, Profiling- und Debuggingmöglichkeiten. Dabei

Kriterium	Entity Framework	Dapper	NHibernate
Fluent Interfaces	Nein	Ja	Nein[2]
Forward Mapping (Database-First)	Ja	Nein	Ja
Reverse Mapping (Model-First)	Ja	Ja	Ja
Migrations	Ja	Nein	Ja
Modeldesigner[3]	Ja	Nein	Nein[4]
Assistentenunterstützung	Ja	Nein	Nein
XML-basiertes Mappping/Mapping-Datei	Nein	Nein	Ja
Codebasiertes Mapping über Annotationen	Ja	Ja	Nein
Monitoring[5]	Ja	Nein	Ja
Unterstützung von Hersteller	Sehr viel	Wenig	Mittel
Inhalt Community-Webseiten (Blogs, Codeproject.com, ...)	Sehr viel	Mittel	Viel
StackOverflow Posts	25000	400	18000

Tabelle 3.1: Kriterien Benutzerfreundlichkeit

wird je nach Kriterium auf Grund einer Tabelle der Gewinner erkoren oder einfach nur auf textueller Basis.

3.3.1 Benutzerfreundlichkeit

Mit Benutzerfreundlichkeit ist der Ease of Use für einen Entwickler gemeint. Die Bewertung der Benutzerfreundlichkeit ist auch stark vom Geschmack des einzelnen Entwicklers ab. Ich persönlich präferiere zum Beispiel Fluent Interfaces da ich dann bereits vom Compiler gewarnt werde, falls ich mich beim Namen einer Eigenschaft verschrieben habe. Andere Entwickler bevorzugen vielleicht XML-basiertes Mapping, da sie die baumartige Struktur von XML präferieren. Wiederum andere Entwickler werden das Vorhandensein eines Mapping-Designers schätzen, da das Mapping abstrahiert wird.

NHibernate hat den grossen Vorteil das viele der für Hibernate zugeschnittenen 3rd-Party-Erweiterungen auch funktionieren und es eine grosse Nutzercommunity gibt. Der klare Sieger dieser Kategorie ist jedoch das Entity Framework. Einerseits weil es Forward- und Reverse-Mapping beherrscht und einen sehr mächtigen, in Visual Studio mitgelieferten, Designer besitzt. Auch ist der Hilfestellungscontent auf der Seite von Microsoft bzw. der Erfahrungsschatz der Community auf Seiten wie StackOverflow ein gewichtiges Argument für die Wahl des Entity Frameworks als OR-Mapper.

3.3.2 Plattformunterstützung

Plattformunterstützung beschreibt die Unterstützung von Entwicklungsframeworks (.NET Framework, Java, usw.) sowie die Unterstützung von Datenbanken.

Entity Framework und Dapper sind seit dem .NET Framework 3.5 verfügbar bzw. benutzen LINQ und Lamda-Expressions die mit dieser Version im .NET Framework eingeführt wurden. NHibernate ist hingegen schon seit dem .NET Framework 1.1 verfüg-

Kriterium	Entity Framework	Dapper	NHibernate
Unterstützt .NET Framework 1.0	Nein	Nein	Nein
Unterstützt .NET Framework 1.1	Nein	Nein	Ja
Unterstützt .NET Framework 2.0	Nein	Nein	Ja
Unterstützt .NET Framework 3.0	Nein	Nein	Ja
Unterstützt .NET Framework 3.5	Ja	Ja	Ja
Unterstützt .NET Framework 4.0	Ja	Ja	Ja
Unterstützt .NET Framework 4.5	Ja	Ja	Ja
Unterstützt Mono	Ja	Ja	Ja
Unterstützt JVM	Nein	Nein	Ja[6]
Unterstützt weitere Entwicklungsframeworks	Nein	Nein	Nein
Unterstützt Microsoft SQL Server	Ja	Ja	Ja
Unterstützt Oracle Database Server	Ja	Ja	Ja
Unterstützt weitere Datenbankmanagementsysteme	Ja[7]	Ja[8]	Ja[9]

Tabelle 3.2: Kriterien Plattformunterstützung

bar, beim betrachten des Sourcecodes[10] fällt jedoch auf dass es deutlich weniger elegant geschrieben wurde als zum Beispiel Dapper. Auch kann die Kernbibliothek von NHibernate nicht direkt mit LINQ von Microsoft umgehen sondern musste einen eigenen LINQ-Provider implementieren.

Da alle drei OR-Mapper auf Basis der DbProviderFactories die Datenbankverbindung aufbauen, sind alle grundsätzlich mit sämtlichen .NET Framework Data Providers kompatibel.

NHibernate hat den grossen Vorteil dass es eine Portierung des sehr populären Hibernates ist, welches für Java entwickelt wurde. Dadurch sind sämtliche Mapping- und Konfigurationsdateien automatisch kompatibel mit dem Java-basierten Hibernate.

Auf Grund der frühen .NET Framework-Kompatibilität und Hibernate als Quelle der Portierung ist NHibernate als Gewinner der Plattformunterstützung hervorzuheben.

3.3.3 Performance

Angelehnt an [12] und [9].

- Query auf eine Tabelle mit Filter auf Primärschlüssel:

  ```
  SELECT ArtistId, Name FROM Artist WHERE Artist.ArtistId = @id
  ```

- Query auf mehrer Tabellen mit Joins, Filter auf Spalte einer Tabelle:

  ```
  SELECT
      Album.Title as AlbumName,
      Track.Name as SongName,
  ```

[10] Siehe `https://github.com/nhibernate/nhibernate-core`.

Kriterium	Entity Framework	Dapper	NHiberr
Performance eines Queries auf eine Tabelle (1 Run)	2.2ms	0.1ms	0.8ms
Performance eines Queries auf eine Tabelle (10 Run)	22.5ms	2.5ms	4.4ms
Performance eines Queries auf eine Tabelle (100 Run)	207.2ms	28.0ms	48.1m
Performance eines Queries auf eine Tabelle (500 Run)	1041.0ms	141.5ms	253.6n
Performance eines Queries auf eine Tabelle (1000 Run)	2099.8ms	285.6ms	509.5n
Performance eines Queries auf mehrere Tabellen (1 Run)	1.0 ms	0.4ms	3.4ms
Performance eines Queries auf mehrere Tabellen (10 Run)	9.9ms	6.4ms	35.6m
Performance eines Queries auf mehrere Tabellen (100 Run)	103.3ms	68.0ms	359.5n
Performance eines Queries auf mehrere Tabellen (500 Run)	525.6ms	346.1ms	1806.2r
Performance eines Queries auf mehrere Tabellen (1000 Run)	1071.0ms	685.5ms	3647.1r
Performance Einfügen, Aktualisieren und Löschen (1 Run)	12.8ms	8.6ms	10.1m
Performance Einfügen, Aktualisieren und Löschen (10 Run)	92.1ms	48.0ms	63.7m
Performance Einfügen, Aktualisieren und Löschen (100 Run)	813.3ms	432.2ms	599.1n
Performance Einfügen, Aktualisieren und Löschen (500 Run)	3765.1ms	2013.8ms	2710.5r
Performance Einfügen, Aktualisieren und Löschen (1000 Run)	6521.0ms	3721.4ms	4987.2r

Abbildung 3.6: Kriterien Performance

```
    Artist.Name as ArtistName
FROM Artist
  INNER JOIN Album ON Album.ArtistId = Artist.ArtistId
  INNER JOIN Track ON Track.AlbumId = Album.AlbumId
WHERE Artist.Name = @name
```

- Einfügen, Aktualisieren und Löschen einer Instanz:

```
INSERT INTO Artist (ArtistId, Name) VALUES (@id, "Daft Prunk")
UPDATE Artist SET Name = "Daft Punk" WHERE ArtistId = @id
DELETE FROM Artist WHERE ArtistId = @id
```

In diesem Vergleich zeigt sich sehr stark der Unterschied zwischen eine kompletten und einem leichtgewichtigen OR-Mapper. Dapper als leichtgewichtiger OR-Mapper hat diesen Vergleich klar gewonnen und sämtliche Metriken dominiert. Dies liegt insbesondere an der einfachen Schichtung des ORMs. Entity Framework als der featurestärkste OR-Mapper hat klar die schlechteste aufgewiesene Performance.

3.3.4 Toolunterstützung

Das Entity Framework bietet mehrfache Toolunterstützung. Wenn ein Model-First oder Database-First Ansatz gewählt wird existiert ein Designer welcher es erlaubt Entitäten mit ihren Attributen und Beziehungen zu modellieren. Ausserdem bietet EF verschiedene Assistenten um ein bestehendes Datenbankschema zu importieren. Wird Code-First das

Domain Model entwickelt so existieren eine benutzerfreundliche NuGet Extension welche dem Entwickler viel Unterstützung bietet.

Dapper als leichtgewichtiger OR-Mapper bietet keinerlei Toolunterstützung an. Sämtliche Konfiguration des ORMs geschieht über in Code geschriebene Mappings und muss vom Entwickler selbst erstellt werden. Die Entitäten können jedoch auch über den Visual Studio Klassendesigner erstellt werden.

NHibernate als herunterladbare Library kommt ohne Tools mit. Die Konfiguration geschieht mittels XML-Dateien und die Entitäten können mit dem Visual Studio Klassendesigner erstellt werden. NHibernate liefert jedoch direkt die XSD Schemas für die XML-Dateien, welche dann auch direkt mit Intellisense verwendet werden können. Es gibt zahlreiche Fremdkomponentenhersteller welche zusätzliche Tools für NHibernate entwickeln, so zum Beispiel der NHibernate Designer von Mindscape HQ[11] oder Visual NHibenrate von Slyce[12]. Diese Tools sind durchaus mächtig und für grössere Projekte anzuraten, werden jedoch in dieser Bachelorarbeit nicht betrachtet da sie meist kostenpflichtig sind.

Der Gewinner in dieser Kategorie ist das Entity Framework. Die mit dem EF mitgelieferten Tools machen es dem Entwickler denkbar einfach Mappings zu erstellen und bieten den gewohnen Komfort von Microsoft Produkten wie dem Visual Studio. Würde man die Drittanbietertools für NHibernate betrachten würde die Entscheidung wohl nicht so klar ausfallen da NHibernate auch sehr populär ist und aus diesem Grund viele Tools von Fremdherstellern existieren.

3.3.5 Unterstützung Enterprise Pattern

Fowlers Enterprise Pattern lassen sich mit jedem OR-Mapper implementieren. Die aktuellen Versionen von Entity Framework, NHibernate und Dapper zwingend den Entwickler nicht dazu ein Pattern über das andere zu favorisieren. Entity Framwork und NHibernate liefern eine einfach Möglichkeit Unit of Work[13] zu implementiern, was jedoch nicht Gegenstand dieser Bachelorarbeit ist. Dapper hingegen ist als Extension für IDBConnection am flexibelsten für jedewedes Enterprise Pattern einzusetzen. Im Endeffekt obliegt es dem Entwickler bzw. Software Architekten wie der Persistenzlayer einer SOA implementiert werden soll und welche Pattenr eingesetzt werden sollen. Die Entscheidung ist zu abhängig vom Anwendungsfall des Systems bzw. den Anforderungen an dasselbe als dass im Rahmen dieses Vergleichs von ORMs eine abschliessende Wertung erstellt werden kann.

Aus diesem Grund ist diese Kategorie nicht bewertbar und es gibt keine Gewinner oder Verlierer.

