

Yet Another Encrypted Messenger

Florian Amstutz <florian@amstutz.nu>

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	7
1.4.1	Phasenplanung	7
1.4.2	Meilensteine	8
2	Anforderungen	9
2.1	Was sind Anforderungen?	9
2.1.1	Arten von Anforderungen	9
2.2	Systemkontext	10
2.3	Use-Case-Spezifikationen	11
2.3.1	Gespräch beitreten	12
2.3.2	Gespräch verlassen	13
2.3.3	Nachricht senden	14
2.3.4	Nachricht empfangen	16
2.4	Mockups	18
2.4.1	Dialog Gespräch beitreten	19
2.4.2	Gesprächsdialog	19
3	Konzept und Architektur	21
3.1	Toolchain	21
3.2	Bausteinsicht	21
3.2.1	Komponentendiagramm	22
3.2.2	Domänenmodell	22
3.2.3	Service Contracts	24
3.2.3.1	IUserService	24
3.2.3.2	IMessagingService	24
3.2.3.3	IServiceCallback	25
3.2.4	Kryptosysteme	26
3.2.5	Server	26
3.2.6	Client	28
3.3	Laufzeitsicht	28
3.3.1	Gespräch beitreten	28
3.3.2	Gespräch verlassen	28

Inhaltsverzeichnis

3.3.3	Nachricht senden	28
3.4	Verteilungssicht	30
3.4.1	Verteilungsdiagramm	31
4	Implementierung	32
4.1	Entwicklungsmethoden	32
4.1.1	Testgetriebene Entwicklung	33
4.2	Verwendete Technologien	33
4.2.1	Managed Extensibility Framework	35
4.2.2	Windows Communication Foundation	36
4.2.2.1	Bindings	36
4.2.2.2	Callbacks über WSDualHttpBinding	38
4.2.3	Windows Presentation Foundation	38
4.2.3.1	Model View ViewModel	39
4.2.3.2	Silverlight	40
4.2.4	Mock-Objekte	41
4.3	Kryptosysteme	41
4.3.1	Symmetrische Kryptosysteme	42
4.3.1.1	Advanced Encryption Standard	42
4.3.1.2	Rijndael	42
4.3.1.3	Triple-DES	43
4.3.2	Unterstützung asymmetrischer Kryptosysteme	43
4.4	Komponenten im Detail	43
4.4.1	YAEM.Domain	44
4.4.2	YAEM.Contracts	44
4.4.3	YAEM.Server	45
4.4.4	YAEM.Crypto	45
4.4.4.1	YAEM.Crypto.Aes	46
4.4.4.2	YAEM.Crypto.Rijndael	46
4.4.4.3	YAEM.Crypto.TripleDES	46
4.4.5	YAEM.TestClient	46
4.4.6	YAEM.DesktopClient	47
4.4.7	YAEM.MobileClient	49
4.4.8	Aufgetretene Probleme	49
5	Test	50
5.1	Unit-Tests	50
5.1.1	Testabdeckung	50
5.2	Akzeptanztests	51
6	Fazit	53
6.1	Aufwand Seminararbeit	53
6.2	Schlussfolgerungen	54

1 Einführung

Als einführendes Kapitel dieses Dokuments wird die Semesterarbeit als Projekt kurz vorgestellt und es werden die Rahmenbedingungen der Arbeit 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 festgelegt.

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 und 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 versenden 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 Fokus der Arbeit auf der methodischen Vorgehensweise der Softwareentwicklung gelegt und weniger auf der Implementierung der kryptografischen Algorithmen.

1.2 Über die Semesterarbeit

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

Der Aufwand für die Fertigstellung der Semestarbeit 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 und vergleichsweise starre Modelle wie RUP bis hin zu klassischen, linearen Vorgehen wie dem Wasserfallmodell oder dem V-Modell.

1 Einführung

Nach [Starke(2011)] ist die Auswahl des Entwicklungsprozesses eine der kritischsten Entscheidungen, die man bei einem Softwareprojekt treffen muss. Häufig besitzen Unternehmen 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 [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 meist in fünf verschiedene Phasen ein (siehe Abbildung 1.1). Dabei kann jeweils erst mit der nächsten Phase begonnen werden wenn die Lieferergebnisse und die Ergebnisdokumentation der vorhergehenden Phase fertiggestellt und abgenommen worden sind. Das Wasserfallmodell wurde ausgewählt, da dieses Projekt gut linear abgearbeitet werden kann und da der Betreuer als einziger externer Stakeholder des Projekts zu festdefinierten Zeitpunkten (namtlich dem Kick-Off und dem Design-Review) Einfluss auf das Projekt ausüben kann und danach keine andere Möglichkeit besitzt, im Projektverlauf zu intervenieren. Die grössten Nachteile des Wasserfallmodells sind nach [Elmer(2005)] Abgrenzungsprobleme zwischen den einzelnen Phasen sowie die Schwierigkeit des Abschlusses einzelner Phasen da diese nur mit viel (zusätzlichem) Aufwand isoliert von anderen Phasen abgeschlossen werden können. Dadurch dass der Betreuer nur in der Anforderungs- und Konzeptphase Einfluss auf das Projekt nehmen kann und der Student alleinig den Abschluss der Phasen steuert und verifiziert sowie den Ablauf der Phasen innerhalb des Projekts steuert, können diese Nachteile umgangen werden.

Zu Beginn des Wasserfallmodells steht das Sammeln und Dokumentieren der Anforderungen (Requirements) im Vordergrund. Wenn die Anforderungen umfänglich und in hohem Detaillierungsgrad niedergeschrieben sind, werden diese vom Auftraggeber abgenommen und das Projekt geht in die Phase Design über. Die zu entwickelnde Software wird auf verschiedenen Ebenen von Softwarearchitekten designed und eine Blaupause wird erstellt, nach welcher sich die Softwareentwickler in der Implementationsphase zu halten haben. Das Design sollte einen Plan beinhalten, welcher die Implementierung der Anforderungen aufzeigt.

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

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

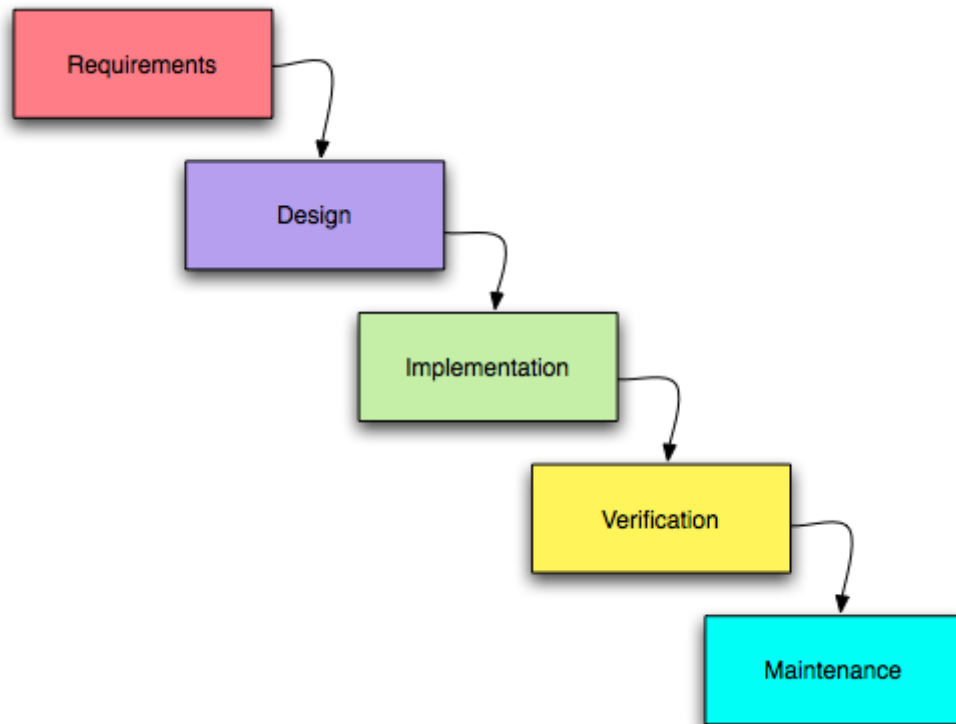


Abbildung 1.1: Wasserfallprozess nach [VO(2007)]

Phase	Kapitelüberschrift	Siehe Seite
Requirements	Anforderungen	9
Design	Konzept und Architektur	21
Implementation	Implementierung	32
Verification	Test	50

Tabelle 1.1: Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument

1 Einführung

Phase	Start	Ende	Endet in Meilenstein
Themenevaluation	15.10.2011	04.11.2011	M1, M2
Erfassen der Anforderungen	04.11.2011	01.12.2011	
Erarbeiten des Konzepts	01.12.2011	13.03.2012	M3
Implementierungs des Konzepts	13.03.2012	01.05.2012	
Überprüfen und Test des Konzepts	01.05.2012	23.05.2012	M4
Erarbeiten der Präsentation	23.05.2012	06.06.2012	M5

Tabelle 1.2: Phasenplan

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 bei Projektstart in einzelne Phasen unterteilt, welche auf die vorgängig bekannten Termine als Meilensteine enden.

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

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

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

1 Einführung

Bezeichner	Meilenstein	Datum
M1	Kick-Off	04.11.2011
M2	Freigabe der Arbeit	06.11.2011
M3	Design-Review	13.03.2012
M4	Abgabe der Arbeit	23.05.2012
M5	Schlusspräsentation	06.06.2012

Tabelle 1.3: Meilensteine

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 Phasenenden überein.

Die Meilensteine des Projekts sind in der Tabelle 1.3 ersichtlich.

2 Anforderungen

In der Phasenplanung (siehe Kapitel 1.4.1 auf Seite 7) wurde festgelegt, dass in der ersten Projektphase die Anforderungen erhoben werden. Dazu wird zu Beginn dieses Kapitels der Begriff Anforderung definiert und es wird auf verschiedene Arten von Anforderungen näher eingegangen. Anschliessend wird der Systemkontext beschrieben und eingegrenzt sowie die konkreten Anforderungen an YAEM als Use-Cases modelliert und spezifiziert. Der Abschluss dieses Kapitels wie auch der Anforderungsphase selbst bildet das Entwerfen der für den Benutzer sichtbaren Dialogfenster.

2.1 Was sind Anforderungen?

Die erste Phase des Wasserfallmodells beschäftigt sich mit den Anforderungen an das zu entwickelnde Softwareprodukt. Damit das Entwicklungsprodukt zum Erfolg geführt werden kann, muss zunächst bekannt sein, was die Anforderungen an das System sind und diese müssen geeignet dokumentiert sein. Nach [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 dieser Semesterarbeit zu entwickelnde System werden in Use-Case-Diagrammen modellhaft dargestellt und als Use-Case-Spezifikationen ausformuliert. Auf eine natürlichsprachige Dokumentation der Anforderungen wird verzichtet, da die modellierten Anforderungen innerhalb der Use-Case-Diagramme verständlich genug sind und zu den Use-Case-Diagrammen noch detaillierte Use-Case-Spezifikationen vorhanden sind, welche die gesamte Anforderung abdecken.

2.1.1 Arten von Anforderungen

Nach [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.
- 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.

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

Rahmenbedingungen werden nicht implizit festgehalten sondern ergeben sich einerseits aus [Stern(2010)] sowohl der technischen Umgebung des Systems. Technische Rahmenbedingungen die das Projekt beeinflussen werden im Konzept beziehungsweise in der Implementierung berücksichtigt und entsprechend hervorgehoben.

2.2 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 [Pohl and Rupp(2011)]).

Der Ursprung der Anforderungen des Systems liegt im Systemkontext des geplanten Systems. Aus diesem Grund wird der Systemkontext vor Erhebung und Dokumentierung

¹ auch Rahmenbedingungen genannt

2 Anforderungen

der Anforderungen festgelegt. Der Systemkontext des Systems YAEM ist in Abbildung 2.1 als Modell 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 ihrer selbst an das System vor.

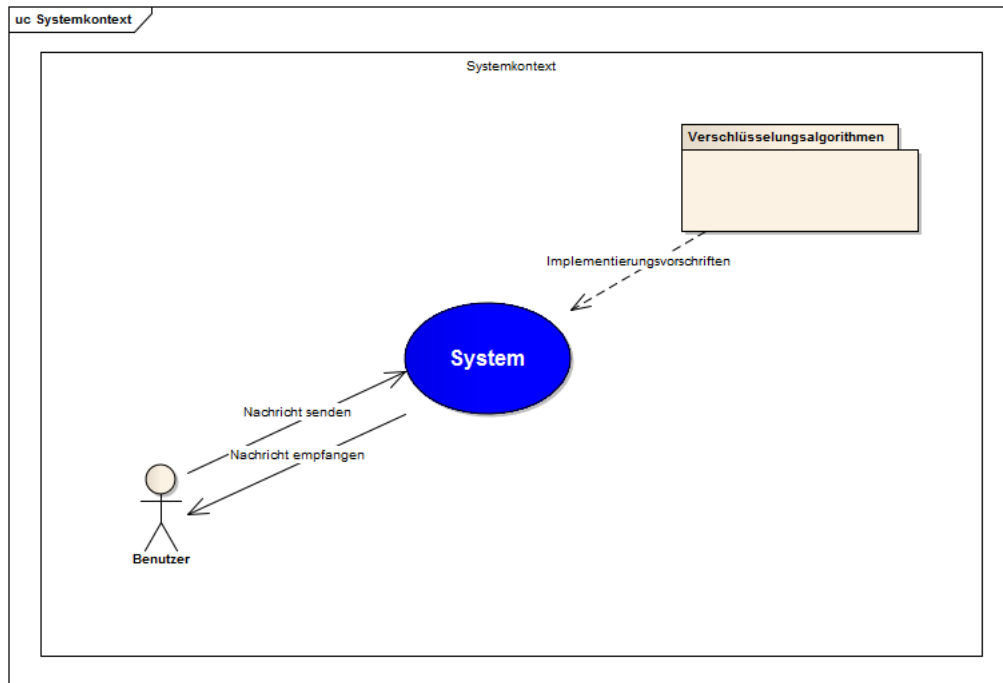


Abbildung 2.1: Systemkontext

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 beziehungsweise zu Aspekten in der Umgebung des Systems. Abgesehen vom Namen des Use-Cases und dessen Beziehungen dokumentieren Use-Case-Diagramme keinerlei weitere Informationen über die einzelnen Use-Cases, wie z.B. die Systematik der Interaktion eines Use-Case mit Akteuren in der Umgebung. Diese Informationen werden unter Verwendung einer geeigneten Schablone zusätzlich zum Use-Case-Diagramm textuell als Use-Case-Spezifikation festgehalten.

Alle funktionalen Anforderungen (siehe Kapitel 2.1.1 auf Seite 9) werden als Use-

2 Anforderungen

Cases modelliert und spezifiziert². Als Quellen der Anforderungen dienen der Betreuer, das Reglement der ZHAW betreffend der Semesterarbeit sowie der Student in der Rolle des Benutzers des Systems. Zusätzlich zu den Use-Cases und den dazugehörigen Use-Case-Spezifikationen wird vorgängig in Prosatext der Anwendungsfall beschrieben. Aus Gründen der Übersichtlichkeit und der limitierten Gesamtfunktionalität des Systems stellen diese Use-Cases die primären Anforderungen an das zu entwickelnde Softwaresystem dar.

Jeder Use-Case wird im Rahmen der Verifizierungsphase (siehe Kapitel 5 auf Seite 50) als Akzeptanztest einzeln getestet.

2.3.1 Gespräch beitreten

Ein Benutzer möchte verschlüsselte Nachrichten an andere Gesprächsteilnehmer versenden und startet dazu 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.

