

Yet Another Encrypted Messenger

Florian Amstutz <florian@amstutz.nu>

14. Mai 2012

Semesterarbeit an der Zürcher Hochschule für Angewandte
Wissenschaften

Inhaltsverzeichnis

1. Einführung	4
1.1. Management Summary	4
1.2. Über die Semesterarbeit	4
1.3. Softwareentwicklungsprozess	4
1.4. Projektplanung	6
1.4.1. Phasenplanung	6
1.4.2. Meilensteine	7
2. Anforderungen	8
2.1. Was sind Anforderungen?	8
2.1.1. Arten von Anforderungen	8
2.2. Systemkontext	9
2.3. Use-Case-Spezifikationen	10
2.3.1. Gespräch beitreten	11
2.3.2. Gespräch verlassen	12
2.3.3. Nachricht senden	13
2.3.4. Nachricht empfangen	15
2.4. Mockups	17
2.4.1. Dialog Gespräch beitreten	18
2.4.2. Gesprächsdialog	18
3. Konzept	20
3.1. Toolchain	20
3.2. Bausteinsicht	20
3.2.1. Komponentendiagramm	21
3.2.2. Domänenmodell	21
3.2.3. Service Contracts	23
3.2.4. Kryptoalgorithmen	26
3.2.5. Server	27
3.2.6. Client	28
3.3. Laufzeitsicht	28
3.3.1. Gespräch beitreten	29
3.3.2. Gespräch verlassen	30
3.3.3. Nachricht senden	31
3.4. Verteilungssicht	33
3.4.1. Verteilungsdiagramm	33

Inhaltsverzeichnis

4. Implementierung	35
4.1. Implementierungsmethodik	35
4.1.1. Testgetriebene Entwicklung	35
4.2. Verwendete Technologien	37
4.2.1. MEF	37
4.2.2. WCF	37
4.2.2.1. Bindings	37
4.2.2.2. Callbacks über NetHttpDualBinding	37
4.2.3. WPF	37
4.2.3.1. Silverlight	37
4.2.4. NuGet	37
4.3. Kryptosysteme	38
4.4. Aufgetretene Probleme	38
5. Test	39
5.1. Unit-Tests	39
5.2. Akzeptanztests	39
6. Fazit	40
6.1. Aufwand SOLL/IST Vergleich	40
6.2. Schlussfolgerungen	40
A. Anhang	41

1. Einführung

Als einführendes Kapitel dieses Dokuments wird die Semesterarbeit als Projekt kurz vorgestellt und es werden die Rahmenbedingungen der Semesterarbeit zusammengefasst niedergeschrieben. Weiter werden verschiedene Softwareentwicklungsprozesse vorgestellt sowie die für diese Semesterarbeit verwendete Methode erklärt. Am Ende dieses Kapitels wird unter Benutzung der vorgestellten Softwareentwicklungsmethode eine Projektplanung erstellt und die wichtigsten Meilensteine definiert.

1.1. Management Summary

Mit dem zunehmenden Aufkommen von Attacken und gezieltem Abhören von Echtzeitkommunikation via E-Mail oder Instant Messaging steigt der Bedarf an eine sichere und einfache Übertragungsart von Nachrichten oder Daten.

Als Nutzer eines Kommunikationskanals über das öffentliche Internet will ich die Möglichkeit haben meine privaten Daten verschlüsselt und sicher an einen oder mehrere Empfänger übertragen zu können. Ich will dabei eine einfach zu bedienende Applikation zur Verfügung haben um meine geheimen Daten übertragen zu können und so potentiellen Mithörern keine Klartextinformationen zur Verfügung zu stellen.

Diese Applikation soll als Prototyp im Rahmen der Semesterarbeit im dritten Studienjahr an der ZHAW entwickelt werden. Dabei wird der Schwerpunkt der Arbeit auf der methodischen Vorgehensweise der Softwareentwicklung und weniger auf der Implementierung der kryptografischen Algorithmen.

1.2. Über die Semesterarbeit

Gemäss Reglement der ZHAW (siehe [Stern(2010)]) dient die Semesterarbeit als Vorbereitung zur Bachelorarbeit. Sie besteht aus einem konzeptionellen Teil und einem Umsetzungsteil, wobei der Schwerpunkt auf der Umsetzung liegen soll.

Der Aufwand für die Fertigstellung der Semesterarbeit beträgt mindestens 120 Stunden und schliesst mit einer Präsentation vor dem Betreuer und einer Vertretung der Leitung des Studiengangs Informatik ab.

1.3. Softwareentwicklungsprozess

Software lässt sich nach einer Vielzahl von Prozessen und Modellen entwickeln. Von iterativen Vorgehen wie Scrum über komplexe Modelle wie RUP hin zu klassischen, linearen Vorgehen wie dem Wasserfallmodell oder dem V-Modell. Nach [Starke(2011)] ist die

1. Einführung

Auswahl des Entwicklungsprozesses eine der schwierigsten Entscheidungen, die man bei einem Softwareprojekt treffen muss. Häufig besitzen Unternehmungen bereits etablierte, auf sie zugeschnittene Entwicklungsmodelle, die mehr oder weniger gut zur Organisation der Unternehmung passen. Ein ungünstig gewählter oder nicht vollständig eingeführter und gelebter Entwicklungsprozess ist nach [Starke(2011)] 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 in meist fünf verschiedene Phasen auf. Dabei kann erst mit der nächsten Phase begonnen werden wenn die Lieferegebnisse und die Ergebnisdokumentation der vorhergehenden Phase fertiggestellt und abgenommen worden sind. Das Wasserfallmodell wurde ausgewählt, da die jeweiligen Phasen eindeutig abgeschlossen werden können, da der Betreuer als einziger, externer Stakeholder des Projekts zu festdefinierten Phasen Einfluss auf das Projekt ausübt und danach keine Möglichkeit mehr besitzt, den Projektverlauf zu beeinflussen. Die grössten Nachteile des Wasserfallmodells sind nach [Elmer(2005)] ein Abgrenzungsproblem zwischen den Phasen sowie die Schwierigkeit des Abschlusses einzelner Phasen. Dadurch dass der Betreuer nur in den Phasen Requirements und Design Einfluss auf das Projekt nehmen kann und Student als einziger Stakeholder den Abschluss der Phasen abnimmt sowie den Ablauf der Phasen innerhalb des Projekts steuern können diese Nachteile umgangen werden.

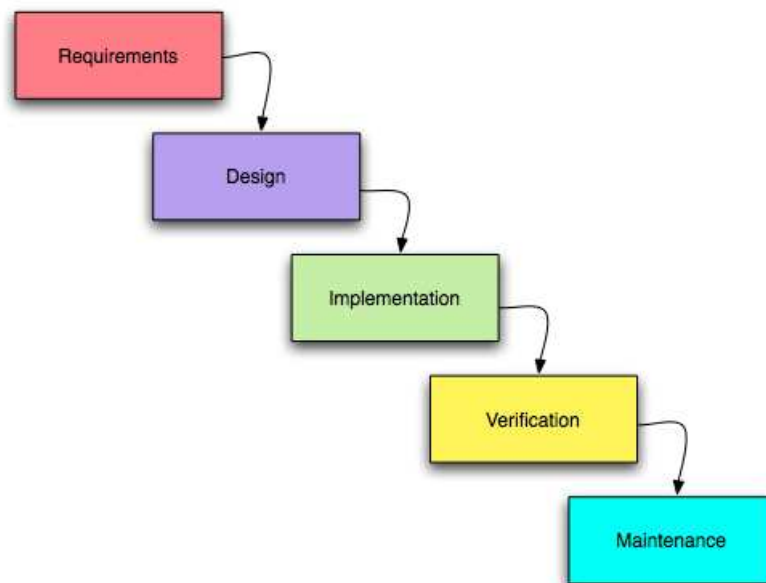


Abbildung 1.1.: Wasserfallprozess nach [VO(2007)]

Zu Beginn des Wasserfallmodells steht das Sammeln und Dokumentieren der Anforderungen (Requirements). Wenn die Anforderungen umfänglich und in hohem Detail-

1. Einführung

lierungsgrad niedergeschrieben sind, werden diese vom Auftraggeber abgenommen und das Projekt geht in die Design-Phase über. Die zu entwickelnde Software wird auf verschiedenen Ebenen von Softwarearchitekten designed und eine Blaupause wird erstellt, nach welcher sich die Software Entwickler 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 Implementationsphase werden die Softwarekomponenten verschiedener Teams integriert. Nachdem die Implementierungs- und Integrationsphasen abgeschlossen sind, wird das Softwareprodukt getestet und allfällige Fehler aus früheren Phasen werden zu diesem Zeitpunkt entfernt. Danach wird das Softwareprodukt installiert und später in der Wartungsphase (Maintenance) um weitere Funktionalitäten erweitert beziehungsweise werden weitere Bugs behoben.

Die Struktur dieses Dokuments hält sich auch an den Wasserfallprozess nach [VO(2007)] (siehe Tabelle 1.1). Die Phase Maintenance wird dabei ausgelassen, da sich die innerhalb des Projekts entwickelte Applikation nach Abschluss der Verifizierungsphase noch im Prototypenstand befinden wird.

Phase	Kapitelüberschrift	Seite
Requirements	Anforderungen	8
Design	Konzept	20
Implementation	Implementierung	35
Verification	Test	39

Tabelle 1.1.: Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument

1.4. Projektplanung

Nach Reglement der ZHAW (siehe [Stern(2010)]) muss die Semesterarbeit sechs Monate nach Freigabe beendet sein. Um die Planbarkeit der Semesterarbeit zu erhöhen wurde das Projekt vorgängig in einzelne Phasen unterteilt, welche auf die bekannten Meilensteine enden.

Die Projektplanung dient dazu möglichst frühzeitig im Verlauf der Seminararbeit schon Teile der Dokumentation fertiggestellt zu haben, so dass nicht in den letzten zwei Wochen vor der Abgabe der Arbeit übermässig viel Zeit und Aufwand in die Arbeit investiert werden muss. Ausserdem sinkt das Risiko, das nicht alle erwarteten Resultate der Arbeit geliefert werden können, da die Lieferergebnisse bei Phasenende schon in genügend hoher Qualität vorliegen, dass diese abgegeben werden können.

1.4.1. 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

1. Einführung

Phase über. Die Phasen in diesem Projekt wurden häufig so modelliert, dass ihr Endzeitpunkt mit dem Erreichen eines Meilensteins zusammenliegt. Das heisst, das bei Erreichen des Meilensteins die vorhergehende Phase zwingend abgeschlossen sein werden muss.

Vor dem eigentlichen Projektstart werden geeigneten Themen für die Semesterarbeit evaluiert, ein Betreuer gesucht sowie ein Projektantrag in EBS erfasst. Diese Phase endet im Kick-Off-Meeting zwischen Betreuer und Student und der formalen Freigabe der Semesterarbeit durch die Studiengangsleitung. Das Erheben und Dokumentieren der Anforderungen ist die erste Phase der eigentlichen Projektumsetzung 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 am Konzept vorschlägt. Nach Fertigstellung des Konzepts folgt die Implementierungsphase und Testphase, nach denen die Arbeit abgegeben werden muss. Als letzte Phase folgt das Erarbeiten der Präsentation, die bei Phasenende vorgetragen wird.

Phase	Von	Bis	Ergebnis	Endet in
Themenevaluierung	15.10.2011	04.11.2011	Antrag in EBS	M1, M2
Erfassen der Anforderungen	04.11.2011	01.12.2011	Dokumentierte Anforderungen	
Erarbeiten des Konzepts	01.12.2011	13.04.2012	Umfassendes Konzept	M3
Implementierung des Konzepts	13.04.2012	01.05.2012	Implementierung	
Überprüfen und Test des Konzepts	01.05.2012	23.05.2012	Erfüllte Integrationstests	M4
Erarbeiten der Präsentation	23.05.2012	06.06.2012	Fertiggestellte Präsentation	M5

Tabelle 1.2.: Phasenplan

1.4.2. 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 Phasen überein.

