

# Yet Another Encrypted Messenger

Florian Amstutz <florian@amstutz.nu>

02. April 2012

Semesterarbeit an der Zürcher Hochschule für Angewandte  
Wissenschaften

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>4</b>
1.1	Management Summary . . . . .	4
1.2	Über die Semesterarbeit . . . . .	4
1.3	Softwareentwicklungsprozess . . . . .	4
<b>2</b>	<b>Anforderungen</b>	<b>7</b>
2.1	Was sind Anforderungen? . . . . .	7
2.1.1	Arten von Anforderungen . . . . .	7
2.2	Systemkontext . . . . .	8
2.3	Use-Case-Spezifikationen . . . . .	9
2.3.1	Gespräch beitreten . . . . .	10
2.3.2	Gespräch verlassen . . . . .	11
2.3.3	Nachricht senden . . . . .	12
2.3.4	Nachricht empfangen . . . . .	14
2.4	Mockups . . . . .	16
2.4.1	Dialog Gespräch beitreten . . . . .	17
2.4.2	Gesprächsdialog . . . . .	17
<b>3</b>	<b>Konzept</b>	<b>19</b>
3.1	Konzept . . . . .	20
3.1.1	Bausteinsicht . . . . .	20
3.1.1.1	Komponentendiagramm . . . . .	20
3.1.1.2	Domänenmodell . . . . .	21
3.1.1.3	Service Contracts . . . . .	22
3.1.1.4	Kryptoalgorithmen . . . . .	23
3.1.1.5	Server . . . . .	24
3.1.2	Laufzeitsicht . . . . .	25
3.1.2.1	Gespräch beitreten . . . . .	25
3.1.2.2	Gespräch verlassen . . . . .	26
3.1.2.3	Nachricht senden . . . . .	27
3.1.3	Verteilungssicht . . . . .	28
<b>4</b>	<b>Implementierung</b>	<b>29</b>
4.1	Implementierung . . . . .	29
<b>5</b>	<b>Test</b>	<b>30</b>
5.1	Test . . . . .	30

## *Inhaltsverzeichnis*

<b>6</b>	<b>Anhang</b>	<b>31</b>
6.1	Akronyme . . . . .	31
6.2	Glossar . . . . .	31
6.3	Bibliographie . . . . .	32
	<b>Literaturverzeichnis</b>	<b>32</b>

# 1 Einführung

Als einführendes Kapitel dieses Dokuments wird die Semesterarbeit als Projekt kurz vorgestellt und es werden die Rahmenbedingungen der Semesterarbeit zusammengefasst niedergeschrieben. Weiter werden verschiedene Softwareentwicklungsprozesse vorgestellt sowie die für diese Semesterarbeit verwendete Methode erklärt.

## 1.1 Management Summary

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

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

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

## 1.2 Über die Semesterarbeit

Gemäss Reglement der ZHAW (siehe [6]) dient die Semesterarbeit als Vorbereitung zur Bachelorarbeit. Sie besteht aus einem konzeptionellen Teil und einem Umsetzungsteil, wobei der Schwerpunkt auf der Umsetzung liegen soll.

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

## 1.3 Softwareentwicklungsprozess

Software lässt sich nach einer Vielzahl von Prozessen und Modellen entwickeln. Von iterativen Vorgehen wie Scrum über komplexe Modelle wie RUP hin zu klassischen, linearen Vorgehen wie dem Wasserfallmodell oder dem V-Modell. Nach [5] ist die Auswahl des Entwicklungsprozesses eine der schwierigsten Entscheidungen, die man bei einem Softwareprojekt treffen muss. Häufig besitzen Unternehmungen bereits etablierte, auf sie

## 1 Einführung

zugeschnittene Entwicklungsmodelle, die mehr oder weniger gut zur Organisation der Unternehmung passen. Ein ungünstig gewählter oder nicht vollständig eingeführter und gelebter Entwicklungsprozess ist nach [5] einer der Hauptgründe wieso Softwareprojekte mit Qualitätsmängeln, Budgetüberschreitungen oder zeitlichen Verzögerungen zu kämpfen haben.

Für dieses Projekt wurde das Wasserfallmodell als Entwicklungsprozess ausgewählt. Das Wasserfallmodell teilt die Softwareentwicklung in meist fünf verschiedene Phasen auf. Dabei kann erst mit der nächsten Phase begonnen werden wenn die Lieferegebnisse und die Ergebnisdokumentation der vorhergehenden Phase fertiggestellt und abgenommen worden sind. Das Wasserfallmodell wurde ausgewählt, da die jeweiligen Phasen eindeutig abgeschlossen werden können, da der Betreuer als einziger, externer Stakeholder des Projekts zu festdefinierten Phasen Einfluss auf das Projekt ausübt und danach keine Möglichkeit mehr besitzt, den Projektverlauf zu beeinflussen. Die grössten Nachteile des Wasserfallmodells sind nach [3] ein Abgrenzungsproblem zwischen den Phasen sowie die Schwierigkeit des Abschlusses einzelner Phasen. Dadurch dass der Betreuer nur in den Phasen Requirements und Design Einfluss auf das Projekt nehmen kann und Student als einziger Stakeholder den Abschluss der Phasen abnimmt sowie den Ablauf der Phasen innerhalb des Projekts steuert können diese Nachteile umgangen werden.

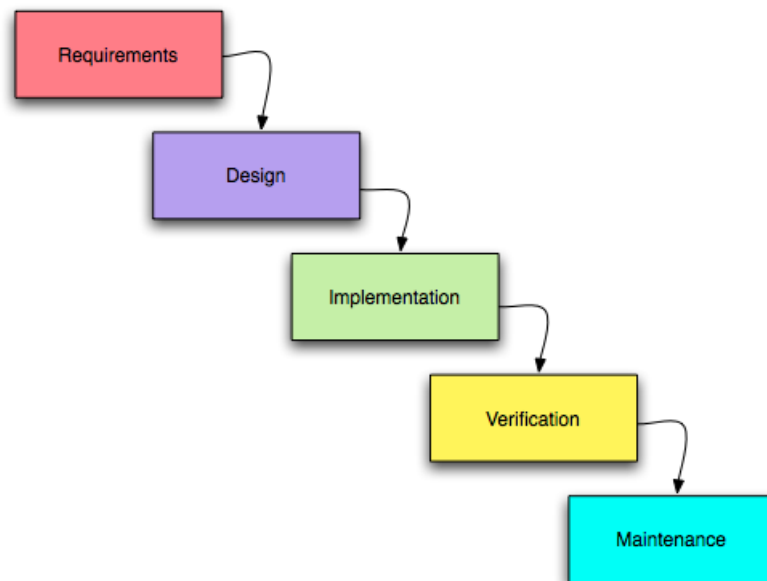


Abbildung 1.1: Wasserfallprozess nach [7]

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

## 1 Einführung

schiedenen Ebenen von Softwarearchitekten designed und eine Blaupause wird erstellt, nach welcher sich die Software Entwickler in der Implementationsphase zu halten haben. Das Design sollte einen Plan beinhalten, welcher die Implementierung der Anforderungen aufzeigt. Wenn das Design fertiggestellt worden ist, wird dieses von den Entwicklern in Programmcode umgesetzt. Gegen Ende der Implementationsphase werden die Softwarekomponenten verschiedener Teams integriert. Nachdem die Implementierungs- und Integrationsphasen abgeschlossen sind, wird das Softwareprodukt getestet und allfällige Fehler aus früheren Phasen werden zu diesem Zeitpunkt entfernt. Danach wird das Softwareprodukt installiert und später in der Wartungsphase (Maintenance) um weitere Funktionalitäten erweitert beziehungsweise werden weitere Bugs behoben.

Die Struktur dieses Dokuments hält sich auch an den Wasserfallprozess nach [7] (siehe Tabelle 1.1). Die Phase Maintenance wird dabei ausgelassen, da sich die innerhalb des Projekts entwickelte Applikation nach Abschluss der Verifizierungsphase noch im Prototypenstand befinden wird.

Phase	Kapitelüberschrift	Seite
Requirements	Anforderungen	7
Design	Konzept	19
Implementation	Implementierung	29
Verification	Test	30

Tabelle 1.1: Zuweisungstabelle der Phasen zu Kapiteln in diesem Dokument

## 2 Anforderungen

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

### 2.1 Was sind Anforderungen?

Die erste Phase des Wasserfallmodells beschäftigt sich mit den Anforderungen an das zu entwickelnde Softwareprodukt. Damit das Entwicklungsprodukt zum Erfolg geführt werden kann, muss zunächst bekannt sein, was die Anforderungen an das System sind und diese müssen geeignet dokumentiert sein. Nach [1] wird eine Anforderung wie folgt definiert:

**Anforderung** Eine Anforderung ist:

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

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

#### 2.1.1 Arten von Anforderungen

Nach [4] unterscheidet man im Allgemeinen zwischen drei Arten von Anforderungen:

- Funktionale Anforderungen legen die Funktionalität fest, die das geplante System zur Verfügung stellen soll. Sie werden typischerweise in Funktions-, Verhaltens- und Strukturanforderungen unterteilt.

