

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 selbständig ausgeführt wurde.

Zürich, 10. Juni 2013

Florian Amstutz

Inhaltsverzeichnis

1	Einführung	6
1.1	Management Summary	6
1.2	Über die Bachelorarbeit	6
1.3	Ziele der Arbeit	7
1.4	Vorgehensweise	7
1.5	Projektplanung	8
1.5.1	Phasenplanung	9
1.5.2	Meilensteine	9
2	Anforderungen	11
2.1	Was sind Anforderungen?	11
2.1.1	Arten von Anforderungen	11
2.2	Anforderungen Persistenzlayer	12
2.2.1	Funktionale Anforderungen	12
2.2.2	Qualitätsanforderungen	13
2.2.3	Randbedingungen	14
2.3	Anforderungen Enterprise Pattern	14
2.3.1	Transaction Script	15
2.3.2	Domain Model	16
2.3.3	Table Module	17
2.3.4	Active Record	18
2.4	Anforderungen serviceorientierte CRM-Applikation	19
2.4.1	Systemkontext	19
2.4.2	Use-Case-Spezifikationen	20
2.4.2.1	Benutzer oder Administrator authentifizieren	21
2.4.2.2	Kunden anzeigen	22
2.4.2.3	Kunde hinzufügen	23
2.4.2.4	Kunde löschen	25
2.4.2.5	Firmen anzeigen	26
2.4.2.6	Firma hinzufügen	27
2.4.2.7	Firma löschen	28
2.4.2.8	Kontakte anzeigen	30
2.4.2.9	Kontakt hinzufügen	31
2.4.2.10	Kontakt löschen	32
2.4.2.11	Benutzer anzeigen	33
2.4.2.12	Passwort setzen	34

Inhaltsverzeichnis

2.4.2.13	Benutzer hinzufügen	36
2.4.2.14	Benutzer löschen	37
3	Konzept und Architektur	39
3.1	Marktübersicht	39
3.1.1	Vergleichskriterien	39
3.1.2	Vergleichsmatrix	46
3.2	Produkteauswahl	49
3.2.1	Umfrage	49
3.2.2	Fragebogen	49
3.2.3	Interpretation des Umfrageergebnis	50
3.2.4	Resultat	53
3.3	Vergleich ausgewählter Produkte	54
3.3.1	Benutzerfreundlichkeit	54
3.3.2	Plattformunterstützung	55
3.3.3	Performance	57
3.3.4	Toolunterstützung	58
3.3.5	Unterstützung Enterprise Pattern	60
3.3.6	Monitoring	60
3.3.7	Vergleichsmatrix	62
3.4	Architektur serviceorientierte CRM-Applikation	62
3.4.1	Bausteinsicht	62
3.4.1.1	Komponentendiagramm	63
3.4.1.2	Domänenmodell	63
3.4.1.3	Service Contracts	65
3.4.2	Laufzeitsicht	70
3.4.2.1	Kontakt hinzufügen	70
3.4.3	Verteilungssicht	73
3.4.3.1	Verteilungsdiagramm	73
4	Implementierung	75
4.1	Evaluation selektierte OR-Mapper	75
4.1.1	Entity Framework	75
4.1.2	Dapper	76
4.1.3	NHibernate	77
4.2	Produkteempfehlung	78
4.2.1	Transaction Script	78
4.2.2	Domain Model	78
4.2.3	Table Module	78
4.2.4	Active Record	79
4.2.5	Abschliessende Bemerkung	79
4.3	Serviceorientierte CRM-Applikation	79
4.3.1	Implementierungsprinzipien	79

Inhaltsverzeichnis

4.3.2	Komponenten im Detail	80
4.3.2.1	SoCrm.Core.Contracts	80
4.3.2.2	SoCrm.Core.Services	80
4.3.2.3	SoCrm.Infrastructure.Persistence.Contracts	82
4.3.2.4	SoCrm.Infrastructure.Persistence.Dapper	82
4.3.2.5	SoCrm.Infrastructure.Persistence.EntityFramework	83
4.3.2.6	SoCrm.Infrastructure.Persistence.NHibernate	83
4.3.2.7	SoCrm.Presentation.App	83
4.3.2.8	SoCrm.Presentation.Contacts	84
4.3.2.9	SoCrm.Presentation.Core	85
4.3.2.10	SoCrm.Presentation.Customers	86
4.3.2.11	SoCrm.Presentation.Security	88
4.3.2.12	SoCrm.Services.Contacts.Contracts	90
4.3.2.13	SoCrm.Services.Contacts.Provider	90
4.3.2.14	SoCrm.Services.Customers.Contracts	90
4.3.2.15	SoCrm.Services.Customers.Provider	91
4.3.2.16	SoCrm.Services.Logging.Contracts	91
4.3.2.17	SoCrm.Services.Logging.Provider	92
4.3.2.18	SoCrm.Services.Security.Contracts	92
4.3.2.19	SoCrm.Services.Security.Provider	93
5	Verifikation	94
5.1	Unit-Tests	94
5.1.1	Testabdeckung	94
5.2	Akzeptanztests	96
6	Fazit	98
6.1	Schlussfolgerungen	98
6.2	Abweichungen von der Aufgabenstellung	98

1 Einführung

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

1.1 Management Summary

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

Heutige serviceorientierte Architekturen verwenden fast in jedem Fall einen Persistenzlayer welcher Objekte in eine Datenbank speichert und sie von dort liest. Diese Aufgabe wird meist nicht von der Applikation selber übernommen sondern durch einen OR-Mapper ausgeführt. Es gibt eine Reihe von Einflussfaktoren und Entscheidungskriterien die beim Entwurf und der Implementierung von Persistenzkomponenten beachtet werden müssen. Die Auswahl des passenden OR-Mappers für die jeweilige Architektur ist eine der zentralen Fragen, die zu Beginn des Implementierungszyklus einer Applikation beantwortet werden muss (nach [Sta11]), denn oft ist der Persistenzlayer das Nadelöhr einer Enterprise Applikation (gemäss [LSL⁺07])

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.

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]) erarbeitet, welche heutzutage in jedem grösseren (Software-) System eine entscheidende Rolle spielen. Es wird pro Enterprise Pattern eine Empfehlung abgegeben werden, welcher frei verfügbare und Open-Source OR-Mapper im .NET-Umfeld eingesetzt werden soll.

Zusätzlich zu diesen Vergleichen und den Empfehlungen wird eine serviceorientierte CRM-Applikation als Proof-of-Concept erstellt. Für diese CRM-Applikation werden Anforderungen erhoben, es wird ein Konzept erstellt 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 nach [Sta11] einer der Hauptgründe wieso Softwareprojekte mit Qualitätsmängeln, Budgetüberschreitungen oder zeitlichen Verzögerungen zu kämpfen haben.

Für dieses Projekt wurde das Wasserfallmodell als Entwicklungsprozess ausgewählt. Das Wasserfallmodell teilt die Softwareentwicklung meist in fünf verschiedene Phasen ein (siehe Abbildung 1.1). Dabei kann jeweils erst mit der nächsten Phase begonnen werden wenn die Lieferergebnisse und die Ergebnisdokumentation der vorhergehenden Phase fertiggestellt und abgenommen worden sind. Das Wasserfallmodell wurde ausgewählt, da dieses Projekt gut linear abgearbeitet werden kann und da der Betreuer als einziger externer Stakeholder des Projekts zu festdefinierten Zeitpunkten (namlich 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 nur in der Anforderungs- und Konzeptphase Einfluss auf das Projekt nehmen kann und der Student alleinig den Abschluss der Phasen steuert und verifiziert sowie den Ablauf der Phasen innerhalb des Projekts steuert, können diese Nachteile entschärft werden.

Zu Beginn des Wasserfallmodells steht das Sammeln und Dokumentieren der Anforderungen (Requirements) im Vordergrund. Wenn die Anforderungen umfänglich und in hohem Detaillierungsgrad niedergeschrieben sind, werden diese vom Auftraggeber abge-

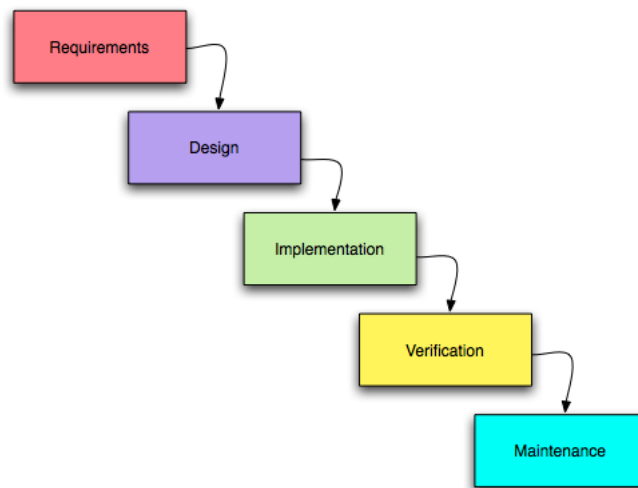


Abbildung 1.1: Wasserfallprozess nach [Hun07]

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

Auch die Struktur dieses Projekts und der dazugehörigen Dokumentation hält sich an den Wasserfallprozess nach [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 Prototypenstadium befinden wird und nicht dem Reifegrad einer Applikation entspricht, die in die Wartung übergehen kann.

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.

1 Einführung

Phase	Kapitelüberschrift	Siehe Seite
Requirements	Anforderungen	11
Design	Konzept und Architektur	39
Implementation	Implementierung	75
Verification	Verifikation	94

Tabelle 1.1: Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument

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 überhastigtes 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 wurden so modelliert, dass ihr Endzeitpunkt möglichst mit dem Erreichen eines Meilensteins zusammenfällt (siehe Tabelle 1.2). Das heisst, dass bei Erreichen des Meilensteins die vorhergehende Phase zwingend abgeschlossen sein muss.

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

1.5.2 Meilensteine

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

Die Meilensteine des Projekts sind in der Tabelle 1.3 ersichtlich.

1 Einführung

Phase	Start	Ende	Endet in Meilenstein
Themenevaluation	02.10.2012	16.11.2012	
Erstellung der Aufgabenstellung	17.11.2012	07.01.2013	M1
Erfassen der Anforderungen	08.01.2013	30.01.2013	M2
Erstellen der Marktübersicht	31.01.2013	02.03.2013	
Auswahl und Vergleich ausgewählter OR-Mapper	03.03.2013	28.03.2013	
Implementierung Proof of Concept	29.03.2013	22.05.2013	M3
Erstellung Produktempfehlung	23.05.2013	01.06.2013	
Finalisierung Dokumentation	02.06.2013	17.06.2013	M4
Erarbeiten der Präsentation	18.06.2013	01.07.2013	M5

Tabelle 1.2: Phasenplan

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

Tabelle 1.3: Meilensteine

2 Anforderungen

In der Phasenplanung (siehe Kapitel 1.5.1 auf Seite 9) wurde festgelegt, dass in der ersten Projektphase die Anforderungen erhoben 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 der betrachteten Enterprise Pattern erhoben. In einem letzten Schritt wird der Systemkontext der CRM-Applikation beschrieben und eingegrenzt sowie die konkreten Anforderungen an die CRM-Applikation als Use-Cases modelliert und spezifiziert.

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.
- Randbedingungen¹ 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 sind auch gleichzeitig auch Teil der Anforderungen der CRM-Applikation (siehe Kapitel 2.4 auf Seite 19).

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, diese im Rahmen dieser Bachelorarbeit jedoch 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], [DC10] 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

2 Anforderungen

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 mehrer 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 einen Persistenzlayer

2.2.2 Qualitätsanforderungen

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

Die Abgrenzung zwischen funktionalen und nicht-funktionalen Anforderungen² ist unscharf. Sie 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 bei sämtlichen 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 Performancewerte erfüllen ³ .

Tabelle 2.2: Qualitätsanforderungen an den Persistenzlayer

²Wie Qualitätsanforderungen auch noch genannt werden

³Im Rahmen dieser Arbeit werden die Performanceanforderungen nicht genauer definiert, da insbesondere das Überprüfen deren 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. Die Randbedingungen sind Umstände, die zwingend erfüllt werden müssen. Wiederum ist die Abgrenzung von den Qualitätsanforderungen schwierig, da der Übergang zwischen den zwei Anforderungstypen fließend ist.

Die Randbedingungen sind Implementierungsvorgaben welche massgeblich dafür verantwortlich sind, während der Erarbeitung und Umsetzung des Konzepts einen austauschbaren Persistenzlayer zu erhalten, wie dies in der Aufgabenstellung der Bachelorarbeit (siehe [Ams13]) 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 Technologie 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 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], [DC10] 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 11) beschrieben, werden die Anforderungen an Enterprise Pattern rein auf funktionaler Ebene beschrieben.

Neben den konkreten Anforderungen werden die einzelnen Enterprise Pattern auch noch im Detail 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 75) berücksichtigt auch die 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 absolut atomar ausgeführt werden. Der Entwickler muss sich nicht um konkurrierende Transaktionen kümmern, da diese jeweils im Kontext eines anderen Transaction Scripts ausgeführt werden. Je kompliziert die Geschäftslogik jedoch wird, je schwieriger wird es diese in einem funktionalen und lesbaren 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 das all 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 jedoch bald als nahezu unmöglich dar.

Das Transaction Script ist meist mit dem Command Pattern aufgebaut⁴. 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 simple 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

⁴Mehr 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 man 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 sehr 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 sehr 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 sehr 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 mehr oder minder aufwändigen Data Mapper⁵, der meist 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 Benutzerbarkeit macht meist den Nachteil des Modellierens wett. Gewöhnt man sich einmal an die Vorteile eines Domain Models so ist es sehr schwierig wieder zurück zu einfacheren Datenbankzugriffsmechanismen zu wechseln, wie zum Beispiel dem Transaction Script.

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

⁵Der Data Mapper ist auch ein Enterprise Pattern. Mehr dazu unter <http://martinfowler.com/eaCatalog/dataMapper.html>.

2 Anforderungen

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 wird.
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 dazugehörigen 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 die verschiedenen 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 liegt darin Unterschied, 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)` aufgerufen werden. Jedes Mal wenn mit diesem 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 wäre zum Beispiel, dass das Table Module bereits mit einem Record Set⁶ oder einem bestehenden Resultat eines Queries initialisiert werden könnte. Dieses Resultat könnte dann mit der Instanz des Table Modules manipuliert werden.

⁶Mehr zum Record Set auf <http://martinfowler.com/eaCatalog/recordSet.html>

2 Anforderungen

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 alle 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 und enthält die Geschäftslogik im Zusammenhang mit dieser Zeile. Ein Objekt enthält Daten und Verhalten. Der Gross-
teil 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 sein und datenorientierter Code im jeweiligen Active Record.

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 neue Instanz des Active Records welche später in die Datenbanktabelle gespeichert wird
- Statisch Suchmethode welche oft genutzte SQL Queries kapselt und als Active Records Objekte zurückgibt
- Aktualisieren des Active Records mit den aktuellen Werten aus der Datenbank
- Lesen und Speichern der Felder (Getter- und Setter-Methoden)

2 Anforderungen

- Implementieren von Teilen der Geschäftslogik

Active Record ist eine gute Option für relativ simple Geschäftslogik. Validierungen auf Basis eines einzelnen Datensatzes funktioniert 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, Vererbung oder Listen hinzugefügt wird das Active Record schnell unübersichtlich und ein Data Mapper ist 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 verwenden.
A3	Das System muss eine einfache Domänenlogik verwenden.

Tabelle 2.7: Anforderungen Active Record

2.4 Anforderungen serviceorientierte CRM-Applikation

Ein Teil dieser Bachelorarbeit besteht darin ein Proof-of-Concept einer serviceorientierten Architektur 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 entwickelt wird. Die folgenden Kapitel beschreiben daher 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]).

2 Anforderungen

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.