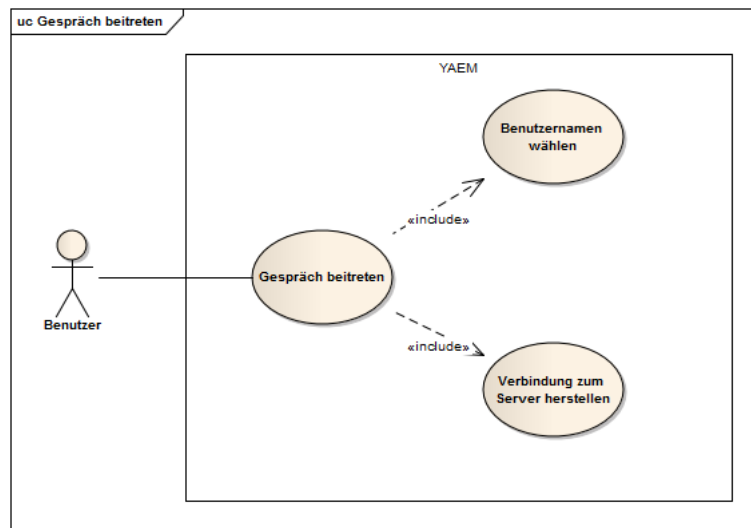


Abbildung 2.2: Use-Case Gespräch beitreten

²Die verwendete Schablone stammt aus [Pohl and Rupp(2011)] 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	Gespräch beitreten
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Hoch
Beschreibung	Der Benutzer tritt einem Gespräch bei.
Auslösendes Ereignis	Der 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	Ein gültiges Session-Ticket wird erstellt.
Hauptszenario	1. Der Benutzer wählt einen Benutzernamen. 2. Der Benutzer stellt eine Verbindung zum Server her. 3. Der Server erstellt ein 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 einem Gespräch beigetreten und möchte dieses verlassen. Er schliesst die Applikation und meldet sich dabei beim Server vom Gespräch ab. Andere Teilnehmer des Gesprächs können ihm nun keine Nachrichten mehr senden.

2 Anforderungen

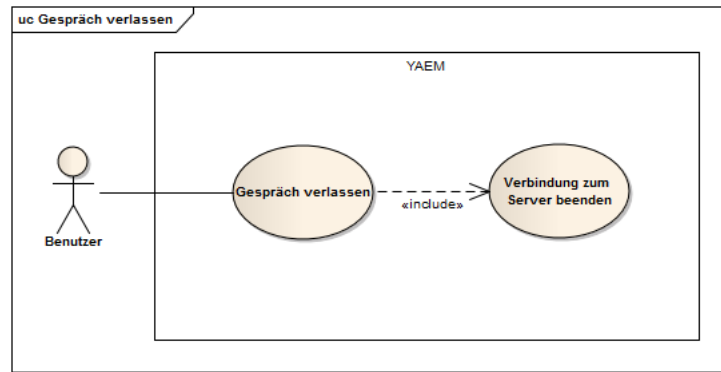


Abbildung 2.3: Use-Case Gespräch verlassen

Abschnitt	Inhalt
Bezeichner	UC2
Name	Gespräch verlassen
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: niedrig
Kritikalität	Hoch
Beschreibung	Der Benutzer verlässt das Gespräch.
Auslösendes Ereignis	Der Benutzer möchte das Gespräch verlassen.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist einem Gespräch beigetreten.
Nachbedingung	Der Benutzer kann erneut einem Gespräch beitreten.
Ergebnis	Das Session-Ticket des Benutzers ist ungültig.
Hauptszenario	1. Der Benutzer verlässt das Gespräch. 2. Der Server erklärt das Session-Ticket des Benutzers für ungültig.
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 (hier in der Rolle des Senders) möchte einem oder mehreren Teilnehmern des Gesprächs (Empfänger) eine Nachricht senden. Er kann dabei wählen, ob er diese verschlüsselt oder unverschlüsselt versenden möchte. Sendet der Benutzer die Nachricht verschlüsselt, so wird zuerst der Initialisierungsvektor festgelegt und der Benutzer wird aufgefordert, einen Schlüssel auszuwählen. Danach wird die Nachricht an den oder die Empfänger übermittelt und startet den Anwendungsfall Nachricht empfangen (siehe Kapitel 2.3.4 auf Seite 16).

2 Anforderungen

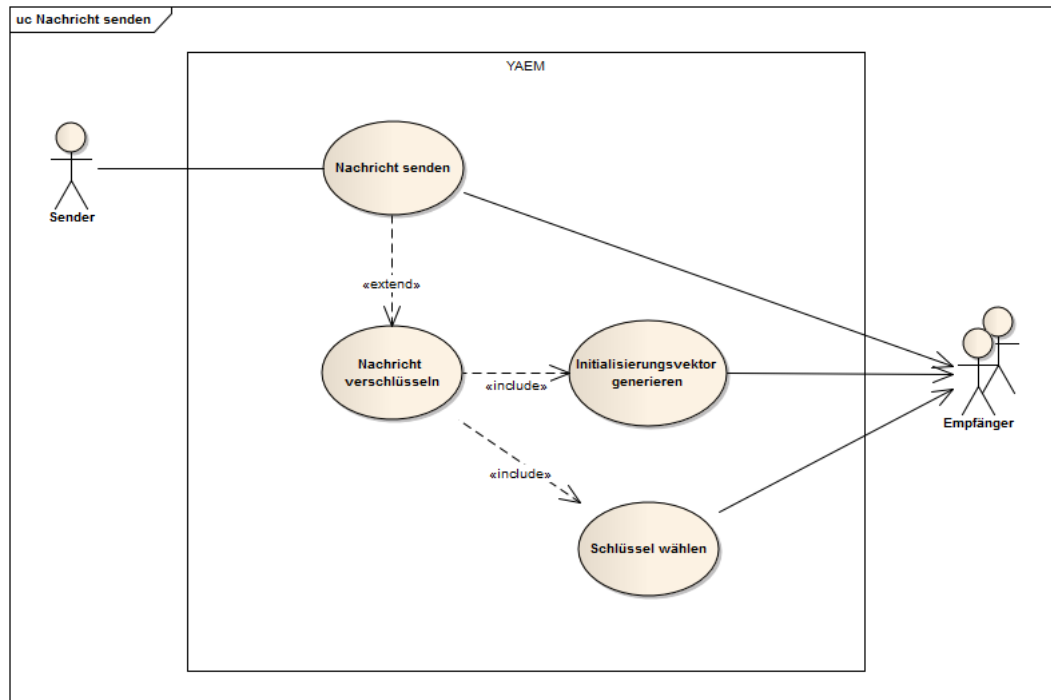


Abbildung 2.4: Use-Case Nachricht senden

2 Anforderungen

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	Der Benutzer möchte eine Nachricht senden.
Akteure	Benutzer (als Empfänger und Sender)
Vorbedingung	Der Benutzer ist dem Gespräch beigetreten und besitzt ein 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 versendende Nachricht. 2. Der Benutzer wählt die Implementierung eines Kryptosystems aus. 3. Der Benutzer generiert einen Initialisierungsvektor. 4. Der Initialisierungsvektor wird an alle Empfänger gesendet. 5. Der Benutzer wählt einen Schlüssel. 6. 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 generiert 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 von einem Benutzer des Systems ausgelöst, sondern vom System selbst. Sobald eine Nachricht, die an den Benutzer gerichtet ist, eintrifft, wird der Anwendungsfall gestartet. Ist die Nachricht verschlüsselt, versucht das System mit vorhandenem Initialisierungsvektor und Schlüssel die Nachricht zu entschlüsseln und dem Benutzer darzustellen. Ist die Nachricht unverschlüsselt, so wird diese dem Benutzer

2 Anforderungen

direkt angezeigt.

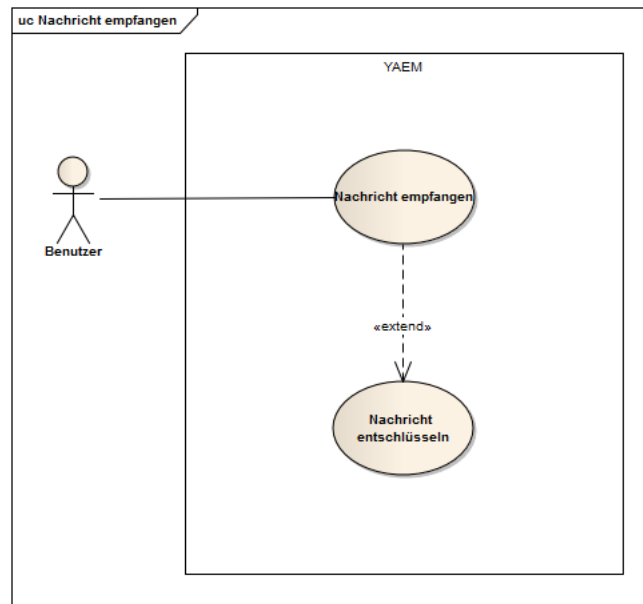


Abbildung 2.5: Use-Case Nachricht empfangen

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, die an den Benutzer gerichtet ist.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist dem Gespräch beigetreten und besitzt ein gültiges Session-Ticket. Ein Teilnehmer des Gesprächs versendet eine Nachricht, die an den Benutzer gerichtet ist.
Nachbedingung	Der Benutzer kann Nachrichten versenden und Nachrichten anderer Gesprächsteilnehmer empfangen.
Ergebnis	Die Nachricht wird dem Benutzer angezeigt.
Hauptszenario	1. Der Benutzer empfängt eine Nachricht. 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 dem Benutzer direkt angezeigt.
Ausnahmeszenarien	Ist kein Initialisierungsvektor, Schlüssel oder Implementierung des verwendeten Kryptosystems vorhanden, so wird der unlesbare Geheimtext angezeigt.

Tabelle 2.4: Use-Case-Spezifikation Nachricht empfangen

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 Kapitel 4 auf Seite 32) ein Grundgerüst für die einzelnen Dialogfenster. Je nach Kundenzielgruppe und Wichtigkeit der Mensch-Maschine-Schnittstelle wird mehr oder weniger Aufwand in die Entwicklung der Mockups investiert. Häufig werden diese mit zusammen mit Psychologen entwickelt und enthalten sehr wenig Interpretationsspielraum für den Softwareentwickler beim späteren Umsetzen der Mockups in Programmcode.

2.4.1 Dialog Gespräch beitreten

Startet der Benutzer die Applikation, so wird er gebeten, einen Benutzernamen zu wählen, unter welchem er im Gespräch Nachrichten versenden kann (Mockup siehe Abbildung 2.6). Dabei wird der Benutzername auf eine Länge von 255 Zeichen beschränkt und er darf keine Sonderzeichen enthalten. Nach einem Klick auf den Button “Connect” werden diese Bedingungen überprüft und gegebenenfalls wird eine Verbindung zum Server hergestellt. Bei erfolgreichem Verbindungsaufbau wird der Dialog geschlossen und dem Benutzer wird der Gesprächsdialog (siehe Kapitel 2.4.2) angezeigt.

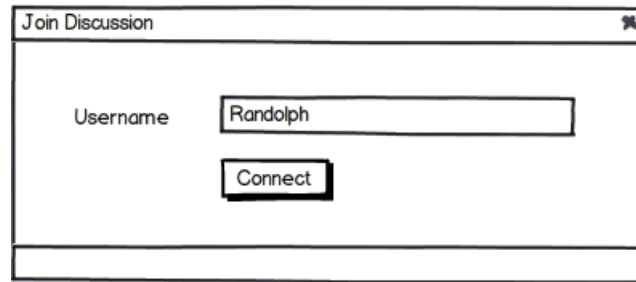


Abbildung 2.6: Mockup Dialog Gespräch beitreten

2.4.2 Gesprächsdialog

Ist der Benutzer einem Gespräch beigetreten so verwendet er den Gesprächsdialog zum Senden und Empfangen von Nachrichten (Mockup siehe Abbildung 2.7). Das Gesprächsprotokoll stellt einen zeitlich geordneten Ablauf aller gesendeten und empfangenen Nachrichten dar. Zusätzlich werden Aktionen wie ein Initialisierungsvektoraustausch oder das Setzen des Schlüssels eines Verschlüsselungsalgorithmus im Gesprächsprotokoll angezeigt.

Auf der rechten Seite des Dialogfensters 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 Kryptosysteme angezeigt, zusätzlich zur Voreinstellung “<None>”, welche die Nachricht unverschlüsselt versendet. Ist kein Kryptosystem installiert, so kann der Benutzer keine Auswahl aus der Dropdownliste treffen.

Wählt der Benutzer ein Kryptosystem aus, für welches noch kein Initialisierungsvektor vorhanden ist, so wird ein neuer Initialisierungsvektor generiert und den anderen Gesprächsteilnehmern zugesandt. Gleichzeitig wählt der Benutzer einen Schlüssel aus. Wird auf den Button “Send” geklickt, 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

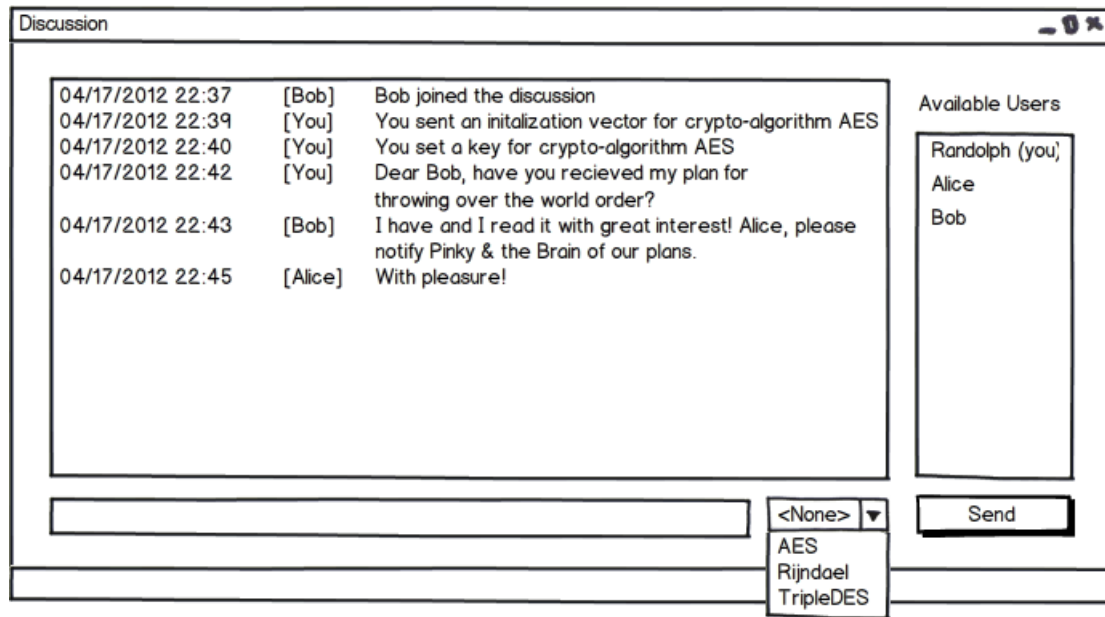


Abbildung 2.7: Mockup Gesprächsdialog

3 Konzept und Architektur

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 weitere (technisch) Rahmenbedingungen für das Konzept und die Architektur des Systems vorgibt. Danach wird das System aus der Bausteinperspektive betrachtet, es wird von der Komponentenebene bis zur Klassenebene das System modelliert und die Systemarchitektur festgelegt. Als weitere Sicht wird die Laufzeitsicht und die Verteilungssicht des Systems beleuchtet und spezifiziert.

3.1 Toolchain

Der erste Schritt der Konzepterstellung ist das Erarbeiten und Evaluieren der Toolchain. Die Toolchain beinhaltet Frameworks und Entwicklungswerkzeuge, mit denen der Entwickler später arbeitet und gibt einen groben technischen Rahmen vor, der bei der Konzepterstellung und insbesondere bei der Erarbeitung der Systemarchitektur berücksichtigt werden muss.

Basis der verwendeten Tools und Frameworks wird Microsofts .NET Framework in der Version 4.0 sein. Dies primär aufgrund der Möglichkeiten welche die Windows Communication Foundation (siehe Kapitel 4.2.2 auf Seite 36) zur Realisierung von Webservices bietet sowie Windows Presentation Foundation (siehe Kapitel 4.2.3 auf Seite 38) beziehungsweise Silverlight (siehe Kapitel 4.2.3.2 auf Seite 40) als Grundlage für grafische Applikationen für die Windows Plattform. Als Entwicklungswerkzeug wird Microsoft Visual Studio 2010 eingesetzt im Zusammenspiel mit dem Package Manager NuGet², der zur Verwaltung der externen Abhängigkeiten dient. Zur Quellcodeverwaltung wird der Hosting-Dienst Github³ eingesetzt, welcher als Versionsverwaltungssystem Git verwendet.

