Vergleich von objektrelationalen Mappern

Florian Amstutz <florian@amstutz.nu>

Bachelorarbeit an der Zürcher Hochschule für Angewandte Wissenschaften

Eidesstattliche Erklärung

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

Zürich, 16. Juni 2013

Florian Amstutz

Inhaltsverzeichnis

1 Einleitung

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 Datenbanken verwenden hingegen Tabellen. Um zwischen diesen grundverschiedenen Paradigmen übersetzen zu können werden objektrelationale Mapper (kurz 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 [Sta11]), denn oft ist der Persistenzlayer das Nadelöhr einer Enterprise Applikation (gemäss [LSL+07]).

Diese Bachelorarbeit erhebt und dokumentiert die Anforderungen an den Persistenzlayer einer serviceorientierten Architektur und ausgewählten Enterprise Pattern (nach [FRF+02]). Es wird eine Marktübersicht aller frei verfügbaren oder Open Source OR-Mapper im .NET-Umfeld erstellt und ausgewählte Produkte werden auf Grund einer Auswahl von Kriterien miteinander verglichen. Die gewählten OR-Mapper werden gegen die erhobenen Anforderungen geprüft und es wird eine Produktempfehlung für jedes Enterprise Pattern erstellt. Benutzt ein Persistenzlayer ein Domain Model, so wird die Verwendung von Entity Framework empfohlen. Für Table Module und Active Record wird der Einsatz von Dapper empfohlen. Wenn die Geschäftslogik eines Persistenzlayers als Transaction Script implementiert ist, wird entweder Entity Framework, NHibernate oder Dapper empfohlen. Diese Produktempfehlungen und die für die Empfehlung verantwortlichen Vergleichsdaten werden anhand eines Proof of Concepts gewonnen. Eine serviceorientierte CRM-Applikation wird exemplarisch implementiert, indem Anforderungen erhoben werden, ein Konzept erstellt wird und nach der Implementierung eine Verifikation der implementierten Anforderungen durchgeführt wird.

Als Fazit erkennt die Arbeit zwei Typen von OR-Mappern: Leichtgewichtige und komplette OR-Mapper. Diese beiden Typen von ORM unterscheiden sich im Funktionsumfang und werden in unterschiedlichen Softwaresystemen eingesetzt.

1.2 Über die Bachelorarbeit

Gemäss Reglement der ZHAW (siehe [Ste12]) dient die Bachelorarbeit zum Nachweis der Fähigkeit ein Projekt auf ingenieurswissenschaftliche Art und Weise zu bearbeiten. Die Bachelorarbeit soll nach den allgemeingültigen Standards für technisch/wissenschaftliches Arbeiten verfasst werden.

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 beziehungsweise der Anwendung des Konzepts.

1.3 Ziele der Arbeit

Das Ziel dieser Arbeit besteht darin die Entscheidung zu vereinfachen, welcher OR-Mapper in welcher Art von Applikation eingesetzt werden soll. Die Entscheidungsgrundlage wird dabei auf Basis von Martin Fowlers Enterprise Pattern (siehe [FRF+02]) gelegt, welche heutzutage in jedem grösseren (Software-) System eine entscheidende Rolle spielen. Es wird pro Enterprise Pattern eine Empfehlung abgegeben, welcher frei verfügbare oder Open Source OR-Mapper im .NET-Umfeld eingesetzt werden soll.

Zusätzlich zu den Empfehlungen wird eine serviceorientierte CRM-Applikation als Proof of Concept erstellt. Für diese CRM-Applikation werden Anforderungen erhoben, es wird ein Konzept erstellt, dieses umgesetzt und die Umsetzung wird verifiziert.

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 [Sta11] ist die Auswahl des Entwicklungsprozesses eine der kritischsten Entscheidungen, die in einem Softwareprojekt getroffen werden 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 einer der Hauptgründe wieso Softwareprojekte mit Qualitätsmängeln, Budgetüberschreitungen oder zeitlichen Verzögerungen zu kämpfen haben.

Für dieses Projekt wird 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 ist. Das Wasserfallmodell wurde ausgewählt, da die Bachelorarbeit als Projekt gut linear abgearbeitet werden kann und der Betreuer und die Studiengangsleitung als einzige externe Stakeholder des Projekts zu festdefinierten Zeitpunkten (namentlich 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 [Elm05] Abgrenzungsprobleme zwischen den einzelnen Phasen sowie die Schwierigkeit des Abschlusses einzelner Phasen da diese nur mit viel (in anderen Entwicklungsmodellen unnötigem) Aufwand isoliert von anderen Phasen abgeschlossen werden können. Dadurch dass der Betreuer und die Studiengangsleitung nur in der Anforderungs- und Konzeptphase Einfluss auf das Projekt nehmen können und der Student alleinig den Abschluss der Phasen steuert und verifiziert sowie den Ablauf der Phasen innerhalb des Projekts koordiniert, können diese Nachteile entschärft werden.

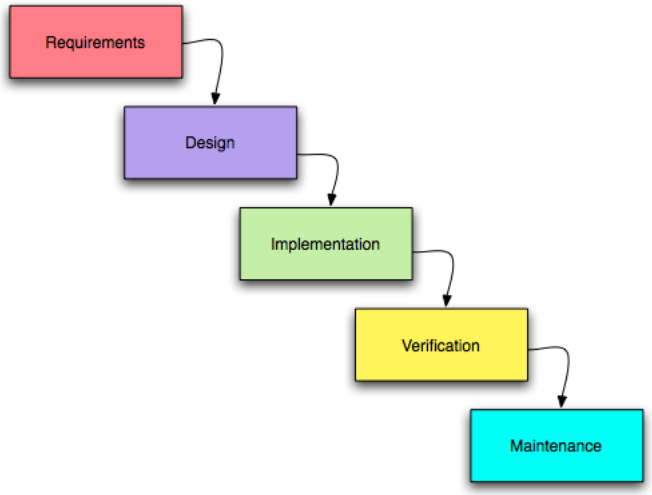

Abbildung 1.1: Wasserfallprozess nach [Hun07]

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 Konzeptphase (Design) über. Das zu realisierende System wird auf verschiedenen Ebenen von Softwarearchitekten entwickelt 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 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 [Hun07] (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, welche in die Wartung übergehen kann.

Phase	Kapitelüberschrift	Ab Seite
Requirements	Anforderungen	10
Design	Konzept und Architektur	40
Implementation	Implementierung	77
Verification	Verifikation	98

Tabelle 1.1: Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument

1.5 Projektplanung

Nach Reglement der ZHAW (siehe [Ste12]) muss die Bachelorarbeit sechs Monate nach Freigabe beendet sein. Um die Planbarkeit der Arbeit zu erhöhen wird das Projekt Bachelorarbeit 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 überhastetes Fertigstellen, gering ist. Ausserdem sinkt die 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 Endergebnissen enden. Bei Erreichen des Endzeitpunkts und bei Lieferung aller Endergebnisse geht das Projekt in die nächste Phase über. Die Phasen dieser Bachelorarbeit werden 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 geeigneten Themen für die Bachelorarbeit evaluiert, ein Betreuer gesucht sowie ein Projektantrag in EBS erfasst. Diese Phase endet mit dem Kick-Off-Meeting zwischen Betreuer, Studiengangsleitung und Student sowie

der formalen Freigabe der Bachelorarbeit durch die Studiengangsleitung. Das Erheben und Dokumentieren der Anforderungen ist die erste Phase der eigentlichen Bachelorarbeit und mündet in der Erarbeitung eines Konzepts auf Basis dieser Anforderungen. Ist das Konzept vollständig abgeschlossen, findet das Design-Review statt, bei welchem der Betreuer und die Studiengangsleitung das Konzept begutachtet und allfällige Anpassungen daran vorschlägt. Nach Fertigstellung des Konzepts folgt die Implementierungs- und Testphase nach denen die Arbeit abgegeben wird. In der letzten Phase des Projekts wird die Präsentation erarbeitet, die an der Schlusspräsentation (Meilenstein M5) vorgetragen wird.

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 dreier 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	17.06.2013	M4
Erarbeiten der Präsentation	18.06.2013	01.07.2013	M5

Tabelle 1.2: Phasenplan

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 8) wurde festgelegt, dass in der ersten Projektphase die Anforderungen erhoben und dokumentiert 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 von Fowlers Enterprise Pattern erhoben. In einem letzten Teil wird der Systemkontext der CRM-Applikation beschrieben und eingegrenzt sowie die konkreten Anforderungen an die CRM-Applikation als Use-Cases modelliert und spezifiziert.

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 [IEE90] 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 oder 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 Bachelorarbeit 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 [PR11] 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.

2.2 Anforderungen Persistenzlayer

Dieses Kapitel enthält die Anforderungen an den Persistenzlayer einer serviceorientierten Architektur. Diese Anforderungen sind gleichzeitig auch Teil der Anforderungen der CRM-Applikation (siehe Kapitel 2.4 auf Seite 18).

Anforderungen werden üblicherweise aus Aktorensicht definiert und sind nicht auf einen Layer einer Softwarearchitektur spezifiziert. Damit jedoch später eine Vergleichsmöglichkeit verschiedener OR-Mapper gegeben werden kann, müssen die Anforderungen im Rahmen dieser Bachelorarbeit auf den Persistenzlayer einer serviceorientierten Softwarearchitektur spezifiziert sein. Anstelle einer modellbasierten Beschreibung der Anforderungen werden diese natürlichsprachig dokumentiert. Dazu wird die Satzschablone nach [PR11] verwendet und die Anforderungen werden in funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen aufgeteilt.

Die Anforderungen stammen aus eigenen Erfahrungen mit serviceorientierten Architekturen sowie [LSL+07], [CdE+10] und [PT06].

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" steht dabei für den Persistenzlayer als eigenständiges System, auf welches über einen Webservice zugegriffen werden kann.

[1] Auch Rahmenbedingungen genannt.

Bezeichner	Anforderung
PF1	Das System muss CRUD-Operationen pro persistierbarer Entität zur Verfügung stellen.
PF2	Das System muss eine einzelne Instanz einer Entität auf Basis seines Identifikators zurückgeben können.
PF3	Das System muss alle Instanzen einer Entität zurückgeben können.
PF4	Das System muss eine oder mehrere Instanzen einer Entität auf Basis von Suchkriterien zurückgeben können.
PF5	Das System muss eine einzelne Instanz persistieren können.
PF6	Das System muss mehrere Instanzen gleichzeitig persistieren können.
PF7	Das System muss eine einzelne Instanz löschen können.
PF8	Das System muss mehrere Instanzen gleichzeitig löschen können.
PF9	Das System muss dem Administrator die Möglichkeit bieten den Ablauf sämtlicher Operationen nachzuverfolgen.

Tabelle 2.1: Funktionale Anforderungen an den Persistenzlayer

2.2.2 Qualitätsanforderungen

Die Qualitätsanforderungen an den Persistenzlayer einer serviceorientierten Architektur beschreiben Anforderungen an die Umstände, unter welchen die geforderte Funktionalität zu erbringen ist. Sie sind in Tabelle 2.2 festgehalten.

Die Grenze zwischen funktionalen und nicht-funktionalen Anforderungen[2] ist unscharf. Qualitätsanforderungen beschreiben die Art und Weise wie der Persistenzlayer zu implementieren ist und geben Bedingungen an die Implementierung vor.

Bezeichner	Anforderung
PQ1	Das System muss fähig sein für sämtliche Operationen ACID zu garantieren.
PQ2	Das System muss die Datenbankmanagementsysteme Microsoft SQL Server und Oracle Database unterstützen.
PQ3	Das System muss dem Entwickler die Möglichkeit bieten die externe Schnittstelle des Systems automatisiert zu testen.
PQ4	Das System soll keine Logik auf die Datenbank auslagern.
PQ5	Das System muss bestimmte Performancekennzahlen erfüllen[3].

Tabelle 2.2: Qualitätsanforderungen an den Persistenzlayer

[2] Wie Qualitätsanforderungen auch nicht-funktionale Anforderungen genannt.

[3] Im Rahmen dieser Arbeit werden die Performanceanforderungen nicht genauer definiert, da insbesondere das Überprüfen dieser den Rahmen dieser Bachelorarbeit sprengen würde.

2.2.3 Randbedingungen

Der Persistenzlayer einer serviceorientierten Softwarearchitektur muss gewisse Randbedingungen erfüllen, welche in Tabelle 2.3 genauer spezifiziert sind. Diese Randbedingungen sind Umstände, die zwingend erfüllt werden müssen. Wiederum ist die Abgrenzung von den Qualitätsanforderungen schwierig, da der Übergang zwischen den zwei Anforderungsarten fliessend ist.

Diese Randbedingungen sind Implementierungsvorgaben welche massgeblich dafür verantwortlich sind, während der Erarbeitung und der Umsetzung des Konzepts einen austauschbaren Persistenzlayer zu erhalten, wie dies in der Aufgabenstellung der Bachelorarbeit (siehe [Ams13a]) gefordert ist.

Bezeichner	Anforderung
PR1	Das System muss alle Operationen als XML-basierten Webservice anbieten.
PR2	Das System muss dem Entwickler die Möglichkeit bieten ohne Wissen über die Datenbank oder datenbanknahe Technologien auf die Operationen zuzugreifen.
PR3	Das System muss dem Entwickler für sämtliche Operationen ein Resultat entweder in Form der Entität oder den Entitäten oder in Form eines Operationsresultats zurückliefern.
PR4	Das System muss dem Entwickler die Möglichkeit bieten durch ein anderes System (einen anderen Persistenzlayer) austauschbar zu sein.

Tabelle 2.3: Randbedingungen an den Persistenzlayer

2.3 Anforderungen Enterprise Pattern

Dieses Kapitel beinhaltet die Anforderungen an Martin Fowlers Enterprise Pattern Transaction Script, Domain Model, Table Module und Active Record. Die Anforderungen stammen sowohl aus [FRF+02] als auch aus [Sta11], [PT06], [CdE+10] und eigenen Erfahrungen im Design und der Entwicklung von serviceorientierten Architekturen. Anstatt wie in den Arten von Anforderungen (siehe Kapitel 2.1.1 auf Seite 10) beschrieben, werden die Anforderungen an Enterprise Pattern rein auf funktionaler Ebene erhoben.

Neben den konkreten Anforderungen werden die einzelnen Enterprise Pattern auch noch in Prosaform beschrieben und deren Anwendung im Persistenzlayer einer serviceorientierten Architektur erläutert. Es hat sich als schwierig herausgestellt aus einem Enterprise Pattern konkrete Anforderungen abzuleiten. Aus diesem Grund sind die Anforderungen in diesem Kapitel nur rudimentär beschrieben. Die spätere Evaluation der OR-Mapper auf Basis dieser Anforderungen (siehe Kapitel 4.1 auf Seite 77) berücksichtigt daher auch die textuelle Beschreibung der einzelnen Enterprise Pattern in den folgenden Unterkapiteln.

2.3.1 Transaction Script

Nach [FRF+02] organisiert das Transaction Script Pattern die Geschäftslogik nach Prozeduren, wobei jede Prozedur einen einzelnen Aufruf der Benutzeroberfläche behandelt. Die meisten Geschäftsapplikationen können als eine Reihe von Transaktionen betrachtet werden. Eine Transaktion kann das Betrachten von Information repräsentieren, eine andere Transaktion bearbeitet diese Information. Das Transaction Script organisiert die Domänenlogik primär nach Transaktionen die der Benutzer auf dem System vornimmt. Jede vom Benutzer initiierte Transaktion besitzt ihr eigenes Transaction Script, auch wenn dieses in einzelne Subprozeduren aufgebrochen werden kann.

Ein Vorteil des Transaction Scripts besteht darin, dass alle Transaktionen atomar ausgeführt werden. Der Entwickler muss sich nicht um konkurrierende Transaktionen kümmern, da diese jeweils im Kontext eines anderen Transaction Scripts ausgeführt werden. Je komplizierter die Geschäftslogik eines Systems jedoch wird, je schwieriger wird es diese in einem funktionalen und verständlichen Zustand zu bewahren. Das grösste Problem besteht dabei in der Vermeidung von Codeduplizierung zwischen Transaktionen. Da das Ziel des Transaction Script Patterns es ist, allen Code einer Transaktion an einer einzelnen Stelle zusammenzufassen, bedeutet es, dass sämtlicher gemeinsamer Code potentiell dupliziert wird, wenn während der Entwicklung nicht besonderes Augenmerk auf diese Tatsache gelegt wird. In grösseren Projekten mit vielen Entwicklern stellt sich dies in der Praxis als nahezu unmöglich dar.

Das Transaction Script ist meist mit dem Command Pattern aufgebaut[4]. Im Command Pattern kapselt das Command-Objekt einen Befehl, um es zu ermöglichen, Operationen z.B. in eine Warteschlange zu stellen, Logeinträge zu führen oder auch Operationen rückgängig zu machen.

Für einfache Geschäftslogik ist das Transaction Script die erste und offensichtliche Wahl. Wenn allerdings komplexere Geschäftsdomänen im System abgebildet werden, so macht es Sinn das Transaction Script in ein Domain Model umzubauen (siehe 2.3.2 auf der nächsten Seite).

Tabelle 2.4 enthält die Anforderungen an den Persistenzlayer einer serviceorientierten Architektur bei Verwendung von Transaction Scripts für die Organisation der Geschäftslogik.

Bezeichner	Anforderung
T1	Das System muss alle Datenbankoperationen durch einen dünnen Datenbankwrapper direkt an die Datenbank stellen.
T2	Das System muss alle Operationen atomar ausführen können.

Tabelle 2.4: Anforderungen Transaction Script

[4] Mehr zum Command Pattern siehe `http://www.codeproject.com/Articles/15207/Design-Patterns-Command-Pattern`.

2.3.2 Domain Model

Das Domain Model ist ein Objektmodell einer Domäne, welches Verhalten und Information beinhaltet (nach [FRF+02]). Die Geschäftslogik kann in einem grossen System sehr komplex sein. Ein Domain Model erstellt ein Netzwerk von miteinander verbundenen Objekten, wobei jedes Objekt für sich selbst eine sinnvolle Einheit ist, egal ob das Objekt eine ganze Bestellung repräsentiert oder nur eine Zeile in einem Bestellformular.

Wird ein Domain Model in eine Applikation integriert, so fügt man einen komplett neuen Layer von Objekten ein, welche das Geschäftsfeld abbilden, in welchem das System sich bewegt. Die Objekte in einem Domain Model imitieren Daten im Unternehmen und beinhalten die Regeln, nach welchen dieses Unternehmen beziehungsweise seine Prozesse funktionieren.

Ein objektorientiertes Domain Model ist häufig dem Datenbankmodell ähnlich, wenn es auch einige Unterschiede geben wird. Das Domain Model vermischt Daten und Prozesse, hat mehrwertige Attribute und ein komplexes Netzwerk von Assoziationen. Es benutzt auch viele Mechanismen aus der objektorientierten Entwicklung wie zum Beispiel die Vererbung.

Laut [FRF+02] gibt es zwei Stile von Domain Models die sich etabliert haben. Das simple Domain Model hat grosse Ähnlichkeiten mit dem Datenbankmodell, wobei meist ein Domänenobjekt für eine Datenbanktabelle existiert. Das reiche Domain Model kann stark vom Datenbankmodell abweichen. Es benutzt eine Vielzahl von Design Patterns (siehe [GHJV94]) und besitzt komplexe Netzwerke von kleinen, miteinander verbundenen Objekten. Ein reiches Domain Model ist besser geeignet komplexe Geschäftslogik abzubilden aber es ist dafür schwieriger auf die Datenbank zu projizieren. Es benötigt in den meisten Fällen einen Data Mapper[5], der oft in einem OR-Mapper implementiert ist.

Das Domain Model ist stark verbreitet, da heute viele Entwickler lieber mit Domänenobjekten arbeiten als mit SQL Daten direkt in der Datenbank zu manipulieren. Es ist sicher nicht der einfachste Weg Domänenobjekte zu modellieren und braucht viel Erfahrung aber die vereinfachte Testbarkeit und Benutzbarkeit macht den Nachteil des Modellierens wett. Gewöhnt man sich einmal an die Vorteile eines Domain Models so ist es schwierig wieder zurück zu einfacheren Datenbankzugriffsmechanismen zu wechseln, wie zum Beispiel dem Transaction Script.

In Tabelle 2.5 sind die Anforderungen an den Persistenzlayer einer serviceorientierten Architektur bei Verwendung eines Domain Models aufgeführt.

[5] Der Data Mapper ist auch ein Enterprise Pattern. Mehr dazu unter `http://martinfowler.com/eaaCatalog/dataMapper.html`.

Bezeichner	Anforderung
D1	Das System muss eine minimale Kopplung des Domain Models an andere Layer ermöglichen.
D2	Das System muss die Möglichkeiten einer Spezifikation bieten, mit welcher der Teil eines zu ladenden Objektgraphen beschrieben werden kann.
D3	Das System muss den Entitäten sowohl Informationen wie auch Verhalten zuweisen.
D4	Das System soll Design Patterns verwenden um ein reiches Domain Model zu erstellen.
D5	Das System muss das Domain Model von der Datenbank vollständig entkoppeln.

Tabelle 2.5: Anforderungen Domain Model

2.3.3 Table Module

Ein Table Module ist eine einzelne Instanz welches die gesamte Geschäftslogik für alle Zeilen einer Datenbanktabelle oder -view beinhaltet (nach [FRF+02]).

Eine der zentralen Aspekte der objektorientierten Programmierung ist es Daten mit dem zu den Daten gehörigen Verhalten zusammenzufassen. Der traditionelle objektorientierte Ansatz basiert auf Objekten mit einer eindeutigen Identität. Das heisst jede Instanz einer Klasse entspricht einer Zeile einer Datenbank. Eines der Probleme des Domain Models ist die Schnittstelle zu relationalen Datenbanken. Ein Table Module organisiert Geschäftslogik in einer Klasse pro Tabelle einer Datenbank und eine einzige Instanz dieser Klasse enthält alle Prozeduren um mit den Daten dieser Tabelle zu interagieren.

Eine Stärke des Table Modules ist die Möglichkeit des Zusammenfassens von Daten und Verhalten zur selben Zeit um vereinfacht mit einer relationalen Datenbank zu interagieren. Das Table Module verhält sich wie ein ganz normales Objekt, der grosse Unterschied liegt darin, dass das Objekt keine Ahnung von der Identität der Objekte hat, mit welchen es interagiert. Das heisst, wenn zum Beispiel ein einzelner Kunde gelesen werden soll, muss ein Aufruf ähnlich wie CustomerModule.GetCustomer(int customerId) ausgeführt werden. Jedes Mal wenn mit dem Kunden mit der ID customerId interagiert werden soll, muss eine Art von Identitätsreferenz ans Table Module übergeben werden (hier in Form des Integers customerId).

Die offensichtliche Verwendung des Table Module Patterns besteht darin ein Table Module für jede Tabelle der Datenbank zu erstellen. Es ist jedoch auch möglich für einzelne Queries oder Views eigene Table Modules zu implementieren. Das Table Module kann bei jeder Verwendung als neue Instanz erstellt werden oder es kann auch als eine Sammlung von statischen Methoden implementiert werden. Der Vorteil einer Instanz ist es, dass das Table Module bereits mit einem Record Set[6] oder einem bestehenden Resultat eines Queries initialisiert werden könnte. Dieses Resultat könnte dann mit der

[6] Mehr zum Record Set auf `http://martinfowler.com/eaaCatalog/recordSet.html`

Instanz des Table Modules manipuliert werden.

Das Table Module baut stark auf tabellarischen Daten auf. Es macht also Sinn das Table Module im Zusammenhang mit einem Record Set zu verwenden.

Die Anforderungen an das Table Module sind in Tabelle 2.6 in natürlichsprachiger Form dokumentiert.

Bezeichner	Anforderung
T1	Das System muss die Objektstruktur analog der Datenbankstruktur abbilden.
T2	Das System muss eine einzelne Instanz zur Verfügung stellen, welche die gesamte Geschäftslogik aller Zeilen einer Datenbanktabelle beinhaltet.
T3	Das System muss eine einfache Domänenlogik verwenden.

Tabelle 2.6: Anforderungen Table Module

2.3.4 Active Record

Gemäss [FRF+02] kapselt eine Implementierung des Active Record Patterns eine Zeile einer Datenbanktabelle oder -view, den Datenbankzugriff darauf und enthält die Geschäftslogik im Zusammenhang mit dieser Zeile. Ein Objekt enthält Daten und Verhalten. Der Grossteil dieser Daten ist persistent und wird in einer Datenbank gespeichert. Active Record verwendet diesen Ansatz und fügt die Geschäftslogik dem Domänenobjekt hinzu.

Die Essenz von Active Record ist ein simples Domain Model in welchem die Domänenobjekte stark mit der darunterliegenden Datenbankstruktur verwandt sind. Jedes Active Record ist verantwortlich für das Speichern und Laden der Objekte aus der Datenbank und auch für jegliche Domänenlogik welche mit diesen Objekten verbunden ist. Dies kann zum Beispiel sämtliche Domänenlogik der Applikation sein oder es kann auch die zwischen Active Records gemeinsame Domänenlogik in Transaction Scripts implementiert werden und datenorientierter Code im jeweiligen Active Record abgelegt sein.

Die Datenstruktur des Active Records sollte exakt mit der Datenbank übereinstimmen: ein Feld in der Klasse für eine Spalte in der Tabelle. Die Namen der Felder sollten direkt mit denjenigen der Tabelle übereinstimmen, es sollte keine Datenkonversion im Active Record implementiert werden. Typischerweise bietet das einzelne Active Record folgende Methoden an:

- Konstruieren einer neue Instanz des Active Records mit dem bestehenden Resultat einer SQL Query
- Konstruieren einer neuen Instanz des Active Records welche später in die Datenbanktabelle gespeichert wird
- Statische Suchmethode welche oft genutzte SQL Queries kapselt und als Active Record Objekte zurückgibt
- Aktualisieren des Active Records mit den aktuellen Werten aus der Datenbank

- Lesen und Speichern der Felder (Getter- und Setter-Methoden)
- Implementieren von Teilen der Geschäftslogik

Active Record ist eine gute Option für Systeme mit vergleichsweise simpler Geschäftslogik. Validierungen auf Basis eines einzelnen Datensatzes funktionieren gut mit dieser Struktur. Wird ein Domain Model verwendet so besteht zu Beginn der Konzeptphase die Entscheidung zwischen der Verwendung von Active Records oder eines Data Mappers. Das Active Record hat den Vorteil der Einfachheit, wird das Domain Model komplex und es werden zusätzliche Beziehungen, Felder, Vererbungen oder Listen hinzugefügt, so wird das Active Record unübersichtlich und ein Data Mapper ist dann einfacher zu verwenden.

