

Masterarbeit

ENTWICKLUNG EINES EMULATORS FÜR DAS AUTOMATISIERTE MÜNZVERARBEITUNGSSYSTEM IN SPIELAUTOMATEN

ausgeführt am



Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von

Ing. Bernd Schuster, BSc

1910322008

betreut und begutachtet von

FH-Prof. Dipl.-Ing. Dieter Lutzmayr

Graz, im Februar 2021



.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.



.....
Unterschrift

DANKSAGUNG

An dieser Stelle bedanke ich mich recht herzlich bei all denjenigen, die mich bei der Anfertigung dieser wissenschaftlichen Arbeit sowie während des gesamten Studiums unterstützt und motiviert haben.

Ein ganz besonderer Dank gebührt meiner Partnerin Bettina sowie unseren beiden Söhnen Janek und Niklas. Ich bedanke mich bei euch für die entgegengebrachte Unterstützung und das viele Verständnis für die Zeit, die ich auf Grund des Studiums nicht mit euch verbringen konnte. Ich freue mich schon darauf, diese Zeit gemeinsam mit euch nachzuholen.

Ich danke außerdem meinen Eltern, Freunden und allen anderen aus meinem Familienkreis, die mich während der Studienzeit moralisch unterstützt haben.

Weiters gebührt mein Dank Herrn FH-Prof. Dipl.-Ing. Dieter Lutzmayr, der diese Masterarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen, die fachliche Betreuung und das konstruktive Feedback bei der Erstellung dieser Arbeit bedanke ich mich herzlich.

Ebenfalls ein großer Dank an Herrn Ing. Wolfgang Schmoltner sowie dem gesamten Technology Team des Unternehmens IGT Austria GmbH, die mir mit viel Rat und Tat zur Seite standen. Ich bedanke mich vor allem für die Idee zu dieser Arbeit sowie die ständige Hilfsbereitschaft, die maßgeblich dazu beigetragen hat, dass diese Masterarbeit in dieser Form vorliegt.

KURZFASSUNG

Das Unternehmen IGT Austria GmbH entwickelt vorwiegend Softwarelösungen für Glücksspielautomaten in regulierten Märkten weltweit. In einigen dieser Märkte spielt konventionelle Münzverarbeitung eine zentrale Rolle. Somit ist es im Rahmen von Softwareentwicklungsprozessen unumgänglich Tests in Verbindung mit dem automatisierten Münzverarbeitungssystem durchzuführen. Schwierigkeiten ergeben sich in diesem Kontext vor allem in Anbetracht der länderspezifischen Währungen, weil es an Testmünzen oder passenden Bezahlssystemkomponenten fehlt.

Die vorliegende Masterarbeit befasst sich daher mit der Systementwicklung eines Emulators, der die grundsätzliche Funktionsweise realer Komponenten des Münzverarbeitungssystems nachbildet. In dieser Intention wird zunächst das Gesamtsystem der automatisierten Münzverarbeitung sowie die Funktionsweise und Interaktion der Einzelkomponenten analysiert. Unter Berücksichtigung der Grundsätze und Methodiken des Requirements-Engineering werden anschließend konkrete Anforderungen an den zu entwickelnden Emulator ermittelt und dokumentiert.

Die Anforderungsspezifikation dient als Ausgangspunkt für konzeptuelle Überlegungen in Bezug auf die Umsetzung. Resultierend aus diesen Überlegungen wird eine geeignete Zielplattform definiert sowie eine Softwarearchitektur entworfen, die durch Erweiterbarkeit und Wartungsfreundlichkeit überzeugt. Nach anschließender Implementierung der Architektur auf Basis plattformübergreifender Entwicklungsansätze, wird der finale Prototyp des Emulators unterschiedlichen Funktionstests unterzogen und evaluiert. Die Tests haben gezeigt, dass der Prototyp funktioniert und die grundlegenden Funktionen erfüllt, allerdings noch weitere Evaluierungen in Richtung Plattform-Abhängigkeit der Software durchzuführen sind.

ABSTRACT

The company IGT Austria GmbH primarily develops software solutions for slot machines across regulated markets around the globe. Some of these markets still focus on conventional coin handling systems. Within the scope of software development processes it is essential to perform tests in conjunction with the automated coin handling system. The central problem in this context is linked to country-specific currencies and occurs due to a lack of suitable test coins or payment system components.

This paper focuses on the system development of an emulator, that emulates the basic mode of operation of real coin handling components. For this purpose, the entire system for automated coin handling as well as the functionality and interactions between individual components are investigated. Taking into consideration the principles and methodologies of the requirements engineering process the requirements for the emulator that is being developed are identified and documented.

The requirements specification serves as the foundation for conceptual ideas in regard to the realization. Resulting from these ideas a suitable target platform is specified, and an extensible and maintainable software architecture is created. After implementing the architecture considering the rudiments of cross-platform development processes, the emulator prototype is verified and evaluated by various functionality tests. These tests have demonstrated that the basic functionality is fulfilled by the prototype, but additional investigations towards platform dependency of the software are required.

INHALTSVERZEICHNIS

1	Einleitung.....	1
1.1	Das Unternehmen IGT.....	1
1.2	Ausgangssituation	2
1.3	Aufgabenstellung und Untersuchungsinteresse	2
1.4	Zieldefinition.....	3
1.5	Aufbau der Arbeit.....	3
2	Das Automatisierte Münzverarbeitungssystem.....	4
2.1	Die Glücksspielbranche	4
2.2	Systemaufbau und Überblick.....	6
2.2.1	Münzprüfung.....	11
2.2.1.1	Münzen und deren Prüfkriterien	12
2.2.1.2	Elektronischer Münzprüfer.....	14
2.2.1.3	Münzsortierer.....	16
2.2.2	Münzauszahleinheit	17
2.2.2.1	Aufbau	17
2.2.2.2	Funktionsweise	18
3	Ermittlung der Anforderungen	19
3.1	Requirements-Engineering	19
3.1.1	Der Zweck von Anforderungen	19
3.1.2	Die Haupttätigkeiten des Requirements-Engineerings.....	19
3.1.3	Einteilung von Anforderungen	23
3.2	Analyse des automatisierten Münzverarbeitungssystems	24
3.2.1	Abbild der Systemumgebung.....	24
3.2.2	Hardwareseitige Analyse	25
3.2.2.1	Modelle von Münzprüfern und Münzauszahleinheiten.....	25
3.2.2.2	ccTalk-Hardware-Schnittstelle.....	26
3.2.3	Softwareseitige Analyse	28
3.2.3.1	ccTalk-Protokoll	28
3.2.3.2	Spielplattform-Software	33
3.3	Definition der Anforderungen an den Emulator	39
3.3.1	Erkenntnisse aus der Systemanalyse	39
3.3.2	Anforderungen an den Emulator.....	39
3.3.2.1	Funktionale Anforderungen	40
3.3.2.2	Qualitätsanforderungen	40
3.3.2.3	Randbedingungen	41
4	Aufbau der Hardware	42
4.1	Wahl der Zielplattform für den Emulator.....	42
4.2	Bussystemanbindung	44
4.2.1	Normung der Steckverbindungen	44

4.2.2	Realisierung der Hardware-Schnittstelle	45
4.3	Ein- und Ausgabegeräte	46
5	Konzeptionierung und Implementierung der Software	47
5.1	Erste Überlegungen	47
5.1.1	Das .NET 5-Framework	47
5.1.2	Plattformübergreifende Entwicklung	48
5.1.3	ASP-Webanwendung vs. Konsolen Applikation	49
5.2	Basissystemkomponenten in der Softwarearchitektur	50
5.2.1	Das Event-Aggregator-Entwurfsmuster	50
5.2.2	NuGet	51
5.2.3	JSON	52
5.2.4	NLog	53
5.2.5	Serial-Port-Klasse	53
5.2.6	ccTalk-Bibliothek	54
5.3	Entwicklung der Emulator-Software	54
5.3.1	Aufbau der Softwarearchitektur	54
5.3.2	Die Startup-Klasse	55
5.3.3	Event Aggregator API	56
5.3.4	Konfigurations-Manager	58
5.3.4.1	JSON-Konfigurationsdatei	58
5.3.4.2	Laden der Systemkonfiguration	59
5.3.5	Serieller Bustreiber	62
5.3.5.1	ReadSerialPortData-Methode	62
5.3.5.2	OnEvent-Methode (SendCcTalkCommand)	63
5.3.5.3	Echo-Filterung	64
5.3.6	Geräte-Manager	65
5.3.6.1	Konfigurieren der Geräte	65
5.3.6.2	OnEvent-Methode (ReceivedCcTalkCommand)	66
5.3.6.3	OnEvent-Methode (InsertedMoneyEvent)	67
5.3.7	ccTalk-Bibliothek	67
5.3.7.1	Aufbau der Bibliothek	67
5.3.7.2	Emulation der Peripheriegeräte	68
5.3.7.3	Verarbeitung von ankommenden Polls	69
5.3.7.4	Verarbeitung von Münzeinwurf-Ereignissen	70
5.3.7.5	Verarbeitung von Münzauszahlungs-Ereignissen	70
5.3.8	Konsolen-Manager	72
5.3.8.1	Eingabehandler	72
5.3.8.2	OnEvent-Methode (PayoutMoneyEvent)	74
5.3.9	Logging-Service	74
5.4	Darstellung der programmtechnischen Abläufe bei Events	76
5.4.1	ReceivedCcTalkCommand und SendCcTalkCommand	76

5.4.2	InsertedMoneyEvent.....	77
5.4.3	PayoutMoneyEvent.....	78
6	Evaluierung des PrototypS.....	79
6.1	Entwicklungsbegleitendes Testen	79
6.1.1	Realisierung des Prüfaufbaus.....	79
6.1.2	Eingesetzte Zusatzprogramme.....	80
6.1.3	Funktionstest einzelner ccTalk-Header	82
6.2	Evaluierung des Emulators im Spielautomaten.....	83
6.2.1	Funktionstests auf der Entwicklungsplattform	83
6.2.2	Funktionstests auf der Zielplattform.....	86
6.3	Erkenntnisse aus den Tests	87
7	Zusammenfassung und Ausblick	88
7.1	Zusammenfassung der Ergebnisse	88
7.2	Fazit	89
7.3	Ausblick.....	89
	Literaturverzeichnis	90
	Abbildungsverzeichnis.....	93
	Tabellenverzeichnis.....	96
	Quelltextverzeichnis	97
	Abkürzungsverzeichnis.....	98

1 EINLEITUNG

Dieses einleitende Kapitel verschafft einen kurzen Überblick über die Themenstellung und den Aufbau der vorliegenden Masterarbeit. Anfangs wird das Unternehmen IGT, mit welchem in Kooperation diese Masterarbeit umgesetzt wird, näher vorgestellt. Im Anschluss folgt die Erläuterung der Ausgangssituation, der Aufgabenstellung und des Untersuchungsinteresses der Arbeit, sowie die genaue Zieldefinition. Eine kurze Beschreibung des Aufbaus der Arbeit rundet das Kapitel ab.

1.1 Das Unternehmen IGT

Bei dem Unternehmen IGT (International Game Technology) handelt es sich um einen weltweit führenden Betreiber und Technologie-Anbieter in regulierten Märkten der Spieleindustrie mit Niederlassungen rund um den Globus. Das Know-how von IGT umfasst innovative Technologien und Gaming Produkte verschiedenster Art. Das Unternehmen bietet seinen Kunden daher eine Vielzahl erstklassiger Produktlinien und Serviceleistungen, von Spielgeräten und Spielen über Online Gaming bis hin zu Lotterien und Sportwetten. Investition in Innovation und die Nutzung von Spitzentechnologien ist IGTs oberstes Ziel, ebenso wie die Verpflichtung zu höchsten Standards bezüglich Integrität, Verantwortung und einer nachhaltigen Unternehmenswertsteigerung.¹

Der Standort in Premstätten (Österreich) blickt auf eine 28-jährige Erfahrung in der Spieleindustrie unter verschiedenen Firmenbezeichnungen zurück. Seit der Übernahme von IGT im Jahr 2015 repräsentiert sich der Standort unter dem Namen IGT Austria GmbH. Der Fokus am Standort liegt in der Entwicklung von Softwarelösungen im Automatenbereich.¹

Unternehmenseckdaten:¹

- Firmengründung: 1993 in Österreich, 1975 in den USA.
- Mitarbeiter: ca. 250 in Österreich, global ca. 12.000.
- Hauptniederlassungen: USA, Kanada, Italien, England und Österreich.
- Produkte: Casino-Spielautomaten und Spiele, Casino-Management-Systeme, Lotterielösungen, Online-Gaming- und Sportwettssysteme.
- Märkte: Nordamerika (NAM); Lateinamerika und Karibik (LAC); Europa, Naher Osten und Afrika (EMEA); Asien-Pazifik (APAC).



