Technische Universität Braunschweig

IAS | INSTITUTE FOR APPLICATION SECURITY

Software-Entwicklungspraktiktum (SEP)

ARID - Augmented Reality in Disguise

Software-Entwicklungspraktikum (SEP)
Sommersemester 2024

Technischer Entwurf

Auftraggeber:
Technische Universität Braunschweig
Institut für Anwendungssicherheit (IAS)
Prof. Dr. Martin Johns
Mühlenpfordstraße 23
38106 Braunschweig

Betreuerin: Alexandra Dirksen

Auftragnehmer:

Name	E-Mail-Adresse
Amir Fakhim Hashemi	a.fakhim-hashemi@tu-braunschweig.de
Ibrahim Abdullah	i.abdullah@tu-braunschweig.de
Jadon-Kim Fischer	jadon-kim.fischer@tu-braunschweig.de
Mohamed Ali Mrabti	m.mrabti@tu-braunschweig.de
Tim Kütemeyer	t.kuetemeyer@tu-braunschweig.de
Vyvy Nguyen	Vyvy.nguyen@tu-braunschweig.de

Braunschweig, 3. Juli 2024

Bearbeiterübersicht

Kapitel	Autoren	Kommentare
1	Amir, Vyvy	—
1.1	Amir, Vyvy	—
2	Ibrahim, Jadon	Entspringt Fachentwurf
3	Ibrahim	—
3.1	Jadon	—
3.2	Jadon	—
3.3	Jadon	—
4	Amir, Vyvy	—
5	Vyvy	—
5.1	Mohamed Ali	—
5.2	Ibrahim, Mohamed Ali	—
6	Tim	Entspringt Fachentwurf
7	Mohamed Ali	Entspringt Fachentwurf
8	Vyvy	—
9	Tim	—
9.1	Tim	—
9.2	Tim	—
9.3	Tim	—
10	Sämtliche	Fortlaufend ergänzt

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Beim technischen Entwurf handelt es sich um eine Erweiterung des Fachentwurfs. Dabei werden die Funktionalitäten genauer und in ihren einzelnen Komponenten analysiert. Es wird die Softwarearchitektur des Projektes thematisiert, in dem genauer auf die Komponenten, anhand eines Komponentendiagramms, sowie auf die Schnittstellen eingegangen wird. Es wird ein Verteilungsdiagramm erstellt und die Erfüllung der Kritierien detailliert beschrieben. Ein weiterer wichtiger Aspekt ist zudem der Implimentierungsentwurf, bei dem die Komponenten mithilfe von Klassendiagrammen genauer erklärt werden.

Die folgenden Absätze des Kapitels werden dem Fachentwurf entnommen.

Bei dem Projekt ARID - Augemented Reality in Disguise ist die Hauptfunktion das Entschlüsseln der versteckten QR-Codes mithilfe des Monocle. Der User, der das Monocle trägt, bedient das vorhandene Touchpad um ein Bild aufzunehmen. Dieses Bild wird dem Host weitergeleitet, dort wird die Nachricht angezeigt und dann entschlüsselt. Diese Nachricht wird dem Host und auf dem Monocle angezeigt. Mögliche Host sind Computer, Laptops sowie mobile Endgeräte.

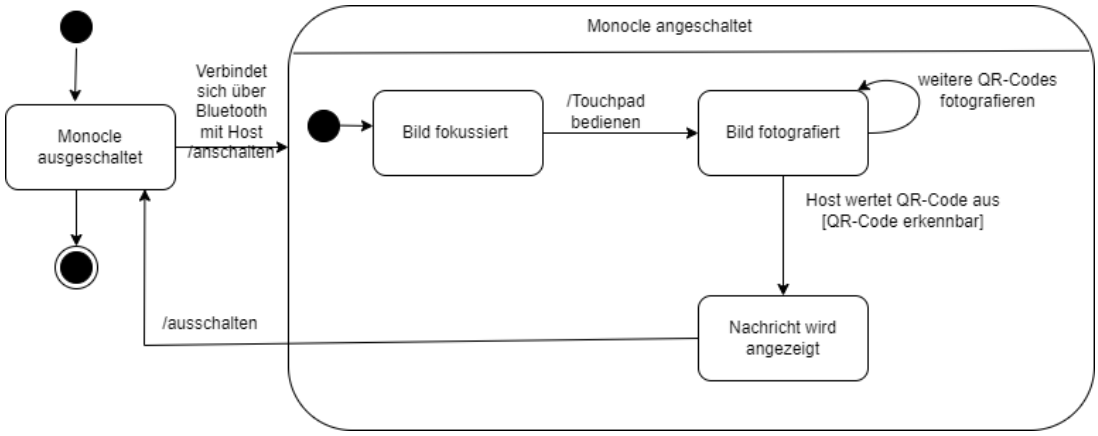

Abbildung 1.1: Zustandsdiagramm des Anwendungsprozesses

In dem Zustandsdiagramm ist ein Überblick des Systems zusehen. Das Monocle wird eingeschaltet und über Bluetooth kann sich dieser mit dem Host verbinden. Wenn das Monocle eingeschaltet ist, fokussiert dieser sich auf den QR-Code, um ein verschwommenes Bild zu vermeiden. Dann kann das Bild fotografiert werden, indem das Touchpad bedient wird. Das Bild wird dem Host geschickt. Ist das Bild scharf genug, wird dieser dann entschlüsselt und die Botschaft wird

dem Monocle gesendet, welcher dann die Nachricht anzeigt. Daraufhin kann das Monocle wieder ausgeschaltet werden.

1.1 Projektdetails

In diesem Abschnitt werden einige interessante Sachverhalte mit Hilfe von Aktivitätsdiagrammen genauer erläutert:

- QR-Code erstellen und verstecken
- QR-Code erkennen und anzeigen
- Ver- und Entschlüsselung

1.1.1 QR-Code erstellen und verstecken

Der Python-Code 'qrcode-encode' generiert aus der verschlüsselten Botschaft einen QR-Code. Mithilfe einer Deep-Learning KI namens Stable Diffusion wird ein Bild generiert, in dem der QR-Code mittels Steganographie versteckt wird. Es existieren verschiedene WebUIs mit denen die Bilder anhand von text-to-image oder image-to-image generiert werden können. Grundlage für das zu entstehende Bild sind Prompts, schriftliche Eingabeaufforderung, die der KI instruieren, wie das zu generierende Bild auzusehen hat. Ein wichtiger Punkt der zu beachten gilt, ist die Relation zwischen QR-Code und Kunst. Der QR-Code muss insofern in dem Bild integriert werden, so dass es vom Monocle erkannt werden kann, er aber für den bloßer Betrachter nicht zu erkennen ist. Um diese Gewichtung zu steuern, existieren verschiedene Parameter.

1.1.2 QR-Code erkennen und anzeigen

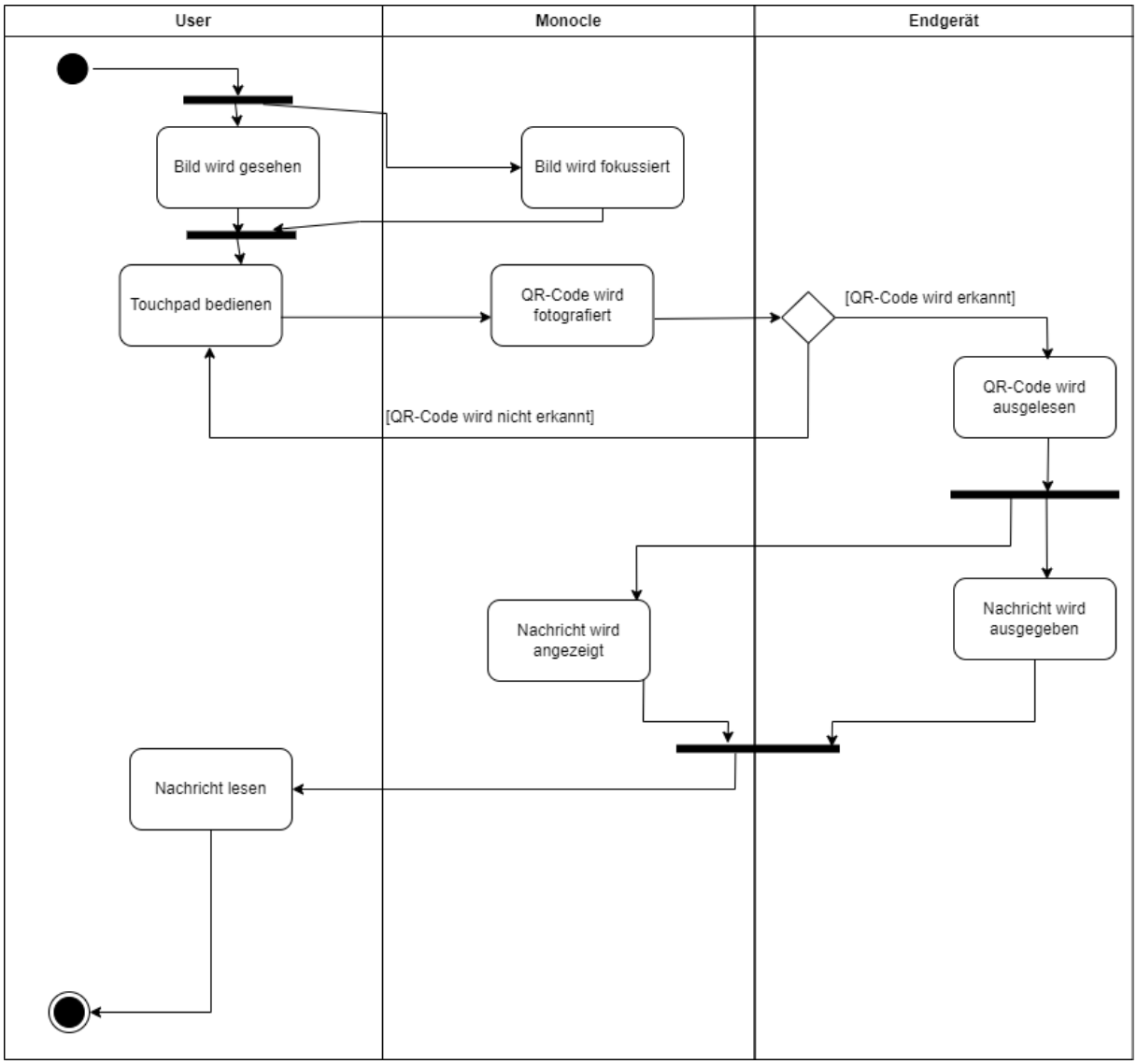

Abbildung 1.2: Aktivitätsdiagramm erkennen und anzeigen von QR-Codes

Der User bedient das Touchpad des Monocles um damit den QR-Code abzufotografieren. Ist das Bild verschwommen oder nicht lesbar, muss der QR-Code erneut abfotografiert werden. Wenn es jedoch auswertbar ist, wird das fotografierte Bild dem Host geschickt und daraufhin ausgelesen. Die Botschaft wird dann sowohl dem Monocle-Display angezeigt als auch dem Host-Endgerät. Anschließend kann die Nachricht dann vom Verwender gelesen werden.

1.1.3 Ver- und Entschlüsselung

Das Monocle kann verschiedene Standard-QR-Codes auslesen, jedoch können auch speziell verschlüsselte und versteckte QR-Codes ausgelesen werden. Für die verschlüsselten Botschaften wurde der Advanced Encryption Standard in der Betriebsart Cipher Block Chaining verwendet. Der Key wird dabei "hard gecoded". Da es sich bei der AES Verschlüsselung um eine symmetrische Verschlüsselung handelt, ist der Key für Ver- und Entschlüsselung derselbe. AES wird weltweit als sehr sicher angenommen, so kann eine vertrauchliche Verschlüsselung der Botschaft gewährleistet werden.