3.2 Bausteinsicht

Nach [Starke(2011)] und [Hruschka and Starke(2012)] 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.

¹auch Designphase genannt

²Mehr zum Package Manager NuGet ist unter <http://nuget.codeplex.com/> zu finden.

³Das Github Repository für YAEM ist unter <https://github.com/famstutz/YAEM> öffentlich verfügbar.

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.

3.2.1 Komponentendiagramm

Das Komponentendiagramm in Abbildung 3.1 stellt das System YAEM aus der Vogelperspektive dar und ist die höchstabstrahierte Ansicht der Bausteinsicht, die in diesem Projekt verfügbar ist.

Der Servicehost bezeichnet die Serverapplikation des Systems und implementiert die beiden Schnittstellen⁴ *IUserService* und *IMessagingService* (siehe Kapitel 3.2.3 auf Seite 24). Er stellt die Schnittstellen als Webservices den Clientapplikationen zur Verfügung.

Die Clientapplikationen (die in einer Vielzahl von Frameworks implementiert sein können) benutzen diese Webservices um mit der Serverapplikation zu kommunizieren. Die Clientapplikationen benutzen das Managed Extensibility Framework (mehr in Kapitel 4.2.1 auf Seite 35) um dynamisch die Implementierungen der Kryptosysteme (siehe Kapitel 3.2.4 auf Seite 26) zur Laufzeit laden und verwenden zu können. Dadurch wird gewährleistet, dass die Serverapplikation zu keinem Zeitpunkt die verschlüsselten Nachrichten, die über sie versendet werden, im Klartext lesen kann.

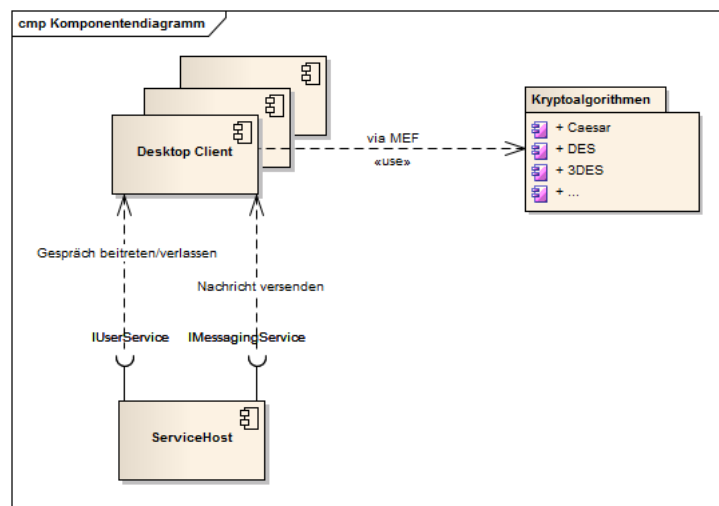


Abbildung 3.1: Komponentendiagramm

3.2.2 Domänenmodell

Das Domänenmodell (siehe Abbildung 3.2) umfasst nur die Klassen der Objekte, die über die Serviceschnittstelle von der Client- an die Serverapplikation beziehungsweise umgekehrt übertragen werden.

⁴Im Zusammenhang mit WCF auch Service Contracts genannt.

3 Konzept und Architektur

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

Sämtliche Domänenobjekte leiten von der Klasse *ObjectBase* ab. *ObjectBase* enthält eine Eigenschaft *Key* in Form einer Globally Unique Identifier zur eindeutigen Identifizierung der Objekte in den Repositories. Weiterhin implementiert *ObjectBase* das Interface *INotifyPropertyChanged*⁵, welches in der Windows Presentation Foundation und in Silverlight dazu dient, UI-Elemente, die an Datenobjekte gebunden sind, über geänderte Eigenschaften zu informieren.

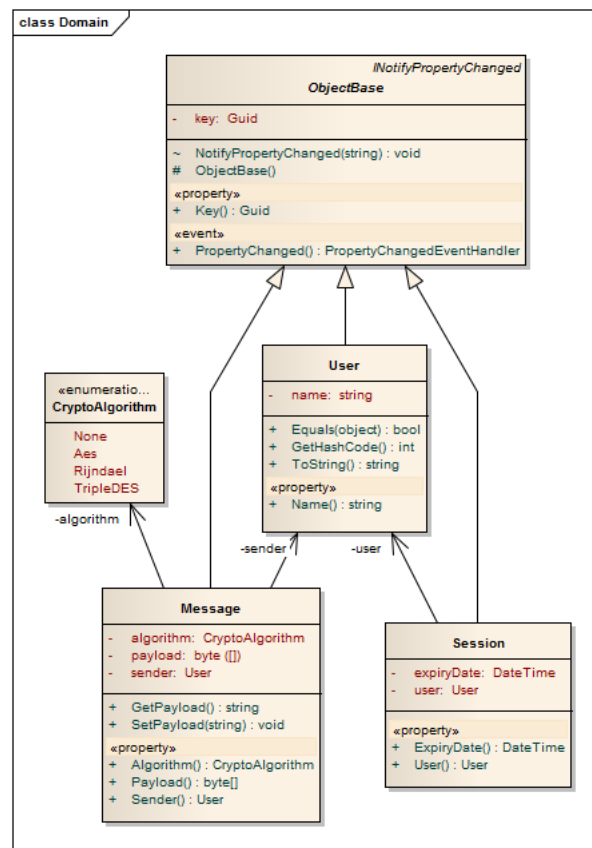


Abbildung 3.2: Klassendiagramm Domänenmodell

⁵Die genaue Schnittstellenbeschreibung ist in der MSDN unter <http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged.aspx> zu finden.

3.2.3 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, vorhergegebenen Anfragen behandelt und es werden auch keine Sitzungsinformationen ausgetauscht.

3.2.3.1 IUserService

Die in Abbildung 3.3 gezeigte Schnittstelle *IUserService* stellt die Benutzerverwaltungsfunktionalitäten des Systems zur Verfügung. Der Konsument des Webservices übergibt beim Anmelden (über die Methode *Join*) ein *User*objekt an den Server, 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 und den Benutzer vom Gespräch abmeldet.

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 Kapitel 3.2.3.3 auf der nächsten Seite) an alle registrierten Clients zu versenden, um sie über dem Gespräch neu hinzugekommene Benutzer oder Benutzer, die das Gespräch verlassen haben, zu informieren.

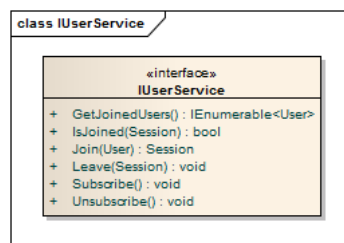


Abbildung 3.3: Klassendiagramm *IUserService*

3.2.3.2 IMessagingService

Über die Schnittstelle *IMessagingService* werden Nachrichten in Form eines *Message*objekts übertragen (siehe Klassendiagramm in Abbildung 3.4). Die Clientapplikation als Sender einer Nachricht übergibt 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.

3 Konzept und Architektur

Methode	Aufgabe
<i>NotifyNegotiateInitializationVector</i>	Teilt dem Client mit, dass für ein Kryptosystem ein neuer Initialisierungsvektor generiert wurde.
<i>NotifyNegotiateKey</i>	Teilt dem Client mit, dass für ein Kryptosystem 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*

Die Methoden *NegotiateInitializationVector* und *NegotiateKey* dienen zur Übermittlung des Initialisierungsvektors beziehungsweise des Schlüssels für ein Kryptosystem vom Typ *CryptoAlgorithm*.

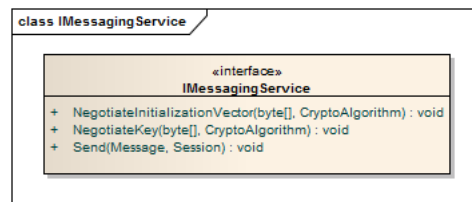


Abbildung 3.4: Klassendiagramm *IMessagingService*

3.2.3.3 IServiceCallback

Damit über HTTP Callbacks (der Mechanismus wird detaillierter in Kapitel 4.2.2.2 auf Seite 38 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))]` annotiert⁶ so dass beim Abonnieren des Webservices der generierte Serviceclient des Webservices 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 sowie in der Abbildung 3.5 als Klassendiagramm grafisch dargestellt sind.

⁶Details zum *ServiceContractAttribute* sind in der MSDN unter <http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute.aspx> zu finden.

3 Konzept und Architektur

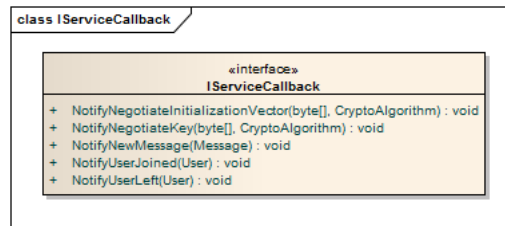


Abbildung 3.5: Klassendiagramm *IServiceCallback*

3.2.4 Kryptosysteme

Ein Ziel dieses Projekts war es, die Implementierungen von Kryptosystemen als eigenständige Assemblies zu kompilieren und zur Laufzeit den Clientapplikationen zur Verfügung zu stellen. Aus diesem Grund müssen alle Kryptosystemen die Schnittstelle *ICryptoProvider* implementieren.

ICryptoProvider liefert die grundlegenden Funktionalitäten zur Implementierung von symmetrischen Kryptosystemen, bei welchen beide Teilnehmer den gleichen Schlüssel verwenden (mehr zu symmetrischen und asymmetrischen Kryptosystemen in Kapitel 4.3 auf Seite 41). Alle Nutzdaten (verschlüsselt oder unverschlüsselt) werden als Bytearrays übergeben, so dass sie einfach und ohne Zusatzaufwand über einen Webservice serialisiert übermittelt werden können. Jedes symmetrische Kryptosystem besitzt 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 innerhalb eines Kryptosystems nicht zugänglich sein darf.

Die Methoden *Encrypt* beziehungsweise das Pendant *Decrypt* verschlüsseln bzw. entschlüsseln übergebene Nutzdaten. Im Klassendiagramm (siehe Abbildung 3.6) werden zusätzlich zur Schnittstelle *ICryptoProvider* auch die drei im Rahmen der Semesterarbeit realisierten Implementierungen der symmetrischen Kryptosystemen AES, Rijndael und Triple-DES dargestellt. Jedes dieser Kryptosysteme wird in eine eigene Assembly kompiliert und zur Laufzeit der Clientapplikationen mit Hilfe des Managed Extensibility Frameworks (siehe 4.2.1 auf Seite 35) geladen.

3.2.5 Server

Abbildung 3.7 zeigt das Klassendiagramm der Serverapplikation. Die Serverapplikation stellt die beiden Webservices *IMessagingService* und *IUserService* (siehe Kapitel 3.2.3 auf Seite 24) im zusammengeführten Servicehost *Services* der Aussenwelt zur Verfügung.

Dank des Bindings (siehe Kapitel 4.2.2.1 auf Seite 36) kann die Serverapplikation Delegates der Callbackoperationen aller registrierten Clientapplikationen verwalten und dadurch z.B. beim Eintreffen einer neuen Nachricht die Callbackoperation *NotifyNewMessage* (siehe *IServiceCallback* in Kapitel 3.1 auf der vorherigen Seite) der entsprechenden registrierten Clients aufrufen.

3 Konzept und Architektur

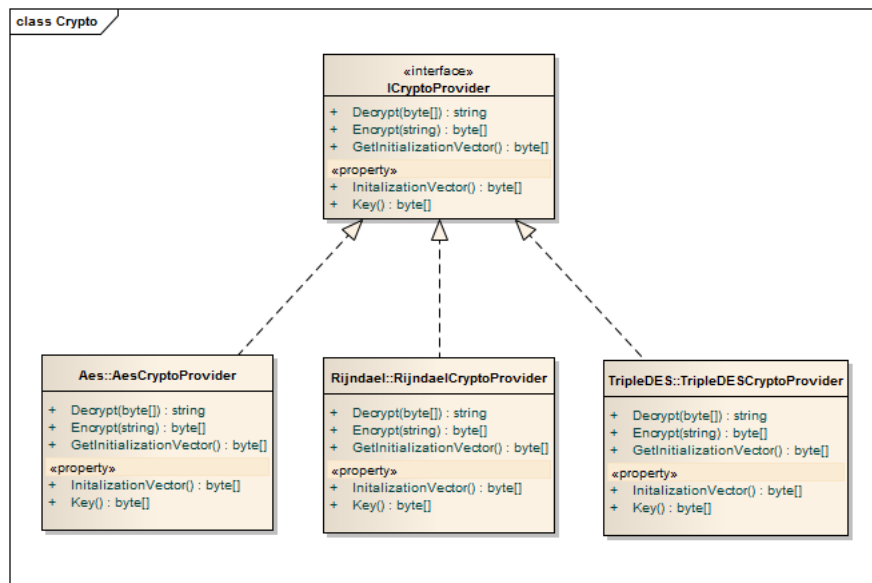


Abbildung 3.6: Klassendiagramm Kryptoalgorithmen

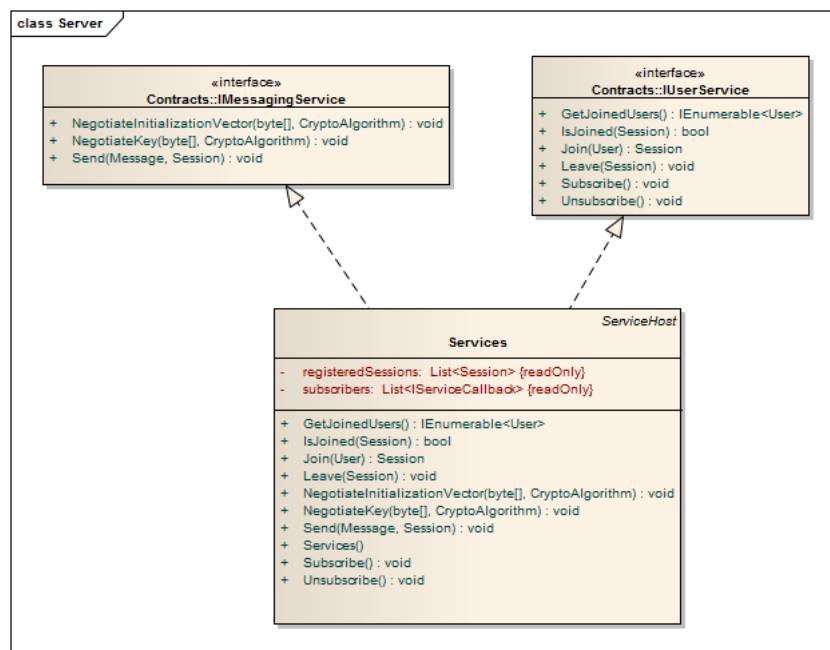


Abbildung 3.7: Klassendiagramm Server

3.2.6 Client

Für die Clientapplikationen wird kein gesondertes Konzept erstellt, da sich ihre Logik auf das Benutzen der Webservices der Serverapplikation (siehe 3.2.5 auf Seite 26) beschränkt, die über eine Service Reference⁷ eingebunden werden. Dadurch werden die Clientapplikationen gezwungen, das Interface *IServiceCallback* (siehe 3.2.3.3 auf Seite 25) zu implementieren.

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 diese zusammenwirken (nach [Starke(2011)]). 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.

3.3.1 Gespräch beitreten

Möchte ein Benutzer einem Gespräch beitreten so gibt er über den Gesprächsbeitrittdialog seinen gewünschten Benutzernamen ein. Die Schleife “UserName in use” stellt dar, dass der Benutzer solange einen Benutzernamen auswählen muss bis er einen Benutzernamen wählt, der im Gespräch noch nicht vergeben ist. Ist der Benutzername frei, so wird von der Serverapplikation in der Sessionverwaltung ein neues Session-Ticket gelöst, das der Clientapplikation zurückgegeben wird.

Das Sequenzdiagramm ist in Abbildung 3.8 ersichtlich.