[11] Siehe `http://www.mindscapehq.com/products/nhdesigner`.
[12] siehe `http://www.slyce.com/VisualNHibernate/`.
[13] Mehr zum Pattern `http://martinfowler.com/eaaCatalog/unitOfWork.html`

3.3.6 Monitoring

Montoring eines Persistenzlayers und des dazugehörigen ORMs wird bei der Evaluation von ORMs oft fälschlicherweise ausser acht gelassen. Dies, da das Montoring häufig erst spät in der Entwicklung eines Systems bezeiehungsweise erst in der Wartung des Systems eine Rolle spielt wenn Performanceengpässe spürbar werden. Wird dann ein ORM eingesetzt der nur sehr eingeschränkt debug- und überwachbar ist wird es schwierig die eigentlichen Bottlenecks zu erkennen und beheben.

Natürlich lassen sich alle ORMs über einen SQL Profiler monitoren. Nur leider ist das meistens schwierig zu sagen woher genau die abgesetzten Queries stammen beziehungsweise ist die Nachverfolgbarkeit der einzelnen Queries zum Mapping im OR-Mapper schwierig.

Das Entity Framework bietet out-of-the-box nur das Intellitrace debugging mit. Da jedoch für das Entity Framework die PDB Symbole auf Microsofts Symbol Server verfügbar sind ist das debugging ohne grosse Hürden wie selbstkompilieren des Entity Frameworks möglich. Da das Entity Framework auf Grund seiner Vielzahl von Funktionalitäten sehr komplex ist, ist es für einen Entwickler nicht ganz einfach ein einfaches Mapping nach zu vollziehen. Es existieren eine grosse Anzahl an Dritterherstellertools wie der Entity Framework Profiler von Hibernating Rhinos[14] oder der EF Provider Wrapper in der Visual Studio Gallery[15] welche bei Performanceengpässen in produktiven Umgebungen zum Einsatz kommen können.

Da Dapper in Sourcecodeform vorliegt ist das Debugging denkbar einfach gestaltet. Der Entwickler kann auch einfach eigene Performancecounters implementieren oder zum Beispiel ein Tracing aller SQL-Statements realisieren da der Quellcode relativ einfach lesbar ist. Der Nachteil ist allerdings, dass dies alles selber entwickelt werden muss. Dapper selbst liefert keinerlei Montioringtools oder ähnliches mit.

NHibernate ist ein Open-Source Projekt und aus diesem Grund kann dieses für Debuggingzwecke selbst kompiliert werden und debugged werden. Auf Grund der Natur von NHibernate (1-zu-1 Portierung von Hibernate in Java) ist die lesbarkeit und Verständlichkeit für einen .NET Entwickler nicht immer geben. Auch ist das Kompilieren von NHibernate mit erheblichem Konfiguraitonsaufwand verbunden nicht ohne weiteres durchführbar. Für NHibernate existiert wie auch für das Entity Framework der NHibernate Profiler von Hibernating Rhinos[16] welcher auf Grund der Session Statistiken zur Verfügung stellt und auch mit dem Sourcecode verknüpft werden kann. NHibernate bietet auch die Möglichkeit eigene Interceptoren[17] zu schreiben. So kann zum Beispiel jedes abgesetzte SQL-Statement mit einem Interceptor abgefangen werden und in ein Tracefile geschrieben werden. Auch lässt sich direkt eine log4net[18] Konfiguration erstellen, welche erweiterte Tracemeldungen von NHibenrate in den gewünschten Kanal persistiert.

[14] Siehe `http://www.hibernatingrhinos.com/products/EFProf`
[15] Mehr dazu unter `http://code.msdn.microsoft.com/EFProviderWrappers`
[16] Siehe `http://www.hibernatingrhinos.com/products/nhprof`.
[17] Mehr zum Interceptor Pattern siehe `http://bosy.dailydev.org/2007/04/interceptor-design-pattern.html`.
[18] Mehr zu log4net siehe `http://logging.apache.org/log4net/`.

Kriterium	Stärkster OR-Mapper
Benutzerfreundlichkeit	Entity Framework
Plattformunterstützung	NHibernate
Performance	Dapper
Toolunterstützung	Enitity Framework
Unterstützung Enterprise Pattern	-
Monitoring	Entity Framework

Tabelle 3.3: Vergleichsmatrix

Abschliessend lässt es sich sagen dass alle drei betrachteten OR-Mapper von Haus aus wenig Montoringmöglichkeiten mitbringen. Dies ist wohl auch so gewünscht, bieten insbesondere NHibernate und das Entity Framework zahlreiche Möglichkeiten für Entwickler oder Drittanbieter Montoringsuiten zu entwickeln und auch verkaufen. Dieser Vergleich bezieht sich somit auf die Möglichkeit zu Monitoring und auch auf die verfügbaren Monitoringtools von Drittherstelltern. Bei dieser Betrachtung ist das Entity Framework der knappe Gewinner vor NHibernate dieses Vergleichs.

3.3.7 Vergleichsmatrix

3.4 Architektur SoCRM

3.4.1 Bausteinsicht

Nach [15] und [11] lassen sich unter dem Begriff "Bausteine" sämtliche Software- oder Implementierungskomponenten zusammenfassen, die letztendlich Abstraktionen von Quellcode darstellen. Dazu gehören Klassen, Prozeduren, Programme, Pakete, Komponenten (nach der UML-Definition) oder Subsysteme.

Die Bausteinsicht bildet die Aufgaben des System auf Software-Bausteine oder -Komponenten ab. Diese Sicht macht Struktur und Zusammenhänge zwischen den Bausteinen der Architektur explizit. Bausteinsichten zeigen die statischen Aspekte des Systemes und entsprechen in dieser Hinsicht den konventionellen Implementierungsmodellen.

3.4.1.1 Komponentendiagramm

Das Komponentendigramm in Abbildung 3.7 stellt das System SoCRM aus der Vogelperspektive dar und ist die höchstabstrahierte Ansicht der Bausteinsicht, die in diesem Projekt verfügbar ist.

3.4.1.2 Domänenmodell

Das Domänenmodell (siehe Abbildung 3.8) umfasst nur die Klassen der Objekte, die über die Serviceschnittstelle von der Client- an die Serverapplikation beziehungsweise umgekehrt übertragen werden.

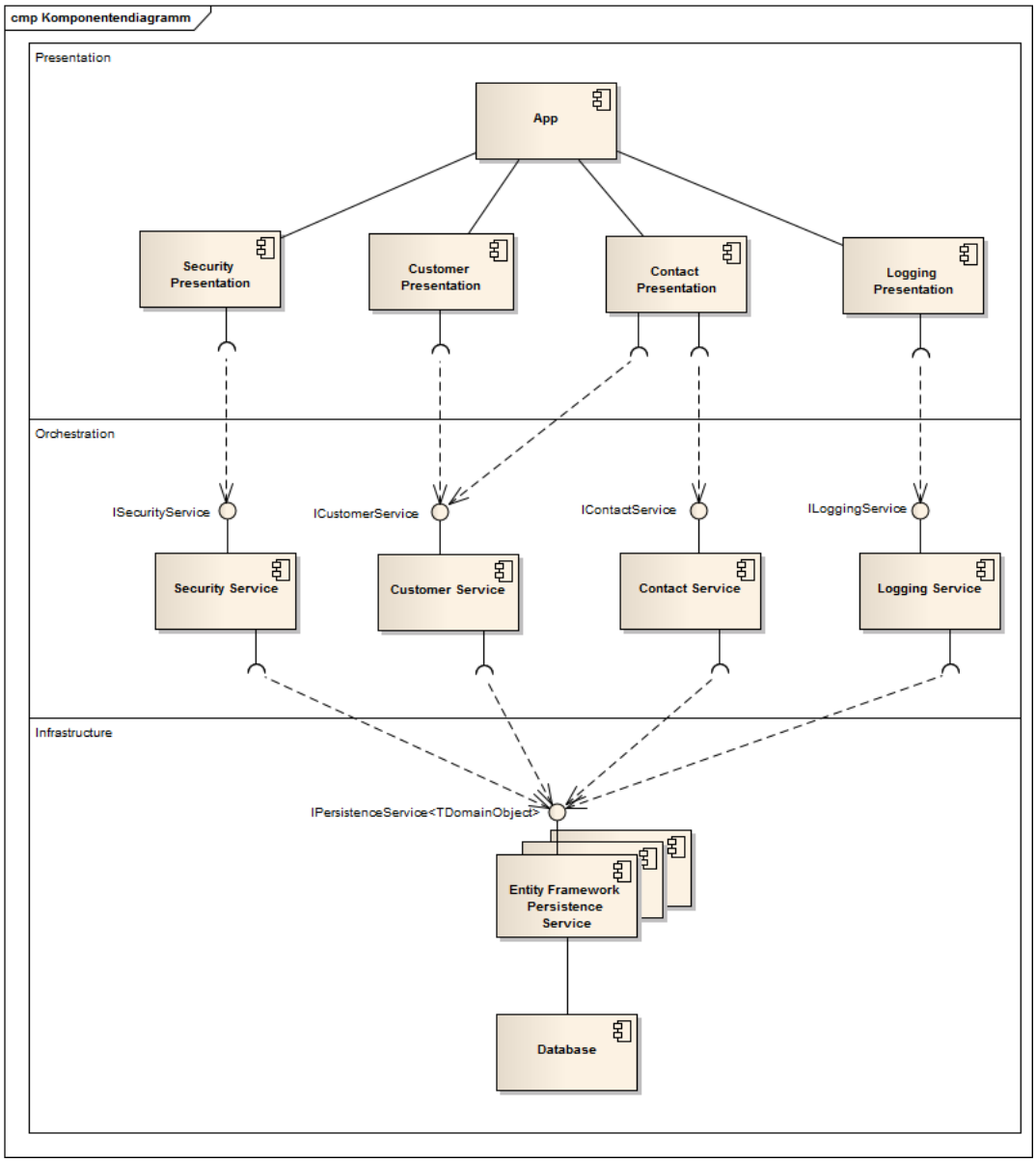

Abbildung 3.7: Komponentendiagramm SoCRM

YAEM verwendet ein vergleichsweise simples Domänenmodell. Jeder Benutzer wird als *User*objekt im Repository der Serverapplikation gespeichert. Bei erfolgreichem Gesprächsbeitritt erstellt die Serverapplikation ein *Ticket*objekt, welches an die Clientapplikation zurückgegeben wird. Wird eine Nachricht versendet, so erstellt der Client ein *Message*objekt, welches die Nachricht selbst als Bytearray in der Eigenschaft *Payload* speichert. Will der Benutzer eine verschlüsselte Nachricht versenden, so setzt er die Eigenschaft *Algorithm* des *Message*objekts auf einen Wert des Enumerators *CryptoAlgorithm*, der ungleich *None* ist. Wenn die Eigenschaft *Algorithm* gesetzt ist, so muss der Benutzer den *Payload* verschlüsselt in der *Message* ablegen.

Sämtliche Domänenobjekte leiten von der Klasse *ObjectBase* ab. *ObjectBase* enthält eine Eigenschaft *Key* in Form einer Globally Unique Identifier zur eindeutigen Identifizierung der Objekte in den Repositories. Weiterhin implementiert *ObjectBase* das Interface *INotifyPropertyChanged*[19], welches in der Windows Presentation Foundation und in Silverlight dazu dient, UI-Elemente, die an Datenobjekte gebunden sind, über geänderte Eigenschaften zu informieren.

3.4.1.3 Service Contracts

Ein Service Contract spezifiziert eine Schnittstelle zur Kommunikation verschiedener Applikationen innerhalb eines verteilten Systems. Häufig werden diese Service Contracts als Webservices angeboten, da sie dadurch plattform- und frameworkunabhängig genutzt werden können.

Typischerweise umfasst ein Service Contract mehrere Operationen, deren Rückgabewerte als XML-Fragmente an die konsumierende Applikation zurückgegeben werden. Ausserdem ist ein Service Contract per Definition grundsätzlich zustandslos, er behandelt mehrere Anfragen (auch desselben Auftraggebers) immer als unabhängige Transaktionen. Anfragen werden ohne Bezug zu früheren, vorhergegeganen Anfragen behandelt und es werden auch keine Sitzungsinformationen ausgetauscht.