**Funktionale Anforderung** Eine funktionale Anforderung ist eine Anforderung bezüglich des Ergebnisses eines Verhaltens, das von einer Funktion des Systems bereitgestellt werden soll.

- 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.

**Qualitätsanforderung** Eine Qualitätsanforderung ist eine Anforderung, die sich auf ein Qualitätsmerkmal bezieht, das nicht durch funktionale Anforderungen abgedeckt ist.

- Randbedingungen (auch: Rahmenbedingungen) können von den Projektbeteiligten nicht beeinflusst werden. Randbedingungen können sich sowohl auf das betrachtete System beziehen (z.B. “Das System soll über Webservices mit Aussensysteme kommunizieren”) als auch auf den Entwicklungsprozess des Systems (z.B. “Das System soll bis spätestens Mitte 2013 am Markt verfügbar sein”). Randbedingungen werden, im Gegensatz zu funktionalen Anforderungen und Qualitätsanforderungen, nicht umgesetzt, sondern schränken die Umsetzungsmöglichkeiten, d.h. den Lösungsraum im Entwicklungsprozess ein.

**Randbedingung** Eine Randbedingung ist eine Anforderung, die den Lösungsraum jenseits dessen einschränkt, was notwendig ist, um die funktionalen Anforderungen und die Qualitätsanforderungen zu erfüllen.

## 2.2 Systemkontext

Als erster Schritt in der Erhebung und Dokumentierung wird der Systemkontext ermittelt. Es wird eine Sollperspektive eingenommen, d.h., es wird eine Annahme getroffen, wie das geplante System sich in die Realität integriert. Hierdurch wird der Realitätsausschnitt identifiziert, der das System und damit potenziell auch dessen Anforderungen beeinflusst. Um die Anforderungen an das geplante System korrekt und vollständig spezifizieren zu können, ist es notwendig, die Beziehung zwischen den einzelnen materiellen und immateriellen Aspekten im Systemkontext und dem geplanten System exakt zu definieren. Der für die Anforderungen des Systems relevante Ausschnitt der Realität wird als Systemkontext bezeichnet.

**Systemkontext** Der Systemkontext ist der Teil der Umgebung eines Systems, der für die Definition und das Verständnis der Anforderungen des betrachteten Systems relevant ist (nach [4]).



## 2 Anforderungen

Der Ursprung der Anforderungen des Systems liegt im Systemkontext des geplanten Systems. Aus diesem Grund wird der Systemkontext vor Erhebung und Dokumentierung der Anforderungen festgelegt. Der Systemkontext YAEM wird wie folgt dargestellt. Die Benutzer als Stakeholder an das System senden und empfangen Nachrichten und befinden sich innerhalb des Systemkontexts da sie direkt mit dem System interagieren. Die Verschlüsselungsalgorithmen sind in der Fachliteratur klar geregelt und normiert und geben aus diesem Grund die konkreten Implementierungsvorschriften an das System vor.

### Systemkontext YAEM

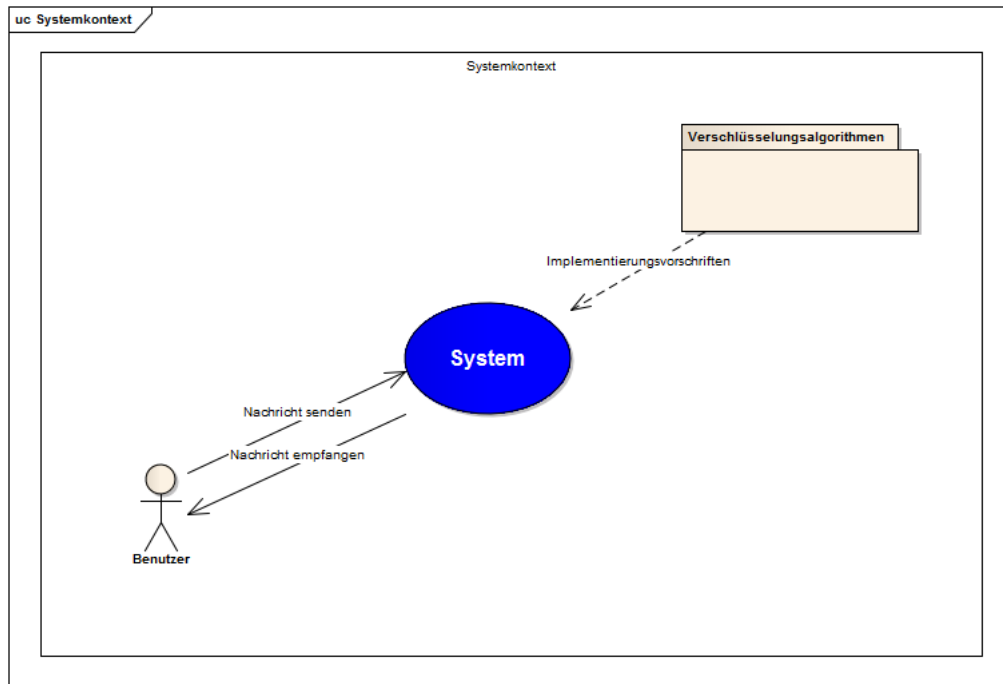


Abbildung 2.1: Systemkontext YAEM

## 2.3 Use-Case-Spezifikationen

Nach [4] zeigen Use-Case-Diagramme die aus einer externen Nutzungssicht wesentlichen Funktionalitäten des betrachteten Systems sowie spezifische Beziehungen der einzelnen Funktionalitäten untereinander bzw. zu Aspekten in der Umgebung des Systems. Abgesehen vom Namen eines Use-Cases und dessen Beziehungen dokumentieren Use-Case-Diagramme allerdings keinerlei weitere Informationen über die einzelnen Use-Cases, wie z.B. die Systematik der Interaktion eines Use Case mit Akteuren in der Umgebung. Diese Informationen werden unter Verwendung einer geeigneten Schablone zusätzlich zum Use-Case-Diagramm textuell dokumentiert.

Alle funktionalen Anforderungen (siehe 2.1.1 auf Seite 7) werden nun als Use-Cases

## 2 Anforderungen

modelliert und spezifiziert<sup>1</sup>. Als Quellen der Anforderungen dienten der Betreuer, die Reglemente der ZHAW betreffend der Semesterarbeit sowie der Student in der Rolle als Benutzer des Systems. Zusätzlich zu den Use-Cases und der dazugehörigen Use-Case-Spezifikation wird vorgängig in Prosatext der Anwendungsfall beschrieben. Aus Gründen der Übersichtlichkeit und des überschaubaren Umfangs dienen diese Use-Cases primär als Anforderungen an das zu entwickelnde Softwaresystem. Jedes Use-Case wird im Rahmen der Verifizierungsphase (siehe 5 auf Seite 30) als Integrationstest einzeln getestet.

### 2.3.1 Gespräch beitreten

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

#### Use-Case

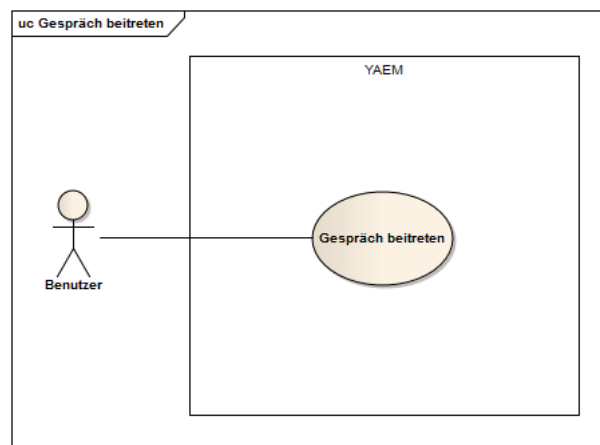


Abbildung 2.2: Use-Case “Gespräch beitreten”

---

<sup>1</sup>Die verwendete Schablone für die Use-Case-Spezifikationen stammt aus [Pohl2011] und dient zur zweckmässigen Strukturierung von Typen von Informationen, die einen Use-Case betreffen. Die Abschnitte Autor, Quelle, Verantwortlicher und Qualität werden ausgelassen, da sie für die Semesterarbeit keine Relevant besitzen.