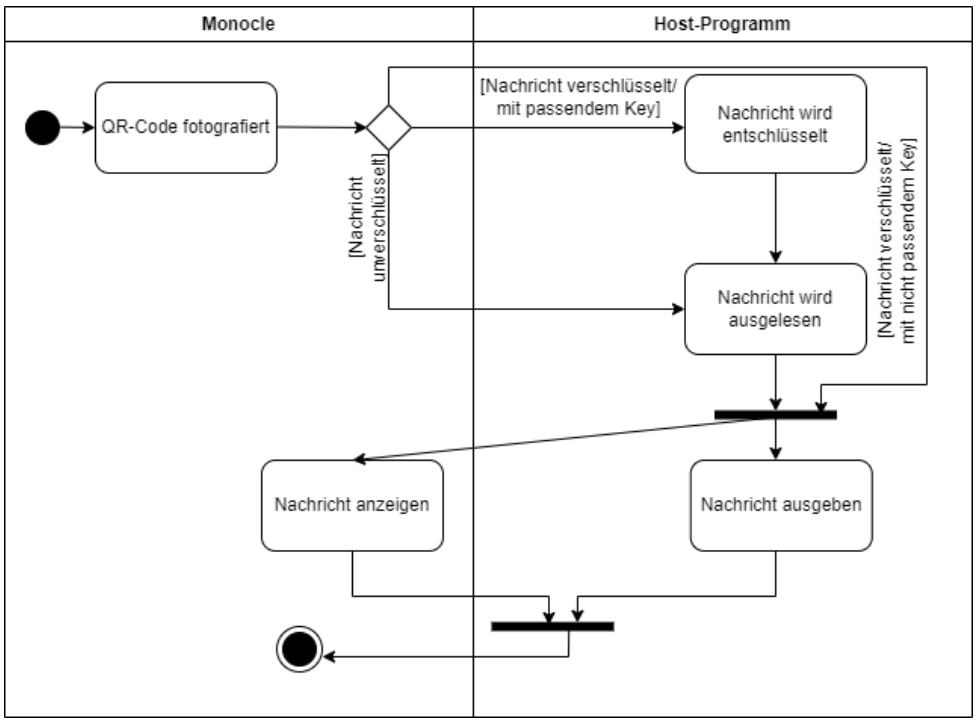

Abbildung 1.3: Aktivitätsdiagramm QR-Codes entschlüsselte Nachricht anzeigen

Die QR-Codes werden anhand des Monocle abfotografiert. Daraufhin entschlüsselt der Host die Nachricht und zeigt dann die Botschaft an. Das Monocle kann jedoch auch unverschlüsselte Botschaften anzeigen lassen.

2 Analyse der Produktfunktionen

In diesem Kapitel analysieren wir die im Pflichtenheft beschriebenen Funktionen und werden diese mithilfe von Sequenzdiagrammen visualisieren. Jede Funktion wird dabei auf ihre Anforderungen und ihre Rolle innerhalb der Gesamtsysteme untersucht. Diese Diagramme sollen dabei helfen, zu verstehen, welche Komponenten beteiligt sind und wie die einzelnen Schritte zwischen den verschiedenen Komponenten ablaufen.

2.1 Analyse von Funktionalität F10: Automatisches Ein- und Ausschalten

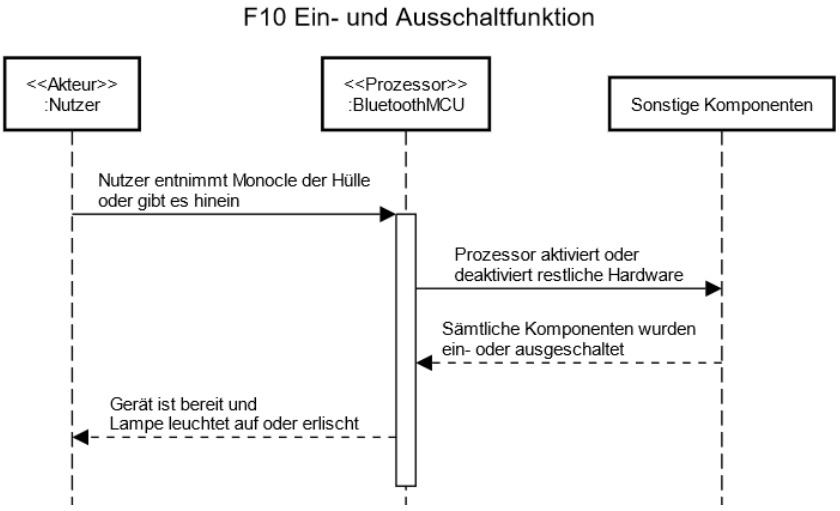

Abbildung 2.1: F10 als Sequenzdiagramm

Das Monocle-Gerät ist in der Lage sich automatisch ein- und auszuschalten. Um diese Funktionalität zu realisieren erkennt das Gerät per Sensor, ob es sich in dem für das Gerät vorgesehenen Gehäuse befindet. Bei dem Ein- und Ausschalten spielt sowohl der Nutzer und spätere Träger des Monocle eine Rolle, als auch der Prozessor, als zentrale Hardware, sowie weitere im Pflichtenheft aufgeführte Hardware-Komponenten. Entnimmt der Nutzer nun das Monocle-Gerät aus seinem Gehäuse, schlägt der dafür konzipierte Sensor aus und benachrichtigt den im Standby befindlichen Bluetooth-MCU-Prozessor. Dieser fährt daraufhin sämtliche Komponenten, wie

etwa die Omnivision OV 5640-Kamera hoch und wartet auf dessen Rückmeldung. Sind sämtliche Rückmeldungen bei dem Prozessor eingegangen, meldet der Prozessor die Bereitschaft des Monocle-Geräts über eine kleine Lampe außen am Prozessor zurück. Der Ausschalte-Vorgang verhält sich hierbei analog.

2.2 Analyse von Funktionalität F20: Kommunikation zwischen Endgerät und Monocle

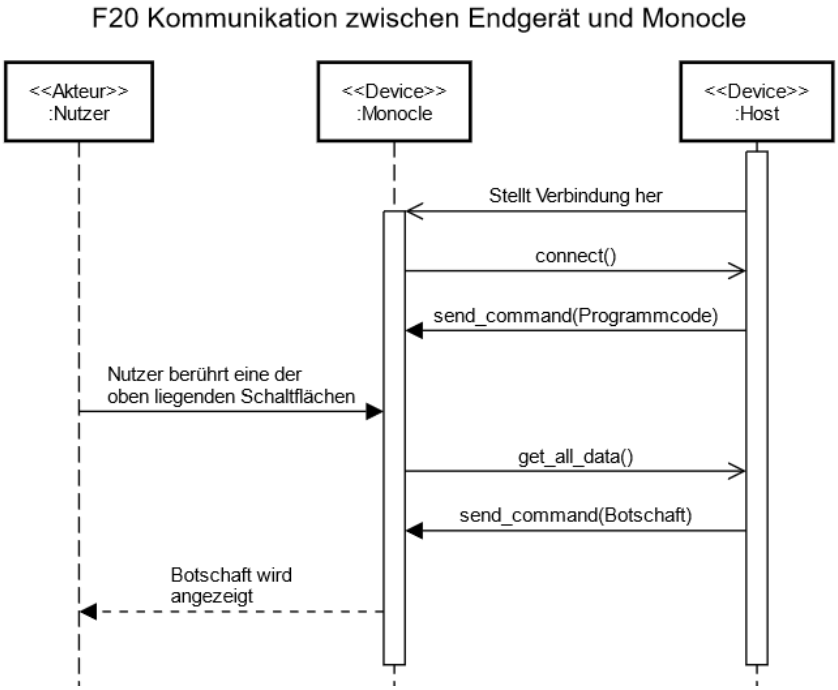

Abbildung 2.2: F20 als Sequenzdiagramm

Um den Anforderungen unserer Monocle-Anwendung gerecht zu werden, müssen das Monocle und das Host-Gerät beim Starten der Anwendung, sowie bei jedem Einlesevorgang in der Lage sein, Daten untereinander auszutauschen. An dieser Stelle kommt die im Pflichtenheft beschriebene Bluetooth Low Energy-Kommunikation (BLE) in das Spiel. Die Akteure der BLE-Kommunikation sind dabei wie folgt miteinander in Verbindung zu setzen: Sobald das Monocle-Gerät wie in 2.1 beschrieben hochgefahren ist und die Anwendung auf dem Host-Gerät ausgeführt wird, unternimmt der Host einen Verbindungsversuch zum Monocle und der auf dem Monocle auszuführende Programmcode wird mithilfe der Funktion „send_command()“ versucht zu übermitteln. Dieses erwidert bei funktionierender BLE-Kommunikation den so genannten Handshake mittels der asynchronen „connect()“-Funktion, wodurch die Verbindung erfolgreich hergestellt wird. Bei jeder weiteren Berührung durch eines der Touchpads auf dem Monocle, unternimmt dieses wie unten Beschrieben einen Einlesevorgang. Innerhalb dieses Einlesevorgangs kommunizieren Monocle und Host abermals miteinander, um erst das eingelesene Bild und später die entschlüsselte Nachricht miteinander auszutauschen. Dabei kommen die Funktionen „get_all_data()“ zum asynchronen Senden einer Fotoaufnahme an das Host-Gerät und „send_command()“ zum übermitteln der entschlüsselten Botschaft zum Tragen. Die übertragene Nachricht wird zuletzt auf dem OLED-Display des Monocle dem Träger angezeigt, wie Sie der

Analyse 2.4 entnehmen können.

2.3 Analyse von Funktionalität F30: Einlesen per Touchpads

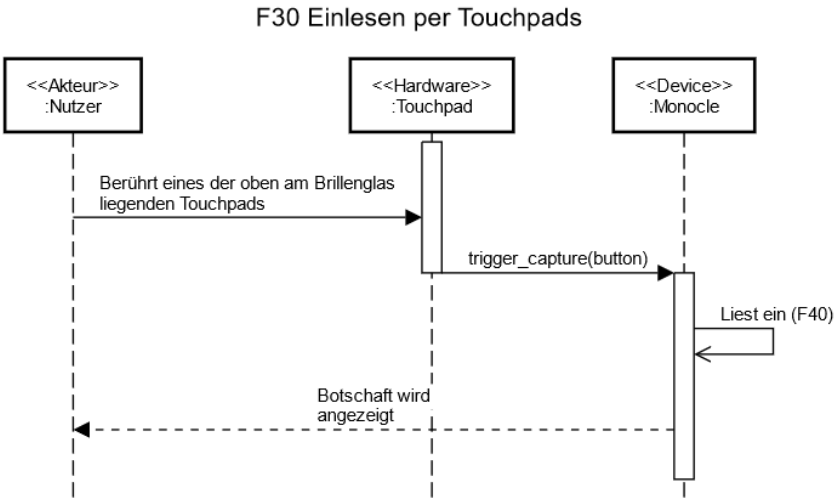

Abbildung 2.3: F30 als Sequenzdiagramm

Der in F40 beschriebene Einlesevorgang eines QR-Codes findet nur auf Nachfrage des Trägers statt. Dieser Kommuniziert der Anwendung seine Absicht über eines der beiden oberhalb des Brilenglases liegenden Azoteq IQS620A Touch Controllern. Hierzu müssen die Touchpads wie in F10 beschrieben von dem Prozessor des Monocle hochgefahren worden sein. Ein weiterer Akteur bei dem Einlesen des QR-Codes per Touchpad ist unsere auf dem Monocle-Gerät ausgeführte Anwendung. Diese wird per „trigger_capture(button)“-Funktion über die Berührung des Trägers über eines der Touchpads informiert. Zur Folge hat der Funktionsaufruf schließlich, dass der in Analyse 2.4 beschriebene asynchrone Einlesevorgang in Kraft tritt. Der mit der Berührung der Touchpads initialisierte Vorgang wird durch das Anzeigen der Nachricht im Display des Monocle abgeschlossen, wie Ihnen die Analyse der Funktionalität 2.6 erläutert.