Kürzel	Meilenstein	Datum
M1	Kick-Off	04.11.2011
M2	Freigabe der Arbeit	06.11.2012
M3	Design Review	13.04.2012
M4	Abgabe der Arbeit	23.05.2012
M5	Präsentation der Semesterarbeit	06.06.2012

Tabelle 1.3.: Meilensteine

2. Anforderungen

Gemäss dem verwendeten Wasserfallmodell werden als erstes die Anforderungen erhoben. Dazu wird der Begriff “Anforderung” definiert und auf verschiedene Arten von Anforderungen eingegangen. Anschliessend wird der Systemkontext aufgezeigt sowie die konkreten Anforderungen an YAEM als Use-Cases modelliert und spezifiziert. Der Abschluss dieses Kapitels wie auch der Anforderungsphase bildet die Erstellung und Erklärung der für den Benutzer sichtbaren Dialogfenster.

2.1. Was sind Anforderungen?

Die erste Phase des Wasserfallmodells beschäftigt sich mit den Anforderungen an das zu entwickelnde Softwareprodukt. Damit das Entwicklungsprodukt zum Erfolg geführt werden kann, muss zunächst bekannt sein, was die Anforderungen an das System sind und diese müssen geeignet dokumentiert sein. Nach [IEE(1990)] wird eine Anforderung wie folgt definiert:

Anforderung Eine Anforderung ist:

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

Die Anforderungen an das im Rahmen der Semesterarbeits zu entwickelnden Systems werden in Use-Case-Diagrammen modellhaft dargestellt und als Use-Case-Spezifikationen ausformuliert. Auf eine natürlichsprachige Dokumentation der Anforderungen wird verzichtet, da die Anforderungen aufgrund der Use-Case-Diagrammen verständlich genug sind und alle zusätzlich zu den Diagrammen zu beachtenden Punkte in den Use-Case-Spezifikationen enthalten sind.

2.1.1. Arten von Anforderungen

Nach [Pohl and Rupp(2011)] unterscheidet man im Allgemeinen zwischen drei Arten von Anforderungen:

2. Anforderungen

- Funktionale Anforderungen legen die Funktionalität fest, die das geplante System zur Verfügung stellen soll. Sie werden typischerweise in Funktions-, Verhaltens- und Strukturanforderungen unterteilt.
- Qualitätsanforderungen legen gewünschte Qualitäten des zu entwickelnden Systems fest und beeinflussen häufig, in grösserem Umfang als die funktionalen Anforderungen, die Gestalt der Systemarchitektur. Typischerweise beziehen sich Qualitätsanforderungen auf die Performance, die Verfügbarkeit, die Zuverlässigkeit, die Skalierbarkeit oder die Portabilität des betrachteten Systems. Anforderungen dieses Typs werden häufig auch der Klasse “nicht funktionaler Anforderungen” zugeordnet.
- Randbedingungen (auch: Rahmenbedingungen) 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, d.h. den Lösungsraum im Entwicklungsprozess ein.

2.2. Systemkontext

Als erster Schritt in der Erhebung und Dokumentierung wird der Systemkontext ermittelt. Es wird eine Sollperspektive eingenommen, d.h., es wird eine Annahme getroffen, wie das geplante System sich in die Realität integriert. 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.

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 YAEM wird wie folgt dargestellt. Die Benutzer als Stakeholder an das System senden und empfangen Nachrichten und befinden sich innerhalb des Systemkontexts da sie direkt mit dem System interagieren. Die Verschlüsselungsalgorithmen sind in der Fachliteratur klar geregelt und normiert und geben aus diesem Grund die konkreten Implementierungsvorschriften an das System vor.

2. Anforderungen

Systemkontext YAEM

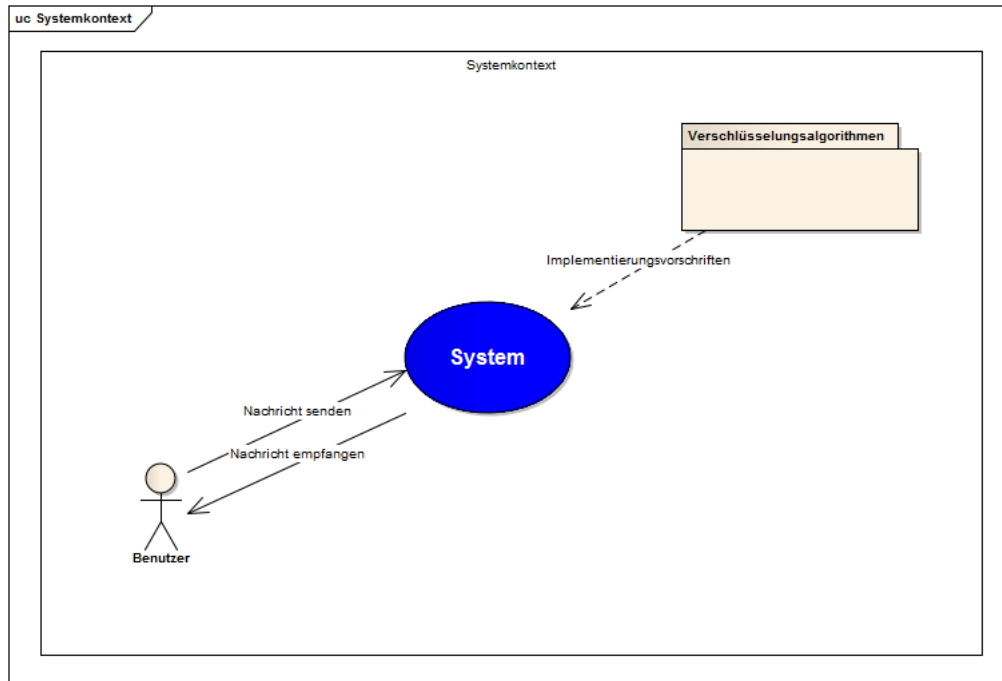


Abbildung 2.1.: Systemkontext

2.3. Use-Case-Spezifikationen

Nach [Pohl and Rupp(2011)] zeigen Use-Case-Diagramme die aus einer externen Nutzungssicht wesentlichen Funktionalitäten des betrachteten Systems sowie spezifische Beziehungen der einzelnen Funktionalitäten untereinander bzw. zu Aspekten in der Umgebung des Systems. Abgesehen vom Namen eines Use-Cases und dessen Beziehungen dokumentieren Use-Case-Diagramme allerdings keinerlei weitere Informationen über die einzelnen Use-Cases, wie z.B. die Systematik der Interaktion eines Use Case mit Akteuren in der Umgebung. Diese Informationen werden unter Verwendung einer geeigneten Schablone zusätzlich zum Use-Case-Diagramm textuell dokumentiert.

Alle funktionalen Anforderungen (siehe 2.1.1 auf Seite 8) werden nun als Use-Cases modelliert und spezifiziert¹. Als Quellen der Anforderungen dienten der Betreuer, die Reglemente der ZHAW betreffend der Semesterarbeit sowie der Student in der Rolle als Benutzer des Systems. Zusätzlich zu den Use-Cases und der dazugehörigen Use-Case-Spezifikation wird vorgängig in Prosatext der Anwendungsfall beschrieben. Aus Gründen

¹Die verwendete Schablone für die Use-Case-Spezifikationen stammt aus [Pohl and Rupp(2011)] und dient zur zweckmässigen Strukturierung von Typen von Informationen, die einen Use-Case betreffen. Die Abschnitte Autor, Quelle, Verantwortlicher und Qualität werden ausgelassen, da sie für die Semesterarbeit keine Relevanz besitzen.

2. Anforderungen

der Übersichtlichkeit und des überschaubaren Umfangs dienen diese Use-Cases primär als Anforderungen an das zu entwickelnde Softwaresystem. Jedes Use-Case wird im Rahmen der Verifizierungsphase (siehe 5 auf Seite 39) als Integrationstest einzeln getestet.

2.3.1. Gespräch beitreten

Ein Benutzer möchte Nachrichten über YAEM versenden und startet die Applikation. Er wählt einen Benutzernamen, stellt eine Verbindung zum Server her und nimmt am Gespräch teil. Er kann nun anderen Teilnehmern des Gesprächs Nachrichten versenden.

Use-Case

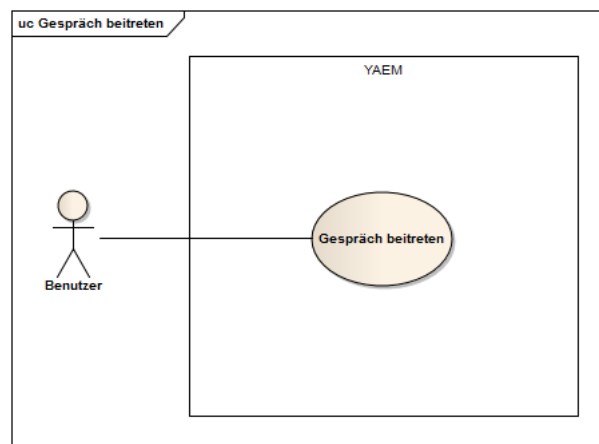


Abbildung 2.2.: Use-Case Gespräch beitreten

2. Anforderungen

Use-Case-Spezifikation

Abschnitt	Inhalt
Bezeichner	UC1
Name	Gespräch beitreten
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Hoch
Beschreibung	Der Benutzer tritt einem Gespräch bei.
Auslösendes Ereignis	Benutzer möchte einem Gespräch beitreten.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist nicht schon einem Gespräch beigetreten.
Nachbedingung	Der Benutzer kann Nachrichten versenden und Nachrichten anderer Gesprächsteilnehmer empfangen.
Ergebnis	Session-Ticket wird erstellt.
Hauptszenario	1. Der Benutzer wählt einen Benutzernamen. 2. Der Benutzer stellt eine Verbindung zum Server her. 3. Der Server erstellt eine Session-Ticket für den Benutzer und gibt ihm dieses zurück.
Alternativszenarien	2a. Der gewählte Benutzername ist bereits im Gespräch vorhanden. 2a1. Der Benutzer wird aufgefordert einen anderen Benutzernamen auszuwählen.
Ausnahmeszenarien	Auslösendes Ereignis: Der Benutzer kann keine Verbindung zum Server herstellen.

Tabelle 2.1.: Use-Case-Spezifikation Gespräch beitreten

2.3.2. Gespräch verlassen

Der Benutzer ist im Gespräch und möchte dieses Verlassen. Er schliesst die Applikation und meldet sich am Server vom Gespräch ab. Andere Teilnehmer des Gesprächs können ihm nun keine Nachrichten mehr senden.

2. Anforderungen

Use-Case

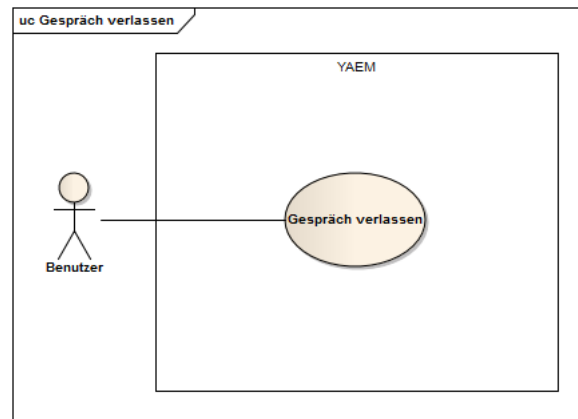


Abbildung 2.3.: Use-Case Gespräch verlassen

Use-Case-Spezifikation

Abschnitt	Inhalt
Bezeichner	UC2
Name	Gespräch verlassen
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Hoch
Beschreibung	Der Benutzer verlässt ein Gespräch.
Auslösendes Ereignis	Benutzer möchte eine Gespräch verlassen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist einem Gespräch beigetreten.
Nachbedingung	Der Benutzer kann erneut einem Gespräch beitreten.
Ergebnis	Session-Ticket ist abgelaufen.
Hauptszenario	1. Der Benutzer verlässt das Gespräch. 2. Der Server erklärt das Session-Ticket des Benutzers für abgelaufen und sendet das aktualisierte Ticket dem Benutzer zu.
Alternativszenarien	Keine
Ausnahmeszenarien	Keine

Tabelle 2.2.: Use-Case-Spezifikation Gespräch verlassen

2.3.3. Nachricht senden

Dies ist der wichtigste und meistgenutzte Anwendungsfall des Systems. Der Benutzer als Sender möchte einem oder mehreren Teilnehmern des Gesprächs (Empfänger) eine

2. Anforderungen

Nachricht senden. Er kann dabei wählen, ob er diese verschlüsselt oder unverschlüsselt versenden möchte. Sendet der Benutzer die Nachricht verschlüsselt, so werden zuerst der Initialisierungsvektor festgelegt sowie der Schlüssel gewählt. Danach wird die Nachricht an den oder die Empfänger übermittelt und startet den Anwendungsfall “Nachricht empfangen” (siehe 2.3.4 auf der nächsten Seite).