ISecurityService

ICustomerService

IContactService

ILoggingService

3.4.2 Laufzeitsicht

Die Laufzeitsicht beschreibt, welche Bestandteile des Systems zur Laufzeit existieren und wie diese zusammenwirken (nach [15]). Dabei kommen wichtige Aspekte des Systembe-

[19] Die genaue Schnittstellenbeschreibung ist in der MSDN unter `http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged.aspx` zu finden.

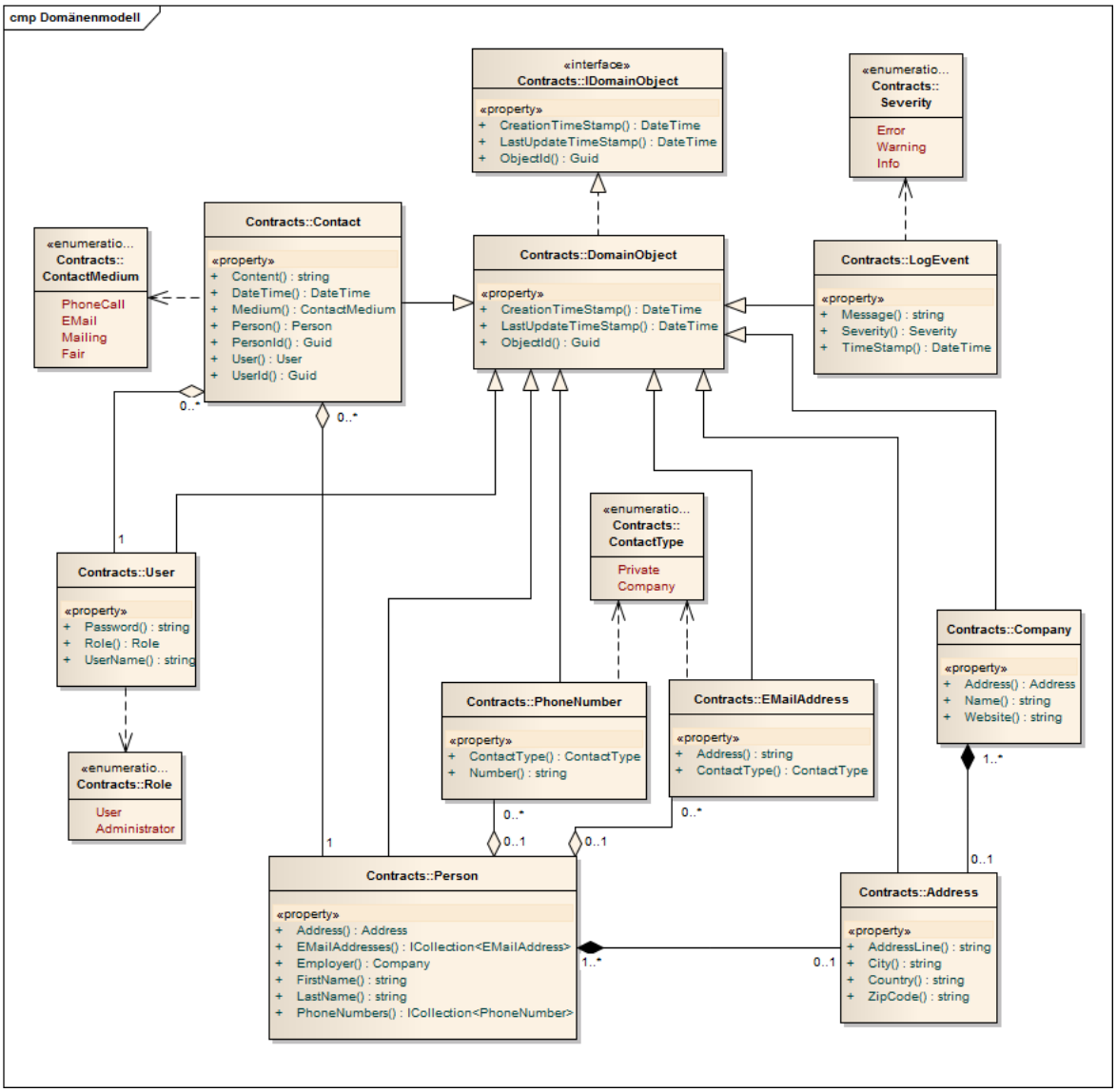

Abbildung 3.8: Klassendiagramm Domänenmodell SoCRM

Methode	Aufgabe
CreateUser	Erstellt einen neuen Benutzer mit den übergebenen Werte
DeleteUser	Löscht einen Benutzer auf Basis der ObjectId
GetAllRoles	Gibt alle Rollen zurück
GetAllUsers	Gibt alle Benutzer zurück
GetUserByCredentials	Gibt einen Benutzer auf Grund seiner Benutzerinformationen z
GetUserByObjectId	Gibt einen Benutzer auf Grund seiner ObjectId zurück
GetUsersByRole	Gibt Benutzer auf Grund ihrer Rolle zurück
GetUsersByRoleAndUserName	Gibt Benutzer auf Grund ihrer Rolle und Benutzernamen zur
SetPassword	Setzt das Password für einen bestimmten Benutzer
ValidateCredentials	Überprüft die übergebenen Benutzerinformationen und liefert wahr ode

Tabelle 3.4: Methoden von ISecurityService

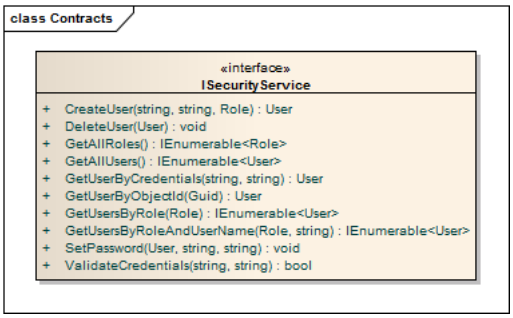

Abbildung 3.9: Klassendiagramm ISecurityService

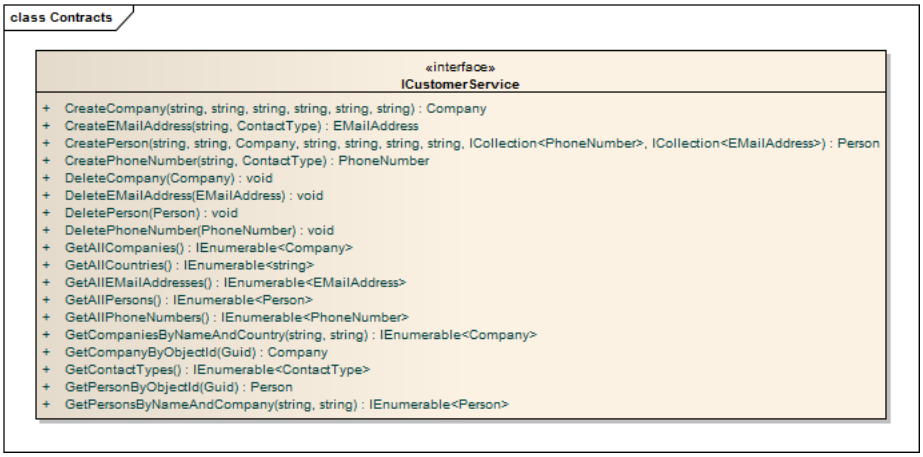

Abbildung 3.10: Klassendiagramm ICustomerService

Methode	Aufgabe
CreateCompany	Erstellt eine neue Firma mit den übergebenen Werten
CreateEMailAddress	Erstellt eine neue E-Mail-Adresse für einen Kunden
CreatePerson	Erstellt einen neuen Kunden mit den übergebenen Werten
CreatePhoneNumber	Erstellt eine neue Telefonnummer für einen Kunden
DeleteCompany	Löscht eine Firma auf Basis ihrer ObjectId
DeleteEMailAddress	Löscht eine E-Mail-Adresse auf Basis ihrer ObjectId
DeletePerson	Löscht einen Kunden auf Basis seiner ObjectId
DeletePhoneNumber	Löscht eine Telefonnummer auf Basis ihrer ObjectId
GetAllCompanies	Gibt alle Firmen zurück
GetAllCountries	Gibt alle Länder zurück
GetAllEMailAddresses	Gibt alle E-Mail-Adressen eines Kunden zurück
GetAllPersons	Gibt alle Kunden zurück
GetAllPhoneNumbers	Gibt alle Telefonnummern eines Kunden zurück
GetCompaniesByNameAndCountry	Gibt alle Firmen auf Grund ihres Namen und des Landes zurück
GetCompanyByObjectId	Gibt eine Firma auf Grund ihrer ObjectId zurück
GetContactTypes	Gibt alle Kontaktarten zurück
GetPersonByObjectId	Gibt einen Kunden auf Grund seiner ObjectId zurück
GetPersonsByNameAndCompany	Gibt Kunden auf Grund ihres Namens und Arbeitgebers zurück

Tabelle 3.5: Methoden von ICustomerService

Methode	Aufgabe
CreateContact	Erstellt einen Kontakt für einen Kunden mit den i
DeleteContact	Löscht einen Kontakt auf Grund seiner
GetAllContactMediums	Gibt alle Kontaktmedien zurüc
GetContactByObjectId	Gibt einen Kontakt auf Grund seiner Obje
GetContactsByCompany	Gibt Kontakte auf Grund des Arbeitgebers des
GetContactsByContactMedium	Gibt Kontakte auf Grund des Kontaktmed
GetContactsByContactMediumAndPersonName	Gibt Kontakte auf Grund des Kontaktmediums und de
GetContactsByPerson	Gibt Kontakte auf Grund des Kunden
GetContactsByUser	Gibt Kontakte auf Grund des Benutzer

Tabelle 3.6: Methoden von IContactService

Methode	Aufgabe
GetAllLogEvents	Gibt alle Logeinträge zurück
GetAllSeverities	Gibt alle Schweregrade zurück
GetLogEventByObjectId	Gibt einen Logeintrag auf Grund seiner ObjectId zurück
GetLogEventsByServerity	Gibt alle Logeinträge auf Grund ihres Schweregrads zurück
LogEvent	Erstellt einen neuen Logeinträge mit den übergebenen Werten

Tabelle 3.7: Methoden von ILoggingService

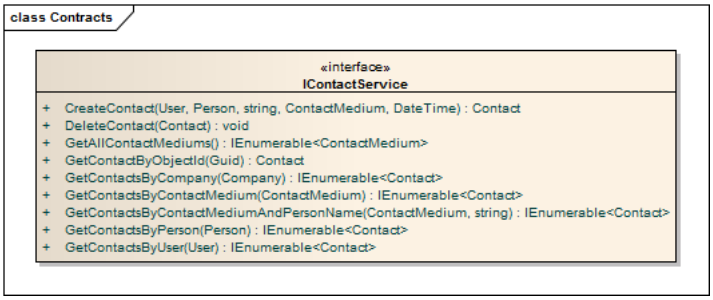

Abbildung 3.11: Klassendiagramm IContactService

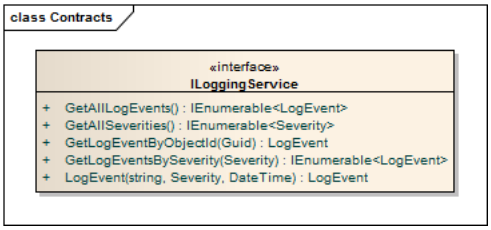

Abbildung 3.12: Klassendiagramm ILoggingService

triebs ins Spiel, die beispielsweise den Systemstart, die Laufzeitkonfiguration oder die Administration des Systems betreffen.