2.4 Analyse von Funktionalität F40: Einlesen von QR-Codes

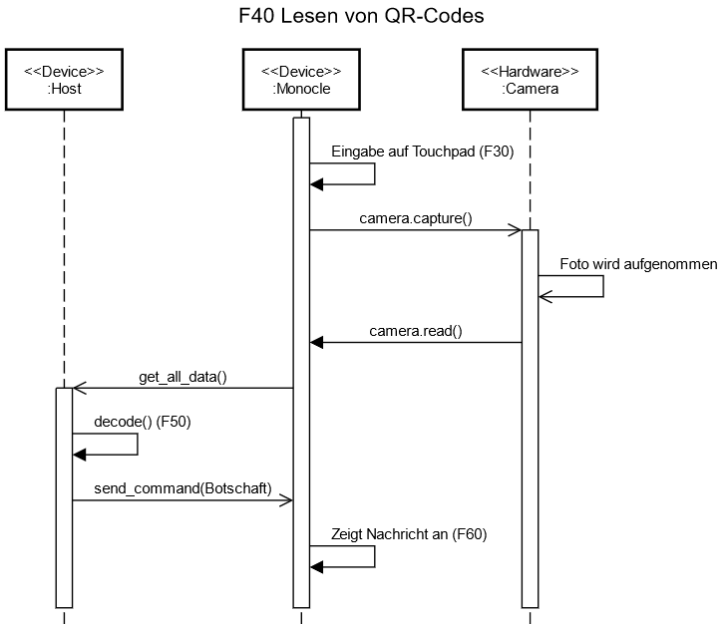

Abbildung 2.4: F40 als Sequenzdiagramm

Bei dem Einlesevorgang von QR-Codes durch das Monocle-Gerät handelt es sich um eine Kombination der vorangegangenen Funktionalitäten, dessen maßgebliche Akteure neben dem Träger auch das Host-Gerät, das Monocle selbst, sowie dessen Kamera sind. Das Verhalten des Monocle wird hierbei natürlich durch unsere Anwendung definiert. Empfängt das Monocle eine Eingabe auf einem der Touchpads, wie in Funktionalität 2.3 beschrieben, wird diese als Anfrage des Trägers zur Erkennung eines QR-Codes interpretiert. Unmittelbar nach der Berührung eines der Touchpads wird deshalb über die interne Funktion „capture()“ der Kamera, eine asynchrone Foto-Aufnahme derselben in Gang gesetzt. Ist das Foto aufgenommen, lässt es sich per „read()“-Funktion aus dem Zwischenspeicher der Kamera auslesen. Diese Foto-Aufnahme wird anschließend jedoch noch an das Host-Gerät per „get_all_data()“ asynchron weitergeleitet, wo die in Analyse 2.5 erläuterte Verschlüsselung durchgeführt wird. Zuvor wird die verschlüsselte Botschaft des QR-Codes über die „decode()“-Funktion aus der für die Extrahierung des QR-Codes vorgesehene Bibliothek entnommen. Die entschlüsselte Botschaft wird zuletzt von dem Host-Gerät an das Monocle zurückgegeben, welches die Botschaft gemäß Funktionalität 2.6 über das Display dem Träger des Geräts zugänglich macht. Genauer wird über die aus Funktionalität 2.2 bekannte „send_command()“-Funktion der gesamte Befehl zum Anzeigen einer Nachricht auf dem Monocle übermittelt

2.5 Analyse von Funktionalität F50: Entschlüsselung von Nachrichten

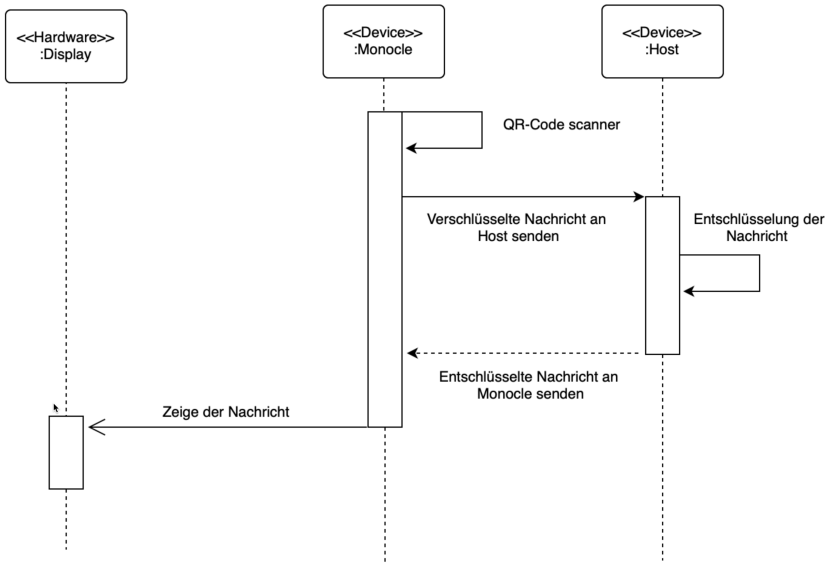

Abbildung 2.5: F50 als Sequenzdiagramm

Im Sequenzdiagramm (2.8) wird die Funktionalität beschrieben, wie Nachrichten, die im QR-Code versteckt sind, entschlüsselt werden. Zuerst sendet das Monocle das aufgenommene Bild an den Host. Der Host durchsucht das Bild nach einem QR-Code. Sobald er den QR-Code findet, entschlüsselt der Host ihn. Dieser Prozess umfasst das Scannen des QR-Codes, das Dekodieren der darin enthaltenen verschlüsselten Informationen mit unserer Entschlüsselungsfunktion und die Umwandlung dieser Daten in eine lesbare Nachricht. Es ist wichtig zu beachten, dass die Nachricht zuvor verschlüsselt und mithilfe unserer Verschlüsselungsfunktion im QR-Code eingebettet wurde. Das entschlüsselte Ergebnis wird dann zurück an das Monocle gesendet. Schließlich zeigt das Monocle die entschlüsselte Nachricht auf dem Display an.

2.6 Analyse von Funktionalität F60: Anzeigen der Nachrichten

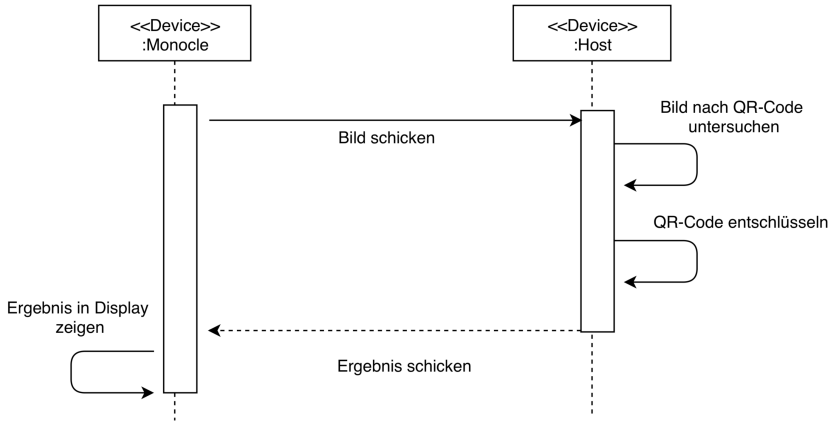

Abbildung 2.6: F60 als Sequenzdiagramm

Bei der Anzeige der Nachricht sollte bereits ein Foto aufgenommen und an den Host gesendet worden sein, wo die Nachricht entschlüsselt wird. Die entschlüsselte Nachricht wird anschließend dann an das Monocle übertragen. Wenn dieser Prozess korrekt durchgeführt wurde, zeigt das Monocle die entschlüsselte Nachricht auf dem Display an. Dieser Ablauf stellt sicher, dass die Nachricht zuverlässig und korrekt verarbeitet wird.

2.7 Analyse von Funktionalität F70: Funktionalität der Kamera

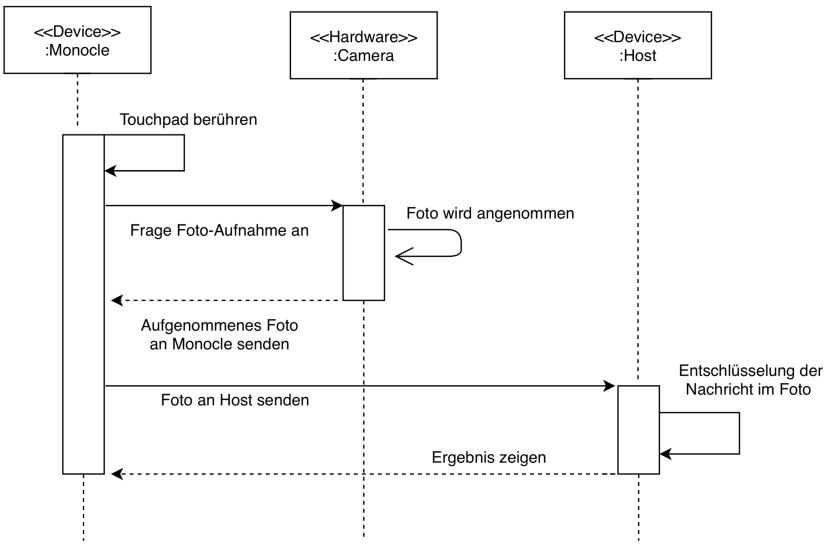

Abbildung 2.7: F70 als Sequenzdiagramm

Im Sequenzdiagramm (2.7) wird die Funktionalität der Kamera des Monocle beschrieben. Zuerst berührt man auf das Touchpad, um mit der ‘capture()‘-Funktion ein Foto aufzunehmen. Die Kamera nimmt dann ein Foto auf und sendet die Bilddaten an das Monocle. Das Monocle leitet diese Bilddaten anschließend an den Host weiter. Der Host verarbeitet das Bild und entschlüsselt die Nachricht darin. Wie in Funktionalität 2.4 beschrieben, sucht er nach einem QR-Code und dekodiert die Nachricht. Schließlich zeigt das Monocle das entschlüsselte Ergebnis an. Diese Reihenfolge von Aktionen stellt sicher, dass die Bilddaten korrekt verarbeitet werden und die enthaltenen Daten erfolgreich gelesen werden können.

2.8 Analyse von Funktionalität F80: AI Kunst erstellen

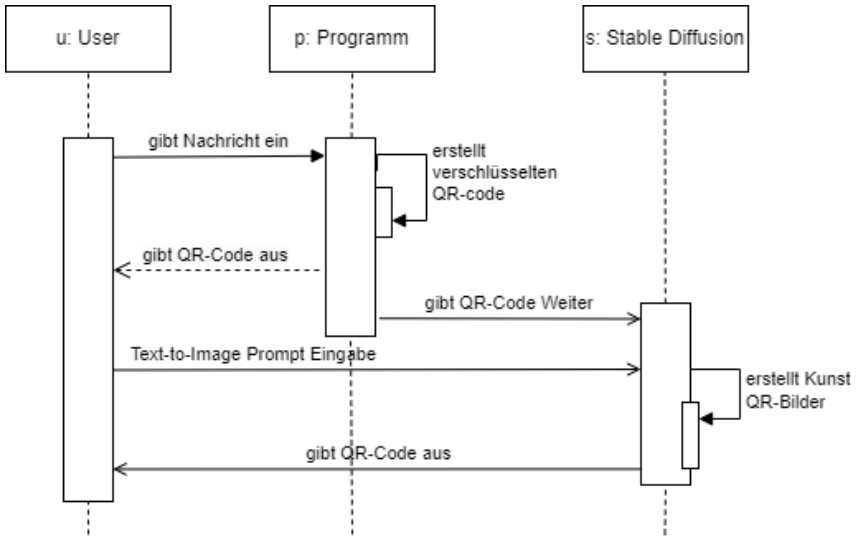

Abbildung 2.8: F80 als Sequenzdiagramm

Zuerst gibt der Benutzer eine Nachricht ein, die anschließend mit einem AES Algorithmus verschlüsselt wird. Aus der verschlüsselten Nachricht wird ein QR-Code generiert. Anschließend wird der QR Code verwendet, um mit Stable Diffusion AI-generierte Kunst zu erstellen. Ein ControlNet stellt sicher, dass die Merkmale des QR Codes erhalten bleiben und das generierte Bild noch als QR Code funktioniert.

3 Resultierende Softwarearchitektur

In diesem Abschnitt wird die resultierende Softwarearchitektur beschrieben, insbesondere die Komponenten, die an diesem Projekt beteiligt sind. Dabei liegt der Fokus auf den einzelnen Komponenten und ihrer Rolle im Gesamtsystem. Es wird auch erläutert, wie diese Komponenten zusammenarbeiten, um die Funktionen der Software zu realisieren. Ziel ist es, ein klares Verständnis der Architektur der Komponenten zu vermitteln.