Use-Case

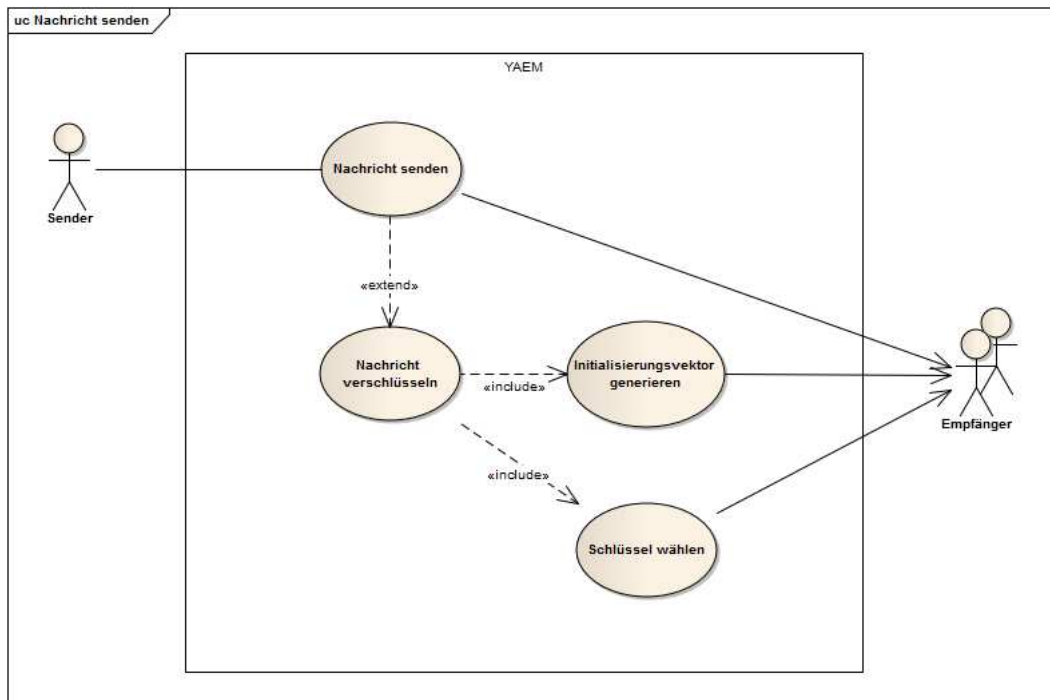


Abbildung 2.4.: Use-Case Nachricht senden

2. Anforderungen

Use-Case-Spezifikation

Abschnitt	Inhalt
Bezeichner	UC3
Name	Nachricht senden
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: mittel
Kritikalität	Hoch
Beschreibung	Der Benutzer versendet eine Nachricht.
Auslösendes Ereignis	Benutzer möchte eine Nachricht senden.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist im Gespräch angemeldet und besitzt eine gültiges Session-Ticket.
Nachbedingung	Der Benutzer kann erneut eine Nachricht versenden und Nachrichten anderer Gesprächsteilnehmer empfangen.
Ergebnis	Die Empfänger haben die versendete Nachricht empfangen.
Hauptszenario	1. Der Benutzer erfasst die zu versenden Nachricht 2. Der Benutzer wählt einen Kryptoalgorithmus aus. 3. Der Benutzer generiert einen Initialisierungsvektor. 4. Der Initialisierungsvektor wird an alle Empfänger gesendet. 5. Der Benutzer wählt einen Schlüssel. 6. Der Schlüssel wird an alle Empfänger gesendet. 7. Der Benutzer verschickt die (verschlüsselte) Nachricht.
Alternativszenarien	2a. Der Benutzer wählt keinen Kryptoalgorithmus aus. 2a1. Der Benutzer versendet die Nachricht unverschlüsselt. 3a. Der Benutzer hat bereits einen Initialisierungsvektor erstellt oder einen Initialisierungsvektor von einem anderen Teilnehmer des Gesprächs erhalten und generiert keinen neuen Initialisierungsvektor. 4a. Der Benutzer hat bereits einen Schlüssel erstellt oder einen Schlüssel von einem anderen Teilnehmer des Gesprächs erhalten und wählt keinen neuen Schlüssel.
Ausnahmeszenarien	Auslösendes Ereignis: Der Benutzer kann keine Verbindung zum Server herstellen.

Tabelle 2.3.: Use-Case-Spezifikation Nachricht senden

2.3.4. Nachricht empfangen

Dieser Anwendungsfall wird nicht vom Benutzer ausgelöst, sondern vom System. Sobald eine Nachricht, die an den Benutzer gerichtet ist, eintrifft, wird der Anwendungsfall gestartet. Ist die Nachricht verschlüsselt, versucht das System mit vorhandenem Initialisie-

2. Anforderungen

rungsvektor und Schlüssel die Nachricht zu entschlüsseln und dem Benutzer darzustellen. Ist die Nachricht unverschlüsselt, so wird diese dem Benutzer direkt angezeigt.

Use-Case

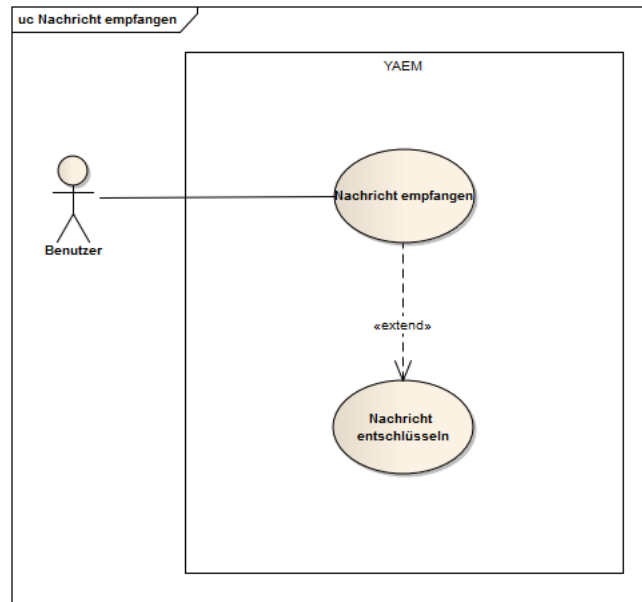


Abbildung 2.5.: Use-Case Nachricht empfangen

2. Anforderungen

Abschnitt	Inhalt
Bezeichner	UC4
Name	Nachricht empfangen
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: mittel
Kritikalität	Hoch
Beschreibung	Der Benutzer empfängt eine Nachricht.
Auslösendes Ereignis	Ein anderer Teilnehmer des Gesprächs versendet eine Nachricht.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist im Gespräch angemeldet und besitzt eine gültiges Session-Ticket. Ein Teilnehmer des Gesprächs versendet eine Nachricht.
Nachbedingung	Der Benutzer kann Nachrichten versenden und Nachrichten anderer Gesprächsteilnehmer empfangen.
Ergebnis	Die Nachricht wird dem Benutzer angezeigt.
Hauptszenario	1. Der Benutzer empfängt die Nachricht und prüft ob diese verschlüsselt ist. 2. Der Benutzer verwendet den Initialisierungsvektor und Schlüssel zum entschlüsseln der Nachricht. 3. Die entschlüsselte Nachricht wird angezeigt.
Alternativszenarien	1a. Ist die Nachricht nicht verschlüsselt, wird sie direkt angezeigt.
Ausnahmeszenarien	Ist kein Initialisierungsvektor, Schlüssel oder Implementierung des verwendeten Kryptoalgorithmus vorhanden, so wird der unlesbare Geheimtext angezeigt.

Tabelle 2.4.: Use-Case-Spezifikation Nachricht empfangen

Use-Case-Spezifikation

2.4. Mockups

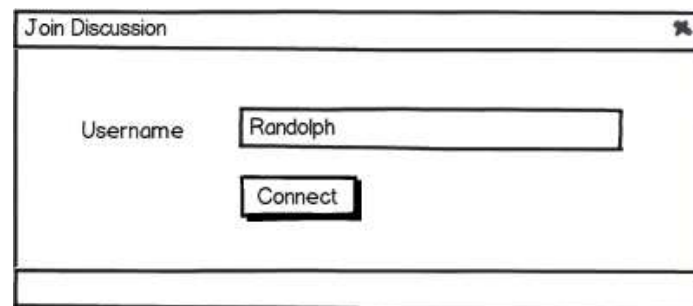
Mockups dienen zur Visualisierung der Benutzeroberfläche des zu entwickelnden Softwareprodukts und werden häufig bereits in der Anforderungsphase zusammen mit den Stakeholdern entwickelt. Sie liefern dem Softwareentwickler in der Implementationsphase (siehe 4 auf Seite 35) ein Grundgerüst für die einzelnen Dialogfenster. Je nach Kundenzielgruppe und Wichtigkeit der Mensch-Maschine-Schnittstelle wird mehr oder weniger Zeit in die Entwicklung der Mockups investiert. Häufig werden diese mit zusammen mit Psychologen entwickelt und enthalten sehr weniger Interpretationsspielraum für den Softwareentwickler.

2. Anforderungen

2.4.1. Dialog Gespräch beitreten

Startet der Benutzer die Applikation so wird ihm als erstes die Möglichkeit gegeben, einen Benutzernamen zu wählen, unter welchem er im Gespräch Nachrichten verschicken möchte. Dabei wird der Benutzername auf eine Länge von 255 Zeichen beschränkt und er darf keine Sonderzeichen enthalten. Nach einem Klick auf Connect werden diese Bedingungen geprüft und eine Verbindung zum Server hergestellt. Bei Erfolg wird der Dialog geschlossen und dem Benutzer der Gesprächsdialog (siehe 2.4.2) angezeigt.

Mockup



The mockup shows a standard Windows-style dialog box titled "Join Discussion". It contains a label "Username" and a text input field with the text "Randolph". Below the input field is a button labeled "Connect". The dialog box has a close button in the top right corner.

Abbildung 2.6.: Mockup Dialog Gespräch beitreten

2.4.2. Gesprächsdialog

Ist der Benutzer einem Gespräch beigetreten so verwendet er den Gesprächsdialog zum Senden und Empfangen von Nachrichten. Das Gesprächsprotokoll zeigt einen zeitlich geordneten Ablauf aller gesendeten und empfangenen Nachrichten. Weiterhin werden Aktionen wie ein Initialisierungsvektoraustausch oder das Setzen des Schlüssels eines Verschlüsselungsalgorithmus angezeigt. Auf der rechten Seite werden alle dem Gespräch beigetretenen Benutzer angezeigt.

Über die Textbox unterhalb des Gesprächsprotokolls lassen sich Nachrichten erfassen, die Nachricht muss dabei mindestens ein Zeichen lang sein. In der Dropdownliste rechts neben der Textbox werden alle installierten Kryptoalgorithmen angezeigt, zusätzlich zur Voreinstellung "<None>", welche die Nachricht unverschlüsselt versendet. Ist kein Algorithmus angezeigt, so kann der Benutzer keine Auswahl treffen. Wählt der Benutzer einen Algorithmus aus, für welchen noch kein Initialisierungsvektor vorhanden ist, so wird eine Vektor generiert und den anderen Gesprächsteilnehmern zugesendet. Gleichzeitig wählt der Benutzer einen Schlüssel aus. Beim klicken auf den "Send" Button, wird die Nachricht an die anderen Gesprächsteilnehmer versendet.