3.3.2 Gespräch verlassen

Möchte der Benutzer das Gespräch verlassen, so initiiert er über den Gesprächsdialog eine Anfrage an die Serverapplikation zum Verlassen des Gesprächs. In der Sessionverwaltung wird dann das Session-Ticket des Benutzers für ungültig erklärt und der Clientapplikation wird mitgeteilt, dass der Benutzer erfolgreich das Gespräch verlassen hat und keine Nachrichten mehr senden oder empfangen kann.

Das Sequenzdiagramm hierzu ist unter Abbildung 3.9 zu finden.

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 ausgewählten Kryptosystems. Ist dieser

⁷In der MSDN finden sich unter <http://msdn.microsoft.com/en-us/library/bb907578.aspx> mehr Informationen zu Service References.

3 Konzept und Architektur

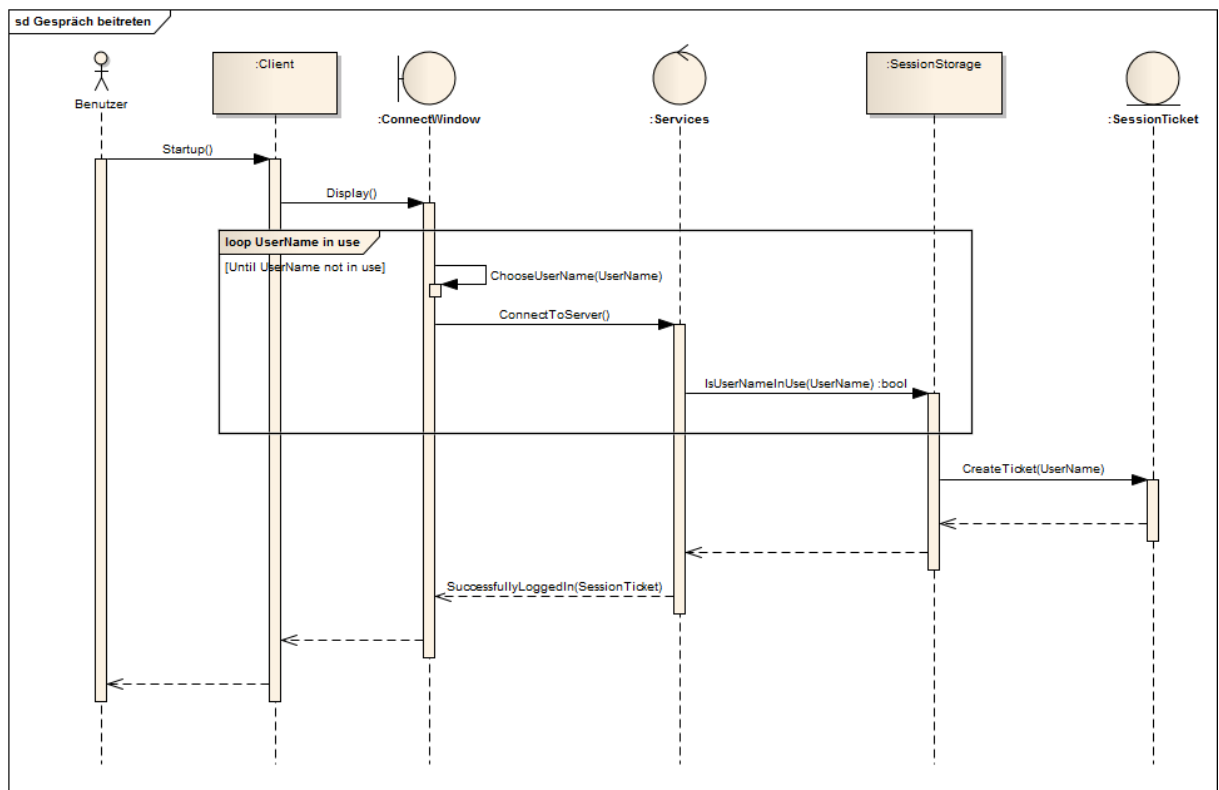


Abbildung 3.8: Sequenzdiagramm Gespräch beitreten

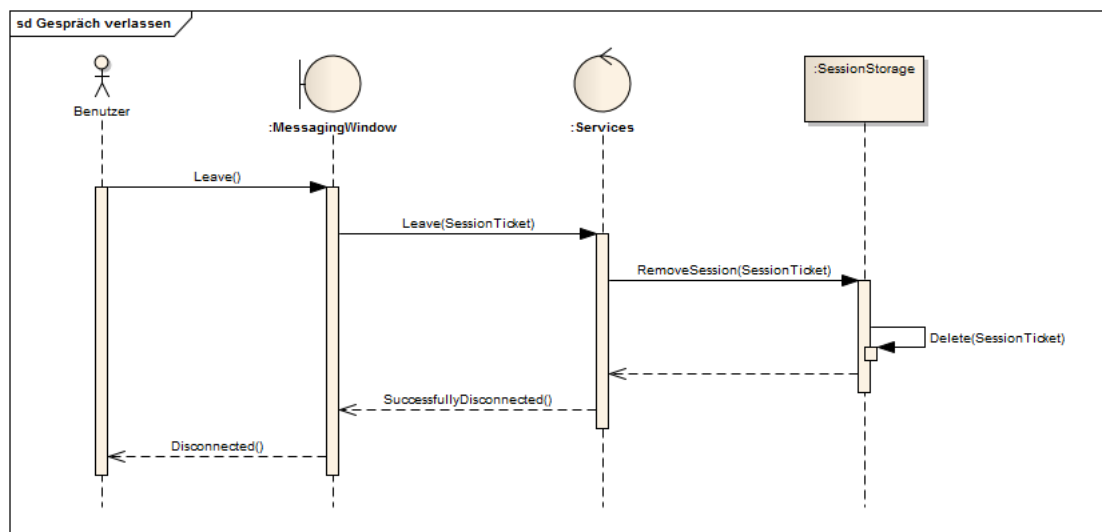


Abbildung 3.9: Sequenzdiagramm Gespräch verlassen

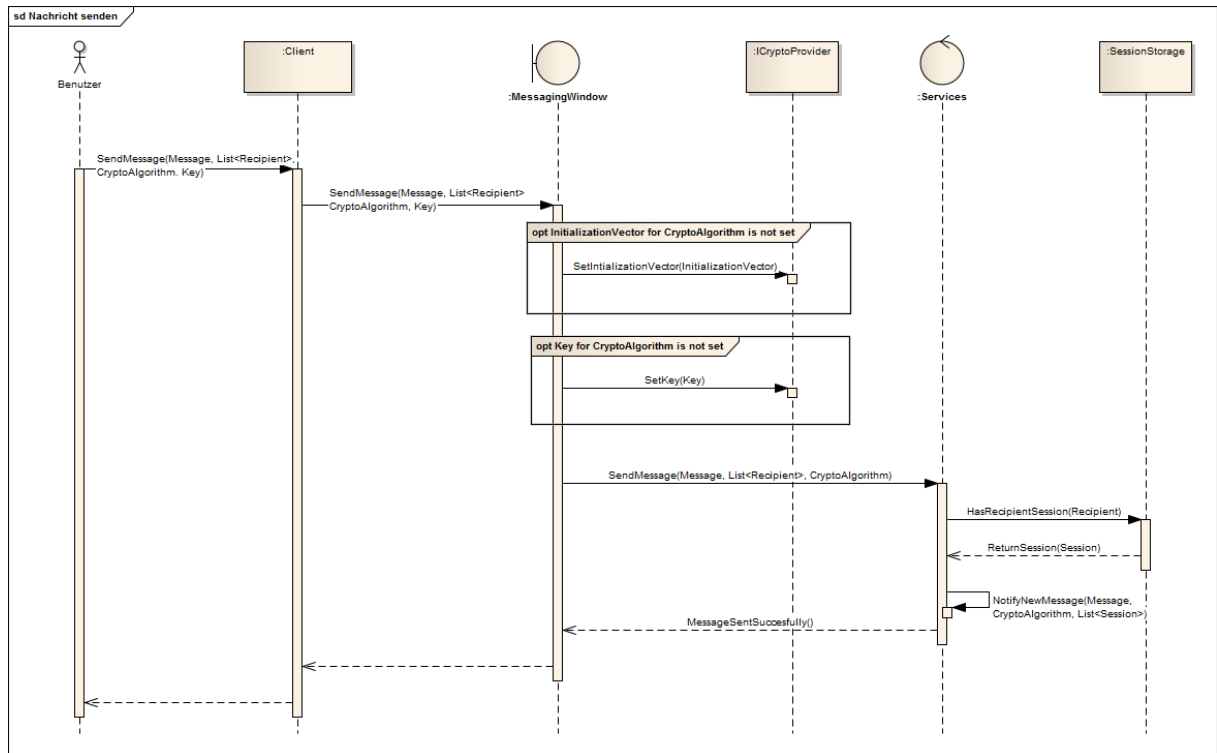


Abbildung 3.10: Sequenzdiagramm Nachricht senden

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 der Serverapplikation weitergegeben. Besitzen die ausgewählten Empfänger jeweils ein gültiges Session-Ticket und haben diese beim Webservice einen Delegaten als Callback angemeldet, so wird ihnen die Nachricht übermittelt. Der Benutzer erhält anscheinend eine Bestätigung des erfolgreichen Nachrichtenversandes im Gesprächsprotokoll.

Dieses Sequenzdiagramm ist in Abbildung 3.10 ersichtlich.

3.4 Verteilungssicht

Nach [Starke(2011)] 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

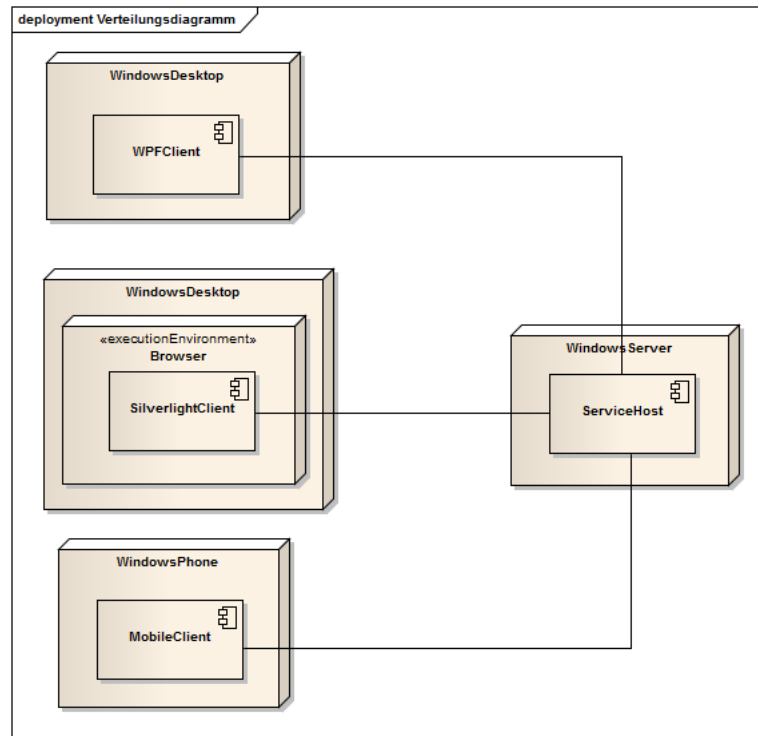


Abbildung 3.11: Verteilungsdiagramm

[Hruschka and Starke(2012)]).

3.4.1 Verteilungsdiagramm

Die Verteilungssicht dieser Projektdokumentation enthält nur ein sehr rudimentär ausgearbeitetes Verteilungsdiagramm (siehe Abbildung 3.11). 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 auf die Verteilung des Systems einzugehen.

Generell können die einzelnen Clientapplikationen auf jeglicher Hardware laufen, benötigen jedoch jeweils das passende Betriebssystem beziehungsweise im Fall des Silverlight-Clients einen Browser mit Silverlight-Plugin. Zwischen den Clientapplikationen und der Serverapplikation in Form des Servicehosts wird eine Netzwerkverbindung vorausgesetzt bei welcher die passenden Ports (siehe Binding Kapitel 4.2.2.1 auf Seite 36) geöffnet und zugänglich sind. Auch ist eine Verteilung über das WWW denkbar.

4 Implementierung

Das Kapitel Implementierung beschäftigt sich zunächst mit der Wahl der geeigneten Entwicklungsmethode für dieses Projekt. Als Nächstes werden verschiedene Technologien vorgestellt, die innerhalb des Projekts Anwendung finden und deren Kenntniss für das weitere Verständnis des Quellcodes oder dieser Dokumentation notwendig sind. Auch wird auf einzelne besondere Entwurfsmuster näher eingegangen, insbesondere im Zusammenhang mit den UI-Frameworks.

Anschliessend wird ein theoretischer Überblick über asymmetrische und symmetrische Kryptosysteme gegeben sowie aufgezeigt, wie diese in YAEM eingebunden werden können.

Am Ende des Kapitels werden die einzelnen Visual Studio-Projekte als Teilkomponenten des Projekts vorgestellt und es wird auf in der Implementierungsphase aufgetretene Probleme hingewiesen, sowie ihre Auswirkungen auf Teile des Projekterfolgs erläutert.

4.1 Entwicklungsmethoden

Als pragmatischer Softwareentwickler versucht man den Blick auf das Wesentliche zu konzentrieren: Anforderungen in ein funktionierendes und warbares Programm zu überführen, das die Anwender begeistert (nach [Hunt and Thomas(1999)]). Um dieses Ziel zu erreichen existiert eine breite Palette an Entwicklungsmethoden, die den Entwickler bei seiner (pragmatischen) Arbeitsweise unterstützen. Insbesondere agile Entwicklungsmethoden wie Scrum oder Extreme Programming (XP) unterstützen einen Entwickler oder ein Entwicklungsteam bei der Konzentration auf die Implementierung von Funktionalitäten in einem Softwareprodukt und beugen Ablenkungen vor.

Für dieses Projekt wurde Test Driven Development (TDD) als Entwicklungsmethode gewählt, da die iterative Entwicklung, die auf dem Erstellen des Unit-Tests¹ fundiert (genauer beschrieben in Kapitel 4.1.1 auf der nächsten Seite) die Fokussierung auf die für den Benutzer relevanten Funktionalitäten ermöglicht.

Studien wie [Nachiappan Nagappan and Williams(2008)] beweisen, dass mit Hilfe von TDD die allgemeine Qualität eines Softwareprodukts gesteigert werden kann, da z.B. schon die Fehlerdichte² zwischen 40% und 90% relativ kleiner ist als bei Softwareprodukten, die nicht mit TDD entwickelt wurden. Es wird zwar der initiale Entwicklungsaufwand einer Funktion zwischen 15% und 35% erhöht, jedoch wurde dies im Nachhinein durch geringere Wartungskosten ausgeglichen.

¹Mehr zu automatisierten Tests findet sich im Kapitel 5.1 auf Seite 50.

²gemessen in Anzahl Fehler pro tausend Zeilen Code

4.1.1 Testgetriebene Entwicklung

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

Die Tests werden meist innerhalb eines Unit-Test-Frameworks (siehe Kapitel 5.1 auf Seite 50) entwickelt und ausgeführt. Zu Beginn werden die Tests fehlschlagen, da noch gar kein Code geschrieben worden ist. Ziel des Entwicklers ist es, so lange Code zu implementieren und zu verbessern, bis alle zur Softwarekomponente zugehörigen Tests bestanden werden und kein bisheriger Test fehlschlägt.

Kommen neue Anforderungen oder Funktionen zur Anwendung hinzu, werden zuerst neue Tests implementiert und anschliessend die bestehende Codebasis erweitert. Danach werden alle Tests erneut durchgeführt und es werden solange Anpassungen am Code durchgeführt, bis alle Tests bestanden werden.

Zusammengefasst folgt man drei einfachen Schritten bei der Anwendung von TDD (frei nach [Folwer(2012)], siehe Abbildung 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 er ist leicht zu ändern und zu warten da eventuelle Seiteneffekte einer Codeanpassung schnell entdeckt werden. Die enthaltenen Funktionen werden durch die Tests zugleich dokumentiert, Fehler werden früher entdeckt und sind durch die durchgeführten Mini-Iterationen leichter lokalisierbar. Besonders in grossen Teams entsteht so ein Qualitätsbewusstsein über das ganze Projekt hinweg.