Abbildung 1: IGT Logo, Quelle: International Game Technology (o.J.), Online-Quelle [24.02.2021].

¹ Vgl. IGT Austria GmbH (2020), Quelle: Unternehmensinterne Quelle (Human Resources).

1.2 Ausgangssituation

Wie bereits aus der Beschreibung des Unternehmens in Kapitel 1.1 zu entnehmen ist, befasst sich das Unternehmen IGT neben anderen Geschäftsbereichen mit der Hardware- und Softwareentwicklung von Spielautomaten für verschiedenste regulierte Märkte weltweit. Trotz der hohen Verfügbarkeit von modernen bargeldlosen Bezahlssystemen, wie Ticket-, Karten- oder NFC-Systeme es sind, erfordern Spielautomaten in einigen Märkten noch immer konventionelle Münzverarbeitungssysteme (engl. Coin Handling Systems).

Im Zuge der Entwicklung von maschinennaher Software überwacht die Abteilung für Software-Qualitätssicherung (SQA, Software Quality Assurance) den Entwicklungsprozess. Die Aufgabe der Software-Qualitätssicherung besteht unter anderem darin, Systemtests der Spielplattform-Software in einer realen Testumgebung durchzuführen. Diese Umgebung bildet im Regelfall ein voll assemblierter Spielautomat, der neben allen anderen Komponenten auch das automatisierte Münzverarbeitungssystem inkludiert. Das System besteht aus einem elektronischen Münzprüfer (engl. Coin Acceptor) mit zugehöriger Münzsortiereinheit (engl. Coin Sorter) und einer oder mehreren Münzauszahleinheiten (engl. Coin Hopper) zur Auszahlung von Münzen an die Spieler*innen.

Auf Grund der unterschiedlichen Märkte gibt es unzählige länderspezifische Währungen. Schnelle und unkomplizierte Tests erweisen sich deshalb oft als sehr schwierig. Entweder fehlt es an Testmünzen in der jeweiligen Währung, oder Münzprüfer und Münzauszahleinheiten sind für die falsche Währung konfiguriert oder schlichtweg gerade nicht verfügbar. Das Besorgen von Münzen oder das Umrüsten des Spielautomaten auf eine andere Währung kann daher einiges an Zeit in Anspruch nehmen.

1.3 Aufgabenstellung und Untersuchungsinteresse

Die Aufgabe besteht darin die Systemtests der Software-Qualitätssicherung zeitsparender und effizienter zu gestalten. Im Fokus der Arbeit steht daher die Systementwicklung eines geeigneten Emulators für das automatisierte Münzverarbeitungssystem im Spielautomaten. Der Emulator soll es ermöglichen ohne jegliches Münzgeld oder Zusatzsoftware Kredite auf den jeweiligen Spielautomaten aufzubuchen oder Auszahlungen an die Spieler*innen zu signalisieren.

Zunächst gilt es hierfür das Gesamtsystem zu untersuchen und auf die Einzelkomponenten herunterzubrechen, um deren Funktionsweise zu analysieren. Basierend auf den gewonnenen Erkenntnissen sollen schließlich die erforderlichen Anforderungen an den Emulator ermittelt, und seine Funktionen definiert werden. Auf der Grundlage der Anforderungen soll wiederum eine passende Zielplattform für den Emulator gewählt und eine geeignete Softwarearchitektur abgeleitet werden. Das Hauptaugenmerk der Architektur soll dabei auf Modularität, Wartungsfreundlichkeit und Erweiterbarkeit gelegt werden. Anschließend soll die erarbeitete Architektur programmtechnisch auf der Zielplattform realisiert und unter realen Einsatzbedingungen getestet werden.

1.4 Zieldefinition

Das Ziel dieser Masterarbeit ist es, aus den zuvor gewonnenen Erkenntnissen über Hardware und Software des automatisierten Münzverarbeitungssystems, ein nutzbringendes Gesamtsystem zu entwickeln, das die realen Hardwarekomponenten im Spielautomaten emuliert, und somit einen positiven Nutzen bei Tests durch die Software-Qualitätssicherung bewirkt.

1.5 Aufbau der Arbeit

Die vorliegende Masterarbeit gliedert sich in theoretische und praktische Teilbereiche, welche aufeinander aufbauen.

Der theoretische Teil setzt sich nach einer kurzen Erläuterung der Glücksspielbranche vorwiegend mit den Grundlagen des automatisierten Münzverarbeitungssystems auseinander. Dazu wird zunächst der Aufbau des Gesamtsystems erklärt, als auch dessen Einzelkomponenten im Detail beschrieben. Anschließend folgt die Ermittlung der Systemanforderungen, die an den zu entwickelnden Emulator gestellt werden. Nach einer kurzen Einführung in den grundlegenden Prozess des Requirements-Engineering, folgt die genaue Systemanalyse, aus der die endgültigen Anforderungen an den Emulator abgeleitet werden.

Im praktischen Teil der Arbeit folgt schließlich die eigentliche Entwicklung des Emulators. Das zuvor erlangte Wissen über das automatisierte Münzverarbeitungssystem, dessen Funktionsweise sowie die definierten Anforderungen werden deshalb dazu genutzt, um geeignete Ansätze für die Umsetzung zu bilden. Basierend auf diesen Ansätzen wird sich der Wahl einer geeigneten Hardware angenommen und anschließend eine adäquate Softwarearchitektur festgelegt. Die nachfolgende Implementierung dieser Architektur auf der zuvor gewählten Zielplattform, sowie erste Evaluierungen des fertigen Prototyps, vervollständigen diese wissenschaftliche Arbeit.

2 DAS AUTOMATISIERTE MÜNZVERARBEITUNGSSYSTEM

Trotz des aktuellen Trends zum bargeldlosen Zahlungsverkehr finden sich in allen gängigen Automatentypen flächendeckend automatisierte Systeme zur Verarbeitung von Münzgeld. Die Systeme prüfen eingeworfene Münzen oder Wertmarken auf ihre Echtheit, bestimmen die Wertigkeit und sortieren sie zugleich. Weiters kann auch Wechsel- oder Retourgeld durch entsprechende Münzauszahlungen ausgegeben werden. Das Einsatzgebiet ist sehr breit gefächert und reicht von einfachen Parkschein- oder Fahrkartenautomaten, über Geldwechselautomaten bis hin zu Waren- oder Getränkeautomaten, um nur einige Beispiele zu nennen.

Im Fokus dieser Arbeit steht ausschließlich das spezifische Einsatzgebiet von automatisierten Münzverarbeitungssystemen in Glücksspielautomaten. Der erste Teil dieses Kapitels widmet sich daher einer kurzen Erläuterung der Glücksspielbranche und speziellen Märkten, in denen Münzgeld vorwiegend noch eine zentrale Rolle spielt. Anschließend wird der Einsatz von Münzverarbeitungssystemen in Spielautomaten thematisiert. Dazu wird zunächst der Aufbau des Gesamtsystems beschrieben und anschließend näher auf die Funktionsweise der Einzelkomponenten des Systems eingegangen.

2.1 Die Glücksspielbranche

Das Spiel mit dem Glück reicht in der Menschheitsgeschichte bis in die frühen Anfänge zurück. Bis in das Mittelalter hinein war das Glücksspiel allerdings sehr eng mit Vorhersehungen und Gottesurteilen verknüpft. Erst mit der Zeit der Frühaufklärung versuchte man mathematische Zusammenhänge in Verbindung mit dem Glücksspiel zu bilden. Als typisches Beispiel hierfür gilt die Stochastik, mit der man erstmals versuchte die erreichten Augenwerte beim Würfeln durch Wahrscheinlichkeiten auszudrücken. Neben den Würfelspielen sind mit dem Lauf der Geschichte unzählige andere Glücksspielvarianten von Kartenspielen, wie Baccarat und Poker, Zahlenlotos wie Roulette, und unter anderem auch die verschiedensten Wettspiele entstanden.²

Schließlich wurde mit dem Beginn der Industrialisierung im 19. Jahrhundert durch Automaten eine neue Ära eingeläutet. Zunächst erkannte man, dass durch den Einsatz von Münzverarbeitungssystemen Automaten einen erheblichen wirtschaftlichen Nutzen im Verkaufs- und Dienstleistungsbereich erbringen konnten. Später hat man auch die Bedeutung im Spiel- und Unterhaltungsbereich erkannt, woraus die ersten Karten- und Würfelspielautomaten mit Münzeinwurf hervorgingen. Anfangs wurden Gewinne lediglich in Sach- und Warenwerten wieder ausbezahlt. Erst mit dem 20. Jahrhundert war die Technik so weit fortgeschritten, dass erstmals Münzen direkt vom Automaten ausbezahlt werden konnten.³

Heute umfasst das Glücksspiel vorwiegend den Lotteriebereich, den Sportwettbereich, den Automatenspielbereich sowie den Onlinespielbereich. Im Sinne des österreichischen Gesetzes (§ 1 GSpG Abs. 1) lautet die genaue Definition des Glücksspiels wie folgt:

² Vgl. Gebhardt/Grüsser-Sinopoli (2008), S. 3 f.

³ Vgl. Nähter (o.J.), S. 15, zitiert nach: Haberbosch (1994), Online-Quelle [18.09.2020].

„Ein Glücksspiel im Sinne dieses Bundesgesetzes ist ein Spiel, bei dem die Entscheidung über das Spielergebnis ausschließlich oder vorwiegend vom Zufall abhängt.“⁴

Um diese Abhängigkeit des Spielergebnisses zu gewährleisten, werden heute in der Spielautomaten-Software hochkomplexe mathematische Algorithmen für Zufallszahlen implementiert, die anschließend durch akkreditierte Labors zertifiziert werden.

Mechanische Geschicklichkeitsspiele mit Geldgewinn und Glücksspielgeräte fallen seit 1953 in den Bereich der Geldspielgeräte. Zum Spielerschutz teilt man seit diesem Zeitpunkt Geldspielgeräte in die folgenden beiden Kategorien ein:⁵

- Das staatlich regulierte Spiel mit hohen Geldeinsätzen und
- das gewerbliche Glücksspiel mit geringeren Einsätzen und Gewinnen.

Die Geldspielgeräte in Märkten des gewerblichen Glücksspiels werden oft auch als Amusement With Prize (AWP) oder Video Lottery Terminals (VLT) bezeichnet. Während sich das staatlich regulierte Glücksspiel in Casinos immer mehr in die Richtung von bargeldlosen Bezahlssystemen bewegt, kommt in den AWP- und VLT-Märkten auch heute noch vermehrt Münzgeld zum Einsatz. Je nach Rechtssystem (engl. Jurisdiction) sind die Märkte durch unterschiedliche gesetzliche Aspekte, in Hinblick auf Finanz, Ein- und Auszahlungsraten, Spielerschutz, Wettbewerb etc. reguliert.