Beim Schliessen der Applikation verlässt der Benutzer das Gespräch und der Gesprächsdialog schliesst sich selbst.

2. Anforderungen

Mockup

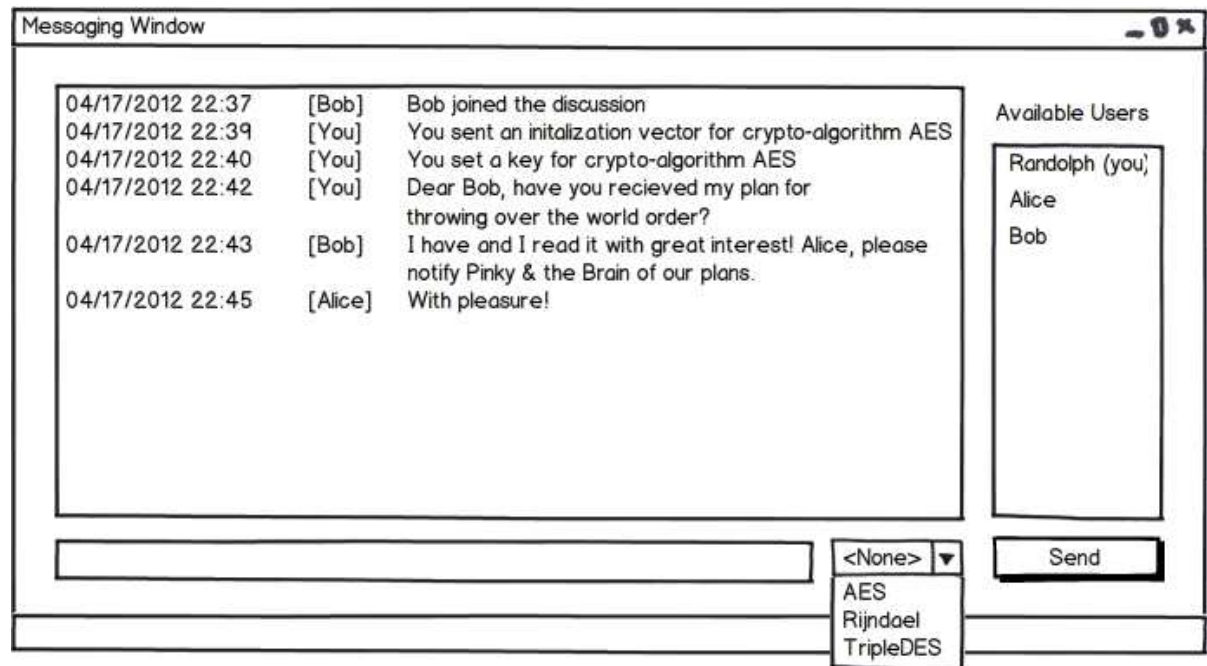


Abbildung 2.7.: Mockup Gesprächsdialog

3. Konzept

Die Konzeptphase¹ des Wasserfallmodells behandelt die Entwicklung eines vollständigen und umfassenden Lösungskonzepts auf Basis der dokumentierten Anforderungen (nach [Bernd Oestereich(2006)]). Als Grundlage für das Konzept wird in einem ersten Schritt die Toolchain erfasst, welche die weiteren (technologischen) Rahmenbedingungen für das Konzept vorgibt. Weiter wird zuerst das Konzept von der Bausteinsicht betrachtet, es wird von der Komponentenebene bis zur Klassenebene das System modelliert. Als weitere Sicht wird die Laufzeitsicht und die Verteilungssicht des Systems beleuchtet und spezifiziert.

3.1. Toolchain

Der erste Schritt der Konzepterstellung ist das Erarbeiten und Evaluieren der Toolchain. Die Toolchain bestimmt mit welchen Frameworks und Entwicklungswerkzeugen der Entwickler später arbeiten wird und gibt einen groben technischen Rahmen vor, der bei der Konzepterstellung berücksichtigt werden muss.

Basis der verwendeten Tools und Frameworks wird Microsofts .NET Framework in der Version 4.0 sein. Dies primär aufgrund der Möglichkeiten von WCF² zur Realisierung von Webservices und WPF bzw. Silverlight für Applikationen mit Oberflächen für die Windows Plattform. Als Entwicklungswerkzeug wird Microsoft Visual Studio 2010 eingesetzt und als Package Manager NuGet³. Zur Dateiverwaltung der online Hosting-Dienst Github⁴ verwendet, welcher als Versionsverwaltungssystem Git einsetzt.

3.2. Bausteinsicht

Nach [Starke(2011)] und [Hruschka and Starke(2012)] lassen sich unter dem Begriff “Bausteine” sämtliche Software- oder Implementierungskomponenten zusammenfassen. Sie stellen letztendliche Abstraktionen von Quellcode dar. 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 statische Aspekte von Systemen. In dieser Hinsicht entsprechen sie den konventionellen Implementierungsmodellen.

¹auch Designphase genannt

²Eine umfangreiche Einführung in WCF ist in der MSDN unter <http://msdn.microsoft.com/en-us/library/ee958158.aspx> erhältlich.

³Mehr zum Package Manager NuGet ist unter <http://nuget.codeplex.com/> zu finden.

⁴Das Github Repository für YAEM ist unter <https://github.com/famstutz/YAEM> öffentlich verfügbar.

3.2.1. Komponentendiagramm

Das Komponentendiagramm stellt das System YAEM aus Vogelperspektive dar und ist die höchstabstrahierte Ansicht der Bausteinsicht.

Der ServiceHost stellt die Serverapplikation des Systems dar und implementiert die beiden Schnittstellen⁵ *IUserService* (siehe 3.2.3 auf Seite 24) und *IMessagingService* (siehe 3.2.3 auf Seite 24). Er stellt die Schnittstellen als Webservice den Clientapplikationen zur Verfügung. Die Clientapplikationen (die in einer Vielzahl von Frameworks implementiert sein können) benutzen diese Schnittstellen um mit der Serverapplikation zu kommunizieren. Die Clientapplikationen benutzen MEF um dynamisch Kryptoalgorithmen (siehe 3.2.4 auf Seite 26) zur Laufzeit laden zu können. Dadurch wird gewährleistet, dass der Server zu keinem Zeitpunkt die verschlüsselten Nachrichten, die über ihn versendet werden, im Klartext lesen kann.

Komponentendiagramm

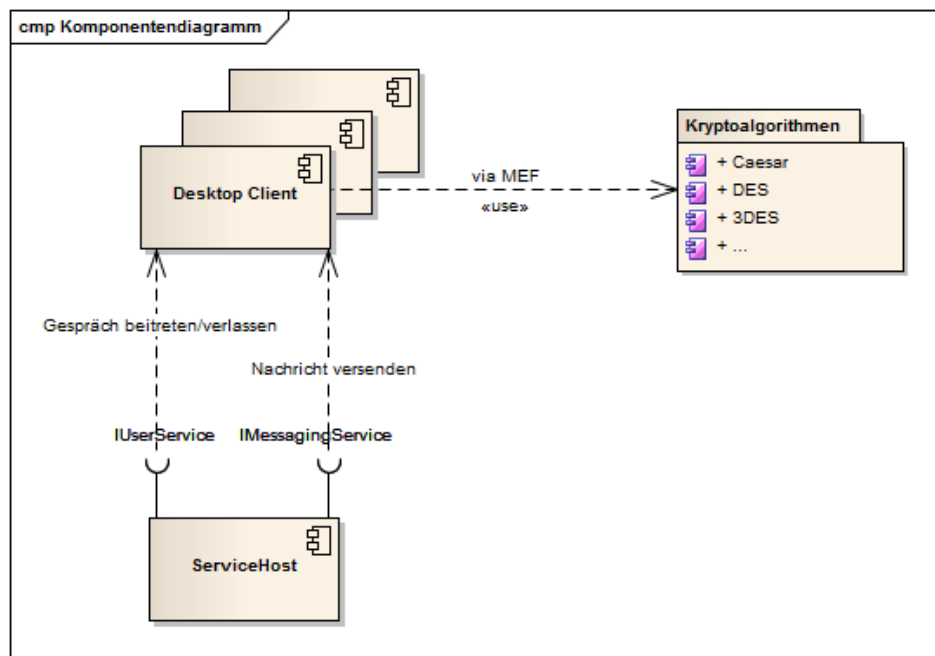


Abbildung 3.1.: Komponentendiagramm

3.2.2. Domänenmodell

Das Domänenmodell umfasst nur die Businessobjekte, die über die Serviceschnittstelle vom Client- an die Serverapplikation beziehungsweise umgekehrt übertragen werden.

YAEM verwendet ein vergleichsweise simples Domänenmodell. Jeder Benutzer wird als *User*objekt im Repository gespeichert. Bei erfolgreichem Gesprächsbeitritt erstellt

⁵In diesem Zusammenhang auch Service Contracts genannt

3. Konzept

der Server ein *Ticket*objekt, welches an den Client zurückgegeben wird. Wird eine Nachricht versendet, so erstellt der Client ein *Message*objekt, welches die Nachricht selbst als Bytearray in der Eigenschaft *Payload* speichert. Will der Benutzer eine verschlüsselte Nachricht versenden, so setzt er die Eigenschaft *Algorithm* des *Message*objekts auf einen Wert des Enumerators *CryptoAlgorithm*, der ungleich *None* ist. Wenn die Eigenschaft *Algorithm* gesetzt ist, so muss der Benutzer den *Payload* verschlüsselt in der *Message* ablegen.

Sämtliche Domänenobjekte leiten von *ObjectBase* ab. *ObjectBase* enthält einen Schlüssel *Key* in Form einer Globally Unique Identifier zur eindeutigen Identifizierung der Objekte in den Repositories. Weiterhin implementiert *ObjectBase* das Interface *INotifyPropertyChanged*⁶, welches in WPF und Silverlight dazu dient, UI-Elemente, die an Datenobjekte gebunden sind, über geänderte Eigenschaften zu informieren.

⁶Die genaue Schnittstellenbeschreibung ist in der MSDN unter <http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged.aspx> zu finden.

3. Konzept

Klassendiagramm

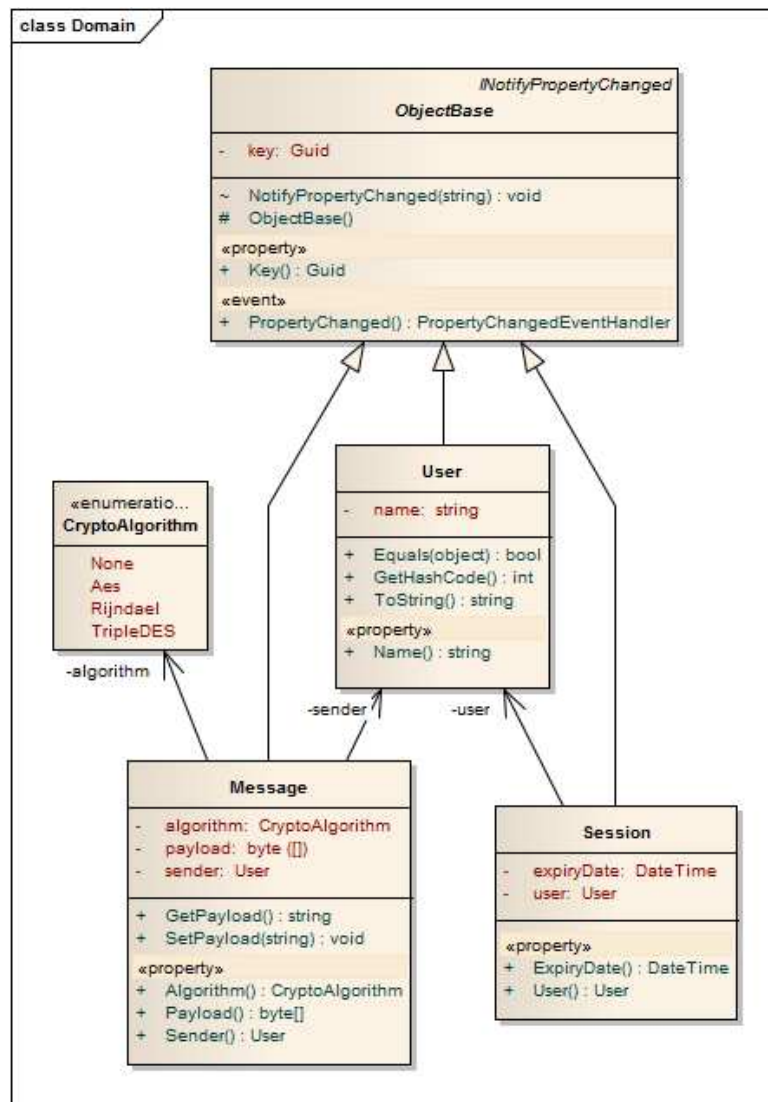


Abbildung 3.2.: Klassendiagramm Domänenmodell

3.2.3. Service Contracts

Ein Service Contract ist das Ausstellen einer Schnittstelle zur Kommunikation verschiedener Applikationen innerhalb eines verteilten Systems. Häufig werden diese Contracts als Webservice angeboten, da sie dadurch plattform- und frameworkunabhängig implementiert werden können.