Nachteilig ist anzumerken, dass konsequent sämtlicher Code testgetrieben 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 funktioniert TDD auch nur, wenn alle Entwickler ein fundiertes Wissen über Testmethodiken 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 die in der Semesterarbeit verwendeten Technologien und dient zum besseren Verständnis der einzelnen Teile des Softwareprodukts, die in einem späteren Kapitel (siehe Kapitel 4.4 auf Seite 43) genauer beschrieben werden.

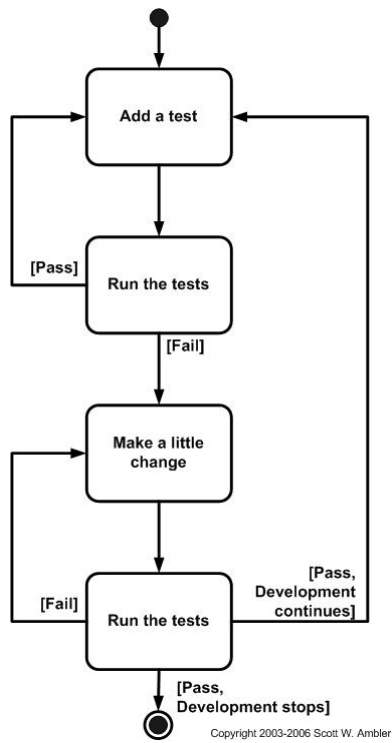


Abbildung 4.1: Schritte in TDD (aus [Ambler(2011)])

Die einzelnen Unterkapitel bieten einen losen Überblick über verschiedenste Technologien und Produkte und erläutern in welcher Komponenten und warum die jeweilige Technologie eingesetzt wird.

4.2.1 Managed Extensibility Framework

Ein Ziel des Softwareproduktes YAEM war es, zur Laufzeit beliebige Kryptosysteme den Clientapplikationen zur Verfügung zu stellen, die als Erweiterungen in die Applikation eingebunden werden können. MEF bietet genau diese Funktionalität.

Das Managed Extensibility Framework³ (oder kurz MEF) ist eine Bibliothek für die Erstellung von leichtgewichtigen, erweiterbaren Anwendungen. Es erlaubt Anwendungsentwicklern Erweiterungen zu Applikationen zu schreiben, die keinerlei Konfiguration benötigen. Ausserdem können Erweiterungen leicht gekapselt werden und es können somit fragile, harte Abhängigkeiten vermieden werden. MEF ermöglicht nicht nur Erweiterungen, die innerhalb von Anwendungen wiederverwendet werden können, sondern auch über Anwendungsgrenzen hinweg.

MEF besteht aus einem Katalog (*Catalog*) und einem sogenannten Behälter (*CompositionContainer*). Ein Katalog ist verantwortlich für das Entdecken der Erweiterungen und der Behälter koordiniert die Instanziierung der in ihnen gelagerten Ersatzteile und erfüllt deren Abhängigkeiten. MEF-Erweiterungen sind Ersatzteile (*ComposablePart*), die ein oder mehrere Exporte bieten und von einem oder mehreren extern gelieferten Importe abhängen können. Exporte und Import haben jeweils einen Vertrag, der als Schnittstelle zwischen Exporten und Importen dient. Der Behälter interagiert mit Katalogen um Zugriff zu den Ersatzteilen zu erhalten. Er behebt die Abhängigkeiten des Ersatzteils und stellt seine Exporte der Aussenwelt zur Verfügung.

Sämtliche Exporte und Importe werden über Annotationen gesteuert und mittels Reflection können zur Laufzeit diese Annotationen aus Assemblies gelesen werden.

Abbildung 4.2 verdeutlicht die interne Struktur von MEF. Innerhalb von MEF entspricht ein Ersatzteil (Part) einem Kryptosystem, das in einer eigenen Assembly residiert. Alle Kryptosysteme implementieren dabei die Schnittstelle *ICryptoProvider* (siehe Kapitel 3.2.4 auf Seite 26) und werden mit `[Export(typeof(ICryptoProvider))]` sowohl `[CryptoAlgorithm(Algorithm = CryptoAlgorithm.SpecificCryptoAlgorithm)]` annotiert.

³MEF ist Open Source und unter <http://mef.codeplex.com/> zusammen mit Beispielen herunterzuladen.

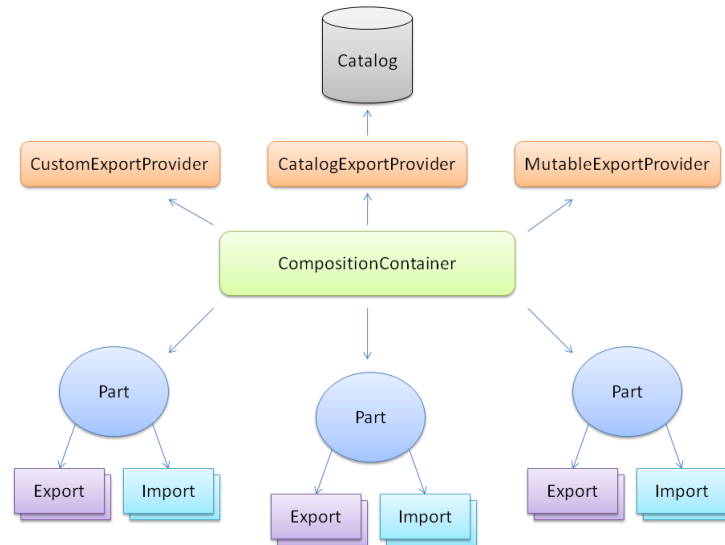


Abbildung 4.2: MEF Struktur

4.2.2 Windows Communication Foundation

Zur Kommunikation zwischen den Clientapplikationen und der Serverapplikation wird die Windows Communication Foundation (kurz WCF) eingesetzt. WCF ist seit der Version 3.0 ein Bestandteil des .NET Frameworks und ist eine dienstorientierte Kommunikationsplattform für verteilte Anwendungen.

Durch WCF werden verschiedene Kommunikationstechnologien wie DCOM, MSMQ oder Webservices unter einer einheitlichen Programmierschnittstelle zusammengefasst. WCF wird hauptsächlich zur Entwicklung von serviceorientierten Architekturen (SOA) verwendet. Dabei abstrahiert WCF das Konzept des Endpunkts durch eine Trennung in Address, Binding und Contract. Die Adresse (Address) ist ein URI, die den Ort des Dienstes beschreibt und somit seine Erreichbarkeit für Dienstkonsumenten kennzeichnet. Die Anbindung (Binding) beschreibt die Art der Kommunikation, worunter unter anderem die Merkmale der Kodierung des Protokolls fallen. Der Vertrag (Contract) stellt die Dienstdefinition, insbesondere die zur Verfügung gestellten Methoden, dar.

4.2.2.1 Bindings

WCF unterscheidet zwischen der Implementierung einer Softwarekomponente und wie diese mit anderen Softwarekomponenten kommuniziert. Mithilfe von Bindings werden Transport, Kodierung und Protokolldetails angegeben, die von Client- und Serverapplikationen zur Kommunikation benötigt werden. WCF generiert mit Bindings die zugrundeliegende Übertragungsdarstellung des Endpunkts. Deshalb müssen die an der Kommunikation beteiligten Parteien die Details des Bindings aufeinander abstimmen. Tabelle 4.1 bietet eine Übersicht über die im Lieferumfang von WCF vorhandenen Bindings an.

4 Implementierung

Binding	Beschreibung
<i>BasicHttpBinding</i>	Ein Binding, das sich für die Kommunikation mit Webservices eignet. Das Binding verwendet HTTP als Transportprotokoll und XML als Kodierung.
<i>WSHttpBinding</i>	Ein sicheres Binding, das sich für nicht-duplexfähige Service Contracts eignet.
<i>WsDualHttpBinding</i>	Ein sicheres Binding, das die Kommunikation mit Service Contracts über SOAP mit Duplexkanälen anbietet.
<i>WsFederationHttpBinding</i>	Das <i>WSFederationHttpBinding</i> unterstützt das WS-Federationprotokoll und ermöglicht es den Dienstbenutzer zu authentifizieren und zu autorisieren.
<i>NetTcpBinding</i>	Dieses Binding ist ein sicheres und optimiertes Binding, das sich für die computerübergreifende Kommunikation zwischen WCF-Anwendungen eignet.
<i>NetNamedPipeBinding</i>	Das <i>NetNamedPipeBinding</i> eignet sich zur Kommunikation zwischen WCF-Anwendungen auf einem Computer.
<i>NetMsmqBinding</i>	Eine warteschlagenbasiertes (queue-based) Binding, das zur computerübergreifenden Kommunikation zwischen WCF-Applikationen eingesetzt werden kann.
<i>NetPeerTcpBinding</i>	Zwischen mehreren Rechnern wird mit Hilfe dieses Bindings eine sichere Kommunikation ermöglicht.
<i>WebHttpBinding</i>	Dieses Binding dient zur Übertragung von HTTP-Requests die nicht über SOAP bereitgestellt werden können.
<i>MsmqIntegrationBinding</i>	Zur computerübergreifenden Kommunikation zwischen WCF-Applikation und MSMQ-Anwendungen wird dieses Binding benutzt.

Tabelle 4.1: Bindings

Da für YAEM zwingend ein duplexfähiges Binding verwendet muss, steht einzig das *WSDualHttpBinding* zur Verfügung, das sowohl dem Webservice wie auch dem konsumierenden Client die Möglichkeit bietet, Nachrichten zu versenden, wie auch Nachrichten zu empfangen.

4.2.2.2 Callbacks über WSDualHttpBinding

Das *WSDualHttpBinding* definiert ein sicheres und interoperables Binding, das für duplexfähige Service Contracts oder für die Kommunikation über SOAP-Vermittler geeignet ist⁴. *WSDualHttpBinding* bietet dieselbe Unterstützung von Webservices wie *WSHttpBinding*, erlaubt jedoch zusätzlich die Verwendung von Duplexchannels. Das Binding macht es erforderlich, dass der Client einen öffentlichen URI zur Verfügung stellt, der einen Callback für den Webservice beinhaltet.

Wird ein Service Contract mit dem *ServiceContractAttribute.CallbackContract*⁵ annotiert, so muss dieser *CallbackContract* von Webservicekonsumenten implementiert werden. Dadurch stellt der Client implizit einen öffentlichen URI zur Verfügung, der vom Webservice aufgerufen werden kann.

Die Serverapplikation kann so über den Aufruf der Methode *OperationContext.GetCallbackChannel<T>*⁶ einen Callback-Delegaten erhalten, über welchen die Serverapplikation direkt Nachrichten an den registrierten Webservicekonsumenten versenden kann.

4.2.3 Windows Presentation Foundation

Die Windows Presentation Foundation (kurz WPF) ist ein Framework zur Erstellung von grafischen Benutzeroberflächen für Windows-basierte Applikationen, das in 2006 zusammen mit dem .NET Framework 3.0 auf den Markt gekommen ist. WPF versucht, ein konsistentes Programmiermodell für die Erstellung von Anwendungen zu bieten und lieferte eine Trennung zwischen der Benutzeroberfläche und der Businesslogik.

WPF benutzt Extensible Markup Language (XAML), ein XML-Derivat, um verschiedene UI-Elemente zu definieren und miteinander zu verknüpfen. WPF-Applikationen können sowohl als eigenständige Programme als auch innerhalb einer Website als eingebettetes Objekt verteilt werden. WPF kombiniert UIs, 2D Grafiken, 3D Grafiken, Dokumente und Multimediaobjekte in einem einzigen Framework. Diese Elemente können miteinander verbunden werden und unterschiedliche Zustände aufgrund der anzuzeigenden Daten annehmen. Die Abbildung 4.3 zeigt die Hauptfunktionalitäten von WPF.

⁴Siehe MSDN Library <http://msdn.microsoft.com/en-us/library/ms731821.aspx>.

⁵Siehe MSDN Library <http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute.callbackcontract.aspx>.

⁶Siehe MSDN Library <http://msdn.microsoft.com/en-us/library/ms575542.aspx>.

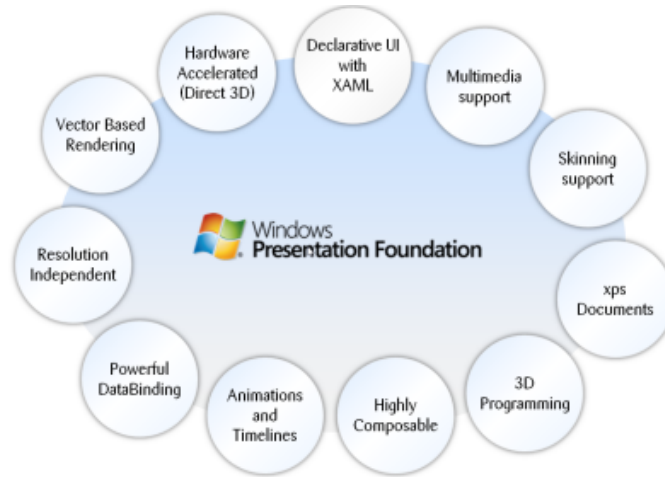


Abbildung 4.3: Hauptfunktionalitäten von WPF (aus[Moser(2012)])

Vor WPF wurden die meisten Desktopanwendungen für windowsbasierte Systeme mit Windows Forms entwickelt. Dabei ist Windows Forms ein Wrapper für die Windows-API in Managed Code. WPF basiert nicht mehr auf der Windows-API sondern zeichnet die Benutzerfläche selbst. Dadurch ist eine bessere Unterstützung von Templates gegeben und auf längere Sicht eine erhöhte Portierbarkeit. Ausserdem ist WPF hardwarebeschleunigt, nutzt also den Prozessor der 3D-Grafikkarte, eine Ressource die insbesondere im normalen Alltagsgebrauch von Windows brachliegt und Applikationen merkbar beschleunigt.

WPF ist im Moment die Referenztechnologie zur Entwicklung von windowsbasierten Desktopapplikationen. Aus diesem Grund wird es für die YAEM Desktopapplikation als UI-Framework verwendet.

4.2.3.1 Model View ViewModel

Einer der grossen Vorteile von WPF ist die Möglichkeit zur Trennung von Logik und Präsentation. Dies kann über das Architekturmuster MVVM erreicht werden, das eine Spezialisierung des Entwurfsmusters Presentation Model von Martin Fowler ist (siehe [Fowler(2004)]). MVVM basiert zum grössten Teil auf dem Muster Model View Controller (MVC), und ist auf moderne UI-Entwicklungsplattformen wie WPF oder Silverlight ausgelegt, bei denen es eine Rollentrennung zwischen einem UI-Entwickler und einem Back-End-Entwickler gibt. Das View-Model von MVVM ist dafür verantwortlich, die Datenobjekte des Modells derart blosszulegen, dass sie einfach verwaltet und benutzt werden können.

MVVM wurde entworfen, um mittels der spezifischen Funktionen in WPF durch Entfernung praktisch allen Code-Behind Quellcode von der View-Schicht eine bessere Trennung der View-Schicht vom Rest zu ermöglichen. Statt View-Code zu schreiben, kann ein Designer die XAML verwenden und Bindungen zum View-Model erzeugen, die durch Anwendungsentwickler geschrieben und gewartet werden. Diese Rollentrennung erlaubt es Designern, ihren Aufmerksamkeit auf die Umsetzung von UX-Anforderungen zu legen,

4 Implementierung

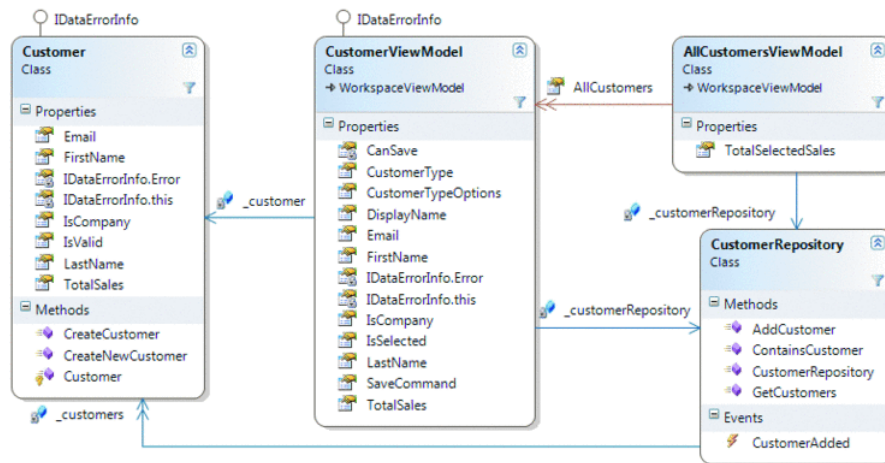


Abbildung 4.4: MVVM-Beispielimplementierung (aus [Smith(2009)])

als auf Programmierung oder Geschäftslogik. Dadurch können die einzelnen Schichten einer Anwendung von mehreren Arbeitsgruppen gleichzeitig entwickelt werden können.