Darüber hinaus dokumentiert die Laufzeitsicht, wie Laufzeitkomponenten sich aus Instanzen von Implementierungsbausteinen zusammensetzen.

3.4.3 Verteilungssicht

Nach [15] beschreibt die Verteilungssicht die Ablaufumgebung des Systems in Form von Hardwarekomponenten (wie Prozessoren, Speicher, Netzwerk, Router und Firewalls) sowie den beteiligten Protokollen. In der Infrastruktursicht können die Leistungsdaten und Parameter der beteiligten Elemente dargestellt werden. Ausserdem werden zusätzlich Betriebssysteme oder externe Systeme aufgenommen.

Die Verteilungssicht ist von grosser Beduetung für die Betreiber des Systems, die Hardwarearchitekten, das Entwicklungsteam sowie Management und Projektleitung (gemäss [?]).

3.4.3.1 Verteilungsdiagramm

Die Verteilungssicht dieser Projektdokumentation enthält nur ein sehr rudimentär ausgearbeitetes Verteilungsdiagramm (siehe Abbildung **??**). Dies, da kein konkretes Verteilungsszenario der Applikation innerhalb des Projekts geplant wurde. Das Projekt beeinhaltet die Erarbeitung des Konzepts sowie die konkrete Implementierung der Applikation ohne jedoch auf die Verteilung des Systems einzugehen.

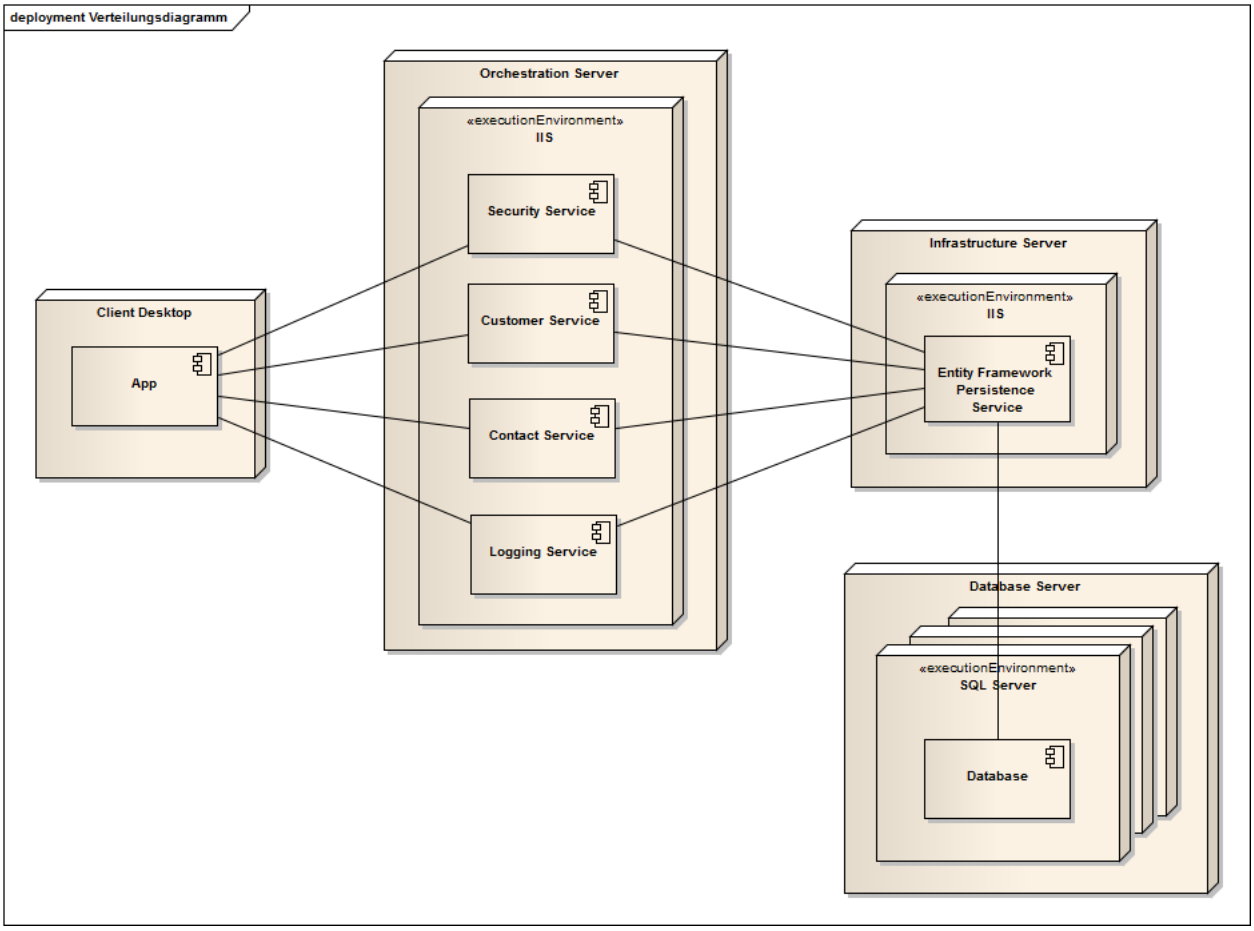

Abbildung 3.13: Verteilungsdiagramm SoCRM

Generell können die einzelnen Clientapplikationen auf jeglicher Hardware laufen, benötigen jedoch jeweils das passende Betriebssystem beziehungsweise im Fall des Silverlight-Clients einen Browser mit Silverlight-Plugin. Zwischen den Clientapplikationen und der Serverapplikation in Form des Servicehosts wird eine Netzwerkverbindung vorausgesetzt bei welcher die passenden Ports (siehe Binding Kapitel **??** auf Seite ??) geöffnet und zugänglich sind. Auch ist eine Verteilung über das WWW denkbar.

4 Implementierung

4.1 Evaluierung selektierte OR-Mapper

4.2 Produkteempfehlung

4.3 Service Oriented CRM

4.3.1 Designprinzipien

- Sämtliche Interfaces und Entitäten sind in eigenen Klassenbibliotheken abgelegt. Dies um die Wiederverwendbarkeit zu erhöhen und die Assemblygrösse zu verkleinern. Die Service Referenzen zwischen den Projekten und den Web Services verwenden immer diese Assemblies anstatt die Entitäten aus dem WSDL des Services zu erzeugen.
- Die Persistence-Services (in SoCrm.Infrastructure.Persistence) besitzen keinerlei Geschäftslogik sondern dienen rein dazu Objekte aus dem jeweils verwendeten ORM aus der Datenbank zu lesen, zu aktualisieren, hinzuzufügen bzw. zu löschen. Sämtliche Geschäftslogik ist in den Services (in SoCrm.Services) implementiert.
- Das Präsentationspattern, welches verwendet wird, ist MVVM+C (nach [14]). MVVM+C stellt eine Weiterentwicklung von MVVM dar und verschiebt die Erzeugung der View und des ViewModels in den Controller. Auch wird die Beziehung zwischen der View und dem ViewModel im Controller erzeugt und nicht mehr über das Binding gelöst.
- Um dynamisch Views innerhalb einer Shell laden zu können sind alle Views als UserControls implementiert, welche in den ControlContainer der ShellView eingefügt werden können.
- Die Serviceimplementierungen (.svc Code-Behind-Dateien) sollen möglichst schlank und entkoppelt sein. Deswegen wird möglichst viel Logik in Basisklassen bzw. andere Orte ausgelagert insbesondere im Zusammenhang mit dem Zugriff auf ORMs. Dadurch wird die Les- und Testbarkeit der Serviceimplementeirungen erhöht.

4.3.2 Komponenten im Detail

Dieses Kapitel beschreibt im Detail die Implementierung der einzelnen Komponenten. Dabei wird auf Besonderheiten in der Architektur der Komponenten hingewiesen und es wird ein gesamter technischer Überblick über die Umsetzung des Konzepts gegeben.

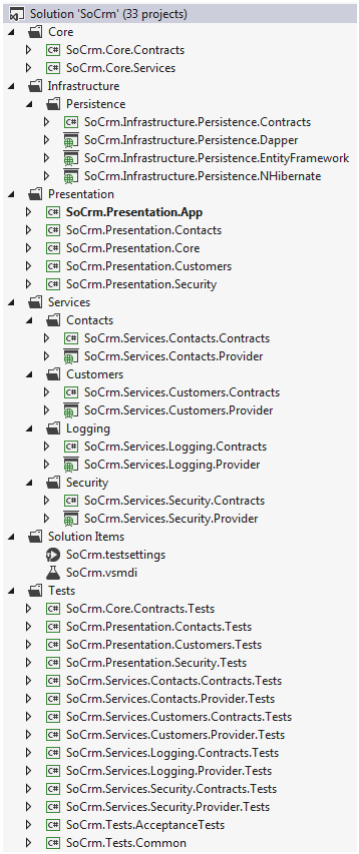

Abbildung 4.1: Visual Studio Solution

Zur besseren Orientierung innerhalb des Systems wird auf Abbildung 4.1 verwiesen, welche die Struktur Visual Studio-Solution darstellt.

Die Namensgebung der Unterkapitel hält sich dabei an die Namespaces der Projekte. Die verwendeten Grafiken zeigen nur die beschriebenen Klassen und Interfaces an, die Projekte enthalten noch weitere Codedateien die jedoch nicht erwähnenswert sind und deswegen in den Klassendiagrammen auch nicht gezeichnet sind. Auch wurden nur in denjenigen Unterkapiteln Klassendiagramme eingefügt wo es für den Leser von Interesse sein könnte.

4.3.2.1 SoCrm.Core.Contracts

Diese Klassenbibliothek enthält die Basisklasse aller Entitäten, das DomainObject. Sie implementiert das Interface IDomainObject welches unter anderem den Identifier, die ObjectId, deklariert.

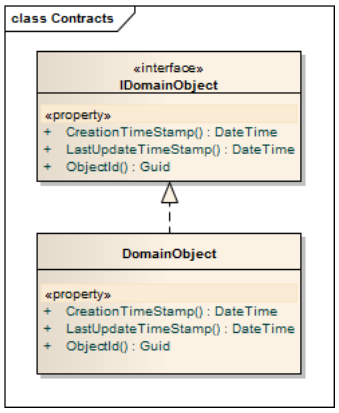

Abbildung 4.2: Klassendiagramm SoCrm.Core.Contracts

4.3.2.2 SoCrm.Core.Services

In der Klassenbibliothek SoCrm.Core.Services befindet sich die Infrastruktur um WCF Services mit dem Dependency Injection Container Unity[1] betreiben zu können. Damit die WCF Services mit Unit Tests versehen werden können wurden die externen Abhängigkeiten mit Dependency Injection entkoppelt. Für das Starten der Services wird jedoch ein spezieller Service-Host, der UnityServiceHost, benötigt, welcher beim Aufstarten ein Behavior mit dem Unity-Container erzeugt und dem Service injected. Der spezifische Service-Host, das Service-Behavior und das Instance-Behavior sind in diesem Projekt zu finden.

4.3.2.3 SoCrm.Infrastructure.Persistence.Contracts

Für jeden Entitätstyp existiert ein Persistenzservice, wessen Interface in dieser Assembly abgelegt ist. Dabei erbt jedes Inteface von einem generischen Interface IPersistenceService<T>. Da WCF jedoch nicht ohne weiteres generische Interfaces unterstützt wurde für jeden Entitätstyp ein eigenes Persistence-Service Inteface erstellt.