Typischerweise umfasst ein Service Contract mehrere Operationen, deren Rückgabewerte als XML-Fragmente an die konsumierende Applikation zurückgegeben werden. Aus-

3. Konzept

serdem ist ein Service Contract per Definition grundsätzlich zustandslos, er behandelt mehrere Anfragen (auch desselben Auftraggebers) immer als unabhängige Transaktionen. Insbesondere werden Anfragen ohne Bezug zu früheren Anfragen behandelt und keine Sitzungsinformationen ausgetauscht.

IUserService

Das Interface *IUserService* stellt die Benutzerverwaltungsfunktionalitäten zur Verfügung. Der Konsument des Service Contracts übergibt beim anmelden an den Server ein *User*objekt, über welches der Benutzer identifiziert werden kann. Wird der Benutzer erfolgreich angemeldet, wird ein *Session*objekt an den Konsumenten zurückgegeben. Das Gegenstück zur Methode *Join* ist die Methode *Leave*, die eine *Session* für ungültig erklärt.

Die Methoden *Subscribe* und *Unsubscribe* werden vor dem Aufruf von *Join* bzw. nach dem Aufruf von *Leave* aufgerufen und ermöglichen es der Serverapplikation Callbacks (siehe 3.2.3 auf der nächsten Seite) an den Client zu versenden.

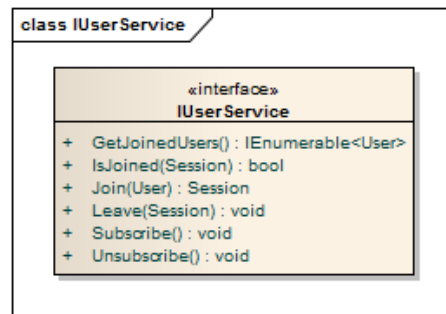


Abbildung 3.3.: Klassendiagramm *IUserService*

Klassendiagramm

IMessagingService

Über die Schnittstelle *IMessagingService* werden Nachrichten in Form eines *Message*objekts übertragen. Der Sender gibt beim Aufruf der Methode *Send* neben der *Message* auch noch seine *Session* mit, damit die Serverapplikation überprüfen kann, ob der Benutzer eine gültige, nicht abgelaufene *Session* besitzt. Die Methoden *NegotiateInitializationVector* und *NegotiateKey* dienen zur Übermittlung des Initialisierungsvektors bzw. Schlüssels für den jeweils mitgelieferten Kryptoalgorithmus vom Typ *CryptoAlgorithm*.

3. Konzept

Methode	Aufgabe
<i>NotifyNegotiateIntializationVector</i>	Teilt dem Client mit, dass für einen bestimmten Kryptoalgorithmus ein neuer Intialisierungsvektor gesetzt wurde.
<i>NotifyNegotiateKey</i>	Teilt dem Client mit, dass für einen bestimmten Kryptoalgorithmus ein neuer Schlüssel gesetzt wurde.
<i>NotifyNewMessage</i>	Teilt dem Client mit, dass eine neue Nachricht an ihn gesendet wurde.
<i>NotifyUserJoined</i>	Teilt dem Client mit, dass ein neuer Benutzer dem Gespräch beigetreten ist.
<i>NotifyUserLeft</i>	Teilt dem Client mit, dass ein Benutzer das Gespräch verlassen hat.

Tabelle 3.1.: Methoden von *IServiceCallback*



Abbildung 3.4.: Klassendiagramm *IMessagingService*

Klassendiagramm

IServiceCallback

Damit über das HTTP Callbacks (der Mechanismus wird detaillierter [REFERENZ] beschrieben) versendet werden können, müssen alle Clientapplikationen, die den Webservice der Serverapplikation verwenden, die Schnittstelle *IServiceCallback* implementieren. Die beiden Schnittstellen *IUserService* und *IMessagingService* werden mit *[ServiceContract(CallbackContract = typeof(IServiceCallback))]* annotiert⁷ so dass beim Abbonnieren des Webservices der generierte Serviceclient gezwungen wird die Schnittstelle *IServiceCallback* zu implementieren.

Die Schnittstelle ist eine zusammengefasste Schnittstelle aller Callbackoperationen der beiden Serviceschnittstellen *IUserService* und *IMessagingService*, die in der Tabelle 3.1 beschrieben sind.

⁷Details zum ServiceContractAttribute sind in der MSDN unter <http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute.aspx> zu finden.

3. Konzept

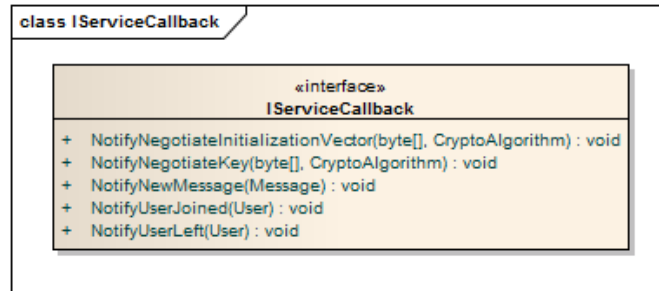


Abbildung 3.5.: Klassendiagramm *IServiceCallback*

Klassendiagramm

3.2.4. Kryptoalgorithmen

Ein Ziel dieses Projekts war es, einzelne Kryptoalgorithmen als eigenständige Assemblies zu implementieren und zur Laufzeit den Clientapplikationen anzuhängen. Aus diesem Grund müssen alle Kryptoalgorithmen gegen die Schnittstelle *ICryptoProvider* implementiert werden. *ICryptoProvider* liefert die grundlegenden Funktionalitäten zur Implementierung von symmetrischen Kryptosystemen, bei welchen beide Teilnehmer den gleichen Schlüssel verwenden. Alle Nutzdaten (verschlüsselt oder unverschlüsselt) werden als Bytearrays (*byte[]*) übergeben, so dass sie einfach und ohne Zusatzaufwand über einen Webservice übergeben werden können. Jedes symmetrische Kryptosystem enthält einen Initialisierungsvektor, der einen Block von Zufallsdaten bezeichnet, sowie einen Schlüssel, der nur dem Sender und Empfänger bekannt ist. Aus diesem Grund enthält *ICryptoProvider* auch keine Methode *GetKey()*, da der Schlüssel nicht öffentlich bekannt sein darf.

Die Methoden *Encrypt(string)* bzw. *Decrypt(byte[])* verschlüsselt bzw. entschlüsselt übergebene Nutzdaten. Im nachfolgenden Klassendiagramm werden zusätzlich zur Schnittstelle *ICryptoProvider* auch noch die drei beispielhaften Implementierungen von symmetrischen Kryptosystemen AES, Rijndael und TripleDES dargestellt. Jedes dieser Kryptosysteme wird in eine eigene Assembly kompiliert und zur Laufzeit der Clientapplikationen via MEF (siehe [REFERENZ]) geladen.

3. Konzept

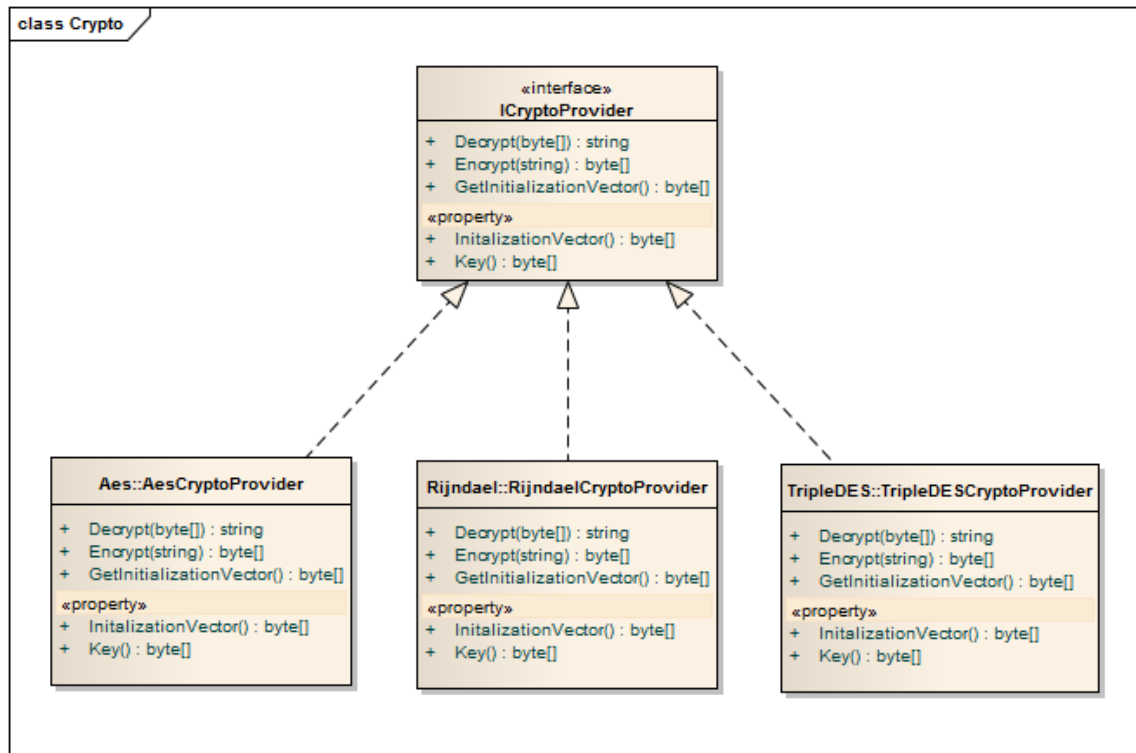


Abbildung 3.6.: Klassendiagramm Kryptoalgorithmen

Klassendiagramm

3.2.5. Server

Die Serverapplikation hosted die beiden Webservices *IMessagingService* (siehe 3.2.3 auf Seite 24) und *IUserService* (siehe 3.2.3 auf Seite 24) in einem gemeinsamen Servicehost *Services*. Daraus ergibt sich eine Verwaltung der angemeldeten Benutzer und deren Sessions sowie das Aufrufen der passenden Callbacks bei Serviceoperationen. Dank des Bindings (siehe [REFERENZ]) kann die Serverapplikation Delegates der Callbackoperationen aller registrierten Clientapplikationen verwalten und dadurch z.B. beim Eintreffen einer neuen Nachricht die Callbackoperation *NotifyNewMessage* (siehe *IServiceCallback* 3.1 auf Seite 25) der registrierten Clients aufrufen.