Die Anforderungen an den Persistenzlayer einer serviceorientierten Architektur in Bezug auf das Active Record Pattern sind in Tabelle 2.7 festgehalten.

Bezeichner	Anforderung
A1	Das System muss die Objektstruktur analog der Datenbankstruktur abbilden.
A2	Das System muss keine Konversion von Objektdatentypen und Datentypen der Datenbanktabelle durchführen.
A3	Das System muss eine einfache Domänenlogik verwenden.

Tabelle 2.7: Anforderungen Active Record

2.4 Anforderungen serviceorientierte CRM-Applikation

Ein Ziel dieser Bachelorarbeit besteht darin ein Proof of Concept einer serviceorientierten Architektur unter dem Einsatz von Enterprise Pattern zu erstellen. Dies einerseits um die Anforderungen aus den vorhergehenden Kapiteln zu überprüfen aber auch um Erfahrung mit den verschiedenen OR-Mappern zu sammeln, welche im Rahmen dieser Arbeit miteinander verglichen werden.

Da das Proof of Concept einen grossen Teil der aufgewendeten Stunden für diese Bachelorarbeit ausmacht, wird Wert darauf gelegt, dass die Applikation methodisch korrekt entwickelt wird. Die folgenden Kapitel beschreiben daher in einem ersten Schritt ausführlich die Anforderungen an die CRM-Applikation als Proof of Concept.

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 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 [PR11]).

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 2.1 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 Geschäftsprozesse, welche im Umgang mit dem Kunden und Kontakten eingehalten werden müssen, geben Workflows als Rahmenbedingungen vor. Diese Prozesse sind in der Geschäftslogik spezifiziert und interagieren über diese Workflows mit dem System. Der Administrator verwaltet die Benutzer des Systems.

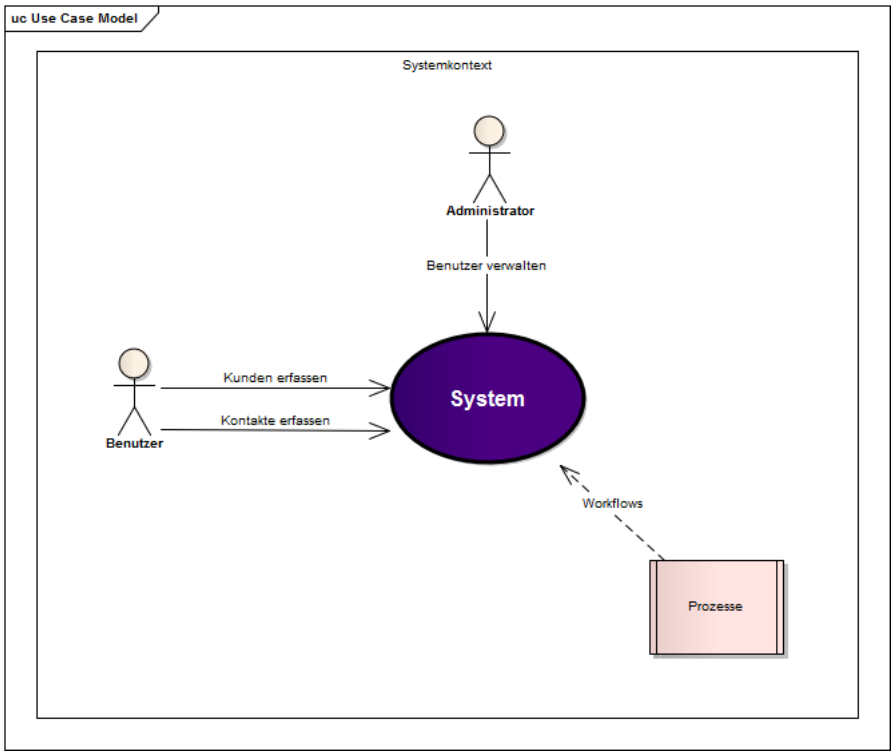

Abbildung 2.1: Systemkontext

2.4.2 Use-Case-Spezifikationen

Nach [PR11] 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-Cases 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 10) werden als Use-Cases modelliert und spezifiziert[7]. Als Quellen der Anforderungen dienen der Betreuer, das Reglement der ZHAW betreffend der Bachelorarbeit 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 Gesamtfunktionalität des Systems stellen 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 authentifiziert falls diese dem System bekannt sind. Er kann nun weitere Aktionen mit dem System tätigen.

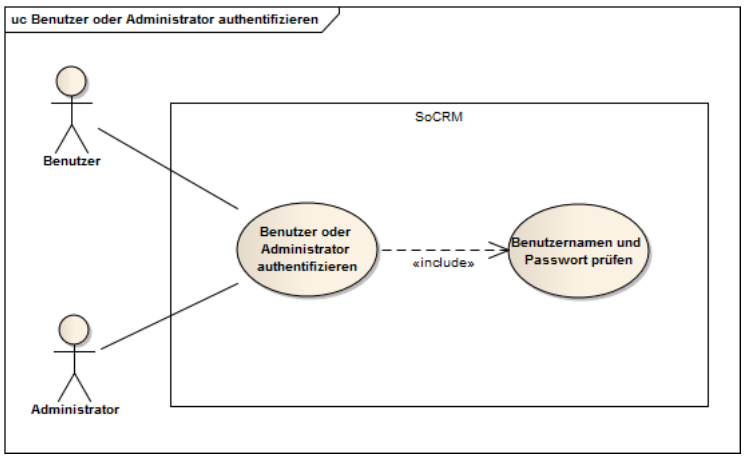

Abbildung 2.2: Use-Case Benutzer oder Administrator authentifizieren

[7] Die verwendete Schablone stammt aus [PR11] 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.

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 authentifiziert 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 diese dem Benutzer oder Administrator.
Alternativszenarien	2a. Der Benutzername oder das Passwort ist nicht korrekt. 2a1. Der Benutzer oder Administrator wird aufgefordert seinen Benutzernamen und Passwort zu überprüfen und erneut einzugeben.
Ausnahmeszenarien	Auslösendes Ereignis: Der Benutzer oder Administrator kann keine Verbindung zum System herstellen.

Tabelle 2.8: Use-Case-Spezifikation Benutzer oder Administrator authentifizieren

2.4.2.2 Kunden anzeigen

Der Benutzer ist beim System authentifiziert und zeigt eine Liste aller im System gespeicherten Kunden an. Der Benutzer kann die Liste über verschiedene Suchkriterien (Name des Kunden und dessen Arbeitgeber) einschränken und mit den gefundenen Kunden interagieren. So kann er zum Beispiel neue Kontakte für die gefundenen Kunden hinzufügen oder den Kunden aus dem System löschen.

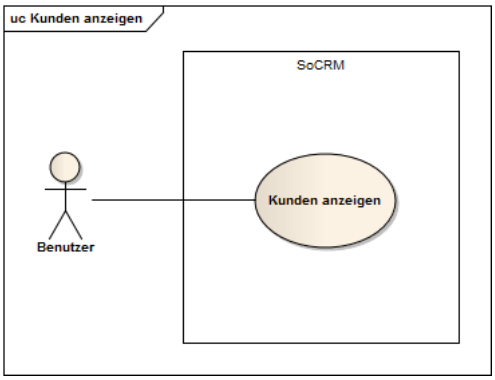

Abbildung 2.3: Use-Case Kunden anzeigen

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 gefundenen Kunden an.
Alternativszenarien	1a. Der Benutzer gibt keinen Namen ein. 2a. Der Benutzer gibt keinen Firmennamen ein.
Ausnahmeszenarien	Keine

Tabelle 2.9: Use-Case-Spezifikation Kunden anzeigen

2.4.2.3 Kunde hinzufügen

Der Benutzer fügt dem System einen neuen Kunden hinzu. Jeder Kunde hat neben Vor- und Nachnamen eine Adresse und einen Arbeitgeber. Ist der Arbeitgeber noch nicht als Firma im System eingetragen, so kann sie beim Erstellen des Kunden dem System hinzugefügt werden. Zusätzlich kann jedem Kunden eine oder mehrere E-Mail-Adressen sowie eine oder mehrere Telefonnummern hinzugefügt werden.

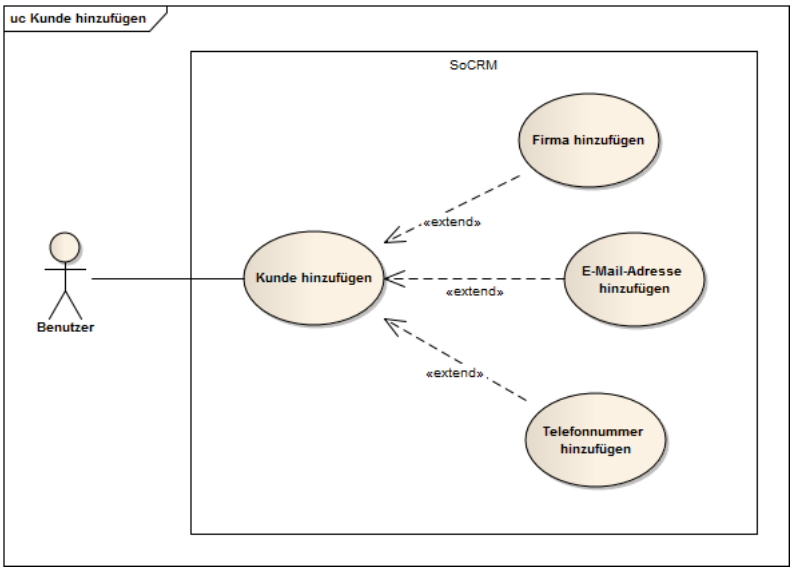

Abbildung 2.4: Use-Case Kunde hinzufügen

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 neu 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 28).
Ausnahmeszenarien	Auslösenden Ereignis: Der Kunde existiert schon. Der Benutzer wird aufgefordert zu überprüfen ob die Daten fehlerhaft eingetragen wurden oder ob dieser Kunde ein Duplikat darstellt.

Tabelle 2.10: 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. Gelöschte Kunden und ihre dazugehörigen Kontakte sind unwiederbringlich verloren und können nicht wiederhergestellt werden.

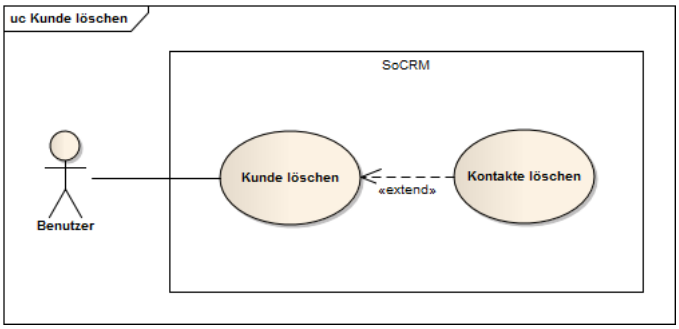

Abbildung 2.5: Use-Case Kunde löschen

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 im System (siehe Kapitel 2.4.2.2 auf Seite 22).
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 2.11: Use-Case-Spezifikation Kunde löschen

2.4.2.5 Firmen anzeigen

Der Benutzer ist beim System authentifiziert und zeigt eine Liste aller im System gespeicherten Firmen an. Jeder Firma können Kunden hinzugefügt werden, für welche wiederum Kontakte generiert werden können. Der Benutzer kann die Liste aller Firmen nach verschiedenen Suchkriterien (Firmenname und Land) filtern und mit den gefundenen Firmen interagieren.

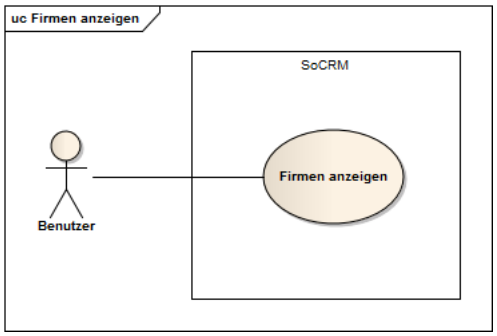

Abbildung 2.6: Use-Case Firmen anzeigen

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 Angestellten (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 gefundenen Firmen an.
Alternativszenarien	1a. Der Benutzer gibt keinen Firmennamen ein. 2a. Der Benutzer gibt kein Land an.
Ausnahmeszenarien	Keine

Tabelle 2.12: Use-Case-Spezifikation Firmen anzeigen

2.4.2.6 Firma hinzufügen

Der Benutzer fügt dem System eine neue Firma hinzu. Die Firma hat einen Firmennamen, eine Webseite sowie eine Firmenadresse. Nach dem Hinzufügen der Firma können der Firma Angestellte (Kunden) hinzugefügt werden. Für diese Kunden können danach Kontakte generiert werden.

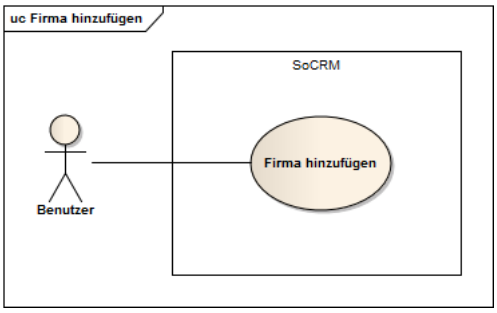

Abbildung 2.7: Use-Case Firma hinzufügen

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 diese Firma ein Duplikat darstellt.
Ausnahmeszenarien	Keine

Tabelle 2.13: 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 die Angestellten der Firma Kontakte, so werden auch die Kontakte aus dem System gelöscht. Gelöschte Firmen, Kunden oder Kontakte sind endgültig gelöscht und unwiederbringlich verloren.

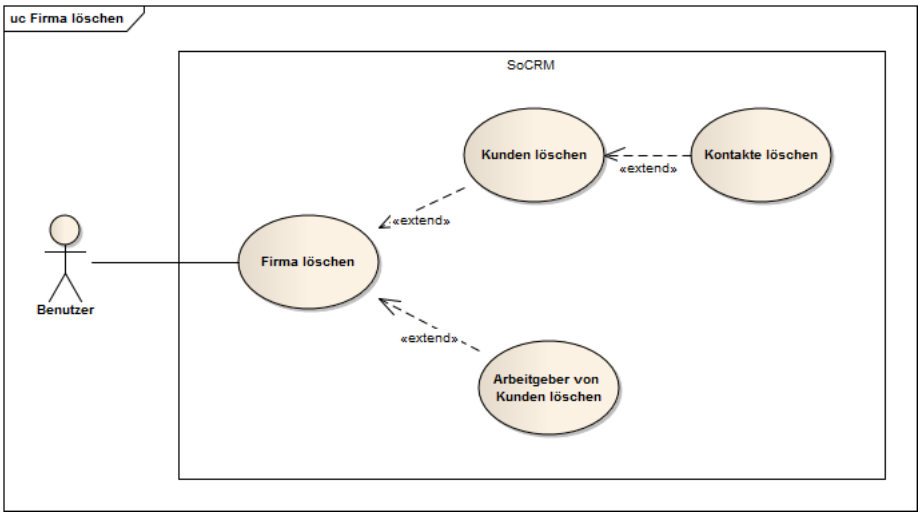

Abbildung 2.8: Use-Case Firma löschen

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 im System (siehe Kapitel 2.4.2.5 auf Seite 27).
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 (Kunden) besitzt wird der Benutzer darauf hingewiesen, dass auch die zur Firma gehörigen Angestellten gelöscht werden. 1b. Falls die Angestellten Kontakte besitzen wird der Benutzer darauf hingewiesen, dass auch die zum Angestellten gehörigen Kontakte gelöscht werden.
Ausnahmeszenarien	Keine

Tabelle 2.14: Use-Case-Spezifikation Firma löschen

2.4.2.8 Kontakte anzeigen

Der Benutzer möchte Kontakte anzeigen, welche im System erfasst sind. Dazu sucht er entweder nach einem Kunden oder nach dem Medium, über welches der Kontakt zustandegekommen ist (mögliche Suchkriterien: Vor- oder Nachname des Kunden, Kontaktmedium). Die gefunden Kontakte kann der Benutzer entweder im Detail anzeigen lassen oder diese aus dem System löschen.

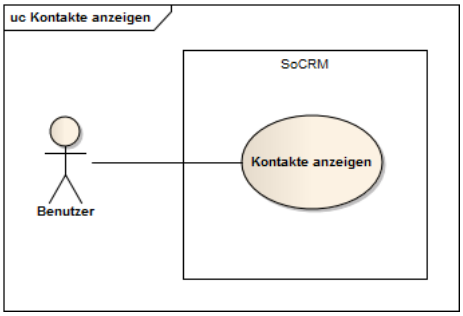

Abbildung 2.9: Use-Case Kontakte anzeigen

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 2.15: Use-Case-Spezifikation Kontakte anzeigen

2.4.2.9 Kontakt hinzufügen

Der Anwendungsfall "Kontakt hinzufügen" wird ausgeführt wenn der Benutzer einem Kunden einen neuen Kontakt hinzufügen möchte. Der Kontakt umfasst den Kontaktzeitpunkt, das Kontaktmedium[8] sowie die textuelle Beschreibung des Kontakts.

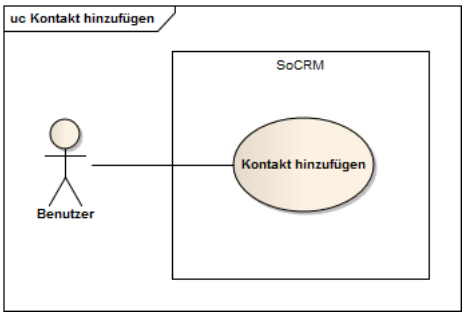

Abbildung 2.10: Use-Case Kontakt hinzufügen

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 und das Kontaktmedium aus und gibt den Inhalt des Kontakts ein. 2. Der Benutzer speichert den Kontakt und dieser wird ihm angezeigt.
Alternativszenarien	Keine
Ausnahmeszenarien	Keine

Tabelle 2.16: Use-Case-Spezifikation Kontakt hinzufügen.

[8] Mögliche Kontaktmedien umfassen: Messe, Telefon oder E-Mail.

2.4.2.10 Kontakt löschen

Der Benutzer möchte den Kontakt eines Kunden löschen. Dazu führt er diesen Anwendungsfall aus. Nach erfolgreichem Löschen des Kontakts ist dieser nicht mehr wiederherzustellen.

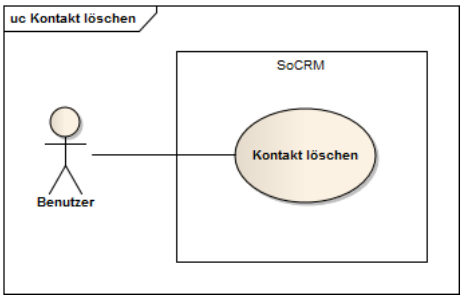

Abbildung 2.11: Use-Case Kontakt löschen

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 31).
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 2.17: Use-Case-Spezifikation Kontakt löschen

2.4.2.11 Benutzer anzeigen

Der Administrator möchte die im System gespeicherten Benutzer anzeigen. Er kann entweder alle Benutzer anzeigen oder er sucht nach dem Namen eines spezifischen Benutzers oder dessen Rolle (Benutzer oder Administrator). Den oder die gefundenen Benutzer kann er entweder im Detail betrachten, löschen oder ihr Passwort zurücksetzen.

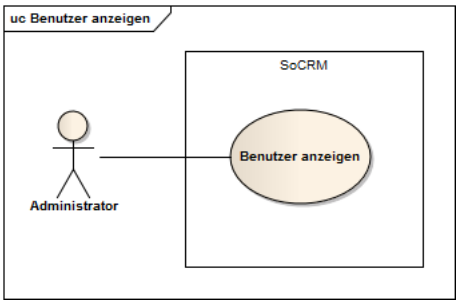

Abbildung 2.12: Use-Case Benutzer anzeigen

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, welche die zu suchenden Benutzer innehaben.
Alternativszenarien	1a. Der Administrator gibt keinen Benutzernamen ein. 2a. Der Administrator wählt keine Rolle aus.
Ausnahmeszenarien	Keine

Tabelle 2.18: Use-Case-Spezifikation Benutzer anzeigen

2.4.2.12 Passwort setzen

Der Administrator kann die Passwörter aller Benutzer neu setzen, der Benutzer jeweils nur sein eigenes Passwort. Dazu wird verlangt dass das alte Passwort eingegeben wird und danach das neue Passwort eingegeben wird. Nach dem Setzen des eigenen Passworts wird der Benutzer ausgeloggt und muss sich mit seinem eben gesetzten Passwort neu einloggen.

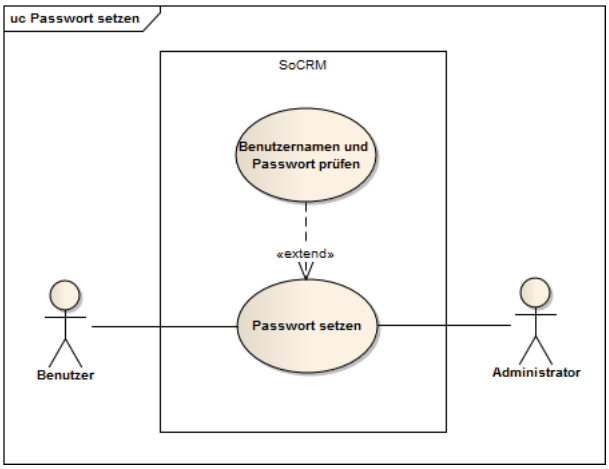

Abbildung 2.13: Use-Case Passwort setzen

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 dasjenige jedes anderen Benutzers.
Auslösendes Ereignis	Der Benutzer möchte sein eigenes Passwort setzen. Der Administrator möchten sein eigenes oder dasjenige jedes anderen Benutzers setzen.
Akteure	Benutzer, Administrator
Vorbedingung	Der Benutzer oder Administrator ist beim System authentifiziert.
Nachbedingung	Der Benutzer 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 2.19: Use-Case-Spezifikation Passwort setzen

2.4.2.13 Benutzer hinzufügen

Um einen neuen Benutzer dem System hinzuzufügen führt der Administrator diesen Anwendungsfall aus. Der neue Benutzer erhält einen Benutzernamen, ein Passwort sowie eine Rolle. Nach dem Hinzufügen des Benutzers durch den Administrators kann sich dieser beim System mit den vom Administrator eingegebenen Benutzerdaten anmelden.

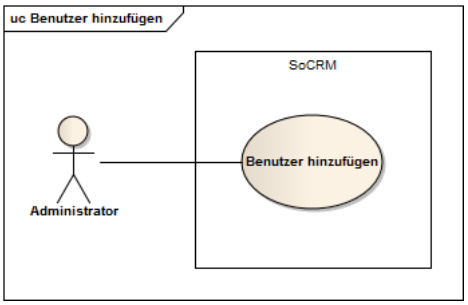

Abbildung 2.14: Use-Case Benutzer hinzufügen

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 dessen 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 dieser Benutzer ein Duplikat darstellt.

Tabelle 2.20: Use-Case-Spezifikation Benutzer hinzufügen.

2.4.2.14 Benutzer löschen

Der Administrator kann Benutzer aus dem System entfernen. Dazu führt er den Anwendungsfall “Benutzer löschen” aus. Falls der Benutzer zum Zeitpunkt der Löschung beim System angemeldet ist, kann der Administrator entscheiden, ob dessen aktuelle Authentifizierung für ungültig erklärt werden soll bevor er vom System gelöscht wird.

Ist der Benutzer aus dem System gelöscht, kann sich dieser nicht mehr beim System anmelden und er wird in der Liste der im System verfügbaren Benutzer nicht mehr angezeigt.

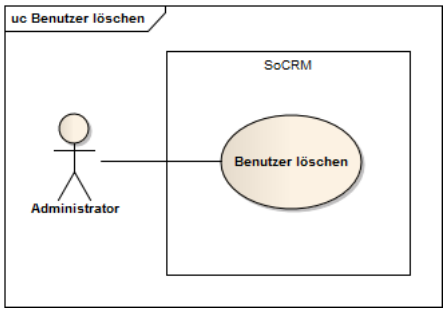

Abbildung 2.15: Use-Case Benutzer löschen

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 34). Der gelöschte Benutzer kann sich nicht mehr beim System authentifizieren (siehe Kapitel 2.4.2.1 auf Seite 20).
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 widerrufen soll.

Tabelle 2.21: 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 [OSKZ06]). Das Konzept dieser Bachelorarbeit umfasst das Erarbeiten einer Marktübersicht aller frei verfügbaren oder Open Source OR-Mapper im .NET-Umfeld, das Auswählen von drei Produkten aus der Marktübersicht für weitergehende Vergleiche und das Erarbeiten dieser Vergleichsresultate.

Zusätzlich wird die Architektur des Proof of Concepts in Form der CRM-Applikation erarbeitet. 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 folgende Marktübersicht enthält die aktuell frei verfügbaren oder Open-Source OR-Mapper im .NET-Umfeld. Jeder OR-Mapper wird auf Basis einer Vielzahl von Kriterien untersucht und miteinander verglichen. Die Marktübersicht ist kein wertender Vergleich der Produkte sondern eine reine Erhebung von Tatsachen zum heutigen Zeitpunkt. Es ist wichtig anzumerken, dass diese Marktübersicht schnell veraltet sein wird, da viele der Produkte einen kurzen Releasezyklus besitzen in welchem sie zusätzliche Funktionalitäten zur Verfügung erhalten.

In diesem Kapitel werden auch die einzelnen Vergleichskriterien erklärt und beschrieben warum diese wichtig sind im Zusammenhang mit der Untersuchung eines OR-Mappers. Das Ausfüllen der Marktübersicht erfolgt einerseits durch Konsultation der Dokumentation bzw. der Anbieterwebseite andererseits durch das Sammeln von Erfahrung in der Benutzung des OR-Mappers und durch die Analyse des verfügbaren Quellcodes (wo Open Source). Innerhalb der Marktübersicht wird nicht dokumentiert welche Information aus welcher Quelle stammt, da dies den Rahmen der Marktübersicht sprengen würde.

3.1.1 Vergleichskriterien

Untenstehend finden sich alle Vergleichskriterien, welche auf die betrachteten OR-Mapper angewendet werden. Die Kriterien werden einzeln im Detail beschrieben und sollen eine Hilfestellung beim Betrachten der Vergleichsmatrix (siehe Kapitel 3.1.2 auf Seite 47) bieten.

[1] auch Designphase genannt

Die Vergleichskriterien, die in der Marktübersicht verwendet werden, sind auf [Sch08] aufgebaut. Alle weiteren Kriterien stammen aus eigener Erfahrung beziehungsweise aus Literatur wie [Koc11] und [CdE+10].

3.1.1.1 Produkt

Name Der Name des OR-Mappers.

Anbieter Der Anbieter des OR-Mappers falls ein Unternehmen oder eine Gruppierung hinter dem Produkt steht. Bei Open Source Produkten wird "Open Source" verwendet.