4.3.2.4 SoCrm.Infrastructure.Persistence.Dapper

Die Persistenzlayerimplementierung von Dapper in der WCF Service Application SoCrm.Infrastructure.Persistence.Dapper besteht aus einer Basisklasse PersistenceServiceBase<T>, welche als Parameter den Entitätstypen akzeptiert, und den acht *PersistenceServices, welche jeweils einen eigenen WCF Service darstellen. Jeder PersistenceService implementiert das entsprechende PersistenceService Interface aus SoCrm.Infrastructure.Persistence.Contract und leitet von der PersistenceServiceBase ab.

Die PersistenceServiceBase beeinhaltet sämtliche Zugriffslogik auf Dapper um einzelne Entitäten zu Lesen und zu löschen, das Speichern und das Lesen von Objektgraphen ist

[1] Mehr zu Unity und Dependency Injection unter `http://unity.codeplex.com/`

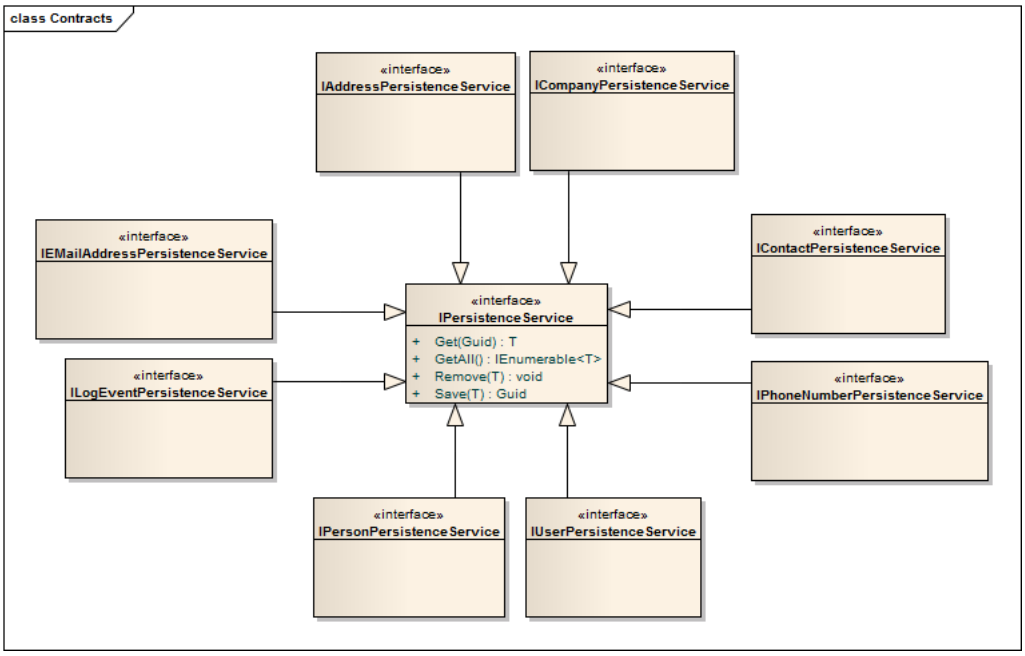

Abbildung 4.3: Klassendiagramm SoCrm.Infrastructure.Persistence.Contracts

im jeweiligen *PersistenceService implementiert da ein manuelles Mapping innerhalb von Dapper dafür notwendig ist.

4.3.2.5 SoCrm.Infrastructure.Persistence.EntityFramework

Die Persistenzlayerimplementierung von EntityFramework als WCF Service Applikation in SoCrm.Infrastructure.Persistence.EntityFramework besitzt wiederum acht *PersistenceServices welche die entsprechendne Service-Inferfaces aus SoCrm.Infrastructure.Persistence.Contracts implementieren. Sämtliche Logik ist dabei in der generischen Basisklasse PersistenceServiceBase abgelegt, die einzelnen *PersistenceServices sind leer und dienen lediglich als Endpoints für die WCF Services.

Die PeristenceServiceBase kümmert sich um das vorbereiten der Entitäten (Validierung, setzen von Änderungsdatum und -Benutzer, usw.) und holt sich aus der DataServiceFactory den pro Entität spezfischen DataService. Der DataService interagiert mit dem jeweiligen ObjectContext des Entity Frameworks.

Aus Gründen der Übersichtlichkeit und ERweiterbarkeit wurden vier ObjectContexte eingeführt, einer für jede Funktionalität des Systems: SecurityContext, LoggingContext, CustomerContext und ContactContext.

Da das EntityFramework 5.0 verwendet wird, wurden die Entitäten Code-First entwickelt. Das EntityFramework bietet für den diesen Anwdungsfalls Migrationen an, welche von der PowerShell-Extension über die NuGet-Konsole erzeugt werden. Diese Migraitonen sind auch in dieser Assembly gespeichert im Namespace SoCrm.Infrastructure.Persistence.EntityFramew

4.3.2.6 SoCrm.Infrastructure.Persistence.NHibernate

Die WCF Service Application SoCrm.Infrastructure.Persistence.NHibernate enthält die Persistenzlayerimplementeriung des ORMs NHibernate. Alle acht *PersistenceServices benutzen die PersistenceServiceBase<T> Klasse als Basisklasse welche den Session-Container von NHibernate wrappt und diesen über den NHibernateHelper SessionFactory erzeugt. Der NHibernateHelper erstellt die Mapping-Konfiguration von NHibernate und lädt alle Mappings der Entitäten.

Die Mappings für alle Entitäten sind im Namespace SoCrm.Infrastructure.Persistence.NHibernate.Mappin abgelegt. Dabei werden FluentNHibernate[2] verwendet um die XML-Konfiguration zu vereinfachen und den aufwand zu verkleinern.

4.3.2.7 SoCrm.Presentation.App

Dieses Projekt liefert eine ausführbare Assembly als Build-Output. Die ShellView beeinhaltet ein Dockpanel, welches ein Menu, eine Toolbar, ein zentrales ContentControl sowie eine StatusBar enthält. Das ShellViewModel besitzt eine Referenz auf den AppController welcher aus dem UnityContainer die Controller der SoCrm.Infrastructure.Presentation.Contacts, SoCrm.Infrastructure.Presentation.Customers oder SoCrm.Infrastructure.Presentation.Security Assembly bezieht und dort auf die gewünschten Seiten navigiert.

Beim Aufstarten der Applikation werden im Bootstrapper die Module der Presentation-Assemblies registriert, welche wiederum die Controller und ViewModels im Unity-Container registrieren.

4.3.2.8 SoCrm.Presentation.Contacts

In der Assembly SoCrm.Infrastructure.Presentation.Contacts sind die Views und ViewModels der Kontaktfunktionalität untergebracht. Wie bei allen Presentation-Assemblies ist der Controller (hier der ContactController) Dreh- und Angelpunkt. Er instanziiert bei Bedarf die ConstactListView und das ContactListViewModel (für das Anzeigen bestehender Kontakte verantwortlich) sowie die CreateContactView und das CreateContactViewModel (enthält die Funktionalität zum Erstellen von neuen Kontakten für einen Kunden).

4.3.2.9 SoCrm.Presentation.Core

Diese Assembly enthält die gesamte Infrastruktur für die WPF Anwendung. Dazu gehören die Basisklassen für ViewModel und Controller als auch der StatusBarService, welcher Nachrichten in der Statusbar der ShellView darstellt. Das RegionModel dient zum Darstellen der aktuellen View im ContentControl der ShellView. Die ViewModelBase implementiert das Interface INotifyPropertyChanged (siehe MSDN `http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged.aspx`) welches das Binding bei geänderten Objektendaten im Modell benachrichtigt.