3. Konzept

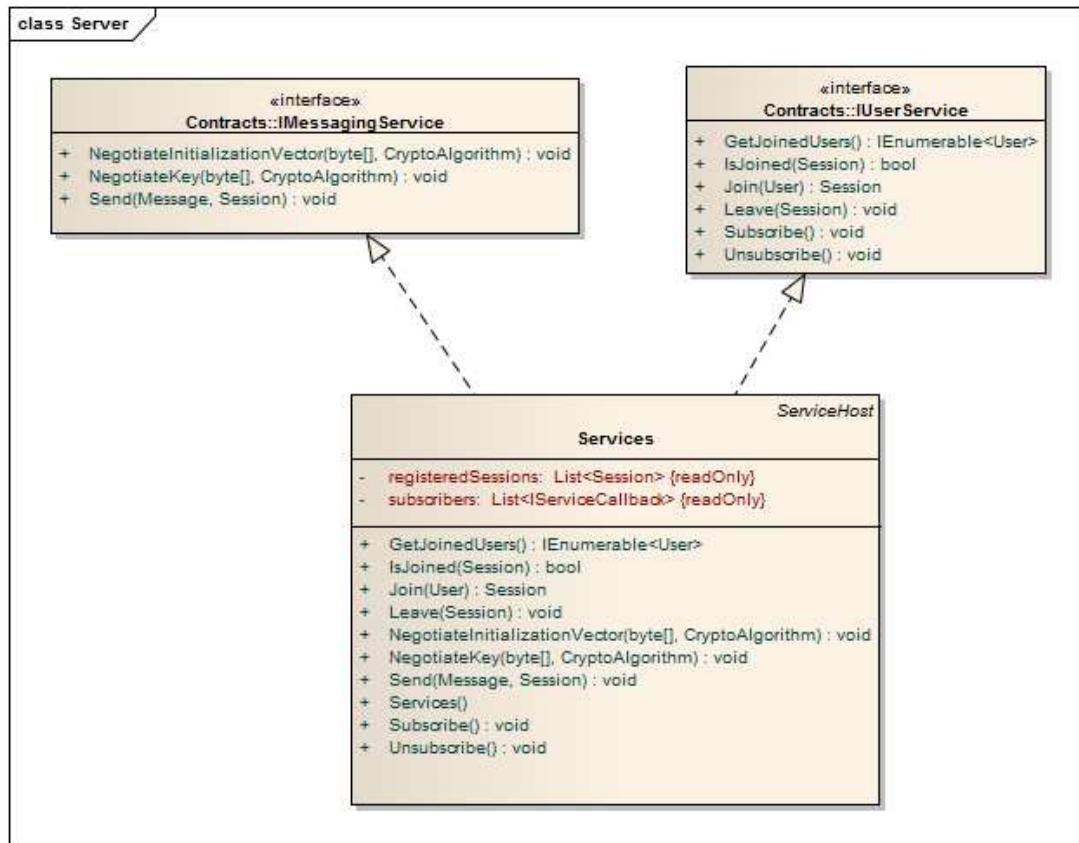


Abbildung 3.7.: Klassendiagramm Server

Klassendiagramm

3.2.6. Client

Für die Clientapplikationen wird kein Konzept erstellt, da diese über eine Service Reference⁸ die Services der Serverapplikation (siehe 3.2.5 auf der vorherigen Seite) anziehen und zusätzlich noch das Interface IServiceCallback (siehe 3.2.3 auf Seite 25) implementieren müssen. Ansonsten ergibt sich die logische Struktur der Clientapplikation aus dem verwendeten GUI-Framework.

3.3. Laufzeitsicht

Die Laufzeitsicht beschreibt, welche Bestandteile des Systems zur Laufzeit existieren und wie sie zusammenwirken (nach [Starke(2011)]). Dabei kommen wichtige Aspekte dese

⁸In der MSDN finden sich unter <http://msdn.microsoft.com/en-us/library/bb907578.aspx> mehr Informationen zu Service References.

3. Konzept

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.

3.3.1. Gespräch beitreten

Möchte ein Benutzer einem Gespräch beitreten so gibt er über das ConnectWindow (siehe [REFERENZ]) seinen gewünschten Benutzernamen ein. Der Loop UserName in use stellt dar, dass der Benutzer solange einen Benutzernamen auswählen muss bis er einen Benutzernamen wählt, der noch nicht dem Gespräch beigetreten ist. Ist der Benutzername frei, so erstellt wird von der Serverapplikation in der SessionsStorage ein neues SessionTicket gelöst, das der Clientapplikation zurückgegeben wird.

3. Konzept

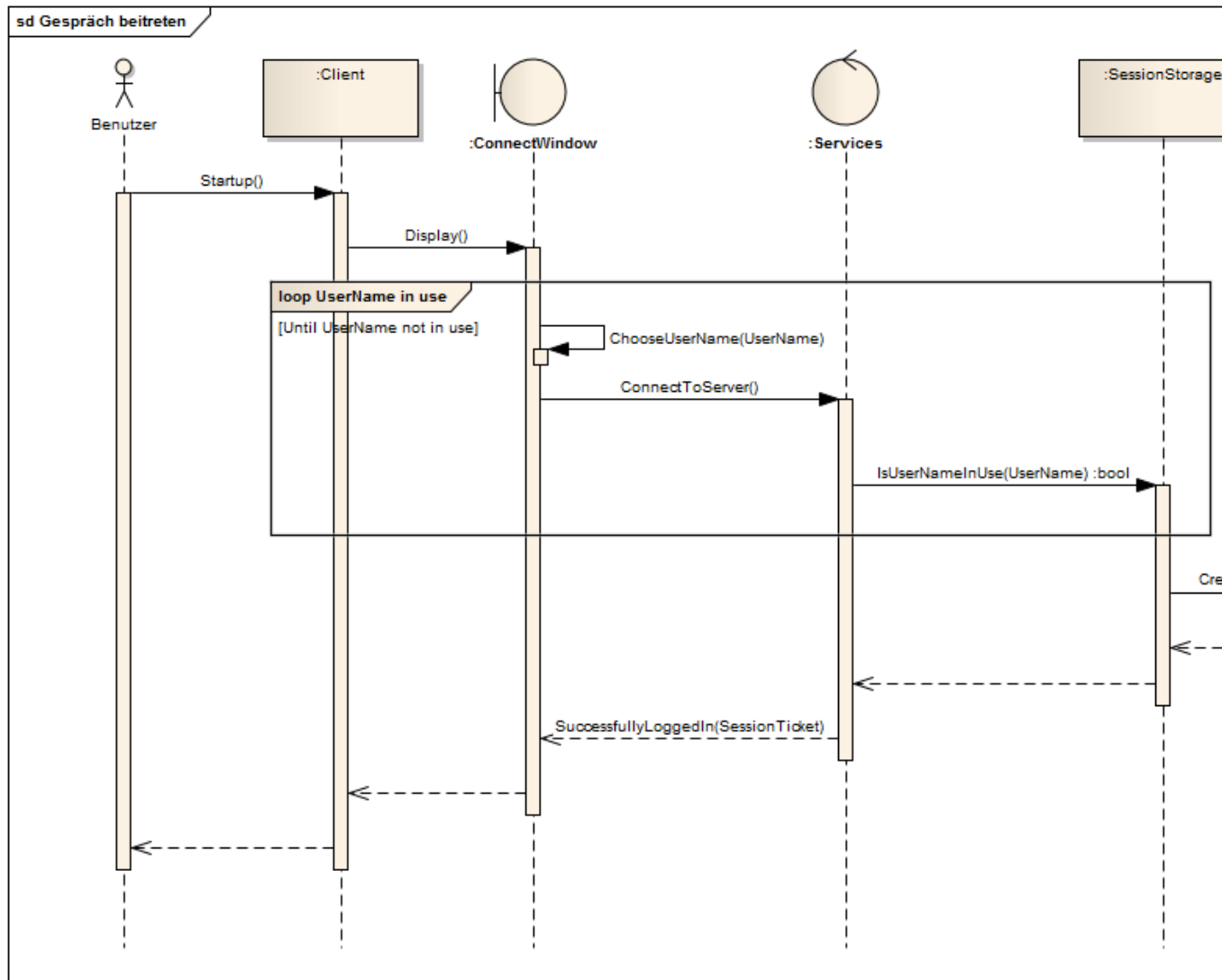


Abbildung 3.8.: Sequenzdiagramm Gespräch beitreten

Sequenzdiagramm

3.3.2. Gespräch verlassen

Möchte der Benutzer das Gespräch verlassen so initiiert er über das MessagingWindow eine Anfrage an die Serverapplikation zum Verlassen des Gesprächs. In der SessionStorage wird dann das SessionTicket des Benutzers entfernt und der Clientapplikation wird mitgeteilt, dass der Benutzer erfolgreich das Gespräch verlassen hat und keine Nachrichten mehr senden oder empfangen kann.

3. Konzept

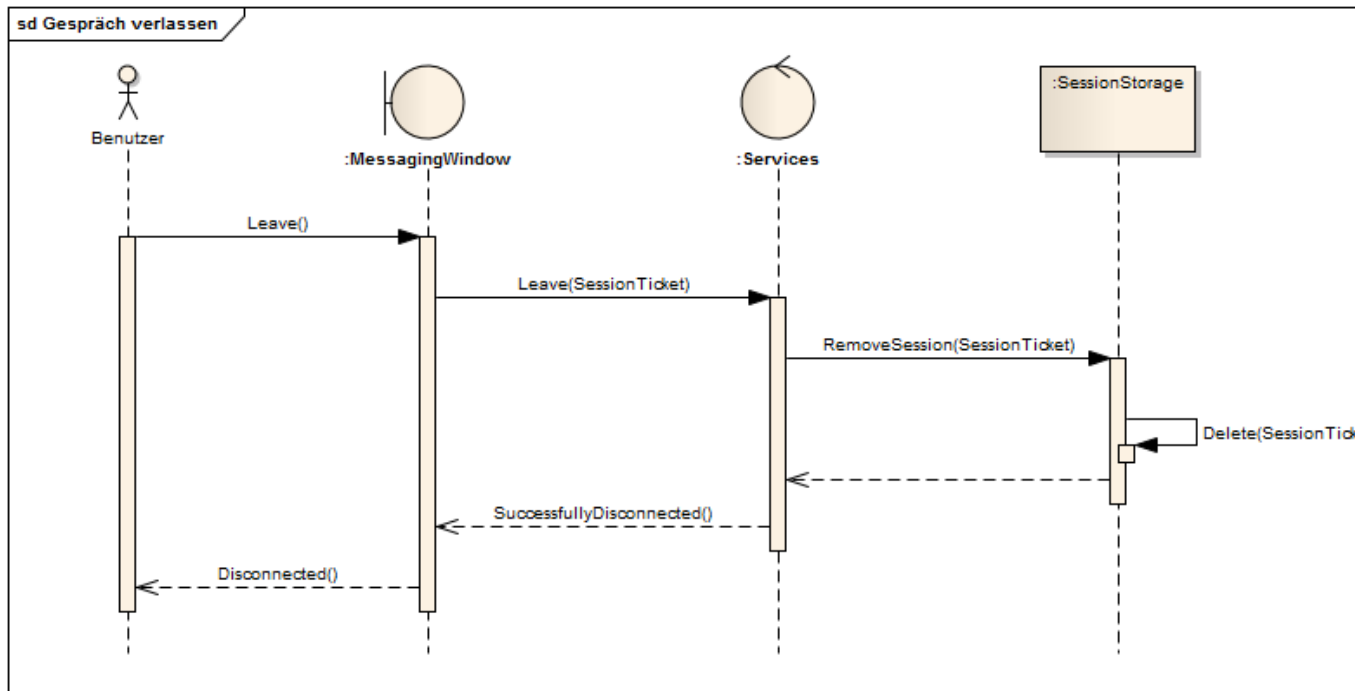


Abbildung 3.9.: Sequenzdiagramm Gespräch verlassen

Sequenzdiagramm

3.3.3. Nachricht senden

Möchte der Benutzer eine Nachricht versenden so muss er, falls noch kein Initialisierungsvektor für den gewählten Kryptoalgorithmus gesetzt ist, zuerst einen Initialisierungsvektor setzen. Dasselbe gilt für den Schlüssel des gewählten Kryptoalgorithmus, ist er innerhalb des gesamten Gesprächs noch nie gesetzt worden, so muss der Schlüssel vom Benutzer gewählt werden.