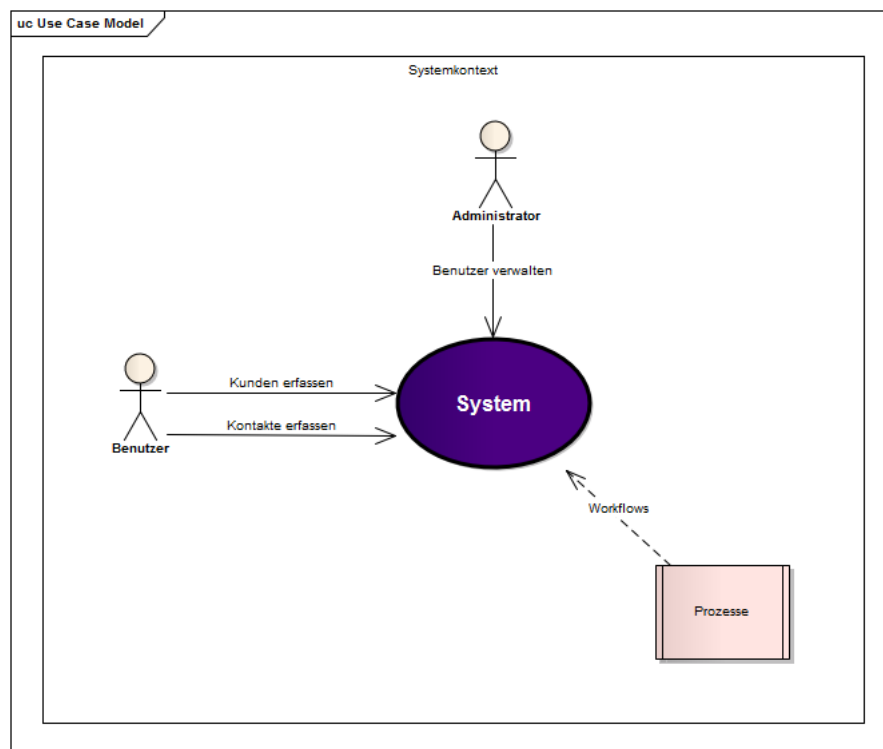


Abbildung 2.1: Systemkontext

2.4.2 Use-Case-Spezifikationen

Nach [PR11] zeigen Use-Case-Diagramme 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

2 Anforderungen

Use-Case-Diagramm textuell als Use-Case-Spezifikation festgehalten.

Alle funktionalen Anforderungen (siehe Kapitel 2.1.1 auf Seite 11) werden als Use-Cases modelliert und spezifiziert⁷. 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.

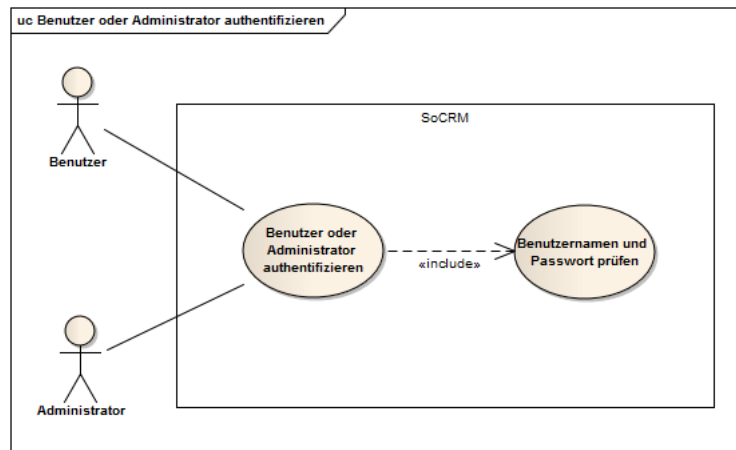


Abbildung 2.2: Use-Case Benutzer oder Administrator authentifizieren

⁷Die 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.

2 Anforderungen

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 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.

2 Anforderungen

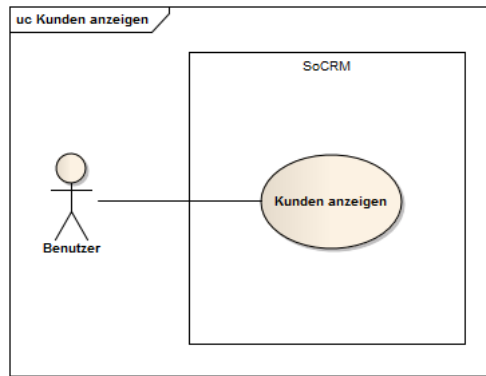


Abbildung 2.3: Use-Case Kunden anzeigen

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 passenden 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

2 Anforderungen

als Firma im System eingetragen, so kann sie beim Erstellen des Kunden hinzugefügt werden. Zusätzlich kann jedem Kunden eine oder mehrere E-Mail-Adressen sowie eine oder mehrere Telefonnummern hinzugefügt werden.

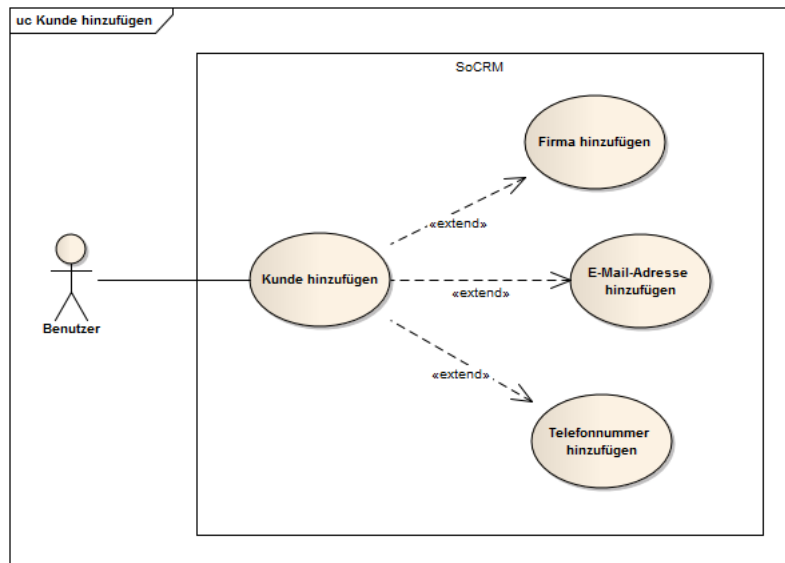


Abbildung 2.4: Use-Case Kunde hinzufügen

2 Anforderungen

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 27).
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 unwiderbringlich verloren und können nicht wiederhergestellt werden.

2 Anforderungen

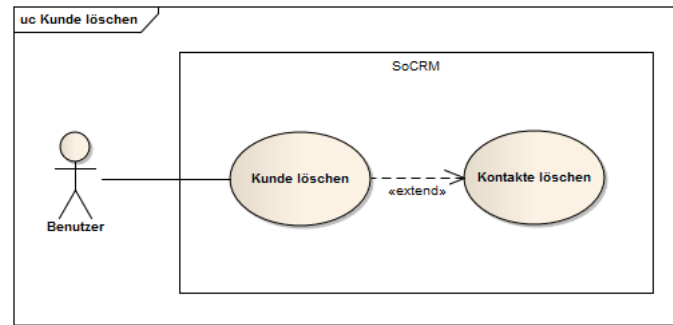


Abbildung 2.5: Use-Case Kunde löschen

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, mit welchen 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.

2 Anforderungen

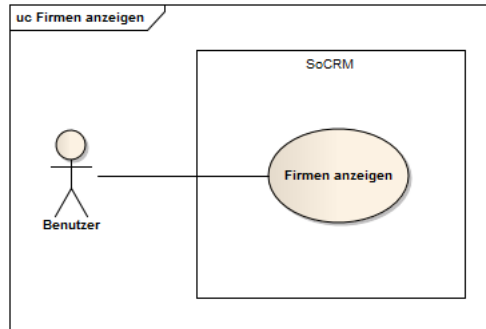


Abbildung 2.6: Use-Case Firmen anzeigen

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 Kunden hinzufügen oder die Firma löschen.
Ergebnis	Eine Liste mit Firmen wird angezeigt.
Hauptszenario	1. Der Benutzer gibt den Namen der zu suchenden Firma ein. 2. Der Benutzer gibt das Land der zu suchenden Firma ein. 3. Das System zeigt dem Benutzer die passenden Firmen an.
Alternativszenarien	1a. Der Benutzer gibt keinen Namen ein. 2a. Der Benutzer gibt kein Land an.
Ausnahmeszenarien	Keine.

Tabelle 2.12: 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.

2 Anforderungen

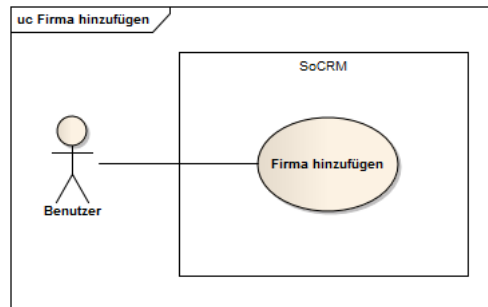


Abbildung 2.7: Use-Case Firma hinzufügen

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 unwider-

2 Anforderungen

bringlich verloren.

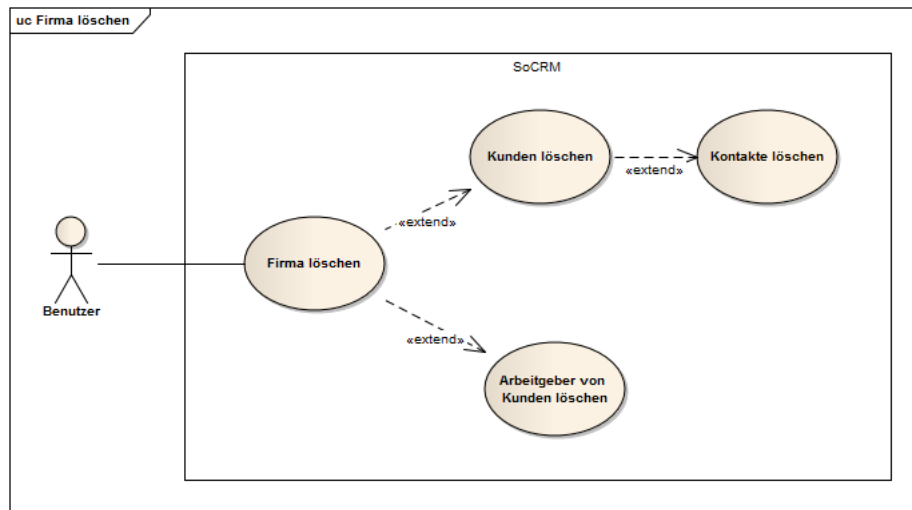


Abbildung 2.8: Use-Case Firma löschen

2 Anforderungen

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 26).
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 zustande gekommen ist (mögliche Suchkriterien: Vor- oder Nachname des Kunden, Kontaktmedium). Die gefundenen Kontakte kann der Benutzer entweder im Detail anzeigen lassen oder diese aus dem System löschen.

2 Anforderungen

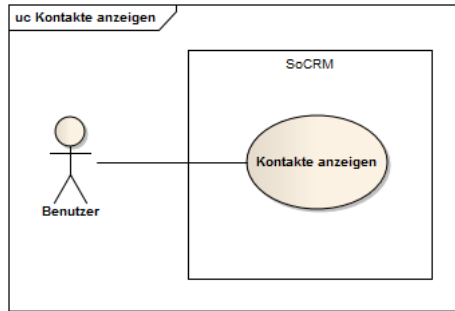


Abbildung 2.9: Use-Case Kontakte anzeigen

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 Kontaktmedium⁸ sowie die textuelle Beschreibung des Kontakt.

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

2 Anforderungen

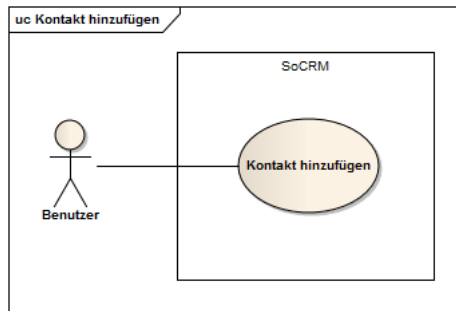


Abbildung 2.10: Use-Case Kontakt hinzufügen

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

Tabelle 2.16: Use-Case-Spezifikation Kontakt hinzufügen.

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.

2 Anforderungen

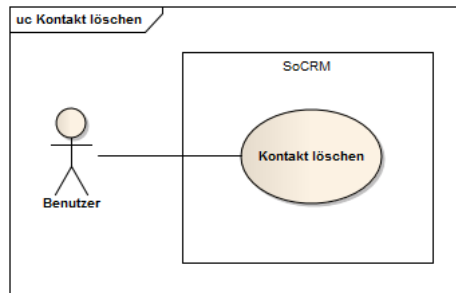


Abbildung 2.11: Use-Case Kontakt löschen

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 30).
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.

2 Anforderungen

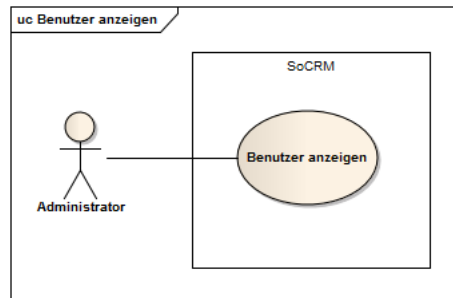


Abbildung 2.12: Use-Case Benutzer anzeigen

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

Tabelle 2.18: 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 angegeben wird. Nach dem Setzen des eigenen Passworts wird der Benutzer ausgeloggt und muss sich neu einloggen mit seinem eben gesetzten Passwort.

2 Anforderungen

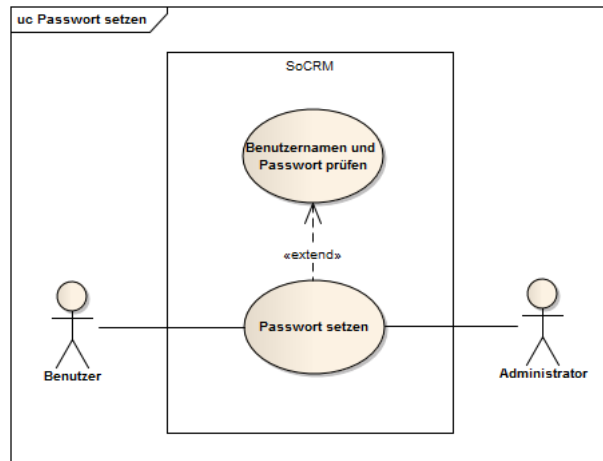


Abbildung 2.13: Use-Case Passwort setzen

2 Anforderungen

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 oder Administrator kann sich mit seinem neuen Passwort beim System authentifizieren.
Ergebnis	Eine Bestätigung wird angezeigt.
Hauptszenario	1. Der Benutzer gibt sein altes Passwort sowie das neue Passwort ein. 2. Das System setzt das neue Passwort.
Alternativszenarien	1a. Der Administrator gibt das neue Passwort eines Benutzers ein. 2a. Das alte Passwort des Benutzers ist falsch, der Benutzer wird aufgefordert das alte Passwort erneut einzugeben.
Ausnahmeszenarien	Keine.

Tabelle 2.19: 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 Administrator kann sich dieser beim System mit den Benutzerdaten anmelden.

2 Anforderungen

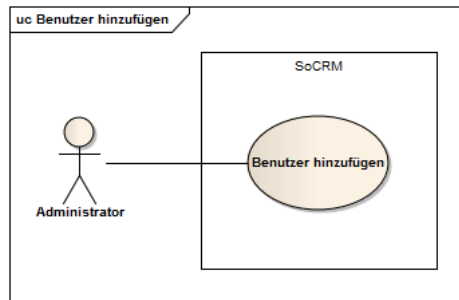


Abbildung 2.14: Use-Case Benutzer hinzufügen

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 des 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.

2 Anforderungen

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.