### Use-Case-Spezifikation

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

Tabelle 2.1: Use-Case-Spezifikation “Gespräch beitreten”

### 2.3.2 Gespräch verlassen

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

## 2 Anforderungen

### Use-Case

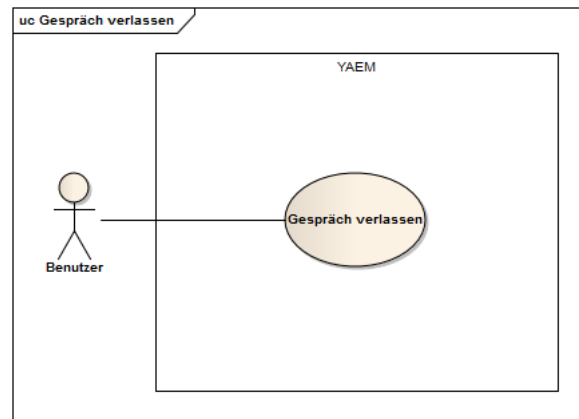


Abbildung 2.3: Use-Case “Gespräch verlassen”

### Use-Case-Spezifikation

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

Tabelle 2.2: Use-Case-Spezifikation “Gespräch verlassen”

#### 2.3.3 Nachricht senden

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

## 2 Anforderungen

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

### Use-Case

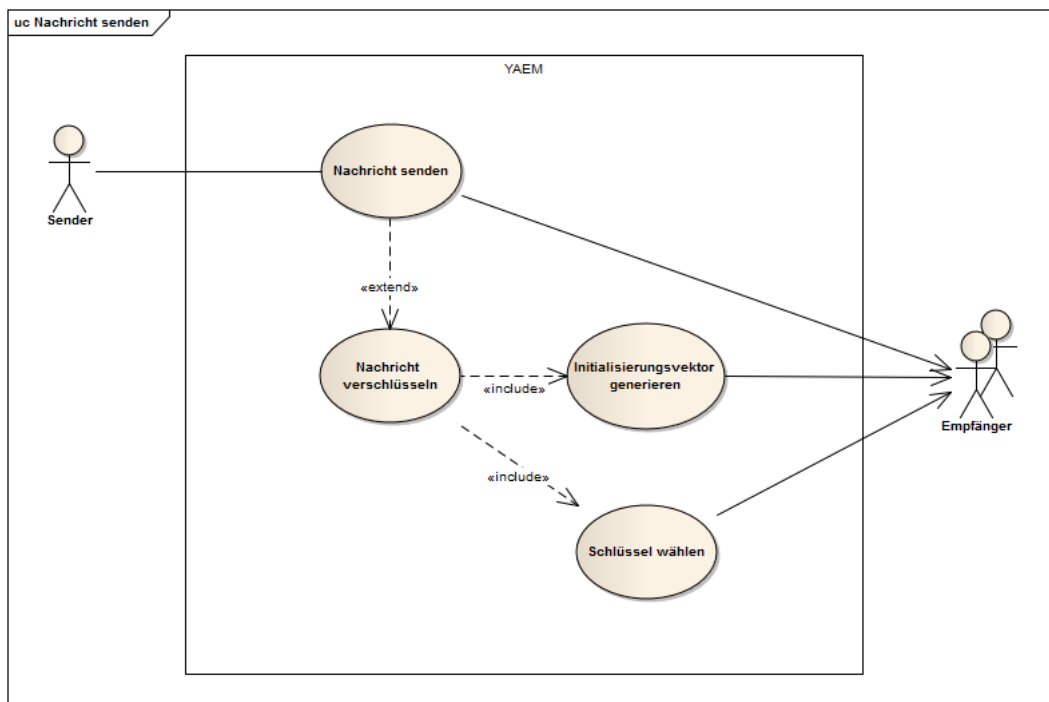


Abbildung 2.4: Use-Case “Nachricht senden”

## Use-Case-Spezifikation

Abschnitt	Inhalt
Bezeichner	UC3
Name	Nachricht senden
Priorität	Wichtigkeit für Systemerfolg: hoch Technologisches Risiko: mittel
Kritikalität	Hoch
Beschreibung	Der Benutzer versendet eine Nachricht.
Auslösendes Ereignis	Benutzer möchte eine Nachricht senden.
Akteure	Benutzer
Vorbedingung	Der Benutzer ist im Gespräch angemeldet und besitzt eine gültiges Session-Ticket.
Nachbedingung	Der Benutzer kann erneut eine Nachricht versenden und Nachrichten anderer Gesprächsteilnehmer empfangen.
Ergebnis	Die Empfänger haben die versendete Nachricht empfangen.
Hauptszenario	<ol style="list-style-type: none"> <li>1. Der Benutzer erfasst die zu versenden Nachricht</li> <li>2. Der Benutzer wählt einen Kryptoalgorithmus aus.</li> <li>3. Der Benutzer generiert einen Initialisierungsvektor.</li> <li>4. Der Initialisierungsvektor wird an alle Empfänger gesendet.</li> <li>5. Der Benutzer wählt einen Schlüssel.</li> <li>6. Der Schlüssel wird an alle Empfänger gesendet.</li> <li>7. Der Benutzer verschickt die (verschlüsselte) Nachricht.</li> </ol>
Alternativszenarien	<ol style="list-style-type: none"> <li>2a. Der Benutzer wählt keinen Kryptoalgorithmus aus.</li> <li>2a1. Der Benutzer versendet die Nachricht unverschlüsselt.</li> <li>3a. Der Benutzer hat bereits einen Initialisierungsvektor erstellt oder einen Initialisierungsvektor von einem anderen Teilnehmer des Gesprächs erhalten und generiert keinen neuen Initialisierungsvektor.</li> <li>4a. Der Benutzer hat bereits einen Schlüssel erstellt oder einen Schlüssel von einem anderen Teilnehmer des Gesprächs erhalten und wählt keinen neuen Schlüssel.</li> </ol>
Ausnahmeszenarien	Auslösendes Ereignis: Der Benutzer kann keine Verbindung zum Server herstellen.

Tabelle 2.3: Use-Case-Spezifikation “Nachricht senden”

### 2.3.4 Nachricht empfangen

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

## 2 Anforderungen

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

### Use-Case

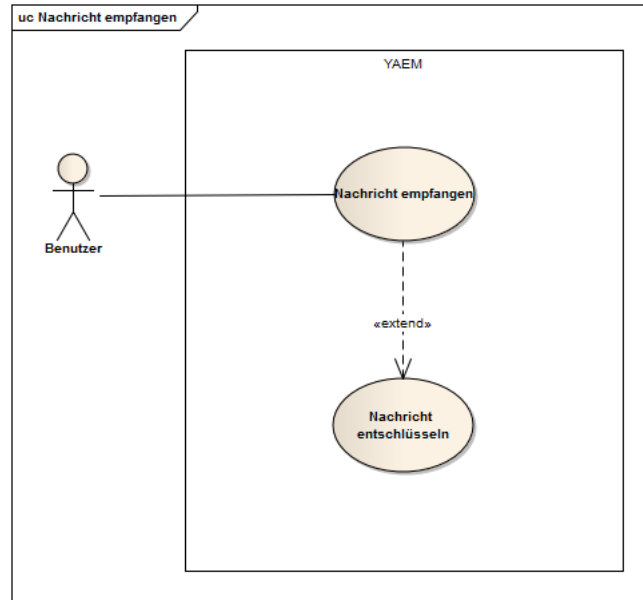


Abbildung 2.5: Use-Case "Nachricht empfangen"

## 2 Anforderungen

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

Tabelle 2.4: Use-Case-Spezifikation “Nachricht empfangen”

### Use-Case-Spezifikation

## 2.4 Mockups

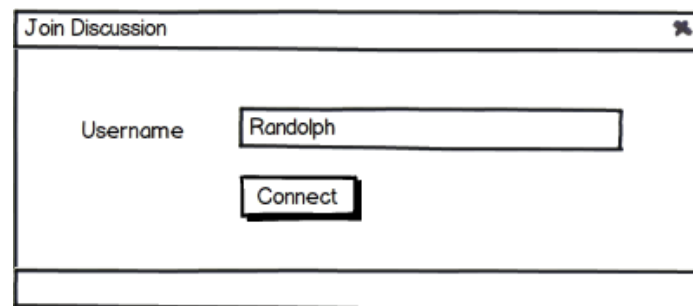
Mockups dienen zur Visualisierung der Benutzeroberfläche des zu entwickelnden Softwareprodukts und werden häufig bereits in der Anforderungsphase zusammen mit den Stakeholdern entwickelt. Sie liefern dem Softwareentwickler in der Implementationsphase (siehe 4 auf Seite 29) ein Grundgerüst für die einzelnen Dialogfenster. Je nach Kundenzielgruppe und Wichtigkeit der Mensch-Maschine-Schnittstelle wird mehr oder weniger Zeit in die Entwicklung der Mockups investiert. Häufig werden diese mit zusammen mit Psychologen entwickelt und enthalten sehr weniger Interpretationsspielraum für den Softwareentwickler.