Sind die beiden Bedingungen (Initialisierungsvektor und Schlüssel gesetzt) erfüllt, wird die Nachricht an den Webservice weitergegeben und falls die gewählten Empfänger ein gültiges SessionTicket besitzen und beim Webservice einen Delegaten als Callback hinterlegt haben wird ihnen die Nachricht übermittelt. Der Benutzer erhält anscheinend eine Bestätigung des Nachrichtenversandes.

3. Konzept

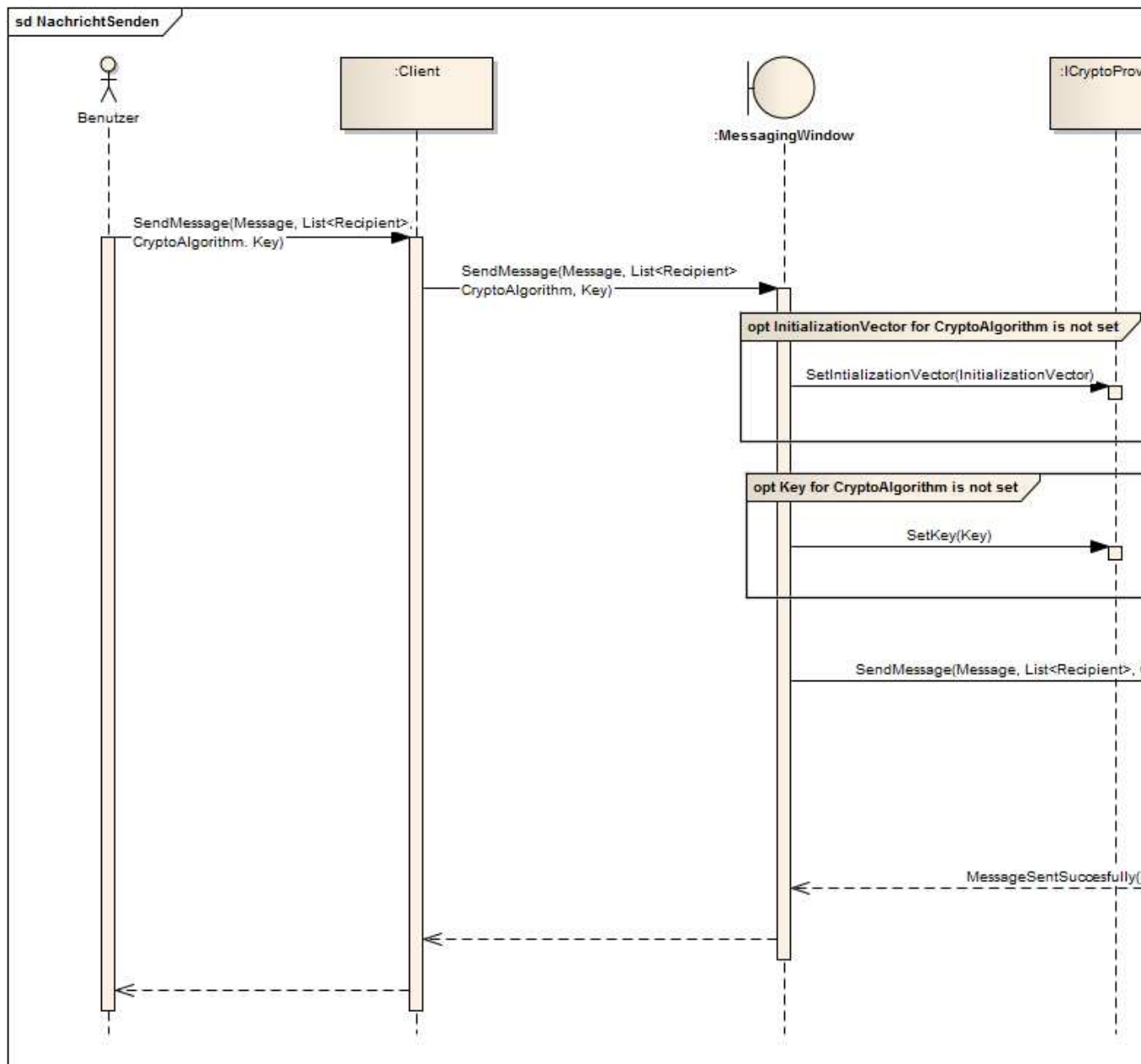


Abbildung 3.10.: Sequenzdiagramm Nachricht senden

Sequenzdiagramm

3.4. Verteilungssicht

Nach [Starke(2011)] beschreibt die Verteilungssicht die Ablaufumgebung des Systems in Form von Hardwarekomponenten (wie Prozessoren, Speicher, Netzwerk, Router und Firewalls) mit 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 [Hruschka and Starke(2012)]).

3.4.1. Verteilungsdiagramm

Die Verteilungssicht in dieser Dokumentation beinhaltet nur ein Verteilungsdiagramm das sehr rudimentär und ohne grosse Details aufgezeichnet wird. Dies, da kein konkretes Verteilungsszenario der Applikation innerhalb des Projekts geplant wurde. Das Projekt beinhaltet die Erarbeitung des Konzepts sowie die konkrete Implementierung der Applikation ohne jedoch konkrete Hardware bzw. vorzugeben auf welchen die einzelnen Applikationen laufen sollen.

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

3. Konzept

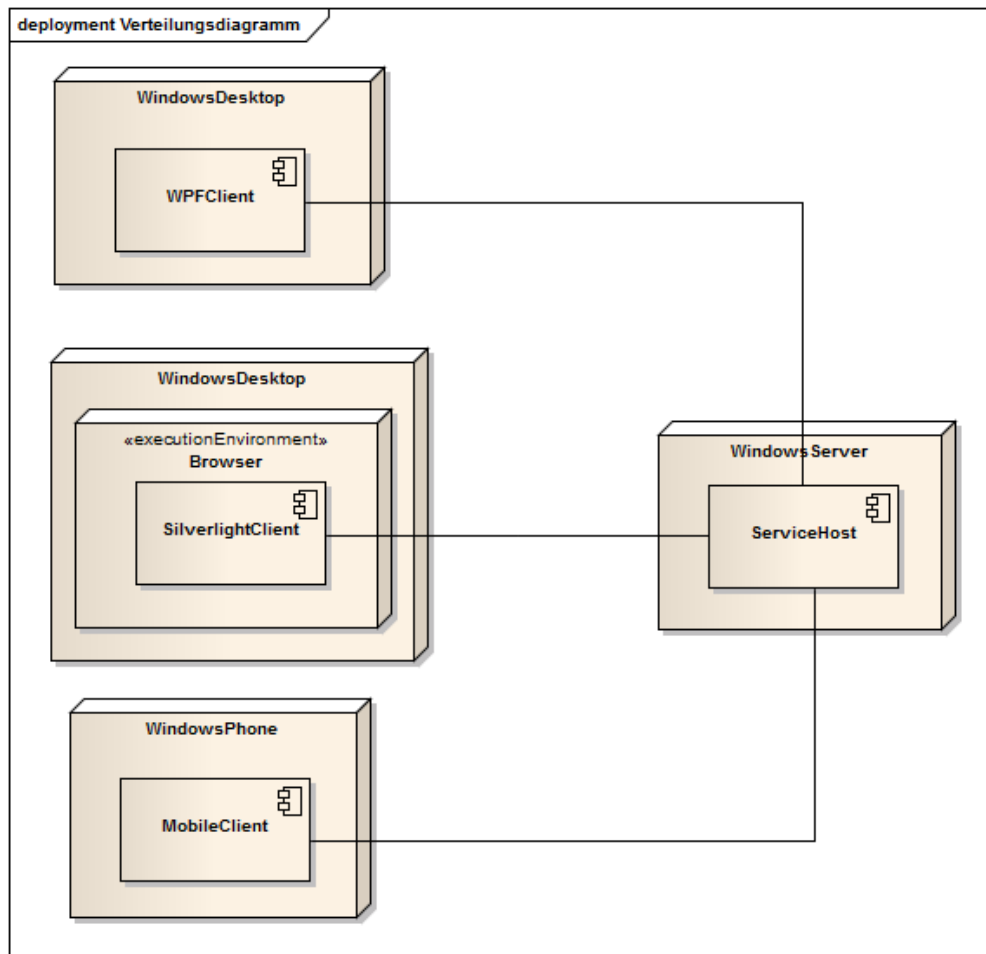


Abbildung 3.11.: Verteilungsdiagramm

Verteilungsdiagramm

4. Implementierung

[KAPITELEINFÜHRUNG]

4.1. Implementierungsmethodik

[WAHL DER IMPLEMENTIERUNGSMETHODIK TDD ERKLÄREN, VORTEILE VON TDD, VERWEIS AUF UNITTESTS]

4.1.1. Testgetriebene Entwicklung

Test Driven Development (kurz TDD, deutsch Testgetriebene Entwicklung) ist eine evolutionäre Entwicklungsmethode, die häufig zusammen mit agilen Methoden Anwendung findet. Dabei werden Tests entsprechend den Anforderungen an eine Funktion erstellt und erst im Nachhinein der funktionale Code implementiert (gemäss [Bullinger(2010)]).

Die Tests werden meist innerhalb eines Unit-Test-Frameworks (siehe 5.1 auf Seite 39) implementiert und laufen gelassen. Zu Beginn werden die Tests fehlschlagen, da noch gar kein Code implementiert worden ist. Ziel des Entwicklers ist es, den Code so lange zu verbessern, bis alle zugehörigen Tests bestanden werden.

Kommen neue Anforderungen oder Funktionen hinzu, werden zuerst neue Tests implementiert und danach der Code erweitert. Danach werden alle Test erneut durchgeführt, solange bis wieder alle Tests bestanden werden.

Zusammengefasst folgt man drei einfachen Schritten bei der Anwendung von TDD (frei nach [Folwer(2012)]), siehe auch 4.1 auf der nächsten Seite):

1. Einen Test schreiben für das nächste Stück von Funktionalität, das der Applikation hinzugefügt werden soll
2. Funktionalen Code schreiben bis der Test erfüllt wird
3. Neuen und alten Code umgestalten um ihn besser zu strukturieren

Durch die Anwendung von TDD wird der bestehende und neue Code fortlaufend optimiert und ist leicht zu ändern. Die enthaltenen Funktionen werden durch die Tests zugleich dokumentiert und Fehler werden früher entdeckt und sind durch Mini-Iterationen leichter lokalisierbar. Besonders in grossen Teams entsteht ein Qualitätsbewusstsein über das ganze Projekt hinweg, nicht nur bei einer kleinen Expertengruppe.

Nachteilig ist anzumerken, dass konsequent sämtlicher Code testdriven erstellt werden muss. Für Entwickler, die noch nie mit TDD in Berührung gekommen sind, ist es schwierig sich vorzustellen, wie etwas getestet werden soll, das noch nicht existiert. Generell

4. Implementierung

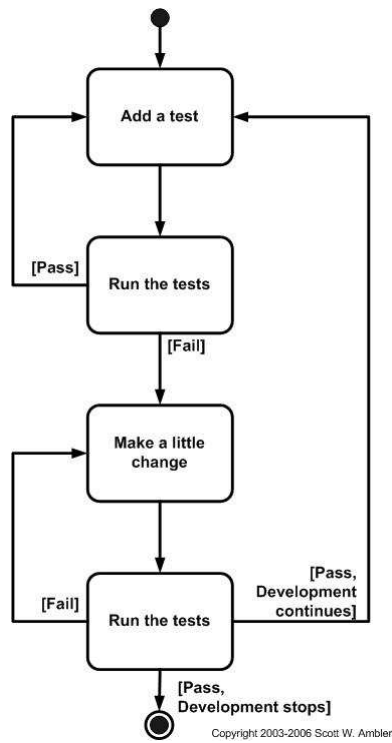


Abbildung 4.1.: Schritte in TDD (aus [Ambler(2011)])

4. Implementierung

funktioniert TDD auch nur, wenn alle Entwickler ein fundiertes Wissen über Testmethoden besitzen. Man muss sich auch bewusst sein, dass TDD keine weiteren Tests wie Integrations- oder Akzeptanztests ersetzt.