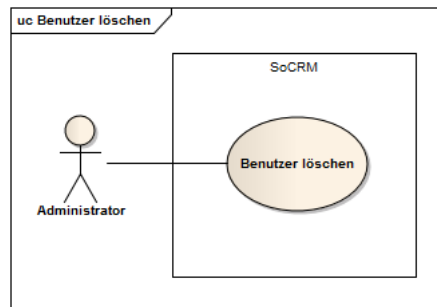


Abbildung 2.15: Use-Case Benutzer löschen

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 33). Der gelöschte Benutzer kann sich nicht mehr beim System authentifizieren (siehe Kapitel 2.4.2.1 auf Seite 21).
Ergebnis	Die Löschung wird bestätigt.
Hauptszenario	1. Der Administrator wählt den zu löschenden Benutzer aus. 2. Der Benutzer wird vom System gelöscht.
Alternativszenarien	Keine.
Ausnahmeszenarien	2a. Der Benutzer ist gegenwärtig auf dem System eingeloggt. Der Administrator wird gefragt, ob das System die Authentifizierung löschen soll.

Tabelle 2.21: Use-Case-Spezifikation Benutzer löschen

3 Konzept und Architektur

Die Konzeptphase¹ des Wasserfallmodells behandelt die Entwicklung eines vollständigen und umfassenden Lösungskonzepts auf Basis der dokumentierten Anforderungen (nach [OSKZ06]). Das Konzept dieser Bachelorarbeit umfasst das Erarbeiten der 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 Vergleichsergebnisse.

Zusätzlich wird die Architektur des Proof-of-Concepts 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 eine Auswahl von frei verfügbaren oder Open-Source OR-Mapper im .NET-Umfeld. Jeder OR-Mapper wurde 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 haben in welchem sie zusätzliche Funktionalitäten zur Verfügung stellen.

In diesem Kapitel werden auch die einzelnen Vergleichskriterien erklärt und beschrieben warum diese wichtig sind im Zusammenhang mit OR-Mappern. Das Ausfüllen der Marktübersicht erfolgt einerseits durch Konsultation der Dokumentation bzw. der Anbieterwebseite andererseits durch die Erfahrung in der Benutzung des OR-Mappers und unter Analyse des verfügbaren Quellcodes (wo Open-Source). Innerhalb der Marktübersicht wird nicht dokumentiert welche Information aus welcher 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 46) bieten.

¹ auch Designphase genannt

3 Konzept und Architektur

Die Vergleichskriterien, die in der Marktübersicht verwendet wurden, sind auf [Sch08] aufgebaut. Alle weiteren Kriterien stammen aus eigener Erfahrung beziehungsweise aus Literatur wie [Koc11] und [DC10].

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 wurde “Open-Source” verwendet.

Website Die Webseite des OR-Mappers.

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

Aktuelle und untersuchte Version Die im Rahmen dieser Marktübersicht untersuchte Version. Dies ist in jedem Fall die aktuellste verfügbare Version.

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

Typ Der Typ des OR-Mappers. Es gibt zwei unterschiedliche Typen von OR-Mapper: Leichtgewichtige und komplette OR-Mapper. Ein leichtgewichtiger OR-Mapper bietet meist ausser der Implementierung eines Data Mappers keine weitere Funktionalität 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.

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 ist. Deshalb sind in dieser Zeile nur Datenbankmanagementsysteme aufgelistet die auch so vom Hersteller des OR-Mappers getestet und freigegeben wurden.

Minimale .NET-Version Die minimal notwendige .NET Framework-Version um den OR-Mapper zu benutzen.

Unterstützt Mono Beschreibt ob der OR-Mapper die Softwareplattform Mono³ unterstützt.

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

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

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 noch zusätzliche kommandozeilenbasierte Tools mitliefert sind diese in dieser Zeile aufgelistet.

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 der relationalen Datenbank zu Domänenobjekten ermöglicht. Dies kann zum Beispiel Codegenerierung auf Grund eines Designers sein oder Techniken wie Reflection. Dieses Kriterium ist meist ein Anhaltspunkt um die mögliche 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 modelliert wird und danach die Datenbank vom OR-Mapper auf Basis dieses Domänenmodells aufgebaut wird. Dies setzt voraus, dass der OR-Mapper SQL DDL⁴ 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 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

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

eine Datenmigration ausgeführt, so wird das Datenschema inklusive den darin enthaltenen Daten automatisiert auf den Zielstand der Datenmigration gebracht. Im Rahmen dieser Marktübersicht gelten Datenmigrationen nur als unterstützt, wenn diese auch rückgängig gemacht werden können.

POCO Support POCOs sind Plain Old CLR Objects. Das heisst ein POCO ist ein simples Objekt welches ohne die Verwendung eines zusätzlichen Frameworks wie eine bestimmter Basisklasse, implementierten Interfaces oder Annotationen. 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 ableitet, 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 unterstützten, sind in dieser Zeile der Vergleichsmatrix die implementierten und unterstützten Interfaces aufgelistet.

Erzeugte Business Objects sind als [Serializable] gekennzeichnet Sind die generierten Domänenobjekte von Haus aus bereits mit dem SerializableAttribute⁵ gekennzeichnet, so wird dies unter diesem Kriterium 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

⁵Mehr zum SerializableAttribute in der MSDN unter <http://msdn.microsoft.com/en-us/library/system.serialization.serializeattribute.aspx>.

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

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

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 Eigenschaften⁷ wird dies in dieser Zeile eingetragen.

Unterstützung für nullbasierte Werte Unterstützt der OR-Mapper das Persistieren von nullbasierten Werten⁸ 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-Mapper 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.

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.

⁷Eigenschaften in englisch sind Properties.

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

Zusammengesetzte Schlüssel Unterstützt der OR-Mapper zusammengesetzte Schlüssel⁹ 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 Datenbanktabellen¹⁰ in den Domänenobjekten als Objektreferenzen darstellen, wird dies in dieser Zeile eingetragen.

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 Abfragesprachen 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.

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.

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

⁹Zusammengesetzte 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](http://msdn.microsoft.com/en-us/library/ms181043(v=sql.105).aspx).

¹⁰In 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](http://msdn.microsoft.com/en-us/library/ms177463(v=sql.105).aspx).

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 gesichert, 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 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 trotzdem die Änderungen am Domänenobjekt erkennen und wieder persistieren, so wird dies in dieser Zeile vermerkt.

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

Second Level Cache (Arbeitsspeicher/DB/...) Existiert ein Second Level Cache auf Basis des Arbeitsspeichers, der Datenbank oder des Festplatte so wird dies in dieser Zeile festgehalten.

Verteiltes Caching Unterstützt der OR-Mapper bei einem Multi-Server-Betrieb 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 Windows-Leistungsindikatoren 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.

[illegible]

3.2 Produkteauswahl

Das Ziel der Produkteauswahl 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 ausgewählt 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 wird auf Grund der Umfragergebnisse die Auswahl von drei OR-Mappern begründet.

Um die Umfrage einzugrenzen wurden nur die folgenden, in der Marktübersicht (siehe Kapitel 3.1 auf Seite 39) betrachteten, Produkte zur Auswahl zugelassen:

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

3.2.2 Fragebogen

Der Fragebogen, der den Entwicklern ausgeteilt wurde 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 verbreitesten ein?
5. Für ein privates Projekt, welchen ORM würdest du einsetzen?

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

3.2.3 Interpretation des Umfrageergebnis

Die nachfolgenden Kapitel zeigen das Umfrageergebnis für jede Frage grafisch auf und liefern den Versuch einer Interpretation dieser Grafiken und den darunterliegenden Umfragewerte.

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-2-SQL sind auch entsprechend bekannt. Überraschend ist, dass Dapper als relativ junges Projekt (erstes Releasedatum März 2011) schon solch eine Bekanntheit unter Entwicklern besitzt. Ich vermute das dies an den vielen Beiträgen auf Entwicklerplattformen wie StackOverflow oder CodeProject liegt.

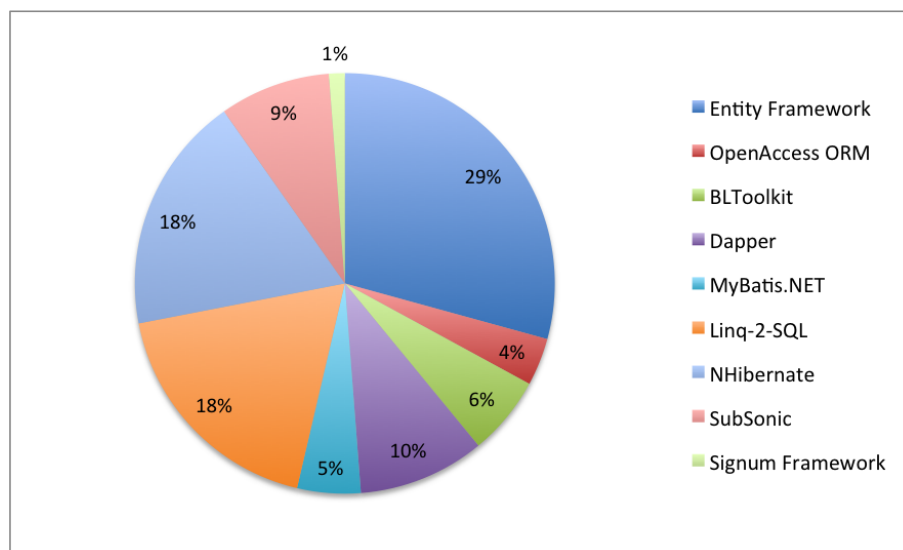


Abbildung 3.1: Umfrageresultate Welche der folgenden .NET OR-Mapper kennst du?

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-2-SQL nehmen zwei Drittel der Umfrageergebnisse ein. Vor dem Release des Entity Framework 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 Opportunitätskosten wohl auch keinen Wechsel des OR-Mappers planen.

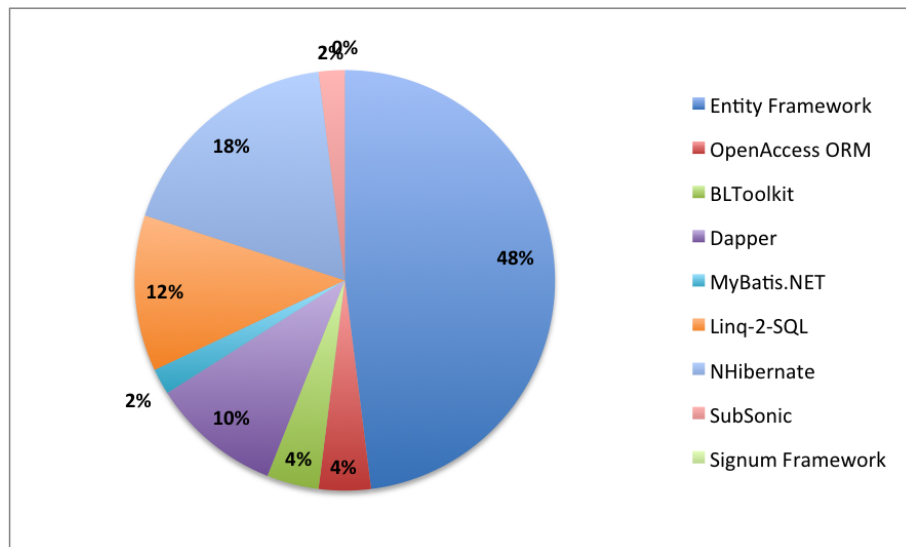


Abbildung 3.2: Umfrageresultate Welche davon hast du schon in einem Projekt eingesetzt?

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 sehr guten Dokumentaiton von Microsoft liegt und daran dass das Entity Framework das wohl entwicklerfreundlichste Proudkt 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 als Source-Code ausgeliefert wird sehr einfach zu debuggen und zu erweitern. Dies kann insbesondere für erfahrene Entwickler ein grosser Vorteil sein, da das Not-Invented-Here-Syndrom¹¹ unter Softwareentwicklern doch weit verbreitet ist.

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

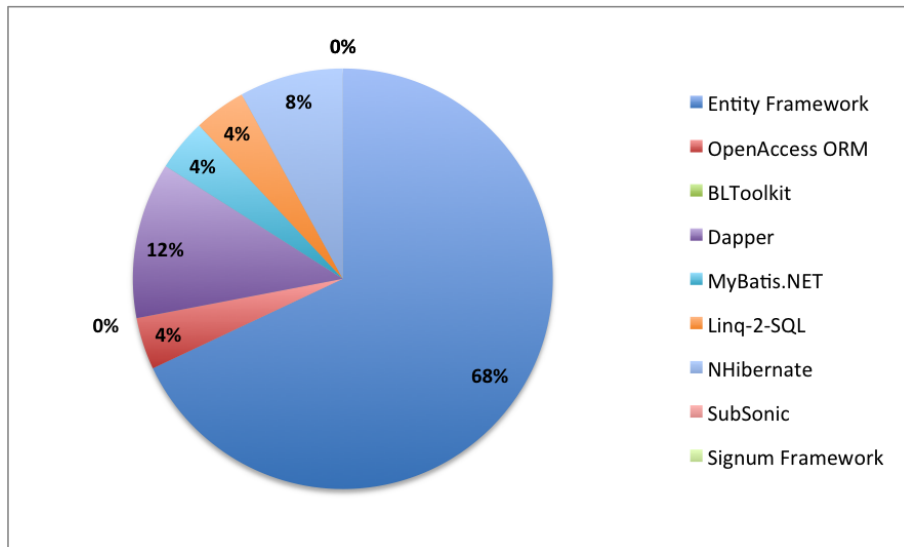


Abbildung 3.3: Umfrageresultate Mit welchem Produkt hast du die besten Erfahrungen gemacht?

Welches Produkt schätzt du am verbreitesten ein? Auf die Frage "Welches Produkt schätzt du am verbreitesten 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.

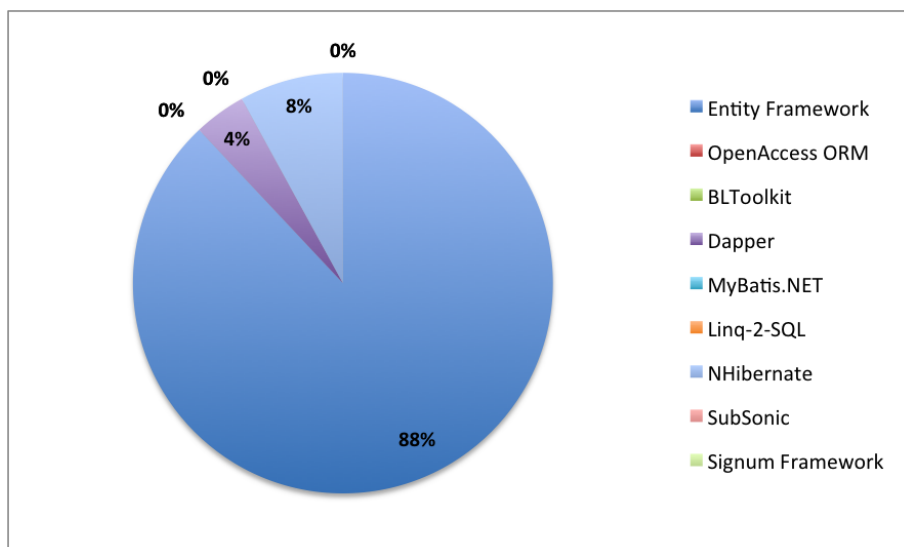


Abbildung 3.4: Umfrageresultate Welches Produkt schätzt du am verbreitesten ein?

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 Datenbankschemas oder 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.

