- Systementwurf -

„Geheime Kommunikation mit Quantenkryptographie - eine Lernsoftware zum Verstehen der Grundlagen“

Version: 1.0

|  |  |  |
| --- | --- | --- |
| Projektbezeichnung | Quakrypto | |
| Projektleiter | Simon Schisslbauer | |
| Verantwortlich | Team | |
| Erstellt am | 29.04.2023 | |
| Zuletzt geändert | 27.05.2023 20:15 | |
| Bearbeitungszustand | X | in Bearbeitung |
|  | vorgelegt |
|  | fertig gestellt |
| Dokumentablage | Main-Branch im Projekt | |

Änderungsverzeichnis

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Änderung | | | Geänderte Kapitel | Beschreibung der Änderung | Autor | Zustand |
| Nr. | Datum | Version |
| 1 |  | 1.1 | Alle | Initiale Produkterstellung |  |  |

Prüfverzeichnis

Die folgende Tabelle zeigt einen Überblick über alle Prüfungen – sowohl Eigenprüfungen wie auch Prüfungen durch eigenständige Qualitätssicherung – des vorliegenden Dokumentes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datum | Geprüfte Version | Anmerkungen | Prüfer | Neuer Produktzustand |
|  |  |  |  |  |

Inhalt

[1 Einleitung 4](#_Toc382581489)

[2 Architekturprinzipien und Entwurfsalternativen 4](#_Toc382581490)

[3 Übersicht über die Zerlegung des Systems 5](#_Toc382581491)

[4 Schnittstellenübersicht 5](#_Toc382581492)

[5 Systemkomponenten 5](#_Toc382581493)

[6 Designabsicherung 5](#_Toc382581494)

[7 Abkürzungsverzeichnis 5](#_Toc382581495)

[8 Literaturverzeichnis 6](#_Toc382581496)

[9 Abbildungsverzeichnis 6](#_Toc382581497)

# Einleitung

Dieses Dokument soll ein Grundverständnis der Systemstruktur vermitteln ohne den Entwurf bis in letzte Einzelheiten darzulegen. Das Grundverständnis soll jedoch ausreichen, um sich ggf. anhand des Quellcodes in weitere Einzelheiten leicht einarbeiten zu können.

Kernthemen in diesem Dokument sind:

* Übersicht über die Zerlegung des Systems: Welche (größeren) Systemkomponenten gibt es? Wofür ist jede einzelne davon zuständig? Wie hängen diese Komponenten voneinander ab?

* Schnittstellenübersicht: Welche Schnittstellen stellt das System und jede Systemkomponente für seine/ihre Umgebung bereit?
* Systemkomponenten: Wie ist jede Systemkomponente aufgebaut?
* Designabsicherung: Zeigt für ausgewählte „architektur-relevante“ Use-Case-Szenarien, dass und wie diese mit dem gewählten Systementwurf realisierbar sind.

Der Systementwurf wird auf Grundlage der funktionalen und nicht-funktionalen Anforderungen sowie des konzeptuellen Datenmodells gewonnen, etwa indem man für ausgewählte „architektur-relevante“ Use-Case-Szenarien untersucht, welche Teile des Systems zur Realisierung in welcher Weise zusammenarbeiten müssen.

Die Gliederung dieses Dokuments orientiert sich grob am Aufbau der V-Modell-XT®[[1]](#footnote-1)-Produkte „System-Architektur“ und „SW-Architektur“, ist jedoch zur Verwendung für die Veranstaltung **„Software-Projekte“** in Informatik-Curricula der **OTH-Amberg-Weiden** angepasst worden (und nicht konform zum V-Modell-XT).

# Architekturprinzipien und Entwurfsalternativen

Realisierung des Systems unter der Verwendung der MVVM-Architektur, das aus drei Kernkomponentnen besteht, Model, View und ViewModel. Das MVVM-Architektur bietet dadurch eine klare Trennung von Daten und Benutzeroberfläche, was eine bessere Wartbarkeit und Testbarkeit der Anwendung ermöglicht. Durch die Verwendung von Datenbindungen wird der Code deutlich schlanker und einfacher zu lesen, da Daten automatisch aktualisiert werden, wenn sich ihre zugrunde liegenden Werte ändern.



Die View ist für die Darstellung der Benutzeroberfläche verantwortlich. Sie zeigt dem Benutzer die Daten an, die vom ViewModel bereitgestellt werden, und ermöglicht es dem Benutzer, Aktionen auszuführen, die vom ViewModel verarbeitet werden.