Website Die Webseite des OR-Mappers.

Zeitpunkt der ersten Version Der Zeitpunkt der ersten verfügbaren Version. Hier werden auch Betaversionen berücksichtigt, welche im produktiven Umfeld eingesetzt wurden (nach Dokumentation). Bei Open Source Produkten wird der erste Zeitpunkt eines Releases mit Dokumentation berücksichtigt.

Aktuelle und untersuchte Version Die im Rahmen dieser Marktübersicht untersuchte Version.

Lizenz Die Lizenz, unter welchem der OR-Mapper nutzbar ist. Es werden nur OR-Mapper berücksichtigt, welche ohne Lizenzabgaben kommerziell nutzbar sind.

Typ Der Typ des OR-Mappers. Es gibt zwei unterschiedliche Typen von OR-Mappern: Leichtgewichtige und komplette OR-Mapper. Ein leichtgewichtiger OR-Mapper bietet meist ausser der Implementierung eines Data Mappers keine weiteren Funktionalitäten wie Monitoringtools, Assistenten oder grafische Designer. Ein kompletter OR-Mapper kommt hingegen oft mit einer ganzen Armada an zusätzlichen Funktionalitäten, welche den Entwickler beim Integrieren des OR-Mappers und der Verwendung desselben unterstützen.

3.1.1.2 Anforderungen

Unterstützte Datenbankmanagementsysteme Eine Liste der laut Produktbeschreibung unterstützten Datenbanksysteme. Fast alle OR-Mapper basieren auf der Schnittstelle System.Data.IDbConnection[2] des .NET Frameworks. Daher sind sie theoretisch datenbankunabhängig solange der DBMS-Hersteller einen .NET Datenprovider anbietet. Es hat sich jedoch herausgestellt, dass diese Datenbankunabhängigkeit auf Grund der unterschiedlichen Qualität der Datenprovider schwierig zu realisieren ist. Deshalb sind in dieser Zeile nur Datenbankmanagementsysteme aufgelistet die auch so vom Hersteller des OR-Mappers getestet und freigegeben wurden.

[2] IDbConnection repräsentiert eine beliebige Datenquelle zu relationalen Datenbanken, siehe `http://msdn.microsoft.com/en-us/library/system.data.idbconnection.aspx`.

Minimale .NET-Version Die minimal notwendige Version des .NET Frameworks um den OR-Mapper integrieren zu können.

Unterstützt Mono Beschreibt ob der OR-Mapper die Softwareplattform Mono[3] unterstützt.

3.1.1.3 Tooling

Projektvorlagen für Visual Studio Dieses Kriterium sagt aus ob der OR-Mapper Projektvorlagen für Microsofts Visual Studio mitliefert.

Elementvorlagen für Visual Studio Dieses Kriterium sagt aus ob der OR-Mapper Elementvorlagen für Microsofts Visual Studio mitliefert.

Extension für Visual Studio Beschreibt ob der OR-Mapper eine Erweiterung für Microsofts Visual Studio mitliefert.

Designer Sagt aus ob der OR-Mapper einen grafischen Designer mitliefert womit vereinfacht Mappings erstellt werden können.

Assistenten Sagt aus ob der OR-Mapper Assistenten mitliefert, welche den Entwickler beim Erstellen der Mappings unterstützen.

Kommandozeilentools Falls der OR-Mapper zusätzliche kommandozeilenbasierte Tools mitliefert sind diese in dieser Zeile aufgelistet.

3.1.1.4 Mapping-Techniken

Benutzte interne Datenzugriffs-API Beschreibt die innerhalb des OR-Mappers verwendete API-Datenzugriffsschnittstelle mit welcher auf das Datenbankmanagementsystem zugegriffen wird.

Verwendete Techniken Unter diesem Kriterium sind die vom OR-Mapper verwendeten Techniken beschrieben, welche das Mapping von Zeilen in der Tabelle einer relationalen Datenbank zu Domänenobjekten realisiert. Dies kann zum Beispiel Codegenerierung auf Grund eines Designers sein oder Techniken wie Reflection. Dieses Kriterium ist ein möglicher Anhaltspunkt um die Performance des OR-Mappers abzuschätzen. Wird nur Code zur Laufzeit erzeugt und stark Techniken wie Reflection verwendet, so wird sich dies mit Sicherheit auf die Laufzeitperformance des Produktes auswirken.

Forward Mapping (Model-First) Beschreibt ob der OR-Mapper Forward Mapping unterstützt. Forward Mapping bedeutet, dass zuerst das Domänenmodell in Code

[3] Mehr zu Mono unter `http://www.mono-project.com/Main_Page`.

modelliert wird und danach die Datenbank vom OR-Mapper auf Basis dieses Domänenmodells generiert wird. Dies setzt voraus, dass der OR-Mapper SQL DDL[4] generieren kann.

Reverse Mapping (Schema-First) Beschreibt ob der OR-Mapper Reverse Mapping unterstützt. Reverse Mapping ist eine Mappingtechnik, welche auf Grund einer bestehenden Datenbank ein Domänenmodell erzeugt. Dies setzt voraus, dass der OR-Mapper C# oder VB.NET Code generieren kann.

Datenmigrationen Unterstützt ein OR-Mapper Datenmigrationen so ist dies in dieser Zeile festgehalten. Eine Datenmigration ist das Unterstützen von versionierten Domänenmodellen bzw. Datenbankschemas und den darin enthaltenen Daten. Wird eine Datenmigration durchgeführt, so wird das Datenschema inklusive den darin gespeicherten Daten automatisiert auf den Zielstand der Datenmigration gebracht. Im Rahmen dieser Marktübersicht gelten Datenmigrationen nur als unterstützt, wenn diese auch rückgängig gemacht werden können.

3.1.1.5 Business Objects

POCO Support POCOs sind Plain Old CLR Objects. Das heisst ein POCO ist ein simples Objekt welches ohne die Verwendung eines zusätzlichen Frameworks wie einer bestimmter Basisklasse, implementierten Interfaces oder Annotationen auskommt. Unterstützt ein OR-Mapper direkt POCOs bei der Verwendung von Forward Mapping, so wird dies unter diesem Kriterium festgehalten.

Erfordern spezifische Basisklasse Verlangt ein OR-Mapper, dass jedes Domänenobjekt von einer spezifischen Basisklasse erbt, so ist dies in der Matrix unter dieser Zeile festgehalten.

Erfordern Annotation Unter diesem Kriterium wird beschrieben ob ein OR-Mapper bestimmte Annotationen auf Klassen, Feldern, Eigenschaften oder Methoden voraussetzt, um mit diesen Domänenobjekten umgehen zu können.

Erfordern parameterlosen Konstruktor Falls der OR-Mapper verlangt, dass alle Domänenobjekte einen parameterlosen Konstruktor aufweisen, wird dies unter diesem Kriterium beschrieben.

Müssen abstrakt sein Einige, insbesondere ältere OR-Mapper, verlangen dass alle Domänenobjekte als abstrakt definiert sind, so dass zur Laufzeit eine Subklasse generiert werden kann. Falls dem so ist, wird dies in dieser Zeile festgehalten.

Erzeugte Business Objects unterstützen Datenbindung Falls die generierten Domänenobjekte bei Verwendung von Reverse Mapping Datenbindungen für WPF, Silverlight und ASP.NET unterstützen, sind in dieser Zeile der Vergleichsmatrix die implementierten und unterstützten Interfaces aufgelistet.

[4] DDL steht für Data Definition Language und ist die Syntax um Datenbankschemas und Datenstrukturen zu erstellen.

Erzeugte Business Objects sind als [Serializable] gekennzeichnet Sind die generierten Domänenobjekte von Haus aus bereits mit dem SerializableAttribute[5] gekennzeichnet, so wird dies unter diesem Kriterium in der Marktübersicht festgehalten. Das SerializableAttribute beschreibt, dass Klassen von einem .NET Serialisierer (z.B. beim Übertragen über WCF-Schnittstellen) serialisiert werden können.

Erzeugte Business Objects unterstützen WCF Contracts Unter diesem Kriterium wird beschrieben ob die vom betrachteten OR-Mapper generierten Domänenobjekte bei Verwendung von Reverse Mapping mit den von WCF vorausgesetzten Attributen annotiert sind oder nicht.[6]

Erzeuge Business Objects sind partielle Klassen Sind die vom OR-Mapper erzeugten Domänenobjekte als partielle Klassen deklariert, wird dies in dieser Zeile eingetragen.

3.1.1.6 Mapping-Arten

Persistiert Felder Unterstützt der OR-Mapper das Persistieren von Feldern wird dies in dieser Zeile eingetragen.

Persistiert Eigenschaften Unterstützt der OR-Mapper das Persistieren von Eigenschaften[7] wird dies in dieser Zeile eingetragen.

Unterstützung für nullbasierte Werte Unterstützt der OR-Mapper das Persistieren von nullbasierten Werten[8] wird dies in dieser Zeile eingetragen.

1 Klasse -> 1 Tabelle Unterstützt der OR-Mapper das Mappen einer Klasse auf eine Tabelle wird dies in dieser Zeile vermerkt.

1 Klasse -> n Tabellen Unterstützt der OR-Mappen das Mappen einer Klasse auf mehrere Tabellen wird dies in dieser Zeile vermerkt.

n Klassen -> 1 Tabelle Unterstützt der OR-Mapper das Mappen mehrerer Klassen auf eine Tabelle wird dies in dieser Zeile vermerkt.

n Klassen -> m Tabellen Unterstützt der OR-Mapper das Mappen mehrerer Klassen auf mehrere Tabellen wird dies in dieser Zeile vermerkt.

Unterstützung für Collection-Typen Dieses Kriterium beschreibt die vom OR-Mapper unterstützten Collection-Typen auf den Domänenobjekten (bei 1-zu-n-Beziehungen). Die Namen der unterstützten Collection-Typen werden in dieser Zeile in der Vergleichsmatrix eingetragen.

[5] Mehr zum SerializableAttribute in der MSDN unter `http://msdn.microsoft.com/en-us/library/system.serializableattribute.aspx`.

[6] Mehr zu den Data Contract Attributes in der MSDN unter `http://msdn.microsoft.com/en-us/library/ms733127.aspx`.

[7] Eigenschaften in Englisch sind Properties.

[8] Nullbasierte Werte sind vom Typ Nullable<T>, siehe MSDN `http://msdn.microsoft.com/en-us/library/b3h38hb0.aspx`.

Codebasiertes Mapping über Annotationen Geschieht das Mapping der Domänenobjekte beziehungsweise deren Felder oder Eigenschaften auf die Datenbanktabelle und deren Spalten mittels Annotationen innerhalb der Klassendefinition, wird dies in dieser Zeile vermerkt.

XML-basiertes Mapping/Mapping-Datei Benutzt der OR-Mapper für das Mapping der Domänenobjekte auf das Datenbankschema ein XML- oder dateibasiertes Mapping wird dies in dieser Zeile festgehalten.

Mapping zur Kompilierzeit Diese Zeile enthält ein "Ja", wenn das Mapping der Domänenobjekte auf das Datenbankschema zur Kompilierzeit erstellt wird.

Zusammengesetzte Schlüssel Unterstützt der OR-Mapper zusammengesetzte Schlüssel[9] und bildet diese Schlüssel auch korrekt auf den Domänenobjekten ab, so wird dies in dieser Zeile der Marktübersichtsmatrix vermerkt.

Referenz über den Schlüssel Kann der OR-Mapper korrekt Fremdschlüssel aus Datenbanktabellen[10] in den Domänenobjekten als Objektreferenzen darstellen, wird dies in dieser Zeile eingetragen.

3.1.1.7 Abfragen

Containername Alle OR-Mapper besitzen einen zentralen Datencontainer über welchen die Domänenobjekte ausgelesen oder gespeichert werden können. Diese Zeile enthält den Namen des Containers, welcher vom jeweiligen OR-Mapper zur Verfügung gestellt wird.

Abfragesprache Diese Zeile enthält die vom jeweilig betrachteten OR-Mapper unterstützte Abfragesprachen. Zu diesen Abfragensprachen gehören einerseits proprietäre Abfragesprachen wie HQL oder eSQL aber auch standardisierte Zugriffsmöglichkeiten wie LINQ oder SQL.

Parameterisierung der erzeugten SQL-Abfragen Falls der OR-Mapper Abfragen via SQL erlaubt, enthält diese Zeile eine Information darüber ob der OR-Mapper die Parameterisierung dieser Abfragen unterstützt.

Kompilierte Abfragen Um Abfragen über den OR-Mapper zu beschleunigen, können (vor-) kompilierte Abfragen verwendet werden, welche bereits zur Kompilierzeit optimiert werden. Unterstützt der OR-Mapper kompilierte Abfragen, wird dies unter diesem Kriterium festgehalten.

[9] Zusammengesetzte Schlüssel werden in T-SQL mit dem DDL Statement PRIMARY KEY (column1, column2) erzeugt. Siehe MSDN `http://msdn.microsoft.com/en-us/library/ms181043(v=sql.105).aspx`.

[10] In T-SQL mittels dem DDL Statement FOREIGN KEY (columnFK) REFERENCES table (columnID) erzeugte Spalten, siehe MSDN `http://msdn.microsoft.com/en-us/library/ms177463(v=sql.105).aspx`.

Lazy Loading Unterstützt der OR-Mapper das Lazy Loading von Eigenschaften, Feldern, Listen oder referenzierten Objekten eines Domänenobjekts wird dies in dieser Zeile notiert. Lazy Loading ist ein Entwurfsmuster welches das Initialisieren von Objekten, Feldern oder Eigenschaften eines Objekts so lange aufschiebt, bis auf die Information zugegriffen wird.

Eager Loading Unterstützt der OR-Mapper das explizite Eager Loading von Eigenschaften, Feldern, Listen oder referenzierten Objekten eines Domänenobjekts wird dies in dieser Zeile festgehalten. Eager Loading ist das Gegenteil von Lazy Loading. Bei explizitem Eager Loading von Informationen werden Performanceeinbussen beim initialen Lesen von Domänenobjekten bewusst in Kauf genommen um die Laufzeitperformance des Persistenzlayers zu erhöhen.

3.1.1.8 Datenaktualisierung

Speichern von geänderten Daten Kann der OR-Mapper geänderte Daten eines Domänenobjekts wieder in die Datenbank persistieren, wird dies in dieser Zeile vermerkt.

Änderungen bei einzelnen Objekten verfolgen Besitzt der OR-Mapper eine Möglichkeit Änderungen an einzelnen Domänenobjekten nachzuvollziehen, rückgängig zu machen oder vor dem Persistieren noch einmal zu überprüfen, so wird dies unter diesem Vergleichskriterium notiert.

Änderungen bei Collections verfolgen Kann der OR-Mapper zusätzlich zu den Änderungen an einzelnen Domänenobjekten auch ganze Collections überwachen wird dies in dieser Zeile festgehalten.

Unterstützung von Stored Procedures Insbesondere bei Reverse Mapping Szenarien ist es wichtig, dass der OR-Mapper Daten nicht nur über generierte SQL-Statements lesen kann sondern auch über (bereits bestehende) Stored Procedures. Unterstützt dies der betrachtete OR-Mapper wird dies in dieser Zeile vermerkt.

Unterstützung von Transaktionen Sind sämtliche Datenbankoperationen (ausser SELECT-Abfragen) über Transaktionen realisiert, wird dies in dieser Zeile festgehalten. Transaktionsbehandlung ist wichtig bei konkurrierenden Systemen mit mehreren gleichzeitigen Benutzern.

Pessimistisches Sperren (bei Transaktionen) Falls der OR-Mapper bei der Verwendung von Transaktionen auch pessimistische Sperren auf einzelnen Datensätzen oder Tabellen erlaubt wird dies hier vermerkt.

Optimistisches Sperren Optimistisches Sperren nimmt an, dass mehrere Transaktionen gleichzeitig ausgeführt werden können ohne dass sie sich gegenseitig behindern und aus diesem Grund keine Sperren notwendig sind. Vor dem Abschliessen der Transaktion wird geprüft ob keine andere Transaktion die in der Transaktion gelesenen oder modifizierten Daten verändert hat. Unterstützt der OR-Mapper diese Funktionalität wird dies unter diesem Vergleichskriterium festgehalten.

Konflikterkennung Kann der OR-Mapper Konflikte in Domänenobjekten erkennen (zum Beispiel wenn ein anderer Benutzer zur selben Zeit dasselbe Objekt in der Datenbank verändert) wird dies in der Marktübersichtsmatrix unter dieser Zeile notiert.

Änderungen bei unverbunden Objekten verfolgen Werden Domänenobjekte serialisiert und zum Beispiel über WCF übertragen, verliert der Datencontainer des OR-Mappers den Bezug zu diesem Domänenobjekt. Wird das Objekt verändert und wieder zurück über WCF an den Persistenzlayer übertragen und der OR-Mapper kann trotz verlorenem Bezug die Änderungen am Domänenobjekt erkennen und wieder persistieren, so wird dies in dieser Zeile vermerkt.

3.1.1.9 Cache

First Level Cache Besitzt der OR-Mapper eine Art von First Level Cache auf Basis des Datencontainers des ORMs wird dies in dieser Zeile festgehalten.

Second Level Cache Existiert ein Second Level Cache unter Benutzung des Arbeitsspeichers, der Datenbank oder der Festplatte so wird dies in dieser Zeile festgehalten.

Verteiltes Caching Unterstützt der OR-Mapper bei einem Multiserverbetrieb eine Möglichkeit seinen Cache mit anderen Instanzen des OR-Mappers (bzw. des Persistenzlayers) zu teilen, wird dies verteiltes Caching genannt. Falls der betrachtete OR-Mapper dieses Szenario unterstützt wird dies in dieser Zeile vermerkt.

Logging/Tracing Besitzt der OR-Mapper Möglichkeiten zum Tracing oder Logging von Operationen wie dem Ausführen von SQL-Statements oder dem Mapping von Werten aus der Datenbank auf Eigenschaften und Felder von Domänenobjekten, wird dies in dieser Zeile festgehalten. Falls der OR-Mapper Interceptions erlaubt, welche das Implementieren eigener Tracer ermöglichen, gilt dies auch als Möglichkeit zum Logging/Tracing.

Windows-Leistungsindikatoren Benutzt der OR-Mapper die Leistungsindikatoren von Windows um Informationen zu seinem Status zu publizieren, so wird dies in dieser Zeile vermerkt.

3.1.2 Vergleichsmatrix

Die Vergleichsmatrix vergleicht die OR-Mapper Entity Framework, NHibernate, LINQ-to-SQL, OpenAccess ORM, mybatis.NET, Business Logic Toolkit, SubSonic, Dapper, nHydrate und Signum Framework auf Basis der im vorgehenden Kapitel erfassten Vergleichskriterien miteinander auf den folgenden Seiten.

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 Objects 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 Produktauswahl

Das Ziel der Produktauswahl ist es, aus der Marktübersicht drei OR-Mapper zu identifizieren welche genauer betrachtet und verglichen werden sollen. Dazu wird eine nicht repräsentative Umfrage unter 30 erfahrenen .NET-Entwicklern durchgeführt, welche Verbreitung, Erfahrungswerte in der Benutzung und die Beliebtheit der in der Marktübersicht vorhandenen OR-Mapper bei Entwicklern messen soll.

Die drei auf Grund dieser Umfrage selektierten OR-Mapper werden in den nächsten Kapiteln im Detail miteinander verglichen, so dass eine Empfehlung erstellt werden kann, für welches Enterprise Pattern welcher OR-Mapper eingesetzt werden soll.

3.2.1 Umfrage

Das Ziel der Umfrage besteht darin drei OR-Mapper für weitergehende Vergleiche auszuwählen. Um dieses Ziel zu erreichen werden verschiedene Fragen betreffend der Bekanntheit, dem Ruf des Produkts und der Nutzung von OR-Mappern im .NET-Umfeld gestellt. Die 25 erhaltenen Antworten werden in den folgenden Unterkapiteln grafisch dargestellt und interpretiert. Am Ende der Umfrage werden auf Grund der Umfrageergebnisse drei OR-Mapper ausgewählt und die Auswahl begründet.

Um die Umfrage einzugrenzen wurden nur die folgenden, in der Marktübersicht (siehe Kapitel 3.1 auf Seite 40) betrachteten Produkte zur Auswahl zugelassen:

- Entity Framework
- OpenAccess ORM
- BLToolkit
- Dapper
- MyBatis.NET
- LINQ-to-SQL
- NHibernate
- SubSonic
- Signum Framework

3.2.2 Fragebogen

Der Fragebogen, der den Entwicklern ausgeteilt wird, umfasst die folgenden Fragen:

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 verbreitetsten ein?

5. Für ein privates Projekt, welchen ORM würdest du einsetzen?

Die Fragen 1 und 2 erlauben die Nennung mehrerer Produkte. Die Fragen 3, 4 und 5 erlauben es nur ein einziges Produkt zu nennen.

3.2.3 Interpretation der Umfrageergebnisse

Die nachfolgenden Kapitel zeigen die Umfrageergebnisse für jede Frage grafisch auf und liefern den Versuch einer Interpretation dieser Grafiken und den dazugehörigen Umfragewerten.

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

Das Umfrageergebnis für die Frage "Welche der folgenden .NET OR-Mapper kennst du?" ist wenig überraschend. Das Entity Framework ist klar der bekannteste OR-Mapper auf Grund des Marketings, welches Microsoft betreibt. Das Entity Framework wird sehr stark von Microsoft als der Standard im OR-Mapping dargestellt und Frameworks wie ASP.NET MVC, welche das Entity Framework bereits integrieren, tun ihr übriges zum Resultat dieser Frage.

Die beiden weit verbreiteten OR-Mapper NHibernate und LINQ-to-SQL sind auch entsprechend bekannt. Überraschend ist dass Dapper als relativ junges Projekt (erstes Releasedatum März 2011) schon eine grosse Bekanntheit unter Entwicklern besitzt. Ich vermute dies liegt an den vielen Beiträgen auf Entwicklerplattformen wie StackOverflow oder CodeProject zum Thema Dapper.

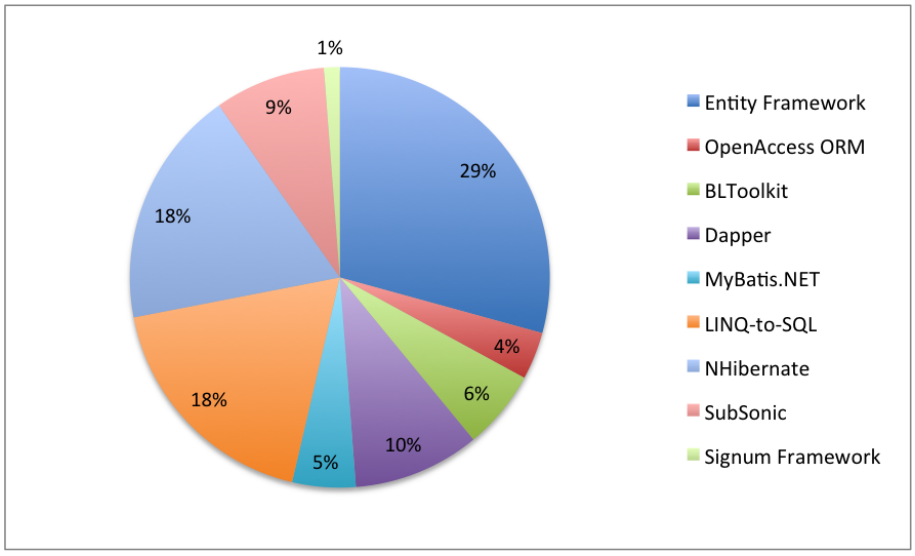

Abbildung 3.1: Umfrageresultate Welche der folgenden .NET OR-Mapper kennst du?

3.2.3.2 Welche davon hast du schon in einem Projekt eingesetzt?

Auch das Ergebnis auf die Frage "Welche davon hast du schon in einem Projekt eingesetzt?" ist wenig überraschend. Das Entity Framework und dessen Quasivorgänger LINQ-to-SQL nehmen zwei Drittel der Umfrageergebnisse ein. Vor dem Release des Entity Framework in der Version 4.0 mit seinen Möglichkeiten zum Reverse Mapping war NHibernate der wohl populärste OR-Mapper, was sich auch in dieser Frage widerspiegelt. Es gibt heute noch viele Legacy-Projekte welchen NHibernate zu Grunde liegt und welche auf Grund der Wechselkosten wohl auch keinen Austausch des OR-Mappers planen.

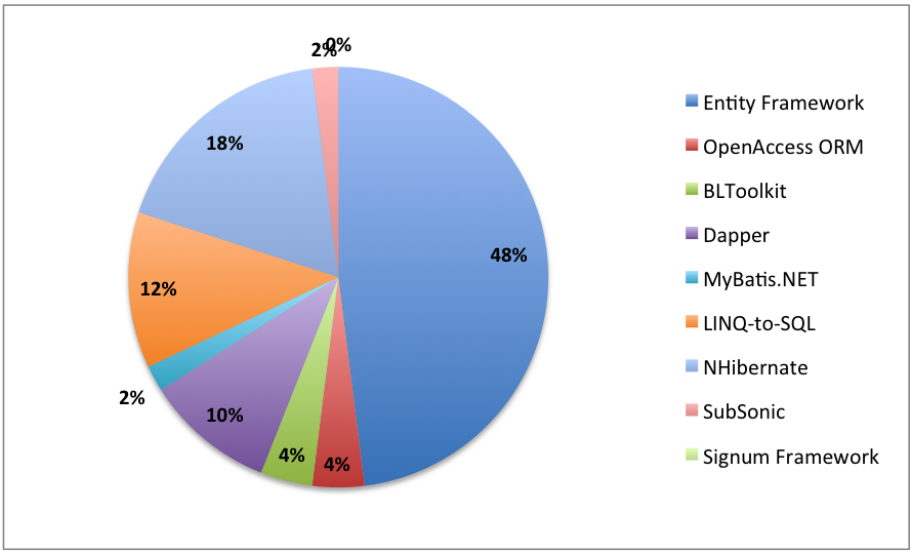

Abbildung 3.2: Umfrageresultate Welche davon hast du schon in einem Projekt eingesetzt?

3.2.3.3 Mit welchem Produkt hast du die besten Erfahrungen gemacht?

Die sehr subjektive Frage "Mit welchem Produkt hast du die besten Erfahrungen gemacht?" zielt darauf ab die teilweise langjährige Erfahrung der Umfrageteilnehmer zu ergründen. Das Entity Framework ist hier mit über zwei Dritteln der Stimmen wiederum klar in Führung. Ich vermute das dies primär an der guten Dokumentation von Microsoft liegt und daran dass das Entity Framework das wohl entwicklerfreundlichste Produkt der hier vertretenen kompletten OR-Mappern ist.