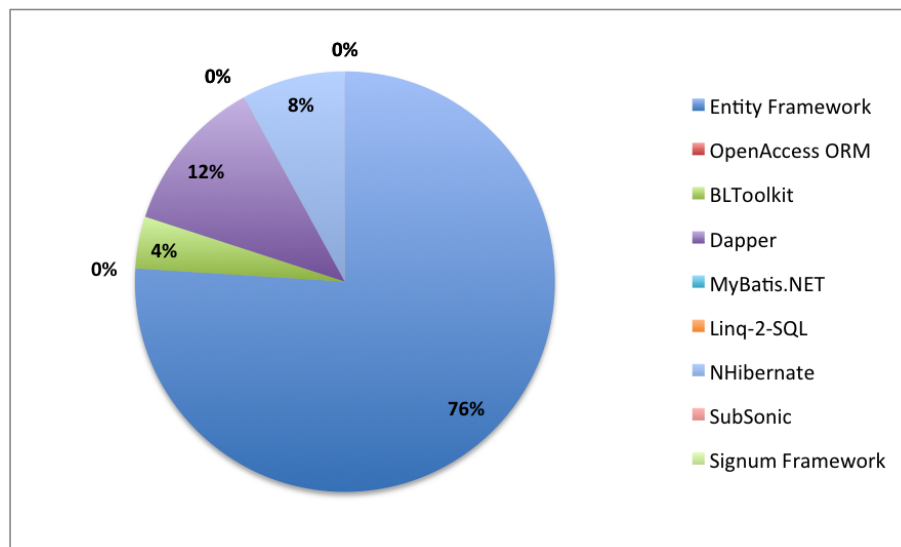


Abbildung 3.5: Umfrageresultate Für ein privates Projekt, welchen ORM würdest du einsetzen?

3.2.4 Resultat

Das Resultat der Umfrage war wenig überraschend. Die weite Verbreitung und der guten Ruf des Entity Frameworks war aufgrund des Marketings und der guten 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 Umfrage (siehe Kapitel 3.2.1 auf Seite 49) ausgewählten Produkte Entity Framework, Dapper und NHibernate auf Grund der folgenden Kriterien miteinander im Detail verglichen:

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

Im Gegensatz zur Marktübersicht setzten alle Vergleiche voraus, dass mit 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 79).

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 ist hängt vom Geschmack und der Präferenz jedes einzelnen Entwicklers ab und das Vergleichsresultat in Tabelle 3.1 kann für einen anderen Entwickler durchaus andere Werte enthalten. Ich persönlich präferiere zum Beispiel flüssige Schnittstellen¹² da ich als Entwickler bereits vom Compiler gewarnt werde, falls ich mich zum Beispiel beim Namen einer Eigenschaft eines Domänenobjekts verschrieben habe. Andere Entwickler bevorzugen vielleicht XML-basiertes Mapping, da sie die baumartige Struktur von XML bevorzugen. Wiederum andere Entwickler werden das Vorhandensein eines Mapping-Designers schätzen, da das Mapping grafisch abstrahiert

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

3 Konzept und Architektur

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

wird. Aus diesem Grund sind gewisse Kriterien in dieser Kategorie stark von persönlichen Vorlieben abhängig und die Wertung derselben nicht für jeden Entwickler gleich.

NHibernate hat den grossen Vorteil das viele für Hibernate entwickelte Drittanbietererweiterungen auch 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 einen sehr mächtigen, in Microsofts Visual Studio integrierten, Designer mitliefert. Auch ist die MSDN als erste Anlaufstelle für Dokumentation oder 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 Datenbankmanagementsystemen der betrachteten OR-Mapper. Die Werte für die einzelnen Kriterien stammen einerseits von den Produktwebseiten der Hersteller und wurden mittels eigens implementierten Proof-of-Concepts verifiziert.

¹³Die Erweiterung Fluent NHibernate (siehe <http://www.fluentnhibernate.org/>) erlaubt die Verwendung von flüssigen Schnittstellen beim Erstellen der Mappings.

¹⁴Wird später (siehe Kapitel 3.3.4 auf Seite 58) noch im Detail behandelt.

¹⁵Das kommerziell verfügbare Produkt NHibernate Designer (siehe <http://www.mindscapehq.com/products/nhdesigner>) bietet einen Modeldesigner.

¹⁶Wird in einem späteren Vergleich (siehe Kapitel 3.3.6 auf Seite 60) eingehender betrachtet.

3 Konzept und Architektur

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 ¹⁷
Unterstützt weitere Entwicklungsframeworks	Nein	Nein	Nein
Unterstützt Microsoft SQL Server	Ja	Ja	Ja
Unterstützt Oracle Database Server	Ja	Ja	Ja
Unterstützt weitere Datenbankmanagementsysteme	Ja ¹⁸	Ja ¹⁹	Ja ²⁰

Tabelle 3.2: Kriterien Plattformunterstützung

Das Entity Framework und Dapper benutzen als Basis das .NET Framework 3.5 da sie LINQ und Lambda-Expressions verwenden, welche mit dieser Version im .NET Framework 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 Sourcecodes²¹ ist es sehr viel schwieriger sich als Entwickler zurecht zu finden als zum Beispiel innerhalb von Dappers Sourcecode²². 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 das es eine Portierung des populären Open-Source Projekts Hibernate ist, welches in Java entwickelt wurde. Dadurch sind sämtliche Mapping- und Konfigurationsdateien, welche in NHibernate erstellt wurden, 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 Legacyprojekten darstellt.

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

¹⁷NHibernate ist eine Portierung von Hibernate, einem Java OR-Mapper (siehe <http://www.hibernate.org/>).

¹⁸Unterstützt auch SQLCE und weitere DBMS via Drittkomponentenanbieter.

¹⁹Unterstützt auch SQLCE, Firebird und MySQL.

²⁰Unterstützt auch Microsoft Access, Firebird, DB2, UDB, MySQL und SQLite.

²¹Der Sourcecode von NHibernate ist unter <https://github.com/nhibernate/nhibernate-core> zu finden.

²²Siehe <https://github.com/SamSaffron/dapper-dot-net>.

ben.

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, kann es beim produktiven Einsatz des System unangenehme Überraschungen in Bezug auf die Performance des verwendeten OR-Mapper geben.

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-Mapper²³ 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 1600 MHz 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 dieser 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.

Algorithmus 3.1 Abfrage auf eine Tabelle

```
SELECT ArtistId , Name FROM Artist WHERE Artist.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

²³Mehr zur Aufstartzeit am Beispiel von Entity Framework: <http://neverindoubt.net.blogspot.pt/2012/03/squash-entity-framework-startup-time.html>.

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 Domänenobjektes 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
```

In diesem Vergleich (siehe Resultate in Tabelle 3.6) 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 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

3 Konzept und Architektur

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

Abbildung 3.6: Kriterien Performance

eine Erweiterung für den NuGet Packetmanager²⁴ 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 geschriebenen Mappings welche vom Entwickler komplett von Hand erstellt werden müssen. Die Entitäten selber 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 jedoch die XML Schema Definitionen für die XML-Dateien mit, welche dann auch direkt mit Intellisense im Visual Studio verwendet werden können²⁵. Es gibt zahlreiche Fremdkomponentenhersteller welche Tools für NHibernate entwickeln, so zum Beispiel der NHibernate Designer von Mindscape HQ²⁶ oder Visual NHibernate von Slyce²⁷. 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 Packets 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. Wür-

²⁴Mehr zu NuGet unter <http://nuget.org/>.

²⁵Siehe hierzu <http://stackoverflow.com/questions/3565156/how-to-add-nhibernate-xml-schema-to-visual-studio>.

²⁶Siehe <http://www.mindscapehq.com/products/nhdesigner>.

²⁷siehe <http://www.slyce.com/VisualNHibernate/>.

de man die Drittanbietertools für NHibernate auch betrachten, würde die Entscheidung wohl nicht so klar ausfallen, da für NHibernate auf Grund seiner Popularität viele Tools von Drittanbietern existieren.

3.3.5 Unterstützung Enterprise Pattern

Fowlers Enterprise Pattern lassen sich mit jedem OR-Mapper implementieren. Die aktuellen Versionen von Entity Framework, NHibernate und Dapper zwingend den Entwickler nicht dazu ein bestimmtes Enterprise Pattern zu wählen. Das Entity Framework und NHibernate besitzen eine Möglichkeit, das Pattern Unit of Work²⁸ 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 Software Architekten 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 das 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 in der Wartung des Systems eine Rolle spielt, wenn Performanceengpässe spürbar werden. Wird dann ein ORM eingesetzt der nur sehr eingeschränkt debug- und überwachbar ist, wird es schwierig, 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 Ursache 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 Debugging²⁹. Da für das Entity Framework die Debuggingssymbole 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

²⁸Mehr zum Pattern <http://martinfowler.com/eaCatalog/unitOfWork.html>

²⁹Intellitrace Debugging bietet einen gesamtheitlichen Überblick über den Zustand einer laufenden Applikation im Gegensatz zu traditionellen Debuggern welche einen isolierten Zustand betrachten.

3 Konzept und Architektur

Drittherstellertools wie der Entity Framework Profiler von Hibernating Rhinos³⁰ oder der EF Provider Wrapper in der Visual Studio Gallery³¹ welche bei Performanceengpässen in produktiven Umgebungen zum Einsatz kommen können und dem Entwickler teilweise grafisch aufzeigen, welche Operationen innerhalb des Entity Frameworks wieviel Performance kosten.

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

NHibernate ist ein Open-Source Projekt und aus diesem Grund kann dieses für Debuggingzwecke selbst kompiliert werden und debugged 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 Profiler³². 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 Interceptoren³³ zu schreiben und in NHibernate zu injizieren. So kann zum Beispiel jedes auf das Datenbankmanagementsystem abgesetzte SQL-Statement mit einem Interceptor abgefangen werden und in ein Tracefile geschrieben werden. Auch lässt sich direkt eine log4net³⁴ 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 Monitoringmöglichkeiten mitbringen. Dies ist wohl auch so gewünscht, bieten insbesondere NHibernate und das Entity Framework zahlreiche Möglichkeiten für Entwickler oder Drittanbieter Monitoringsuiten zu entwickeln und auf die eigenen Bedürfnisse zuzuschneiden. Dieser Vergleich bezieht sich somit auf die Möglichkeit zur Implementierung von Überwachungsmechanismen und bezieht auch auf die verfügbaren Monitoringtools von Drittherstellern in das Vergleichsergebnis ein. Mit dieser gesamtheitlichen Betrachtung ist das Entity Framework der knappe Gewinner vor NHibernate dieses Vergleichs.

³⁰Siehe <http://www.hibernatingrhinos.com/products/EFProf>

³¹Mehr dazu unter <http://code.msdn.microsoft.com/EFProviderWrappers>

³²Siehe <http://www.hibernatingrhinos.com/products/nhprof>.

³³Mehr zum Interceptor Pattern siehe <http://bosy.dailydev.org/2007/04/interceptor-design-pattern.html>.

³⁴Mehr zu log4net siehe <http://logging.apache.org/log4net/>.

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

Tabelle 3.3: Vergleichsmatrix

3.3.7 Vergleichsmatrix

In Tabelle 3.3 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 bietet die Grundlage für die Evaluation der OR-Mapper (siehe Kapitel 4.1 auf Seite 75) und die Produktempfehlung (siehe Kapitel 4.2 auf Seite 78).

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

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 ein vollständiges und umfassendes Lösungskonzept zu entwickeln, welches in der Implementierungsphase 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 Systemes und entsprechen in dieser Hinsicht den konventionellen Implementierungsmodellen.

3.4.1.1 Komponentendiagramm

Das Komponentendiagramm in Abbildung 3.7 stellt das System SoCrm³⁵ 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 in dem alle drei Layer als eigens ausführ- und verteilbare Komponenten realisiert werden.

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

Die Orchestrationsservices 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.

3.4.1.2 Domänenmodell

Das Domänenmodell (siehe Abbildung 3.8) 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 `IDomainObject`. 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 (`EEmailAddress`) und Telefonnummern (`PhoneNumber`) als Kontaktmöglichkeiten haben.

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

3 Konzept und Architektur

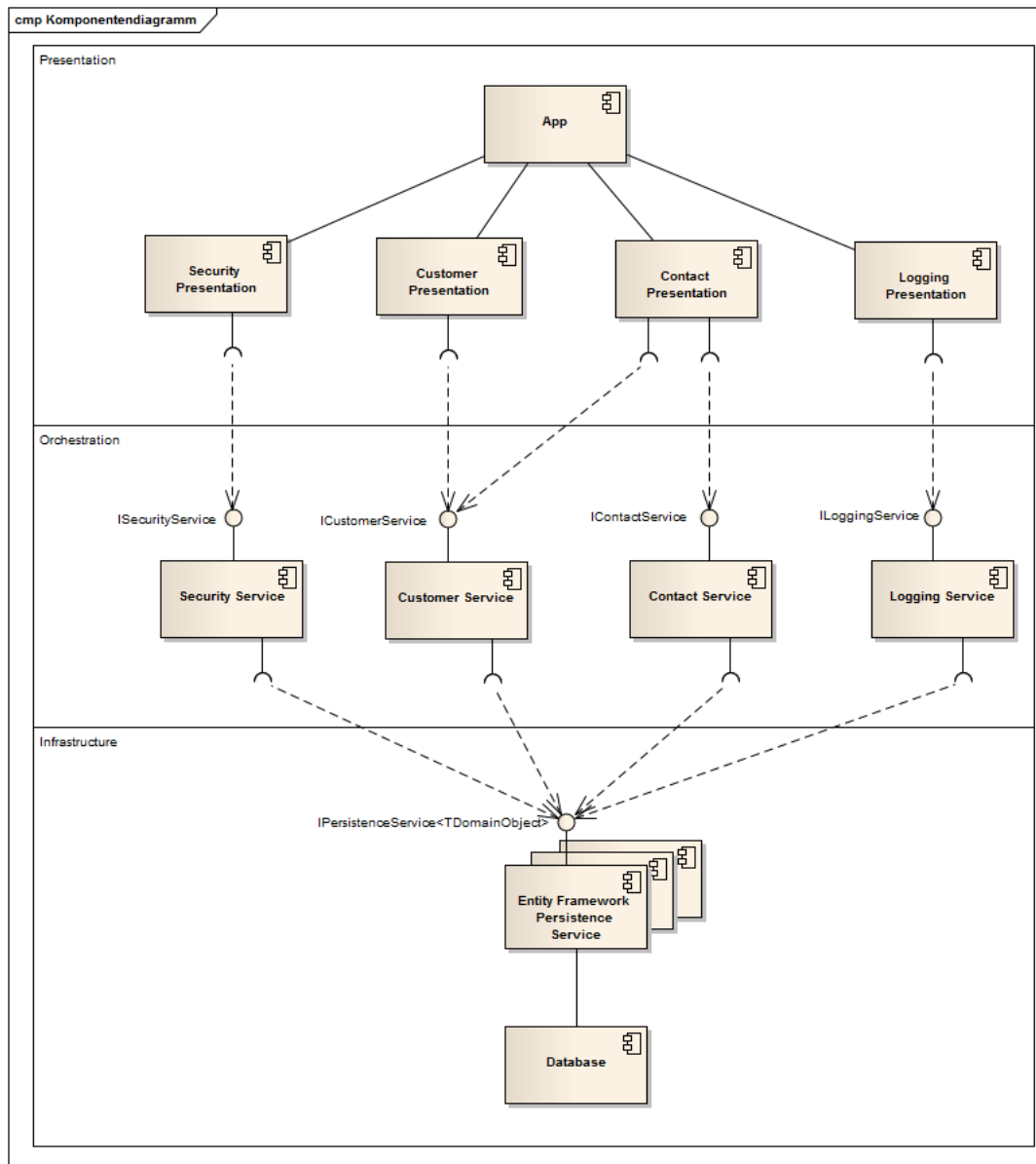


Abbildung 3.7: Komponentendiagramm SoCrm

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 auch gleichzeitig zur Authentifizierung am System.

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.

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 3.4) 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.