3.1 Komponentenspezifikation

Im Folgenden werden die aus der Analyse der Produktfunktionen (Kapitel 2) resultierende Komponenten hinsichtlich ihrer Verteilung und Interaktion als Komponentendiagramm dargestellt und textuell erläutert.

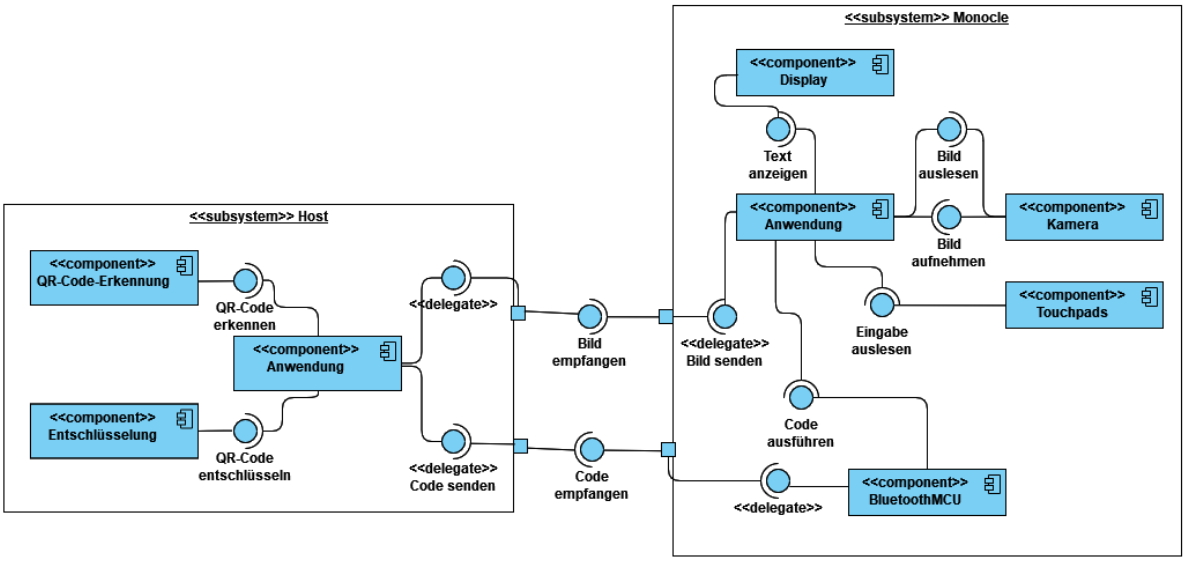

Abbildung 3.1: Komponentendiagramm für Monocle- und Hostgerät

Die Komponenten unserer Gesamt-Anwendung sind wie folgt zu benennen:

Komponente $\langle C10 \rangle$: Host-Anwendung
Bei der Host-Anwendung handelt es sich um ein Computerprogramm, welches auf einem Laptop oder Smartphone in einer Umgebung von höchstens zehn Metern um das Monocle-Gerät herum ausgeführt werden soll. Die Aufgaben des Programms sind zum Einen der Austausch mit dem Monocle und zum Anderen die Weiterverarbeitung von Informationen, die die Leistungsfähigkeit des Monocle überschreiten. Austausch bezieht sich hierbei auf das Senden von Programmcode an das Monocle und das Empfangen von Bildern, die durch das Monocle-Gerät aufgenommen wurden. Auch das Erkennen von versteckten QR-Codes in KI-Kunst zählt zu den Prozessen, die aufgrund ihres Leistungsanspruchs auf dem Host-Gerät passieren. Mehr dazu unten.

Komponente $\langle C20 \rangle$: QR-Code-Erkennung
Die QR-Code-Erkennung ist eine Prozedur innerhalb desselben Skripts wie die Host-Anwendung, die jedoch unter Zuhilfenahme einer Software-Bibliothek von dritten umgesetzt wurde und sich deshalb als separate Komponente innerhalb unserer Software-Architektur betrachten lässt. Sie kommt während des in $\langle C40 \rangle$ beschriebenen Einlesevorgangs ins Spiel und folgt unmittelbar auf die Auslesung der Bildaufnahme aus dem dafür vorgesehenen Zwischenspeicher. Die Aufgaben der QR-Code-Erkennung sind die für das Erkennen des QR-Codes hilfreiche Bild-Optimierung mittels Filtern, wie etwa einer Steigerung des Kontrastes, und die anschließende, algorithmische Suche nach dem QR-Code innerhalb der Bildaufnahme durch die dafür integrierte Bibliothek. Es folgt die im Folgenden beschriebene Entschlüsselung des Inhalts des eingelesenen QR-Codes.

Komponente $\langle C30 \rangle$: Entschlüsselung
Sollte es sich bei dem eingelesenen QR-Code um einen verschlüsselten Code handeln, so wird dies unmittelbar nach der in $\langle C20 \rangle$ ausgeführten QR-Code-Erkennung anhand seines Inhalts deutlich. Bei der für die Entschlüsselung gedachte Komponente handelt es sich um ein gesondertes, eigenständig von uns programmiertes Python-Programm, das nach der Erkennung des verschlüsselten QR-Code-Inhalts als solcher mit dem verschlüsselten Inhalt als Eingabe auf dem Host-Gerät ausgeführt wird. Die Aufgabe der Entschlüsselungs-Komponente besteht dann darin, die verschlüsselte Eingabe unter Betrachtung des fest im Host-Gerät hinterlegten Schlüssels mit dem im Fachentwurf beschriebenen AES-Verfahren in seine entschlüsselte Ursprungsform zurückzuführen.

Komponente $\langle C40 \rangle$: Monocle-Anwendung
Die Monocle-Anwendung ist eines eben dieser Programme, die in Form von Code von dem Host-Gerät per BLE an das Monocle gesendet werden, wo sie von dem BluetoothMCU-Mikroprozessor ausgeführt werden. Nachdem sie auf dem Monocle ausgeführt wird, verknüpft die Monocle-Anwendung die restlichen Funktionen der Hardware des Monocle-Systems. So wartet sie unter anderem auf Eingaben über die Touchpads ($\langle C50 \rangle$) des Monocle und initialisiert daraufhin den Einlesevorgang eines QR-Codes, indem sie eine Bildaufnahme bei der im Monocle verbauten Kamera anfragt und diese bei Fertigstellung aus dem dafür vorgesehenen Zwischenspeicher aus-

liest. Auch macht die Monocle-Anwendung dem Nutzer diesen Prozess über einen Text über das Display des Monocle kenntlich und steuert damit die Interaktion mit dem Träger des Monocle-Systems maßgeblich.

Komponente $\langle C50 \rangle$: BluetoothMCU
Bei dem BluetoothMCU handelt es sich um den im Monocle verbauten Mikroprozessor, der dessen zentrale Steuerungseinheit darstellt. Der in $\langle C10 \rangle$ beschriebene Austausch zwischen dem Monocle-System und dem Host-Gerät läuft damit über die durch den Mikroprozessor aufgebaute Bluetooth Low Energy-Verbindung ab. Außerdem ist der BluetoothMCU-Prozessor das Hardware-Element, auf das die Ausführung der oben beschriebenen Monocle-Anwendung zurückzuführen ist. Zuletzt stellt der Mikroprozessor auch die Schnittstellen zu den anderen Hardware-Komponenten des Monocle bereit, die von dessen Anwendung ($\langle C20 \rangle$) aufgerufen werden.

Komponente $\langle C60 \rangle$: Kamera
Die Kamera zählt zu den oben beschriebenen Hardware-Komponenten des Monocle, die durch den BluetoothMCU-Prozessor physisch und unter Zuhilfenahme der Monocle-Anwendung logisch vernetzt werden. Sobald der Träger des Monocle eine der oberhalb des Brillenglases liegenden Schaltflächen ($\langle C50 \rangle$) berührt, beginnt die Kamera eine Fotoaufnahme, aus der später der verschlüsselte QR-Code extrahiert werden soll. Bei Vollendung der Fotoaufnahme signalisiert die Kamera dies dem Mikroprozessor über die dafür vorgesehene Schnittstelle des Prozessors. Außerdem speichert die Kamera das aufgenommene Foto in einem temporären Zwischenspeicher, wo es von der Monocle-Anwendung ($\langle C20 \rangle$) ausgelesen werden kann und bei der nächsten Aufnahme mit dieser überschrieben wird.

Komponente $\langle C70 \rangle$: Touchpads
Ebenso wie die Kamera, fungieren auch die oben am Monocle angebrachten Touchpads A und B als Hardware-Komponente, dessen Nutzung über die Monocle-Anwendung durch den in $\langle C30 \rangle$ beschriebenen Mikroprozessor ermöglicht wird. Sowohl das linke, als auch das rechte der beiden Touchpads gilt bei der Nutzung unserer Anwendung als Auslöser für den Einleseprozess eines in KI-Kunst versteckten QR-Codes. Der Einleseprozess wird daraufhin mit einer Fotoaufnahme durch die im Monocle verbaute Kamera fortgeführt. Der Träger des Monocle-Systems wird über seine erkannte Eingabe über eines der Touchpads mittels einer Änderung der im Folgenden erläuterten Display-Komponente in Kenntnis gesetzt.

Komponente $\langle C80 \rangle$: Display
Bei der Display-Komponente des Monocle-Subsystems handelt es sich um die letzte der für unsere Anwendung erforderlichen Hardware-Bestandteile des Monocle-Geräts selbst. Wie auch die oben beschriebene Kamera des Monocle, sowie dessen beiden Touchpads, wird das Zusammenspiel der Hardware-Bestandteile des Monocle zentral durch die von uns dafür entworfene Monocle-Anwendung koordiniert und durch den im Monocle-Gerät verbauten Mikroprozessor ermöglicht. Genauer lässt sich das Display des Monocle als transparenten Balken in der unteren

Hälfte des Brillenglases des Monocle beschreiben, das in der Lage ist, Text und Symbole visuell für den Träger des Geräts zugänglich zu machen, ohne dessen Durchsicht in seine Umgebung zu unterbrechen. Konkret innerhalb unserer Anwendung übernimmt die Display-Komponente die Aufgabe Zustandswechsel während des Einlesevorgangs ($\langle C60 \rangle$) für den Träger des Monocle sichtbar zu machen. Die Monocle-Anwendung nutzt dabei die weiter unten beschriebene Schnittstelle, um der Display-Komponente am Ende jeder Phase des Einleseprozesses über eben diese Phase zu unterrichten. Wie jedoch eingangs erläutert, lassen sich sämtliche Anfragen der Monocle-Anwendung an das Display direkt auf die Host-Anwendung zurückführen, da diese die Monocle-Anwendung über die Übermittlung des Programmcodes an den Mikroprozessor überhaupt erst erzeugt.

3.2 Schnittstellenspezifikation

Näheres zu den Schnittstellen zwischen den oben beschriebenen Komponenten ist der unten liegenden Auflistung zu entnehmen. Hierbei wird das reine Annehmen und Übergeben von Informationen (Getter und Setter) vernachlässigt. Auch werden wir delegierende Schnittstellen auf die dazugehörigen funktionalen Schnittstellen beschränken.

Schnittstelle $\langle I10 \rangle$: QR-Code erkennen

Operation	Beschreibung
O11	Es liegt eine Fotoaufnahme als Bitstream vor, die vor der Suche nach einem darin enthaltenen QR-Code auf dieselbe mittels verschiedener Filter optimiert wird.
O12	Die Fotoaufnahme als Bitstream wird algorithmisch nach einem (versteckten oder offensichtlichen) QR-Code abgesucht. Wird dieser gefunden, wird sein Inhalt ausgelesen und an den Aufrufer der Operation zurückgegeben.

Schnittstelle $\langle I20 \rangle$: QR-Code entschlüsseln

Operation	Beschreibung
O21	Vorliegend ist eine verschlüsselte Zeichenkette, dessen Inhalt für den Menschen unlesbar ist. Diese wird daraufhin mit dem AES-Verfahren unter Zuhilfenahme des im Host-Gerät fest hinterlegten Schlüssels entschlüsselt und dessen für den Menschen nun verständlicher Inhalt an den Aufrufer zurückgegeben.