Das ViewModel dient als Verbindung zwischen der View und dem Model. Es stellt der View die Daten zur Verfügung, die für die Darstellung benötigt werden, und verarbeitet die Aktionen, die der Benutzer ausführt. Es enthält auch die Logik zur Aktualisierung des Models, wenn Änderungen an den Daten vorgenommen werden.

Das Model repräsentiert die Daten und die Geschäftslogik der Anwendung. Es enthält die Datenstrukturen, die für die Anwendung benötigt werden, sowie Methoden zur Verarbeitung und Aktualisierung dieser Daten.

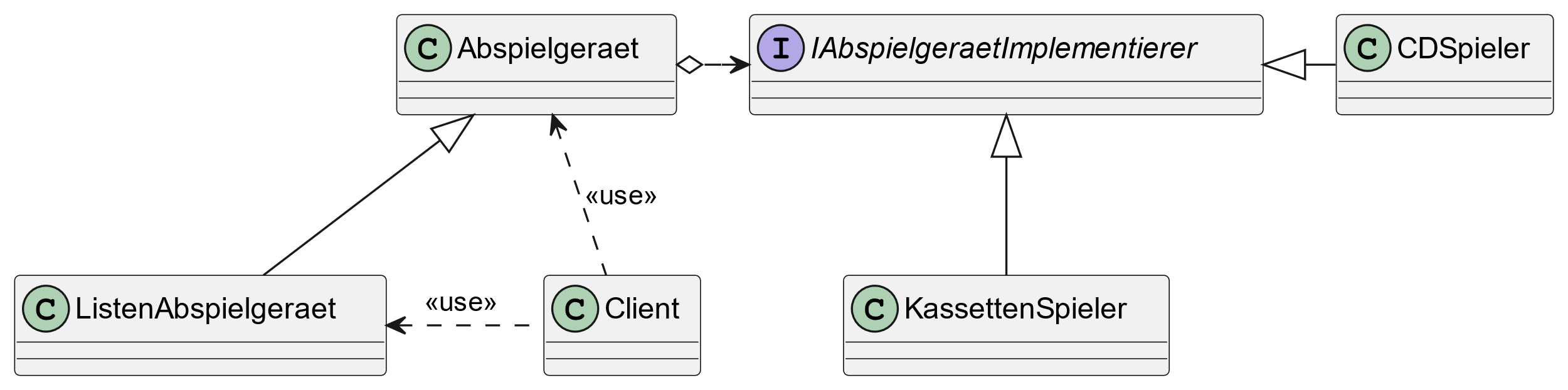
Die Verwendung des MVVM-Prinzips erleichtert auch die Zusammenarbeit zwischen Entwicklern, da die Trennung von Daten und Benutzeroberfläche die Möglichkeit bietet, dass Entwickler unabhängig an verschiedenen Teilen der Anwendung arbeiten können, ohne sich gegenseitig zu behindern. Dadurch wird ermöglicht Änderungen an der Benutzeroberfläche vorzunehmen, ohne die Logik zu beeinträchtigen.

Durch die Trennung der Zuständigkeiten wird es einfacher, Unit-Tests für das ViewModel und das Model zu schreiben, die die Geschäftslogik repräsentieren. Das bedeutet, dass die Logik unabhängig von der Benutzeroberfläche getestet werden kann. Da das ViewModel keine Kenntnis von der View hat, kann es einfach getestet werden, ohne dass eine tatsächliche Benutzeroberfläche benötigt wird. Somit führt die Verwendung von MVVM zu einer verbesserten Testbarkeit und Wartbarkeit des Codes.