3 Konzept und Architektur

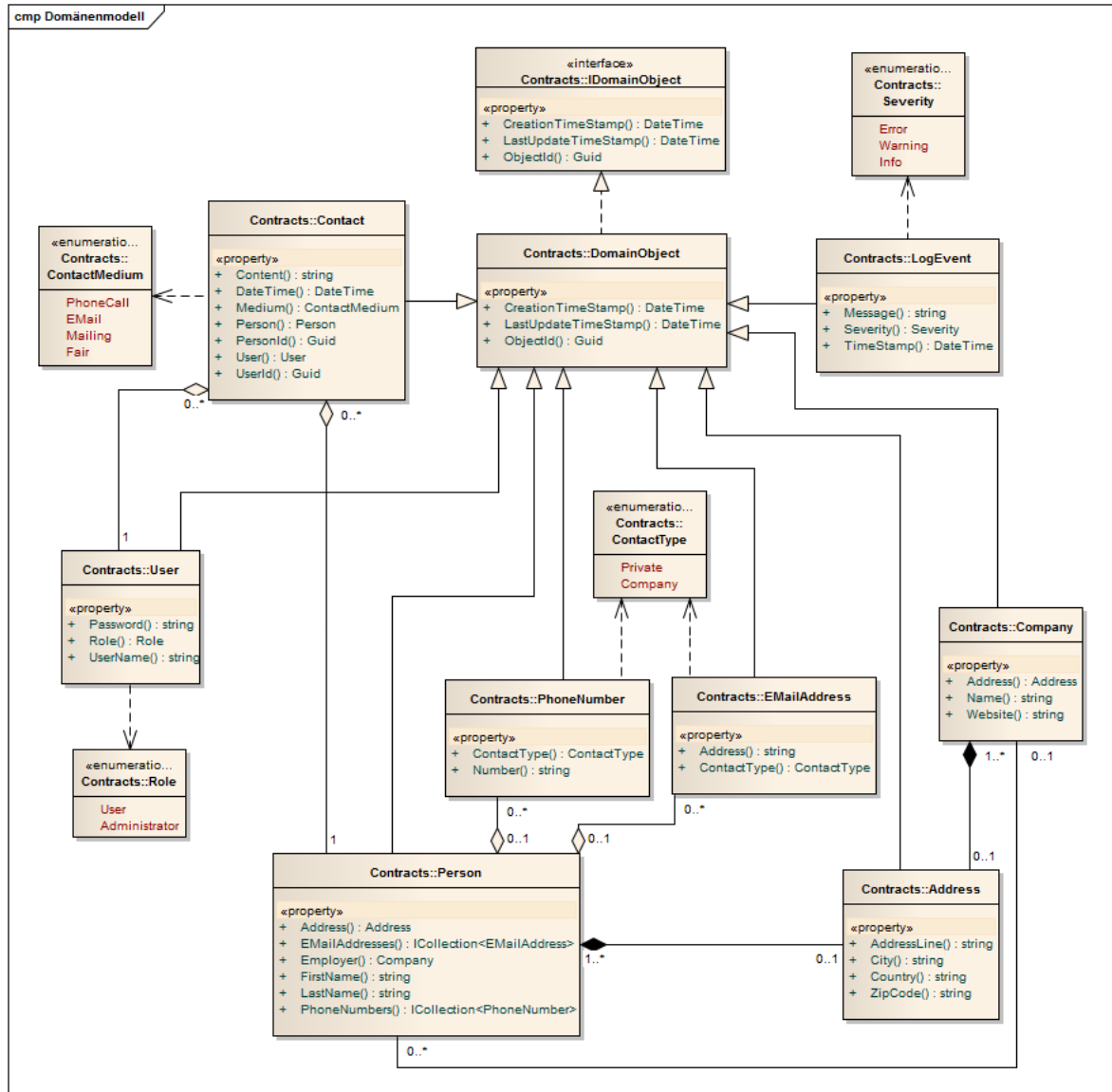


Abbildung 3.8: Klassendiagramm Domänenmodell SoCrm

3 Konzept und Architektur

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
GetUserById	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 Passwort für einen bestimmten Benutzer
ValidateCredentials	Überprüft die übergebenen Benutzerinformationen und liefert wahr oder falsch zurück

Tabelle 3.4: Methoden von ISecurityService

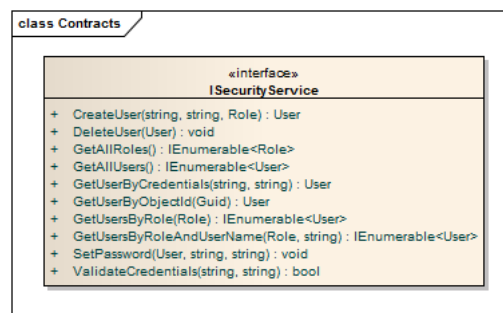


Abbildung 3.9: Klassendiagramm ISecurityService

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.

3 Konzept und Architektur

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

Tabelle 3.5: Methoden von ICustomerService

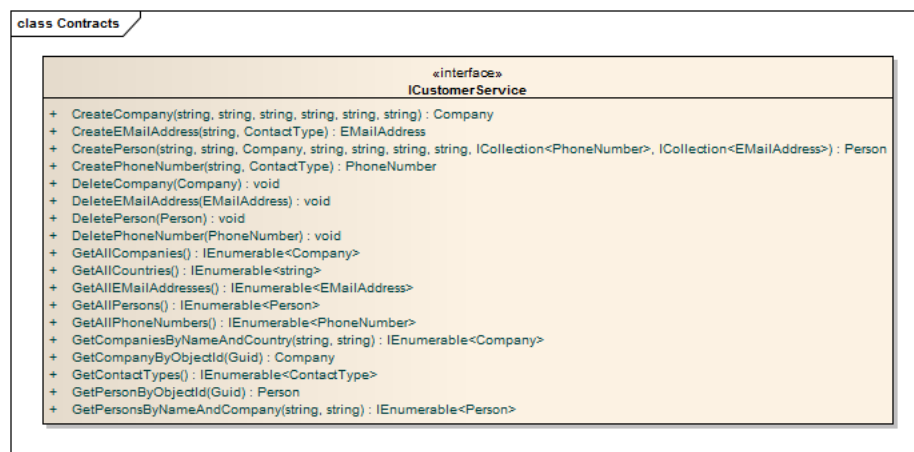


Abbildung 3.10: Klassendiagramm ICustomerService

IContactService Die vom Contact Service im Orchestrationlayer offerierte Schnittstelle IContactService liefert Informationen zu den Kontakten eines Kunden. Die Methode

3 Konzept und Architektur

CreateContact erstellt einen neuen Kontakt. Die Methode DeleteContact ermöglicht es dem UI einen Kontakt zu löschen. Über einen Aufruf an GetAllContactMediums werden alle möglichen Kontaktmedien zurückgegeben. Die Get-Methoden geben gefilterte List von Kontakten zurück auf Basis der übergebenen Parameter.

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
GetContactById	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
GetContactsByContactMediumAndPersonName	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 3.6: Methoden von IContactService

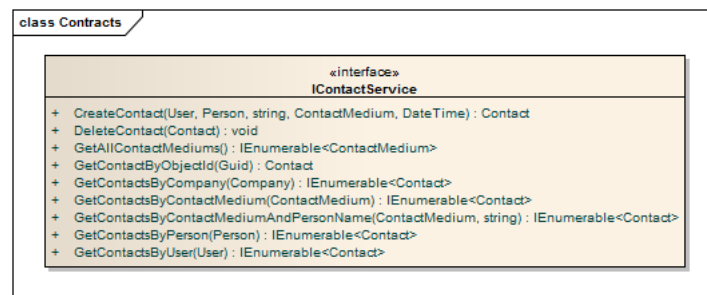


Abbildung 3.11: Klassendiagramm IContactService

ILoggingService Die Schnittstelle ILoggingService wird innerhalb des Orchestration-layers vom Logging Service implementiert und erlaubt es der Benutzeroberfläche oder anderen Services Logeinträge zu schreiben beziehungsweise diese auszulesen. Die GetAll-Methoden geben jeweils ungefiltert die gesamte Liste von verfügbaren 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.

3 Konzept und Architektur

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

Tabelle 3.7: Methoden von ILoggingService

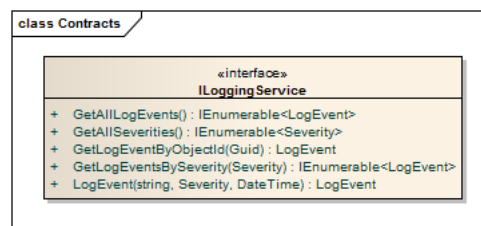


Abbildung 3.12: Klassendiagramm ILoggingService

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, die beispielsweise den Systemstart, die Laufzeitkonfiguration oder die Administration des Systems betreffen. Darüber hinaus dokumentiert die Laufzeitsicht, wie Laufzeitkomponenten sich aus Instanzen von Implementierungsbausteinen zusammensetzen.

Dieses Kapitel enthält stellvertretend für alle Use-Cases den Use-Case “Kontakt hinzufügen” (siehe Kapitel 2.4.2.9 auf Seite 31) modelliert als Sequenzdiagramm. Es wird nur dieser Use-Case aus der Laufzeitsicht betrachtet da alle anderen Use-Cases praktisch identisch ausgeführt werden. Dieses Sequenzdiagramm steht exemplarisch für alle anderen Use-Cases.

3.4.2.1 Kontakt hinzufügen

Der Benutzer ist bereits beim System authentifiziert und besitzt einen gültigen Sessio-
 ntoken. Möchte ein Benutzer einem Kunden einen neuen Kontakt hinzufügen, so na-
 vigiert er zuerst auf die CustomerListView, welche alle Kunden anzeigt. Die Customer-
 ListView erhält ihre Daten aus dem CustomerService welcher wiederum den Persistence-
 Service<Customer> aufruft, der die Kundendaten aus der Datenbank liest und mittels
 ORM auf Domänenobjekte vom Typ Customer mappt.

3 Konzept und Architektur

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 3.14 lesbarer zu gestalten wurde auf einzelne Elemente im Ablauf des Use-Cases verzichtet beziehungsweise diese wurden vereinfacht. So fehlen zum Beispiel die Controller und ViewModels und die Persistenzservices sind zu einem einzigen generischen Service zusammengefasst.

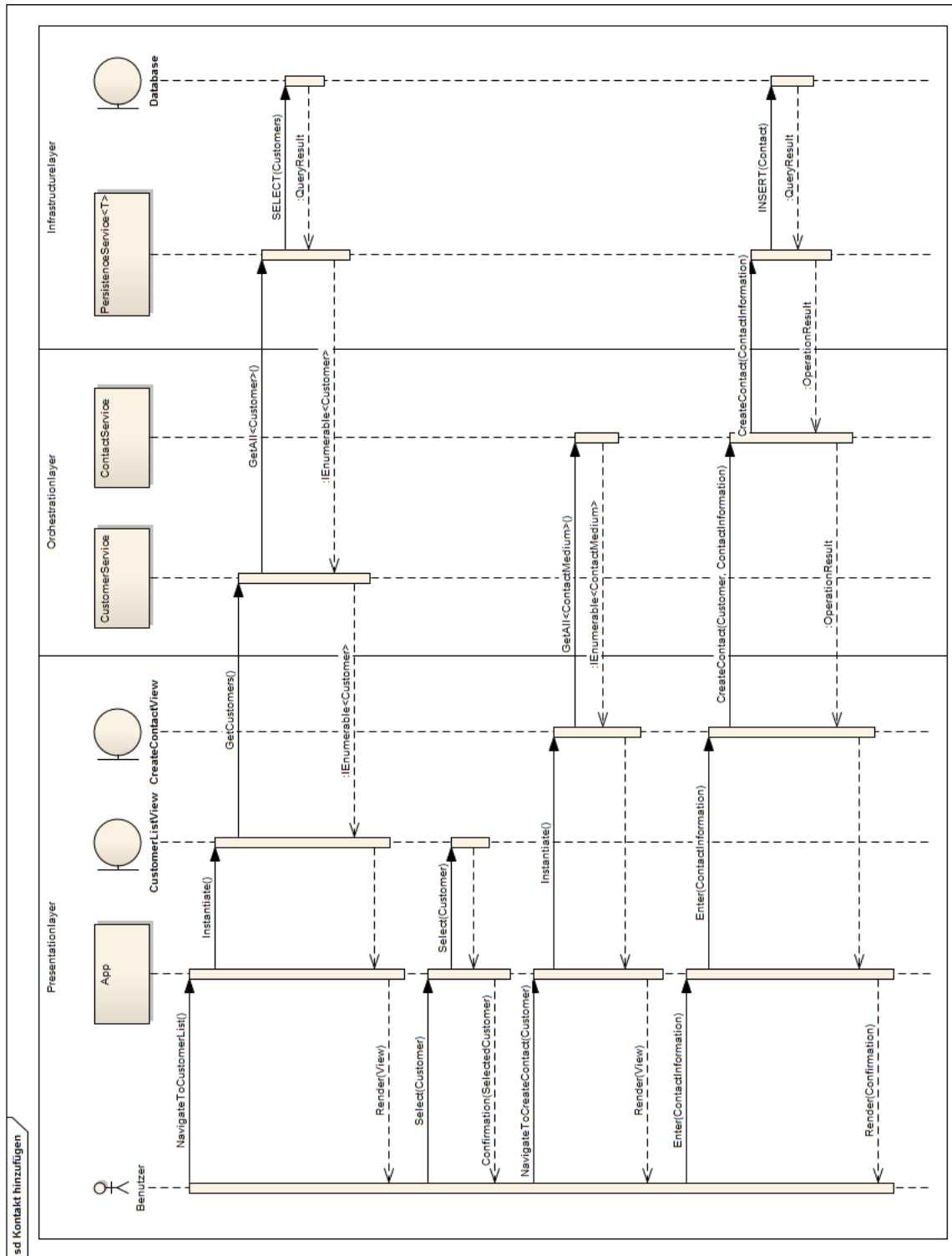


Abbildung 3.13: Sequenzdiagramm Kontakt hinzufügen

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 [?]).

3.4.3.1 Verteilungsdiagramm

Die Verteilungssicht dieser Projektdokumentation enthält nur ein sehr rudimentär ausgearbeitetes Verteilungsdiagramm (siehe Abbildung 3.14). 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 Rechnern 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 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 allzu grosse Latenz entsteht.

Die Clientapplikation App ist eine WPF-Anwendung die auf jedem beliebigen Windows Betriebssystem verteilt werden kann. Das einzig notwendig ist eine Änderung der Konfiguration der Endpunkte der Orchestrationlayerservices.