Im Folgenden wird der AWP- und VLT-Markt anhand vom Beispiel Italien kurz erläutert:⁶

Amusement With Prize (AWP)

Prinzipiell unterscheidet man zwischen landbasiertem (engl. land-based) und Online-Glücksspiel. In Italien war landbasiertes Glücksspiel ursprünglich nur in konventionellen Casinos vorgesehen. Im Jahr 2004 erfolgte neben herkömmlichen Casino-Spielautomaten die Klassifizierung von AWP, die vorrangig das Ziel haben, die Spieler*innen zu unterhalten und kleine Geldbelohnungen auszubezahlen. Im Gegensatz zu Casino Spielautomaten dürfen AWP an vielen öffentlichen Plätzen wie Bars, Cafés, Tabakläden und Tankstellen betrieben werden. Dabei handelt es sich um eine vereinfachte Art von Spielautomaten, bei dem der Einwurf von Münzen sowie auch die Gewinnauszahlungen begrenzt sind.

Video Lottery Terminals (VLT)

Im Zuge eines Erdbebens in Zentralitalien im Jahr 2009 hat die italienische Regierung schließlich auch den Video Lottery-Bereich legalisiert, mit dem Ziel die damit verbundenen Steuereinnahmen zum Wiederaufbau einzusetzen. VLTs sind im Gegensatz zu AWP serverbasierte Spielterminals, die in Wettbüros, sowie in Bingo- und Spielhallen aufgestellt werden dürfen. Auch hier spielt Münzgeld noch eine zentrale Rolle.

⁴ Österreichisches Glücksspielgesetz (2021), § 1 Abs. 1, Online-Quelle [24.02.2021].

⁵ Vgl. Nähter (o.J.), S. 16, Online-Quelle [18.09.2020].

⁶ Vgl. Harris (2012), S. 177.

2.2 Systemaufbau und Überblick

Der Systemaufbau von Münzverarbeitungssystemen hängt neben der Beschaffenheit des jeweiligen Spielautomaten auch von Kunden- und marktspezifischen Anforderungen ab, d.h. je nachdem in welchen Märkten die Spielautomaten zum Einsatz kommen. Je nach Markt und Kunde müssen vom System unterschiedliche Münzwertigkeiten akzeptiert und sortiert werden. Darüber hinaus ist durch das System genau vorgegeben, welche Münzwertigkeiten wieder an die Spieler*innen ausbezahlt werden können. Die Beschaffenheit des Spielautomaten spielt insofern eine Rolle, weil es von den räumlichen Gegebenheiten und der Platzierung anderer Komponenten abhängt, wie die Komponenten des Münzverarbeitungssystems platziert werden und wie die Münzkanäle verlaufen.

Im Folgenden wird nun anhand des Spielautomaten mit der Modellbezeichnung delite™, der in der vorliegenden Konfiguration für den AWP Markt Spanien und Italien entwickelt wurde, der genaue Systemaufbau der Münzverarbeitung erklärt. Der delite™-Spielautomat wurde im Jahr 2012 unter der damaligen Firmenbezeichnung Atronic Austria GmbH eigens entwickelt und produziert. Abbildung 1 zeigt den delite™-Spielautomaten und das darin verbaute Münzverarbeitungssystem.



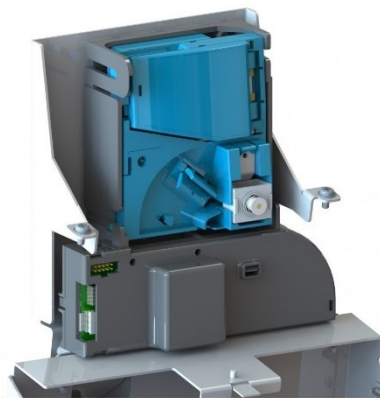
Abbildung 2: Automatisiertes Münzverarbeitungssystem im Spielautomatenmodell delite™, Quelle: Eigene Darstellung.

Im vorliegenden Fall besteht das Münzverarbeitungssystem aus einem elektronischen Münzprüfer mit dazugehöriger Sortiereinheit, zwei Münzauszahlungen und zwei Münzbehältern. Der Münzprüfer ist für die Euro-Währung konfiguriert und akzeptiert 5 verschiedene Münzwertigkeiten (10 Cent, 20 Cent, 50 Cent, 1 Euro und 2 Euro). Die Münzsortiereinheit ist in der Lage die Münzen in 4 verschiedene Kanäle zu

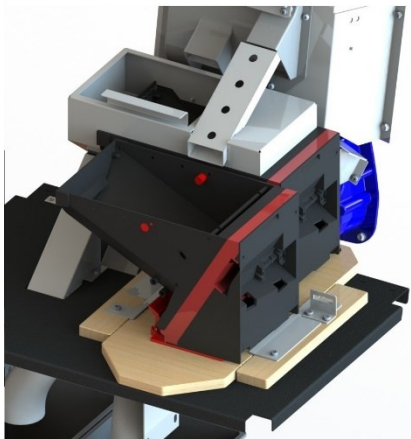
sortieren. Weiters ist eine der Münzauszahleinheiten für die Auszahlung von 2 Euro Münzen, und die andere für die Auszahlung einer kleineren Münzwertigkeit (z.B. 10 Cent oder 20 Cent) bestimmt, um möglichst genaue Beträge ausbezahlen zu können. Im untersten Bereich des Münzverarbeitungssystems befinden sich die zwei Münzbehälter, welche für die Entleerung über eine separate Tür im Spielautomaten zugänglich sind. Einer der Münzbehälter dient bei Überlauf der 2-Euro-Münzauszahleinheit als Auffangbehälter. In den anderen Behälter werden alle Münzwertigkeiten sortiert, für die in den Münzauszahleinheiten, wegen dem Erreichen des oberen Limits oder einer anderen Münzwertigkeit, kein Platz vorhanden ist. Abbildung 3 gibt einen Überblick über die einzelnen Komponenten des automatisierten Münzverarbeitungssystems.



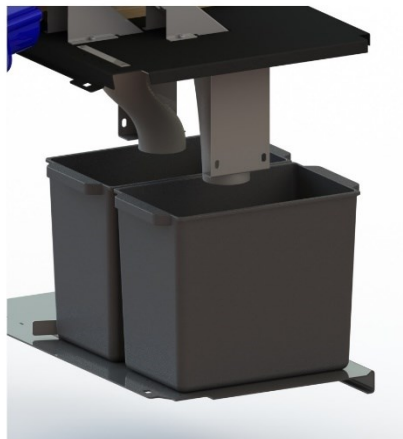
Münzeinwurfschlitz



Münzprüfer und Sortierer



Münzauszahleinheiten



Münzbehälter



Münzschale

Abbildung 3: Einzelkomponenten des Münzverarbeitungssystems, Quelle: Eigene Darstellung

Der Ablauf beim Einwurf von Münzen

Wird eine Münze in den Münzeinwurfschlitz eingeworfen, dann gelangt diese durch die Gewichtskraft zum elektronischen Münzprüfer, welcher die Münze auf Echtheit und auf ihre Wertigkeit überprüft. Die Funktion der Münzprüfung wird in Kapitel 2.2.1 genauer erläutert. Generell werden nicht erkannte Münzen und nicht akzeptierte Münzwertigkeiten umgehend in die Münzschale aussortiert. Münzen deren Wertigkeit erkannt wurde, gelangen unmittelbar nach dem Münzprüfer zur zugehörigen Sortiereinheit, welche die Münzen in

die jeweiligen Münzkanäle sortiert. Über die Münzkanäle gelangen die Münzen entweder in die Münzauszahleinheiten, in die Münzbehälter oder eben zurück in die Münzschale.

Im Folgenden wird nun anhand verschiedener Szenarien erklärt, auf welchem Prinzip die Sortierung beruht. Hierbei wird davon ausgegangen, dass es jeweils für 2 Euro- und für 20 Cent-Münzen eine eigene Auszahleinheit gibt. Dieses Prinzip lässt sich in der Praxis allerdings konfigurieren und kann somit variieren.

- **Einwurf von Falschgeld oder nicht akzeptierten Münzen**

Werden unechte oder vom System nicht akzeptierte Münzen in den Münzeinwurfschlitz des Spielautomaten eingeworfen, dann werden diese vom Münzprüfer erkannt und von der Sortiereinheit unmittelbar über den Auswurfkanal wieder in die Münzschale ausgegeben.

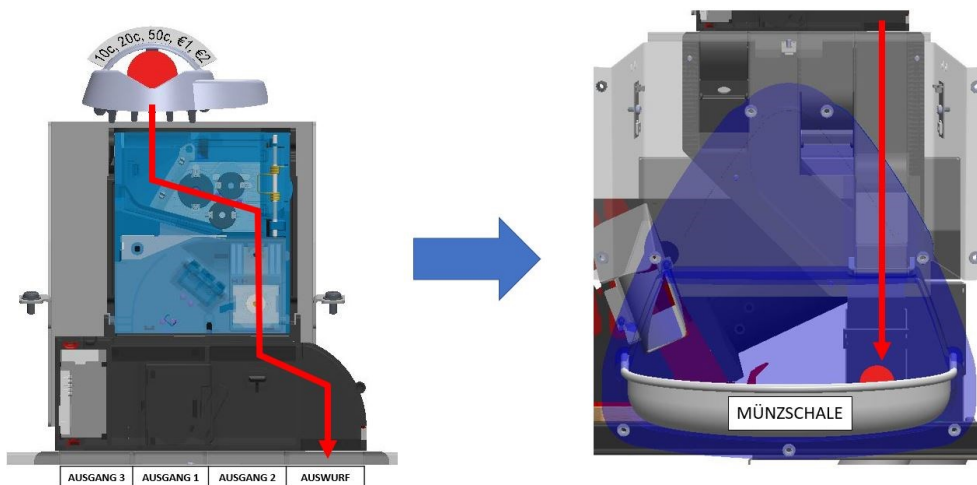


Abbildung 4: Münzpfad bei Einwurf von Falschgeld, Quelle: Eigene Darstellung.

- **Einwurf einer 2 Euro-Münze**

Beim Einwurf einer 2 Euro-Münze in den Münzeinwurfschlitz erkennt der Münzprüfer ihre Wertigkeit und die Münze gelangt über den Ausgang 2 der Sortiereinheit in den Münzkanal direkt zur Münzauszahleinheit für 2 Euro-Münzen. Weiters besitzt die Münzauszahleinheit einen mechanischen Überlauf, der die Münzen bei Erreichen des Füllstands in Münzbehälter 1 befördert.

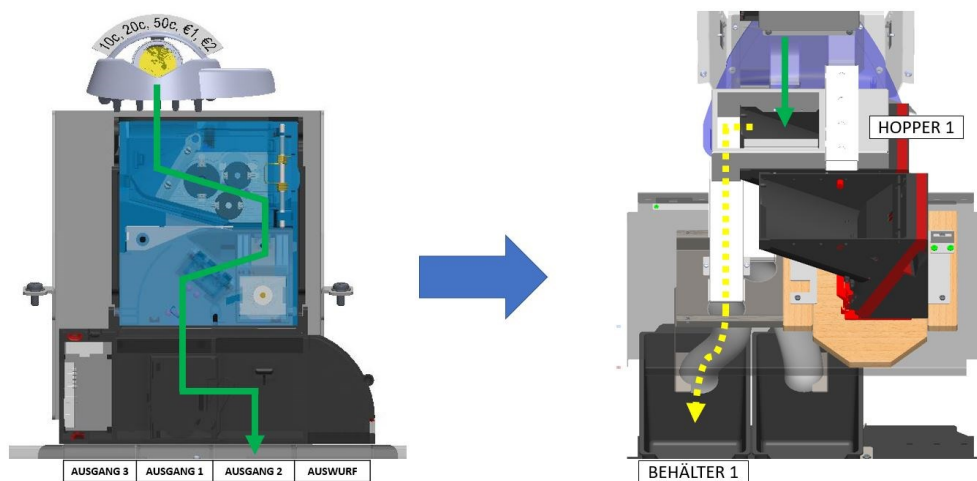


Abbildung 5: Münzpfad bei Einwurf einer 2 Euro-Münze, Quelle: Eigene Darstellung.