Erstaunlich ist das gute Abschneiden von Dapper. Dapper ist sehr flexibel und transparent im Erstellen der Mappings und da es in Quellcodeform ausgeliefert wird, sehr einfach zu debuggen und zu erweitern. Dies kann insbesondere für erfahrene Entwickler ein grosser Vorteil sein, da das Not-Invented-Here-Syndrom[11] unter Softwareentwicklern doch weit verbreitet ist.

[11] Mehr zum Not-Invented-Here-Syndrom bei Joel on Software: `http://www.joelonsoftware.com/articles/fog0000000007.html`.

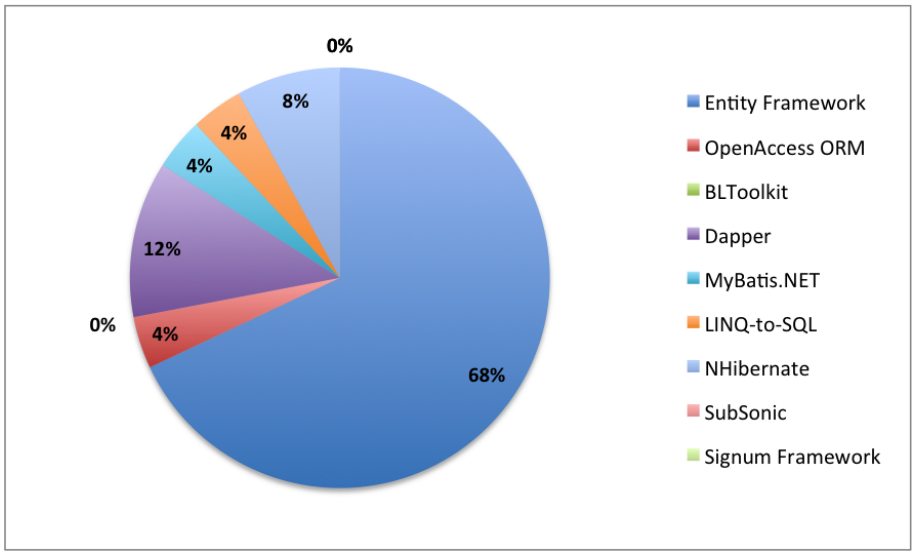

Abbildung 3.3: Umfrageresultate Mit welchem Produkt hast du die besten Erfahrungen gemacht?

3.2.3.4 Welches Produkt schätzt du am verbreitetsten ein?

Auf die Frage "Welches Produkt schätzt du am verbreitetsten ein?" wurden nur die drei Produkte Entity Framework, NHibernate und Dapper genannt. Das Entity Framework ist laut den Umfrageteilnehmern ganz klar das am weitesten verbreitete aller Produkte.

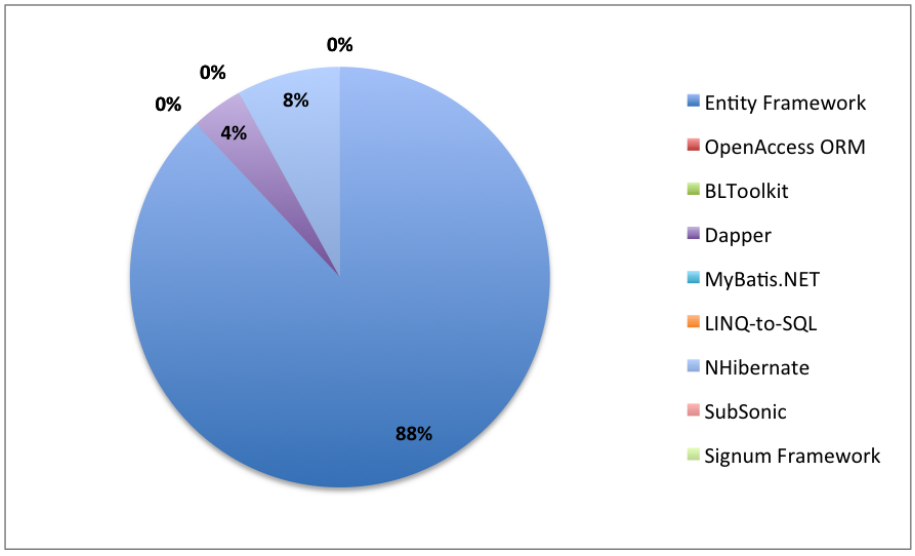

Abbildung 3.4: Umfrageresultate Welches Produkt schätzt du am verbreitetsten ein?

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

Das Umfrageergebnis für die Frage "Für ein privates Projekt, welchen ORM würdest du einsetzen?" zeigt, dass sich die in der Frage "Mit welchem Produkt hast du die besten Erfahrungen gemacht?" erhaltenen Umfrageergebnisse fast eins zu eins auf diese Frage abbilden lassen. Für private Projekte steht oft der Entwicklerkomfort im Vordergrund bei der Auswahl eines OR-Mappers. Es bestehen meist auch keine Altlasten in Form eines bestehenden Datenbankschemas oder eines veralteten DBMS dem man sich anzupassen hat.

Das Entity Framework ist wiederum der klare Gewinner dieser Umfrage vor Dapper, welcher als leichtgewichtiger OR-Mapper einfach benutzbar ist und auf Grund seiner Architektur bei Entwicklern beliebt ist.

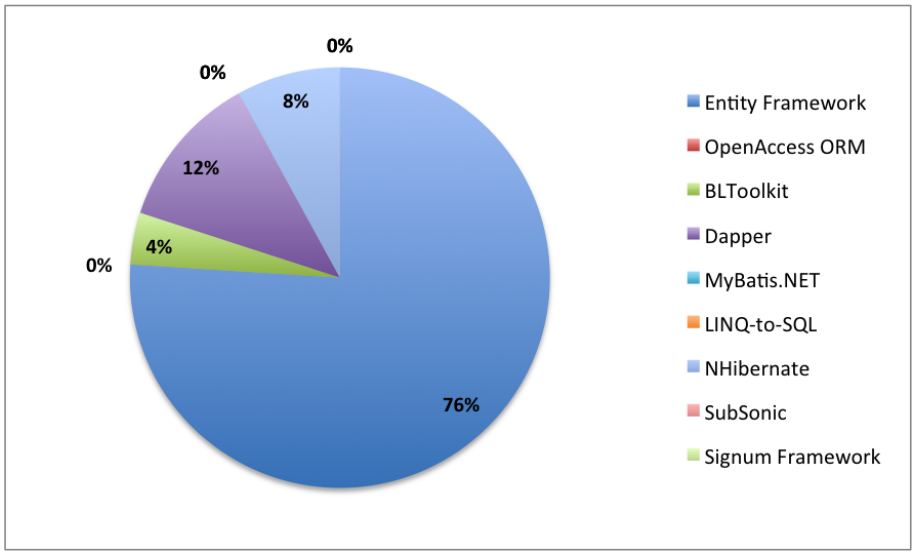

Abbildung 3.5: Umfrageresultate Für ein privates Projekt, welchen ORM würdest du einsetzen?

3.2.4 Resultat

Das Resultat der Umfrage war wenig überraschend. Die weite Verbreitung und der gute Ruf des Entity Frameworks war aufgrund des Marketings und der ausführlichen und qualitativ hochwertigen Dokumentation von Microsoft vorauszusehen. Aus den Umfrageergebnissen wurden die drei meistgenannten Produkte identifiziert, welche für die weiteren Schritte nun genauer betrachtet werden. Namentlich sind dies:

- Entity Framework
- Dapper
- NHibernate

Diese drei Produkte aus der Marktübersicht werden in den nächsten Kapitel genauer betrachtet und sie stehen für die Empfehlung des OR-Mappers pro Enterprise Pattern zur Auswahl.

3.3 Vergleich ausgewählter Produkte

Die Marktübersicht dient als Basis für weitergehende Vergleich der OR-Mapper. Da ein Ziel dieser Arbeit die Erstellung einer Produktempfehlung pro Enterprise Pattern ist, werden die drei in der Umfange (siehe Kapitel 3.2.1 auf Seite 50) ausgewählten Produkte Entity Framework, Dapper und NHibernate auf Grund der folgenden Kriterien miteinander verglichen:

- Benutzerfreundlichkeit
- Plattformunterstützung
- Performance
- Toolunterstützung
- Unterstützung Enterprise Pattern
- Monitoring

Im Gegensatz zur Marktübersicht setzten alle Vergleiche voraus, dass im Umgang mit den betrachteten OR-Mappern Erfahrung gesammelt wurde. Diese Erfahrung wird mittels der Implementierung eines Persistenzlayers innerhalb der serviceorientierten CRM-Applikation gewonnen (siehe Kapitel 4.3 auf Seite 82).

Da der objektive Vergleich nicht für alle Vergleichskriterien möglich ist wird besonderen Wert auf die Erläuterung der Auswahl des jeweils führenden Produktes gelegt.

3.3.1 Benutzerfreundlichkeit

Mit Benutzerfreundlichkeit ist der Bedienungskomfort für einen Entwickler gemeint. Dazu gehören die Einfachheit der Installation, die Unterstützung verschiedener Mappingtechniken, der Aufwand zur Erstellung des Mappings sowie das Vorhandensein von Designern oder Assistenten.

Die Bewertung der Benutzerfreundlichkeit hängt vom Geschmack und der Präferenz jedes einzelnen Entwicklers ab und das Vergleichsresultat in Tabelle 3.1 kann für einen anderen Entwickler durchaus unterschiedliche Werte enthalten. Der Autor präferiert zum Beispiel flüssige Schnittstellen[12] da man als Entwickler bereits vom Compiler gewarnt wird, falls man sich zum Beispiel beim Namen einer Eigenschaft eines Domänenobjekts verschrieben hat. Andere Entwickler bevorzugen vielleicht XML-basierte Mappings, da sie

[12] Flüssige Schnittstellen (in englisch Fluent Interfaces) bieten eine lesbarere Implementierung einer API. Mehr dazu unter `http://www.hanselman.com/blog/TheWeeklySourceCode14FluentInterfaceEdition.aspx`.

die baumartige Struktur von XML bevorzugen. Wiederum andere Entwickler werden das Vorhandensein eines Mapping-Designers schätzen, da das Mapping grafisch entwickelt werden kann. Aus diesem Grund sind gewisse Kriterien in dieser Kategorie stark von persönlichen Vorlieben abhängig und die Wertung derselben nicht für jeden Entwickler gleich.

Kriterium	Entity Framework	Dapper	NHibernate
Flüssige Schnittstellen	Nein	Ja	Nein[13]
Forward Mapping (Database-First)	Ja	Nein	Ja
Reverse Mapping (Model-First)	Ja	Ja	Ja
Migrationen	Ja	Nein	Ja
Modeldesigner[14]	Ja	Nein	Nein[15]
Assistentenunterstützung	Ja	Nein	Nein
XML-basiertes Mapping/Mapping-Datei	Nein	Nein	Ja
Codebasiertes Mapping über Annotationen	Ja	Ja	Nein
Monitoring[16]	Ja	Nein	Ja
Unterstützung von Hersteller	Sehr viel	Wenig	Mittel
Inhalt Community-Webseiten (Blogs, CodeProject.com, ...)	Sehr viel	Mittel	Viel
StackOverflow Beiträge	~25000	~400	~18000

Tabelle 3.1: Kriterien Benutzerfreundlichkeit

NHibernate hat den grossen Vorteil dass viele für Hibernate entwickelte Drittanbietererweiterungen auch mit NHibernate funktionieren und es eine grosse Entwicklercommunity in Onlineforen wie StackOverflow gibt.

Der klare Sieger dieser Kategorie ist jedoch das Entity Framework. Einerseits weil es Forward- und Reverse-Mapping beherrscht und andererseits weil es einen mächtigen, in Microsofts Visual Studio integrierten Designer mitliefert. Auch ist die MSDN als erste Anlaufstelle für Dokumentation und Tutorials oder auch der Erfahrungsschatz der Community auf Seiten wie StackOverflow ein gewichtiges Argument für die Wahl des Entity Frameworks als OR-Mapper.

3.3.2 Plattformunterstützung

Das Kriterium Plattformunterstützung beschreibt die Unterstützung von Entwicklungsframeworks (.NET Framework, Java, usw.) sowie die Unterstützung von Datenbankma-

[13] Die Erweiterung Fluent NHibernate (siehe `http://www.fluentnhibernate.org`) erlaubt die Verwendung von flüssigen Schnittstellen beim Erstellen der Mappings.
[14] Wird später (siehe Kapitel 3.3.4 auf Seite 60) noch im Detail betrachtet.
[15] Das kommerzielle Produkt NHibernate Designer (siehe `http://www.mindscapehq.com/products/nhdesigner`) bietet einen Modeldesigner.
[16] Wird in einem späteren Vergleich (siehe Kapitel 3.3.6 auf Seite 61) eingehender betrachtet.

nagementsystemen der betrachteten OR-Mapper. Die Werte für die einzelnen Kriterien (siehe Tabelle 3.2) stammen einerseits von den Produktewebseiten der Hersteller und wurden mittels eigens implementierten Proof of Concepts verifiziert.

Das Entity Framework und Dapper benutzen als Basis das .NET Framework 3.5 da sie LINQ und Lamda-Expressions verwenden, welche mit dieser Version des .NET Frameworks eingeführt wurden. NHibernate ist hingegen schon seit dem .NET Framework 1.1 verfügbar da es deutlich weniger moderne C# Sprachelemente verwendet als das Entity Framework oder Dapper. Dies, da es sich bei NHibernate um eine Portierung aus dem in Java geschriebenen Hibernate handelt. Beim Betrachten des Sourcecodes[17] ist es sehr viel schwieriger sich als Entwickler zurecht zu finden als zum Beispiel innerhalb von Dappers Sourcecode[18]. Auch kann die Kernbibliothek von NHibernate nicht direkt mit LINQ von Microsoft umgehen sondern hat einen eigenen LINQ-Provider implementiert der nicht in allen Belangen zu Microsofts Implementierung kompatibel ist.

NHibernate hat den grossen Vorteil dass es eine Portierung des populären Open Source Projekts Hibernate ist, welches in Java implementiert wurde. Dadurch sind sämtliche Mapping- und Konfigurationsdateien, welche in NHibernate erstellt werden, mit dem Java-basierten Hibernate kompatibel und wiederverwendbar. Ausserdem unterstützt es das .NET Framework seit der Version 1.1, was einen Vorteil im Umgang mit Legacy-Projekten darstellt.

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 Java Virtual Machine	Nein	Nein	Ja[19]
Unterstützt weitere Entwicklungsframeworks	Nein	Nein	Nein
Unterstützt Microsoft SQL Server	Ja	Ja	Ja
Unterstützt Oracle Database Server	Ja	Ja	Ja
Unterstützt weitere Datenbankmanagementsysteme	Ja[20]	Ja[21]	Ja[22]

Tabelle 3.2: Kriterien Plattformunterstützung

[17] Der Sourcecode von NHibernate ist unter `https://github.com/nhibernate/nhibernate-core` zu finden.

[18] Siehe `https://github.com/SamSaffron/dapper-dot-net`.

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

3.3.3 Performance

Die Performance der einzelnen OR-Mapper ist ein wichtiges Auswahlkriterium bei der Evaluation von OR-Mappern für den Einsatz in einer Softwareapplikation. Werden während der Entwicklungsphase keine Lasttests auf dem Persistenzlayer oder dem Gesamtsystem durchgeführt, können beim produktiven Einsatz des System unangenehme Überraschungen in Bezug auf die Performance des verwendeten OR-Mappers auftreten.

Das in diesem Kapitel behandelte Kriterium vergleicht die drei OR-Mapper indem drei typische Anwendungsfälle in der Benutzung eines OR-Mappers innerhalb einer Messumgebung ausgeführt werden. Die Testumgebung beinhaltet eine Integration der drei verglichenen OR-Mapper. Um Messwerte zu erhalten werden jeweils fünf Durchgänge mit 1, 10, 100, 500 oder 1000 Ausführungen desselben Queries ausgeführt und gemittelt. Die mehrfache Ausführung desselben Queries dient dazu Effekte wie die vergleichsweise lange Aufstartzeit beim ersten Ausführen eines Queries der kompletten OR-Mapper[23] zu minimieren und die Messumgebung einer produktiven Umgebung anzugleichen. Die Queries sind angelehnt an [Roc11] und [Koc11].

Als Testrechner wurde ein 2.6 GHz Intel Core i7 mit 16 GB DDR3 RAM und einer 500 GB grossen SSD-Festplatte verwendet. Als Betriebssystem wurde ein frisch installiertes Windows 7 Professional verwendet. Die Messapplikation wurde im .NET Framework 4.5 geschrieben und wird als optimierter Releasebuild kompiliert und ausgeführt. Das Datenbankmanagementsystem ist ein Microsoft SQL Server 2012 Standard Edition welches auf der selben Maschine installiert ist, auf welcher die Messapplikation ausgeführt wird.

Zur besseren Illustration und Verständlichkeit sind nachfolgend die drei Testqueries in SQL Syntax erklärt. Die Erwartung an den OR-Mapper ist es, die folgenden SQL-Queries in der abgebildeten oder einer ähnlichen Form an das Datenbankmanagementsystem abzusetzen.

Das erste Query ist eine Anfrage auf eine Tabelle mit einem Filter auf den Primärschlüssel der Tabelle. Dabei wird jeweils ein einzelner Datensatz aus der Tabelle der Datenbank gelesen, auf ein Domänenobjekt gemappt und der Messapplikation zurückgegeben. Es wird die Zeit zwischen dem Absetzen der Anfrage und dem Erhalten des initialisierten Domänenobjekts gemessen.

[19] NHibernate ist eine Portierung von Hibernate, einem Java OR-Mapper (siehe `http://www.hibernate.org`).
[20] Unterstützt auch SQLCE und weitere DBMS via Drittkomponentenanbieter.
[21] Unterstützt auch SQLCE, Firebird und MySQL.
[22] Unterstützt auch Microsoft Access, Firebird, DB2, UDB, MySQL und SQLLite.
[23] Mehr zur Aufstartzeit am Beispiel von Entity Framework: `http://neverindoubtnet.blogspot.pt/2012/03/squash-entity-framework-startup-time.html`.

Algorithmus 3.1 Abfrage auf eine Tabelle

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

Das zweite Query ist eine Anfrage auf mehrere Tabellen, bei welcher eine Anzahl von Datensätzen gelesen wird. Dies wird unter Mithilfe von Verknüpfungen über Fremdschlüssel und einem Filter ermöglicht. Wiederum wird die Zeit gemessen vom Absetzen der Anfrage an den OR-Mapper bis zum Zeitpunkt der Rückgabe der Liste der gemappten und initialisierten Domänenobjekte.

Algorithmus 3.2 Abfrage auf mehrere Tabellen

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

Das dritte Query ist keine Abfrage von bereits persistierten Daten wie die beiden anderen Queries sondern es sind drei verschiedene Queries welche zuerst einen neuen Datensatz in eine Tabelle einfügen, diesen danach verändern und am Schluss wieder aus der Tabelle entfernen. Dadurch wird geprüft wie effizient das Mapping der OR-Mapper im Zusammenhang mit datenverändernden Abfragen ist. Es wird die Zeit gemessen zwischen dem Übergeben des neu zu persistierenden Domänenobjekts und der Bestätigung der Löschung des OR-Mappers.

Algorithmus 3.3 Einfügen, Aktualisieren und Löschen

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

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

Tabelle 3.3: Kriterien Performance

In diesem Vergleich (siehe Resultate in Tabelle 3.3) zeigt sich sehr stark der Unterschied zwischen den beiden OR-Mapper Typen, den kompletten und den leichtgewichtigen OR-Mappern. Dapper als leichtgewichtiger OR-Mapper hat diesen Vergleich klar gewonnen und sämtliche Metriken dominiert. Dies liegt insbesondere am einfachen internen Aufbau des ORMs und dessen vergleichsweise simplen Mappingalgorithmen. Das Entity Framework als der featurestärkste OR-Mapper der drei verglichenen Produkte hat klar die schlechteste aufgewiesene Performance.

3.3.4 Toolunterstützung

Das Entity Framework bietet eine vielfältige Toolunterstützung. Wenn ein Forward Mapping Ansatz gewählt wird bietet das Entity Framework einen Designer, welcher es erlaubt Entitäten mit ihren Attributen und Beziehungen grafisch zu modellieren. Ausserdem werden verschiedene Assistenten mitgeliefert um bestehende Datenbankschemata bei Reverse Mapping Szenarien zu importieren und als Entity Framework Modell zu generieren. Möchte der Entwickler ohne Designer direkt POCOs erstellen, so liefert das Entity Framework eine Erweiterung für den NuGet Paketmanager[24] mit, welcher dem Entwickler Hand bietet zum Beispiel beim Erstellen von Datenmigrationen oder neuen Konfigurationen.

Dapper als leichtgewichtiger OR-Mapper bietet keinerlei Toolunterstützung an. Sämtliche Konfiguration des ORMs geschieht über direkt in Code geschriebene Mappings welche vom Entwickler vollständig von Hand erstellt werden müssen. Die Entitäten selber

[24]Mehr zu NuGet unter `http://nuget.org`.

können jedoch auch über den Visual Studio Klassendesigner erstellt werden.

NHibernate als herunterladbare Klassenbibliothek kommt ohne zusätzliche Tools mit. Die Konfiguration der Mappings geschieht mittels XML-Dateien und die Entitäten selbst können mit dem Visual Studio Klassendesigner erstellt werden. NHibernate liefert die XML Schema Definitionen für die XML-Dateien mit, welche dann auch mit Intellisense im Visual Studio verwendet werden können[25]. Es gibt zahlreiche Fremdkomponentenhersteller welche Tools für NHibernate entwickeln, so zum Beispiel der NHibernate Designer von Mindscape HQ[26] oder Visual NHibernate von Slyce[27]. Diese Tools sind durchaus mächtig und für grössere Projekte zu empfehlen, werden jedoch in dieser Bachelorarbeit nicht weiter in Betracht gezogen da sie meist kostenpflichtig und nicht Teil des herunterladbaren Pakets von NHibernate 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 gewohnten Komfort von Microsoft-Produkten wie dem Visual Studio. Würde man die erwähnten Drittanbietertools für NHibernate auch betrachten, würde die Entscheidung wohl nicht so klar ausfallen.

3.3.5 Unterstützung Enterprise Pattern

Martin Fowlers Enterprise Pattern lassen sich mit jedem OR-Mapper implementieren. Die aktuellen Versionen von Entity Framework, NHibernate und Dapper zwingen den Entwickler nicht dazu ein bestimmtes Enterprise Pattern zu wählen. Das Entity Framework und NHibernate besitzen eine Möglichkeit, das Pattern Unit of Work[28] vereinfacht zu nutzen beziehungsweise zu implementieren, dieses ist jedoch nicht Gegenstand der betrachteten Enterprise Pattern in dieser Bachelorarbeit. Dapper hingegen ist als Erweiterung für IDbConnection am flexibelsten für jegliches Enterprise Pattern einzusetzen da der Zugriff auf die Funktionalitäten des OR-Mappers sehr transparent und einfach ist.

Im Endeffekt obliegt es dem Entwickler oder dem Softwarearchitekten wie der Persistenzlayer eines Systems implementiert werden soll und welche Pattern eingesetzt werden sollen. Die Entscheidung ist zu abhängig vom Anwendungszweck des Systems beziehungsweise den Anforderungen an dasselbe als dass im Rahmen dieses Vergleichs von ORMs eine abschliessende Wertung erstellt werden könnte.

Aus diesem Grund ist diese Kategorie nicht bewertbar und es wird keines der betrachteten Produkte über die anderen Produkte favorisiert.

3.3.6 Monitoring

Die Möglichkeiten zur Überwachung eines Persistenzlayers und des innerhalb verwendeten OR-Mappers wird bei der Evaluation von ORMs oft ausser Acht gelassen. Dies, da das Monitoring häufig erst spät in der Entwicklung eines Systems beziehungsweise erst

[25] Siehe `http://stackoverflow.com/questions/3565156/how-to-add-nhibernate-xml-schema-to-visual-studio`.
[26] Siehe `http://www.mindscapehq.com/products/nhdesigner`.
[27] siehe `http://www.slyce.com/VisualNHibernate`.
[28] Mehr zum Pattern siehe `http://martinfowler.com/eaaCatalog/unitOfWork.html`

im Betrieb und 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, Flaschenhälse zu erkennen und auch zu beheben.

Natürlich lassen sich alle ORMs über einen SQL Profiler auf der Ebene des Datenbankmanagementsystems überwachen. Doch meist ist es bei dieser Art von Überwachung schwierig zu sagen, welches die Ursachen der beobachteten Performanceprobleme sind. Die Rückverfolgung von auf der Datenbank beobachteten Queries zum Mapping eines Domänenobjekts innerhalb des ORMs ist schwierig und oft sogar unmöglich.

Das Entity Framework bietet einzig Intellitrace Debugging[29]. Da für das Entity Framework die Debuggingsymbole verfügbar sind, ist das Debuggen des in der Applikation integrierten EF ohne weiteres möglich. Die Architektur des Entity Frameworks ist auf Grund seiner Vielzahl von Funktionalitäten sehr komplex. Daher ist es für einen Entwickler nicht ganz einfach Mappings schlüssig nachzuvollziehen. Es existieren eine grosse Anzahl an Drittherstellertools wie der Entity Framework Profiler von Hibernating Rhinos[30] oder der EF Provider Wrapper in der Visual Studio Gallery[31] welche bei Performanceengpässen in produktiven Umgebungen zum Einsatz kommen können und dem Entwickler teilweise grafisch aufzeigen, welche Operationen innerhalb des Entity Frameworks wie viel Performance kosten.

Da Dapper in Sourcecodeform vorliegt ist das Debugging für den Entwickler denkbar einfach gestaltet. Der Entwickler kann auch eigene Performancezähler und Messverfahren implementieren und so zum Beispiel selber ein Tracing aller SQL-Statements realisieren. Zusätzlich bietet Dapper die Möglichkeit an bestimmten Stellen Beobachterobjekte einzufügen, welche Operationen von Dapper überwachen können. Der Nachteil ist allerdings, dass sämtliche Überwachungslogik selbst entwickelt werden muss. Dapper liefert keinerlei vorgefertigte Überwachungsmöglichkeiten mit.