[2] Siehe `http://www.fluentnhibernate.org/`

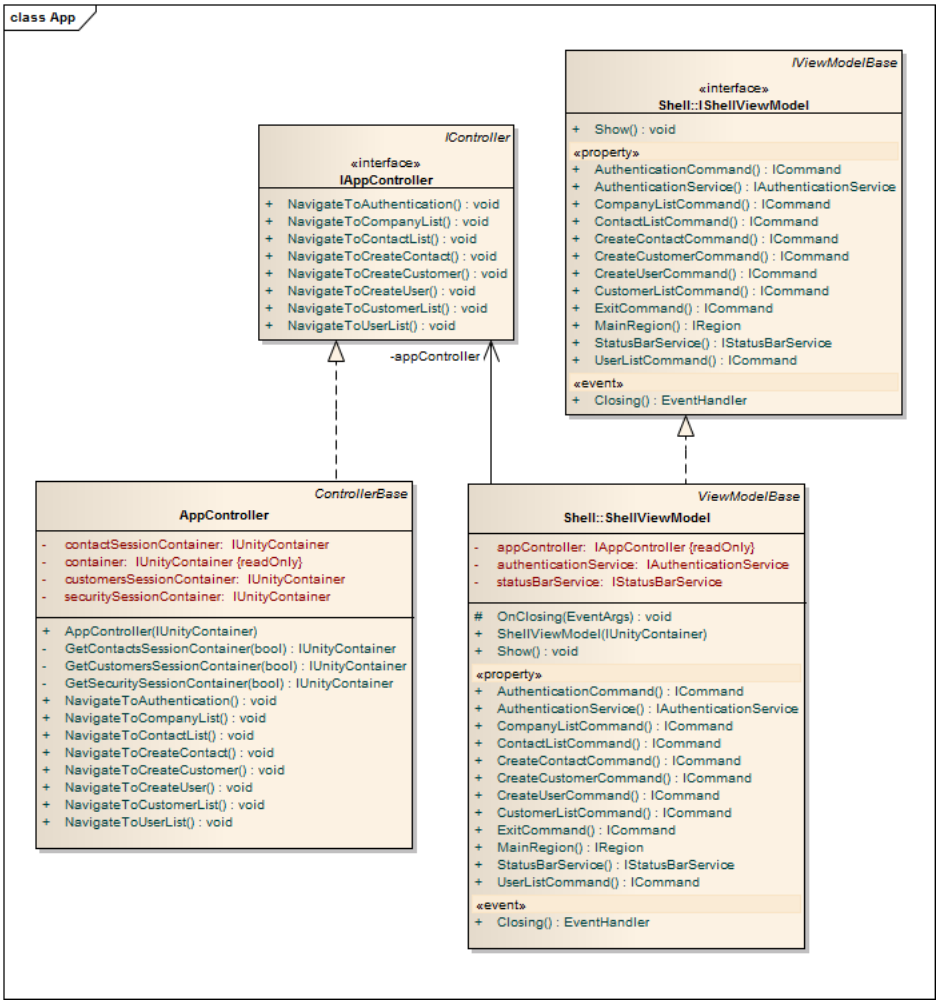

Abbildung 4.4: Klassendiagramm SoCrm.Presentation.App

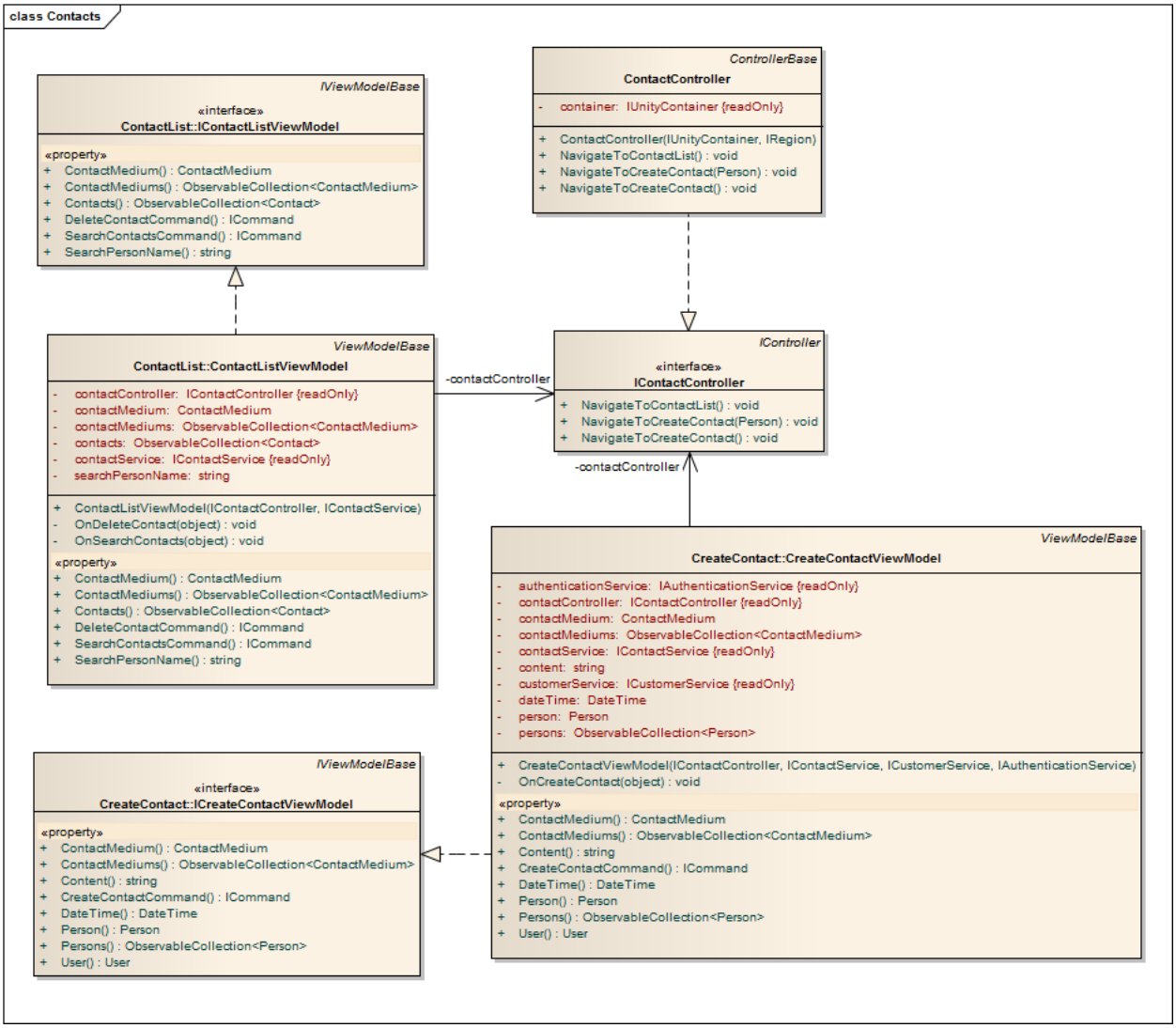

Abbildung 4.5: Klassendiagramm SoCrm.Presentation.Contracts

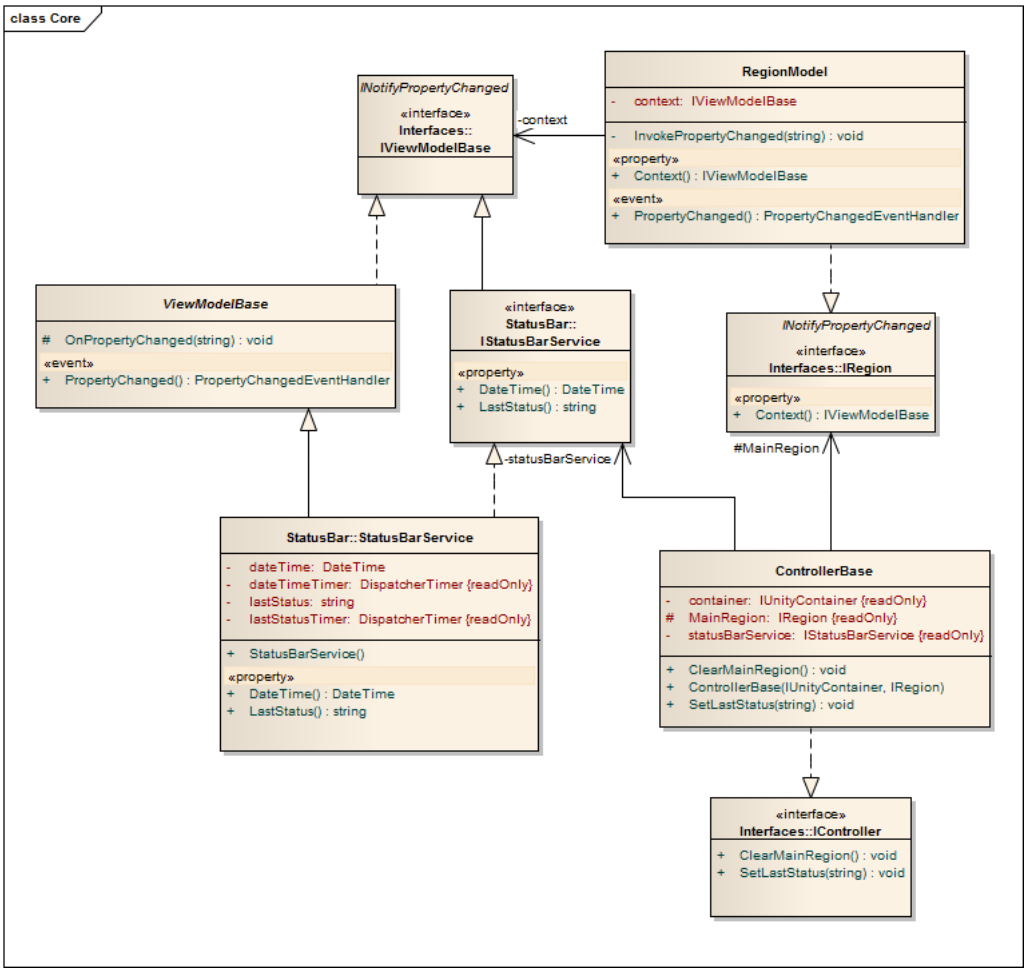

Abbildung 4.6: Klassendiagramm SoCrm.Presentation.Core

4.3.2.10 SoCrm.Presentation.Customers

Die Assembly SoCrm.Infrastructure.Presentation.Customers enthält sämtlich Funktionalitäten für das Anzeigen Hinzufügen und Löschen von Kunden als auch Firmen. Alle Views und ViewModels werden über den CustomerController instanziiert und angezeigt.

4.3.2.11 SoCrm.Presentation.Security

SoCrm.Infrastructure.Presentation.Security enthält die Views und ViewModels für die Benutzerverwaltung sowie das Authentifizieren beim System. Auch ist die SetPassword-View und das SetPasswordViewModel ein Teil dieser Assembly. Das Authentication-ViewModel besitzt eine Referenz auf den AuthenticationService welcher als Singelton im Unity-Container registriert ist und den jeweils angemeldeten Benutzer verwaltet. Der SecurityController verwaltet die Instanziierung der Views und ViewModels.

4.3.2.12 SoCrm.Services.Contacts.Contracts

In diesere Assembly befindet sich die Entität Contact, welche von DomainObject (in SoCrm.Core.Contracts, siehe Kapitel 4.3.2.1 auf Seite 56) erbt und das bereits im oberen Kapitel beschriebene Service-Interface IContactService (siehe Kapitel 3.4.1.3 auf Seite 49).

4.3.2.13 SoCrm.Services.Contacts.Provider

Die WCF Service Applikation SoCrm.Services.Contacts.Provider liefert die Implementierung zum Service-Interface IContactService in SoCrm.Services.Contacts.Contracts. Der ContactService besitzt dabei seine eigene ContactServiceHostFactory die von ServiceHostFactory ableitet und den UnityServiceHost (siehe Kapitel 4.3.2.2 auf Seite 57) instaziiert und einen neuen Unity-Container erstellt. Der ContactService implementiert sämtliche Geschäftslogik betreffend Kontaktdaten und ruft den Persistenzservice für Kontakte (implementiert in einem der SoCrm.Infrastructure.Persistence.* Projekte) auf um die Entitäten aus dem Persistenzlayer zu lesen.

4.3.2.14 SoCrm.Services.Customers.Contracts

Die Klassenbibliothek SoCrm.Services.Customers.Contracts beeinhaltet alle Entitäten den Kunden sowie Firmen betreffend. Ausserdem ist das Interface ICustomerService in diesere Assembly definiert (siehe Kapitel 3.4.1.3 auf Seite 49), welches in SoCrm.Service.Customers.Provider als WCF Service CustomerService implementiert ist.

4.3.2.15 SoCrm.Services.Customers.Provider

SoCrm.Services.Customers.Provider ist eine WCF Service Applikation welche das Interface ICustomerService implementiert und auf die Persistenceservices IAddresPersistence-Service, ICompanyPersistenceService, IEMailAddressPersistenceService, IPersonPersis-