### 2.4.1 Dialog Gespräch beitreten

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

#### Mockup



The mockup shows a window titled "Join Discussion" with a close button in the top right corner. Inside the window, there is a label "Username" followed by a text input field containing the text "Randolph". Below the input field is a button labeled "Connect". At the bottom of the window, there is a horizontal line.

Abbildung 2.6: Mockup "Dialog Gespräch beitreten"

### 2.4.2 Gesprächsdialog

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

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

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

## 2 Anforderungen

### Mockup

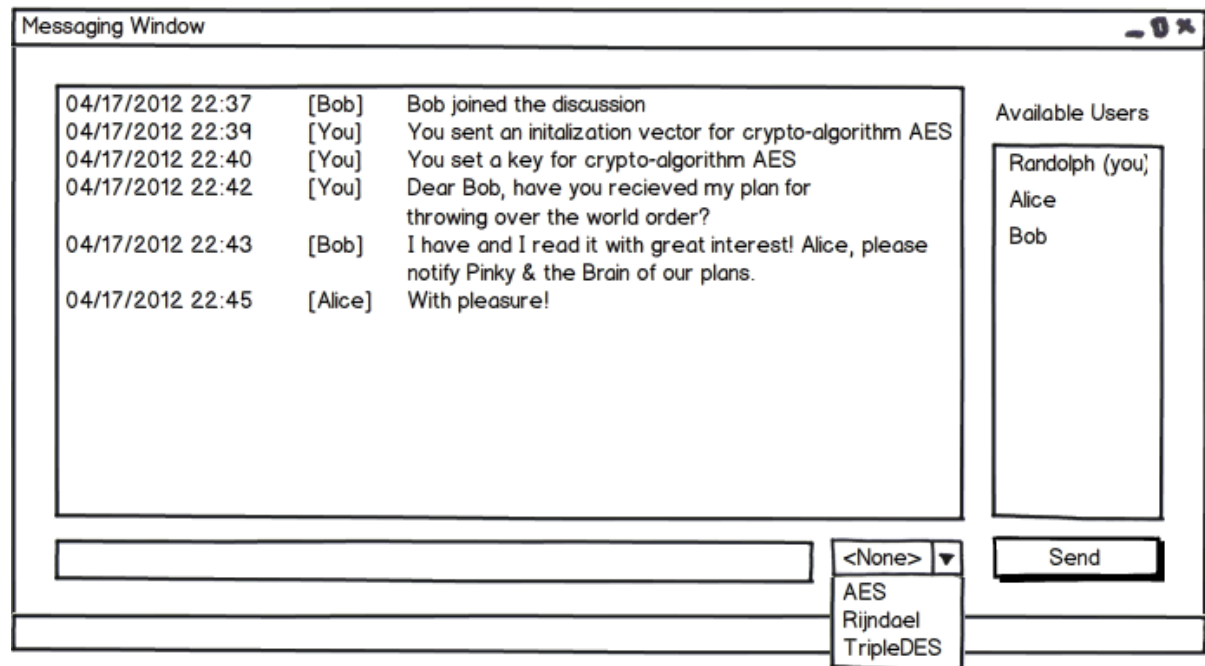


Abbildung 2.7: Mockup "Gesprächsdialog"

## 3 Konzept

Die Konzeptphase<sup>1</sup> des Wasserfallmodells behandelt die Entwicklung eines vollständigen und umfassenden Lösungskonzepts auf Basis der dokumentierten Anforderungen (nach [2]). [KAPITEL ÜBERSICHT HIER].