- **Einwurf einer 20 Cent-Münze**

Die 20 Cent-Münzen werden nach der Erkennung durch den Münzprüfer üblicherweise über Ausgang 3 der Sortiereinheit in die Münzauszahleinheit für 20 Cent-Münzen befördert. Diese Münzzahleinheit verfügt allerdings über einen Sensor für das obere Limit. Erkennt der Sensor, dass die Münzauszahleinheit voll ist, erfolgt die Sortierung über Ausgang 1 in den Münzbehälter 2.

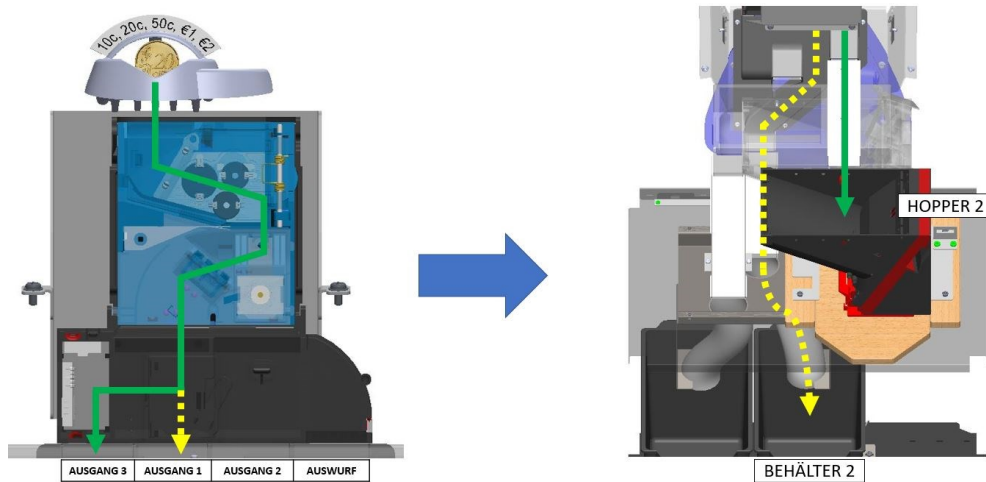


Abbildung 6: Münzpfad bei Einwurf einer 20 Cent-Münze, Quelle: Eigene Darstellung.

- **Einwurf von 1 Euro-, 50 Cent- oder 10 Cent-Münzen**

Alle anderen Münzwertigkeiten, die vom Münzverarbeitungssystem akzeptiert werden, gelangen über Ausgang 1 des Sortierers und entsprechende Münzkanäle direkt in den Münzbehälter 2. Alle Münzen, die in die Münzbehälter sortiert werden, können später vom Betreiber des Spielautomaten entleert werden.

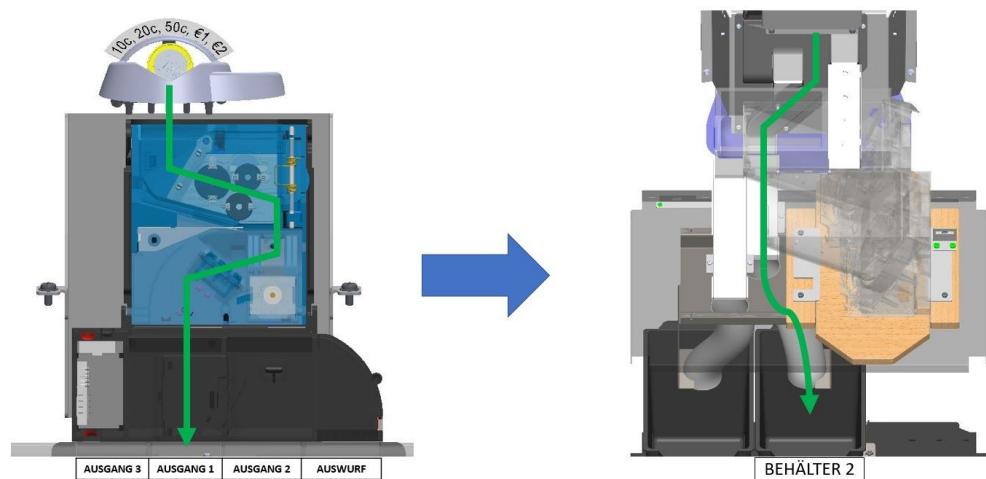


Abbildung 7: Münzpfad bei Einwurf von 1 Euro-, 50 Cent- oder 10 Cent-Münzen, Quelle: Eigene Darstellung.

Ablauf bei der Auszahlung von Münzen

Die Auszahlung von Münzen erfolgt über die beiden Münzauszahleinheiten. Das Funktionsprinzip der Münzauszahleinheiten wird in Kapitel 2.2.2 näher beschrieben. Wie auch schon zuvor, wird wieder angenommen, dass es eine Münzauszahleinheit für 2 Euro- und eine für 20 Cent-Münzen gibt. Signalisiert der Spieler*in dem Spielautomaten durch Betätigung des *Payout*-Buttons, dass dieser die Kredite ausbezahlen soll, dann steuert die Software des Spielautomaten je nach Auszahlungsbetrag, die Auszahleinheiten an. Sollen beispielsweise € 4,60 an Krediten ausbezahlt werden, dann werden 2 Stück der 2 Euro- und 3 Stück der 20 Cent-Münzen ausbezahlt. Die Summe der Münzen ergibt folglich € 4,60.

Die beiden folgenden Punkte erläutern anhand von Abbildungen wie die Auszahlung für jede der Münzauszahleinheiten funktioniert:

- **Auszahlung von 2 Euro-Münzen**

Die Auszahlung der 2 Euro-Münzen erfolgt über die vordere Auszahleinheit, von der die Münzen direkt über einen kurzen Münzkanal in die Münzschale gelangen. Dort kann sie der Spieler*in dann entnehmen.

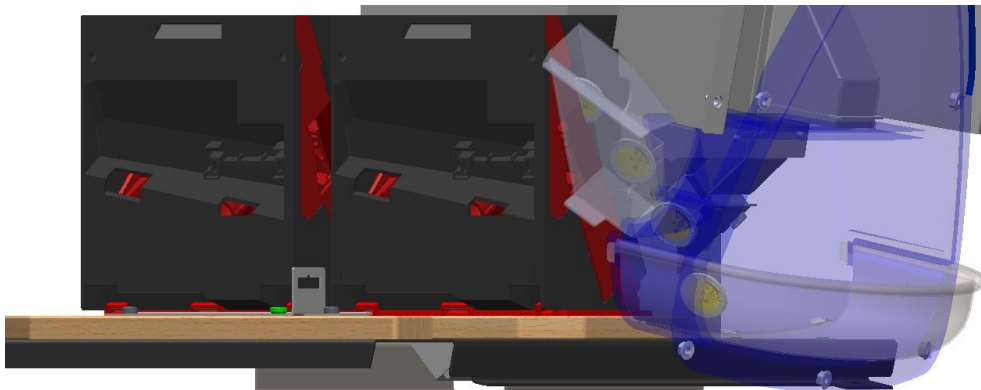


Abbildung 8: Auszahlung von 2 Euro-Münzen, Quelle: Eigene Darstellung.

- **Auszahlung von 20 Cent-Münzen**

Die 20 Cent-Münzen werden durch die hintere Münzauszahleinheit ausgegeben. Wie in Abbildung 9 gut zu erkennen ist, gelangen die Münzen durch einen Schacht mit leichtem Gefälle in der vorderen Auszahleinheit zur Münzschale des Spielautomaten.

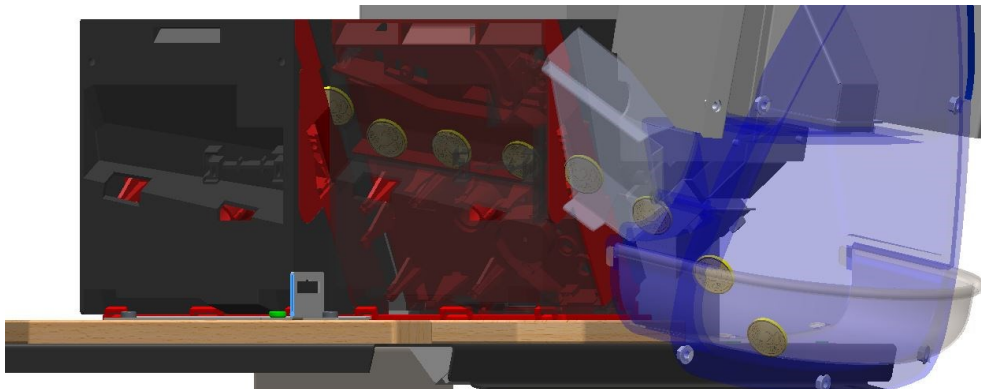


Abbildung 9: Auszahlung von 20 Cent-Münzen, Quelle: Eigene Darstellung.

2.2.1 Münzprüfung

Die Münzprüfung ist ein zentrales Thema beim Einsatz von Münzgeld oder Wertmarken als Zahlungsmittel in Automaten unterschiedlicher Ausführungen. Sie dient als Grundlage zur Vorbeugung von Missbrauch und Betrügereien. Durch die Prüfung anhand unterschiedlicher Kriterien können so Falschgeld, nicht akzeptierte Münzwertigkeiten oder Münzen einer fremden Währung zielgerecht erkannt und aussortiert werden.

Prinzipiell unterscheidet man bei der Prüfung von Münzen zwischen den folgenden Technologien:⁷

- **Mechanische Münzprüfung**

Das Prinzip der mechanischen Münzprüfung beruht auf der Prüfung verschiedener physikalischer Eigenschaften, wie z.B. Durchmesser, Dicke, Gewicht und der magnetischen Eigenschaft der Münze. Entspricht die Münze den Vorgaben, dann setzt die Münze ihren Weg fort und betätigt einen Mikroschalter, welcher dem Endgerät signalisiert, dass eine gültige Münze eingeworfen wurde. Falsche Münzen hingegen werden durch einen mechanischen Mechanismus ausgeworfen. Nachteile dieser Technologie sind die geringe Betrugssicherheit und die Erkennung von nur einer oder wenigen Münzen.

- **Münzkomparatoren**

Das Prinzip von Münzkomparatoren beruht auf dem Vergleich von Münzen mit einer Vergleichsmünze, welche in den Komparator eingelegt wird. Mit der Hilfe von verschiedenen Sensoren wird anhand unterschiedlicher Parameter geprüft, ob die eingeworfenen Münzen der Vergleichsmünze entsprechen. Die nicht übereinstimmenden Münzen werden über eine Weiche ausgeworfen. Der größte Nachteil von Komparatoren ist, dass nur eine einzige Münze erkannt werden kann. Allerdings besitzen Münzkomparatoren einen recht hohen Sicherheitslevel.

- **Elektronische Münzprüfung**

Die elektronische Münzprüfung ist die heutzutage am meisten eingesetzte und missbrauchs- und betrugssicherste Form der Münzprüfung. Die eingeworfenen Münzen durchlaufen im elektronischen Münzprüfer induktive, optische und magnetische Sensoren, durch die verschiedene elektrische Parameter erfasst werden. Die erfassten Parameter werden zur Erkennung der Münzen mit den im Münzprüfer gespeicherten Annahmebändern verglichen, um zu bestimmen, dass die Münzen echt sind und welche Wertigkeit sie besitzen. Elektronische Münzprüfer können somit eine Vielzahl an unterschiedlichen Münzen erkennen und zuordnen. Darüber hinaus können die Prüfer leicht für andere Währungen oder Münztypen umprogrammiert werden.

Da als Ausgangsbasis für die Entwicklung des Emulators ausschließlich die elektronische Form der Münzprüfung von Bedeutung ist, wird in den folgenden Unterkapiteln darauf näher eingegangen. Zuerst werden die Prüfkriterien betrachtet und anschließend die Funktionsweise des Münzprüfers und der Sortiereinheit beschrieben.

⁷ Vgl. Honeywell International Inc. (o.J.), Online-Quelle [24.02.2021].