Schnittstelle $\langle I30 \rangle$: Bild empfangen

Operation	Beschreibung
O31	Das Host-Gerät wird darauf angewiesen, per Bluetooth auf den Eingang eines Bildes vom Monocle in Form eines Bitstreams zu warten. Bei Eingang des Bildes wird dieses in die dafür vorgesehene Variable geschrieben, ansonsten wird das Warten durch eine Zeitschranke unterbrochen.

Schnittstelle $\langle I40 \rangle$: Code empfangen

Operation	Beschreibung
O41	Kurz nach dem Funktionsaufruf wird eine Zeichenkette per Bluetooth an den Mikroprozessor des Monocle-Geräts gesendet, der den darin enthaltenen Code speichert und automatisch Operation O81 darauf ausführt.

Schnittstelle $\langle I50 \rangle$: Bild auslesen

Operation	Beschreibung
O51	Sollte seit dem letzten Start die Operation O61 augeführt worden sein, so befindet sich eine Fotoaufnahme der Kamera des Monocle-Geräts in dem dafür vorgesehenen Zwischenspeicher. Diese wird mittels dieser Operation als Bitstream ausgelesen und in eine Variable geschrieben, um sie für die weitere Verwendung nutzbar zu machen.

Schnittstelle $\langle I60 \rangle$: Bild aufnehmen

Operation	Beschreibung
O61	Fragt die Aufnahme eines Fotos durch die im Monocle-System verbaute Kamera an. Ruft bei Finalisierung der Aufnahme automatisch Operation O62 auf.
O62	Schreibt die von der Kamera aufgenommene Bildaufnahme in den von Mikroprozessor und Kamera des Monocle geteilten, temporären Zwischenspeicher, wodurch die Aufnahme mithilfe von Interface I50 ausgelesen werden kann.

Schnittstelle $\langle I70 \rangle$: Eingabe auslesen

Operation	Beschreibung
O71	Wird bei Berührung eines der oben am Brillenglas des Monocle-Systems liegenden Touchpads aufgerufen. Stellt im Allgemeinen die einzige Schnittstelle zum Träger des Monocle-Geräts dar. In unserer konkreten Anwendung wird O71 daher als Auslöser für den Einlesevorgang eines QR-Codes verwendet.

Schnittstelle $\langle I80 \rangle$: Code ausführen

Operation	Beschreibung
O81	Sobald Programmcode im Rahmen von Operation O41 bei dem im Monocle-Gerät verbauten Mikroprozessor eingegangen ist, führt dieser eine Syntax-Überprüfung durch ohne jedoch den Code dabei zu einem seperaten Programm zu kompilieren, wie es bei Python-Skripten üblich ist. Schließlich geht der Mikroprozessor automatisch zur unten beschriebenen Operation O82 über.
O82	Unmittelbar nach der obigen Aktion führt der Mikroprozessor den eingegangenen Code automatisch aus, bis das Monocle-System wieder deaktiviert wird oder neuer Programmcode mithilfe der Schnittstelle empfangen wird.

Schnittstelle $\langle I90 \rangle$: Text anzeigen

Operation	Beschreibung
O91	Übergibt der Display-Komponente des Monocle-Geräts einen Text als Zeichenkette, der daraufhin in gewünschter Größe und Farbe an der übergebenen Stelle auf dem physischen Display des Monocle-Systems angezeigt wird. Bis zum Aufruf von O92 verhalten sich Text- und Grafikelemente auf dem Display des Monocle persistent.
O92	Reinigt das Display des Monocle-Systems auf grafischer Ebene, wodurch sämtliche durch Operation O91 hinzugefügte Grafikelemente entfernt werden und somit für den Träger des Monocle-Geräts verschwinden.

3.3 Protokolle für die Benutzung der Komponenten

Im Folgenden befinden sich Zustandsdiagramme für die korrekte Verwendung und Arbeitsweise jeder im obigen Komponentendiagramm erfassten Komponente, sowie eine Beurteilung der Wiederverwendbarkeit.

3.3.1 Komponente C10: Host-Anwendung

Mit dem Ausführen der Host-Anwendung auf dem Host-Gerät beginnt der gesamte Programmablauf. Um den Programmcode für das Monocle-System auf diesem auszuführen, muss dessen Code per Bluetooth an das Monocle gesendet werden. Schlägt diese Bluetooth-Verbindung zu Beginn fehl, so schlägt auch die Gesamtanwendung fehl und muss bei funktionierender Verbindung neu gestartet werden. Entsteht ein Fehler bei dem Einlesevorgang eines QR-Codes, so kehren die Host-Anwendung und die Monocle-Anwendung in den Bereit-Zustand zurück, was für den Träger des Monocle-Systems lediglich eine erneute Anfrage bedeutet.

Für Komponente C10 eignet sich keine Wiederverwendung, da diese den gesamten Programmablauf von Host- und Monocle-Gerät steuert und damit nicht zu Aufruf als Schnittstelle in einem anderen Programm nützlich wäre.

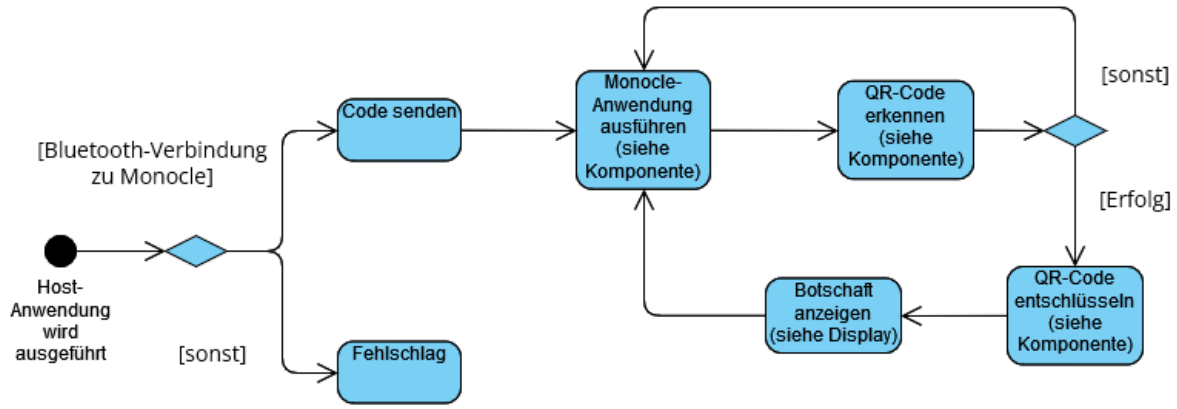

Abbildung 3.2: Zustandsdiagramm für C10

3.3.2 Komponente C20: QR-Code-Erkennung

Der kritische Abschnitt der QR-Code-Erkennung besteht in der algorithmischen Suche nach dem verschlüsselten QR-Code innerhalb des Bildes. Schlägt diese Suche fehl, etwa aufgrund mangelnder Bildqualität der Aufnahme, so gibt das Monocle-Gerät dem Nutzer die entsprechende Rückmeldung (siehe Fachentwurf) und kehrt in den Bereit-Zustand zurück, was dem Träger des Geräts die erneute Anfrage ermöglicht.

Die Komponente C20 eignet sich zur Wiederverwendung, da sie eine universelle Schnittstelle zur Erkennung von QR-Codes in Bildern darstellt und damit keineswegs auf die Verwendung im Zusammenhang mit dem von BrilliantLabs entwickelten Monocle angewiesen ist. Darüber hinaus arbeitet Komponente C20 wie in Kapitel 2 beschrieben ohnehin auf dem Host-Gerät, was die Wiederverwendung erleichtert.

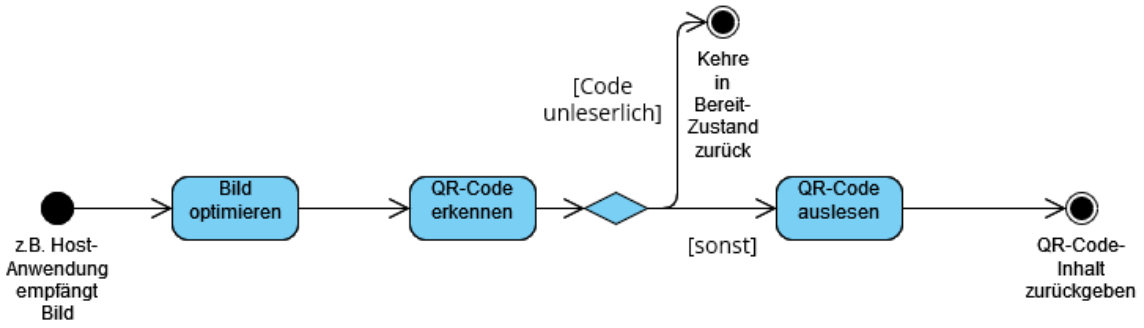

Abbildung 3.3: Zustandsdiagramm für C20

3.3.3 Komponente C30: Entschlüsselung

Die Entschlüsselung findet in der aktuellen Anwendung unmittelbar nach dem Einsatz von C20 statt und dient der Entschlüsselung der ausgelesenen Botschaft, um diese für den Träger des Monocle-Systems einsehbar zu machen. Durch die garantierte Ausführung der Entschlüsselung hinter C20 besteht das Risiko in dem Umgang mit Nachrichten, bei denen keine Entschlüsselung vorliegt oder, die mithilfe eines anderen Schlüssels verschlüsselt wurden. In der derzeitigen Anwendung wird C30 dennoch die Entschlüsselung auf die Botschaft anwenden, wodurch diese unter Umständen wieder unkenntlich gemacht würde.

Da es sich bei Komponente C30 um ein separates Programm handelt, das nur auf die Entschlüsselung einer Zeichenkette mithilfe des AES-Verfahrens und eines Schlüssels, ist auch bei Komponente C30 die Wiederverwendung sinnvoll.

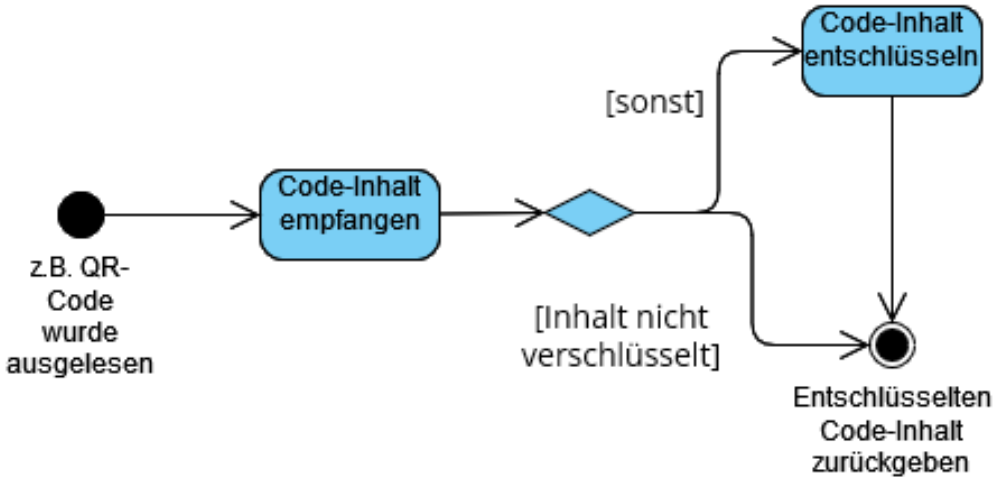

Abbildung 3.4: Zustandsdiagramm für C30

3.3.4 Komponente C40: Monocle-Anwendung

Die Monocle-Anwendung befindet sich stets im Einlesevorgang eines QR-Codes oder im passiven Bereit-Zustand, der bei Eingang einer Eingabe auf einem der Touchpads des Monocle unterbrochen wird. Sollte die Berührung des Trägers des Monocle-Systems nicht das dafür vorgesehene Touch-Ereignis auslösen, so wurde dessen Berührung nicht erkannt und die Monocle-Anwendung verlässt nicht den Bereit-Zustand. Für die korrekte Ausführung des Einlesevorgangs und das Verhalten im Fehlerfall, siehe Komponente C20.