NHibernate ist ein Open Source Projekt und aus diesem Grund kann dieses für Debuggingzwecke selbst kompiliert und gedebuggt werden. Auf Grund des Ursprungs von NHibernate als 1-zu-1 Portierung von Hibernate in Java nach C# ist die Lesbarkeit und Verständlichkeit für einen .NET Entwickler nicht immer gegeben. Auch ist das Kompilieren von NHibernate mit erheblichem Konfigurationsaufwand verbunden und nicht ohne vertieftes Studium der Dokumentation und einigem Ausprobieren von verschiedenen Konfigurationen durchführbar. Für NHibernate existiert wie auch für das Entity Framework ein Profilingtool von Hibernating Rhinos, der NHibernate Profiler[32]. Der NHibernate Profiler liefert auf Grund des Datencontainers von NHibernate Statistiken zum Zustand des ORMs und kann auch mit dem Sourcecode der Applikation verknüpft werden. NHibernate bietet auch die Möglichkeit eigene Interceptoren[33] zu schreiben und in NHibernate zu injizieren. So kann zum Beispiel jedes auf das Datenbankmanagement-

[29] Intellitrace Debugging bietet einen gesamtheitlichen Überblick über den Zustand einer laufenden Applikation im Gegensatz zu traditionellen Debuggern welche einen isolierten Zustand betrachten.
[30] Siehe `http://www.hibernatingrhinos.com/products/EFProf`
[31] Mehr dazu unter `http://code.msdn.microsoft.com/EFProviderWrappers`
[32] Siehe `http://www.hibernatingrhinos.com/products/nhprof`.
[33] Mehr zum Interceptor Pattern siehe `http://bosy.dailydev.org/2007/04/interceptor-design-pattern.html`.

system abgesetzte SQL-Statement mit einem Interceptor abgefangen und in ein Tracefile geschrieben werden. Auch lässt sich eine log4net[34] Konfiguration erstellen, welche erweiterte Tracemeldungen von NHibernate in den gewünschten Ausgabekanal schreibt.

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 und Drittanbieter Montoringsuiten zu entwickeln und auf die eigenen Bedürfnisse zuzuschneiden. Dieser Vergleich bezieht sich deswegen auf die Möglichkeit zur Implementierung von Überwachungsmechanismen und bezieht auch auf die verfügbaren Monitoringtools von Drittherstellern in das Vergleichsresultat ein. Mit dieser gesamtheitlichen Betrachtungsweise ist das Entity Framework der knappe Gewinner vor NHibernate dieses Vergleichs.

3.3.7 Vergleichsmatrix

In Tabelle 3.4 sind noch einmal alle Vergleichskriterien der oberen Kapitel mit dem jeweils führenden OR-Mapper zusammengefasst. Diese Tabelle liefert einen tieferen Einblick in die Verwendung der OR-Mapper aus Sicht des Entwicklers und stellt die Grundlagen für die Evaluation der OR-Mapper (siehe Kapitel 4.1 auf Seite 77) und die Produktempfehlung (siehe Kapitel 4.2 auf Seite 80) zur Verfügung.

Wie schon in den vorherigen Kapitel ausgeführt ist diese Vergleichsmatrix teilweise subjektiv beeinflusst und könnte je nach Betrachter unterschiedlich ausfallen.

Kriterium	Gewinner
Benutzerfreundlichkeit	Entity Framework
Plattformunterstützung	NHibernate
Performance	Dapper
Toolunterstützung	Entity Framework
Unterstützung Enterprise Pattern	-
Monitoring	Entity Framework

Tabelle 3.4: Vergleichsmatrix

3.4 Architektur serviceorientierte CRM-Applikation

Dieses Kapitel enthält das Konzept und die Architektur für die Implementierung des Proof of Concepts im Rahmen dieser Bachelorarbeit. Das zu realisierende System wird aus den drei Architektursichten Bausteinsicht, Laufzeitsicht und Verteilungssicht betrachtet und spezifiziert.

Das Ziel ist es auf Grund der dokumentierten Anforderungen (siehe Kapitel 2.4) ein vollständiges und umfassendes Lösungskonzept zu entwickeln, welches in der Implemen-

[34] Mehr zu log4net siehe `http://logging.apache.org/log4net/`.

tierungsphase des Wasserfallmodells von Softwareentwicklern umgesetzt werden kann (nach [OSKZ06]).

3.4.1 Bausteinsicht

Nach [Sta11] und [HS13] 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 Systems und entsprechen in dieser Hinsicht den konventionellen Implementierungsmodellen.

3.4.1.1 Komponentendiagramm

Das Komponentendiagramm in Abbildung 3.6 stellt das System SoCrm[35] aus der Vogelperspektive dar und ist die höchstabstrahierte Ansicht der Bausteinsicht, die in diesem Projekt verfügbar ist.

Es wird eine grobe Schichtung erstellt indem zwischen dem Presentation-, dem Orchestration- und dem Infrastructurelayer unterschieden wird. Die Schichtung wird erreicht indem alle drei Layer als eigens ausführ- und verteilbare Komponenten realisiert werden.

Der Infrastructurelayer beinhaltet nur den Persistenzlayer und wird dreifach implementiert (eine Implementierung mit Entity Framework, eine mit Dapper und eine mit NHibernate). Der Infrastructurelayer offeriert über WCF die Schnittstelle IPersistenceService<TDomainObject>, welche von den Orchestrationservices Security Service, Customer Service, Contact Service und Logging Service konsumiert wird und worauf diese ihre Geschäftslogik aufbauen.

Die Orchestrationservices beinhalten sämtliche Geschäftslogik des Systems und offerieren wiederum jeweils eine WCF Schnittstelle, auf welcher die Serviceoperationen für den Presentationlayer definiert sind.

Der Presentationlayer besteht aus einer ausführbaren App sowie vier Presentationmodulen, welche die Präsentationslogik enthalten: Security Presentation, Customer Presentation, Contact Presentation und Logging Presentation. Diese Presentationmodule greifen auf die Schnittstellen des Orchestrationlayers zu und visualisieren die so gelesenen Daten beziehungsweise geben vom Benutzer eingegebene Daten weiter an den Orchestrationlayer zur späteren Persistierung in der Datenbank.

[35] SoCrm steht für Service Oriented CRM und ist der Projektname der zu realisierenden serviceorientierten CRM-Applikation.

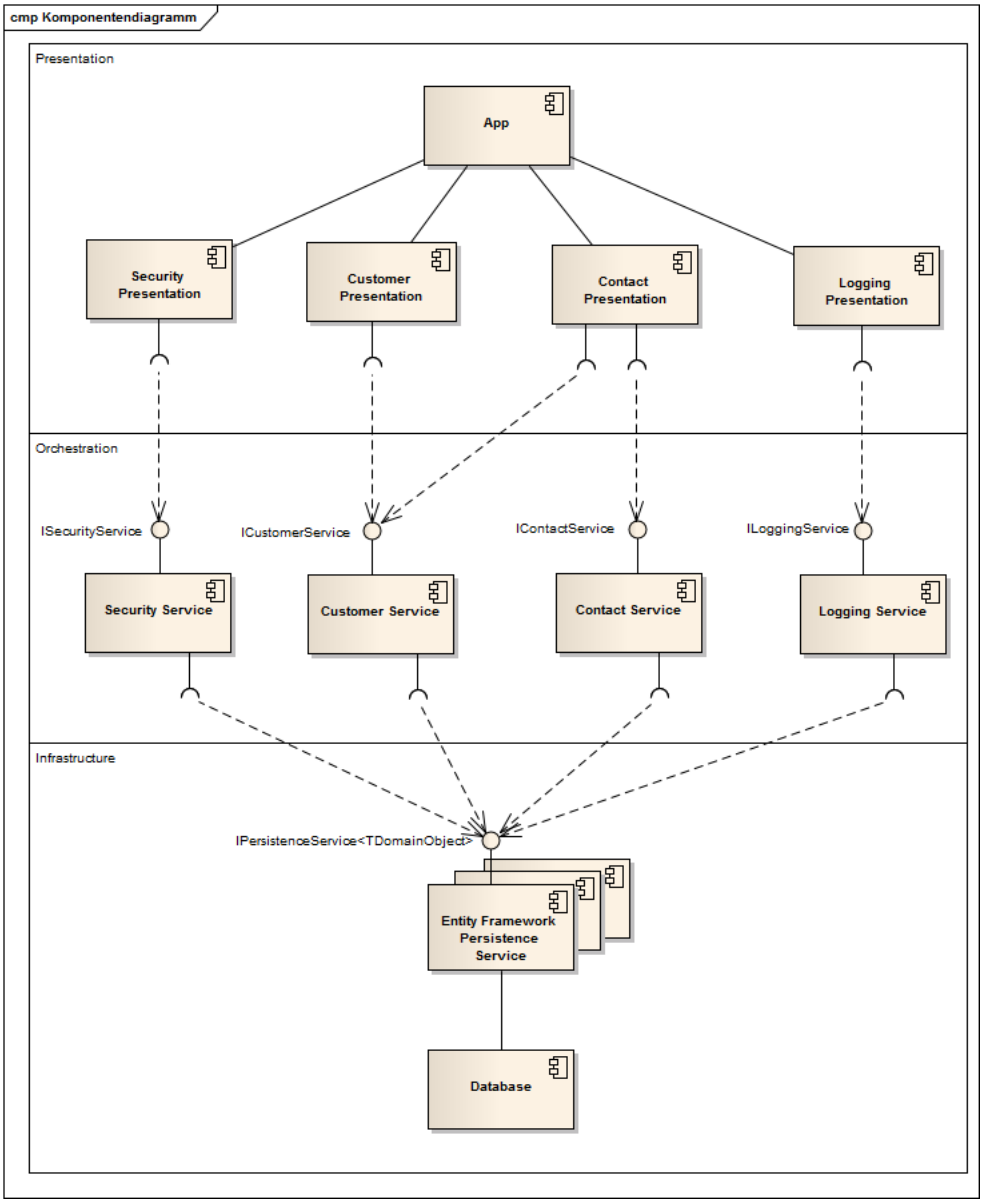

Abbildung 3.6: Komponentendiagramm SoCrm

3.4.1.2 Domänenmodell

Das Domänenmodell (siehe Abbildung 3.7) umfasst nur die Klassen der Objekte, die über die Serviceschnittstelle von WCF zwischen dem Persistenzlayer und dem Orchestrationlayer sowie zwischen dem Orchestrationlayer und dem Presentationlayer übertragen werden. Die Domänenobjekte welche im Domänenmodell dargestellt sind, stellen das Domain Model des Systems SoCrm dar.

Alle Domänenobjekte (auch Entitäten genannt) implementieren die Schnittstelle IDo-

mainObject. Diese Schnittstelle garantiert, dass jedes Domänenobjekt einen eindeutigen Identifikator ObjectId in Form einer Guid besitzt. Die Basisklasse aller Domänenobjekte, die Klasse DomainObject, implementiert dieses Interface.

Die wichtigsten Domänenobjekte werden nachfolgend kurz erklärt: Die Klasse Person steht für einen einzelnen Kunden. Der Kunde kann eine Firma (Klasse Company) als Arbeitgeber besitzen. Jede Person und jede Firma beinhalten eine Adresse (Klasse Address), welche als Komposition modelliert ist. Zusätzlich kann jede Person mehrere E-Mail-Adressen (Klasse EMailAddress) und Telefonnummern (Klasse PhoneNumber) als Kontaktmöglichkeiten haben.

Für eine Person können Kontakte (Klasse Contact) erfasst werden, welche neben der Person auch eine Beziehung zum erfassenden Benutzer (Klasse User) besitzen. Der User dient gleichzeitig auch zur Authentifizierung am System.

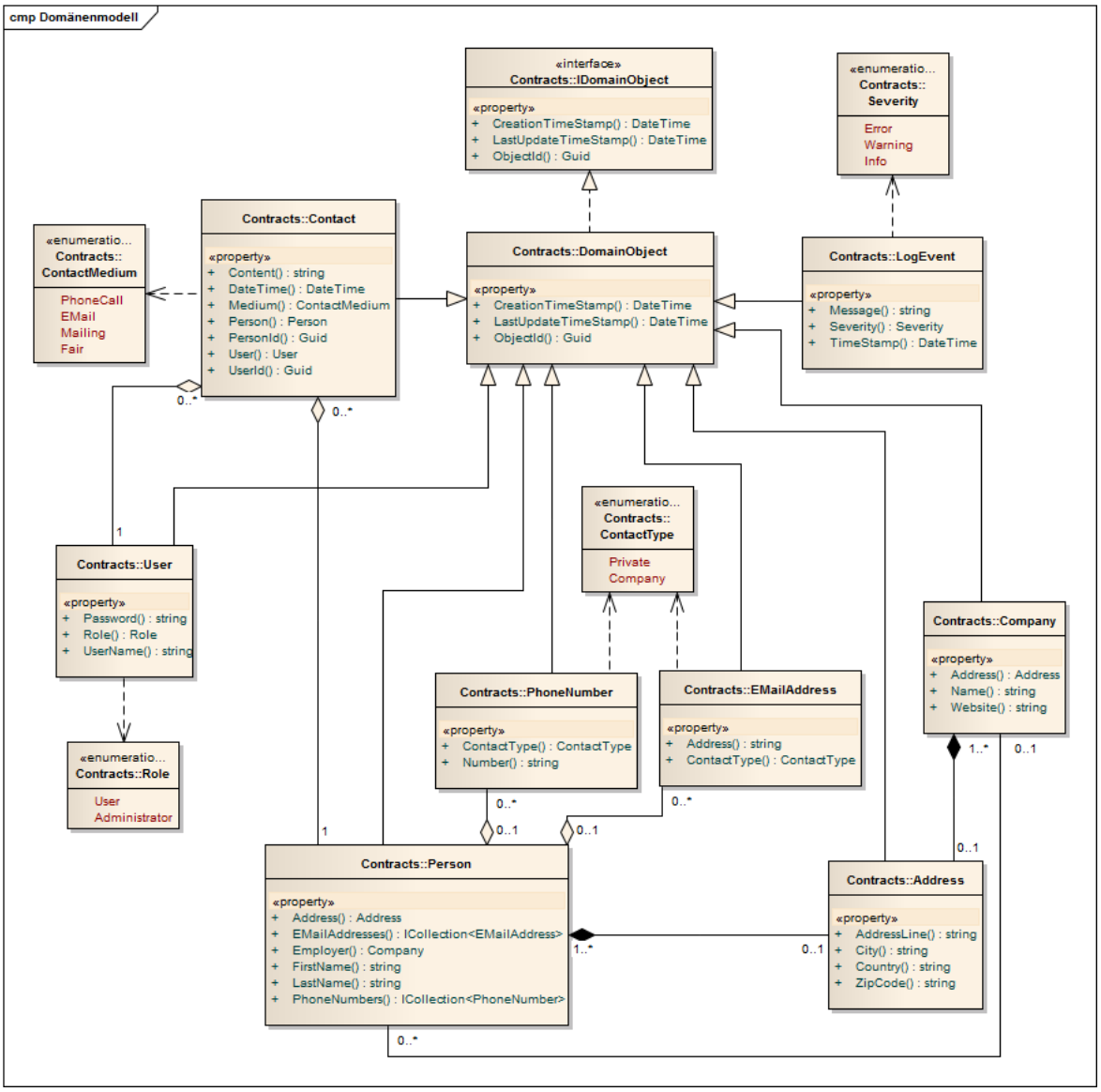

Abbildung 3.7: Klassendiagramm Domänenmodell SoCrm

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, vorhergegangenen Anfragen behandelt und es werden auch keine Sitzungsinformationen ausgetauscht.

Nachfolgend werden die Service Contracts des Systems SoCrm beschrieben und deren Operationen vorgestellt. Die Service Contracts umfassen die Schnittstellen des Orchestrationlayers des Systems (siehe Kapitel 3.4.1.1 auf Seite 64).

ISecurityService Über die Schnittstelle ISecurityService werden Informationen über die Benutzer des Systems SoCrm ausgetauscht. Mit Hilfe der Methode CreateUser können neue Benutzer im System erfasst werden, welche mit der Methode DeleteUser auch wieder gelöscht werden können. Die verschiedenen Get-Methoden ohne Übergabeparameter (siehe Tabelle 3.5) liefern Rollen oder Benutzer zurück. Es sind auch verschiedene Get-Methoden mit vordefinierten Filtern auf der Schnittstelle definiert, welche vom UI für verschiedene Views benötigt werden. Die Methode SetPassword ermöglicht es das Passwort für einen bestimmten Benutzer neu zu setzen. Dazu wird jeweils auch das alte Passwort mit übergeben und validiert. Die ValidateCredentials Methode übernimmt die Aufgabe der Validierung von Benutzerdaten und liefert einen boolschen Wert zurück.

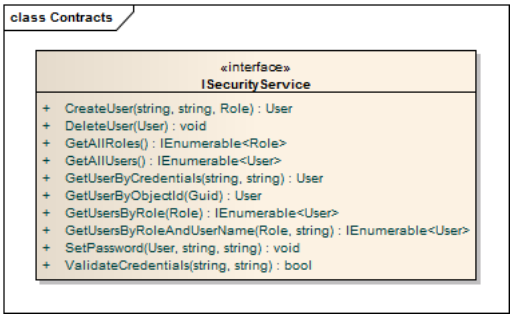

Abbildung 3.8: Klassendiagramm ISecurityService

Methode	Aufgabe
CreateUser	Erstellt einen neuen Benutzer mit den übergebenen Werten
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 zurück
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 zurück
SetPassword	Setzt das Password für einen bestimmten Benutzer
ValidateCredentials	Überprüft die übergebenen Benutzerinformationen und liefert wahr oder falsch zurück

Tabelle 3.5: Methoden von ISecurityService

ICustomerService Die Schnittstelle ICustomerService wird vom Orchestrationlayer offeriert um Kundendaten zur Verfügung zu stellen. Die vier Create-Methoden erstellen entweder eine neue Firma, eine neue E-Mail-Adresse, einen neuen Kunden oder eine neue Telefonnummer. Es sind auch die dazugehörigen Delete-Methoden auf der Schnittstelle definiert. Die GetAll-Methoden liefern wie der Name schon sagt jeweils alle Entitäten des jeweiligen Typs als IEnumerable<T> zurück. Zusätzlich sind mehrere parameterisierte Get-Methoden definiert, welche dem UI für Spezialfälle gefilterte Listen zurückgeben.

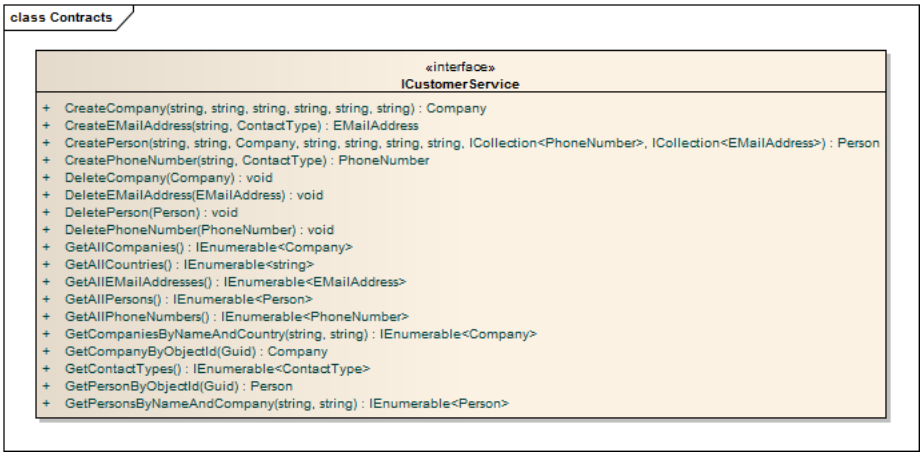

Abbildung 3.9: Klassendiagramm ICustomerService

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
GetCompaniesByName AndCountry	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
GetPersonsByName AndCompany	Gibt Kunden auf Grund ihres Namens und Arbeitgebers zurück

Tabelle 3.6: Methoden von ICustomerService

IContactService Die vom Contact Service im Orchestrationlayer offerierte Schnittstelle IContactService liefert Informationen zu den Kontakten eines Kunden. Die Methode CreateContact erstellt einen neuen Kontakt. Die Methode DeleteContact ermöglicht es einen Kontakt zu löschen. Über einen Aufruf an GetAllContactMediums werden alle möglichen Kontaktmedien zurückgegeben. Die Get-Methoden geben gefilterte Listen von Kontakten zurück auf Basis der übergebenen Parameter.

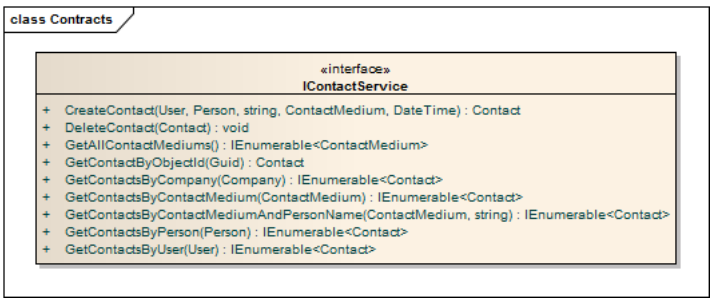

Abbildung 3.10: Klassendiagramm IContactService

Methode	Aufgabe
CreateContact	Erstellt einen Kontakt für einen Kunden mit den übergebenen Werten
DeleteContact	Löscht einen Kontakt auf Grund seiner ObjectId
GetAllContactMediums	Gibt alle Kontaktmedien zurück
GetContactByObjectId	Gibt einen Kontakt auf Grund seiner ObjectId zurück
GetContactsByCompany	Gibt Kontakte auf Grund des Arbeitgebers des Kunden zurück
GetContactsByContactMedium	Gibt Kontakte auf Grund des Kontaktmediums zurück
GetContactsByContactMedium AndPersonName	Gibt Kontakte auf Grund des Kontaktmediums und des Kundennamen zurück
GetContactsByPerson	Gibt Kontakte auf Grund des Kunden zurück
GetContactsByUser	Gibt Kontakte auf Grund des Benutzers zurück

Tabelle 3.7: Methoden von IContactService

ILoggingService Die Schnittstelle ILoggingService wird innerhalb des Orchestrationlayers vom Logging Service implementiert und erlaubt es der Benutzeroberfläche oder anderen Services Logeinträge zu persistieren beziehungsweise diese auszulesen. Die GetAll-Methoden geben jeweils ungefiltert die gesamte Liste von vorhandenen Logeinträgen bzw. den möglichen Schweregraden zurück. Die beiden Get-Methoden liefern gefilterte Listen zurück. Über den Aufruf der Methode LogEvent können neue Logeinträge dem System hinzugefügt werden.

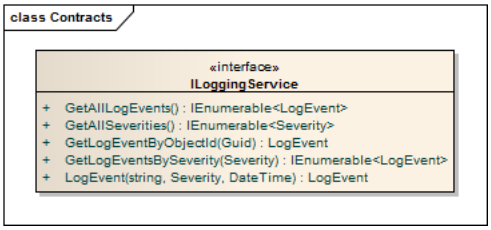

Abbildung 3.11: Klassendiagramm ILoggingService

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

Tabelle 3.8: Methoden von ILoggingService

3.4.2 Laufzeitsicht

Die Laufzeitsicht beschreibt, welche Bestandteile des Systems zur Laufzeit existieren und wie diese zusammenwirken (nach [Sta11]). Dabei kommen wichtige Aspekte des Systembetriebs ins Spiel, welche beispielsweise den Systemstart, die Laufzeitkonfiguration oder die Administration des Systems betreffen. Darüber hinaus dokumentiert die Laufzeitsicht wie Laufzeitkomponenten sich aus Instanzen von Implementierungsbausteinen zusammensetzen.

Dieses Kapitel enthält stellvertretend für alle Use-Cases den Use-Case "Kontakt hinzufügen" (siehe Kapitel 2.4.2.9 auf Seite 32) modelliert als Sequenzdiagramm. Es wird nur dieser Use-Case aus der Laufzeitsicht betrachtet und steht exemplarisch für alle anderen Use-Cases.

3.4.2.1 Kontakt hinzufügen

Der Benutzer ist bereits beim System authentifiziert und besitzt eine gültige Authentifizierung. Möchte ein Benutzer einem Kunden einen neuen Kontakt hinzufügen, so navigiert er zuerst auf die CustomerListView, welche alle Kunden anzeigt. Die CustomerListView erhält ihre Daten aus dem CustomerService welcher wiederum den PersistenceService<Person> aufruft, der die Kundendaten aus der Datenbank liest und mittels ORM auf Domänenobjekte vom Typ Person mappt.

In der CustomerListView wählt der Benutzer den Kunden aus, für welchen er einen neuen Kontakt erstellen möchte. Bei der Auswahl des Kunden navigiert das System zur CreateContactView welche die notwendigen Felder für das Erstellen des Kontaktes bereitstellt. Die CreateContactView enthält auch eine Auswahl der möglichen Kontaktmedien, welche vom ContactService zurückgegeben werden. Die vollständigen Kontaktinformationen werden an den ContactService übergeben, welcher diese über den PersistenceService<Contact> in die Datenbank speichert. Das Resultat der Datenbankoperation wird dem Benutzer als Bestätigung angezeigt.

Um das Sequenzdiagramm im Abbildung 3.13 lesbarer zu gestalten wird auf einzelne Elemente im Ablauf des Use-Cases verzichtet beziehungsweise diese werden vereinfacht

dargestellt. So fehlen zum Beispiel die Controller und ViewModels und die Persistenzservices sind zu einem einzigen generischen Service zusammengefasst.