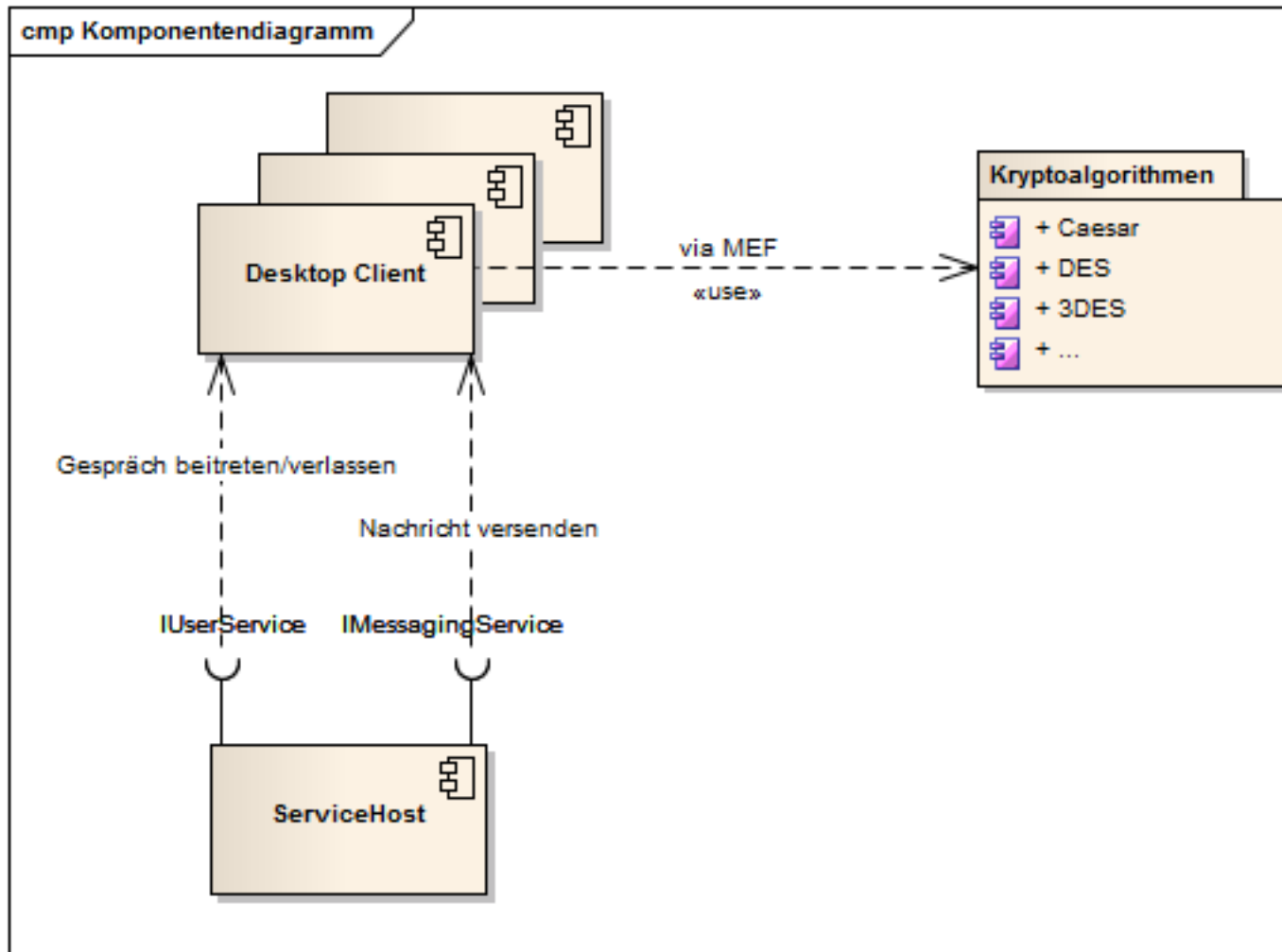
---

<sup>1</sup> auch Designphase genannt

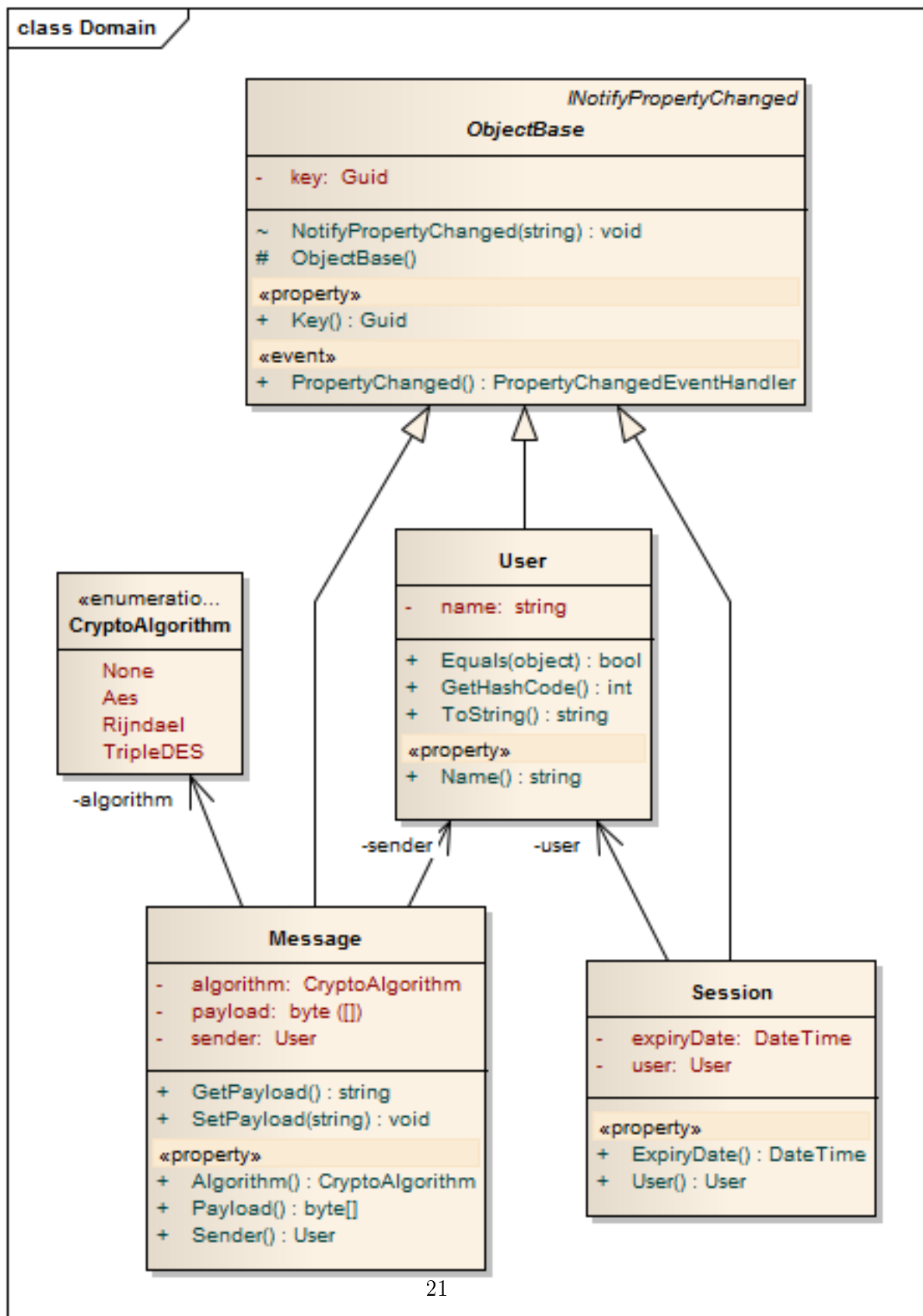
## 3.1 Konzept

### 3.1.1 Bausteinsicht

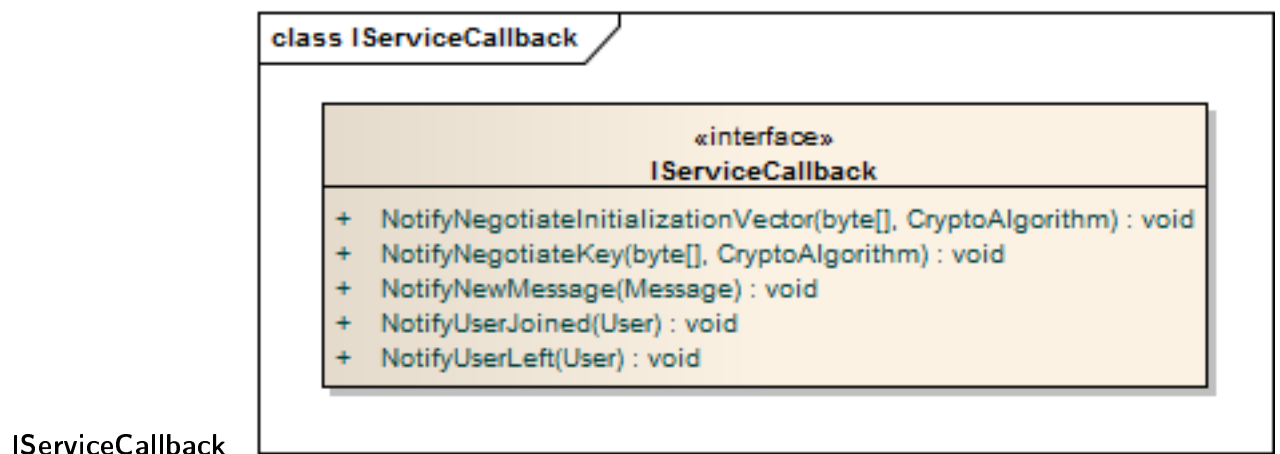
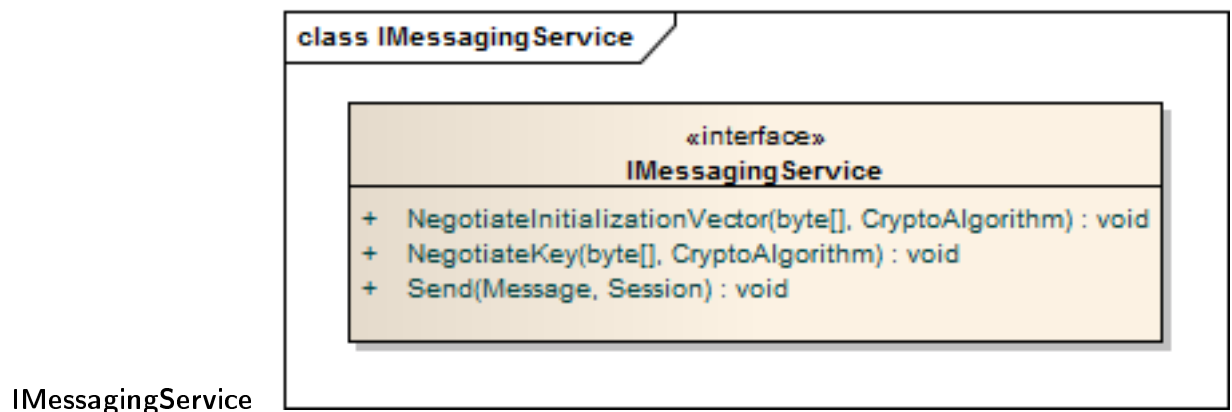
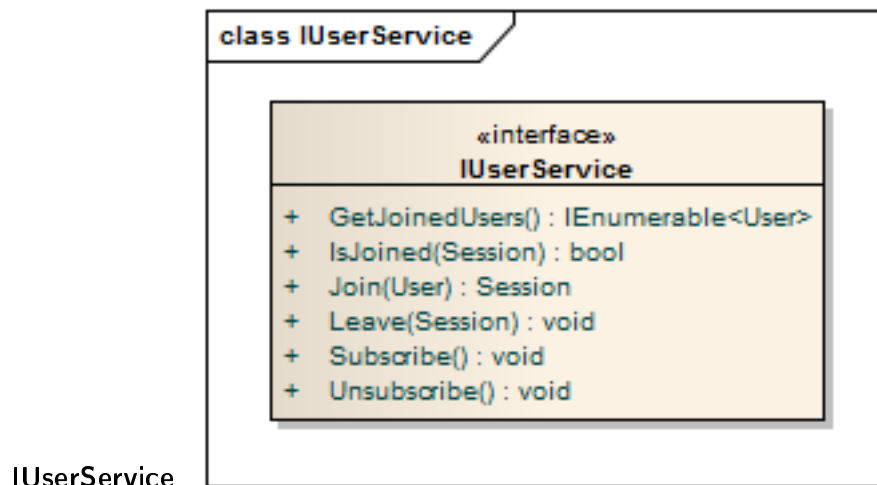
#### 3.1.1.1 Komponentendiagramm