Das MVVM-Muster beinhaltet die folgenden Elemente (nach [Smith(2009)]):

- **Model:** wie im klassischen MVC-Muster bezieht sich das Modell entweder auf ein Objektmodell das den realen Zustandsinhalt repräsentiert (objektorientierter Ansatz) oder die Datenzugriffsschicht, welche diesen Inhalt repräsentiert (datenzentrischer Ansatz).
- **View:** wie im klassischen MVC-Muster bezieht sich die View auf alle Elemente die durch die GUI angezeigt werden, etwa Schaltflächen, Fenster, Grafiken, und andere Steuerelemente.
- **View-Model:** das View-Model ist ein “Modell der View” was bedeutet, dass es eine Abstraktion der View ist, die auch der Datenbindung zwischen View und Model dient. Es könnte als ein spezieller Aspekt eines Controllers (im MVC-Muster) für die Datenbindung und -konvertierung betrachtet werden, der Information vom Model zu solcher für die View konvertiert und Befehle von der View zum Model reicht. Das View-Model legt öffentliche Eigenschaften, Befehle und Abstraktionen bloss. Das View-Model kann mit einem konzeptionellen Zustand der Daten verglichen werden, im Gegensatz zum realen Zustand der Daten im Model.

Die Abbildung 4.4 zeigt eine beispielhafte Implementierung des MVVM-Musters von [Smith(2009)] mit allen wichtigen Elementen.

4.2.3.2 Silverlight

2007 vorgestellt ist Silverlight ein Applikationsframework zur Entwicklung von Rich Internet Applications (RIA). Silverlight Applikationen werden dabei innerhalb eines Brow-

serplugins ausgeführt, das für eine Vielzahl von Browsern und Betriebssystemen existiert. Silverlight benutzt ein ähnliches Grafiksysteme wie WPF (siehe Kapitel 4.2.3 auf Seite 38) und integriert Multimedia, Grafiken, Animationen und Interaktivität in eine einzige Laufzeitumgebung. Silverlight dient ausserdem als Entwicklungsplattform für Windows Phone.

Wie in WPF wird auch in Silverlight XAML als Beschreibungssprache der Benutzeroberflächen benutzt, allerdings in einem leicht verminderten Umfang. Da Silverlightapplikationen gleichzeitig im Browser und auf einem Winows Phone laufen können, wurde es als UI-Framework für die YAEM Mobileapplikation ausgewählt mit Hinblick auf eine eventuelle Webapplikation. Dank Silverlight können mit minimsten Entwicklungsaufwand für zwei unterschiedliche Plattformen Anwendungen entwickelt werden.

4.2.4 Mock-Objekte

Bei Unit-Tests möchte man möglichst nur eine kleine Einheit eines Programms isoliert testen. Meistens stehen jedoch Abhängigkeiten zu anderen Einheiten diesem Vorhaben im Weg, die man für diesen Test ignorieren möchte. Soll die Interaktion eines Objektes mit seiner Umgebung überprüft werden, muss vor dem eigentlichen Test die Umgebung nachgebildet werden. Das kann umständlich, zeitaufwändig, oder gar nur eingeschränkt oder überhaupt nicht möglich sein (nach [Freeman and Pryce(2009)]). Aus diesem Grund kann man Mock-Objekte, auch “Dummy”-Objekte oder Attrappen genannt, einsetzen, die als Platzhalter für echte Objekte und Abhängigkeiten dienen.

Mock-Objekte (in YAEM kommt das Framework Moq⁷ zum Einsatz) implementieren Schnittstellen, über die das zu testende Objekt auf seine Umgebung zugreift. Sie stellen sicher, dass die erwarteten Methodenaufrufe vollständig, mit den korrekten Parametern und in der erwarteten Reihenfolge durchgeführt werden. Das Mock-Objekt liefert keine Echtdaten zurück, sondern vorher zum Testfall passend festgelegte Werte.

Durch den Einsatz von Mock-Objekten kann man einzelne Softwarteile testen ohne dass die Tests die Ausmasse von Integrationstests annehmen.

4.3 Kryptosysteme

Ein Kryptosystem besteht aus einem Algorithmus der eine Eingabemenge parametergesteuert in eine Ausgabemenge transformiert. Es dient zur Geheimhaltung von übertragenen und gespeicherten Informationen gegenüber Dritten. Geheimhaltung wird dabei durch Transformation der Nachricht erzielt, so dass die transformierte Nachricht für Dritte keine konstruierbaren semantischen, statistischen oder strukturellen Korrelationen zum Original mehr aufweist (nach [Geiselmann(2009)]).

Ein symmetrisches Kryptosystem ist ein Kryptosystem, bei welchem im Gegensatz zu einem asymmetrischen Kryptosystem beide Teilnehmer den gleichen Schlüssel verwenden.

Ein asymmetrisches Kryptosystem (oder Public-Key-Kryptosystem) hingegen ist ein kryptographisches Verfahren, bei dem die kommunizierenden Parteien keinen gemeinsa-

⁷Details zu Moq sind unter <http://code.google.com/p/moq/> zu finden.

men Schlüssel kennen müssen. Ein Benutzer erzeugt hierzu ein Schlüsselpaar, das aus einem geheimen Teil (privater Schlüssel) und einem nicht geheimen Teil (öffentlicher Schlüssel) besteht. Der öffentliche Schlüssel ermöglicht es jedem, Daten für den Inhaber des privaten Schlüssels zu verschlüsseln, dessen digitale Signaturen zu prüfen oder ihn zu authentifizieren. Der private Schlüssel ermöglicht es seinem Inhaber, mit dem öffentlichen Schlüssel verschlüsselte Daten zu entschlüsseln, digitale Signaturen zu erzeugen oder sich zu authentisieren.

4.3.1 Symmetrische Kryptosysteme

Da der Fokus dieses Projekts nicht auf der Implementierung von Kryptosystemen liegt, werden im Moment nur symmetrische Kryptosysteme unterstützt, die einfacher zu handhaben sind, da zum Ent- und Verschlüsseln derselbe Schlüssel benutzt wird.

Man teilt die symmetrischen Verfahren in Blockchiffren-basierte und Stromchiffren-basierte Verfahren auf. Mit Stromchiffren wird der Klartext Zeichen für Zeichen verschlüsselt, um den Geheimtext zu erhalten, beziehungsweise entschlüsselt, um den Klartext zu erhalten. Eine Blockchiffre arbeitet mit einer festen Blockgröße und ver- beziehungsweise entschlüsselt mehrere Zeichen in einem Schritt. Um damit Texte beliebiger Länge verschlüsseln zu können, sind Betriebsmodi festgelegt, die bestimmen, wie die Blockchiffre verwendet wird.

Im Rahmen dieses Projekts werden drei Algorithmen mittels der Schnittstelle *ICryptoProvider* (siehe Kapitel 4.4.4 auf Seite 45) implementiert, die allesamt aus dem *System.Security.Cryptography* Namespace des .NET Frameworks stammen und deren Funktionsweise in den folgenden Unterkapiteln kurz beschrieben wird.

4.3.1.1 Advanced Encryption Standard

Der Advanced Encryption Standard (AES) ist ein symmetrisches Kryptosystem, das als Nachfolger des DES und Triple-DES im Oktober 2000 vom National Institute of Standards and Technology (NIST) als Standard bekanntgegeben wurde.

Der AES ist eine Blockchiffre, wobei sie auf einer Verflechtung von Substitutions- und Permutationsvorgängen basiert, was bedeutet, dass eine Reihe mathematischer Vorgänge miteinander verbunden werden um Daten höchstmöglich zu modifizieren beziehungsweise zu verschlüsseln.

4.3.1.2 Rijndael

Nach [Shawnfa(2006)] ist der Rijndael Algorithmus identisch mit demjenigen des AES, allerdings können bei Rijndael die Block- und Schlüsselgrößen unabhängig voneinander aus 128, 160, 192, 224 oder 256 Bits gewählt werden. AES legt die Blockgröße immer auf 128 bits fest, die Schlüsselgröße kann hingegen eine Länge von entweder 128, 192 oder 256 Bits aufweisen.

4.3.1.3 Triple-DES

Der Triple-DES (Data Encryption Standard) Algorithmus ist ein Nachfolger des DES-Algorithmus und wurde im Jahr 1976 als offizieller Standard der US-Regierung bestätigt. Im Gegensatz zum DES mit seiner Schlüssellänge von 56 Bits (die heute als nicht ausreichend sicher betrachtet wird) kann die Schlüssellänge durch Mehrfachfachverwendung des DES einfach vergrößert werden.

Der DES verwendet eine Blockchiffre, die den Klartext in 64 Bit-Blöcke unterteilt und verschlüsselt. Triple-DES verwendet drei 64 Bit-Schlüssel, die dadurch eine gesamte Schlüssellänge von 192 Bit ergeben. Dabei wird die erste Verschlüsselung mit dem zweiten Schlüssel verschlüsselt und der resultierende Kryptotext erneut mit dem dritten Schlüssel verschlüsselt.

4.3.2 Unterstützung asymmetrischer Kryptosysteme

Wie schon in Kapitel 4.3.1 erwähnt, unterstützt *YAEM.Crypto* im Moment nur symmetrische Kryptosysteme, da die Schnittstelle *ICryptoProvider* darauf basiert, dass derselbe Schlüssel zum Ver- und Entschlüsseln der Nutzdaten verwendet wird.

Auch bei asymmetrischen Verfahren ist der Austausch von Schlüsseln wichtig. Muss man den öffentlichen Schlüssel doch zugänglich machen damit er zur Verschlüsselung benutzt werden kann und den dazu passende privaten Schlüssel irgendwo sicher verwahren um nicht das gesamte Konzept auszuhebeln.

Würde YAEM auch asymmetrische Kryptosysteme unterstützen müssen, so müsste neben *ICryptoProvider* eine zweite Schnittstelle mit dem Namen *IASymmetricCryptoProvider* definiert werden, welche anstelle der Properties *Key* und *InitializationVector* eine Möglichkeit besitzt, den öffentlichen Schlüssel aus dem installierten Zertifikat eines Benutzers auszulesen⁸. Der private Schlüssel wird dabei in dem dafür vorgesehenen Container *CspParameters*⁹ abgelegt, der ein Wrapper von Microsofts Cryptography API (CAPI) darstellt.

4.4 Komponenten im Detail

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

Zur besseren Orientierung innerhalb des Systems wird auf Abbildung 4.5 verwiesen, welche die Struktur Visual Studio-Solution darstellt. Die Namensgebung der Unterkapitel hält sich dabei an die Namespaces der Projekte.

⁸Dazu findet sich ein Beispiel in der MSDN: <http://msdn.microsoft.com/en-us/library/system.security.cryptography.asymmetricalgorithm.aspx>.

⁹Siehe <http://msdn.microsoft.com/en-us/library/system.security.cryptography.cspparameters.aspx>.

4 Implementierung

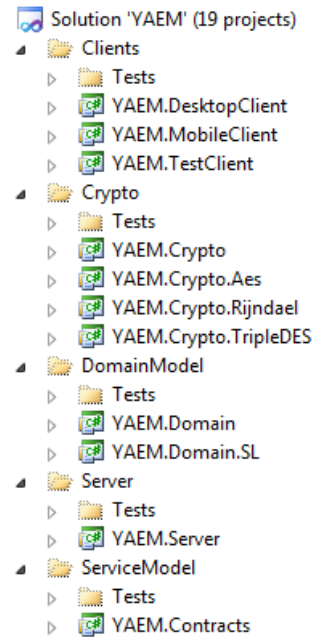


Abbildung 4.5: Visual Studio-Solution

4.4.1 YAEM.Domain

Die Assembly *YAEM.Domain* enthält alle Klassen, die über die Webservice Schnittstellen *IUserService* und *IMessagingService* übertragen werden (Inhalt des Namespaces siehe Kapitel 3.2.2 auf Seite 22). Alle Klassen sind mit *[DataContract]*¹⁰ annotiert, da sie serialisiert über den WCF-Webservice übertragen werden sollen. Die Objekte *Message*, *Session* und *User* leiten von der Klasse *ObjectBase* ab, welche neben der Schlüsselverwaltung der einzelnen Instanzen auch die Schnittstelle *INotifyPropertyChanged* implementiert, die WPF- und Silverlight-Datenbindings über geänderte Eigenschaften benachrichtigt.

Da dieser Namespace auch in Silverlight verfügbar sein soll, wird eine Kopie als Silverlight-Bibliothek erstellt, in welcher alle Klassen aus *YAEM.Domain* als Link im Projekt eingebunden sind und danach als eigenständige Assembly (hier *YAEM.Domain.SL*) kompiliert werden (siehe [Betz(2008)]).

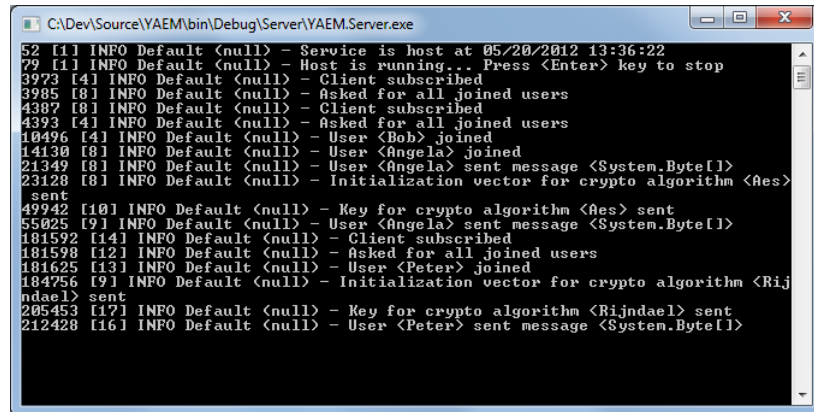
4.4.2 YAEM.Contracts

Dieses Assembly enthält nur die Service Contracts¹¹ *IUserService* und *IMessagingService*, die von der Serverapplikation implementiert als Webservice zur Verfügung gestellt werden

¹⁰Ein *DataContract* ist eine formale Vereinbarung zwischen einem Dienst und einem Client, in dem die auszutauschenden Daten abstrakt beschrieben werden. Mehr Informationen zu DataContracts sind in der MSDN Library unter <http://msdn.microsoft.com/en-us/library/ms733127.aspx> zu finden.

¹¹Ein Service Contract beschreibt eine Schnittstelle oder Klasse die einen Servicevertrag einer Applikation definiert. Siehe MSDN <http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute.aspx>.

4 Implementierung



```
C:\Dev\Source\YAEM\bin\Debug\Server\YAEM.Server.exe
52 [1] INFO Default <null> - Service is host at 05/20/2012 13:36:22
79 [1] INFO Default <null> - Host is running... Press <Enter> key to stop
3973 [4] INFO Default <null> - Client subscribed
3985 [8] INFO Default <null> - Asked for all joined users
4387 [8] INFO Default <null> - Client subscribed
4393 [4] INFO Default <null> - Asked for all joined users
10496 [4] INFO Default <null> - User <Bob> joined
14130 [8] INFO Default <null> - User <Angela> joined
21349 [8] INFO Default <null> - User <Angela> sent message <System.Byte[]>
23128 [8] INFO Default <null> - Initialization vector for crypto algorithm <Aes>
sent
49942 [10] INFO Default <null> - Key for crypto algorithm <Aes> sent
55025 [9] INFO Default <null> - User <Angela> sent message <System.Byte[]>
181592 [14] INFO Default <null> - Client subscribed
181598 [12] INFO Default <null> - Asked for all joined users
181625 [13] INFO Default <null> - User <Peter> joined
184756 [9] INFO Default <null> - Initialization vector for crypto algorithm <Rij
ndael> sent
205453 [17] INFO Default <null> - Key for crypto algorithm <Rijndael> sent
212428 [16] INFO Default <null> - User <Peter> sent message <System.Byte[]>
```

Abbildung 4.6: Serverapplikation

(Details zu den einzelnen ServiceOperations sind in Kapitel 3.2.3 auf Seite 24 zu finden).