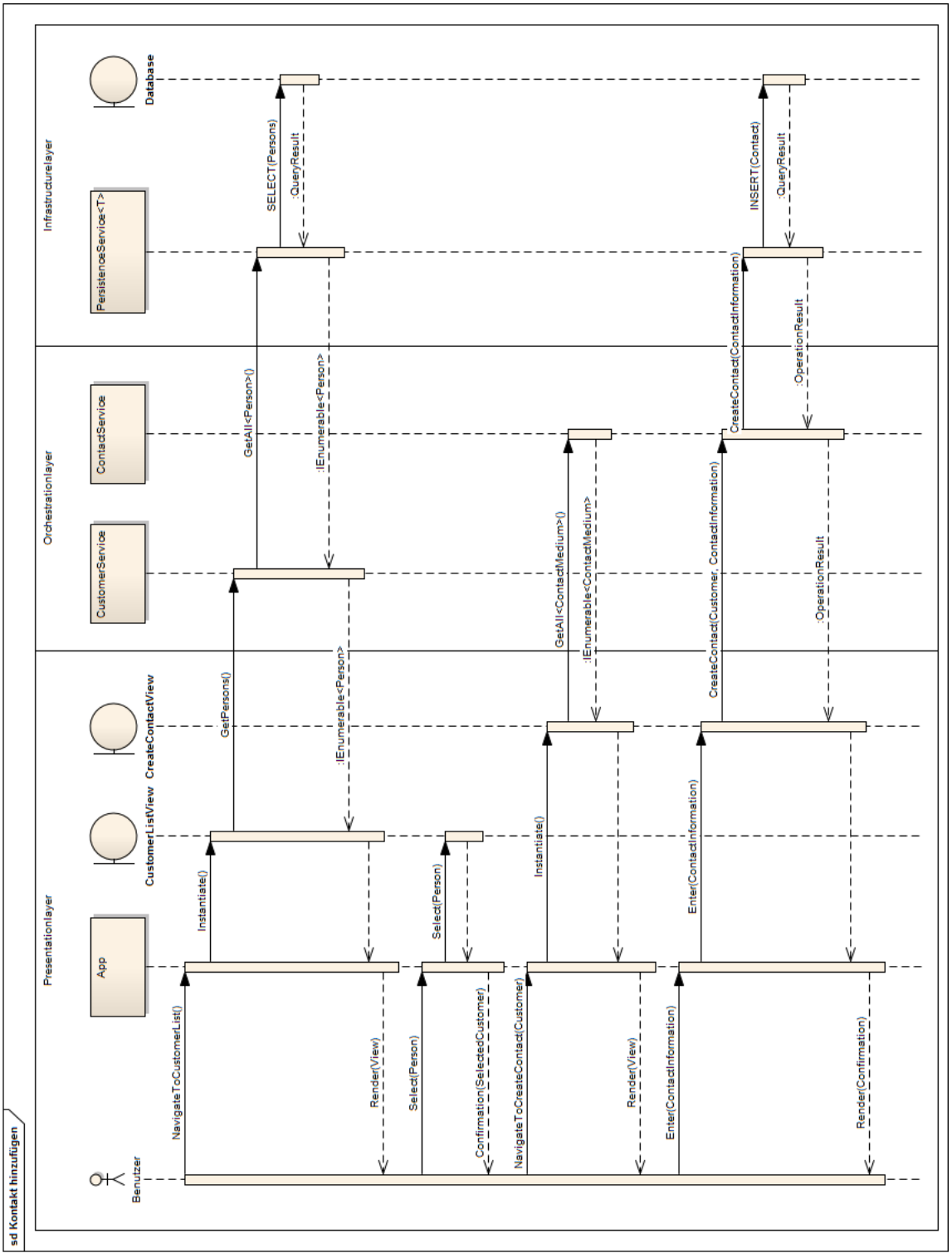

Abbildung 3.12: Sequenzdiagramm Kontakt hinzufügen

3.4.3 Verteilungssicht

Nach [Sta11] 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 Bedeutung für die Betreiber des Systems, die Hardwarearchitekten, das Entwicklungsteam sowie Management und Projektleitung (gemäss [HS13]).

3.4.3.1 Verteilungsdiagramm

Die Verteilungssicht dieser Projektdokumentation enthält nur ein sehr rudimentär ausgearbeitetes Verteilungsdiagramm (siehe Abbildung 3.13). Dies, da kein konkretes Verteilungsszenario der Applikation innerhalb dieser Arbeit geplant wird. Das Projekt beinhaltet die Erarbeitung des Konzepts sowie die konkrete Implementierung der Applikation ohne jedoch auf die Verteilung des Systems einzugehen.

Generell sind die drei verschiedenen Layer Presentationlayer, Orchestrationlayer und Infrastructurelayer unabhängig voneinander verteilbar. Sie können auf verschiedenen Rechner verteilt werden, auch eine Verteilung über das Internet wäre möglich. Der Orchestrationlayer und der Infrastructurelayer sind als WCF Service Applikationen implementiert und benötigen deshalb einen Internet Information Server (IIS) als Ausführungsumgebung. Aus Performancegründen macht es Sinn wenn der Infrastructureserver im gleichen Netzwerk wie der Datenbankserver ist, so dass bei Abfragen an die Datenbank keine grosse Latenz entsteht.

Die Clientapplikation App ist eine WPF-Anwendung die auf jedem beliebigen Windows Betriebssystem verteilt werden kann.

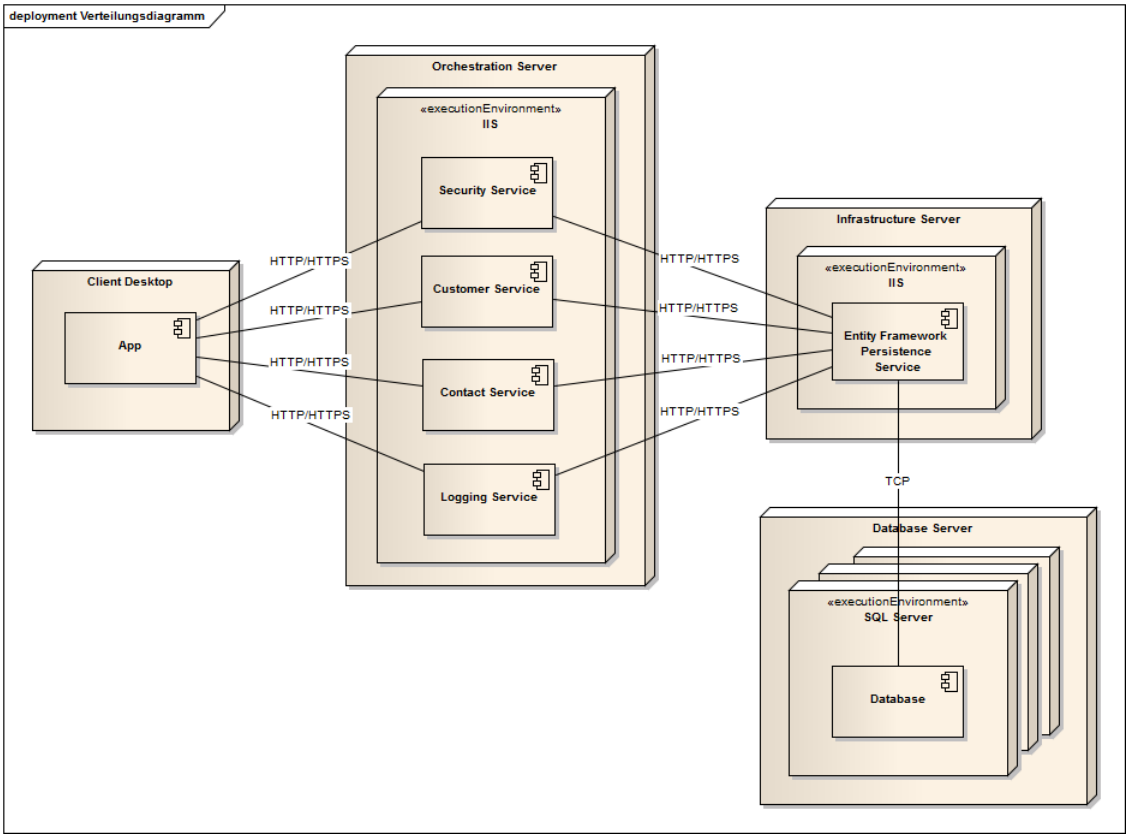

Abbildung 3.13: Verteilungsdiagramm SoCrm

4 Implementierung

Das Kapitel Implementierung beschäftigt sich mit der abschliessenden Evaluation der selektierten OR-Mapper aus den vorgehenden Kapiteln und erstellt eine Produktempfehlung auf Basis der Vergleiche sowie der erhobenen Anforderungen an den Persistenzlayer und Fowlers Enterprise Pattern.

Anschliessend wird die Implementierung des Proof of Concepts SoCrm im Detail vorgestellt. Es werden die einzelnen Visual Studio Projekte als Teilkomponenten im Detail beleuchtet und die vorgängig festgelegten Implementierungsprinzipien vorgestellt.

4.1 Evaluation selektierte OR-Mapper

Dieses Kapitel enthält die finale Evaluation der drei in den vorherigen Kapiteln behandelten OR-Mapper Entity Framework, Dapper und NHibernate. Dazu werden die drei Produkte rückblickend betrachtet, die im Rahmen der Evaluation gewonnenen Erkenntnisse und Erfahrung in der Benutzung Revue passiert. Die OR-Mapper werden in diesem Kapitel an den erhobenen Anforderungen an den Persistenzlayer einer serviceorientierten Architektur (siehe Kapitel 2.2 auf Seite 11) und an den Anforderungen der Enterprise Pattern (siehe Kapitel 2.3 auf Seite 13) gemessen. Es wird erneut darauf hingewiesen, dass die Anforderungen an den Persistenzlayer als Anforderungen an die Implementierung eines Persistenzlayers formuliert sind, nicht nur an den im Layer verwendeten ORM.

Die Evaluation der drei Produkte dient als Entscheidungsgrundlage für die nachfolgenden Produktempfehlungen (siehe Kapitel 4.2 auf Seite 80).

4.1.1 Entity Framework

Das Entity Framework ist klar der bekannteste ORM der in dieser Arbeit betrachteten OR-Mapper. Dies zeigt die Umfrage unter .NET-Entwicklern (siehe Kapitel 3.2.1 auf Seite 50) und beweist, dass Microsoft seit der Einführung des Produkts im August 2005 gute Arbeit bei der Produktqualität, den mitgelieferten Tools und der Dokumentation geleistet hat. Beim Rückblick auf die Marktübersicht (siehe Kapitel 3.1 auf Seite 40) und den erweiterten Vergleichen (siehe Kapitel 3.3 auf Seite 55) zeigt sich, dass nicht nur die Marketingstrategie von Microsoft fruchtet, sondern auch der Funktionsumfang des Entity Frameworks seinesgleichen sucht. Es bietet neben vielfältigen Assistenten und Designern auch die vollständige Unterstützung der beiden Mappingtechniken Forward Mapping und Reverse Mapping.

Dieser Funktionsumfang hat jedoch auch Nachteile was sich bei den Performancemessungen (siehe Kapitel 3.3.3 auf Seite 58) zeigt. Als kompletter OR-Mapper war voraus-

zusehen dass gegenüber eines leichtgewichtigen OR-Mappers Unterschiede in den Leistungsdaten sichtbar werden.

Betrachtet man die Anforderungen an den Persistenzlayer eines serviceorientierten Architektur so sind bis auf die Anforderungen PF9 (Das System muss dem Administrator die Möglichkeit bieten den Ablauf sämtlicher Operationen nachzuverfolgen) und PQ3 (Das System muss dem Entwickler die Möglichkeit bieten die externe Schnittstelle des Systems automatisiert zu testen) alle Anforderungen als erfüllt zu betrachten. PF9 ist über Drittanbieterwerkzeuge oder mittels zusätzlichem Implementierungsaufwand erfüllbar. Die Anforderung PQ3 bedeutet in diesem konkreten Fall, dass der ObjectContext des Entity Frameworks für Unit-Tests abstrahiert (z.B. mit einem Mockingframework) werden sollte. Dies stellt sich in der Praxis als sehr schwierig, wenn nicht gar unmöglich dar. Zu stark ist die Kopplung des ObjectContextes an Komponenten welche die Domänenobjekte des ORMs konsumieren, als dass dieser einfach abstrahiert werden könnte.

Beim Betrachten der Anforderungen der Enterprise Pattern werden bis auf D2 (Das System muss die Möglichkeiten einer Spezifikation bieten, mit welcher der Teil eines zu ladenden Objektgraphen beschrieben werden kann) alle Anforderungen erfüllt. Das Entity Framework bietet von Haus aus keine Möglichkeit der Spezifikation an, welcher Teil eines Objektgraphen geladen werden soll. Die Anforderung D2 kann jedoch mit der Verwendung von Lazy Loading und virtuellen Eigenschaften in den Domänenobjekten teilweise erfüllt werden. Wird eine vollständige Spezifikation des Objektgraphen gefordert (inklusive der Definition welcher Teil des Objektgraphen geladen werden soll und welcher nicht), so muss dies im Persistenzlayer selbst implementiert werden.

4.1.2 Dapper

Dapper hat innerhalb dieser Vergleiche für einige Überraschungen gesorgt. Der leichtgewichtige OR-Mapper, der in Form einer einzigen C#-Datei mit gerade einmal 3500 Zeilen ausgeliefert wird, bietet viele Funktionalitäten (wie zum Beispiel Cachingmechanismen oder Lazy Loading) die sonst nur in deutlich umfangreicheren ORMs anzutreffen sind. Insbesondere die einfache Konfiguration der Mappings über Lamdba Expressions und dynamische Datentypen machen Dapper für einen erfahrenen Entwickler attraktiv. Die Performance von Dapper ist den kompletten OR-Mappern Entity Framework oder NHibernate deutlich überlegen. Dies kann insbesondere bei leistungskritischen Systemen oder Webapplikationen mit vielen gleichzeitigen Zugriffen ein gewichtiges Auswahlkriterium darstellen.

Der minimalistische Ansatz von Dapper hat auch einige Nachteile. So muss der Entwickler zwingend das Datenbankschema hinter dem Domänenmodell kennen, da teilweise SQL-Statements als Parameter an Dapper übergeben werden. Auch ist Dapper nur effizient in Forward Mapping Szenarien einzusetzen, welche auf einem ohne Altlasten berücksichtigenden Domänenmodell basieren. Weicht das Domänenmodell vom Datenbankschema ab, ist ein aufwändiges Mapping nötig, welches schnell unübersichtlich wird.

Beim Vergleich von Dapper mit den funktionalen Anforderungen an den Persistenzlayer zeigt sich der geringere Funktionsumfang gegenüber den kompletten OR-Mappern: PF8 (Das System muss mehrere Instanzen gleichzeitig löschen können) und PF9 (Das

System muss dem Administrator die Möglichkeit bieten den Ablauf sämtlicher Operationen nachzuverfolgen) sind nicht erfüllt. Zwar lässt sich PF9 vom Entwickler nachträglich implementieren, aber die Möglichkeiten zur Überwachung des Zustands von Dapper sind nicht besonders vielfältig und oft schwierig zu nutzen (siehe auch Kapitel 3.3.6 auf Seite 61). Die Qualitätsanforderungen und Randbedingungen sind ähnlich abgedeckt wie die funktionalen Anforderungen. Zwar wurde Dapper mit der Intention entwickelt die Datenbank einfach für automatische Tests abstrahieren zu können, trotzdem ist PR2 (Das System muss dem Entwickler die Möglichkeit bieten ohne Wissen über die Datenbank oder datenbanknahe Technologien auf die Operationen zuzugreifen) und PR3 (Das System muss dem Entwickler für sämtliche Operationen ein Resultat entweder in Form der Entität oder den Entitäten oder in Form eines Operationsresultats zurückliefern) aus den vorher genannten Gründen nicht erfüllt.

Werden die Anforderungen der Enterprise Pattern in Bezug auf Dapper als der verwendete OR-Mapper näher betrachtet, so gelten bis auf D1 (Das System muss eine minimale Kopplung des Domain Models an andere Layer ermöglichen), D2 (Das System muss die Möglichkeiten einer Spezifikation bieten, mit welcher der Teil eines zu ladenden Objektgraphen beschrieben werden kann) und D5 (Das System muss das Domain Model von der Datenbank vollständig entkoppeln) als erfüllt. Bei diesen drei nicht erfüllten Anforderungen zeigt sich auch die Schwierigkeit des Einsatzes von Dapper in einem Persistenzlayer mit einem Domain Model. Theoretisch könnte die fehlende Funktionalität vom Entwickler implementiert werden, andere ORMs wie das Entity Framework oder NHibernate bieten diese jedoch schon als Funktionalität an.

4.1.3 NHibernate

NHibernate erfreut sich immer noch ausserordentlicher Beliebtheit in der Entwicklercommunity. Dies liegt daran, dass NHibernate lange der einzige ausgereifte komplette OR-Mapper im .NET-Umfeld war und dadurch viele erfolgreich abgeschlossene Projekte unter Verwendung von NHibernate realisiert werden konnten. Viele .NET-Entwickler haben NHibernate schon einmal integriert oder in einem Projekt mitgearbeitet in welchem NHibernate für das objektrelationale Mapping eingesetzt wurde. Da NHibernate eine Portierung des auf Java basierten Hibernate ist, ist die Entwicklung in den letzten Jahren etwas eingeschlafen und NHibernate kann heute nicht mehr mit anderen kompletten ORMs wie dem Entity Framework oder dem OpenAccess ORM von Telerik mithalten. Viele der Funktionalitäten die NHiberante im Vergleich zu den Konkurrenzprodukten fehlen, wurden von der OpenSource Community oder Drittkomponentenherstellern entwickelt und zur Verfügung gestellt. So lassen sich heute die Mappings der Domänenobjekte auf Datenbanktabellen nicht nur über XML-Dateien erstellen, sondern auch über grafische Designer oder unter Verwendung von flüssigen Schnittstellen.

NHibernate besitzt eine relativ steile Lernkurve ist dafür jedoch sehr flexibel einsetz- und konfigurierbar. Durch diese umfangreiche Konfigurierbarkeit ist Disziplin vom Entwickler gefordert. So existieren beispielsweise mehrere Wege um Mappings zu erstellen und in grösseren Projekten mit mehreren Entwicklern kann dies zu verwaistem (da nicht verständlichem) Code oder einem Wildwuchs von Konfigurationsdateien führen.

Wird NHibernate mit den Anforderungen an den Persistenzlayers einer serviceorientierten Architektur konfrontiert, so werden bis auf PF9 (Das System muss dem Administrator die Möglichkeit bieten den Ablauf sämtlicher Operationen nachzuverfolgen) und PQ3 (Das System muss dem Entwickler die Möglichkeit bieten die externe Schnittstelle des Systems automatisiert zu testen) sämtliche Anforderungen erfüllt. PF9 und PQ3 sind dieselben Anforderungen die auch das Entity Framework nicht erfüllen konnte. Auch NHibernate leidet unter Defiziten was die Überwachung des ORMs zur Laufzeit sowie das automatisierte Testen des Datencontainers angeht. Zwar können Überwachungsmechanismen selbst in die Kernfunktionalität von NHibernate eingebaut werden (da Open Source), dies ist jedoch nur bedingt anzuraten und setzt vertiefte Kenntnisse des Innenlebens von NHibernate voraus. NHibernate stammt noch aus einer Zeit wo weniger Wert auf das automatisierte Testen von einzelnen Programmelementen wie Klassen oder Komponenten gelegt wurde. Dadurch ist es auch schwierig den Datencontainer von NHibernate für Testzwecke zu abstrahieren.

Die Anforderungen der Enterprise Pattern werden bis auf A2 (Das System muss keine Konversion von Objektdatentypen und Datentypen der Datenbanktabelle verwenden) vollständig erfüllt. Da NHibernate über JDBC auf das Datenbankmanagementsystem zugreift, ist immer eine (wenn auch für den Entwickler transparente) Konvertierung aller Datenwerte nötig.

4.2 Produktempfehlungen

In diesem Kapitel wird für jedes der Enterprise Pattern Transaction Script, Domain Model, Table Module und Active Record eine Empfehlung abgegeben, welcher der drei OR-Mapper Entity Framework, Dapper oder NHibernate eingesetzt werden soll.

Wie schon unter dem Vergleichskriterium Unterstützung Enterprise Pattern (siehe Kapitel 3.3.5 auf Seite 61) festgehalten, sind grundsätzlich alle OR-Mapper für alle betrachteten Enterprise Pattern geeignet. In den folgenden Kapiteln wird trotzdem versucht eine differenzierte Betrachtungsposition anzunehmen und pro Enterprise Pattern eine konkrete Empfehlung abzugeben.

4.2.1 Transaction Script

Das Transaction Script ist das am einfachsten zu implementierende und simpelste der betrachteten Enterprise Pattern (siehe Kapitel 2.3.1 auf Seite 14). Im minimalsten Fall ist ein Transaction Script eine Klasse mit einer statischen Methode. Diese statische Methode kann einen oder mehrere Parameter besitzen, welche die Eingaben für die Transaktion liefern. Die Methode führt dann die Transaktionsbefehle aus und liefert einen Rückgabewert an die aufrufende Funktion.

Dadurch dass dieses Enterprise Pattern so einfach zu implementieren ist, sind alle drei betrachteten OR-Mapper gleich gut dafür geeignet. Somit werden bei Verwendung von Transaction Scripts in einem Persistenzlayer die OR-Mapper Entity Framework, Dapper, NHibernate gleichermassen empfohlen.

4.2.2 Domain Model

Wird das Domain Model als Enterprise Pattern innerhalb eines Persistenzlayers verwendet sind die betrachteten kompletten OR-Mapper Entity Framework und NHibernate klar im Vorteil. Bei einem Domain Model unterscheidet sich das Domänenmodell in grossen Teilen vom Datenbankschema und somit kann Dapper nur mit viel Mappingcode verwendet werden. Das Erstellen der Mappings wird in Dapper unter anderem über SQL-Statements gelöst und insbesondere ein reiches Domain Model mit objektorientierten Funktionalitäten ist schwierig mit Dapper abzubilden (siehe Kapitel 2.3.2 auf Seite 15).

Das Entity Framework erhält die Empfehlung als der zu verwendende OR-Mapper in einem Persistenzlayer mit einem Domain Model. Dies liegt einerseits am im Visual Studio integrierten Model Designer andererseits an der Flexibilität von EF was die Mappingtechniken angeht. Wird das Domain Model zuerst in Code definiert, ist es mit dem Entity Framework ohne weiteres möglich eine Abbildung auf eine Datenbank zu erstellen. Auch NHibernate kann gut mit Domain Models umgehen. Das Mapping ist für den Entwickler jedoch weniger komfortabel zu erstellen, da umfangreiche XML-Mappingdateien angelegt werden müssen. Diese XML-Mappingdateien sind fehleranfälliger (da Fehler erst zur Laufzeit entdeckt werden) als der Model Designer des Entity Frameworks oder die Mappings in Code (da bei dieser Mappingtechnik Fehler schon zur Kompilierzeit entdeckt werden).

4.2.3 Table Module

Dapper erhält die Empfehlung als der zu verwendende OR-Mapper beim Einsatz von Table Modules in einem Persistenzlayer einer serviceorientierten Architektur. Für diese Empfehlung gibt es zwei ausschlaggebende Argumente: Das erste Argument ist die Möglichkeit von Dapper mit einem generischen Record Set direkt Mappings auf Domänenobjekten zu erzeugen. Da ein Table Module oft mit einem Record Set als Konstruktorparameter initialisiert wird, ist dies ein grosser Vorteil. Das an einer anderen Stelle erzeugte Record Set kann dem Table Module übergeben werden und Dapper kann damit ohne weiteres Abfragen der internen Mapping-Engine oder dem Datenbankschema das Mapping auf Domänenobjekte beziehungsweise Datenbanktabellen durchführen. Der zweite Grund ist die Anforderung T1 (Das System muss die Objektstruktur analog der Datenbankstruktur abbilden) des Table Modules (siehe Kapitel 2.3.3 auf Seite 16). Wie in den detaillierten Vergleichen ausgeführt, ist Dapper gut geeignet beim Verwenden von tabellarischen Daten und dem Erzeugen der dazugehörigen Mappings.

Würde man NHibernate oder das Entity Framework für ein Table Module verwenden so würde bei der Initialisierung des Table Modules durch die Nutzung der Datencontainer immer zuerst ein Abgleich mit den im Record Set enthaltenen Objekten und den im Datencontainer bereits vorhandenen Daten notwendig sein. Auch verkompliziert die Nutzung der beiden kompletten OR-Mapper potentiell das ansonsten leicht verständliche Enterprise Pattern.

4.2.4 Active Record

Auch für die Verwendung des Enterprise Patterns Active Record erhält Dapper die Empfehlung als den zu verwendenden OR-Mapper. Die Gründe für die Empfehlung von Dapper bei der Implementierung von Active Records sind ähnlich wie beim Table Module. Auch beim Active Record wird eine der Datenbankstruktur analoge Objektstruktur gefordert (Anforderung A1, siehe Kapitel 2.3.4 auf Seite 17). NHibernate führt auf Grund seiner Architektur in jedem Fall eine Typenkonvertierung durch und ist somit es für Active Record wenig geeignet. In Anbetracht des Funktionsumfangs sind das Entity Framework (und auch NHibernate, würde es keine Konvertierung benötigen) für dieses Pattern des Guten zu viel. Dapper ist als leichtgewichtiger OR-Mapper das einzige der drei betrachteten Produkte welches den Gedanken von Active Record unterstützt, ein simples Zugriffsmodell für höhere Schichten anzubieten, das wenig und nur möglichst einfache Geschäftslogik enthält.

4.3 Serviceorientierte CRM-Applikation

In diesem Kapitel wird die Implementierung der serviceorientierten CRM-Applikation beschrieben. Es werden zunächst die Vorgaben an die Entwicklung festgehalten. Dazu gehören Richtlinien wie zum Beispiel der Aufbau der Visual Studio Solution oder auch die Wahl des UI-Patterns. In einem zweiten Unterkapitel werden die einzelnen Projekte der Visual Studio Solution im Detail beschrieben und als Klassendiagramme dargestellt.

Das Ziel dieses Kapitels ist es einerseits dem Leser einen Überblick über die Implementierung des Proof of Concepts zu geben und andererseits die getroffenen Entscheidungen während der Entwicklung explizit zu machen.

4.3.1 Implementierungsprinzipien

Unter Implementierungsprinzipien werden Richtlinien verstanden, die während der Implementierung eines Projekts von den Softwareentwicklern beachtet werden sollen. Häufig sind diese Implementierungs- oder Designprinzipien durch den Softwarearchitekten oder Lead Developer vorgegeben und werden von diesen auch überwacht.

Die nachfolgende Liste beschreibt die in der Implementierung der serviceorientierten CRM-Applikation SoCrm beachteten Implementierungsprinzipien:

- Die Namen der Assemblies und der darin enthaltenen Namespaces müssen zwingend übereinstimmen. Würden diese nicht übereinstimmen, müsste in der Dokumentation jeweils neben dem Namen des Assemblies auch noch der vollqualifizierte Namen jedes Typen angegeben werden um eine eindeutige Zuweisung zu ermöglichen.

- Sämtliche Interfaces und Entitäten sind in eigenen Klassenbibliotheken abgelegt. Dies, um die Wiederverwendbarkeit zu erhöhen und die Grösse der verteilten und mehrfach verwendeten Assemblies zu minimieren. Servicereferenzen in Projekten, welche Web Services verwenden, benutzen immer die Typen dieser Assemblies anstatt die Entitäten aus dem WSDL der Servicereferenz zu erzeugen.

- Die Persistenzservices (in SoCrm.Infrastructure.Persistence) besitzen keinerlei Geschäftslogik sondern dienen nur dazu Objekte aus dem jeweils verwendeten OR-Mapper aus der Datenbank zu lesen, zu aktualisieren, zu erstellen oder zu löschen. Sämtliche Geschäftslogik ist in den Services des Orchestrationlayers (in SoCrm.Services) implementiert.

- Das Präsentationspattern, welches verwendet wird, ist MVVM+C (nach [Ski12]). MVVM+C stellt eine Weiterentwicklung von MVVM[1] 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 über das Binding gelöst.

- Um dynamisch Views innerhalb einer Shell laden zu können, sind alle Views als UserControls implementiert, welche vom Controller in den ControlContainer der ShellView zur Laufzeit eingefügt werden.