3 Konzept und Architektur

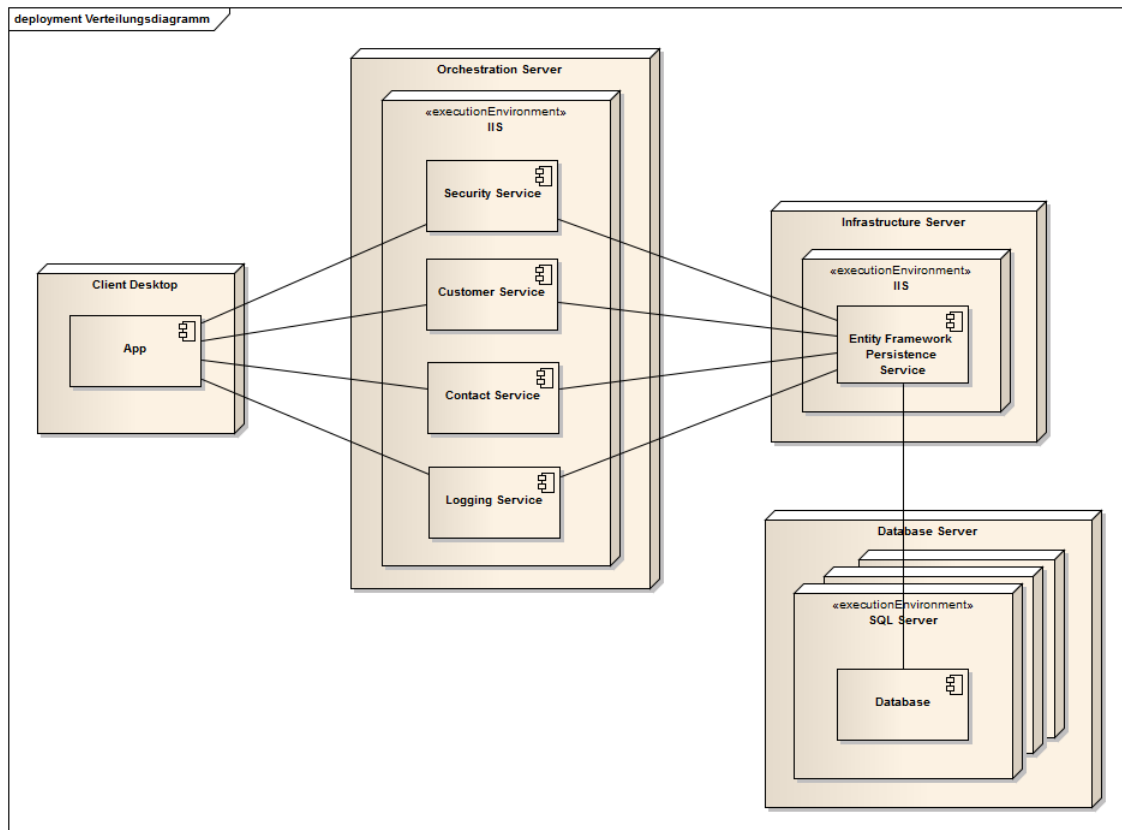


Abbildung 3.14: Verteilungsdiagramm SoCrm

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 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 und die im Rahmen der Evaluation und Benutzung gewonnenen Erkenntnisse beleuchtet. Die OR-Mapper werden in diesem Kapitel an den erhobenen Anforderungen an den Persistenzlayer einer serviceorientierten Architektur (siehe Kapitel 2.2 auf Seite 12) und an den Anforderungen der Enterprise Pattern gemessen. Es ist erneut darauf hinzuweisen, 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 nachfolgende Produktempfehlung (siehe Kapitel 4.2 auf Seite 78).

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 49) und beweist das Microsoft seit der Einführung des Produkts im August 2005 gute Arbeit bei der Dokumentation geleistet hat. Beim Rückblick auf die Marktübersicht (siehe Kapitel 3.1 auf Seite 39) und den erweiterten Vergleichen (siehe Kapitel 3.3 auf Seite 54) zeigt sich, dass nicht nur das umfangreiche Marketing 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 57) gut zeigt. Als kompletter OR-Mapper war vorzusehen dass gegenüber eines leichtgewichtigen OR-Mappers grosse 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 Drittanbieterwerkzeugen oder zusätzlichem Implementierungsaufwand verbunden erfüllbar. Die Anforderung PQ3 bedeutet in diesem konkreten Fall, dass der ObjectContext des Entity Frameworks für Unittests abstrahiert (z.B. mit einem Mockingframework) werden sollte. Dies stellt sich in der Praxis als sehr aufwändig dar 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 wird) alle Anforderungen erfüllt. Das Entity Framework bietet von Haus aus keine Möglichkeit zu spezifizieren 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 nun geladen werden soll und welcher nicht) so muss dies im Persistenzlayer selber 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 Funktionlitäten (wie zum Beispiel Cachingmechanismen oder Lazy Loading) die sonst nur in deutlich umfangreicheren ORMs anzutreffen sind. Insbesondere die einfache Konfiguration der Mappings über Lambda 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 sein.

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

Beim Vergleich von Dapper mit den funktionalen Anforderungen an den Persistenzlayer zeigt sich der geringere Funktionsumfang gegenüber der kompletten OR-Mapper: PF8 (Das System muss mehrere Instanzen gleichzeitig löschen können) und PF9 (Das System muss dem Administrator die Möglichkeit bieten den Ablauf sämtlicher Operatio-

nen 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 60). 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 Technologie 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) nicht aus den vorher genannten Gründen 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 wird) 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 das NHibernate lange der einzige ausgereifte komplette OR-Mapper im .NET-Umfeld war und dass dadurch viele Erfolgsgeschichten von Projekte unter Verwendung von NHibernate geschrieben wurden. Viele .NET-Entwickler haben NHibernate schon einmal integriert oder in einem Projekt mitgearbeitet in welchem NHibernate für das objekrelationale 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 mithalten. Viele der Funktionalitäten die NHibernate im Vergleich zu den Konkurrenzprodukten fehlen wurden von der Open-Source 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 unleserlich) Code oder einem Wildwuchs von Konfigurationsdateien führen.

Wird NHibernate mit den Anforderungen an den Persistenzlayers einer serviceorien-

tierten 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 den selben Schwierigkeiten was die Überwachung des ORMs zur Laufzeit sowie das automatisierte Testen desselben 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 Produkteempfehlung

In diesem Kapitel wird pro 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. Es wird dabei auf Grund der Evaluation in den vorherigen Kapiteln eine Empfehlung ausgesprochen für welches Enterprise Pattern welcher OR-Mapper eingesetzt werden soll.

Wie schon unter dem Vergleichskriterium Unterstützung Enterprise Pattern (siehe Kapitel 3.3.5 auf Seite 60) festgehalten sind eigentlich alle OR-Mapper für alle betrachteten Enterprise Pattern geeignet. In den folgenden Kapiteln wird trotzdem versucht eine differenziertere Betrachtungsposition anzunehmen und pro Enterprise Pattern eine konkrete Empfehlung abzugeben.

4.2.1 Transaction Script

Entity Framework, Dapper, NHibernate

4.2.2 Domain Model

Entity Framework

4.2.3 Table Module

Dapper

4.2.4 Active Record

Dapper

4.2.5 Abschliessende Bemerkung

Fullblown vs lightweight ORMs!

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 soll. Häufig sind diese Implementierungs- oder Designprinzipien durch den Software Architekten 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. Stimmen diese nicht überein müssen in der Dokumentation jeweils neben dem Namen des Assemblies auch noch der vollqualifizierte Namen eines Typs 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. Service Referenzen in Projekten, welche Web Services verwenden, benutzen immer die Typen dieser Assemblies anstatt die Entitäten aus dem WSDL der Service Referenz zu erzeugen.
- Die Persistence-Services (in SoCrm.Infrastructure.Persistence) besitzen keinerlei Geschäftslogik sondern dienen rein 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 MVVM¹ dar und verschiebt die Erzeugung der View und des ViewModels in den Controller. Auch wird die Beziehung zwischen der View und dem ViewModel im Controller erzeugt und nicht mehr über das Binding gelöst.
- Um dynamisch Views innerhalb einer Shell laden zu können sind alle Views als UserControls implementiert, welche 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 Anhaltspunkt 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 des Systems 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 nicht erwähnenswert sind und deswegen in den Klassendiagrammen auch nicht gezeichnet sind. Auch wurden nur in denjenigen Unterkapiteln Klassendiagramme eingefügt wo es für den Leser von Interesse sein könnte.

4.3.2.1 SoCrm.Core.Contracts

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

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 Unity² betreiben zu können. Damit

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

²Mehr zu Unity und Dependency Injection unter <http://unity.codeplex.com/>

4 Implementierung

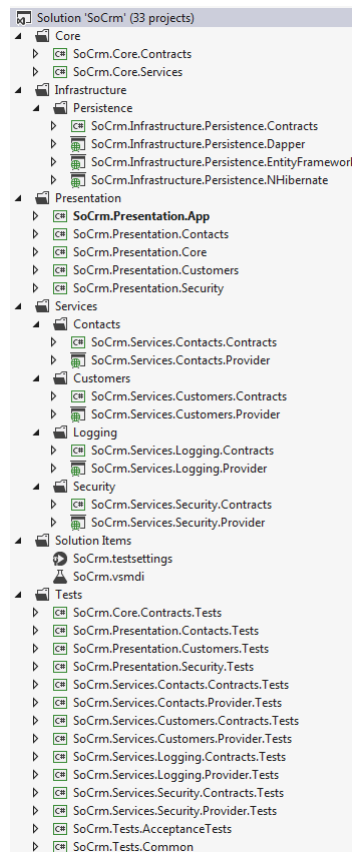


Abbildung 4.1: Visual Studio Solution

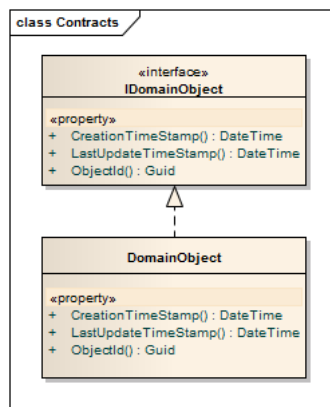


Abbildung 4.2: Klassendiagramm SoCrm.Core.Contracts

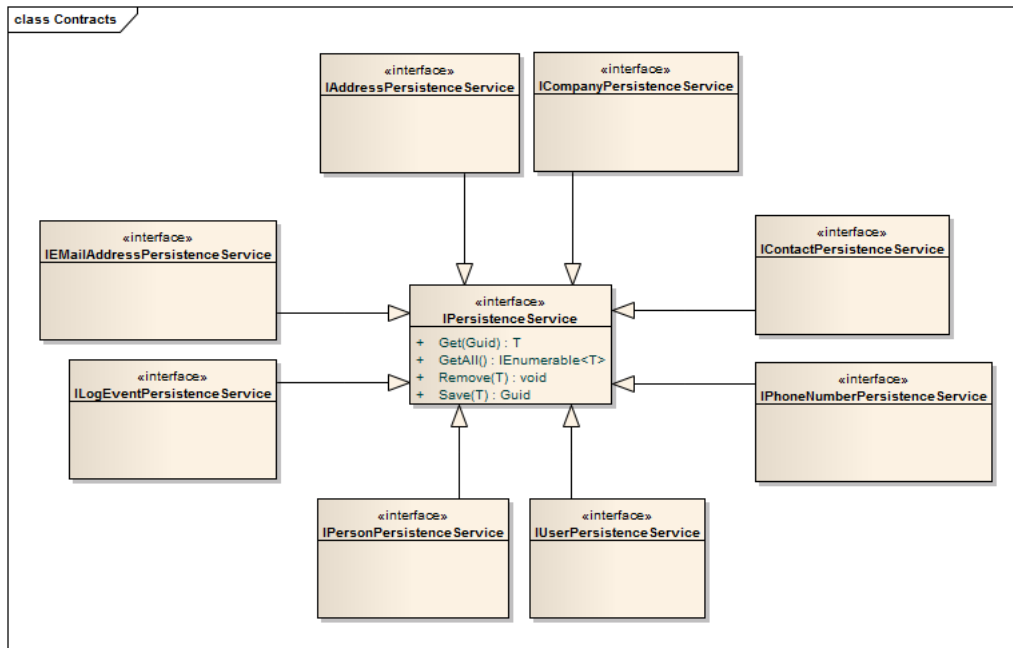


Abbildung 4.3: Klassendiagramm SoCrm.Infrastructure.Persistence.Contracts

die WCF Services mit Unit Tests versehen werden können wurden die externen Abhängigkeiten mit Dependency Injection entkoppelt. Für das Starten der Services wird jedoch ein spezieller Service-Host, der UnityServiceHost, benötigt, welcher beim Aufstarten ein Behavior mit dem Unity-Container erzeugt und dem Service injected. Der spezifische Service-Host, das Service-Behavior und das Instance-Behavior sind in diesem Projekt zu finden.

4.3.2.3 SoCrm.Infrastructure.Persistence.Contracts

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

4.3.2.4 SoCrm.Infrastructure.Persistence.Dapper

Die Persistenzlayerimplementierung von Dapper in der WCF Service Application SoCrm.Infrastructure.Persistence.Dapper besteht aus einer Basisklasse `PersistenceServiceBase<T>`, welche als Parameter den Entitätstypen akzeptiert, und den acht Persistence-Services, welche jeweils einen eigenen WCF Service darstellen. Jeder `PersistenceService` 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 PersistenceService 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 PersistenceServices welche die entsprechenden Service Contracts aus SoCrm.Infrastructure.Persistence.Contracts implementieren. Sämtliche Logik ist dabei in der generischen Basisklasse PersistenceServiceBase abgelegt, die einzelnen PersistenceServices sind leer und dienen lediglich als Endpoints für die WCF Services.

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

Die Mappings aller Entitäten sind im Namespace SoCrm.Infrastructure.Persistence.NHibernate.Mappings abgelegt. Dabei wird FluentNHibernate³ verwendet um die XML-Konfiguration zu vereinfachen und den Aufwand diese zu Erstellen zu verkleinern.

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 ApplicationController

³Siehe <http://www.fluentnhibernate.org/>

4 Implementierung

welcher aus dem UnityContainer die Controller der SoCrm.Infrastructure.Presentation.Contacts, SoCrm.Infrastructure.Presentation.Customers oder SoCrm.Infrastructure.Presentation.Security Assembly bezieht und dort auf die gewünschten Seiten navigiert.

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

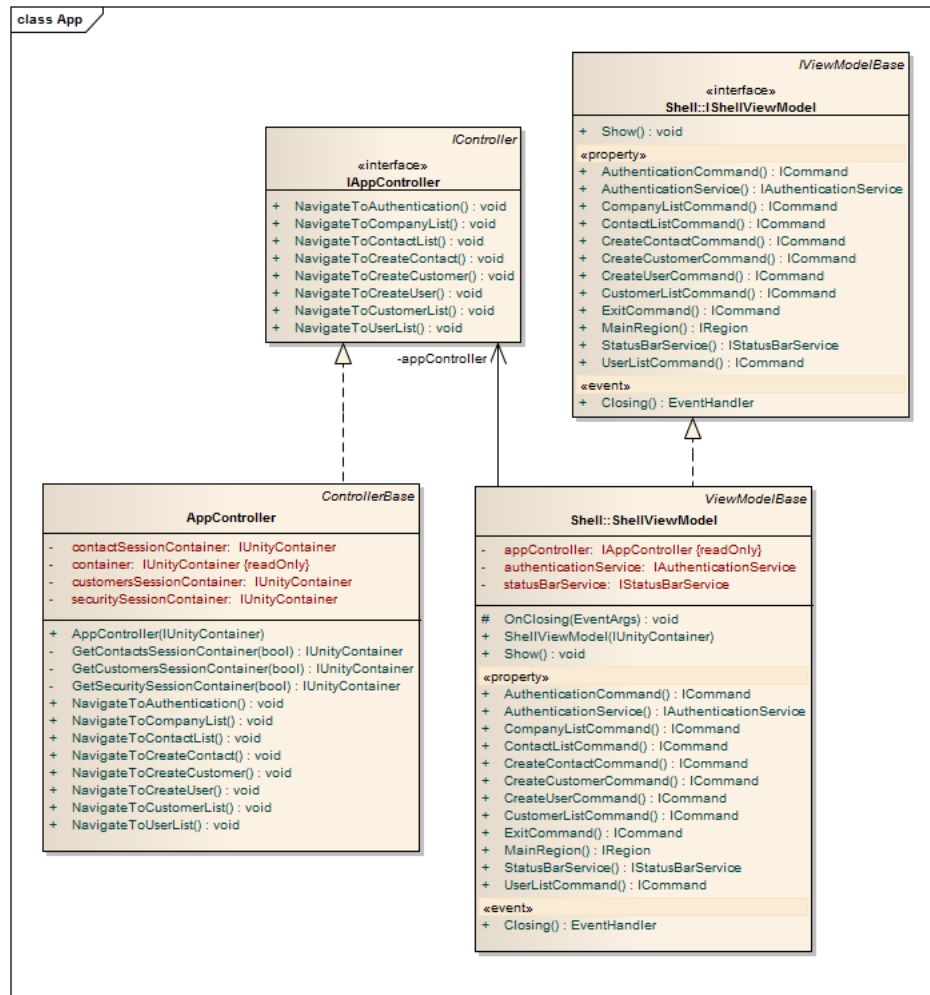


Abbildung 4.4: Klassendiagramm SoCrm.Presentation.App

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 instanziiert bei Bedarf die ContactListView und das ContactListViewModel (für das Anzeigen be-

4 Implementierung

stehender Kontakte verantwortlich) sowie die CreateContactView und das CreateContactViewModel (enthält die Funktionalität zum Erstellen von neuen Kontakten für einen Kunden).