Zusätzlich wird auch der Service Contract *IServiceCallback* in diesem Namespace definiert, der von den Clientapplikationen implementiert werden muss, wenn sie die beiden Webservices *IUserService* und *IMessagingService* nutzen wollen. Diese beiden Schnittstellen sind mit `[ServiceContract(CallbackContract = typeof(IServiceCallback))]` annotiert, was den Webservice-Konsumenten dazu zwingt, den *CallbackContract* zu implementieren.

Alle in den Schnittstellen übertragenen Datenobjekte sind entweder primitive Datentypen oder stammen aus *YAEM.Domain* (siehe Kapitel 4.4.1 auf der vorherigen Seite).

4.4.3 YAEM.Server

Die Applikation *YAEM.Server* dient als Servicehost der beiden Webservices *IUserService* und *IMessagingService*. Ein WCF-Webservice kann entweder im Internet Information Server gehostet werden oder als Selfhost-Applikation z.B. innerhalb einer Konsolenanwendung konsumierenden Anwendungen zur Verfügung gestellt werden. Da insbesondere der geringere Konfigurationsaufwand für Selfhosted-Services spricht, wurde die IIS-hosted Lösung für die Serverapplikation verworfen.

Nach dem Start der Serverapplikation zeigt die Konsolenanplikation ein Protokoll aller aufgerufenen Methoden der Webservices dar (siehe Abbildung 4.6). Wird das Konsolenfenster geschlossen, so terminiert sich *YAEM.Server* von selbst und benachrichtigt alle eventuell verbundenen Clients darüber.

4.4.4 YAEM.Crypto

Die Infrastruktur für die Implementierungen der Kryptosysteme findet sich im Namespace *YAEM.Crypto*. Das Interface *ICryptoProvider* stellt dabei alle Funktionalitäten zur Verfügung, die ein symmetrisches Kryptosystem zum Ver- und Entschlüsseln von Bytearrays benötigt.

Zusätzlich zu *ICryptoProvider* enthält das Projekt auch noch das *ExportAttribute*¹² *CryptoAlgorithmAttribute*, das von MEF benötigt wird um zur Laufzeit die installierten Kryptoalgorithmen zu identifizieren.

4.4.4.1 YAEM.Crypto.Aes

YAEM.Crypto.Aes enthält die Implementierung der Schnittstelle *ICryptoProvider* (siehe Kapitel 4.4.4 auf der vorherigen Seite) mit dem Kryptosystem Advanced Encryption Standard (AES). Der *AesCryptoProvider* ist dabei mit `[CryptoAlgorithm(Algorithm = CryptoAlgorithm.Aes)]` annotiert, so dass beim Importieren der Assembly das Kryptosystem auch im UI identifiziert werden kann.

Der *AesCryptoProvider* stellt dabei einen Wrapper für den *AesCryptoServiceProvider*¹³ aus dem Namespace *System.Security.Cryptography*¹⁴ zur Verfügung, der Microsofts Referenzimplementierung von AES darstellt. Der *AesCryptoServiceProvider* implementiert den AES über die Cryptographic Application Programming Interfaces (CAPI).

4.4.4.2 YAEM.Crypto.Rijndael

Analog der Implementierung des AES enthält der Namespace *YAEM.Crypto.Rijndael* die Implementierung des Rijndael Kryptosystems. Konkret ist der *RijndaelCryptoProvider* ein Wrapper von *RijndaelManaged*¹⁵ der den Rijndael Algorithmus implementiert. Zusätzlich ist der *RijndaelCryptoProvider* mit `[CryptoAlgorithm(Algorithm = CryptoAlgorithm.Rijndael)]` annotiert.

4.4.4.3 YAEM.Crypto.TripleDES

Der *TripleDESCryptoProvider* des *YAEM.Crypto.TripleDES* Namespaces implementiert wiederum *ICryptoProvider* und stellt einen Wrapper für den *TripleDESCryptoServiceProvider*¹⁶ aus dem Namespace *System.Security.Cryptography* bereit. Er implementiert das symmetrische Kryptosystem Triple-DES und ist mit `[CryptoAlgorithm(Algorithm = CryptoAlgorithm.TripleDES)]` annotiert.

4.4.5 YAEM.TestClient

Zu Beginn der Implementationsphase wurde der *YAEM.TestClient* entwickelt, der nicht Teil der Aufgabenstellung war. Der Testclient ist eine in Windows Forms entwickel-

¹² *ExportAttribute* spezifiziert einen Typ, ein Property oder eine Methode die einen Export zur Verfügung stellen. Mehr Informationen in der MSDN unter <http://msdn.microsoft.com/en-us/library/system.componentmodel.composition.exportattribute.aspx>.

¹³ Siehe MSDN <http://msdn.microsoft.com/en-us/library/system.security.cryptography.aescryptoserviceprovider.aspx>.

¹⁴ Sämtliche Inhalte des Namespaces *System.Security.Cryptography* finden sich in der MSDN Library unter <http://msdn.microsoft.com/en-us/library/system.security.cryptography.aspx>.

¹⁵ Siehe <http://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged>.

¹⁶ Siehe <http://msdn.microsoft.com/en-us/library/system.security.cryptography.tripledescriptoserviceprovider.aspx>.

4 Implementierung

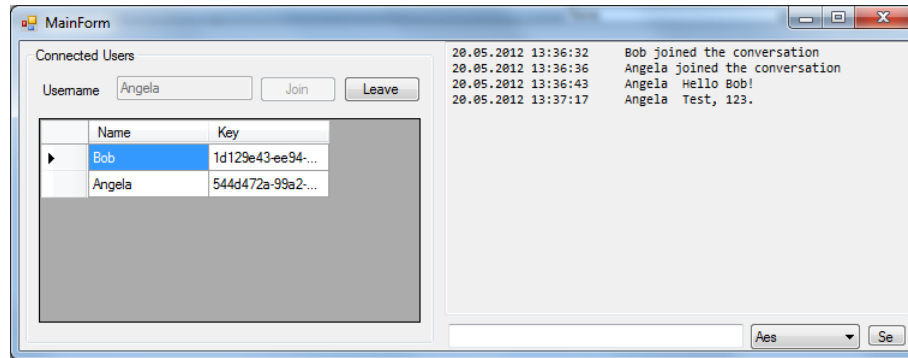


Abbildung 4.7: GesprächsdialogTestClient

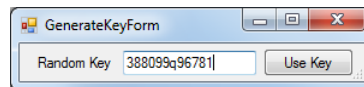


Abbildung 4.8: Schlüsselgenerierungsdialog TestClient

te Desktopapplikation für windowsbasierte Systeme. Die Applikation zeigt zusätzliche Debug-Informationen an, die in den weiteren Clientapplikationen für den Endnutzer nicht ersichtlich sind wie zum Beispiel die GUIDs der Benutzer und Nachrichten sowie weitere Informationen zu den übertragenen Daten. Der *YAEM.TestClient* hält sich somit auch nicht an die Mockups (siehe Kapitel 2.4 auf Seite 18) da er ausserhalb der definierten Anforderungen implementiert wurde.

Der Gesprächsdialog (siehe Abbildung 4.7) enthält zusätzlich zum Gesprächsverlauf auf der rechten Seite des Dialogfensters die Möglichkeit dem Gespräch beizutreten sowie eine Anzeige der dem Gespräch beigetretenen Benutzer.

Wählt der Benutzer der Applikation ein Kryptosystem aus, für das noch kein Schlüssel vorliegt, so wird der Schlüsselgenerierungsdialog (siehe Abbildung 4.8) geöffnet, der einen zufällig generierten Schlüssel enthält.

4.4.6 YAEM.DesktopClient

Der *YAEM.DesktopClient* hält sich hingegen an die Anforderungen der Mockups (siehe Kapitel 2.4 auf Seite 18) und implementiert diese in einer WPF-Applikation unter Berücksichtigung des MVVM-Entwurfsmusters (siehe Kapitel 4.2.3.1 auf Seite 39).

Die Applikation startet mit dem Anzeigen des Verbindungdialogs (siehe Abbildung 4.9) bei welchem man der Benutzer den gewünschten Benutzernamen eingeben kann. Nach Klick auf den "Join"-Button wird der Dialog geschlossen und es öffnet sich der Gesprächsdialog (siehe Abbildung 4.10).

Das *MessagingViewModel*, das an das *MessagingWindow* gebunden ist, implementiert die Schnittstelle *IServiceCallback* (siehe Kapitel 3.2.3.3 auf Seite 25), da das View-Model die Webservices der Serverapplikation benutzt. Bei erfolgreichem Beitreten des Gesprächs

4 Implementierung

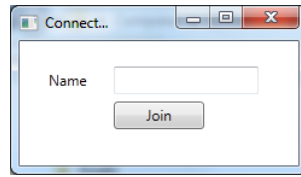


Abbildung 4.9: Verbindungsdialog DesktopClient

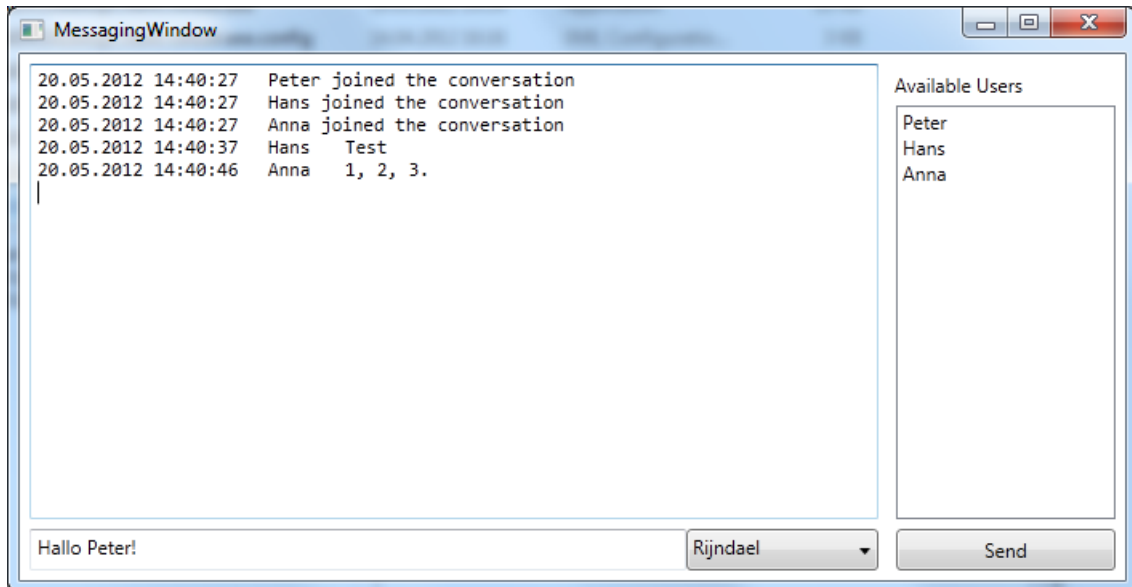


Abbildung 4.10: Gesprächsdialog DesktopClient

wird der Serverapplikation der Callbackchannel der Clientapplikation mitgeteilt, so dass diese beim Eintreffen neuer Nachrichten oder beim Hinzukommen oder Verlassen der angemeldeten Benutzer entsprechend reagieren kann.

Schliesst der Benutzer das *MessagingWindow*, so wird er vom Server abgemeldet und der Callbackchannel wird geschlossen.

Wählt der Benutzer der Applikation ein Kryptosystem aus, für das noch kein Schlüssel vorliegt, so wird der Schlüsselgenerierungsdialog (siehe Abbildung 4.11) geöffnet, der einen zufällig generierten Schlüssel enthält.

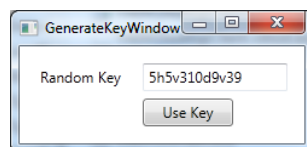


Abbildung 4.11: Schlüsselgenerierungsdialog DesktopClient

4.4.7 YAEM.MobileClient

Zur Entwicklung des *YAEM.MobileClient*, der unter Windows Phone 7 verteilt werden soll, wird Silverlight (siehe Kapitel 4.2.3.2 auf Seite 40) benutzt. Schnell hat sich gezeigt, dass Silverlight mit dem in der Konzeptphase ausgewählten *WSDualHttpBinding* nicht kompatibel ist und aufgrund der fehlenden *DuplexClientBase* (genaueres siehe Kapitel 4.4.8) es nicht möglich ist, eine funktionsfähige mobile Applikation zu entwickeln. Aus diesem Grund wird im Rahmen des Projekts auch kein Clientapplikation für Windows Phone 7 implementiert.

4.4.8 Aufgetretene Probleme

Einer der ersten Schritte beim Erstellen von Applikationen, die Webservices konsumieren, ist das Hinzufügen der Servicereferenzen auf die Webservices im Visual Studio-Projekt. Dadurch werden die im WSDL spezifizierten Datentypen als Programmcode generiert sowie das Binding auf den Webservice in der konsumierenden Applikation hinzugefügt.

Es hat sich herausgestellt, dass Silverlight keine *DuplexClientBase* im *System.ServiceModel* Namespace besitzt und es dadurch nicht möglich ist über das *WSDualHttpBinding* von Silverlight eine Kommunikation zum WCF-Webservice aufzubauen. Dadurch, dass *WSDualHttpBinding* die Grundlage zur Bereitstellung der Webservices in der Serverapplikation ist, ist es im Moment nicht möglich einen Silverlight-Client zu entwickeln¹⁷.

Möchte man trotzdem einen zweikanaligen Webservice zusammen mit WCF entwickeln, so müsste man entweder auf das *PollingDuplexHttpBinding* oder das *NetTcpBinding* ausweichen. *PollingDuplexHttpBinding* ist exklusiv für Silverlight 5 verfügbar und ermöglicht es dem konsumierenden Client in Silverlight in regelmässigen Zeitabständen beim Webservice nachzufragen, ob eine Callback-Aktion existiert, die in der konsumierenden Applikation ausgeführt werden müsste. Da *PollingDuplexHttpBinding* im Moment nicht für WPF verfügbar ist, ist auch dieses Binding für das Projekt unbrauchbar. Die einzige wirkliche Alternative stellt das *NetTcpBinding* dar, das allerdings nicht über HTTP Daten austauscht sondern über TCP. Dadurch ginge der grosse Vorteil der Interoperabilität verloren, da über TCP nur komplett serialisierte Daten übertragen würden.

¹⁷Die genauen Unterschiede zwischen dem Silverlight- und WCF-Stack finden sich in der MSDN unter [http://msdn.microsoft.com/en-us/library/cc896571\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/cc896571(VS.95).aspx).

5 Test

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

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

5.1 Unit-Tests

Unit-Tests (auch Komponententests genannt) überprüfen, ob die von Entwicklern geschriebenen Komponenten so arbeiten, wie diese es beabsichtigen. Zur Qualitätssicherung eines Softwareprodukts wird eine sehr häufige Ausführung der Unit-Tests angestrebt. Das lässt sich nur erreichen, wenn die Tests vollständig automatisiert vorliegen, sie also selbst ein Programm sind, dessen Ausführung nicht mehr Aufwand als einen Knopfdruck erfordert. Insbesondere in der testgetriebenen Entwicklung (siehe Kapitel 4.1.1 auf Seite 33) werden Unit-Tests auch als Regressionstests nach Refactoring verwendet.

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

5.1.1 Testabdeckung

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

Wie in Tabelle 5.1 ersichtlich ist, beträgt die Testabdeckung der einzelnen Namespaces jeweils mehr als 80%, die laut Aufgabenstellung gefordert sind. Die totale Testabdeckung des Projekts YAEM beträgt 94.13%.