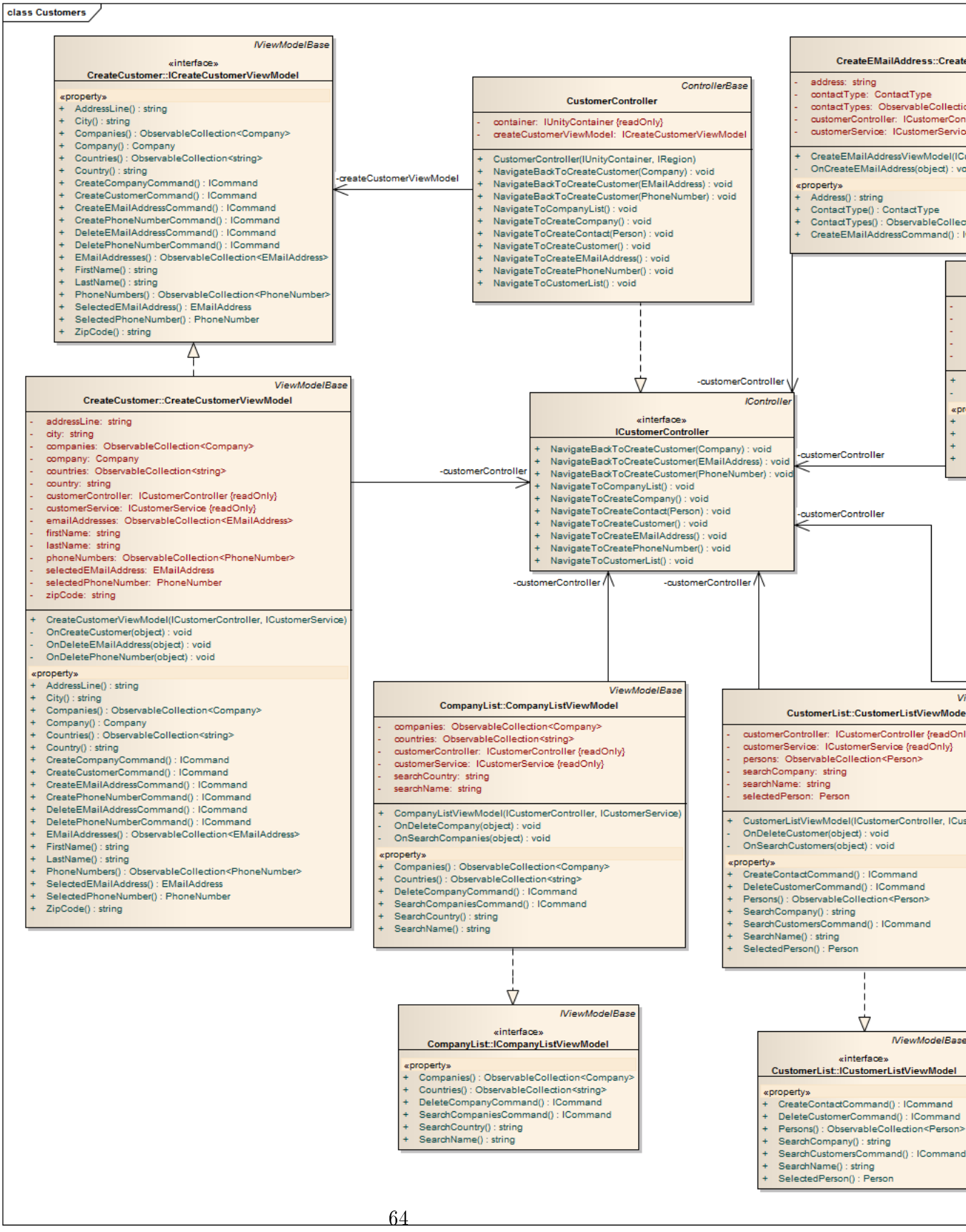

Abbildung 4.7: Klassendiagramm SoCrm.Presentation.Customers

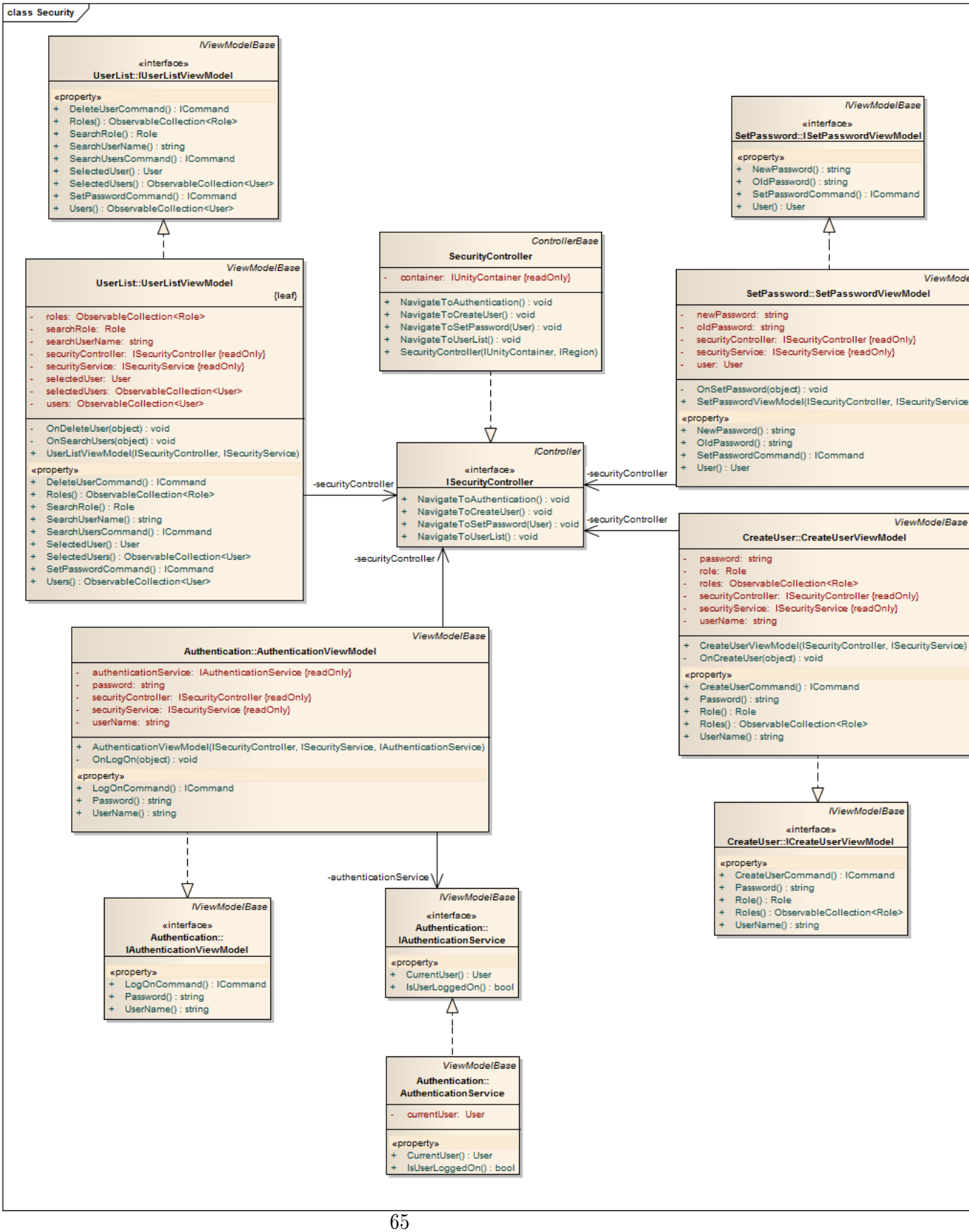

Abbildung 4.8: Klassendiagramm SoCrm.Presentation.Security

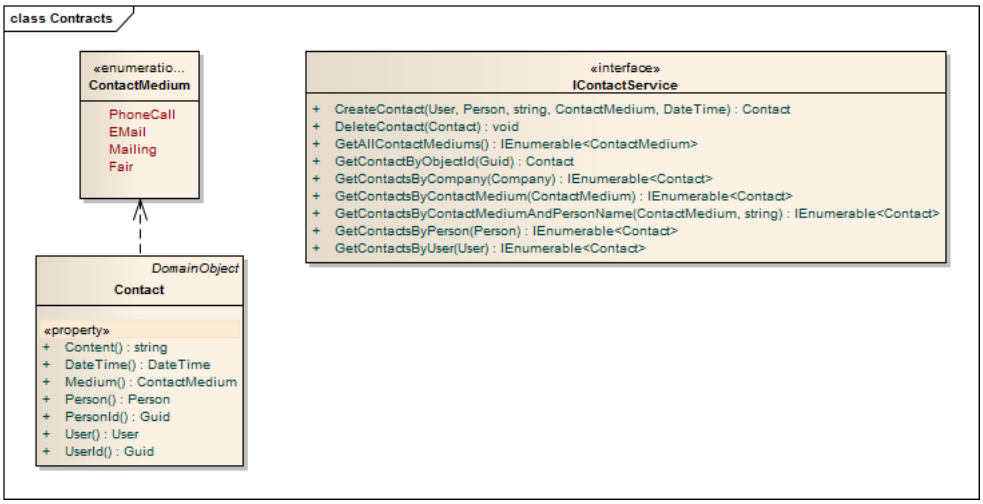

Abbildung 4.9: Klassendiagramm SoCrm.Services.Contacts.Contracts

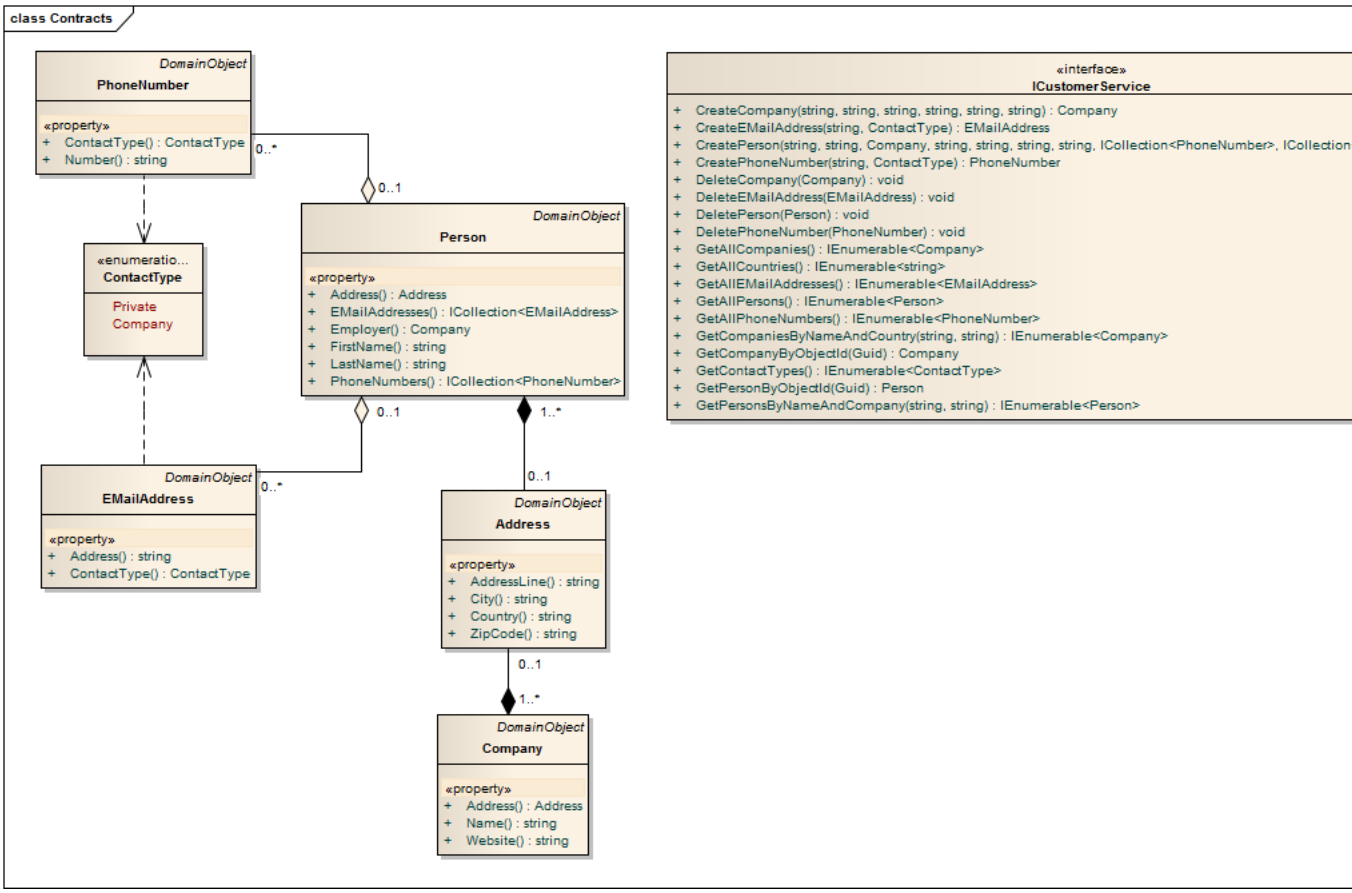

Abbildung 4.10: Klassendiagramm SoCrm.Services.Customers.Contracts

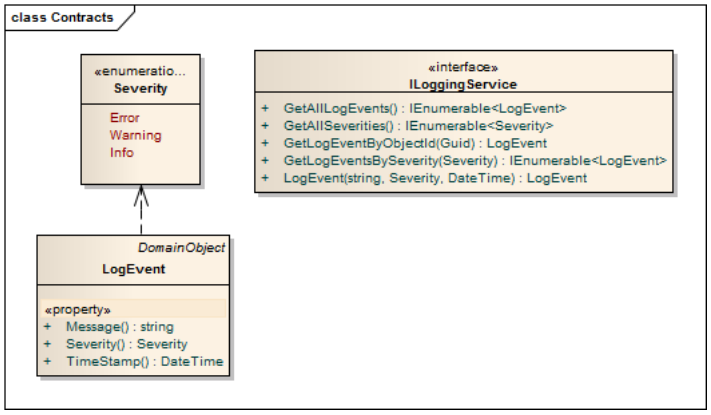

Abbildung 4.11: Klassendiagramm SoCrm.Services.Logging.Contracts

tenceService und IPhoneNumberPersistenceService zugreift. Es beeinhaltet alle Geschäftslogik um mit Personen und Firmen zu interagieren.

4.3.2.16 SoCrm.Services.Logging.Contracts

In der Assembly SoCrm.Services.Logging.Contracts enthält die Entität LogEvent und das Service-Interface ILoggingService (siehe Kapitel 3.4.1.3 auf Seite 49).