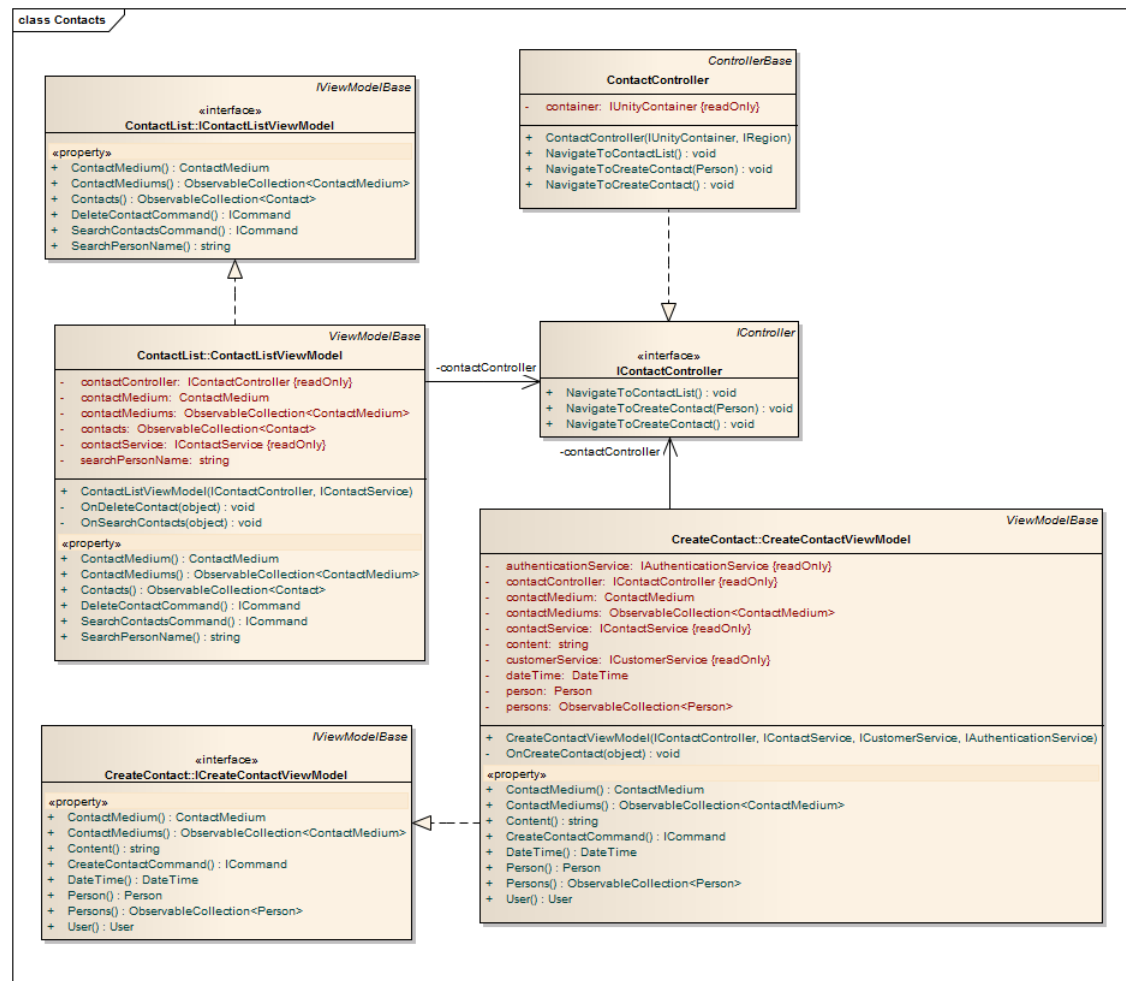


Abbildung 4.5: Klassendiagramm SoCrm.Presentation.Contracts

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.

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

4 Implementierung

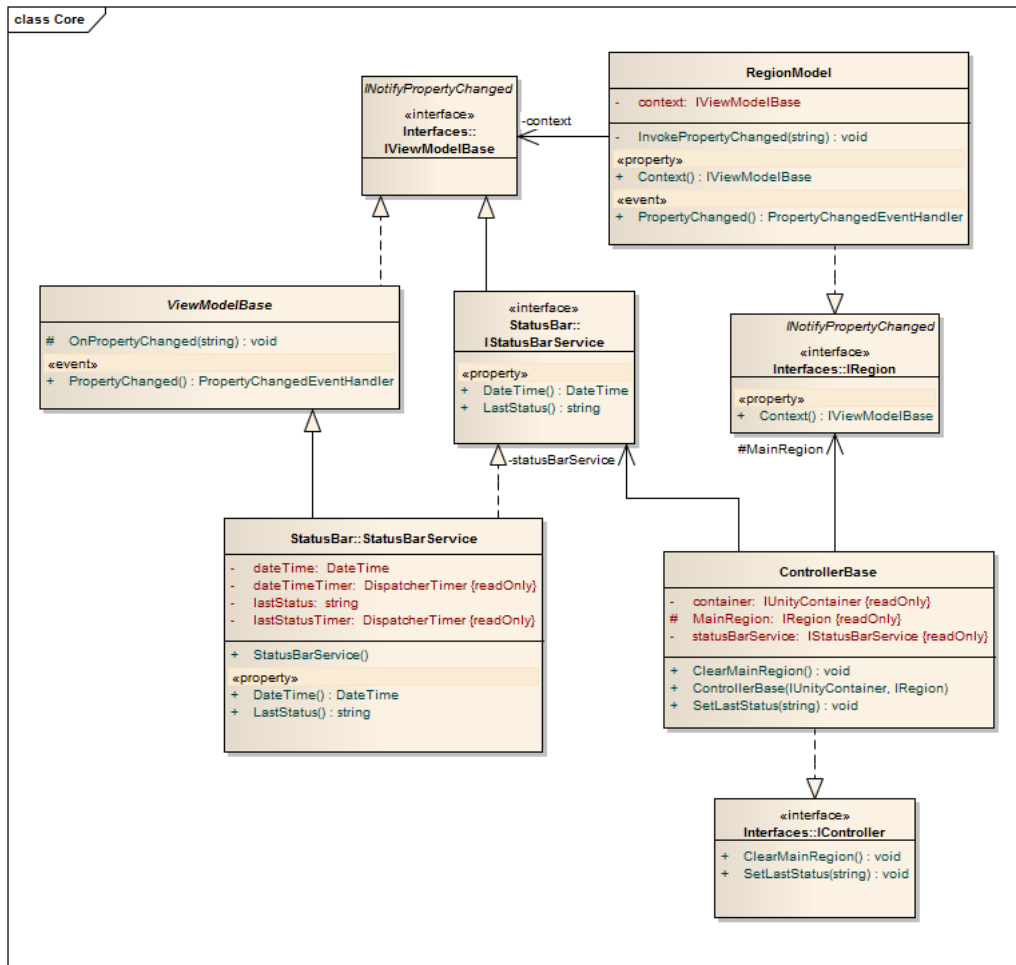


Abbildung 4.6: Klassendiagramm SoCrm.Presentation.Core

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** instanziiert 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.

4 Implementierung

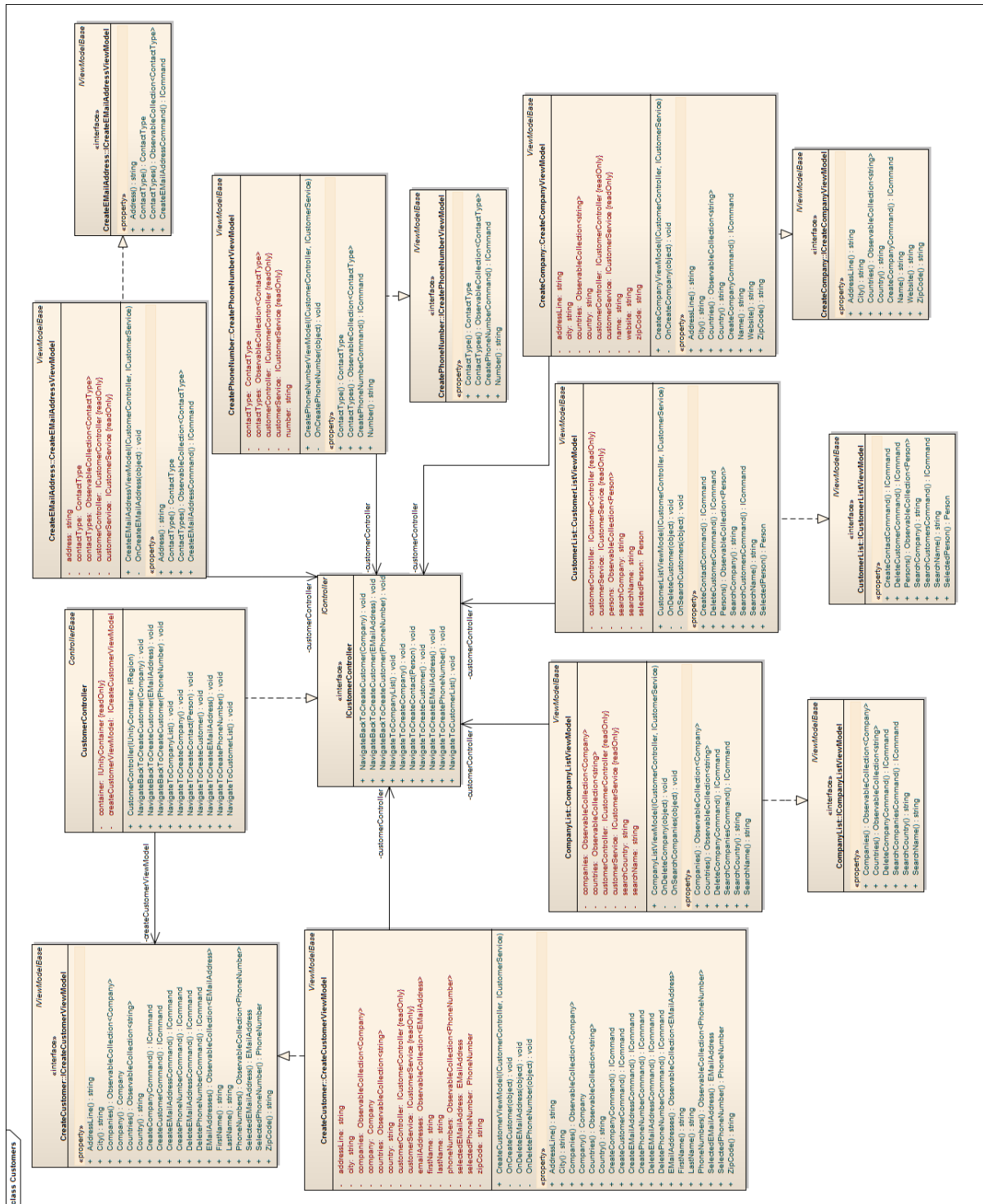


Abbildung 4.7: Klassendiagramm SoCrm.Presentation.Customers

4.3.2.11 SoCrm.Presentation.Security

SoCrm.Infrastructure.Presentation.Security enthält die Views und ViewModels für die Benutzerverwaltung sowie das Authentifizieren beim System. Auch ist die SetPasswordView und das SetPasswordViewModel ein Teil dieser Assembly. Das AuthenticationViewModel besitzt eine Referenz auf den AuthenticationService welcher als Singleton 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.

4 Implementierung

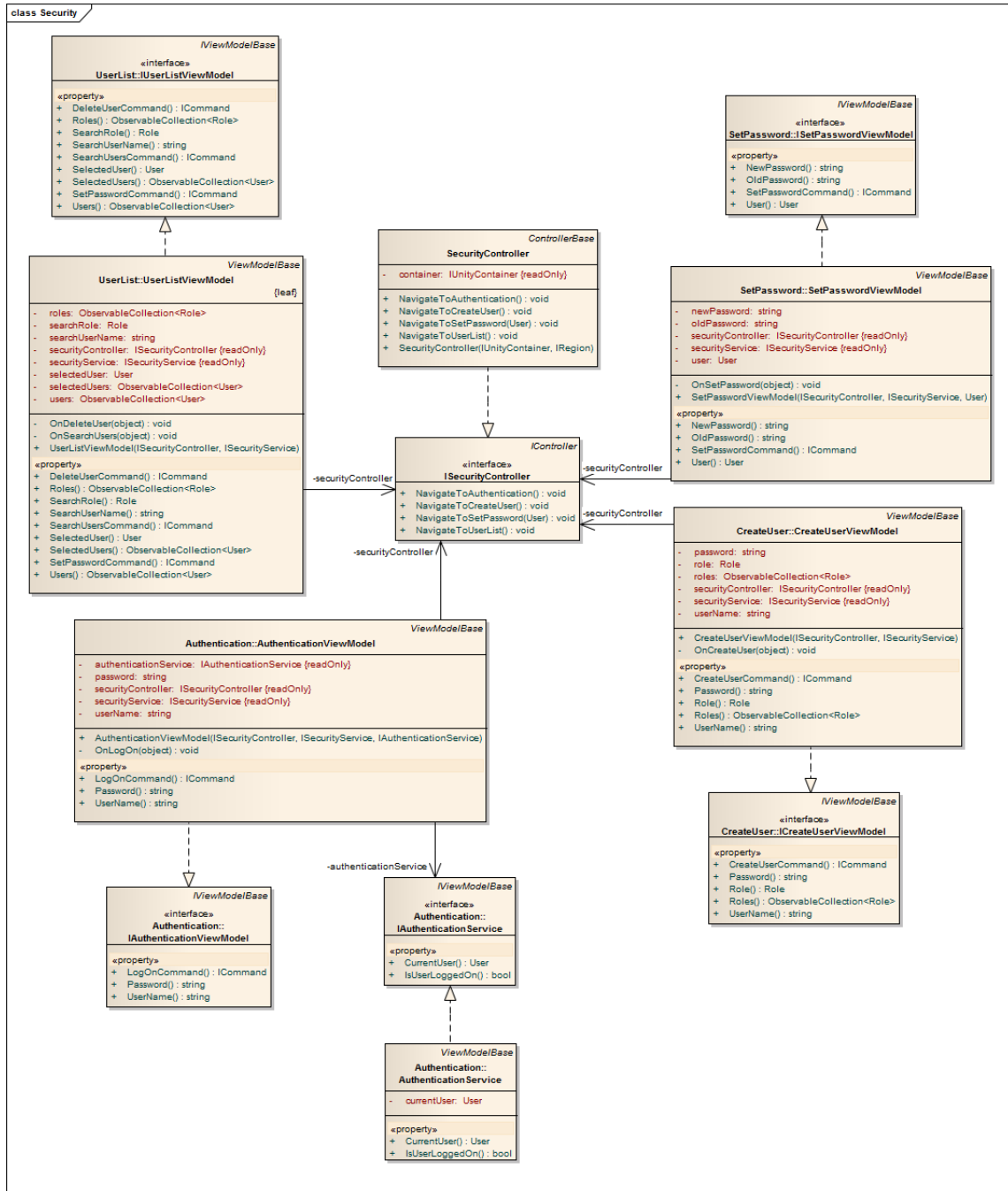


Abbildung 4.8: Klassendiagramm SoCrm.Presentation.Security

4.3.2.12 SoCrm.Services.Contracts.Contracts

In dieser Assembly befindet sich die Entität `Contact`, welche von `DomainObject` (in `SoCrm.Core.Contracts`, siehe Kapitel 4.3.2.1 auf Seite 80) erbt und das bereits in einem vorherigen Kapitel beschriebene Service-Interface `IService` (siehe Kapitel 3.4.1.3 auf Seite 68).