## Structural Patterns

* **Bridge Pattern:**

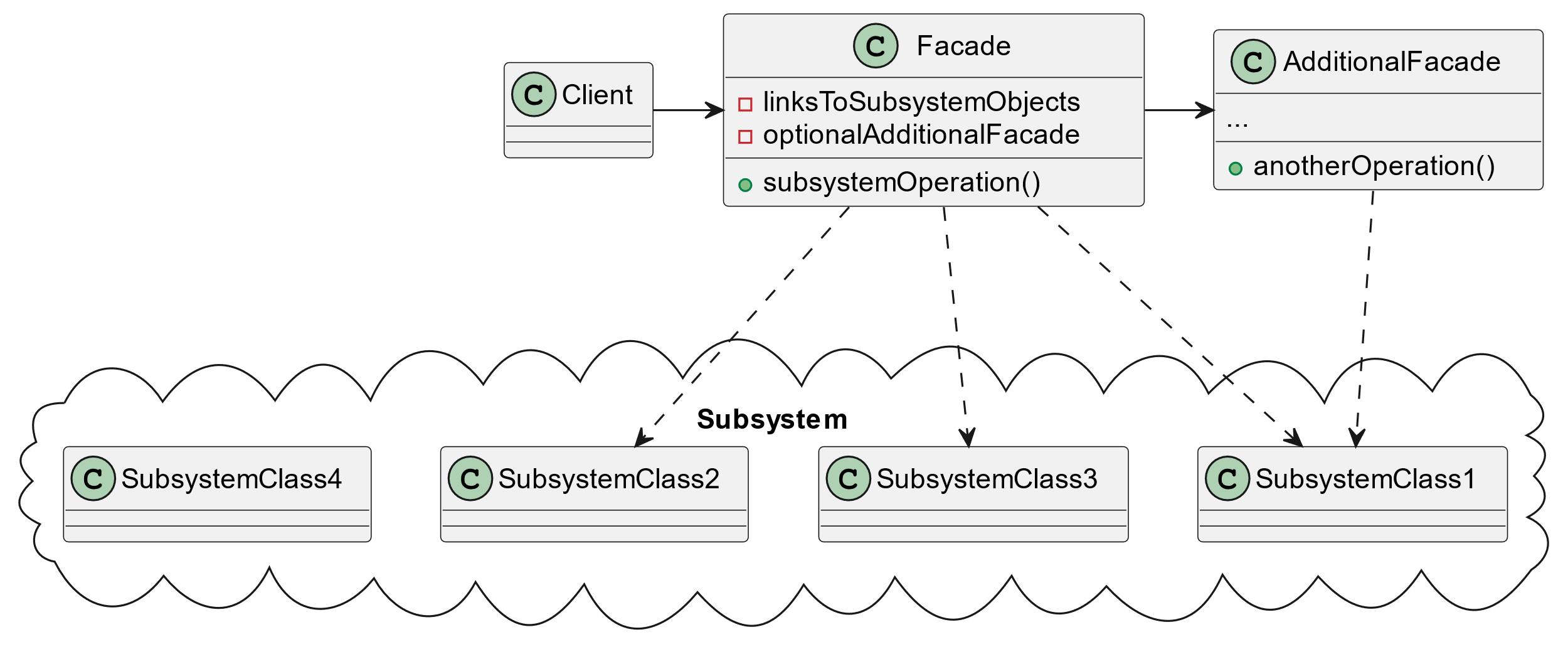
Splitten einer großen Klasse in oder eine Reihe eng verwandter Klassen, in zwei separate Hierarchien (Abstraktion und Implementierung, im Beispiel Shape ist die Abstraktion und Color die Implementierung)



Vorteile:

* + Abstraktion und Implementierung werden entkoppelt.
  + Klassen können unabhängig voneinander implementiert werden.
* **Facade Pattern:**

Das Facade Pattern stellt dynamisch Zugriff auf die zugrunde liegende Funktionalität der Subsysteme bereit und ist ein Interface. Der Client verwendet dann die Facade um auf die Subsysteme zu zugreifen.