- Die Serviceimplementierungen (.svc Code-Behind-Dateien) sollen möglichst schlank und lesbar sein, da sie den ersten Anlaufspunkt für Entwickler darstellen. Deswegen wird möglichst viel Logik in Basisklassen bzw. weitere Objekte ausgelagert insbesondere im Zusammenhang mit dem Zugriff auf die ORMs. Dadurch wird die Les- und Testbarkeit der Serviceimplementierungen 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 gesamtheitlicher technischer Überblick über die Umsetzung des Konzepts gegeben.

Zur besseren Orientierung innerhalb dieses Kapitels wird auf Abbildung 4.1 verwiesen, welche die Struktur der verwendeten 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 für das Ziel dieses Kapitels nicht weiter relevant sind und deswegen in den Klassendiagrammen auch nicht gezeichnet sind. Auch werden nur in denjenigen Unterkapiteln Klassendiagramme eingefügt, wo es für den Leser von Interesse sein könnte.

[1] Mehr zum MVVM Entwurfsmuster im MSDN Magazine: `http://msdn.microsoft.com/en-us/magazine/dd419663.aspx`.

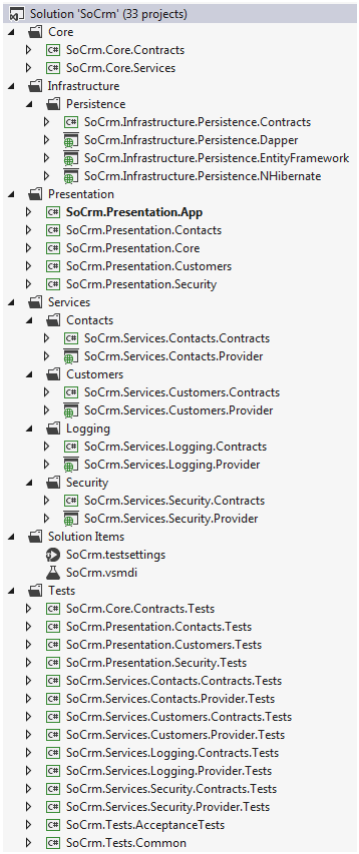

Abbildung 4.1: Visual Studio Solution

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 aller Domänenobjekte, die ObjectId, deklariert.

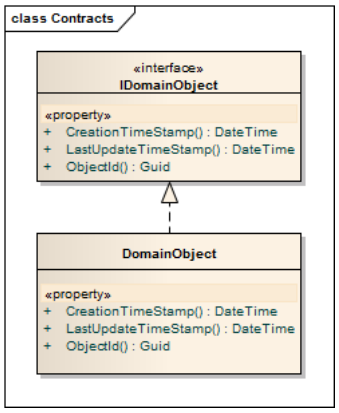

Abbildung 4.2: Klassendiagramm SoCrm.Core.Contracts

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[2] 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 in den Service injiziert. Dieser 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 Interface von einem generischen Interface IPersistenceService<T>. Da WCF jedoch keine generischen Schnittstellen unterstützt wurde für jeden Entitätstyp ein eigenes Persistenceservice Interface erstellt (siehe Abbildung 4.3).

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

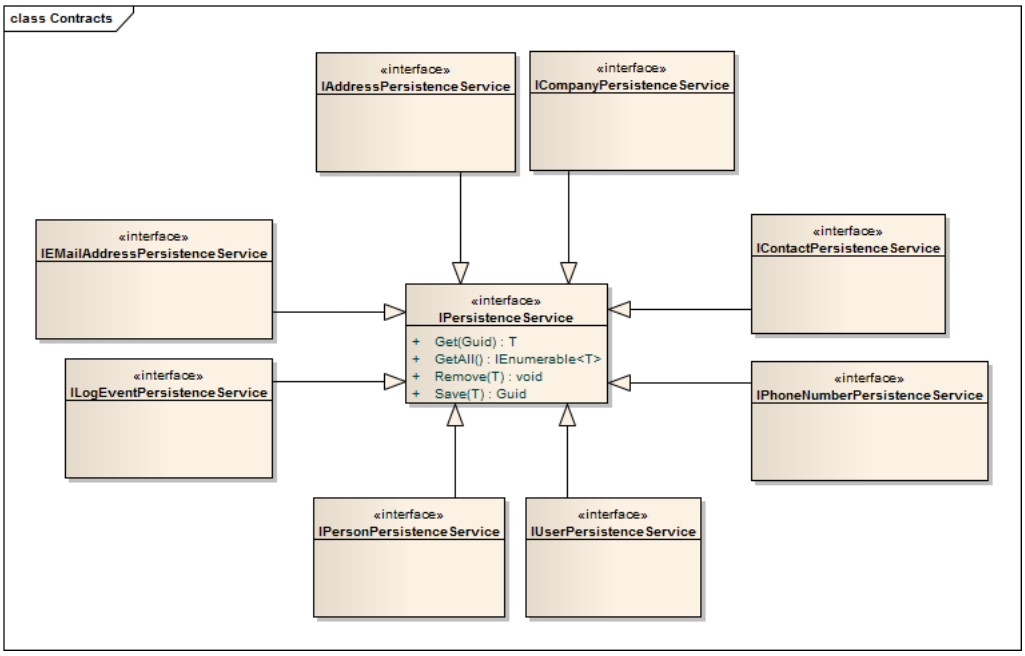

Abbildung 4.3: Klassendiagramm SoCrm.Infrastructure.Persistence.Contracts

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 Persistenzservices, welche jeweils einen eigenen WCF Service realisieren. Jeder Persistenzservice implementiert das entsprechende PersistenceService Interface aus SoCrm.Infrastructure.Persistence.Contracts und leitet von der PersistenceServiceBase ab.

Die PersistenceServiceBase beinhaltet sämtliche Zugriffslogik auf Dapper um einzelne Entitäten zu lesen oder zu löschen. Das Speichern und Lesen von Objektgraphen ist im jeweiligen Persistenzservice implementiert, da ein manuelles Mapping innerhalb von Dapper dafür notwendig ist.

4.3.2.5 SoCrm.Infrastructure.Persistence.EntityFramework

Die Persistenzlayerimplementierung des Entity Frameworks als WCF Service Applikation in SoCrm.Infrastructure.Persistence.EntityFramework besitzt wiederum acht Persistenzservices welche die entsprechenden Service Contracts aus SoCrm.Infrastructure.Persistence.Contracts implementieren. Sämtliche Logik ist dabei in der generischen Basisklasse PersistenceServiceBase<T> abgelegt, die einzelnen PersistenceServices sind leer und dienen lediglich als Endpoints für die WCF Services.

Die PersistenceServiceBase kümmert sich um das Vorbereiten der Entitäten (Validierung von Benutzerdaten, Setzen von Änderungsdatum und -Benutzer, usw.) und holt

sich aus der DataServiceFactory den pro Entität implementierten DataService. Der DataService kümmert sich um den Zugriff auf den jeweiligen ObjectContext des Entity Frameworks.

Aus Gründen der Übersichtlichkeit und Erweiterbarkeit wurden vier ObjectContexte erstellt, einer für jede Funktionalität des Systems (SecurityContext, LoggingContext, CustomerContext und ContactContext).

Da das Entity Framework in der Version 5.0 verwendet wird, wurden die Entitäten Code-First entwickelt. Das Entity Framework bietet für den dieses Szenario Migrationen an, welche von der PowerShell-Extension über die NuGet-Konsole erzeugt werden. Diese Migrationen sind im Namespace SoCrm.Infrastructure.Persistence.EntityFramework.Migrations abgelegt.

4.3.2.6 SoCrm.Infrastructure.Persistence.NHibernate

Die WCF Service Applikation SoCrm.Infrastructure.Persistence.NHibernate enthält die Persistenzlayerimplementierung des ORMs NHibernate. Alle acht Persistenzservices benutzen die PersistenceServiceBase<T> Klasse als Basisklasse. Diese Basisklasse kapselt den Session-Container von NHibernate und welcher über die im NHibernateHelper versteckte SessionFactory erzeugt wird. Der NHibernateHelper erstellt die Mapping-Konfiguration von NHibernate und lädt alle Mappings der Entitäten beim Aufstarten der WCF Service Applikation.

Die Mappings aller Entitäten sind im Namespace SoCrm.Infrastructure.Persistence.NHibernate.Mappings abgelegt. Dabei wird FluentNHibernate[3] verwendet, um die XML-Konfiguration zu vereinfachen und den Aufwand diese zu Erstellen gering zu halten.

4.3.2.7 SoCrm.Presentation.App

Dieses Projekt liefert eine ausführbare Assembly als Build-Output. Die ShellView beinhaltet 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 Unity-Container die Controller der Assemblies SoCrm.Infrastructure.Presentation.Contacts, SoCrm.Infrastructure.Presentation.Customers oder SoCrm.Infrastructure.Presentation.Security 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 ihres Presentationmoduls im Unity-Container registrieren.

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

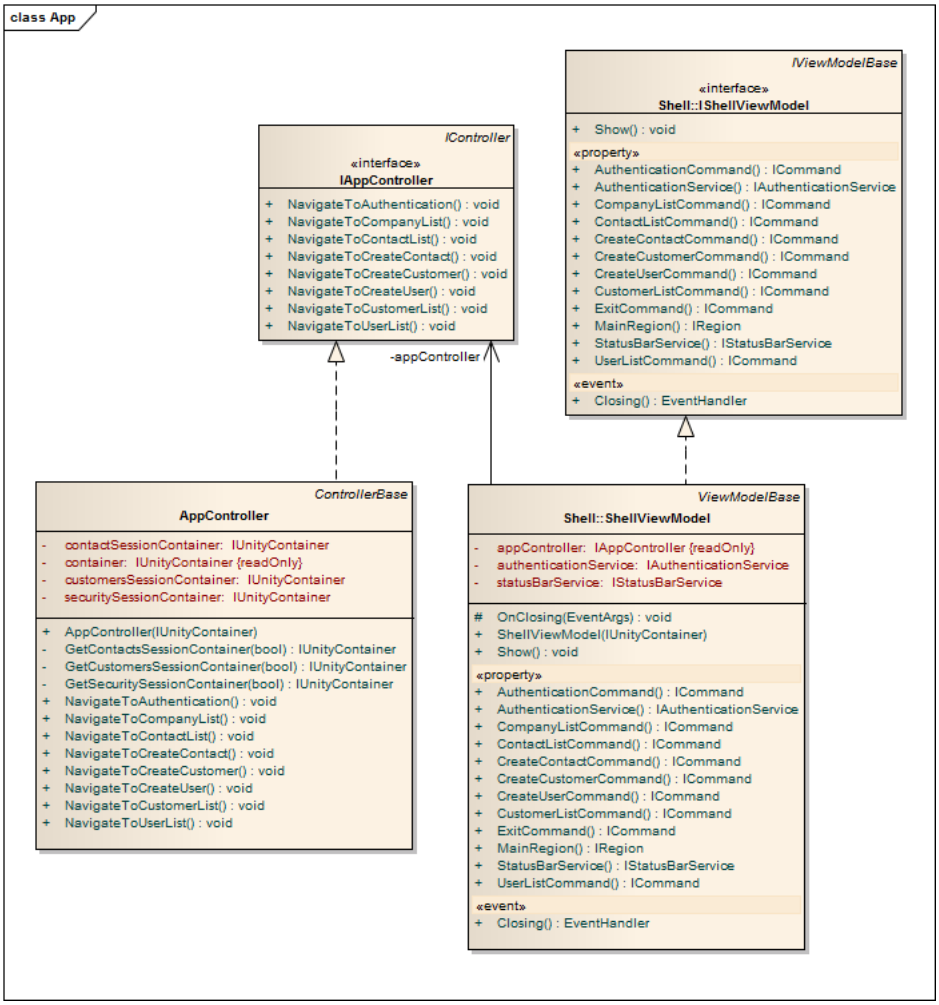

Abbildung 4.4: Klassendiagramm SoCrm.Presentation.App

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 instanziert 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).

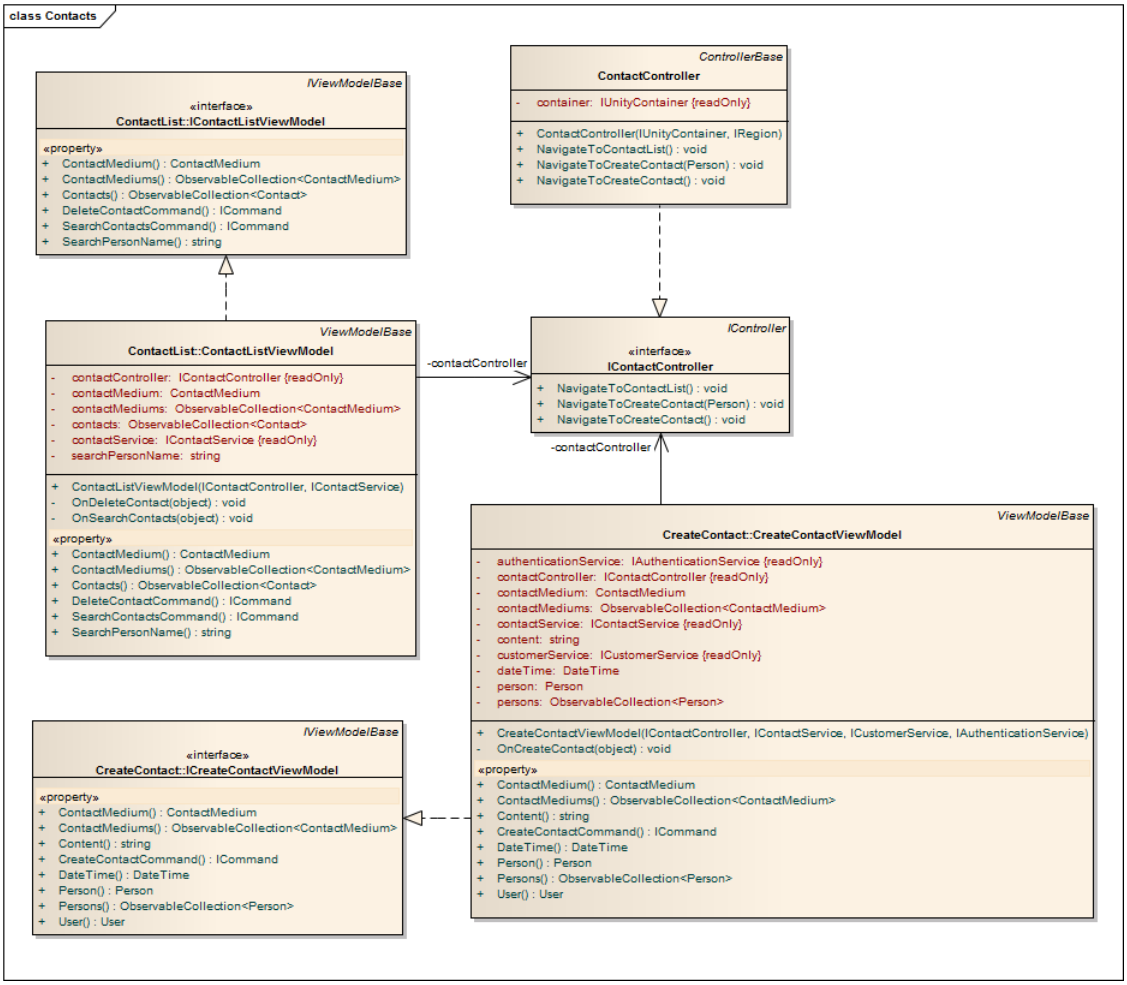

Abbildung 4.5: Klassendiagramm SoCrm.Presentation.Contracts

4.3.2.9 SoCrm.Presentation.Core

Diese Assembly enthält die gesamte Infrastruktur für die WPF Anwendung. Dazu gehören die Basisklassen für ViewModels und Controller als auch der StatusBarService, welcher Nachrichten in der Statusbar der ShellView darstellt. Das RegionModel dient zum Darstellen der aktuellen View (jeweils implementiert als UserControl) im ContentControl der ShellView. Die ViewModelBase implementiert das Interface INotifyPropertyChanged[4] welches das Binding bei geänderten Objektdaten im Modell benachrichtigt.

[4] Mehr zu INotifyPropertyChanged in der MSDN unter `http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged.aspx`.

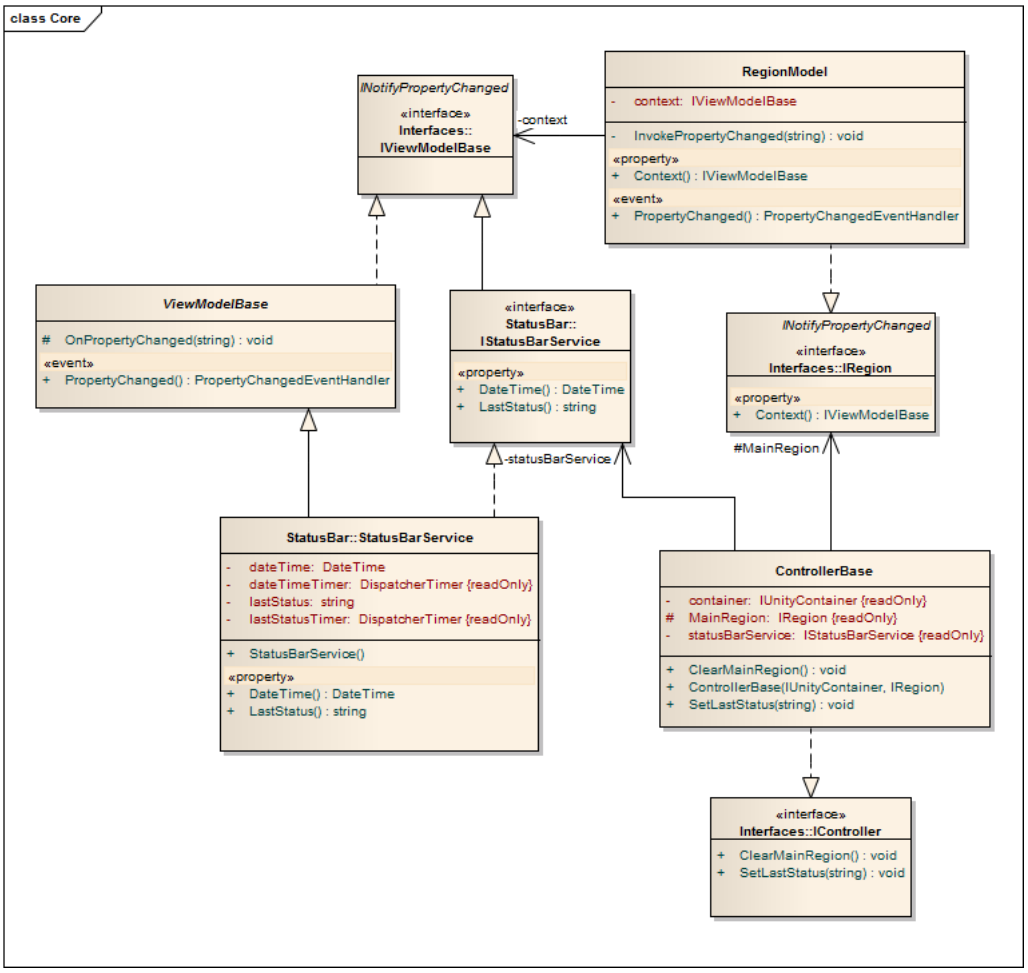

Abbildung 4.6: Klassendiagramm SoCrm.Presentation.Core

4.3.2.10 SoCrm.Presentation.Customers

Die Assembly SoCrm.Infrastructure.Presentation.Customers enthält sämtliche Funktionalität für das Anzeigen, Hinzufügen und Löschen von Kunden wie auch für Firmen. Alle Views und ViewModels werden über den CustomerController instanziert und dem Benutzer der Applikation angezeigt. Die komplexeste Funktionalität in diesem Presentationmodul ist das Hinzufügen von neuen Kunden (CreatePersonView und CreatePersonViewModel). Beim Erstellen eines neuen Kunden können dem Kunden über Subviews auch Arbeitgeber (CreateCompanyView und CreateCompanyViewModel), E-Mail-Adressen (CreateEMailAddressView und CreateEMailAddressViewModel) und Telefonnummern (CreatePhoneNumberView und CreatePhoneNumberViewModel) hinzugefügt werden.

Die Views CustomerListView und CompanyListView (sowie die dazugehörigen ViewModels CustomerListViewModel und CompanyListViewModel) erlauben es dem Benutzer Listen von Kunden bzw. Firmen anzuzeigen und diese zu filtern.

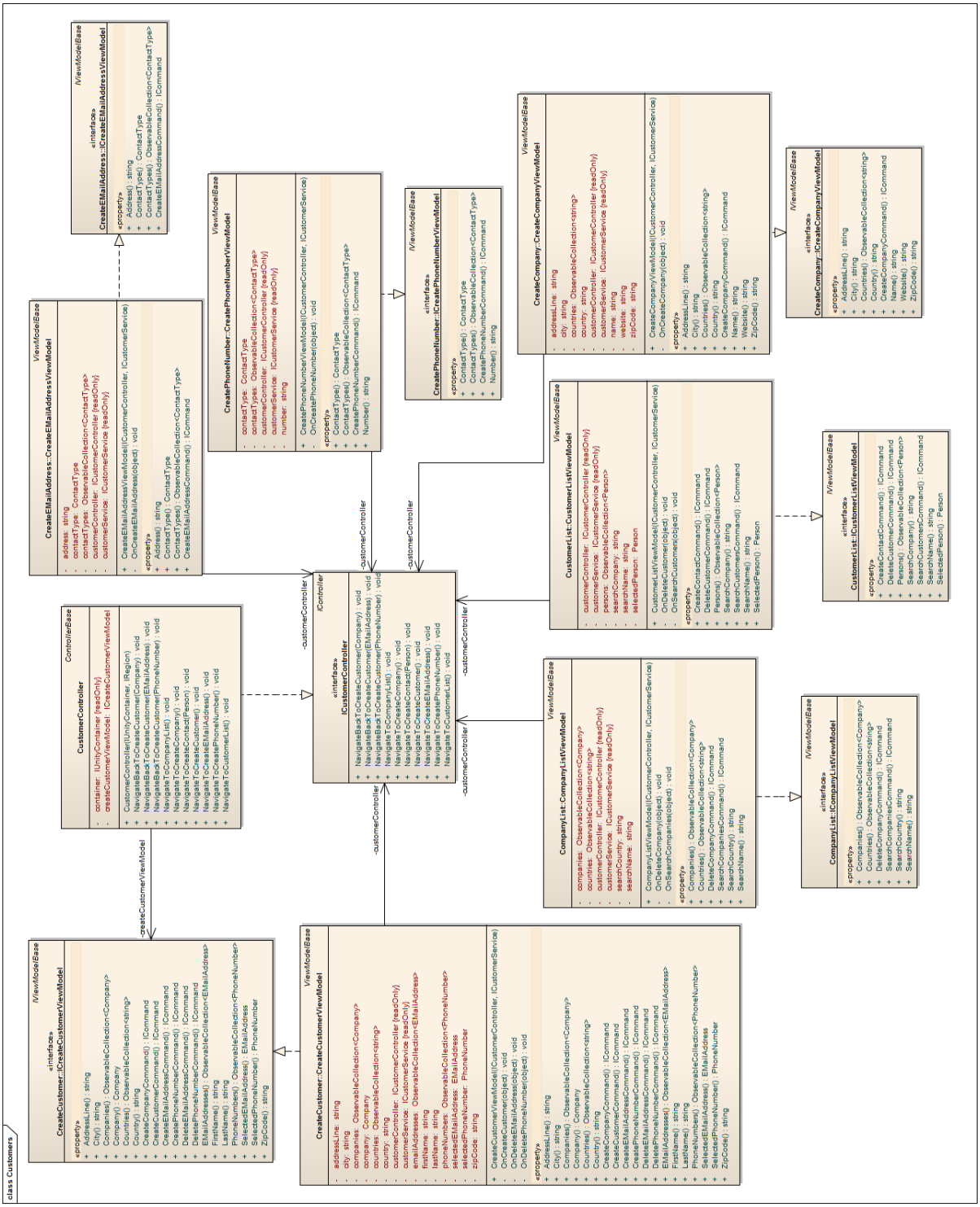

Abbildung 4.7: Klassendiagramm SoCrm.Presentation.Customers

4.3.2.11 SoCrm.Presentation.Security

SoCrm.Infrastructure.Presentation.Security enthält die Views und ViewModels für die Benutzerverwaltung sowie der Authentifizierung beim System. Auch ist die SetPasswordView und das SetPasswordViewModel ein Teil dieser Assembly. Das AuthenticationViewModel besitzt eine Referenz auf den AuthenticationService welcher als Singelton im Unity-Container registriert ist und den jeweils angemeldeten Benutzer verwaltet. Der SecurityController verwaltet wie in allen anderen Presentationmodulen auch hier die Instanzierung der Views und ViewModels.

Der Administrator kann über die CreateUserView und das CreateUserViewModel neue Benutzer dem System hinzufügen. Die UserListView und das UserListView dienen zum Anzeigen aller im System vorhandenen Benutzern.