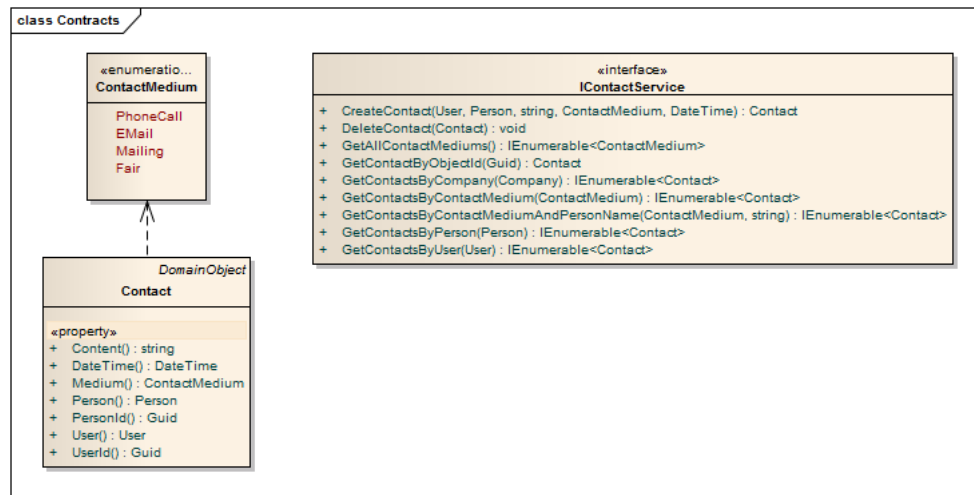


Abbildung 4.9: Klassendiagramm `SoCrm.Services.Contracts.Contracts`

4.3.2.13 SoCrm.Services.Contracts.Provider

Die WCF Service Applikation `SoCrm.Services.Contracts.Provider` liefert die Implementierung zum Service-Interface `IService`, das im Namespace `SoCrm.Services.Contracts.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 80) instanziert und einen neuen Unity-Container erstellt. Der `ContactService` implementiert sämtliche Geschäftslogik betreffend der Kontakte und ruft den Persistenzservice für Kontakte (implementiert in einem der `SoCrm.Infrastructure.Persistence`-Projekte) auf, um die Entitäten aus dem Persistenzlayer zu lesen.

4.3.2.14 SoCrm.Services.Customers.Contracts

Die Klassenbibliothek `SoCrm.Services.Customers.Contracts` 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 67), welches in `SoCrm.Service.Customers.Provider` als WCF Service `CustomerService` implementiert ist.

4 Implementierung

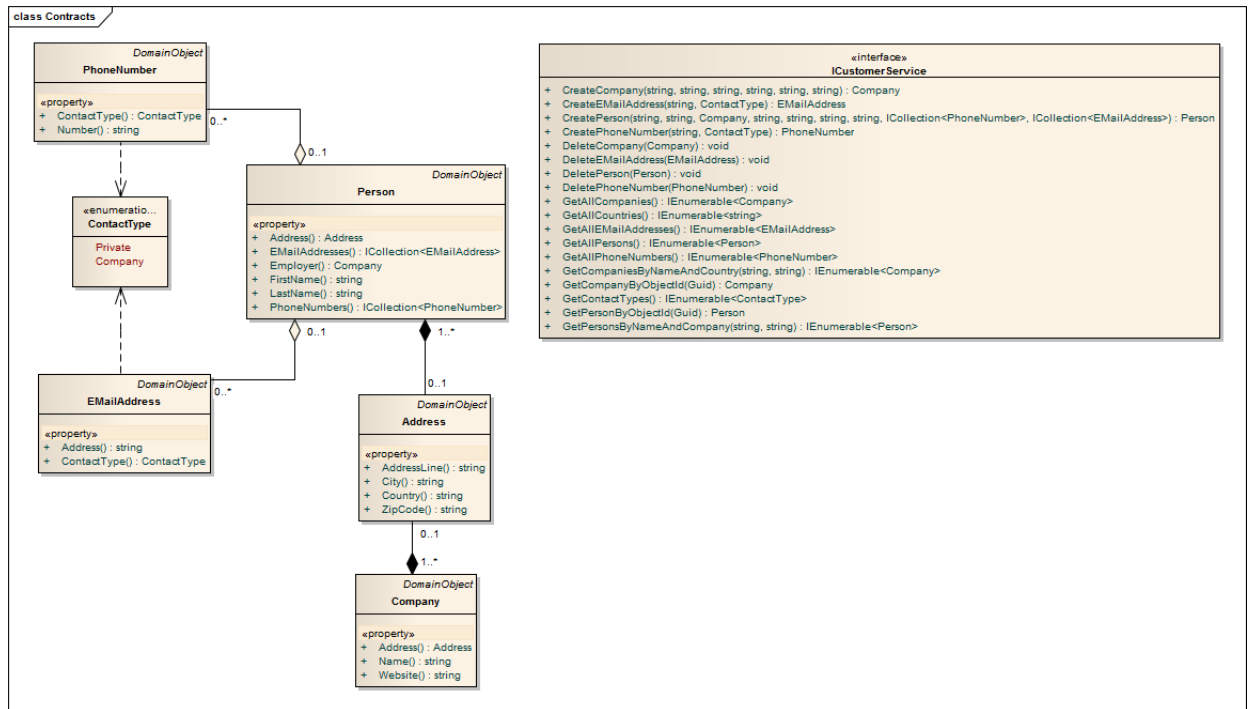


Abbildung 4.10: Klassendiagramm SoCrm.Services.Customers.Contracts

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 69).

4 Implementierung

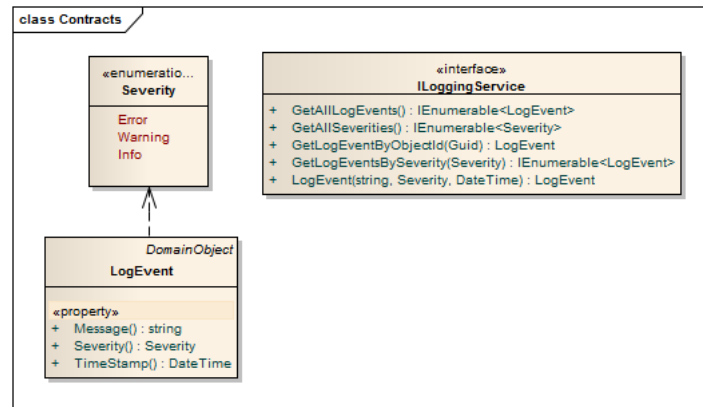


Abbildung 4.11: Klassendiagramm SoCrm.Services.Logging.Contracts

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 Persistence-Service ILogEventPersistenceService aus dem Infrastrukturlayer 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 65) und auch sämtliche Entitäten, welche mit der Benutzerverwaltung verbunden sind.

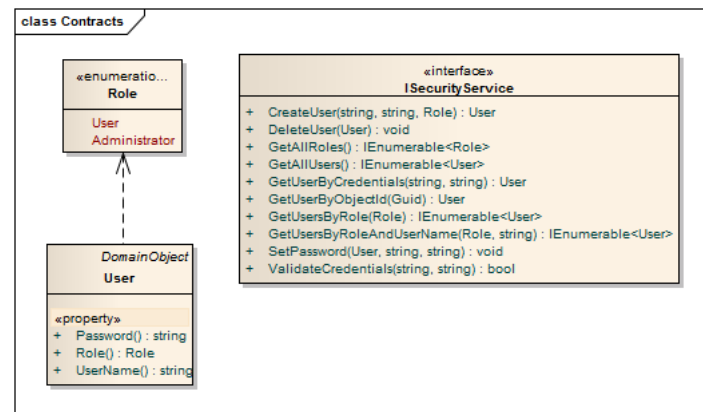


Abbildung 4.12: Klassendiagramm SoCrm.Services.Security.Contracts

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 seinerseits erhält seine Objekte aus dem Persistence-Service IUserPersistenceService, welcher im Infrastructurelayer 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 96) 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`.

5 Verifikation

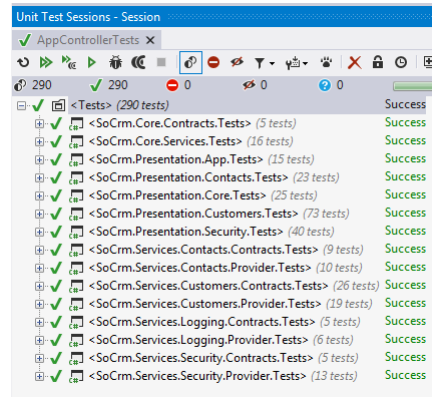


Abbildung 5.1: Resultat Unit Tests

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 Akzeptanztest¹ 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 19) 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 verwendet. Coded UI-Tests² sind automatisierte Tests die auf der Benutzeroberfläche 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 wird 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 dieser Implementierung erfüllt werden und die Akzeptanztests werden als bestanden betrachtet.

¹auch Abnahmetests oder User Acceptance Tests (UAT)

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

5 Verifikation

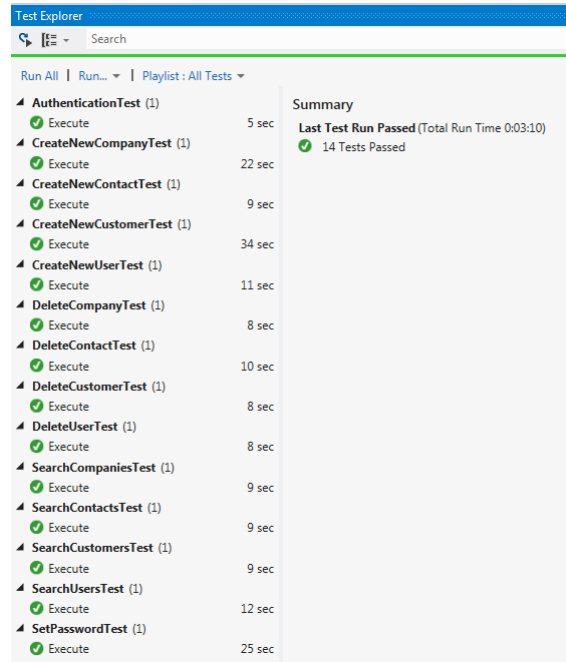


Abbildung 5.2: Resultat Akzeptanztests

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

6.1 Schlussfolgerungen

Erneut: Abschliessende Bemerkung: Fullblown vs lightweight ORMs!

6.2 Abweichungen von der Aufgabenstellung

Akronyme

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

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

Abbildungsverzeichnis

1.1	Wasserfallprozess nach [Hun07]	8
2.1	Systemkontext	20
2.2	Use-Case Benutzer oder Administrator authentifizieren	21
2.3	Use-Case Kunden anzeigen	23
2.4	Use-Case Kunde hinzufügen	24
2.5	Use-Case Kunde löschen	26
2.6	Use-Case Firmen anzeigen	27
2.7	Use-Case Firma hinzufügen	28
2.8	Use-Case Firma löschen	29
2.9	Use-Case Kontakte anzeigen	31
2.10	Use-Case Kontakt hinzufügen	32
2.11	Use-Case Kontakt löschen	33
2.12	Use-Case Benutzer anzeigen	34
2.13	Use-Case Passwort setzen	35
2.14	Use-Case Benutzer hinzufügen	37
2.15	Use-Case Benutzer löschen	38
3.1	Umfrageresultate Welche der folgenden .NET OR-Mapper kennst du?	50
3.2	Umfrageresultate Welche davon hast du schon in einem Projekt eingesetzt?	51
3.3	Umfrageresultate Mit welchem Produkt hast du die besten Erfahrungen gemacht?	52
3.4	Umfrageresultate Welches Produkt schätzt du am verbreitetsten ein?	52
3.5	Umfrageresultate Für ein privates Projekt, welchen ORM würdest du einsetzen?	53
3.6	Kriterien Performance	59
3.7	Komponentendiagramm SoCrm	64
3.8	Klassendiagramm Domänenmodell SoCrm	66
3.9	Klassendiagramm ISecurityService	67
3.10	Klassendiagramm ICustomerService	68
3.11	Klassendiagramm IContactService	69
3.12	Klassendiagramm ILoggingService	70
3.13	Sequenzdiagramm Kontakt hinzufügen	72
3.14	Verteilungsdiagramm SoCrm	74
4.1	Visual Studio Solution	81
4.2	Klassendiagramm SoCrm.Core.Contracts	81

Abbildungsverzeichnis

4.3	Klassendiagramm SoCrm.Infrastructure.Persistence.Contracts	82
4.4	Klassendiagramm SoCrm.Presentation.App	84
4.5	Klassendiagramm SoCrm.Presentation.Contracts	85
4.6	Klassendiagramm SoCrm.Presentation.Core	86
4.7	Klassendiagramm SoCrm.Presentation.Customers	87
4.8	Klassendiagramm SoCrm.Presentation.Security	89
4.9	Klassendiagramm SoCrm.Services.Contacts.Contracts	90
4.10	Klassendiagramm SoCrm.Services.Customers.Contracts	91
4.11	Klassendiagramm SoCrm.Services.Logging.Contracts	92
4.12	Klassendiagramm SoCrm.Services.Security.Contracts	92
5.1	Resultat Unit Tests	95
5.2	Resultat Akzeptanztests	97

Tabellenverzeichnis

1.1	Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument	9
1.2	Phasenplan	10
1.3	Meilensteine	10
2.1	Funktionale Anforderungen an einen Persistenzlayer	13
2.2	Qualitätsanforderungen an den Persistenzlayer	13
2.3	Randbedingungen an den Persistenzlayer	14
2.4	Anforderungen Transaction Script	15
2.5	Anforderungen Domain Model	17
2.6	Anforderungen Table Module	18
2.7	Anforderungen Active Record	19
2.8	Use-Case-Spezifikation Benutzer oder Administrator authentifizieren . . .	22
2.9	Use-Case-Spezifikation Kunden anzeigen	23
2.10	Use-Case-Spezifikation Kunde hinzufügen	25
2.11	Use-Case-Spezifikation Kunde löschen	26
2.12	Use-Case-Spezifikation Firmen anzeigen	27
2.13	Use-Case-Spezifikation Firma hinzufügen	28
2.14	Use-Case-Spezifikation Firma löschen	30
2.15	Use-Case-Spezifikation Kontakte anzeigen	31
2.16	Use-Case-Spezifikation Kontakt hinzufügen.	32
2.17	Use-Case-Spezifikation Kontakt löschen	33
2.18	Use-Case-Spezifikation Benutzer anzeigen	34
2.19	Use-Case-Spezifikation Passwort setzen	36
2.20	Use-Case-Spezifikation Benutzer hinzufügen.	37
2.21	Use-Case-Spezifikation Benutzer löschen	38
3.1	Kriterien Benutzerfreundlichkeit	55
3.2	Kriterien Plattformunterstützung	56
3.3	Vergleichsmatrix	62
3.4	Methoden von ISecurityService	67
3.5	Methoden von ICustomerService	68
3.6	Methoden von IContactService	69
3.7	Methoden von ILoggingService	70
5.1	Testabdeckung	95
5.2	Akzeptanztests	97

Literaturverzeichnis

- [Ams13] AMSTUTZ, Florian: Aufgabenstellung Bachelorarbeit. (2013)
- [Cor11] CORNETT, Steve: Minimum Acceptable Code Coverage. (2011). <http://www.bullseye.com/minimum.html>
- [DC10] DAVID CHOU, Thomas Erl Nitin Gandhi Hanu Kommalapati Brian Loesgen Christoph Schittko Herbjörn Wilhelmsen Mickey W. John deVadoss d. John deVadoss: *SOA with .NET and Windows Azure*. Prentice Hall, 2010
- [Elm05] ELMER, Franz-Josef: Software Engineering, Methodologien. (2005)
- [FRF⁺02] FOWLER, Martin ; RICE, David ; FOEMMEL, Matthew ; HEATT, 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 - Ressourcen 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

Literaturverzeichnis

- [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)
- [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)