2.2.1.1 Münzen und deren Prüfkriterien

Münzen verschiedenster Art werden in allen Kulturen der Menschheitsgeschichte bereits seit jeher als Zahlungsmittel eingesetzt und anerkannt. Im Glücksspielbereich, vorwiegend aber im Automatenbereich, unterscheidet man prinzipiell zwischen Umlaufmünzen, Wertmarken, und sonstigen Geldersatzmarken, die von den Spielautomaten akzeptiert werden. Umlaufmünzen sind allerdings die gängigste Form im Geldspielautomatenbereich, weshalb diese im Folgenden näher erläutert werden.

Per Definition sind Umlaufmünzen als gesetzliches Zahlungsmittel des jeweiligen Staates anerkannt und deren Produktion wird vom Staat in Auftrag gegeben. Der aufgeprägte Nennwert der jeweiligen Münze wird vom Staat und dessen Nationalbank garantiert. Auf Grund dessen gibt es in jeder Währungsunion ein entsprechendes Münzrecht, in dem das Erzeugnis von Münzen und deren Prägemerkmale gesetzlich festgelegt sind.⁸

Um die Betrugsgefahr zu vermindern und vor Falschmünzen zu schützen, die z.B. zur Täuschung von Münzprüfern hergestellt werden, besitzen Umlaufmünzen spezielle technische Merkmale. Fälschungen sind Münzen, die in Gewicht, Material und Prägung dem Original nachempfunden sind und in Umlauf gebracht werden. Münzprüfer müssen daher Falschmünzen von echten Münzen anhand des Durchmessers, der Dicke, der Materialeigenschaften und der Prägertiefe durch entsprechende Sensorik unterscheiden.⁹

In der nachfolgenden Auflistung werden die technischen Merkmale, die für die elektronische Münzprüfung von Bedeutung sind, näher erläutert:¹⁰

- **Durchmesser**

Der äußere Münzdurchmesser ist eines der technischen Merkmale, welches sich meist mit der Münzwertigkeit ändert, und daher geprüft wird.

- **Dicke:**

Die Dicke der Münze ist ein weiteres Merkmal, dass sich je nach Münze und Münzwertigkeit verändern kann.

- **Material:**

Münzen bestehen aus Legierungen verschiedener Metalle, die je nach Zusammensetzung andere elektrische und magnetische Charakteristiken aufweisen.

- **Prägertiefe:**

Bei echten Münzen hebt sich das Bild der Münze klar von der Münzoberfläche ab und die Konturen sind unverkennbar. Die Tiefe dieser Prägung ist ein weiteres Indiz für die Echtheit von Münzen.

⁸ Vgl. Rieck/Krüger (2016), S. 6, zitiert nach: Bongardt (1983), S. 5, Online-Quelle [02.09.2020].

⁹ Vgl. Rieck/Krüger (2016), S. 23 f., zitiert nach: Adameck (2004), S. 16 ff., Online-Quelle [02.09.2020].

¹⁰ Vgl. Rieck/Krüger (2016), S. 5 ff., Online-Quelle [02.09.2020].

Für die Euro-Münzen, die für den Umlauf in der Europäischen Währungsunion bestimmt sind, sind die Stückelungen und deren technische Merkmale in der EU-Verordnung Nr. 729/2014 festgelegt. Die nachfolgende Tabelle 1 gibt einen groben Überblick über einige dieser genannten technischen Merkmale.¹¹

Nennwert	Durchmesser	Dicke	Gewicht	Form	Farbe	Zusammensetzung	Rändelung
€	mm	mm	g				
2,00	25,75	2,20	8,5	rund	außen: weiß	Kupfer-Nickel (Cu75Ni25)	Schriftprägung auf dem Münzrand fein geriffelt
					innen: gelb	dreischichtig Nickel-Messing/Nickel/Nickel-Messing (CuZn20Ni5/Ni12/CuZn20Ni5)	
1,00	23,25	2,33	7,5	rund	außen: gelb	Nickel-Messing (CuZn20Ni5)	Gebrochen geriffelt
					innen: weiß	dreischichtig (Cu75Ni25/Ni7/Cu75Ni25)	
0,50	24,25	2,38	7,8	rund	gelb	Nordisches Gold (Cu89Al5Zn5Sn1)	Randprägung mit feiner Wellenstruktur
0,20	22,25	2,14	5,7	"Spanische Blume"	gelb	Nordisches Gold (Cu89Al5Zn5Sn1)	ohne Randprägung
0,10	19,75	1,93	4,1	rund	gelb	Nordisches Gold (Cu89Al5Zn5Sn1)	Randprägung mit feiner Wellenstruktur
0,05	21,25	1,67	3,9	rund	rot	Stahl mit Kupferauflage	glatt
0,02	18,75	1,67	3,0	rund	rot	Stahl mit Kupferauflage	glatt mit Einkerbung
0,01	16,25	1,67	2,3	rund	rot	Stahl mit Kupferauflage	glatt

Tabelle 1: Technische Merkmale der Euro-Münzen, Quelle: In Anlehnung an VERORDNUNG (EU) Nr. 729/2014 (2014), Online-Quelle [24.02.2021].

Wie nun genau ein elektronischer Münzprüfer aufgebaut ist, und wie die Ermittlung der zuvor genannten vier technischen Merkmale und die damit verbundene Echtheitsprüfung durch den elektronischen Münzprüfer erfolgt, wird in Kapitel 2.2.1.2 genauer erklärt.

¹¹ Vgl. VERORDNUNG (EU) Nr. 729/2014 (2014), S. 1., Online-Quelle [24.02.2021].

2.2.1.2 Elektronischer Münzprüfer

Der erste elektronische Münzprüfer wurde vom damaligen Hersteller National Rejectors Inc. (NRI), welcher heute zum Unternehmen Crane Payments Innovations (CPI) zählt, entwickelt. Dieser Münzprüfer kam erstmalig 1972 in Fahrkartensystemen bei den Olympischen Spielen in München zum Einsatz. Seit diesem Zeitpunkt wurde die elektronische Münzprüfung kontinuierlich verbessert und heute gibt es eine große Anzahl an namhaften Herstellern in dieser Branche.¹²

Die genaue Vorgehensweise der Münzprüfung, die Prüfung der Münzen anhand der technischen Merkmale, sowie der Aufbau des elektronischen Münzprüfers kann je nach Hersteller und Modell variieren. Im nachfolgenden wird anhand des Münzprüfer-Modells RM5 des italienischen Herstellers SUZOHAPP-Comestero musterhaft erklärt, wie die Münzprüfung funktioniert. Die Ansteuerung des Münzprüfers erfolgt über das ccTalk-Protokoll, welches im weiteren Verlauf dieser Arbeit noch genauer erklärt wird.

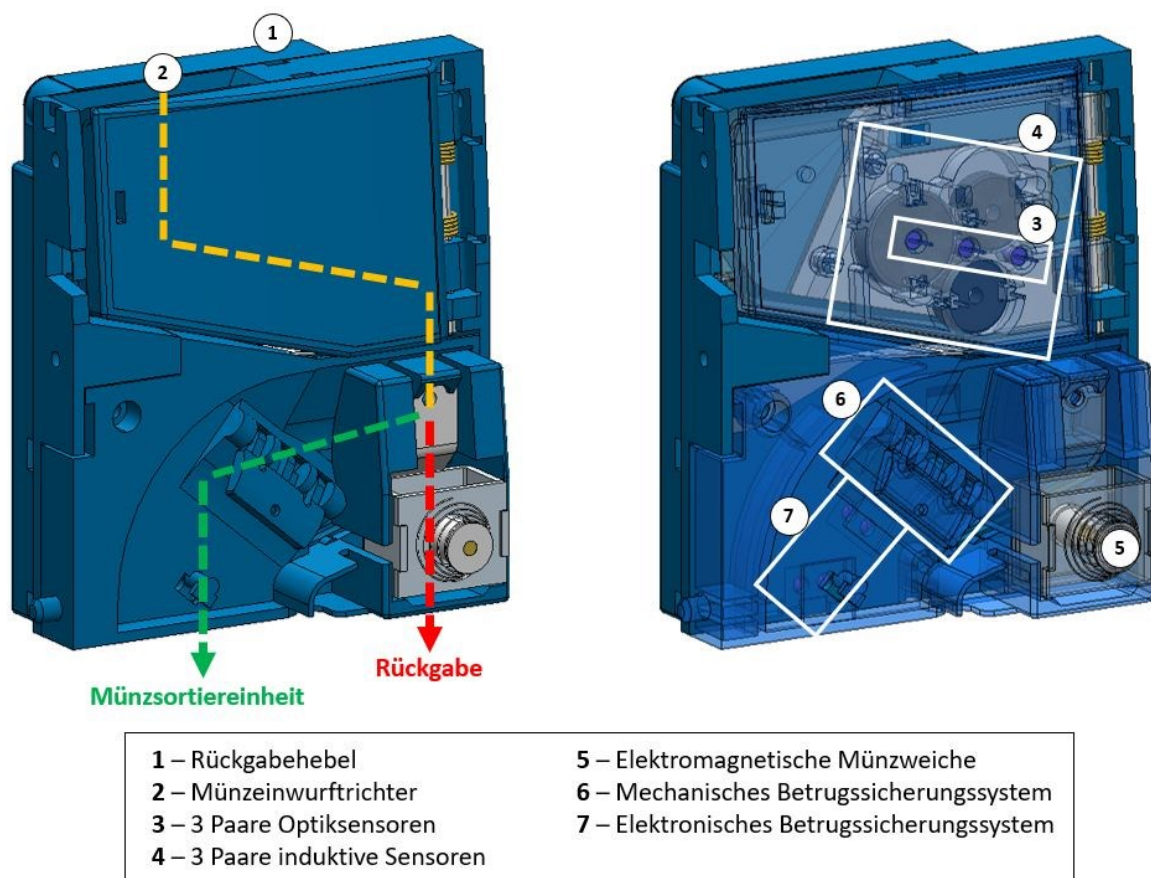


Abbildung 10: Aufbau des Münzprüfers SUZOHAPP-Comestero RM5 Modell G, Quelle: Eigene Darstellung.

Wie in Abbildung 10 zu erkennen ist gelangen die zu überprüfenden Münzen über den Münzeinwurftrichter in den Münzprüfer. Für den Fall, dass sich eine Münze beim Einwurf verklemmt, kann ein mechanischer Rückgabelhebel betätigt werden. Die Münze fällt somit unmittelbar in den Rückgabekanal. Alle anderen Münzen durchlaufen auf ihrem Weg ein Netzwerk aus optischen und induktiven Sensoren im oberen

¹² Vgl. Crane Payment Innovations (2020), Online-Quelle [24.02.2021].

Bereich des Münzprüfers. Durch die Sensorik werden die Münzen sowohl auf ihre Echtheit geprüft als auch deren Wertigkeit bestimmt. Dies geschieht wie folgt:

- **Optische Sensorik:**

Die optische Sensorik umfasst drei speziell angeordnete Paare von Dioden, die jeweils aus gegenüberliegenden Infrarotdioden und Fototransistoren bestehen. Die Dioden sind dabei so angeordnet, dass beim Durchlauf von Münzen deren Münzdurchmesser bestimmt werden kann. Andere Hersteller von elektronischen Münzprüfern bestimmen den Münzdurchmesser auch über induktive Sensoren.