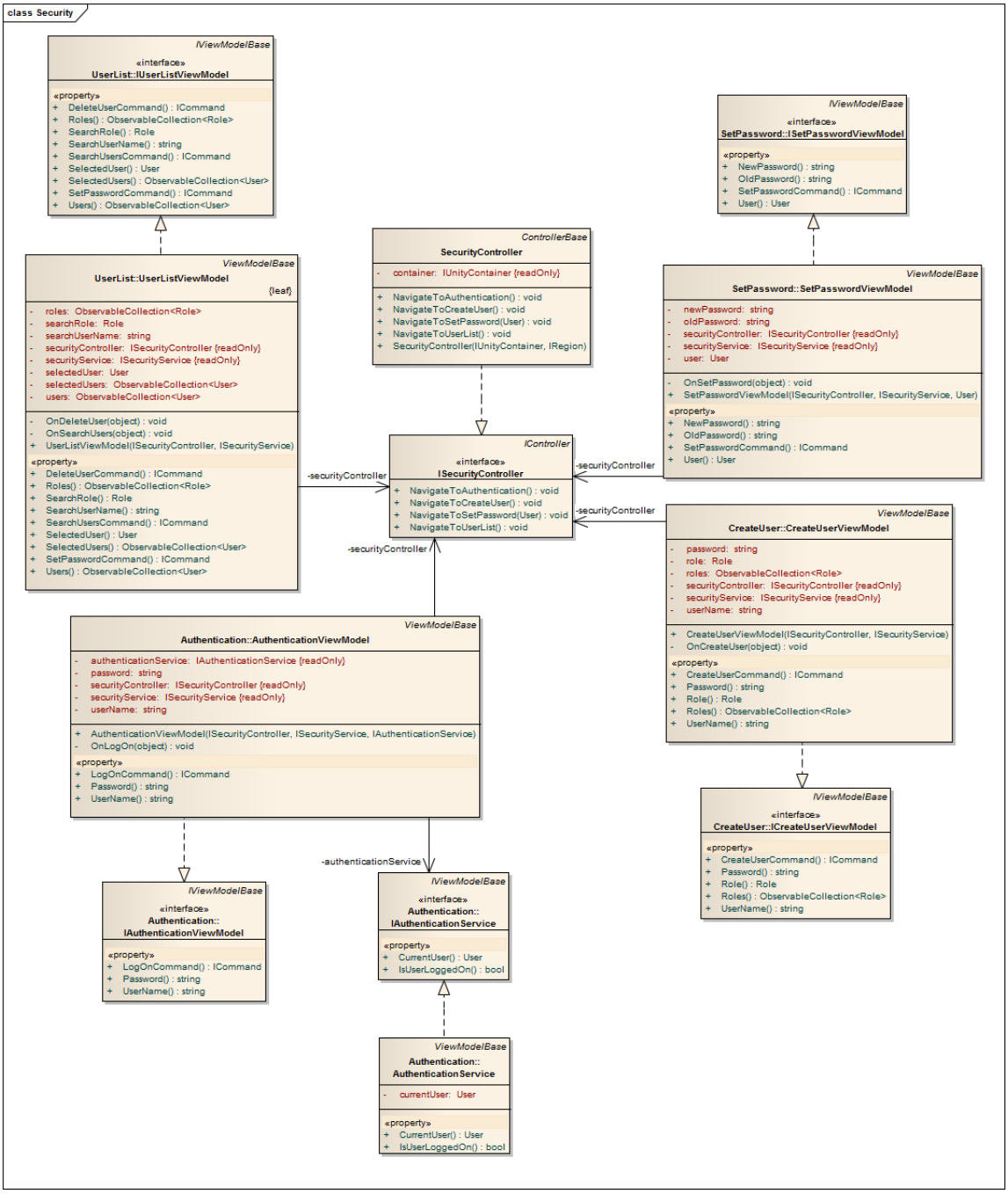

Abbildung 4.8: Klassendiagramm SoCrm.Presentation.Security

4.3.2.12 SoCrm.Services.Contacts.Contracts

In dieser Assembly befindet sich die Entität Contact, welche von DomainObject (in SoCrm.Core.Contracts, siehe Kapitel 4.3.2.1 auf Seite 84) erbt und das bereits in einem vorherigen Kapitel beschriebene Service-Interface IContactService (siehe Kapitel 3.4.1.3 auf Seite 70).

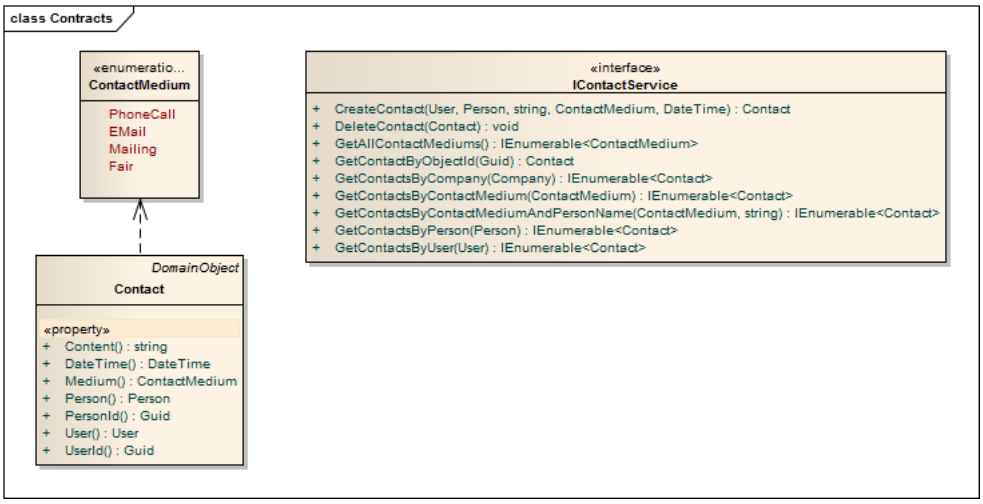

Abbildung 4.9: Klassendiagramm SoCrm.Services.Contacts.Contracts

4.3.2.13 SoCrm.Services.Contacts.Provider

Die WCF Service Applikation SoCrm.Services.Contacts.Provider liefert die Implementierung zum Service-Interface IContactService, das im Namespace SoCrm.Services.Contacts.Contracts definiert ist. Der ContactService besitzt dabei seine eigene ContactServiceHostFactory die von ServiceHostFactory ableitet und den UnityServiceHost (siehe Kapitel 4.3.2.2 auf Seite 85) instanziert und einen neuen Unity-Container erstellt. Der ContactService implementiert sämtliche Geschäftslogik betreffend der Kontaktverwaltung und ruft den Persistenzservice für Kontakte (implementiert in einem der SoCrm.Infrastructure.Persistence-Projekte) auf, um die Entitäten aus dem Persistenzlayer zu lesen und mit diesen zu interagieren.

4.3.2.14 SoCrm.Services.Customers.Contracts

Die Klassenbibliothek SoCrm.Services.Customers.Contracts enthält alle Entitäten den Kunden sowie Firmen betreffend. Ausserdem ist das Interface ICustomerService in dieser Assembly definiert (siehe Kapitel 3.4.1.3 auf Seite 69), welches in SoCrm.Service.Customers.Provider als WCF Service CustomerService implementiert ist.

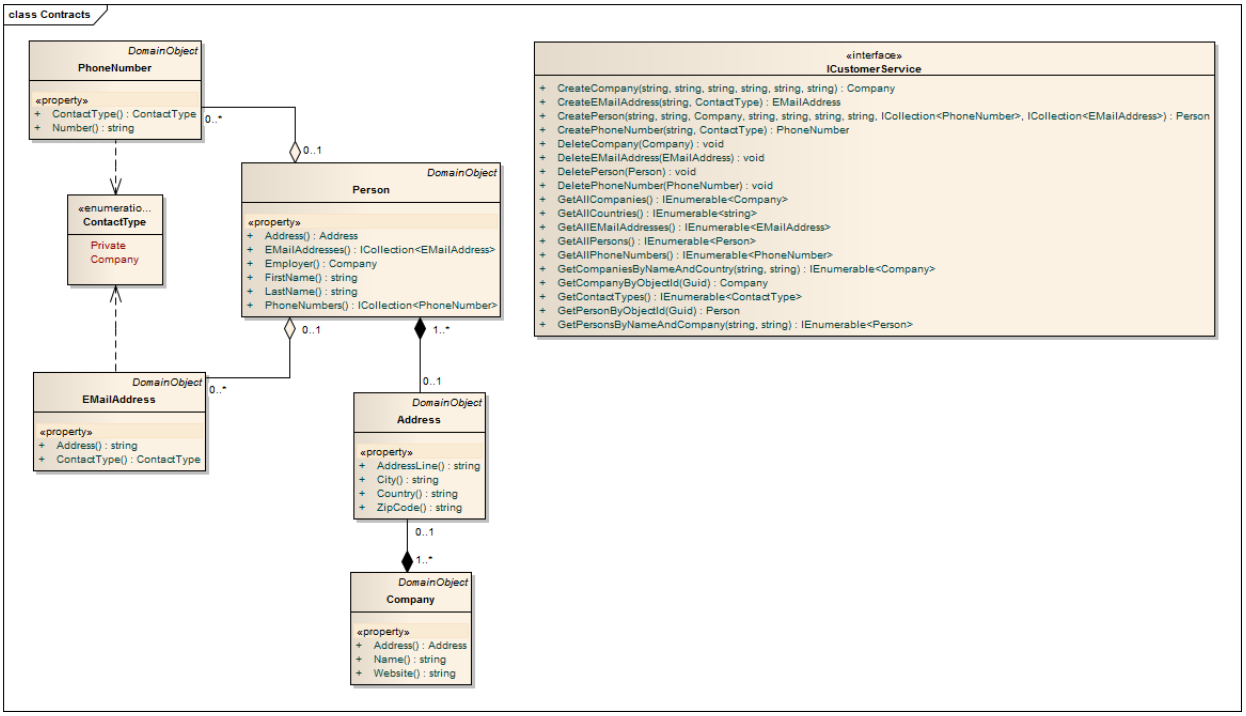

Abbildung 4.10: Klassendiagramm SoCrm.Services.Customers.Contracts

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 IAddressPersistenceService, ICompanyPersistenceService, IEMailAddressPersistenceService, IPersonPersistenceService und IPhoneNumberPersistenceService zugreift. Die Assembly beinhaltet alle Geschäftslogik um mit Personen und Firmen zu interagieren, diese zu Erstellen und zu Löschen.

4.3.2.16 SoCrm.Services.Logging.Contracts

Die Assembly SoCrm.Services.Logging.Contracts enthält die Entität LogEvent und das Service-Interface ILoggingService (siehe Kapitel 3.4.1.3 auf Seite 71).

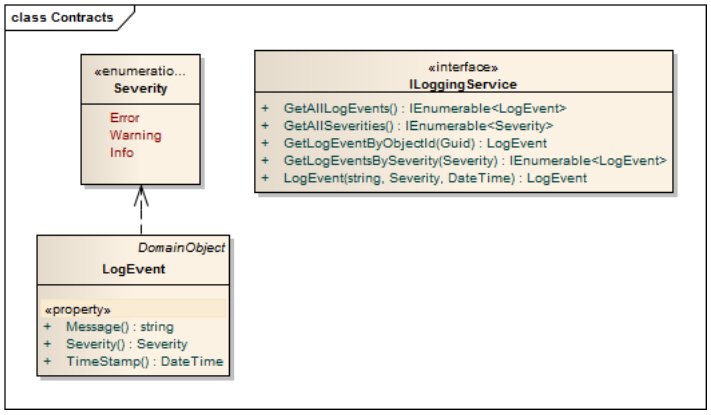

Abbildung 4.11: Klassendiagramm SoCrm.Services.Logging.Contracts

4.3.2.17 SoCrm.Services.Logging.Provider

Die WCF Service Application SoCrm.Services.Logging.Provider beinhaltet den LoggingService welcher das Service-Interface ILoggingService aus dem Namespace SoCrm.Services.Logging.Contracts implementiert. Die Service Application nutzt dafür den Persistenzservice ILogEventPersistenceService aus dem Infrastructurelayer und implementiert die Geschäftslogik für den Umgang mit Logeinträgen.

4.3.2.18 SoCrm.Services.Security.Contracts

Die Assembly SoCrm.Services.Security.Contracts definiert das Service-Interface ISecurityService (siehe Kapitel 3.4.1.3 auf Seite 68) und auch sämtliche Entitäten, welche mit der Benutzerverwaltung verbunden sind.

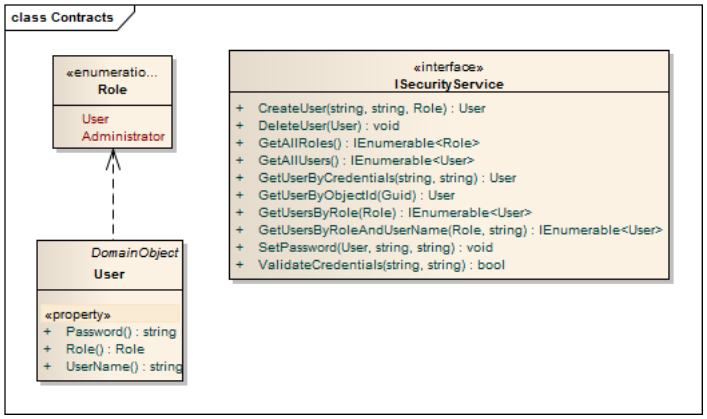

Abbildung 4.12: Klassendiagramm SoCrm.Services.Security.Contracts

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 enthält die Geschäftslogik für den Umgang mit Benutzern des Systems SoCrm. Der SecurityService erhält seine Objekte aus dem Persistenzservice IUserPersistenceService, welcher im Infrastructure-layer implementiert ist.

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.

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 hilft 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 es die OR-Mapper NHibernate und Entity Framework einen eigenen Datencontainer zur Verfügung ist, der sehr aufwendig zu mocken ist und der Persistenzlayer implizit in den Akzeptanztest (siehe Kapitel 5.2 auf Seite 100) mitgetestet wird.

Wie in Tabelle 5.1 ersichtlich ist, beträgt die totale Testabdeckung des Systems (ohne den Persistenzlayer) 85%. Laut [Cor11] ist eine Code Coverage von 70-80% eine angemessene Abdeckung mit automatisierten Unit Tests. Abbildung 5.1 enthält als Beweis einen Screenshot der ausgeführten Testsuite von SoCrm.

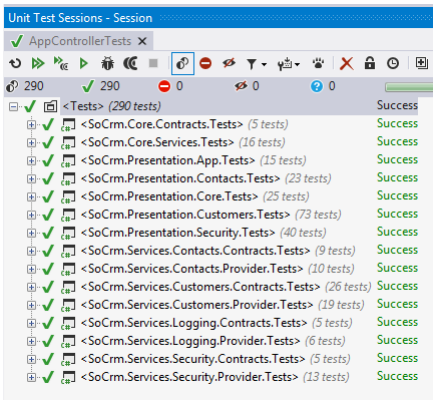

Abbildung 5.1: Resultat Unit Tests

Namespace	Codeblöcke	Getestet (Blöcke)	Getestet (% Blöcke)
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

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 2.4 auf Seite 18) geprüft, das heisst der Test hat keine Kenntnisse über die innere Funktionsweise des Systems und imitiert den Benutzer der Applikation.

In diesem Projekt werden zur Entwicklung der Akzeptanztests Coded UI-Tests[2] verwendet. Coded UI-Tests 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 die Nachricht-Textbox leer sein).

Die Anforderungen in den Use-Cases UC1 bis UC14 aus den Use-Case-Spezifikationen werden als Coded UI-Tests ausformuliert und im Visual Studio Projekt SoCrm.Tests.AcceptanceTests abgelegt. Beim Ausführen eines der Coded UI-Tests wird automatisch das System 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. kann der eben erstellte Kontakt nicht gefunden werden) so schlägt der Test fehl.

Die Ergebnisse der Akzeptanztests sind in Tabelle 5.2 und als Screenshot in Abbildung 5.2 ersichtlich. Alle Anforderungen können im Rahmen der Implementierung des Proof of Concepts erfüllt werden und die Akzeptanztests werden somit als bestanden betrachtet.

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

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

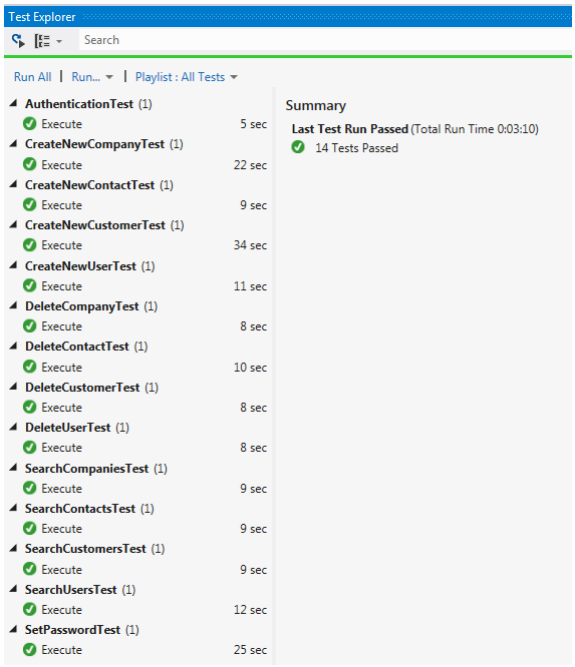

Abbildung 5.2: Resultat Akzeptanztests

Bezeichner	Use-Case	Testklasse	Testergebnis
UC1	Benutzer oder Administrator authentifizieren	AuthenticationTest	bestanden
UC2	Kunden anzeigen	SearchCustomersTest	bestanden
UC3	Kunde hinzufügen	CreateNewCustomerTest	bestanden
UC4	Kunde löschen	DeleteCustomerTest	bestanden
UC5	Firmen anzeigen	SearchCompaniesTest	bestanden
UC6	Firma hinzufügen	CreateNewCompanyTest	bestanden
UC7	Firma löschen	DeleteCompanyTest	bestanden
UC8	Kontakte anzeigen	SearchContactsTest	bestanden
UC9	Kontakt hinzufügen	CreateNewContactTest	bestanden
UC10	Kontakt löschen	DeleteContactTest	bestanden
UC11	Benutzer anzeigen	SearchUsersTest	bestanden
UC12	Passwort setzen	SetPasswordTest	bestanden
UC13	Benutzer hinzufügen	CreateNewUserTest	bestanden
UC14	Benutzer löschen	DeleteUserTest	bestanden

Tabelle 5.2: Akzeptanztests

6 Fazit

Die Bearbeitung dieser Bachelorarbeit war eine ausserordentlich spannende und lehrreiche Aufgabe für welche in diesem Kapitel ein Fazit erstellt wird. Das Kapitel enthält die Schlussfolgerungen welche aus den erarbeiteten Arbeitsergebnissen gezogen werden. Ausserdem werden die Abweichungen von der initialen Aufgabenstellung dargelegt und erläutert.

6.1 Schlussfolgerungen

In Anbetracht der in den vorherigen Kapiteln gewonnenen Erkenntnissen im detaillierten Vergleich von objektrelationalen Mappern und den erstellten Produktempfehlungen werden in diesem Kapitel Schlussfolgerungen im Vergleich und der Verwendung der vorgestellten OR-Mapper gezogen.

Wie schon im Vergleich der OR-Mapper (siehe Kapitel 3.3 auf Seite 55) und den Produktempfehlungen (siehe Kapitel 4.2 auf Seite 80) erkennbar, sind sich die heute verfügbaren und in dieser Arbeit betrachteten OR-Mapper funktions- und leistungsmässig sehr ähnlich. Es fällt schwer, konkrete Differenzierungsmerkmale der einzelnen Produkte herauszuheben, welche sich dann auch bewerten lassen. Es wurde erkannt, dass weniger die betrachteten Produkte ein Auswahlkriterium bei der Verwendung von bestimmten Enterprise Pattern sind, sondern viel mehr der Typ des ORMs. Wie beschrieben, existieren zwei Arten von ORMs: Komplette und leichtgewichtige OR-Mapper. Alle in der Marktübersicht (siehe Kapitel 3.1 auf Seite 40) analysierten OR-Mapper lassen sich einem der zwei Typen zuteilen und der Typ ist innerhalb der Produktempfehlung der ausschlaggebende Faktor für die Empfehlung des Produkts. Für alle Enterprise Pattern ausser dem Domain Model lässt sich sagen, dass leichtgewichtige OR-Mapper besser geeignet sind als komplette. Wird allerdings ein Domain Model verwendet, so macht die Verwendung eines kompletten OR-Mappers Sinn. Die meisten serviceorientierten Systeme setzten heute ein Domain Model ein und somit lässt sich auch in den meisten dieser Systeme (im .NET-Umfeld) einer der in der Marktübersicht betrachteten, kompletten OR-Mapper finden.

In der nahen Zukunft werden in Enterpriseapplikationen verstärkt Domain Models verwendet werden, da Entwickler es heutzutage gewohnter sind mit Objekten zu arbeiten als mit tabellarischen Daten. Auch der Trend in Richtung objektrelationaler Datenbankmanagementsysteme wird die Verwendung von komplexen Domänenmodellen fördern und eventuell sogar OR-Mapper überflüssig machen. Trotzdem sieht der Autor dieser Bachelorarbeit besonders bei bedienerfreundlichen OR-Mappern wie dem Entity Framework grosses Potential in naher Zukunft.

Auch leichtgewichtige OR-Mapper wie Dapper werden in performancekritischen Systemen und in Systemen mit einer einfacheren Domänenlogik eine Daseinsberechtigung haben und wohl noch an Bedeutung gewinnen. Wird sich Dapper in den Möglichkeiten des Monitorings und im Bedienungskomfort näher an die kompletten OR-Mapper wie NHibernate oder dem Entity Framework angleichen, ist der Autor durchaus zuversichtlich, dass in dieser Nische ein Potential für Dapper vorhanden ist.

6.2 Abweichungen von der Aufgabenstellung

Wie in [Ams13b] erwähnt, wurde die initiale Aufgabenstellung in mehreren Punkten geändert was sich auch in der Dokumentation der Bachelorarbeit sowie den gewonnenen Resultaten widerspiegelt. Alle Änderungen sind in der folgenden Liste noch einmal erwähnt und beschrieben:

Empfehlung nach Architekturstil Die ursprüngliche Aufgabenstellung sah einen Vergleich von OR-Mappern auf Basis von Architekturstilen (siehe [SG96]) vor. Da das Buch schwer auffindbar war und die Architekturstile darin auf nur wenigen Seiten abgehandelt wurden, wurde die Aufgabenstellung geändert und es wurde eine Evaluation auf Basis von Martin Fowlers Enterprise Pattern durchgeführt.

Verbreitung ORM Die ursprüngliche Aufgabenstellung sah vor, die Auswahl der näher zu betrachtenden ORMs nach deren Verbreitung zu treffen. Es wurde allerdings keine Studie zu diesem Thema gefunden und darum wurde in der finalen Version eine Umfrage unter Entwicklern durchgeführt um die Selektion der genauer zu betrachtenden OR-Mapper zu treffen.

Zu breiter Scope In der ersten Version der Aufgabenstellung sollten alle frei verfügbaren und kommerziellen OR-Mapper im Java- und .NET-Umfeld miteinander verglichen werden. Da diese Marktübersicht zu viele Produkte umfasst hätte, wurden in der aktuellen Version der Aufgabenstellung nur noch frei verfügbare und Open Source .NET-ORMs verglichen. Desweiteren sollte die serviceorientierte CRM-Applikation auch eine Reporting-Funktionalität umfassen welche nachträglich aus der Aufgabenstellung entfernt worden ist.

Fokus Bachelorarbeit Der Fokus der ursprünglichen Aufgabenstellung sah vor, dass die Empfehlung der OR-Mappers für Enterprise Pattern im Vordergrund steht. Auf Grund der erkannten Schwierigkeit des Erreichen dieses Zieles zum Zeitpunkt des Design Reviews wurde beschlossen mehr Fokus auf die serviceorientierte CRM-Applikation als Proof of Concept zu legen.

Akronyme

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
ASP	Active Server Pages
AT	Acceptance Test
CRM	Customer Relationship Management
CRUD	Create, Read, Update and Delete
DB	Datenbank
DBMS	Database Management System
DDL	Data Definition Language
EBS	Einschreibe- und Bewertungssystem
EF	Entity Framework
eSQL	Entity SQL
GUI	Graphical User Interface
Guid	Globally Unique Identifier
HQL	Hibernate Query Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IIS	Internet Information Server
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
LINQ	Language Integrated Query
MSDN	Microsoft Developer Network

MSSQL	Microsoft SQL Server
MVC	Model, View, Controller
MVVM	Model, View, ViewModel
MVVM+C	Model, View, ViewModel plus Controller
OR	Objektrelational
ORM	Objektrelationaler Mapper
POCO	Plain Old CLR Object
RAM	Random Access Memory
RUP	Rational Unified Process
SL	Silverlight
SOA	Service Oriented Architecture
SoCRM	Service Oriented CRM
SQL	Structured Query Language
SQLCE	SQL Server Compact
SSD	Solid State Disk
SVC	Service
UAT	User Acceptance Test
UC	Use-Case
UI	User Interface
UML	Unified Modelling Language
UoW	Unit of Work
UT	Unit Test
VB	Visual Basic
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
WSDL	Web Services Description Language
WWW	World Wide Web

XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XP	Extreme Programming
ZHAW	Zürcher Hochschule für Angewandte Wissenschaften

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Ams13a] Amstutz, Florian: Aufgabenstellung Bachelorarbeit. (2013)

[Ams13b] Amstutz, Florian: Design Review Bachelorarbeit. (2013)

[CdE+10] Chou, David ; deVadoss, John ; Erl, Thomas ; Gandhi, Nitin ; Kommalapati, Hanu ; Loesgen, Brian ; Schittko, Christoph ; Wilhelmsen, Herbjörn ; Williams, Mickey: *SOA with .NET and Windows Azure.* Prentice Hall, 2010

[Cor11] Cornett, Steve: Minimum Acceptable Code Coverage. (2011). `http://www.bullseye.com/minimum.html`

[Elm05] Elmer, Franz-Josef: Software Engineering, Methodologien. (2005)

[FRF+02] Fowler, Martin ; Rice, David ; Foemmel, Matthew ; Hieatt, Edward ; Mee, Robert ; Stafford, Randy: *Patterns of Enterprise Application Architecture.* Addison Wesley, 2002

[GHJV94] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, 1994

[HS13] Hruschka, Peter ; Starke, Gernot: arc42 - Resourcen für Software-Architekten. (2013)

[Hun07] Hung, Vo-Trung: Software Development Process. (2007). `http://cnx.org/content/m14619/1.2/`

[IEE90] IEEE: Standard Glossary of Software Engineering Terminology. (1990)

[Koc11] Kochetov, Alexis: ORMBattle.NET. (2011). `http://www.ormbattle.net/`

[LSL+07] Liebhart, Daniel ; Schmutz, Guido ; Lattmann, Marcel ; Heinisch, Markus ; Könings, Michael ; Kölliker, Mischa ; Pakull, Perry ; Welkenbach, Peter: *Architecture Blueprints.* Carl Hanser Verlag, 2007

[OSKZ06] Oestereich, Bernd ; Schröder, Claudia ; Klink, Markus ; Zockoll, Guido: *EP - oose Engineering Process: Vorgehensleitfaden für agile Softwareprojekte.* dpunkt.verlag, 2006

[PR11] Pohl, Klaus ; Rupp, Chris: *Basiswissen Requirements Engineering.* dpunkt.verlag, 2011

[PT06] Pulier, Eric ; Taylor, Hugh: *Understanding Enterprise SOA*. Manning Publications, 2006

[Roc11] Rocha, Luis: Data Access Performance Comparison in .NET. (2011). `http://www.luisrocha.net/2011/12/data-access-performance-comparison-in.html`

[Sch08] Schwichtenberg, Holger: Die Qual der Wahl. In: *dotnetpro* (2008)

[SG96] Shaw, Mary ; Garlan, David: *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996

[Ski12] Skimp: MVVM is dead, long live MVVMC! (2012). `http://skimp-blog.blogspot.ch/2012/02/mvvm-is-dead-long-live-mvvmc.html`

[Sta11] Starke, Gernot: *Effektive Software-Architekturen*. Carl Hanser Verlag, 2011

[Ste12] Stern, Olaf: Reglement Bachelorarbeit. (2012)