Da es sich bei der Monocle-Anwendung wie auch bei Komponente C10 um das zentrale Koordinationssystem der Anwendung handelt, erscheint eine Wiederverwendung auch hier nicht sinnvoll. Bei jedem Aufruf der Komponente C40 würde es sich lediglich um eine Emulation der Anwendung handeln.

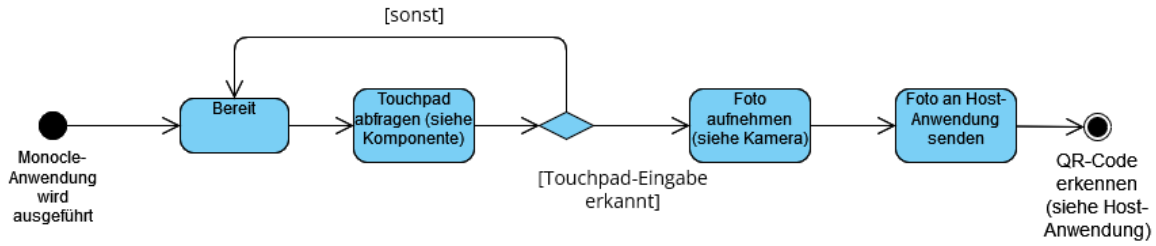

Abbildung 3.5: Zustandsdiagramm für C40

3.3.5 Komponente C50: BluetoothMCU

Der BluetoothMCU-Mikroprozessor bildet die zentrale Schnittstelle für alle Anfragen an das Monocle-Gerät und ist für die Ausführung der Monocle-Anwendung (C40) zuständig. Ist die Hardware des Monocle-Gerät in einem funktionstüchtigen Zustand, so sind die einzigen erwartbaren Fehler im Programmcode selbst zu ersuchen. Sollte dieser etwa Syntax-Fehler aufweisen, so wird dieser von dem BluetoothMCU nicht angenommen und das Monocle-System zeigt keine Reaktion auf eingehende Anfragen.

Die Wiederverwendung der Komponente C50 findet immer dann implizit statt, wenn neuen Anfragen an das Monocle-Gerät gestellt werden und ist damit sinnvoll.

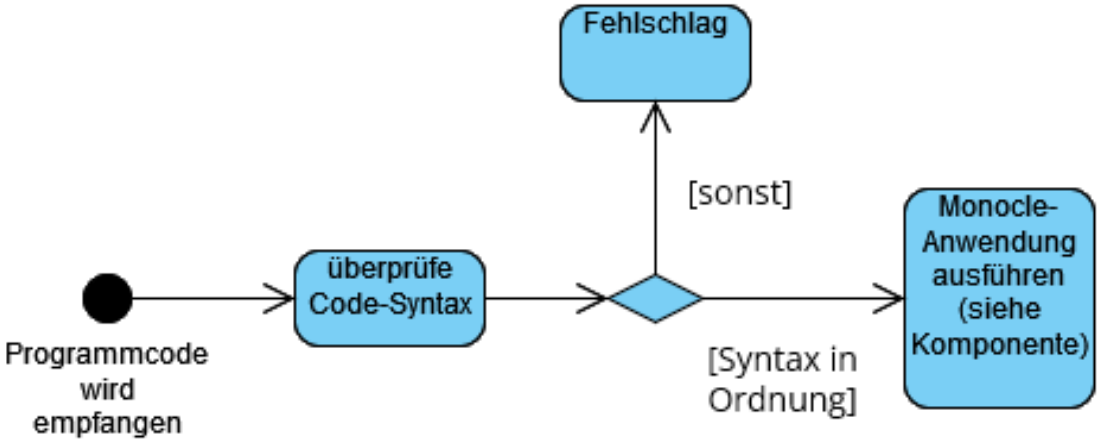

Abbildung 3.6: Zustandsdiagramm für C50

3.3.6 Komponente C60: Kamera

Da es sich bei den Komponenten C60, C70 und C80 um die funktionalen Hardware-Bestandteile des Monocle-Geräts handelt, sind unter der Prämisse, dass die Komponenten C10, C40 und C50 fehlerfrei funktionieren, keine Auffälligkeiten zu erwarten. Sobald das Monocle-System aktiv ist und Komponente C50 ohne Ausgabe von Fehlern arbeitet, so befinden sich auch Kamera, Touchpads und Display des Geräts im Bereit-Zustand und warten auf eingehende Anfragen, die nur aufgrund von physikalischer Dysfunktionalität der jeweiligen Komponente fehlschlagen können.

Die Wiederverwendung der Komponenten C60, C70 und C80 sind entsprechend sinnvoll, insbesondere bei Anwendungen, bei denen der Nutzer direkt mit dem Monocle-Gerät interagiert, wie etwa bei der vorliegenden Anwendung.

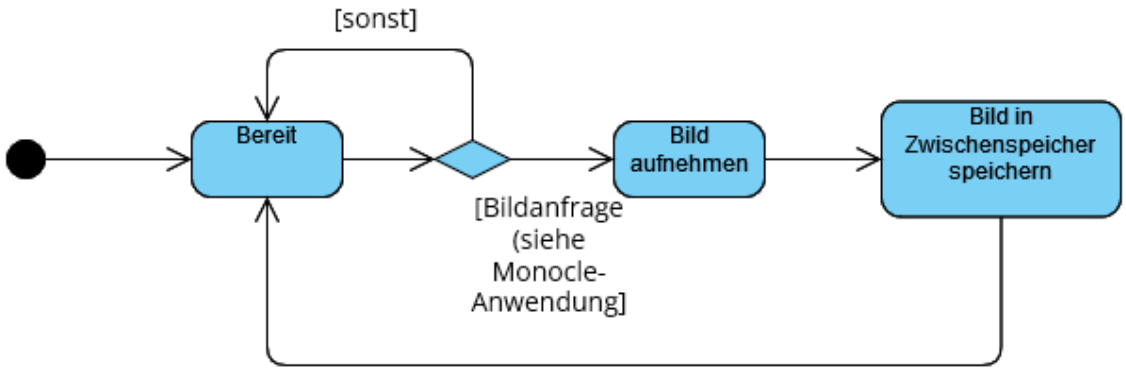

Abbildung 3.7: Zustandsdiagramm für C60

3.3.7 Komponente C70: Touchpads

Für eine Erläuterung der korrekten Funktionsweise und das Verhalten im Fehlerfall, sowie einer Erörterung der Wiederverwendbarkeit, siehe bitte Komponente C60.

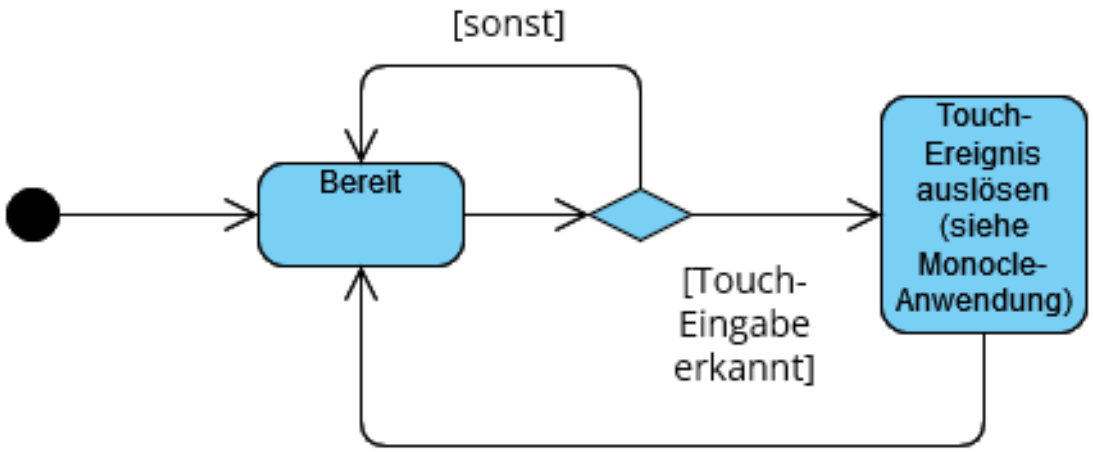

Abbildung 3.8: Zustandsdiagramm für C70

3.3.8 Komponente C80: Display

Für eine Erläuterung der korrekten Funktionsweise und das Verhalten im Fehlerfall, sowie einer Erörterung der Wiederverwendbarkeit, siehe bitte Komponente C60.

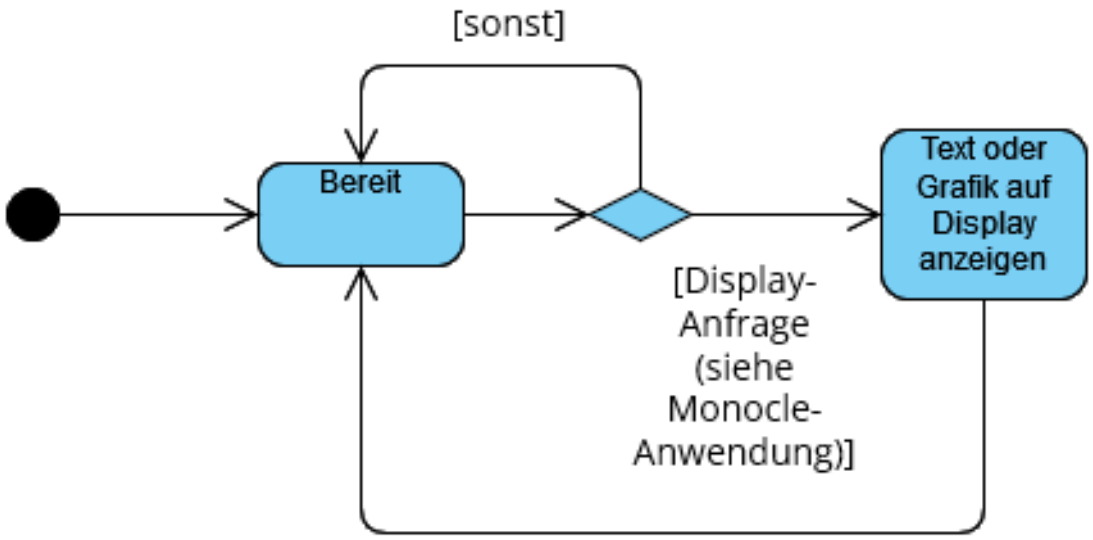

Abbildung 3.9: Zustandsdiagramm für C80

4 Verteilungsentwurf

Im folgenden Kapitel wird ein Verteilungsdiagramm zwischen Host und Monocle dargestellt und beschrieben.

Bei dem Projekt handelt es sich um ein verteiltes System, da das Monocle und der Host über ein Netzwerk via Bluetooth miteinander kommunizieren um gemeinsam eine Aufgabe zu erfüllen. Das verteilte System besteht aus zwei Hauptdevices, dem Monocle und dem Host, welcher ein handelsüblicher Laptop ist. Das Monocle besteht aus dem Display, Touchpad und der Kamera, dessen Hauptaufgabe das einlesen des QR-Codes ist. Die Kommunikation zwischen diesen Komponenten erfolgt über interne Bus-Systeme und Schnittstellen innerhalb des Monocle-Geräts. Die Hauptkommunikationsschnittstelle nach außen ist jedoch die Bluetooth-Verbindung zur Übertragung der QR-Code-Daten an den PC. Es läuft Software auf dem Moncle sowie als auch auf dem Host PC. Auf dem Host läuft Windows als Betriebssystem und Python 3 muss installiert sein. Die Software für das Monocle is ebenfalls in Python implementiert, welche in Zusammenhang mit dem Komponenten für die Hauptfunktion sorgt.