4.3.2.17 SoCrm.Services.Logging.Provider

Die WCF Service Application SoCrm.Services.Logging.Provider beeinhaltet den LoggingService welcher das Service-Interface ILoggingService aus dem Namespace SoCrm.Services.Logging.Cont
Die Service Application nutzt dafür den Persistence-Service ILogEventPersistenceService aus dem Persistenzlayer und implementiert die Geschäftslogik für den Umgang mit Logeinträgen.

4.3.2.18 SoCrm.Services.Security.Contracts

Die Assembly SoCrm.Services.Security.Contracts definiert aus dem Service-Interface ISecurityService (siehe Kapitel 3.4.1.3 auf Seite 49) auch die Entität User.

4.3.2.19 SoCrm.Services.Security.Provider

SoCrm.Services.Security.Provider ist eine WCF Service Application, welche den Service SecurityService enthält. Der SecurityService implementiert das Interface ISecurityService aus dem Namespace SoCrm.Services.Security.Contracts und enhält die Geschäftslogik für den Umgang mit Benutzern des Systems SoCRM. Der SecurityService seinerseits erhält seine Objekte aus dem Persistence-Service IUserPersistenceService.

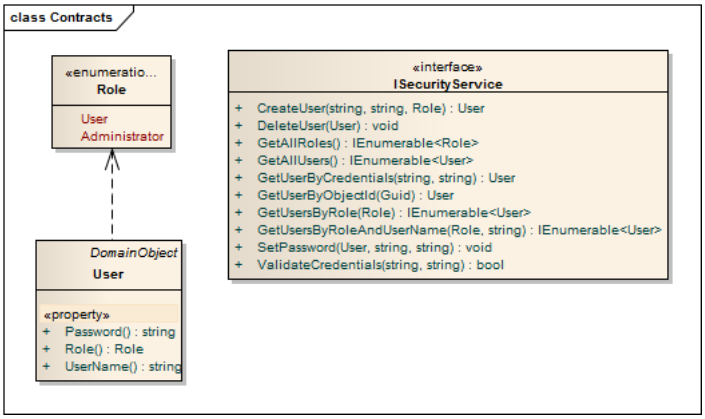

Abbildung 4.12: Klassendiagramm SoCrm.Services.Security.Contracts

5 Verifikation

Die Überprüfung und Verifikation des Softwareprodukts zählt zu den wichtigsten Teilen der Softwareentwicklung. Insbesondere da in den letzten Jahren neue Methoden und Technologien auf den Markt gekommen sind, die dem Softwareentwickler helfen, diese anspruchsvolle Arbeit zu erledigen wird in diesem Kapitel ein Basiswissen auf technischer Ebene vermittelt.

Unit-Tests als automatisierte Tests, die in Quellcode vom Softwareentwickler geschrieben werden, werden als Erstes vorgestellt. Anschliessend wird auf das Thema Akzeptanztest eingegangen, mit Hilfe deren Anforderungen an ein System unter Einbezug der Systemfunktionalitäten selbst getestet werden können.

5.1 Unit-Tests

Unit-Tests (auch Komponententests genannt) überprüfen, ob die von Entwicklern geschriebenen Komponenten so arbeiten, wie diese es beabsichtigen. Zur Qualitätssicherung eines Softwareprodukts wird eine sehr häufige Ausführung der Unit-Tests angestrebt. Das lässt sich nur erreichen, wenn die Tests vollständig automatisiert vorliegen, sie also selbst ein Programm sind, dessen Ausführung nicht mehr Aufwand als einen Knopfdruck erfordert. Insbesondere in der testgetriebenen Entwicklung (siehe Kapitel **??** auf Seite ??) werden Unit-Tests auch als Regressionstests nach Refactoring verwendet.

Durch die testgetriebene Entwicklung von YAEM werden alle implementierten Komponenten schon vorgängig mit Unit-Tests abgedeckt. Die Unit-Tests sind dabei in einem eigenen Visual Studio-Projekt innerhalb der Solution untergebracht und werden nicht zusammen mit dem produktiven Code verteilt.

5.1.1 Testabdeckung

Eine Kenngrösse zur Qualitätssicherung und zur Steigerung der Softwarequalität stellt die Testabdeckung dar. Die Testabdeckung bezeichnet die prozentuale Menge des produktiven Quellcodes, der über automatisierte Unit-Tests abgedeckt ist, im Vergleich zur Gesamtmenge des Codebasis. Insbesondere hift die Testabdeckung bei der Identifizierung von einzelnen Bereichen im Quellcode die potentiell fehleranfällig (da ungetestet beziehungsweise ungenügend getestet) sind.

Die Persistenzlayerimplementierungen SoCrm.Infrastructure.Persistence.Dapper, SoCrm.Infrastructure.Persistence.EntityFramework und SoCrm.Infrastructure.Persistence.NHibernate wurden aus Zeitgründen nicht automatisiert über Unit Tests getestet. Dies da insbesondere in NHibernate und EntityFramwork es sehr aufwendig ist die Session bzw. den

Namespace	Codeblöcke	Getestet (Blöcke)	Getestet (% Blöcl
SoCrm.Core.Contracts	18	18	100%
SoCrm.Core.Services	63	33	52%
SoCrm.Infrastructure.Persistence.Contracts	0	0	100%
SoCrm.Infrastructure.Persistence.Dapper	-	-	-
SoCrm.Infrastructure.Persistence.EntityFramework	-	-	-
SoCrm.Infrastructure.Persistence.NHibernate	-	-	-
SoCrm.Presentation.App	148	114	77%
SoCrm.Presentation.Contacts	168	156	93%
SoCrm.Presentation.Core	162	128	79%
SoCrm.Presentation.Customers	529	497	94%
SoCrm.Presentation.Security	276	252	91%
SoCrm.Services.Contacts.Contracts	42	42	100%
SoCrm.Services.Contacts.Provider	49	42	86%
SoCrm.Services.Customers.Contracts	102	102	100%
SoCrm.Services.Customers.Provider	103	92	92%
SoCrm.Services.Logging.Contracts	18	18	100%
SoCrm.Services.Logging.Provider	28	21	75%
SoCrm.Services.Security.Contracts	18	18	100%
SoCrm.Services.Security.Provider	69	62	85%

Tabelle 5.1: Testabdeckung

ObjectContext zu mocken und der Persistenzlayer implizit in den Akzeptanztest (siehe Kapitel 5.2) getestet wird.

Wie in Tabelle 5.1 ersichtlich ist, beträgt die totale Testabdeckung (ohne den Persistenzlayer) 85%.

5.2 Akzeptanztests

Mithilfe von Akzeptanztest[1] wird geprüft, ob die Software die funktionalen Erwartungen und Anforderungen im Gebrauch erfüllt. Dabei werden Akzeptanztests als Black-Box-Tests gegen die einzelnen Use-Cases der funktionalen Anforderungen (siehe Kapitel **??** auf Seite ??) geprüft, das heisst der Test hat keine Kentnisse über die innere Funktionsweise des Systems und imitiert den Benutzer der Applikation.

In diesem Projekt werden zur Entwicklung der Akzeptanztests Coded UI-Tests verwendet. Coded UI-Tests[2] sind automatisierte Tests die auf der Benutzerfläche festgelegte Aktionen als Skript ausführen lassen. Ein Coded UI-Test kann ausserdem auf einzelnen UI-Elementen Erwartungen definieren (z.B. nach Klick auf den “Senden”-Button muss

[1] auch Abnahmetests oder User Acceptance Tests (UAT)

[2] Mehr zu Coded UI-Tests in der MSDN Library unter `http://msdn.microsoft.com/en-us/library/dd286681.aspx`.

Bezeichner	Use-Case	Testklasse	Testergebnis
UC1	Gespräch beitreten	*UC1Tests*	bestanden
UC2	Gespräch verlassen	*UC2Tests*	bestanden
UC3	Nachricht senden	*UC3Tests*	bestanden
UC4	Nachricht empfangen	*UC4Tests*	bestanden

Tabelle 5.2: Akzeptantests

die Nachricht-Textbox leer sein).

Die Anforderungen in den Use-Cases UC1 bis UC4 aus den Use-Case-Spezifikationen werden als Coded UI-Tests ausformuliert und im Visual Studio-Projekt *YAEM.AcceptanceTests* abgelegt. Beim Ausführen einer der Coded UI-Tests wird automatisch die Serverapplikation sowie die Clientapplikation gestartet und der jeweilige Testfall wird als Skript abgearbeitet. Je nach Use-Case sind unterschiedliche Eingaben sowie Erwartungen definiert. Werden diese nicht erfüllt (z.B. wird nach dem Senden einer Nachricht beim Empfänger keine neue Nachricht angezeigt) so wird schlägt der Test fehl.

Die Ergebnisse der Akzeptanztests sind in Tabelle 5.2 ersichtlich. Alle Anforderungen können erfüllt werden und die Akzeptanztests werden als bestanden betrachtet.

6 Fazit

6.1 Schlussfolgerungen

Akronyme

AES	Advanced Encryption Standard
API	Application Programming Interface
CAPI	Cryptography API
DCOM	Distributed Component Object Model
DES	Data Encryption Standard
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Server
IV	Initialisierungsvektor
MEF	Managed Extensibility Framework
MSDN	Microsoft Developer Network
MSMQ	Microsoft Message Queuing
MVVM	Model View View-Model
NIST	National Institute of Standards and Technology
RUP	Rational Unified Process
SL	Silverlight
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TDD	Test Driven Development
UC	Use Case
UI	User Interface
URI	Uniform Resource Identifier
UX	User Experience

W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WSDL	Web Services Description Language
WPF	Windows Presentation Foundation
WWW	World Wide Web
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XP	Extreme Programming
YAEM	Yet Another Encrypted Messenger
ZHAW	Zürcher Hochschule für Angewandte Wissenschaften

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Markus Klink Guido Zockoll Bernd Oestereich, Claudia Schröder. *EP - oose Engineering Process: Vorgehensleitfaden für agile Softwareprojekte*. dpunkt.verlag, 2006.

[2] Marcel Lattmann Markus Heinisch Michael Könings Mischa Kölliker Perry Pakull Peter Welkenbach Daniel Liebhart, Guido Schmutz. *Architecture Blueprints*. Carl Hanser Verlag, 2007.

[3] Thomas Erl Nitin Gandhi Hanu Kommalapati Brian Loesgen Christoph Schittko Herbjörn Wilhelmsen Mickey Williams David Chou, John deVadoss. *SOA with .NET and Windows Azure*. Prentice Hall, 2010.

[4] Franz-Josef Elmer. Software engineering, methodologien. 2005.

[5] Hugh Taylor Eric Pulier. *Understanding Enterprise SOA*. Manning Publications, 2006.

[6] Vo-Trung Hung. Software development process. 2007.

[7] IEEE. Standard glossary of software engineering terminology. 1990.

[8] Chris Rupp Klaus Pohl. *Basiswissen Requirements Engineering*. dpunkt.verlag, 2011.

[9] Alexis Kochetov. Ormbattle.net. 2011.

[10] Matthew Foemmel Edward Hieatt Robert Mee Randy Stafford Martin Fowler, David Rice. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002.

[11] Gernot Starke Peter Hruschk. arc42 - resourcen für software-architekten. 2013.

[12] Luis Rocha. Data access performance comparison in .net. 2011.

[13] Holger Schwichtenberg. Die qual der wahl. *dotnetpro*, 2008.

[14] Skimp. Mvvm is dead, long live mvvmc! 2012.

[15] Gernot Starke. *Effektive Software-Architekturen*. Carl Hanser Verlag, 2011.

[16] Prof. Dr. Olaf Stern. Reglement bachelorarbeit. 2012.