Namespace	Codeblöcke	Getestet (Blöcke)	Getestet (% Blöcke)
<i>YAEM.DesktopClient</i>	656	612	93.29%
<i>YAEM.Crypto</i>	3	3	100%
<i>YAEM.Crypto.Aes</i>	47	47	100%
<i>YAEM.Crypto.Rijndael</i>	47	47	100%
<i>YAEM.Crypto.TripleDES</i>	47	47	100%
<i>YAEM.Domain</i>	71	61	85.91%
<i>YAEM.Server</i>	157	149	94.90%
<i>YAEM.Contracts</i>	28	28	100%

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 2.3 auf Seite 11) 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 UC4 aus den Use-Case-Spezifikationen werden als Coded UI-Tests ausformuliert und im Visual Studio-Projekt *YAEM.AcceptanceTests* abgelegt. Beim Ausführen einer der Coded UI-Tests wird automatisch die Serverapplikation sowie die Clientapplikation gestartet und der jeweilige Testfall wird als Skript abgearbeitet. Je nach Use-Case sind unterschiedliche Eingaben sowie Erwartungen definiert. Werden diese nicht erfüllt (z.B. wird nach dem Senden einer Nachricht beim Empfänger keine neue Nachricht angezeigt) so wird schlägt der Test fehl.

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

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>.

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

Tabelle 5.2: Akzeptantests

6 Fazit

Die Semesterarbeit stellt ein spannendes und lehrreiches Projekt dar, das insbesondere als Vorbereitung für die Bachelorarbeit hilfreich ist. Das sorgfältige Erfassen der Aufgabenstellung und der zu erwartenden Resultate stellen dabei ein zentrales Element der Projektarbeit dar und es ist von Vorteil genau abzuwägen, welche Punkte in der Aufgabenstellung erfasst sein soll und welche Resultate am Ende des Projekts in welcher Qualität abgeliefert werden soll.

Die Verwendung des Wasserfallmodells hat sich für dieses Projekt als ideal herausgestellt, da so nach jeder Phase die abzugebenden Dokumente und Resultate fertiggestellt vorlagen und erst danach zur nächsten Phase vorgegangen werden konnte. Auch hat es sich bezahlt gemacht, relativ viel Zeit in die Erarbeitung des Konzepts zu investieren, da dadurch bei Erreichen der Implementierungsphase direkt mit der Entwicklung der einzelnen Komponenten begonnen werden konnte und zwischen den Komponenten keine Inkonsistenzen entstanden sind die erst in der Verifikationsphase bemerkt wurden.

Gegenüber der Aufgabenstellung fehlen in der Implementierung beziehungsweise der Dokumentation der Semesterarbeit die Clientapplikation für Windows Phone 7 in Silverlight (Gründe siehe dazu Kapitel 4.4.8 auf Seite 49) sowie der Vergleich der Funktionen und Performance mit Konkurrenzprodukten. Da YAEM sich noch im Prototypenstadium befindet und die verlangten 120 Stunden Aufwand schon überschritten worden sind, wurde darauf verzichtet. Zudem wurden anstatt Caesar und Blowfish die Kryptosysteme AES und Rijndael implementiert, wie im Kapitel Implementierung ausführlich beschrieben wurde (siehe Kapitel 4.3 auf Seite 41).

6.1 Aufwand Seminararbeit

In Tabelle 6.1 wird der geplante Aufwand mit dem (ungefähren) effektiven Aufwand verglichen um die ursprünglichen Schätzungen zu überprüfen. Laut Reglement der Seminararbeit (siehe [Stern(2010)]) sollte der Aufwand der Seminararbeit bei mindestens 120 Stunden pro Student liegen. Diese 120 Stunden wurden auf die sechs Phasen verteilt und während der Erarbeitung der Semesterarbeit wurden die effektiv investierten Stunden pro Phase gemessen.

Wie sich herausgestellt hat, wurde die Planung der einzelnen Phasen zu optimistisch erstellt. Alleine schon für das Erarbeiten des Konzepts wurde mehr als fünf Mal so viel Zeit gebraucht wie geplant und total wurden die geplanten 120 Stunden um den Faktor 2.8 überschritten. Dieser Soll/Ist-Vergleich des investierten Aufwands zeigt insbesondere auf, in welchen Phasen (zu) viel Zeit investiert wurde, namentlich in der Design- und Implementierungsphase.

Phase	Geplante Stunden	Effektive Stunden
Themenevaluation	10	8
Erfassen der Anforderungen	20	26
Erarbeiten des Konzepts	20	103
Implementierungs des Konzepts	40	153
Überprüfen und Test des Konzepts	20	48
Erarbeiten der Präsentation	10	in Arbeit

Table 6.1: Gegenüberstellung geplanter/effektiver Aufwand

6.2 Schlussfolgerungen

Dieses Unterkapitel soll die Lehren aus der Semesterarbeit im Hinblick auf die Bachelorarbeit ziehen.

Die wichtigste Lehre ist, dass eine realistische Aufwandsschätzung von Beginn der Arbeit zwingend notwendig ist. Das Verhältnis zwischen geplantem und effektivem Aufwand, das in dieser Semesterarbeit aus dem Ruder gelaufen ist, muss insbesondere in der Bachelorarbeit realistisch sein. Bereits zum Zeitpunkt des Erfassens der Aufgabenstellung sollte eine grobe Vorstellung des Aufwands der einzelnen Punkte sowie der zu erwartenden Resultate vorhanden sein.

Auch wurde zuviel Zeit verwendet um allfällige Alternativen für Silverlight und *WSDualHttpBinding* zu finden, eine technische Schwierigkeit von welcher der Erfolg dieser Arbeit nicht abhängt, da die mobile Clientapplikation nur eine von mehreren Applikationen darstellt.

Es ist auch fragwürdig ob soviel Zeit und Aufwand in die Erklärung verschiedener Technologien im Kapitel Implementierung (siehe 4.2 auf Seite 33) verwendet werden soll oder ob mehr Verweise auf Fachliteratur gemacht werden sollen. Der Aufwand für die Erstellung solch theoretischer Grundlagen soll in der Bachelorarbeit von Anfang an eingeplant und in der Aufgabenstellung bzw. den erwarteten Resultate festgehalten werden.

Im Allgemeinen hat mir diese Semesterarbeit viel Spass gemacht und ich habe Herzblut hineingesteckt. Im Hinblick auf die Bachelorarbeit konnte ich viel profitieren und hoffe, nicht mehr in dieselben Fallen zu treten, in die ich im Verlauf dieser Semesterarbeit getreten bin.

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

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.
Black-Box-Test	Ein Black-Box-Test bezeichnet einen Test ohne Kenntnisse über die innere Funktionsweise des zu testenden Systems. Er beschränkt sich auf funktionsorientiertes Testen, das heisst für die Ermittlung der Testfälle werden nur die Anforderungen, aber nicht die Implementierung des Testobjekts herangezogen. Die genaue Beschaffenheit des Programms wird nicht betrachtet, sondern vielmehr als Black Box behandelt. Nur nach aussen sichtbares Verhalten fliesst in den Test ein.
Bug	Ein Programm- oder Softwarefehler wird als Bug bezeichnet und beschreibt im Allgemeinen ein fehlerhaftes Verhalten von Computerprogrammen.
Bytearray	Ein Bytearray ist eine Datenstruktur die als Container für Byte-Werte (ganzzahlige Werte von 0 bis 255) dient.
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.
DCOM	Das Distributed Component Object Model (kurz DCOM) ist ein objektorientiertes RPC-System, das von Microsoft entwickelt wurde, damit man mit COM die Möglichkeit hat, über das Netzwerk zu kommunizieren.

6 Fazit

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.
Endpoint	Ein Endpoint (dt. Endpunkt) kennzeichnet einen verfügbaren Dienst. Er besteht aus einer Adresse, einem Binding und einem Contract.
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 Quellcodeverwaltung des Linux-Kernels entwickelt wurde.
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 dient.
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.
Internet Information Server	Der Internet Information Server (IIS) ist eine Serviceplattform über welche Dokumente und Dateien im Netzwerk zugänglich gemacht werden. Als Kommunikationsprotokolle kommen hierbei HTTP, HTTPS, FTP, SMTP, POP3, WebDAV und andere zum Einsatz. Über IIS können ASP- oder .NET-Applikationen (ASP.NET) ausgeführt werden und WCF Services gehostet werden.
Managed Code	Programmcode, der im .NET Framework unter der .NET-Laufzeitumgebung abläuft, wird als Managed Code bezeichnet. Im Gegensatz dazu wird herkömmlicher Code als Unmanaged Code oder Classic Code bezeichnet.

MEF	Das Managed Extensibility Framework (kurz MEF) ist ein Kompositionsframework 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. Mock-ups 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.
MSMQ	MSMQ (oder Microsoft Message Queueing) ist die Implementierung einer Message Queue die von Microsoft entwickelt wurde und seit Windows NT 4 und Windows 95 Teil der Windowsplattform ist. MSMQ ist ein Nachrichtenprotokoll, das es Applikationen erlaubt mit verschiedenen Servern bzw. Prozessen fehlergesichert zu kommunizieren.
Reflection	Reflection (zu deutsch Reflexion oder Introspektion) ermöglicht einem Programm seine eigene Struktur zu kennen und ermöglicht es diesem, sie gegebenenfalls zu modifizieren.
Repository	Ein Repository ist ein Verzeichnis zur Speicherung von digitalen Objekten. In diesem Kontext ist ein Repository eine Datenbank zur Speicherung und zum Wiederfinden von Objektinstanzen.
RUP	Der Rational Unified Process ist ein kommerzielles Vorgehensmodell zur Softwareentwicklung von IBM.
Schlüssel	Als Schlüssel wird in der Kryptologie allgemein eine Information bezeichnet, die einen kryptographischen Algorithmus parametrisiert.
Server	Ein Server (auch Serverapplikation) ist eine serverseitige Anwendung die auf einem zentralen Computer (Server) ausgeführt wird.
Service Contract	Ein Service Contract bezeichnet eine Schnittstelle oder Klasse die zur Kommunikation für verteilte Systeme genutzt werden kann.
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.

Servicehost	Der Servicehost implementiert den Host, der vom Programmiermodell für das Windows Communication Foundation (WCF)-Dienstmodell verwendet wird.
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.
SOAP	SOAP (ursprünglich Simple Object Access Protocol) ist ein vom W3C standardisiertes Nachrichtenprotokoll. Über SOAP wird eine Kommunikation zwischen verteilten Anwendungen ermöglicht und standardisiert und Applikationen werden webfähig.
Stakeholder	Als Stakeholder (engl.) wird eine Person oder Gruppe bezeichnet, die ein berechtigtes Interesse am Verlauf oder Ergebnis eines Prozesses oder Projektes hat.
URI	Ein Uniform Resource Identifier (kurz URI) ist ein Identifikator und besteht aus einer Zeichenfolge, die zur Identifizierung einer abstrakten oder physischen Ressource dient.
Use-Case	Ein Use-Case (dt. 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.
Wasserfallmodell	Das Wasserfallmodell ist ein lineares (nicht iteratives) Vorgehensmodell, das in Phasen organisiert wird. Dabei gehen die Phasenergebnisse wie bei einem Wasserfall immer als bindende Vorgaben für die nächsttiefere Phase ein. Im Wasserfallmodell hat jede Phase vordefinierte Start- und Endpunkte mit eindeutig definierten Ergebnissen.

Webservice	Ein Webservice ist eine Softwareapplikation, auf den über einen URI 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.
WSDL	Die Web Services Description Language (WSDL) ist eine plattform-, programmiersprachen- und protokollunabhängige Beschreibungssprache für Netzwerkdienste (Webservices) zum Austausch von Nachrichten auf Basis von XML.
XAML	Extensible Application Markup Language (XAML) ist eine von Microsoft entwickelte allgemeine Beschreibungssprache für die Oberflächengestaltung von Anwendungen sowie zur Definition von Workflows in WF.

Abbildungsverzeichnis

1.1	Wasserfallprozess nach [VO(2007)]	6
2.1	Systemkontext	11
2.2	Use-Case Gespräch beitreten	12
2.3	Use-Case Gespräch verlassen	14
2.4	Use-Case Nachricht senden	15
2.5	Use-Case Nachricht empfangen	17
2.6	Mockup Dialog Gespräch beitreten	19
2.7	Mockup Gesprächsdialog	20
3.1	Komponentendiagramm	22
3.2	Klassendiagramm Domänenmodell	23
3.3	Klassendiagramm <i>IUserService</i>	24
3.4	Klassendiagramm <i>IMessagingService</i>	25
3.5	Klassendiagramm <i>IServiceCallback</i>	26
3.6	Klassendiagramm Kryptoalgorithmen	27
3.7	Klassendiagramm Server	27
3.8	Sequenzdiagramm Gespräch beitreten	29
3.9	Sequenzdiagramm Gespräch verlassen	29
3.10	Sequenzdiagramm Nachricht senden	30
3.11	Verteilungsdiagramm	31
4.1	Schritte in TDD (aus [Ambler(2011)])	34
4.2	MEF Struktur	36
4.3	Hauptfunktionalitäten von WPF (aus[Moser(2012)])	39
4.4	MVVM-Beispielimplementierung (aus [Smith(2009)])	40
4.5	Visual Studio-Solution	44
4.6	Serverapplikation	45
4.7	GesprächsdialogTestClient	47
4.8	Schlüsselgenerierungsdialog TestClient	47
4.9	Verbindungsdialog DesktopClient	48
4.10	Gesprächsdialog DesktopClient	48
4.11	Schlüsselgenerierungsdialog DesktopClient	48

Tabellenverzeichnis

1.1	Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument	6
1.2	Phasenplan	7
1.3	Meilensteine	8
2.1	Use-Case-Spezifikation Gespräch beitreten	13
2.2	Use-Case-Spezifikation Gespräch verlassen	14
2.3	Use-Case-Spezifikation Nachricht senden	16
2.4	Use-Case-Spezifikation Nachricht empfangen	18
3.1	Methoden von <i>IServiceCallback</i>	25
4.1	Bindings	37
5.1	Testabdeckung	51
5.2	Akzeptantests	52
6.1	Gegenüberstellung geplanter/effektiver Aufwand	54

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.
- [Betz(2008)] David Betz. Reusing .net assemblies in silverlight, 12 2008. URL <http://www.netfxharmonics.com/2008/12/Reusing-NET-Assemblies-in-Silverlight>.
- [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 development, 2012. URL <http://martinfowler.com/bliki/TestDrivenDevelopment.html>.
- [Fowler(2004)] Martin Fowler. Presentation model, 2004. URL <http://martinfowler.com/eaDev/PresentationModel.html>.
- [Freeman and Pryce(2009)] Steve Freeman and Nat Pryce. *Growing Object-Oriented Software, Guided by Tests*. Addison-Wesley Professional, 2009.
- [Geiselmann(2009)] Dr. W. Geiselmann. *Datensicherheitstechnik*. 2009. URL <http://www.iks.kit.edu/fileadmin/User/scc2Uebungen/scc2.pdf>.
- [Hruschka and Starke(2012)] Peter Hruschka and Gernot Starke. arc42 - resourcen für software-architekten, 2012. URL <http://www.arc42.de>.
- [Hunt and Thomas(1999)] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Longman, 1999.
- [Moser(2012)] Christian Moser. Wpf tutorial.net, 2012. URL <http://www.wpftutorial.net>.

- [Nachiappan Nagappan and Williams(2008)] Thirumalesh Bhat Nachiappan Nagappan, E. Michael Maximilien and Laurie Williams. Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering*, 13:289–302, 2008.
- [Pohl and Rupp(2011)] Klaus Pohl and Chris Rupp. *Basiswissen Requirements Engineering*. dpunkt.verlag, 2011.
- [Shawnfa(2006)] Shawnfa. The differences between rijndael and aes, 2006. URL <http://blogs.msdn.com/b/shawnfa/archive/2006/10/09/the-differences-between-rijndael-and-aes.aspx>.
- [Smith(2009)] Josh Smith. Wpf apps with the model-view-viewmodel design pattern. *MSDN Magazine*, 2009. URL <http://msdn.microsoft.com/en-gb/magazine/dd419663.aspx>.
- [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-Semesterarbeit_Studiengang-Informatik-der-HSZ-T_V3.8e.pdf.
- [VO(2007)] Trung Hung VO. Software development process, 07 2007. URL <http://cnx.org/content/m14619/1.2/>.