- **Induktive Sensorik:**

Die induktive Sensorik bilden drei Paare Ferritkernspulen unterschiedlicher Größe, welche gegenüberliegen und symmetrisch zum Münzschacht angeordnet sind. Zwei zusammengehörige Ferritkernspulen sind jeweils Bestandteile von elektrischen Schwingkreisen, die von Oszillatoren unterschiedlicher Grundfrequenzen angeregt werden. Wenn eine Münze die Spulenordnung passiert, führt dies zu einer Änderung der Oszillatorfrequenz, der Spannungsamplitude und anderen Parametern. Die Änderungen hängen dabei von den in Kapitel 2.2.1.1 erwähnten technischen Merkmalen, wie Münzdicke, Material und Oberflächenbeschaffenheit ab. Die Auswertung erfolgt in modernen elektronischen Münzprüfern durch einen Mikrocontroller, der die erfassten Änderungen mit vorprogrammierten Werten vergleicht.¹³

Für die vorprogrammierten Werte steht eine gewisse Anzahl an Speicherplätzen, sogenannte Münzkanäle, zur Verfügung, die mit unterschiedlichen Münzsorten oder Wertmarken belegt werden können. Jedem Münzkanal kann dabei das Annahmehand einer bestimmten Münzsorte zugeordnet werden, welche in diesem Kanal angenommen wird. Die einzelnen Grenzwerte der Münzkanäle liegen nah beieinander, um Falschgeld sicher abweisen zu können. In weiterer Folge dient die Münzinformation des jeweiligen Münzkanals als Sortierinformation für die Münzsortiereinheit.¹⁴

Wenn eine eingeworfene Münze von der Sensorik nicht erkannt wird, und infolgedessen keinem Münzkanal zugeordnet werden kann, fällt diese direkt in den Rückgabekanal. Handelt es sich allerdings um eine echte Münze, die einem der Münzkanäle zugeordnet werden kann, wird die elektromagnetische Münzweiche angesteuert, um die Münze zur Sortiereinheit umzuleiten.

Um vor Betrug durch Münzen zu schützen, die durch einen Rückholfaden wieder aus dem Münzprüfer herausgezogen werden, gibt es ein mechanisches und elektronisches Betrugssicherungssystem. Das mechanische System ist so konzipiert, dass Münzen nur in eine Richtung passieren können. Beim Versuch Münzen wieder mit einem Rückholfaden herauszuziehen, blockiert die Vorrichtung die Münze oder eine Fadenschnittklinge durchtrennt den Faden. Die elektronische Betrugssicherung hingegen erkennt die Bewegungsrichtung der Münze mittels zwei nacheinander geschalteter optischer Sensoren.

¹³ Vgl. Mars Money Systems (o.J.), Online-Quelle [24.02.2021].

¹⁴ Vgl. National Rejectors, Inc. GmbH (2007) S. 17, Online-Quelle [24.02.2021].

2.2.1.3 Münzsortierer

Die Aufgabe des Münzsortierers besteht darin, die Münzen nach der Prüfung in die entsprechenden mechanischen Münzkanäle zu sortieren. Ebenso wie bei den elektronischen Münzprüfern, gibt es bei den Sortiereinheiten unzählige Hersteller. Im Regelfall besteht die Kombination von Münzprüfer und Sortiereinheit aus Komponenten von ein und demselben Hersteller. Folglich wird in diesem Kapitel die Funktionsweise anhand eines SUZOHAPP-Comestero SPS31LCC3 Sortierers erläutert.



Abbildung 11: SUZOHAPP-Comestero Münzprüfer und Münzsortierer, Quelle: SUZOHAPP (o.J.), Online-Quelle [24.02.2021].

Der Münzsortierer ist auf einer gemeinsamen Aufnahme direkt unterhalb des elektronischen Münzprüfers montiert und wird von diesem angesteuert. Der SPS32LCC3 ist ein 3-Kanal-Sortierer mit zusätzlichem Auswurfkanal. Die Münzen, die vom Münzprüfer ausgeworfen werden, fallen unmittelbar nach dem Münzprüfer durch einen separaten Auswurfschacht des Sortierers. Vom Münzprüfer akzeptierte Münzen können in 3 verschiedene Kanäle sortiert werden. Jedem Münzkanal bzw. jeder Münzsorte wird dazu in der Konfiguration des Münzprüfers eine Sortierposition zugewiesen. Für die Sortierung sorgen zwei elektromagnetische Münzweichen, die vom elektronischen Münzprüfer gesteuert werden. Wird keine der beiden Münzweichen angesteuert, fällt die Münze automatisch senkrecht zum Ausgang 1. Soll die Münze auf ihrem Weg den Sortierer über Ausgang 2 verlassen, wird die Weiche 2 angesteuert. Bei Ansteuerung von Weiche 1 erfolgt die Sortierung hingegen zu Ausgang 3. Die nachfolgende Abbildung 12 zeigt die Sortierung der Münzen in Abhängigkeit von den Weichenstellungen des Sortierers.

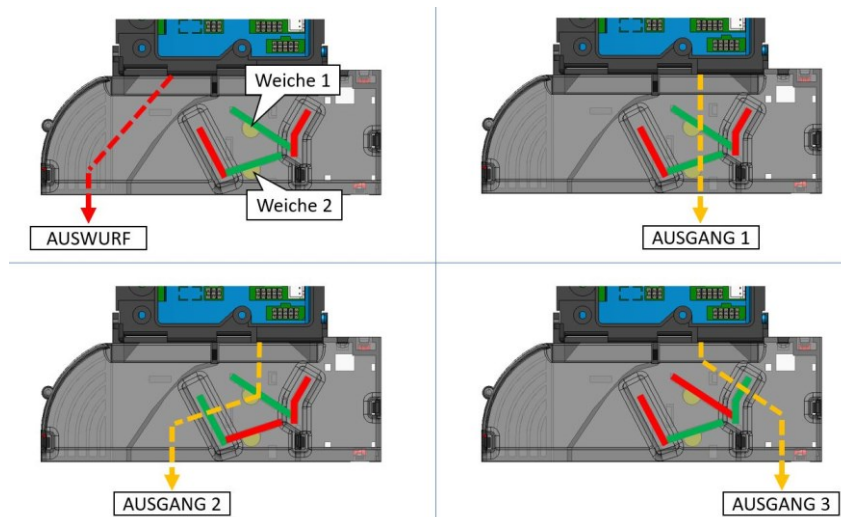


Abbildung 12: Weichenstellungen des SUZOHAPP-Comestero SPS31LCC3 Sortierers (Rückseite), Quelle: Eigene Darstellung.

2.2.2 Münzauszahleinheit

Die Auszahlung von Münzen an die Spieler*innen erfolgt über die bereits in den vorherigen Kapiteln erwähnten Münzauszahleinheiten. In diesem Kapitel wird auf den Aufbau und die Funktionsweise anhand des Hopper U-II des spanischen Herstellers Azkoyen eingegangen (siehe Abbildung 13).



Abbildung 13: Hopper U-II mit kleinster Münzschale, Quelle: AZKOYEN Medios de Pago, S.A (2011), Online-Quelle [24.02.2021].

2.2.2.1 Aufbau

Der Hopper U-II ist eine intelligente mechanische Auszahleinheit mit Rotationsprinzip. Er ist universell einsetzbar und für die Auszahlung von Münzen und Wertmarken mit einem Durchmesser von 12 bis 32 mm und einer Dicke von 1,2 bis 3,5 mm bestimmt. Seine Konstruktion garantiert eine zuverlässige Auszahlung von mindestens 2 Millionen Münzen.¹⁵

Im Wesentlichen besteht der Hopper U-II aus den folgenden Hauptkomponenten:¹⁶

- **Gehäuse und Münztrichter**

Der Münztrichter ist mit dem Hauptgehäuse verbunden und dient als Speicher für die Münzen. Um unterschiedlich viele Münzen fassen zu können, gibt es Münztrichter in verschiedenen Größen.

- **Füllstandsensoren**

Die Sensorik für das untere und obere Limit dient zur Erfassung des Füllstands und besteht jeweils aus gegenüberliegenden Fotodioden und Fototransistoren im oberen und unteren Bereich des Münztrichters.

- **Motorbetriebene Drehscheibe**

Die Drehscheibe wird von einem DC-Motor angetrieben und bezweckt die Aufnahme der Münzen aus dem Münztrichter. Von dort werden sie durch die rotierende Scheibe zum Ausgabepunkt transportiert. Je nach Münzgröße gibt es unterschiedliche Ausführungen von Drehscheiben.

¹⁵ AZKOYEN Medios de Pago, S.A (2011), S. 5, Online-Quelle [24.02.2021]

¹⁶ AZKOYEN Medios de Pago, S.A (2011), S. 13 ff., Online-Quelle [24.02.2021]

- **Sensorik zur Münzzählung**

Unmittelbar am Ausgabepunkt ist eine Infrarot-Lichtschranke montiert, welche die ausgegebenen Münzen erfasst.

- **Ansteuerungsboard**

Das Ansteuerungsboard kommuniziert mit dem Spielautomaten und steuert die Auszahlung der Münzen. Dabei steuert es den Motor bzw. die Drehscheibe und erfasst alle Sensorwerte. Je nachdem wie der Hopper U-II mit dem Spielautomaten kommuniziert, gibt es verschiedene Ansteuerungsboards. Im vorliegenden Zusammenhang erfolgt die Ansteuerung der Auszahleinheit, ebenso wie die Ansteuerung des Münzprüfers, mit dem ccTalk-Protokoll.

2.2.2.2 Funktionsweise

Die Grundvoraussetzung für den regulären Betrieb der Münzauszahleinheit ist, dass diese mit ausreichend Münzen oder Wertmarken befüllt ist und die Spannungsversorgung sowie die Kommunikation mit der Plattform des Spielautomaten über das ccTalk-Bussystem sichergestellt wurde. Empfängt das Ansteuerungsboard von der Plattform nun den entsprechenden ccTalk-Auszahlungsbefehl, beginnt die Drehscheibe gegen den Uhrzeigersinn zu rotieren. Der Trichter ist so geformt, dass die Münzen automatisch zur Drehscheibe gleiten. Durch die Rotation der Drehscheibe bezweckt man, dass sich die Münzen im Trichter parallel zur Scheibe ausrichten, um von den federrückgestellten Kämmen auf der Drehscheibe erfasst werden zu können. Anschließend werden die Münzen auf den einzelnen Kammpositionen der Drehscheibe durch die Rotation zum Ausgabepunkt transportiert. Am Ausgabepunkt sorgt der Auswerfer für die Separierung der Münzen und den Auswurf mittels federrückgestelltem Greifarm. Die Infrarot-Lichtschranke am Ausgabekanal zählt die ausgeworfenen Münzen. Ist die entsprechende Anzahl auszubezahlender Münzen erreicht, stoppt die Drehscheibe wieder.

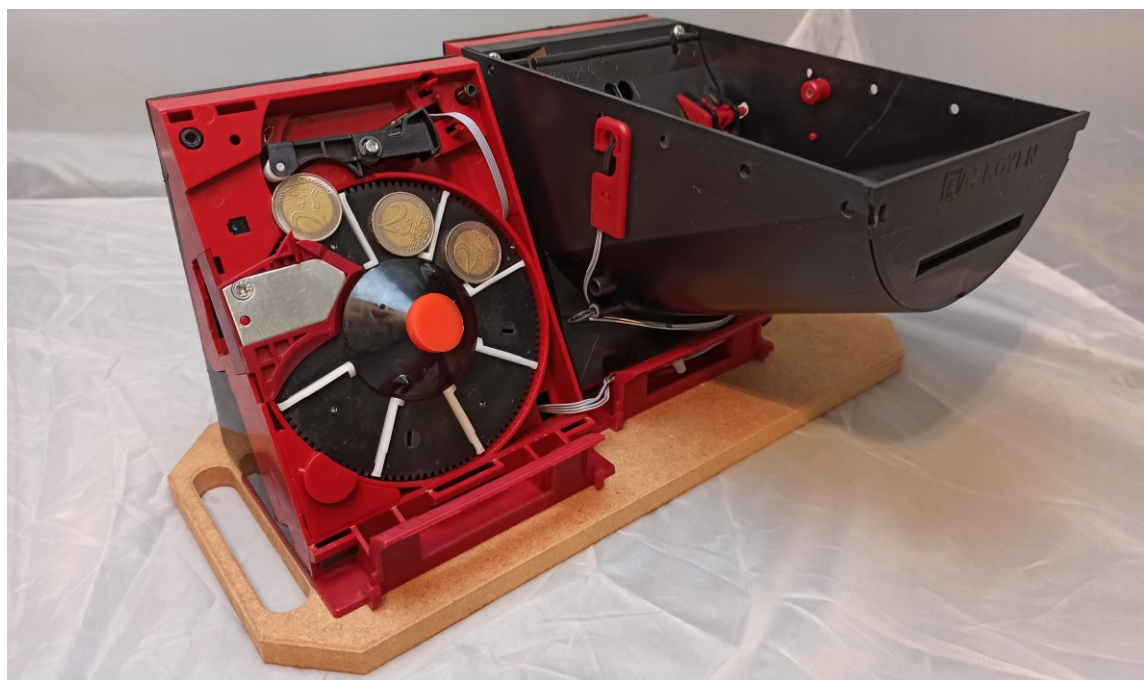


Abbildung 14: Münzauszahleinheiten mit demontiertem Trichter, Quelle: Eigene Darstellung.