**Vorteile:**

* + Isolierung von Subsystemen
  + Simple Fassade

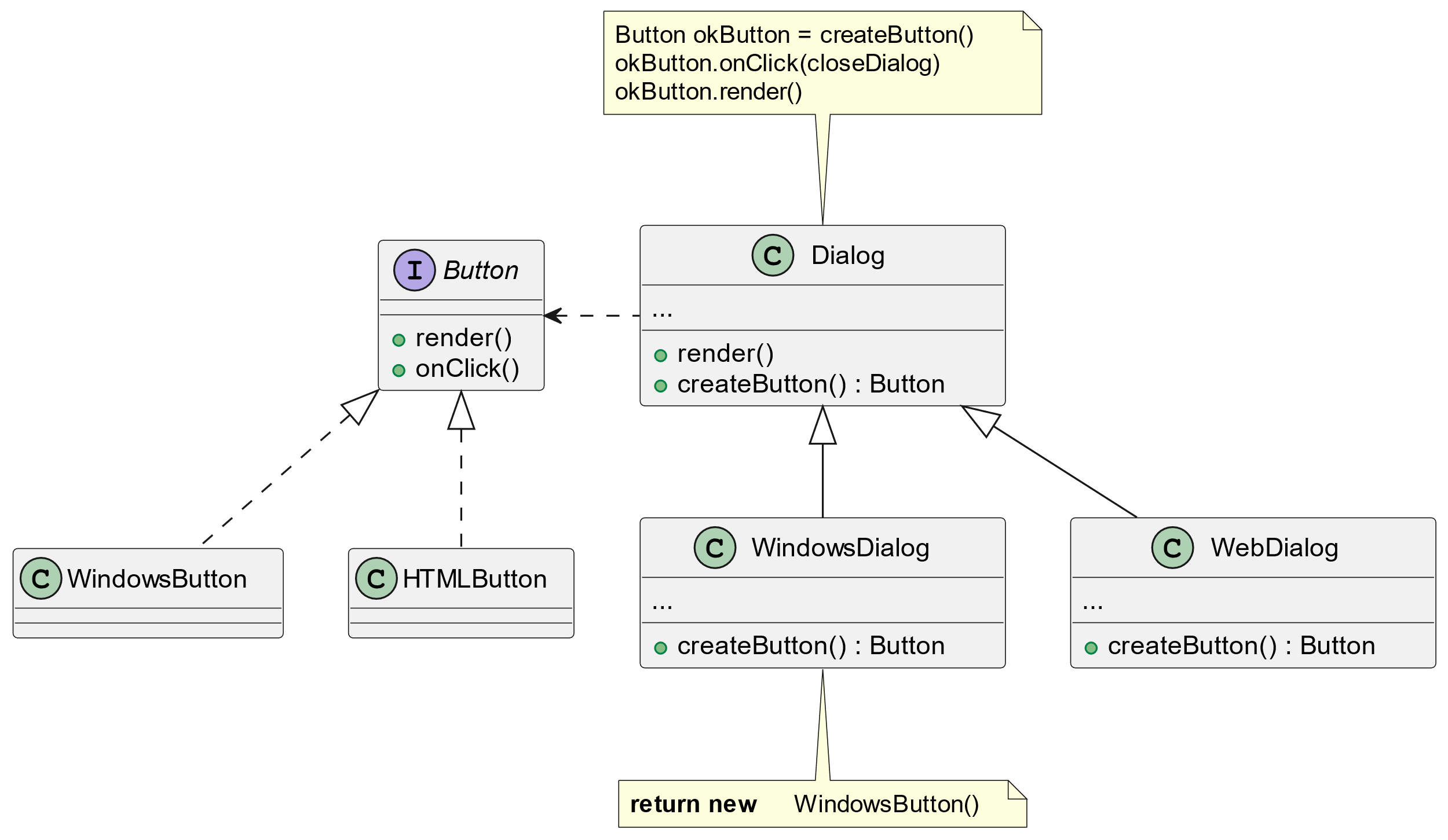
**Nachteile:**

* + Fassade Klasse wird an alle Klassen gekoppelt

## Creational Patterns

* **Factory Pattern:**

Factory Methode ist ein Erzeugungsmuster, das eine Schnittstelle zur Erzeugung von Objekten in einer Superklasse bereitstellt, jedoch erlaubt den Typ der erzeugten Objekte zu ändern.