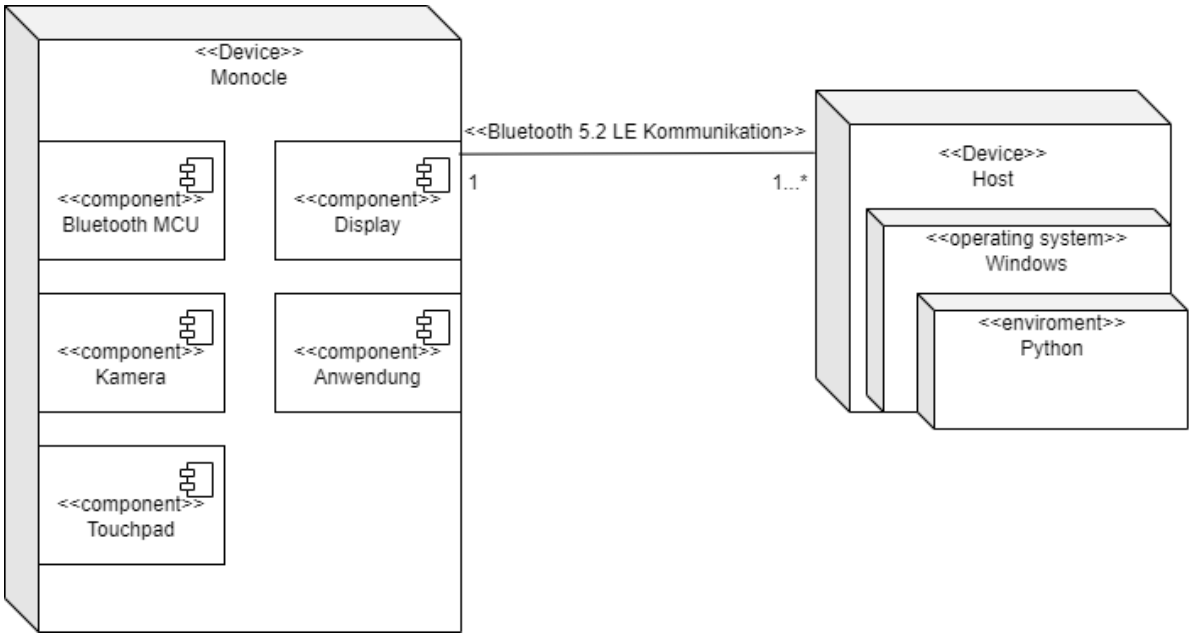

Abbildung 4.1: Verteilungsdiagramm des Projektes

5 Implementierungsentwurf

Dieses Kapitel des technischen Entwurfs beschreibt die verwendeten Klassen und Bibliotheken des Projektes. Die Komponenten sind zudem schon aus Kapitel 3 zu entnehmen. Diese werden in folgendem Abschnitt als Klassendiagramm dargestellt und nochmal schriftlich beschrieben.

5.1 Implementierung von Komponente <C10>: Host-Anwendung 1(main.py):

Die Komponente <C10> Host-Anwendung stellt viele der Basisfunktionalitäten für die Monocle. Da in unsere Code haben wir veschiedene Bibliotheken verwendet für die QR-Code-Erkennung, und davon sind Pyzbar, CV2, PIL. In unsere Remote-Skript haben wir auch Bibliotheken für die Monocle-Anwendung benutzt, wie bluetooth, camera, touch, display.

5.1.1 Paket-/Klassendiagramm

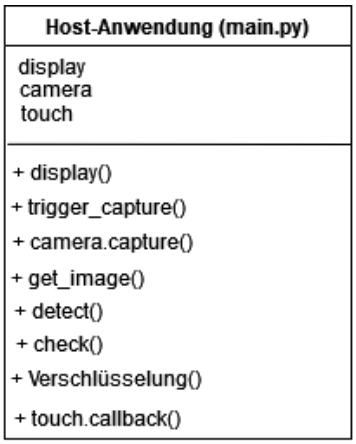

Abbildung 5.1: Darstellung von Komponente <C10>

In Abbildung 5.1 wird die Komponente Host-Anwendung 1 in einem Paketdiagramm dargestellt. Dieses besteht aus 8 Methoden, nämlich **diplay()**, **trigger_capture()**, **camera.capture()**, **detect()**, **get_image()**, **check()**, **entschlüsseln()**, **touch.callback()**.

5.1.2 Erläuterung

HostAnwendung1⟨$CL10$⟩

Aufgabe

Die Hostanwendung erkennt und dekodiert QR-Codes, rendert die Benutzeroberfläche, macht Bilder und ermöglicht dem Benutzer, auf verschiedene Weise mit dem Programm zu interagieren.

Attribute

display: Referenz auf das Display-Objekt zur Darstellung der Benutzeroberfläche.
camera: Referenz auf das Kamera-Objekt zur Aufnahme von Bildern.
touch: Referenz auf das Touch-Objekt zur Erkennung von Berührungseingaben.

Operationen

trigger_capture() und camera.capture(): Lösen die Bildaufnahme aus und ermöglicht es der Anwendung, Bilder von der Kamera zu erfassen.
display.show(): Zeigt wichtige Informationen und Statusmeldungen auf dem Display an.
get_image(): Stellt die Verbindung zum Monocle her und empfängt Bilder, die vom Monocle gesendet werden.
detect(): Analysiert die aufgenommenen Bilder, um QR-Codes zu erkennen und zu entschlüsseln
check(): Überprüft, ob der decodierte QR-Code eine gültige QR-Code / URL darstellt.
entschlüsseln(): Entschlüsselt die verschlüsselte, versteckte Nachricht.
touch.callback(): Ermöglicht dem Benutzer, ein neues Bild aufzunehmen und zu entschlüsseln, indem er das Touchpad berührt.

Kommunikationspartner

Monocle : Die Host-Anwendung kommuniziert mit der Monocle-Hardware, um die verschiedenen Aufgaben auszuführen.

5.2 Implementierung von Komponente <C20>: Host-Anwendung 2(brilliant.py):

Die Komponente <20> Host-Anwendung (brilliant.py) stellt die Verbindung und Kommunikation mit dem Monocle über BLE.

5.2.1 Paket-/Klassendiagramm

In Abbildung 5.2 wird die Komponente Host-Anwendung(brilliant.py) in einem Klassendiagramm dargestellt. Dieses besteht aus UUIDs, die die Kommunikation mit dem Monocle ermöglichen.

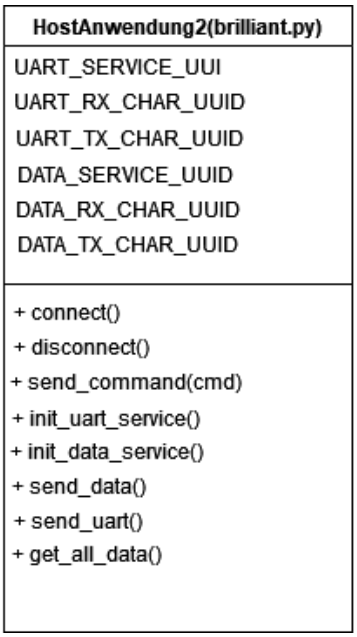

Abbildung 5.2: Darstellung von Komponente <C20>

5.2.2 Erläuterung

HostAnwendung2⟨$CL20$⟩

Aufgabe

In brilliant.py dient die Monocle-Klasse als Schnittstelle für die Bluetooth Low Energy (BLE)-Kommunikation mit dem Monocle. Sie ermöglicht Kommunikation, Datenaustausch und externe Steuerung für die Host-Anwendung.

Attribute

UART_SERVICE_UUID: UUID des UART-Dienstes für die serielle Kommunikation.
UART_RX_CHAR_UUID: UUID der Empfangscharakteristik* für UART.
UART_TX_CHAR_UUID: UUID der Übertragungscharakteristik* für UART.
DATA_SERVICE_UUID: UUID des Datenübertragungsdienstes.
DATA_RX_CHAR_UUID: UUID der Empfangscharakteristik für Datenübertragung.

DATA_TX_CHAR_UUID: UUID der Übertragungscharakteristik für Datenübertragung.

Operationen

connect() : Stellt eine Verbindung zum Monocle her.
disconnect(): Trennt die Verbindung zum Monocle.
send_command(cmd): Sendet einen Befehl an das Monocle.
init_uart_service(): Initialisiert den UART-Dienst.
init_data_service(): Initialisiert den Datenübertragungsdienst.
send_data(), send_uart() und get_all_data(): Diese Methoden sind für die Kommunikation mit dem Monocle verwendet.

Kommunikationspartner

Monocle : Die HostAnwendung2 ermöglicht die Verbindung und Kommunikationmit dem Monocle.

5.3 Implementierung von Komponente <C50>: BluetoothMCU:

BluetoothMCU handelt es sich um den im Monocle verbauten Mikroprozessor, der dessen zentrale Steuerungseinheit darstellt. Der in ⟨C10⟩ beschriebene Austausch zwischen dem Monocle-System und dem Host-Gerät läuft damit über die durch den Mikroprozessor aufge- baute Bluetooth Low Energy-Verbindung ab.

5.3.1 Paket-/Klassendiagramm

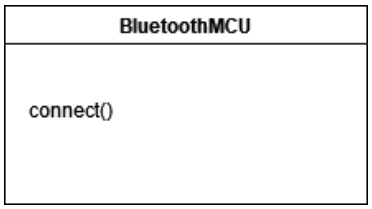

Abbildung 5.3: Darstellung von Komponente <C50>

In Abbildung 5.3 wird die Komponente BluettothMCU in einem Paketdiagramm dargestellt.

5.3.2 Erläuterung

BluetoothMCU⟨*CL*50⟩

Aufgabe

Die BluetoothMCU ist dafür zuständig, die Kommunikation zwischen dem Monocle und einem Endgerät zu ermöglichen.

Operationen

connect()

disconnect()

send_command(cmd)

Kommunikationspartner

Endgerät

5.4 Implementierung von Komponente <C60>: Kamera:

Kamera ist dafür zuständig Bildern von QR-Codes aufzunehmen.

5.4.1 Paket-/Klassendiagramm

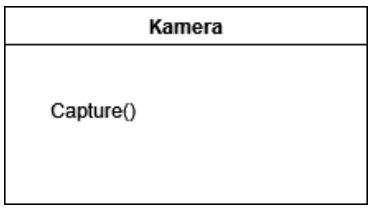

Abbildung 5.4: Darstellung von Komponente <C60>

In Abbildung 5.4 wird die Komponente Kamera in einem Paketdiagramm dargestellt.

5.4.2 Erläuterung

Kamera⟨*CL*60⟩

Aufgabe

Mit der Kamera kann mann die Qr-Codes erkennen und dekodieren.

Operationen

capture()

Kommunikationspartner

Monocle, Touchpad

5.5 Implementierung von Komponente <C70>: Touchpads:

Mit der Berührung von Touchpads, fängt das Monocle die Funktionalität für das Auslesen von Qr-Codes.

5.5.1 Paket-/Klassendiagramm

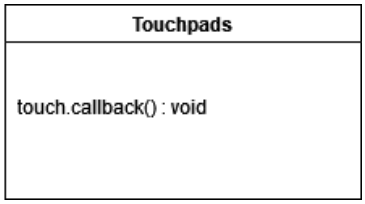

Abbildung 5.5: Darstellung von Komponente <C70>

In Abbildung 5.5 wird die Komponente Kamera in einem Paketdiagramm dargestellt.

5.5.2 Erläuterung

Touchpads$\langle CL70 \rangle$

Aufgabe

Mit dem Touchpads, kann man anfangen, die Qr-Codes zu erkennen und dekodieren.

Operationen

touch.callback()

Kommunikationspartner

Monocle

5.6 Implementierung von Komponente <C80>: Display:

Auf unser Display, wird die dekodierte Nachricht angezeigt.

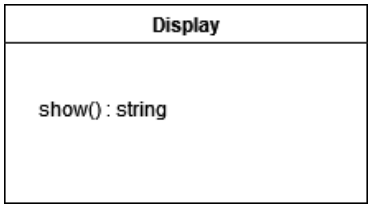

Abbildung 5.6: Darstellung von Komponente <C80>

5.6.1 Paket-/Klassendiagramm

In Abbildung 5.8 wird die Komponente Kamera in einem Paketdiagramm dargestellt.

5.6.2 Erläuterung

Touchpads$\langle CL80 \rangle$

Aufgabe

Mit dem Display, wird es dem nutzer ermöglichen, die Nachricht klaar, und vor den Augen gezeigt.

Operationen

display()

show()

Kommunikationspartner

Monocle

6 Datenmodell

Wie im Pflichteneheft bereits geschrieben, werden in diesem Projekt werden keine Daten dauerhaft gespeichert, die eine ausführliche Darstellung von Entitäten und deren Beziehungen erfordern würden. Das System speichert weder Nutzerdaten noch andere Informationen in relationalen Datenbanken, XML-Dateien oder ähnlichen Formaten.