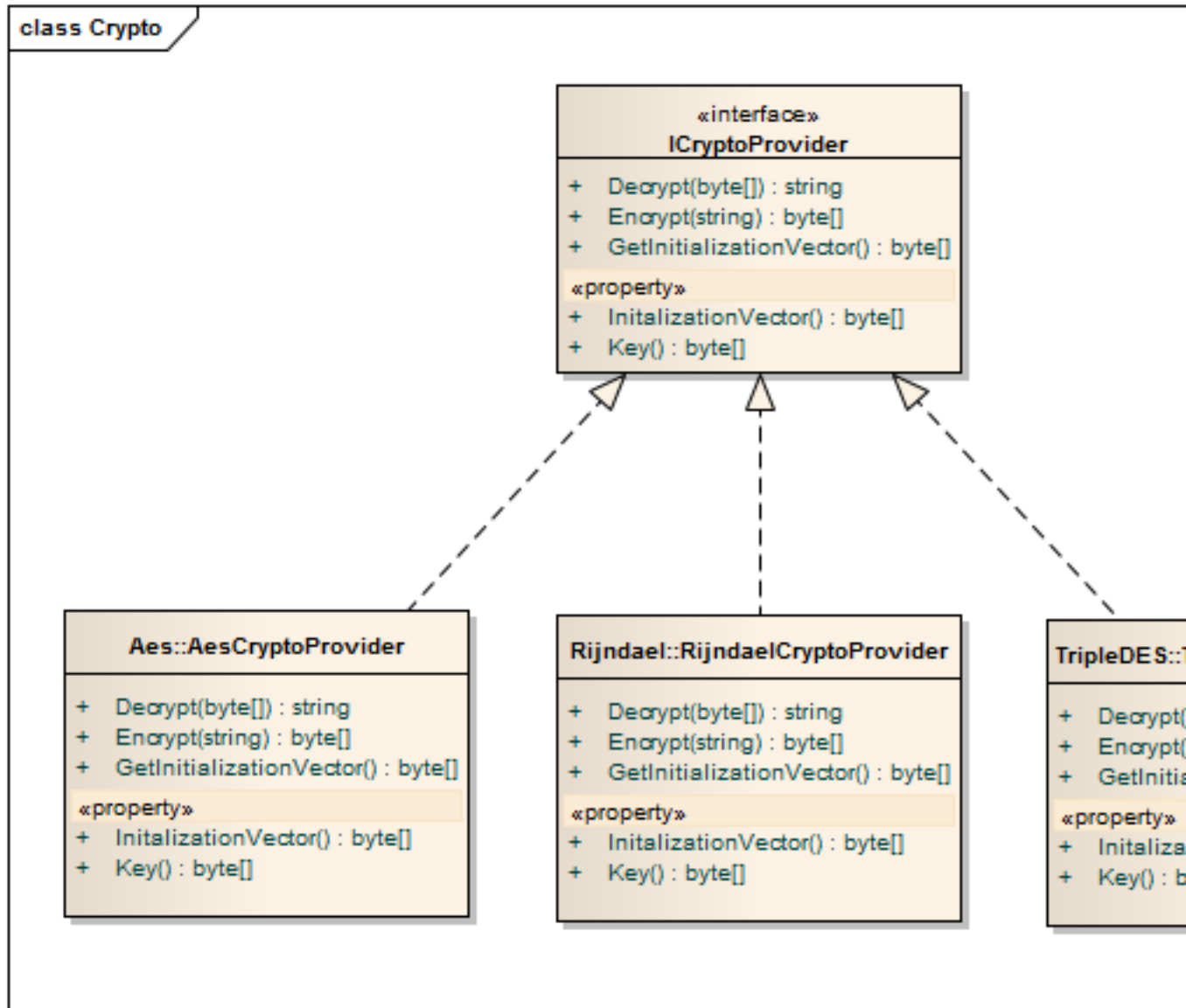
## 3.1.1.2 Domänenmodell



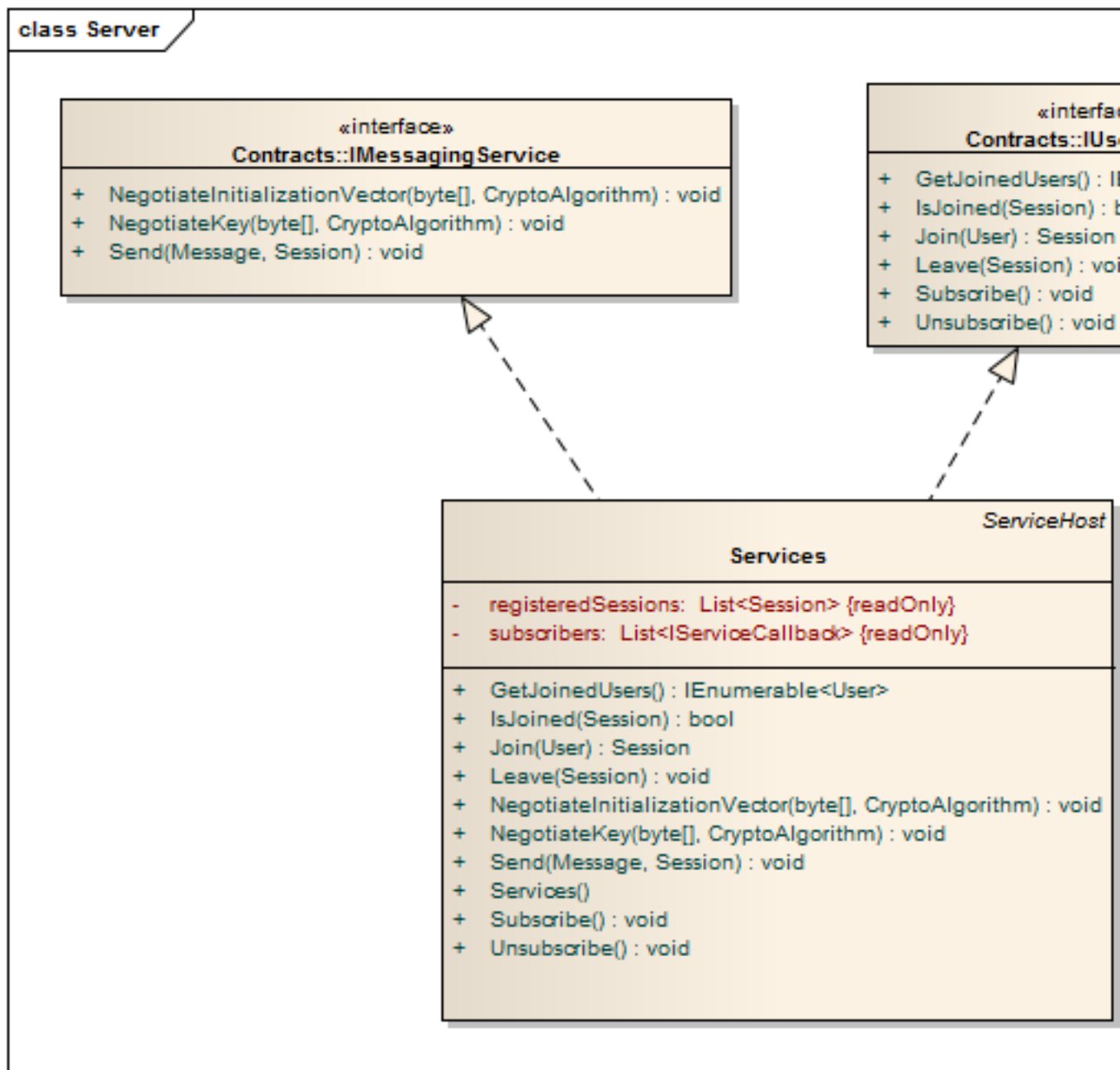
### 3.1.1.3 Service Contracts



## 3.1.1.4 Kryptoalgorithmen



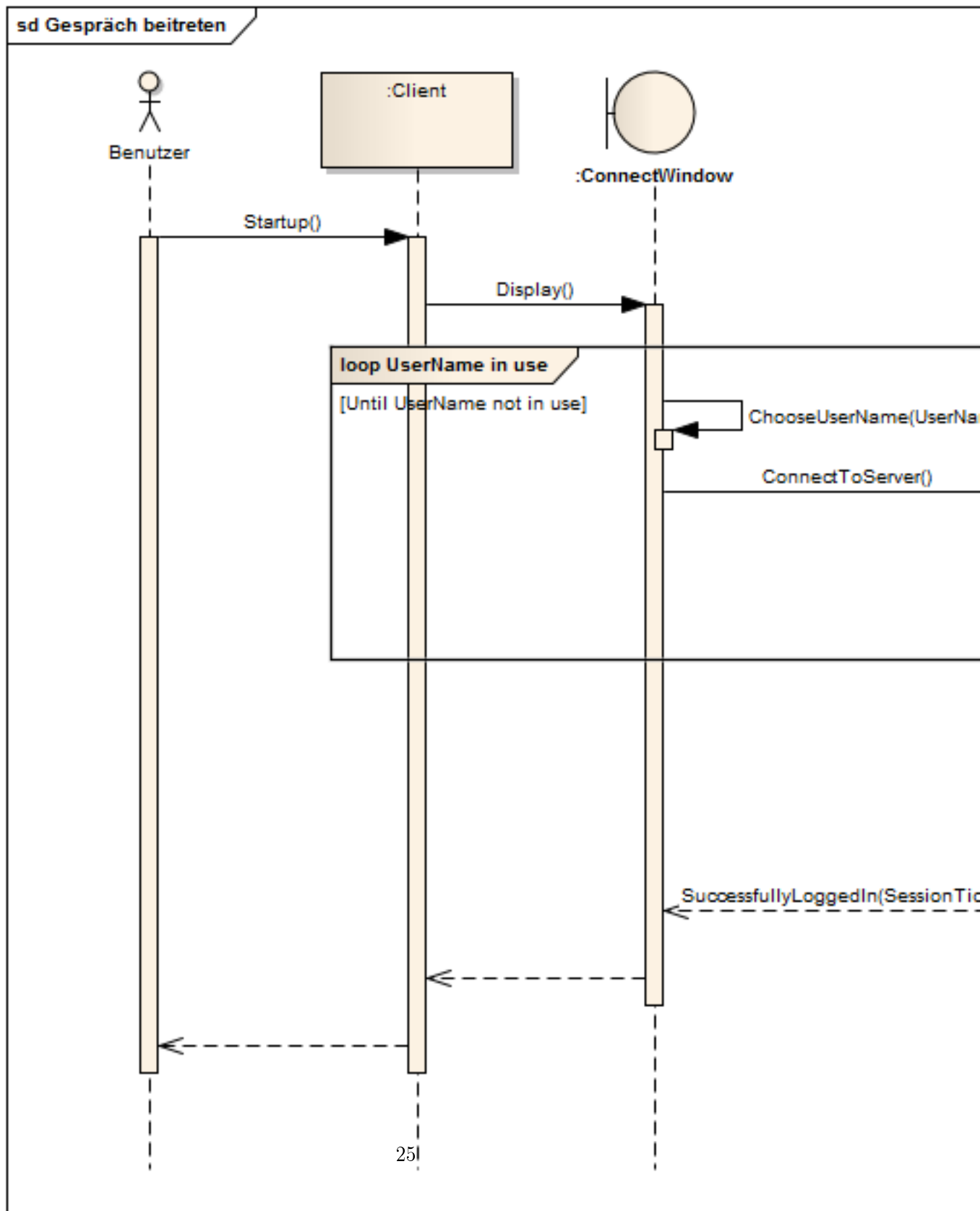
## 3.1.1.5 Server



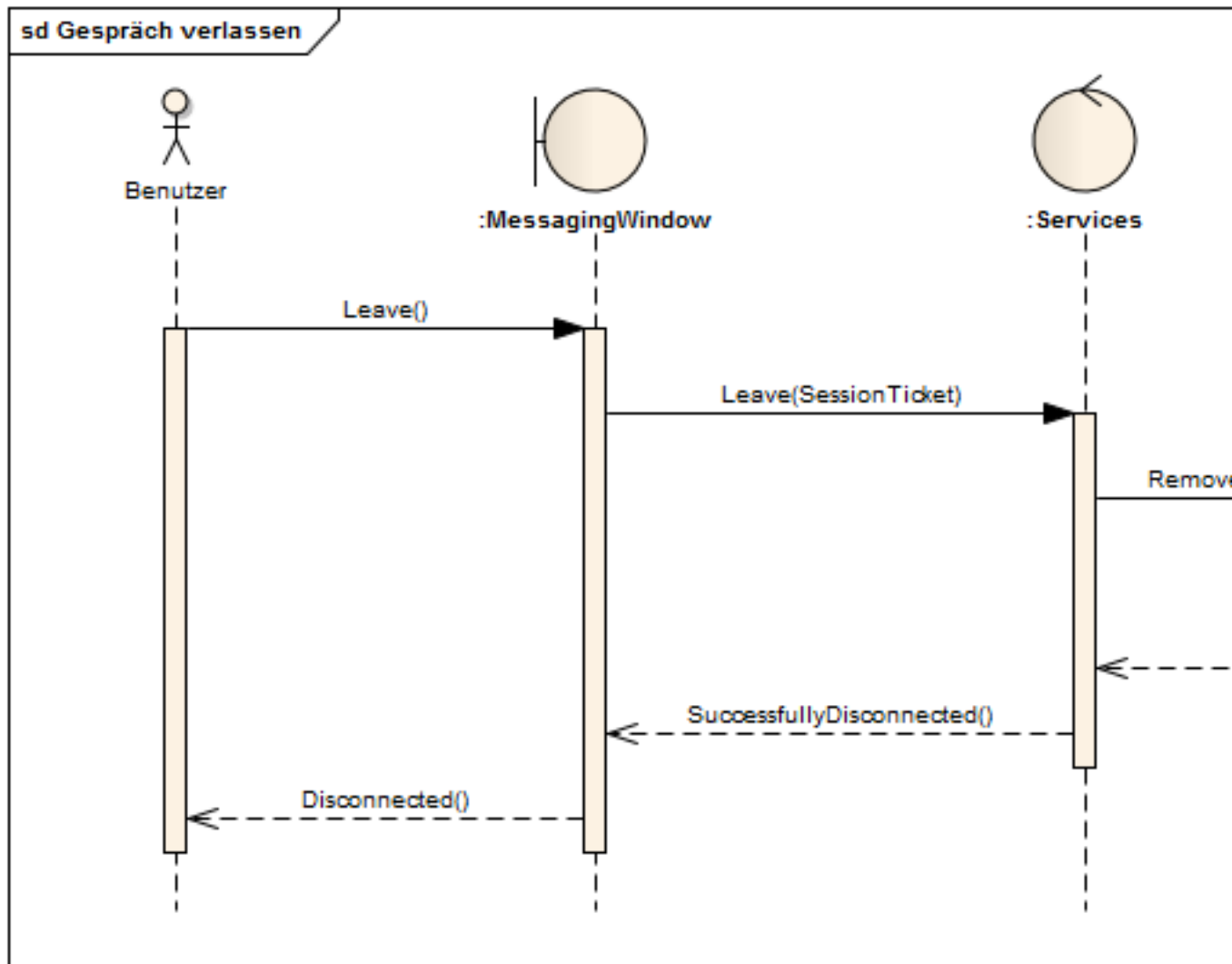


### 3.1.2 Laufzeitsicht

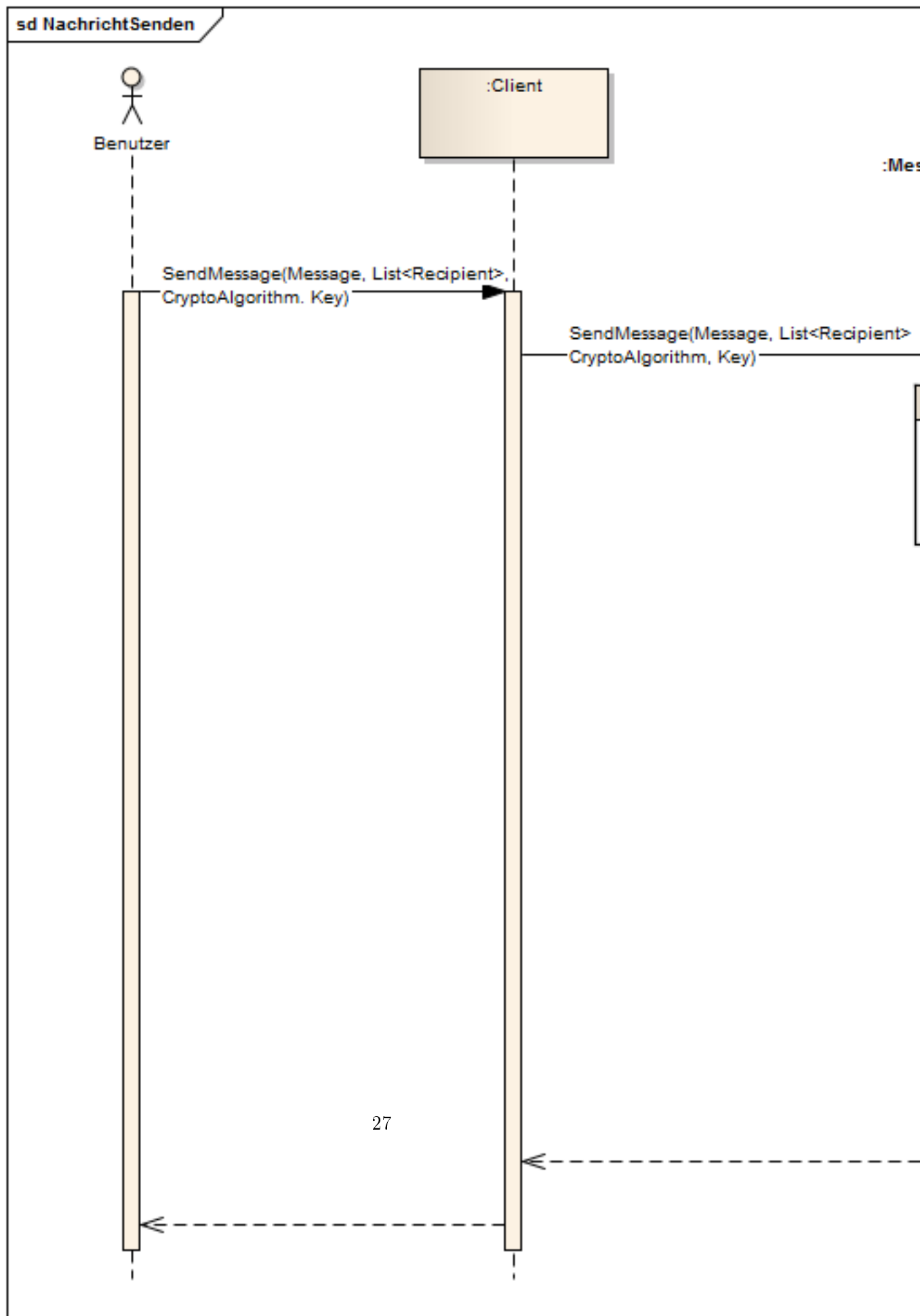
#### 3.1.2.1 Gespräch beitreten



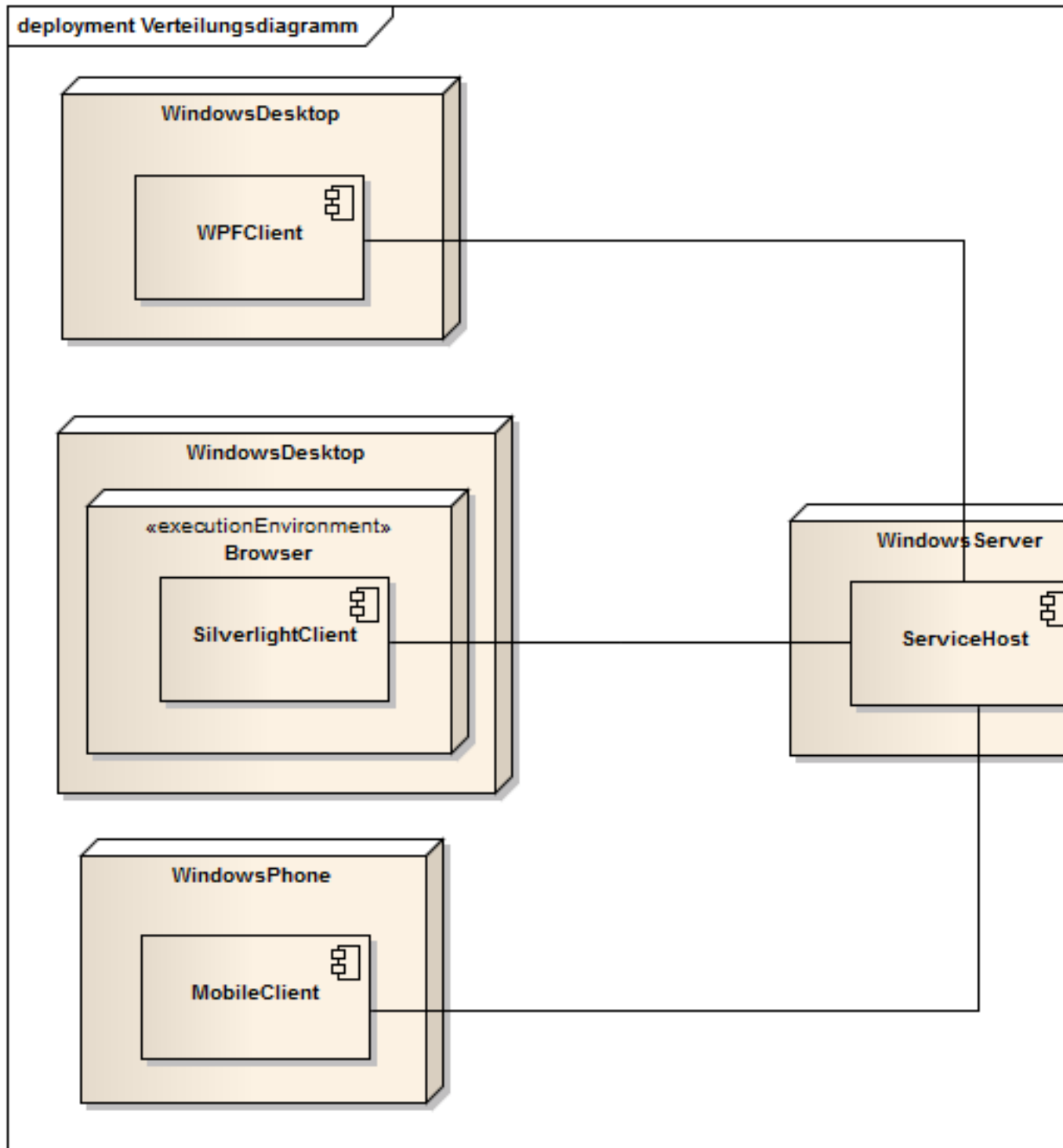
### 3.1.2.2 Gespräch verlassen



### 3.1.2.3 Nachricht senden



### 3.1.3 Verteilungssicht



## 4 Implementierung

### 4.1 Implementierung

## 5 Test

### 5.1 Test

## 6 Anhang

### 6.1 Akronyme

**UC** Use Case

**RUP** Rational Unified Process

**YAEM** Yet Another Encrypted Messenger

**ZHAW** Zürcher Hochschule für Angewandte Wissenschaften

### 6.2 Glossar

# Nomenclature

- Bug** Ein Programm- oder Softwarefehler wird als Bug bezeichnet und beschreibt im Allgemeinen ein fehlerhaftes Verhalten von Computerprogrammen.
- Dialog** Als Dialog (oder Dialogfenster) bezeichnet man in der Softwareentwicklung eine grafische Benutzerschnittstelle zur Mensch-Maschine-Interaktion zwischen Computerprogramm und Benutzer.