3 ERMITTLUNG DER ANFORDERUNGEN

Das folgende Kapitel konzentriert sich auf die Erhebung der Systemanforderungen, die an den zu entwickelnden Emulator gestellt werden. Zunächst wird hierfür auf den grundlegenden Prozess des Requirements-Engineerings eingegangen und dessen Merkmale näher erläutert. Nach der anschließenden Analyse in Bezug auf die Hardware und Software folgt die konkrete Ermittlung der Systemanforderungen. Im weiteren Verlauf der Arbeit sollen die gewonnenen Erkenntnisse aus dem Requirements-Engineering schließlich als Basis für die Planung und Umsetzung dieses Projektes dienen.

3.1 Requirements-Engineering

Das Requirements-Engineering tritt vor allem in frühen Phasen von Entwicklungsprojekten auf und beschreibt einen systematischen und disziplinierten Ansatz, welcher sich zum Spezifizieren und Verwalten von Anforderungen eignet. Das Ziel dieses Prozesses ist es, alle relevanten Anforderungen an ein System zu kennen und einen Konsens über diese unter den Stakeholdern herzustellen. Hierbei gilt es die Wünsche und Bedürfnisse der Stakeholder zu verstehen und diese unter Berücksichtigung vorgegebener Standards in Form von Anforderungen zu spezifizieren, dokumentieren und zu verwalten. Das Risiko, ein System auszuliefern, welches nicht den Wünschen und den Bedürfnissen der Stakeholder entspricht, soll dadurch minimiert bzw. verhindert werden.¹⁷

3.1.1 Der Zweck von Anforderungen

Anforderungen sind ein wesentlicher Bestandteil des Systementwicklungsprozesses und dienen allen Stakeholdern als gemeinsame Basis für Gespräche, Diskussionen und Argumentationen in Hinblick auf das zu entwickelnde System. Die Anforderungsspezifikation in Form eines Dokumentes ist außerdem die Grundlage für Ausschreibungen, die Vertragsgestaltung und den Soll-Ist-Vergleich nach Abschluss der Systementwicklung. Weiters lassen sich mit konkreten Anforderungen unerwartete Fehler bereits von Beginn an vermeiden. Der genauen Definition aller Anforderungen soll daher besondere Beachtung geschenkt werden. Änderungen und Erweiterungen sind typischerweise in der Betriebsphase um ein Vielfaches teurer als bei der direkten Berücksichtigung im Zuge der Ersterstellung eines neuen Systems.¹⁸

3.1.2 Die Haupttätigkeiten des Requirements-Engineerings

Mit dem Prozess des Requirements-Engineerings gehen vier Haupttätigkeiten (siehe Abbildung 15) einher: Ermitteln, Dokumentieren, Prüfen und Verwalten. Jede dieser Tätigkeiten, welchen im Zuge des Requirements-Engineerings besondere Beachtung geschenkt wird, nutzt dabei ihre eigenen Methodiken und Vorgehensweisen und ist maßgeblich am Erfolg oder Misserfolg eines Projektes beteiligt.¹⁹

¹⁷ Vgl. Rupp/SOPHISTen (2014), S. 13.

¹⁸ Vgl. Rupp/SOPHISTen (2014), S. 16.

¹⁹ Vgl. Rupp/SOPHISTen (2014), S. 14.

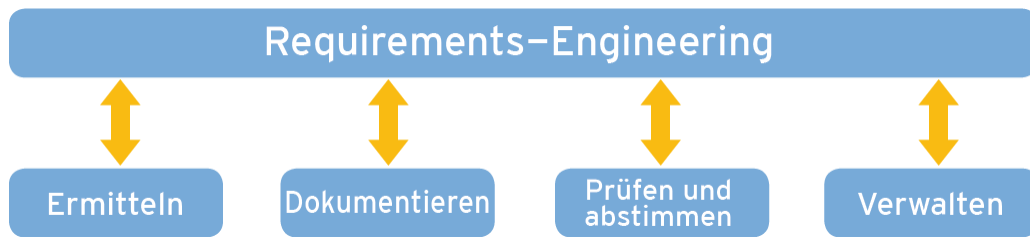


Abbildung 15: Haupttätigkeiten des Requirements-Engineerings, Quelle: Rupp/SOPHISTen (2014), S. 14.

Bestimmung der System- und Kontextgrenzen

Als Grundlage für die zuvor erwähnten Tätigkeiten des Requirements-Engineerings müssen zunächst die System- und Kontextgrenzen (siehe Abbildung 16) bestimmt werden. Die Aufgabe dabei ist es, das zu entwickelnde System klar von seiner Umgebung abzugrenzen und den Teil der Umgebung zu identifizieren, der die Anforderungen bestimmt. Man unterscheidet dabei zwischen den Begriffen Systemkontext, Systemgrenze und Kontextgrenze. Als Systemkontext wird der Teil der Umgebung eines Systems bezeichnet, der für die Definition und das Verständnis der Anforderungen von Relevanz ist. Diesen Systemkontext können z.B. materielle und immaterielle Aspekte, wie andere technische Hard- und Softwaresysteme, Prozesse, Gesetze oder Standards es sind, darstellen. Die Systemgrenze separiert das zu entwickelnde System von seiner Umgebung, die der Systemkontext abbildet. Die im Rahmen des Entwicklungsprozesses veränderbaren Teile der Realität werden von den Teilen abgegrenzt, die nicht im Zuge dieses Prozesses verändert werden können. Schließlich trennt die Kontextgrenze den relevanten Teil der Systemumgebung von der irrelevanten Umgebung. Dies ist der Teil der Umgebung, der keinerlei Einfluss auf das zu entwickelnde System bzw. dessen Anforderungen hat.²⁰

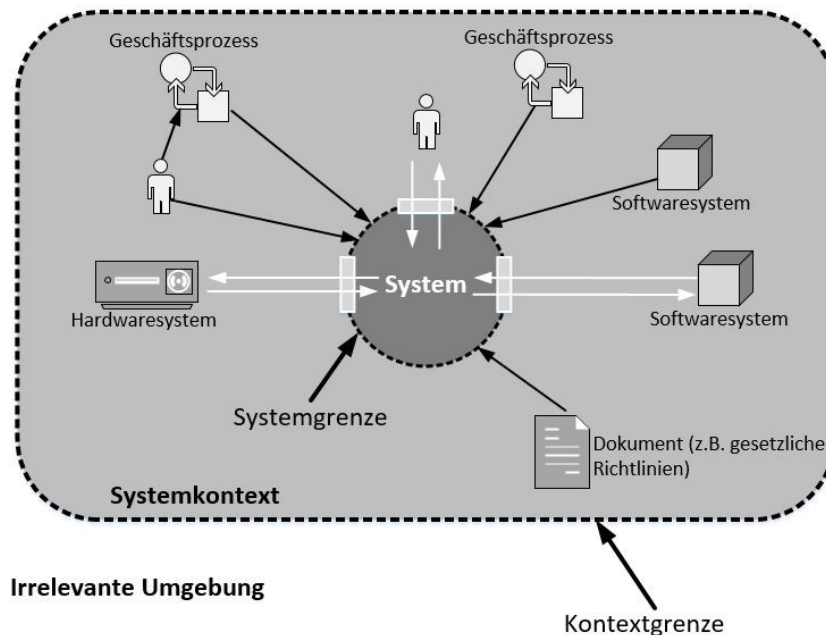


Abbildung 16: System- und Kontextgrenze eines Systems, Quelle: In Anlehnung an Pohl/Rupp (2015), S. 15 f.

²⁰ Vgl. Pohl/Rupp (2015), S. 13 ff.

Im Folgenden werden die vier Haupttätigkeiten und damit verbundene Vorgehensweisen und Methodiken, die beim Requirements-Engineering von Bedeutung sind, näher erläutert.

Ermitteln von Anforderungen

Die Ermittlung der Anforderungen an das zu entwickelnde System ist die erste Haupttätigkeit des Requirements-Engineerings. Die Grundlage dafür ist das erarbeitete Wissen über den zuvor erklärten Systemkontext, der die zu analysierenden Anforderungsquellen identifiziert. Bei den Anforderungsquellen unterscheidet man zwischen Stakeholdern, Dokumenten und Systemen in Betrieb. Stakeholder sind alle Personen oder Organisationen, die in Zusammenhang mit der Systementwicklung stehen und Einfluss auf die Anforderungen haben (z.B. Nutzer, Betreiber, Entwickler, Auftraggeber). Als Dokumente bezeichnet man alle Unterlagen, aus denen Informationen für die jeweiligen Anforderungen gewonnen werden können. Diese Dokumente können z.B. Standards, Normen, Richtlinien, Datenblätter, Gesetzestexte oder organisationsspezifische Unterlagen, wie Anforderungsdokumente oder Fehlerberichte von Altsystemen es sind, sein. Als letzte Anforderungsquelle können mit Systemen in Betrieb Alt-, Vorgänger- oder auch Konkurrenzsysteme gemeint sein, die durch Ausprobieren einen Eindruck des derzeitigen Systems vermitteln. Dieser Eindruck kann zu Erweiterungen oder Änderungen des zu entwickelnden Systems führen.²¹

In der Praxis stellen neben Dokumenten und existierenden Systemen Stakeholder die Hauptquelle für die Anforderungsermittlung dar. Daher ist es zunächst wichtig zu verstehen, welche Bedeutung die jeweiligen Anforderungen für die Zufriedenheit der Stakeholder haben. Anhand des Kano-Modells lassen sich die Anforderungen in drei Kategorien einteilen – Basisfaktoren, Leistungsfaktoren und Begeisterungsfaktoren. Die Basisfaktoren sind essenziell und müssen vom System vollständig erfüllt sein, da sich ansonsten Unzufriedenheit bei den Stakeholdern einstellt. Die Leistungsfaktoren werden von den Stakeholdern explizit gefordert, weshalb dessen Erfüllung zur Zufriedenheitssteigerung besonders erstrebenswert ist. Die Begeisterungsfaktoren werden hingegen nicht von den Stakeholdern gefordert, können aber bei Erkennung des Mehrwerts im System zu einer enormen Steigerung der Zufriedenheit führen.²²

Um schließlich qualitativ hochwertige Anforderungen ermitteln zu können, ist eine gute Kommunikation zwischen den Stakeholdern notwendig. Zur Unterstützung bei der Ermittlung von Wissen und Anforderungen steht daher eine breite Palette an Ermittlungstechniken (z.B. Befragungs-, Kreativitäts- oder Beobachtungstechniken) zur Verfügung. Die Wahl der richtigen Ermittlungstechnik hängt von den organisatorischen und projektspezifischen Bedingungen ab.²³

Dokumentieren von Anforderungen

Sind die Anforderungen an das zu entwickelnde System ermittelt, dann müssen diese geeignet dokumentiert werden. Die Dokumentation in Form eines Anforderungsdokumentes bildet die Basis für die spätere Systementwicklung und nimmt bei der Kommunikation zwischen den Stakeholdern eine

²¹ Vgl. Pohl/Rupp (2015), S. 21.

²² Vgl. Pohl/Rupp (2015), S. 24 f.