****

**Vorteile:**

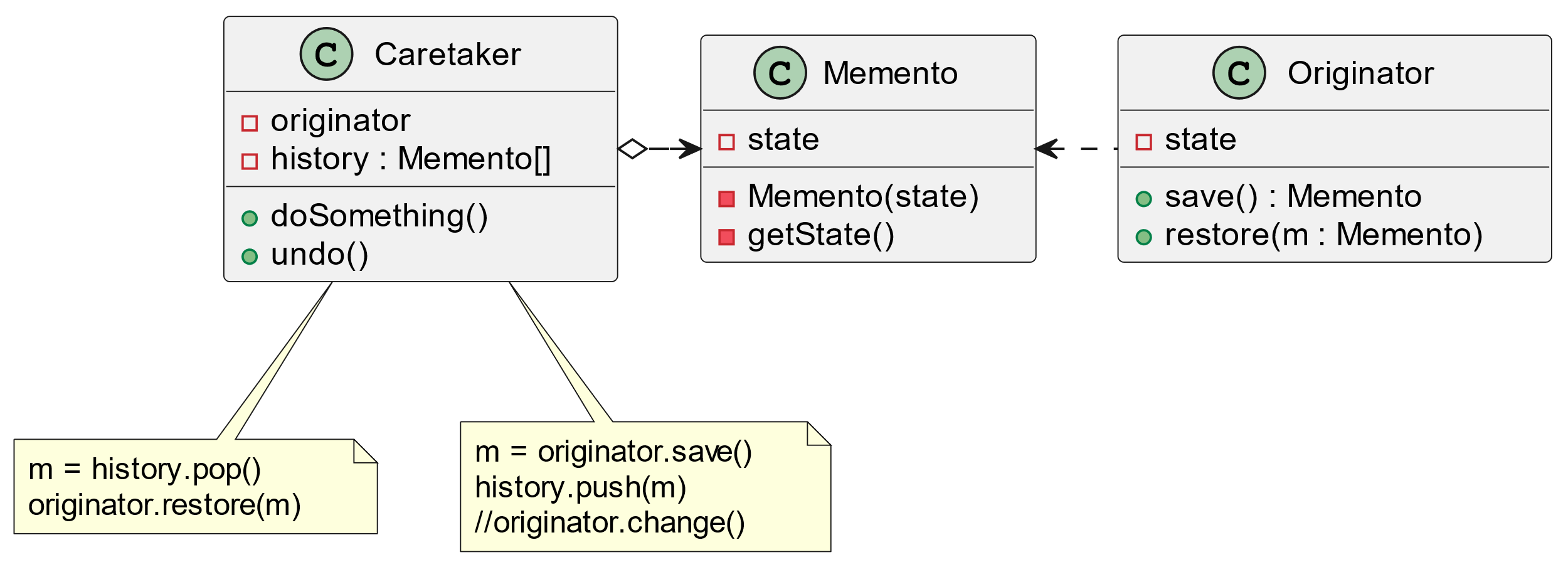
* + Erweiterbarkeit durch Vermeiden von enger Kopplung der Erzeuger und Produkte
  + Testbarkeit

**Nachteil:**

* + Aufwendiger zu implementieren
  + Overhead durch die Instanziierung durch die Factory Klasse

## Behavioral Patterns

* **Memento Pattern:**
* Memento Pattern ermöglicht es den Zustand von Objekten zu speichern und wiederherzustellen, ohne die Implementierungsdetails preiszugeben.



**Vorteile:**

* + Ermöglicht Momentaufnahmen von Objekten zu erzeugen, ohne die Kapselung zu verletzen

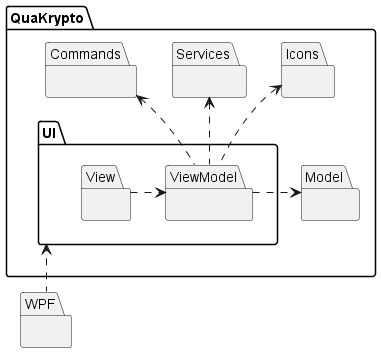
**Nachteile:**

* + Verbraucht viel RAM
  + Momentaufnahmen müssen nach Ende des Lebenszyklus des Originators gelöscht werden

# Übersicht über die Zerlegung des Systems

Hier soll eine Gesamtübersicht über alle Komponenten und deren gegenseitige Abhängigkeiten (z.B. in Form eines UML-Paketdiagramms) gegeben werden.

Für jede Komponente soll ihre Zuständigkeit kurz erklärt sein. Fremdkomponenten (z.B. Open-Source aus dem WWW), die in das System integriert wurden, müssen mit Angabe der Bezugsquelle als solche gekennzeichnet sein. Die evtl. Verwendung einer Fremdkomponente ist vorher mit dem „Auftraggeber“ abzustimmen und darf Lizenzrechte nicht verletzen.



|  |  |
| --- | --- |
| Paket | Aufgaben |
| QuaKrypto.UI.View | Enthält Klassen für die Darstellung der Lernsoftware QuaKrypto |
| QuaKrypto.UI.ViewModel | Enthält die Logik der View und stellt die Informationen für die View bereit. Weist die View darauf hin, dass neue Informationen verfügbar sind, und verarbeitet Änderungen durch die View. |
| QuaKrypto.Model | Enthält die Klassen, die den Problembereich repräsentieren. Dabei wird das Protokoll und der Ablauf der Übungsszenarien realisiert und die Aufzeichnung erzeugt. Weiterhin wird die Kommunikation zwischen den einzelnen Benutzergruppen realisiert. |
| QuaKrypto.Commands | Enthält die Klassen, die die Befehle zur Realisierung der MVVM-Architektur realisieren. |
| QuaKrypto.Services | Enthält die Klassen, die die Netzwerkschnittstelle bereitstellen |
| QuaKrypto.Icons | Enthält die Icons, die zur Darstellung in den Views benötigt werden |
| Externe Pakete |  |
| WPF | WPF (Windows Presentation Foundation) ist ein Framework zur Erstellung von Benutzeroberflächen in der .NET-Umgebung. Es enthält die nötigen Funktionen zur Realisierung von MVVM. |

# Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält. Automatisch generierte BeschreibungSchnittstellenübersicht

Hier soll dargelegt werden, welche Schnittstellen das System und jede Komponente seiner/ihrer Umgebung zur Verfügung stellt. Jede Schnittstelle beschreibt einen bestimmten (zur Zuständigkeit passenden!) Teil des Verhaltens einer Komponente.

Die Klassen aus dem *View* Paket werden auf Grund der Übersichtlichkeit hier nicht dargestellt. Weiterhin wird nur ein Teil der Klassen aus dem Paket *Model* dargestellt, die nicht direkt mit der Schnittstelle zwischen den Paketen *UI* und *Model* in Verbindung stehen.

# Systemkomponenten

Hier sollen für jede Komponente in einem eigenen Unterabschnitt die wichtigsten Klassen dargelegt werden (z.B. durch ein UML-Klassendiagramm). Nach aussen hin sichtbare Klassen sollen von internen Klassen klar unterscheidbar sein. Für jede nach außen hin sichtbare Klasse soll deren Zuständigkeit erkennbar sein (notfalls kurz erläutern). Interne Klassen müssen nicht vollzählig dargelegt werden, wenn sie für das Verständnis (und die Designabsicherung im nächsten Abschnitt) entbehrlich sind.

|  |  |
| --- | --- |
| Klasse | Zuständigkeit |
| Navigator | Die Navigator-Klasse ist eine Hilfsklasse, die die Navigation zwischen verschiedenen Views koordiniert. |
| BaseViewModel | BaseViewModel ist eine Klasse, die als Grundlage für die Implementierung von ViewModels dient. |
| MainViewModel | Die MainViewModel Klasse verwaltet die anzuzeigenden Views. |
| SpielViewModel | Beinhaltet die Logik der View und sorgt dafür, dass die Handlungschritte der Rolle (Benutzer) an die Klasse Uebungsszenario weitergegeben wird. |
| LobbyScreenViewModel | Das LobbyScreenViewModel ist dafür da, einem Uebungsszenario beizutreten und sich die Rolle dafür auszuwählen. |
| DelegateCommand | Die DelegateCommand Klasse implementiert das Interface ICommand und ermöglicht es in den ViewModel-Klassen ein Execute und ein CanExecute zuzuweisen. |
| IUebungsszenario | Stellt ein Übungsszenario dar und definiert seine Eigenschaften und Methoden. Sie enthält Getter und Setter für Eigenschaften wie aktuelle Rolle, Schwierigkeitsgrad, Variante, Start- und Endphasen, Übertragungskanal und Aufzeichnung. Außerdem enthält sie Methoden zum Veröffentlichen einer Lobby, Hinzufügen und Entfernen von Rollen, Fortschreiten zum nächsten Zug und Generieren eines Protokolls. |
| Netzwerk | Die Klasse "Netzwerk" stellt das Netzwerkmodul dar. Sie hat Eigenschaften wie "VerfügbareLobbys", eine Sammlung von Informationen über verfügbare Lobbys im Netzwerk. Die Klasse enthält auch Methoden, um nach Lobbys zu suchen und Lobbyinformationen zyklisch zu senden |

# Designabsicherung

Für die Szenarien ausgewählter („architektur-relevanter“) Anwendungsfälle soll die Zusammenarbeit der beteiligten Systemkomponenten (z.B. in einem Sequenzdiagramm) dargegelegt sein. Die Darlegung soll davon überzeugen, dass der gewählte Systementwurf für die Realisierung der Szenarien wirklich tauglich ist.

Ein Bild, das Text, Quittung, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

# Abkürzungsverzeichnis

|  |  |
| --- | --- |
| Abkürzung | Erklärung |
| WPF | Windows Presentation Foundation |
| MVVM | Model-View-ViewModel |

# Literaturverzeichnis

# Abbildungsverzeichnis

1. V-Modell® ist eine geschützte Marke der Bundesrepublik Deutschland. [↑](#footnote-ref-1)