Der Schwerpunkt des Projekts liegt auf der Erkennung und Dekodierung von QR-Codes in Bildern mithilfe von Augmented Reality, ohne eine persistente Datenspeicherung vorzunehmen. Alle verarbeiteten Informationen sind temporär und werden ausschließlich im Arbeitsspeicher gehalten, sodass sie nach erfolgreicher Entschlüsselung der QR-Codes unmittelbar genutzt und danach verworfen werden.

Die wesentlichen Funktionen unseres Projekts umfassen:

- Erkennung von QR-Codes in Bildern
- Dekodierung der QR-Codes
- Erstellung verschlüsselter QR-Codes zur Einbettung in Kunstwerke
- Integration von QR-Codes in AI-generierte Kunstwerke

Diese Funktionen erfordern keine dauerhafte Speicherung der Daten, da alle Schritte direkt im Arbeitsspeicher ablaufen und es keine Notwendigkeit gibt, die Daten nach der Verarbeitung zu bewahren.

Daher ist das Kapitel “Datenmodell“ in unserem Projekt nicht relevant.

7 Konfiguration

In diesem Abschnitt wird erläutert wie Konfiguration der Bearbeitung und der Anwendung vorgenommen wird. Da die Konfigurationen schon direkt im Code gechrieben sind, haben wir keine Separate config-Dateien. Unser Code besteht aus zwei Python-Dateien, und das heißt, dass wir zwei Konfigurationen haben:

- **Konfiguration für brilliant.py** : brilliant.py, ist ein Skript, das die Verbindung und die Kommunikation mit dem Monokel ermöglicht. Alle relevanten Einstellungen werden direkt im Code festgelegt und in Abbildung 4.1 dargestellt. Um die Konfiguration anzupassen, können Sie den Quellcode von "brilliant.py"mit einem Texteditor Ihrer Wahl öffnen, und da direkt bearbeiten. Die wichtige Parametern in brilliant.py sind:
  - **UART_SERVICE_UUID**: Die UUID des UART-Dienstes, der für die Kommunikation mit der Monocle verwendet wird.
  - **UART_RX_CHAR_UUID**: Die UUID des Empfangscharakteristikums für den Datenübertragungsdienst.
  - **UART_TX_CHAR_UUID**: Die UUID des Übertragungscharakteristikums für den Datenübertragungsdienst.
  - **DATA_SERVICE_UUID**: Die UUID des Datenübertragungsdienstes der Monocle.
  - **DATA_RX_CHAR_UUID**: Die UUID des Empfangscharakteristikums des Datenübertragungsdienstes.
  - **DATA_TX_CHAR_UUID**: Die UUID des Übertragungscharakteristikums des Datenübertragungsdienstes.
- **Konfiguration für main.py:** : Wie beim brilliant.py, main.py ist auch ein Skript, und die Konfiguration ist in der Code geschrieben. Main.py enthält die Steuerlogik und die anwendungsspezifischen Funktionen. Für die Anpassung der Konfiguration, ist es genau sowei bei brilliant.py, der Quellcode mit einem Texteditor öffnen, und die Konfiguration ändern. Die wichtigste Einstellungen sind auch im Code festgelegt und in Abbildung 4.2 erklärt.

```
# brilliant.py

class Monocle:
    UART_SERVICE_UUID = "6e400001-b5a3-f393-e0a9-e50e24dcca9e"
    UART_RX_CHAR_UUID = "6e400002-b5a3-f393-e0a9-e50e24dcca9e"
    UART_TX_CHAR_UUID = "6e400003-b5a3-f393-e0a9-e50e24dcca9e"

    DATA_SERVICE_UUID = "e5700001-7bac-429a-b4ce-57ff900f479d"
    DATA_RX_CHAR_UUID = "e5700002-7bac-429a-b4ce-57ff900f479d"
    DATA_TX_CHAR_UUID = "e5700003-7bac-429a-b4ce-57ff900f479d"
```

Abbildung 7.1: Darstellung

```
# "Waiting" anzeigen auf dem Display
text = display.Text('Waiting...', 100, 0, display.WHITE, justify=display.TOP_LEFT)
display.show(text)

# Auslösen der Bildaufnahme
def trigger_capture(button):
    # Maximale Länge für Bluetooth-Daten
    len = bluetooth.max_length()
    # "Capturing" anzeigen, wenn ein Bild erfasst wird
    text = display.Text('Capturing...', 100, 0, display.WHITE, justify=display.TOP_LEFT)
    display.show(text)
    # Erfassen eines Bildes mit der Kamera
    camera.capture()
    time.sleep_ms(100)
    # Lesen und Senden der Daten mit der Hilfe von Bluetooth
    while data := camera.read(bluetooth.max_length() - 4):
        led.on(led.GREEN)
        while True:
            try:
                # Senden der Daten über Bluetooth
                bluetooth.send((b"img:" + data)[:len])
            except OSError:
                continue
            break
    led.off(led.GREEN)
    # Senden einer Abschlussnachricht über Bluetooth
    bluetooth.send(b'end:')
    # "Done" auf dem Display anzeigen wenn alles fertig ist
    done = display.Text('Done', 100, 0, display.WHITE, justify=display.TOP_LEFT)
    display.show(done)
# Callback-Funktion für das Auslösen der Bildaufnahme bei Berührung
touch.callback(touch.EITHER, trigger_capture)
```

Abbildung 7.2: Darstellung

8 Änderungen gegenüber Fachentwurf

In diesem Kapitel wurden vorgenommene Änderungen, die gegenüber dem Fachentwurf geschehen sind, beschrieben.

Es fehlten im Fachentwurf die Verschlüsselung und das Generieren von QR-Code Kunst als Produktfunktion.

Um einen in Kunst eingebetteten QR-Code zu erhalten, wird eine Nachricht ins Programm übergeben. Dieser verschlüsselt die Nachricht und generiert daraufhin einen QR-Code. Der QR-Code wird dem User und der Software Stable Diffusion weitergegeben. Für die Erstellung der Kunst benötigt Stable Diffusion eine Eingabe von Prompts und unterschiedlichen Parametern des Verwenders. So kann Stable Diffusion die verstecken QR-Codes erstellen und dem User ausgeben.

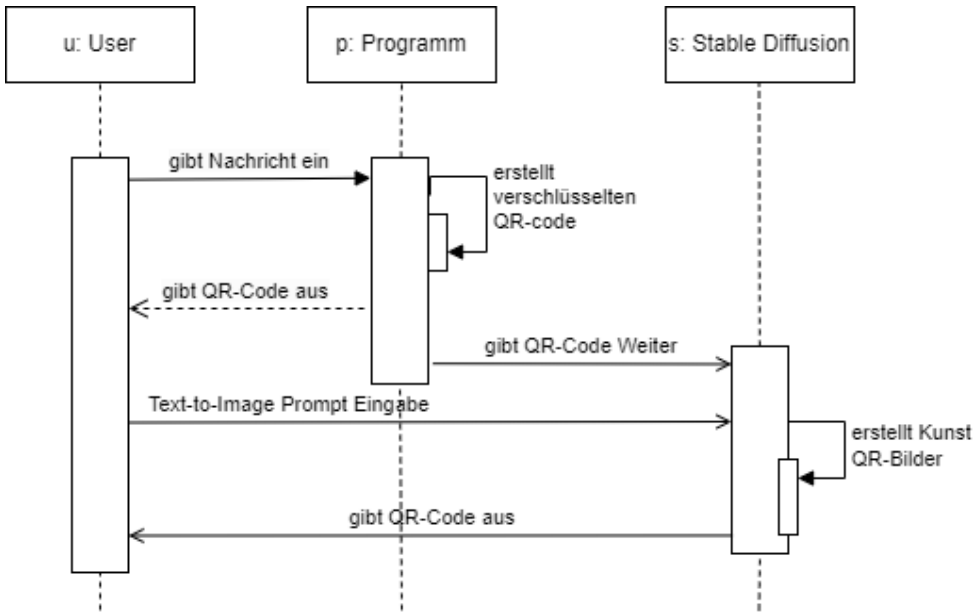

Abbildung 8.1: F80 als Sequenzdiagramm

9 Erfüllung der Kriterien

In diesem Kapitel werden die Kriterien des Pflichtenhefts erneut betrachtet, um den Fortschritt unseres Projekts ARID - Augmented Reality in Disguise zu ermitteln. Zusätzlich werden die verschiedenen Komponenten erläutert, die zur Erreichung der gesetzten Ziele beitragen.

9.1 Musskriterien

Die folgenden Kriterien sind unabdingbar und müssen durch das Produkt erfüllt werden:

<RM1> Erkennung: Das Monocle muss QR Codes in Bildern erkennen können wird durch Komponente <C20> QR-Code Erekknung umgesetzt.
<RM2> Dekodierung: Die Software muss die QR Codes dekodieren können wird durch Komponente <C30> Entschlüsselung umgesetzt.
<RM3> Verschlüsselung: Software zum Erstellen verschlüsselter QR Codes wird durch Funktion F80 umgesetzt.
<RM4> Die QR Codes sollen in Kunst eingebettet sein wird durch die Software Stable Diffusion XL von StabilityAi umgesetzt. Stable Diffusion ist ein deep learning, text-to-image Model, mit der Möglichkeit Controlnets zu nutzen, die in unserem Fall sicherstellen, dass der QR Code weiterhin als solcher gelesen werden kann.

9.2 Sollkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

<RS1> Erkennen, wenn ein QR Code gescannt wurde, aber nicht entschlüsselt werden kann wird teilweise durch die Komponente <C30> umgesetzt. Mit unserer Verschlüsselung erstellte QR Codes werden erkannt und entschlüsselt. Andere QR Codes werden ausgelesen und dargestellt, aber es findet keine Überprüfung statt ob es sich um eine fremde Verschlüsselung handelt.
<RS2> Die entschlüsselten Nachrichten sollten diskret und ansprechend dargestellt werden wird durch Komponente <C80> Display umgesetzt.

<RS3> Die Aufnahme der Bilder sollte im Rahmen der begrenzten Kamera möglichst Robust und zuverlässig gestaltet werden wird durch Komponente <C60> Kamera umgesetzt.

9.3 Kannkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

<RC1> QR Code ist nicht für das bloße Auge als solches erkennbar von der Berwertung dieser Funktion sehen wir ab, da diese schwer objektiv zu beurteilen ist. Als limitierende Faktor für die Stenografie hat sich die Aufnahmequalität der 5 Megapixel des Monocle herausgestellt.

10 Glossar

(Alphabetisch und absteigend sortiert)

AES - Steht für Advanced Encryption Standard, welches eine Blockchiffre ist, die zum Ver- und Entschlüsseln von Informationen verwendet wird.

BLE-Kommunikation - Bluetooth Low Energy-Kommunikation ist eine Technik zur drahtlosen Vernetzung von Hardware, die auf eine Reichweite von unter 10 Metern ausgelegt ist.

Empfangscharakteristik - Bezieht sich auf die spezifischen Eigenschaften eines Systems oder einer Komponente, die bestimmen, wie Daten empfangen werden.

IAS - Institut für Anwendungssicherheit an der TU Braunschweig.

Komponentendiagramm - UML-Diagramm zur Kenntlichmachung der logischen Komponenten einer Software, sowie dessen für andere Komponenten zur Verfügung gestellten Schnittstellen.

MicroPython - ist eine Implementierung der Programmiersprache Python, die speziell für Mikrocontroller und eingebettete Systeme entwickelt wurde.

QR-Code - Kurz für Quick Response, ist ein zweidimensionaler Strichcode, welcher Informationen in Form von Zeichen und Ziffern speichert.

UUID - Steht für Universally Unique Identifier (universell eindeutiger Bezeichner)

UART - Steht für Universal Asynchronous Receiver-Transmitter.

Übertragungscharakteristik - Bezieht sich auf die spezifischen Eigenschaften eines Systems oder einer Komponente, die bestimmen, wie Daten übertragen werden.

Zustandsdiagramm - UML-Diagramm, das sich auf die Zustände eines Produkts oder dessen

Software konzentriert, sowie deren Übergänge und Subzustände.