4.2. Verwendete Technologien

[DIESES KAPITEL GIBT EINEN ÜBERBLICK ÜBER VERSCHIEDENE TECHNOLOGIEN UND TECHNIKEN, SOLL ZUR ERKLÄRUNG VON BESONDERHEITEN DER ARCHITEKTUR DIENEN]

4.2.1. MEF

[MEF ERLÄUTERN, TECHNOLOGIEN DAHINTER (REFLECTION UND ANNOTATIONS), VORTEILE UND NACHTEILE]

4.2.2. WCF

[WCF ERKLÄREN, WOFÜR BRAUCHT MAN WCF']

4.2.2.1. Bindings

[VERSCHIEDENE BINDINGS AUFLISTEN UND VERGLEICHEN, DETAILS ZU WSHTTPDUALBINDING AUFLISTEN, VORTEILE UND NACHTEILE]

4.2.2.2. Callbacks über NetHttpDualBinding

[DELEGATES ERKLÄREN, MECHANISMSU VON CALLBACKS ÜBER WSHTTPDUALBINDING ERKLÄREN]

4.2.3. WPF

[VORTEILE VON WPF GEGENÜBER WINFORMS, EVT. EIN PAAR EINFACHE ANWENDUNGSBEISPIELE]

4.2.3.1. Silverlight

[SILVERLIGHT ALS SUBSET VON WPF, DAS IM BROWSER LÄUFT]

4.2.4. NuGet

[NUGET ALS PACKAGE MANAGER ERKLÄREN. WIESO PACKAGE MANAGER? VORTEILE/NACHTEILE]

4.3. Kryptosysteme

[AES, TRIPLEDES UND RIJNDAEL ERKLÄREN, SYMMETRISCHE/ASYMMETRISCHE KRYPTOSYSTEME ERKLÄREN, ERKLÄREN WIESO IM MOMENT NUR SYMMETRISCHE KRYPTOSYSTEME UNTERSTÜTZT WERDEN, ERKLÄREN, WIE MAN ASYMMETRISCHE KRYPTOSYSTEME UNTERSTÜTZEN WÜRDEN (WELCHE ÄNDERUNGEN AN ICRYPTOProvider WÄREN NÖTIG)]

4.4. Aufgetretene Probleme

[WSHttpDualBinding UND SILVERLIGHT FUNKTIONIERT NICHT -> KEIN SILVERLIGHT CLIENT IMPLEMENTIERT]

5. Test

[KAPITELEINFÜHRUNG]

5.1. Unit-Tests

[WAS SIND AUTOMATISIERTE UNITTESTS? VORTEILE AUFZEIGEN. VERBINDUNG MIT TDD AUFZEIGEN. MSTEST EINFÜHREN]

5.2. Akzeptanztests

[WAS SIND AKZEPTANZTESTS? AUSFORMULIEREN AUFGRUND DER ANFORDERUNGEN, ÜBERSETZUNG IN CODE]

6. Fazit

[AUFÜHRliches FAZIT ERSTELLEN, PROBLEME BEI DER IMPLMENTIERUNG AUFZEIGEN, ZUKUNFT VON YAEM?]

6.1. Aufwand SOLL/IST Vergleich

[TABELLE SOLL/IST AUFWAND PRO PHASE, AUFZEIGEN WO VIEL MEHR ZEIT ALS GEPLANT VERWENDET WURDE]

6.2. Schlussfolgerungen

[ZU VIEL ZEIT FÜR SILVERLIGHT/HTTPDUALBINDING VERWENDET, GENERELL VIEL MEHR ALS 120H AUFGEWENDET, HERZBLUT IN SEMINARARBEIT GESTECKT]

A. Anhang

Akronyme

GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
MEF	Managed Extensibility Framework
MSDN	Microsoft Developer Network
RUP	Rational Unified Process
SL	Silverlight
TDD	Test Driven Development
UC	Use Case
UI	User Interface
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
WWW	World Wide Web
YAEM	Yet Another Encrypted Messenger
ZHAW	Zürcher Hochschule für Angewandte Wissenschaften
XML	Extensible Markup Language

Nomenclature

Annotation	Eine Annotation bezeichnet ein Sprachelement im Quelltext, das zur Einbindung von Metadaten im Quelltext dient.
Assembly	Übersetzter Quellcode als ausführbares Programm wird in .NET in sogenannten Assemblies zusammengefasst, ähnlich den Jar-Dateien in Java. Die Dateiendungen dieser Assemblies sind .exe oder .dll, zusätzlich zum ausführbaren Programmcode enthalten sie auch alle im Manifest notwendigen Metadaten.
Bug	Ein Programm- oder Softwarefehler wird als Bug bezeichnet und beschreibt im Allgemeinen ein fehlerhaftes Verhalten von Computerprogrammen.
Callback	Ein Callback (zu deutsch Rückruffunktion) bezeichnet eine Funktion, der als Parameter eine andere Funktion übergeben wird, die unter gewissen Bedingungen aufgerufen wird.
Client	Ein Client (oder Clientapplikation) ist eine Softwareanwendung die im Gegensatz zu einer Serverapplikation auf dem Rechner des Benutzers ausgeführt wird.
Delegate	Ein Delegate (dt. Delegat) ist ein Methodenzeiger und wird verwendet um auf eine Methode einer Klasse oder eines Objekts zu verweisen.
Dialog	Als Dialog (oder Dialogfenster) bezeichnet man in der Softwareentwicklung eine grafische Benutzerschnittstelle zur Mensch-Maschine-Interaktion zwischen Computerprogramm und Benutzer.
Geheimtext	Der Geheimtext ist der Text, der durch die Verschlüsselung mittels eines kryptografischen Verfahrens unlesbar gemacht wurde.
Git	Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien, die ursprünglich für die Quellcode-Verwaltung des Kernels Linux entwickelt wurde.

A. Anhang

Globally Unique Identifier	Ein Globally Unique Identifier (auch GUID genannt) ist eine eindeutige, 128 Bit lange Zahl, die zur Identifizierung von Objekten in verteilten Systemen dienen.
HTTP	Das Hypertext Transfer Protocol (kurz HTTP) ist ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es ist das meisteingesetzte Protokoll zur Übertragung von Webseiten im World Wide Web (WWW).
Initialisierungsvektor	Ein Initialisierungsvektor (kurz IV) bezeichnet in der Kryptografie ein Block von Zufallszahlen.
Kryptoalgorithmus	Ein Kryptoalgorithmus ist im Kontext von YAEM die konkrete Implementierung des Interfaces ICryptoProvider und bietet die Möglichkeit beliebige Nachrichten zu verschlüsseln beziehungsweise zu entschlüsseln.
Managed Extensibility Framework	Das Managed Extensibility Framework (kurz MEF) ist ein Kompositionframework zum Erstellen einer Plugininfrastruktur innerhalb einer Applikation. Es erhöht die Flexibilität, Erweiterbarkeit und Testbarkeit von grossen Applikationen.
Mockup	Ein Mockup in der Softwareentwicklung bezeichnet einen rudimentären Wegwerfprototypen der Benutzerschnittstelle einer zu erstellenden Software. Mockups werden insbesondere in frühen Entwicklungsphasen eingesetzt, um Anforderungen an die Benutzeroberfläche in Zusammenarbeit mit Auftraggeber und Anwendern besser ermitteln zu können. Es handelt sich meist um ein reines Grundgerüst der Bedienelemente ohne weitere Funktionalität.
Repository	Ein Repository ist ein Verzeichnis zur Speicherung einer von digitalen Objekten. In diesem Kontext ist ein Repository eine Datenbank zur Speicherung und zum Wiederfinden von Objekten.
RUP	Der Rational Unified Process ist ein kommerzielles Vorgehensmodell zur Softwareentwicklung von IBM.
Server	Ein Server (auch Serverapplikation) ist eine serverseitige Anwendung die auf einem zentralen Computer (Server) ausgeführt wird.

A. Anhang

Service Contract	Ein Service Contract bezeichnet eine Schnittstelle oder Klasse die zur Kommunikation für verteilte Systeme genutzt werden können.
Service Reference	Über eine Service Reference erstellt das .NET Framework einen Clientproxy basierend auf einem Webservice und stellt die so verfügbaren Methoden in generiertem Quellcode der Applikation zur Verfügung.
Silverlight	Silverlight ist eine Erweiterung für Webbrowser, welche die Ausführung von Rich Internet Applications erlaubt. Silverlight ist eine abgespeckte Version von WPF und wird für auch als Framework für Windows Phone 7 verwendet.
Use Case	Ein Use Case (deutsch Anwendungsfall) bündelt alle möglichen Szenarien, die eintreten können, wenn ein Akteur versucht, mit Hilfe des betrachteten Systems ein bestimmtes fachliches Ziel zu erreichen. Er beschreibt, was inhaltlich beim Versuch der Zielerreichung passieren kann, und abstrahiert von konkreten technischen Lösungen. Das Ergebnis des Anwendungsfalls kann ein Erfolg oder Fehlschlag/Abbruch sein.
V-Modell	Das V-Modell ist ein Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in Phasen organisiert wird. Neben diesen Entwicklungsphasen definiert das V-Modell auch das Vorgehen zur Qualitätssicherung (Testen) phasenweise.
Webservice	Ein Webservice ist eine Softwareapplikation, auf den über eine URL eindeutig identifiziert ist und Daten als XML-Artefakt zurückgibt. Er wird über internetbasierte Protokolle angesprochen.
WPF	Windows Presentation Foundation (kurz WPF) ist ein Grafik-Framework das zusammen mit dem .NET Framework mitgeliefert wird und zur Darstellung von UI-Elementen dient.

Literaturverzeichnis

- [IEE(1990)] Ieee standard glossary of software engineering terminology, 1990.
- [Ambler(2011)] Scott W. Ambler. Introduction to test driven develop. 2011. URL <http://www.agiledata.org/essays/tdd.html>.
- [Bernd Oestereich(2006)] Markus Klink Guido Zockoll Bernd Oestereich, Claudia Schröder. *OEP - oose Engineering Process: Vorgehensleitfaden für agile Softwareprojekte*. dpunkt, 2006.
- [Bullinger(2010)] Robert Bullinger. Test driven development, 03 2010. URL <http://robert.bullinger.over-blog.com/article-test-driven-development-47200502.html>.
- [Elmer(2005)] Franz-Josef Elmer. Software engineering, methodologien. 2005. URL <http://informatik.unibas.ch/lehre/ws05/cs203/softeng2.pdf>.
- [Folwer(2012)] Martin Folwer. Test driven de, 2012. URL <http://martinfowler.com/bliki/TestDrivenDevelopment.html>.
- [Hruschka and Starke(2012)] Peter Hruschka and Gernot Starke. arc42 - ressourcen für software-architekten, 2012. URL <http://www.arc42.de>.
- [Pohl and Rupp(2011)] Klaus Pohl and Chris Rupp. *Basiswissen Requirements Engineering*. dpunkt.verlag, 2011.
- [Starke(2011)] Gernot Starke. *Effektive Software-Architekturen*. Carl Hanser Verlag, 2011.
- [Stern(2010)] Dr. Olaf Stern. Reglement semesterarbeit. 2010. URL https://ebs.zhaw.ch/files/ebs_files/Reglemente/Bachelor/Semesterarbeit/a_Reglement-Sem
- [VO(2007)] Trung Hung VO. Software development process, 07 2007. URL <http://cnx.org/content/m14619/1.2/>.