- Geheimtext** Der Geheimtext ist der Text, der durch die Verschlüsselung mittels eines kryptografischen Verfahrens unlesbar gemacht wurde.
- Kryptoalgorithmus** Ein Kryptoalgorithmus ist im Kontext von YAEM die konkrete Implementierung des Interfaces `YAEM.Crypto.ICryptoProvider` und bietet die Möglichkeit beliebige Nachrichten zu verschlüsseln beziehungsweise zu entschlüsseln.
- Mockup** Ein Mockup in der Softwareentwicklung bezeichnet einen rudimentären Wegwerfprototypen der Benutzerschnittstelle einer zu erstellenden Software. Mockups werden insbesondere in frühen Entwicklungsphasen eingesetzt, um Anforderungen an die Benutzeroberfläche in Zusammenarbeit mit Auftraggeber und Anwendern besser ermitteln zu können. Es handelt sich meist um ein reines Grundgerüst der Bedienelemente ohne weitere Funktionalität.
- RUP** Der Rational Unified Process ist ein kommerzielles Vorgehensmodell zur Softwareentwicklung von IBM.
- Use Case** Ein Use Case (deutsch Anwendungsfall) bündelt alle möglichen Szenarien, die eintreten können, wenn ein Akteur versucht, mit Hilfe des betrachteten Systems ein bestimmtes fachliches Ziel zu erreichen. Er beschreibt, was inhaltlich beim Versuch der Zielerreichung passieren kann, und abstrahiert von konkreten technischen Lösungen. Das Ergebnis des Anwendungsfalls kann ein Erfolg oder Fehlschlag/Abbruch sein.
- V-Modell** Das V-Modell ist ein Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in Phasen organisiert wird. Neben diesen Entwicklungsphasen definiert das V-Modell auch das Vorgehen zur Qualitätssicherung (Testen) phasenweise.

## 6.3 Bibliographie



# Literaturverzeichnis

- [1] Ieee standard glossary of software engineering terminology, 1990.
- [2] Markus Klink Guido Zockoll Bernd Oestereich, Claudia Schröder. *OEP - oose Engineering Process: Vorgehensleitfaden für agile Softwareprojekte*. dpunkt, 2006.
- [3] Franz-Josef Elmer. Software engineering, methodologien. 2005.
- [4] Klaus Pohl and Chris Rupp. *Basiswissen Requirements Engineering*. dpunkt.verlag, 2011.
- [5] Gernot Starke. *Effektive Software-Architekturen*. Carl Hanser Verlag, 2011.
- [6] Dr. Olaf Stern. Reglement semesterarbeit. 2010.
- [7] Trung Hung VO. Software development process, 07 2007.