²³ Vgl. Pohl/Rupp (2015), S. 26 ff.

unterstützende Funktion ein. Prinzipiell sind bei der Wahl der Dokumentationstechnik keine Grenzen gesetzt, solange für das Dokument einige Qualitätskriterien beachtet werden. Das Dokument soll eine klare Struktur aufweisen, vollständig und konsistent sein, durch seine Nachvollziehbarkeit überzeugen, sowie spätere Modifizierungen und Erweiterungen zulassen. In der Praxis erfolgt die Dokumentation entweder in natürlicher Sprache, durch konzeptuelle Modelle oder aus Mischformen beider Varianten. Ein großer Vorteil des Dokumentierens in natürlicher Sprache ist, dass die Stakeholder keine Notation erlernen müssen und diese Art sehr vielseitig einsetzbar ist. Allerdings birgt sie die Gefahr der Mehrdeutigkeit, d.h. Informationen könnten unterschiedlich interpretiert werden. Hingegen sind konzeptuelle Modelle, wie Use-Case-, Klassen-, oder Aktivitätsdiagramme, oft verständlicher und sie weisen oft einen höheren Eindeutigkeitsgrad auf als die natürliche Sprache. Eher als nachteilig anzusehen sind aber die spezifischen Modellierungskennnisse, die meist erlernt werden müssen. Tatsächlich bewährt haben sich Dokumentenvorlagen in natürlicher Sprache, die mit konzeptuellen Anforderungsmodellen vervollständigt werden.²⁴

Prüfen und Abstimmen von Anforderungen

Um sicherzustellen, dass die dokumentierten Anforderungen den festgelegten Qualitätskriterien entsprechen und die Wünsche und Vorstellungen der Stakeholder korrekt abbilden, gilt es im nächsten Schritt diese noch einmal zu überprüfen. Dazu erfolgt die Prüfung der Anforderungen hinsichtlich der drei Qualitätsaspekte Inhalt, Dokumentation und Abgestimmtheit. Der Qualitätsaspekt „Inhalt“ bezieht sich auf die Überprüfung von inhaltlichen Fehlern in Bezug auf die Qualitätskriterien wie Vollständigkeit, Verfolgbarkeit, Korrektheit, Konsistenz, Überprüfbarkeit und Notwendigkeit. Der zweite Qualitätsaspekt „Dokumentation“ zielt auf die Überprüfung von Mängeln und Verstößen gegen geltende Dokumentations- und Spezifikationsvorschriften ab. Dazu zählen z.B. die Verständlichkeit, Eindeutigkeit sowie die Konformität zu Dokumentationsformat, -struktur und -regeln. Der dritte und letzte Qualitätsaspekt „Abgestimmtheit“ betrifft Mängel in der Abstimmung der Anforderungen unter allen relevanten Stakeholdern. Im Fokus steht dabei die generelle Abstimmung, die Abstimmung nach Änderungen der Anforderungen, als auch die Prüfung, ob alle bekannten Konflikte gelöst wurden. Für die Prüfung hinsichtlich der beschriebenen Qualitätsaspekte stehen unterschiedliche Techniken zur Verfügung. Dies kann einerseits durch einfache Reviews in Form von Stellungnahmen, Inspektionen oder Walkthroughs erfolgen. In der Praxis haben sich aber auch das perspektivenbasierte Lesen, Checklisten oder die Prüfung durch Prototypen bewährt.²⁵

Verwalten von Anforderungen

Die letzte wichtige Haupttätigkeit des Requirements-Engineerings ist das Verwalten von Anforderungen, welches auch als Requirements-Management bezeichnet wird. Das Ziel dieser Tätigkeit besteht darin, die Verfügbarkeit der dokumentierten Anforderungen über den gesamten Produkt- oder Systemlebenszyklus zu gewährleisten. Eine sinnvolle Strukturierung und Kategorisierung der Anforderungen ist daher von Vorteil und kann durch verschiedene Techniken erfolgen. Die Attributierung durch einen eindeutigen

²⁴ Vgl. Pohl/Rupp (2015), S. 35 ff.

²⁵ Vgl. Pohl/Rupp (2015), S. 95 ff.

Identifikator, einen Namen, eine Kurzbeschreibung und anderen Merkmalen, verschafft dem Leser beispielsweise einen guten Überblick über die jeweilige Anforderung. Eine Priorisierung unterstützt im Entwicklungsprozess dabei, zwischen wichtigen und eher unwichtigen Anforderungen zu unterscheiden. Weiters können durch Versionierung und Änderungsmanagement unterschiedliche Entwicklungsstände von bestehenden Anforderungen festgehalten werden.²⁶

3.1.3 Einteilung von Anforderungen

Bei der Einteilung von Anforderungen gibt es unterschiedliche Herangehensweisen. Die Autoren Pohl und Rupp unterscheiden diese in ihrer Literatur „Basiswissen Requirements Engineering“ beispielsweise nach drei Arten:²⁷

Funktionale Anforderungen

Die funktionalen Anforderungen definieren alle Funktionalitäten, die das zu entwickelnde System zur Verfügung stellen muss. Es handelt sich daher um eine Anforderung bezüglich des Ergebnisses eines Verhaltens, welches von einer bestimmten Systemfunktion bereitgestellt werden muss. Man unterteilt die funktionalen Anforderungen wiederum in Funktions-, Verhaltens- und Strukturanforderungen.

Qualitätsanforderungen

Die Qualitätsanforderungen zielen auf die Qualität des zu entwickelnden Systems ab und können die Gestalt der Systemarchitektur häufig sogar in einem größeren Umfang als die funktionalen Anforderungen beeinflussen. Die Anforderungen können massive Auswirkungen auf Performance, Verfügbarkeit, Zuverlässigkeit sowie Skalierbarkeit und Portabilität des Systems haben.

Randbedingungen

Die Rand- oder Rahmenbedingungen werden im Vergleich zu den anderen beiden Arten von Anforderungen nicht direkt umgesetzt und sind daher von den Stakeholdern praktisch nicht beeinflussbar. Sie schränken hingegen den Lösungsraum bzw. die Umsetzungsmöglichkeiten der funktionalen Anforderungen und Qualitätsanforderungen eher ein. Beispielsweise können die Randbedingungen auf technologischen, organisatorischen oder rechtlichen Aspekten basieren.

²⁶ Vgl. Pohl/Rupp (2015), S. 119 ff.

²⁷ Vgl. Pohl/Rupp (2015), S. 8 f.

3.2 Analyse des automatisierten Münzverarbeitungssystems

Wie aus den Haupttätigkeiten des Requirements-Engineerings in Kapitel 3.1.2 hervorgeht, müssen zu Beginn der Systementwicklung zunächst die System- und Kontextgrenzen bestimmt werden, um anschließend die relevanten Teile des Systems analysieren zu können. Basierend auf den Erkenntnissen der Systemanalyse können anschließend konkrete Anforderungen an den Emulator ermittelt und dokumentiert werden.

3.2.1 Abbild der Systemumgebung

Die wesentliche Systemumgebung des Emulators bildet der Spielautomat, in dem ein automatisiertes Münzverarbeitungssystem (siehe Kapitel 2) verbaut ist. Der Systemkontext hängt somit von allen relevanten Teilen des Spielautomaten sowie dessen Umgebung ab, die einen direkten oder indirekten Einfluss auf die Funktionsweise des bestehenden Münzverarbeitungssystems bzw. des zu entwickelnden Emulators haben. All diese Teilbereiche werden in Abbildung 17 schematisch dargestellt.

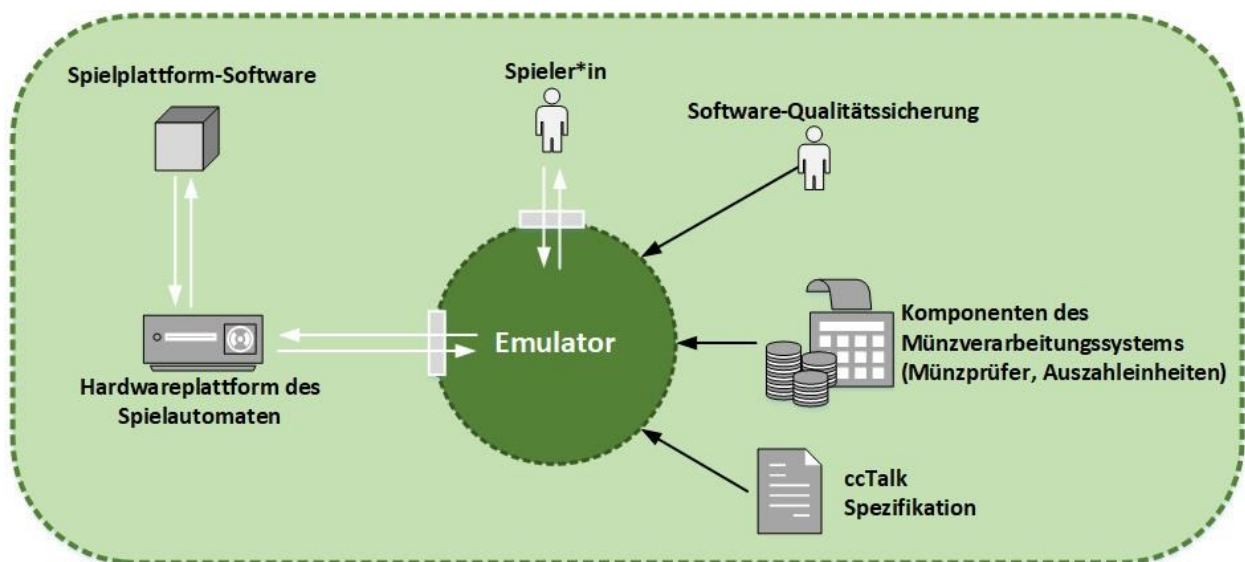


Abbildung 17: Systemkontext des zu entwickelnden Emulators, Quelle: Eigene Darstellung.

Einer der Hauptaspekte des Systemkontexts sind die Wünsche und Vorstellungen der Abteilung für Software-Qualitätssicherung, die als Schlüssel-Stakeholdergruppe in diesem Projekt fungiert. Somit gilt es deren Wünsche unter Berücksichtigung der technischen Machbarkeit bestmöglich in die Anforderungsspezifikation einfließen zu lassen. Ein weiterer wichtiger Aspekt im Systemkontext sind die hardware- und softwarespezifischen Systemteile und deren Schnittstellen. Hierzu zählen die Spielplattform sowie die dazugehörige Software des Spielautomaten, welche für die Ansteuerung des bestehenden Münzverarbeitungssystems verantwortlich sind. Es ist daher wichtig zu verstehen, wie die hardwareseitige Anbindung des Emulators an den Spielautomaten zu erfolgen hat und wie der Protokollstack für die Kommunikation mit den Komponenten des Münzverarbeitungssystems aufgebaut ist und funktioniert. Die Dokumentation der ccTalk-Spezifikation bildet die Grundlage dafür. Der letzte wichtige Aspekt ist es die Perspektive der Spieler*innen bzw. der Tester*innen zu verstehen, da der Emulator von diesen beiden Interessensgruppen später eingesetzt werden soll.

3.2.2 Hardwareseitige Analyse

Der Systementwicklung des Emulators liegt die Analyse der bestehenden Hardwarekomponenten des automatisierten Münzverarbeitungssystems zu Grunde. In Kapitel 2.2 dieser Arbeit wurde deshalb bereits ein grundlegendes Verständnis über den Systemaufbau und die Funktionsweise der Hauptkomponenten eines solchen Systems geschaffen. Hier folgt nun die genaue Ermittlung der eingesetzten Münzprüfer und Münzauszahlseinheiten. Darüber hinaus wird der Frage nachgegangen, wie die elektrische Anbindung der einzelnen Komponenten an die Spielplattform erfolgt.

3.2.2.1 Modelle von Münzprüfern und Münzauszahlseinheiten

Je nach Spielautomaten und Rechtssystem kann das System aus einem elektronischen Münzprüfer und einem oder mehreren Münzauszahlseinheiten verschiedener Dritthersteller zusammengesetzt sein. Jede dieser Komponenten weist eindeutige Identifikatoren, wie Herstellerbezeichnung, Modellname, Seriennummer etc. auf. Unter Verwendung des ccTalk-Protokolls, welches im nächsten Kapitel im Detail beschrieben wird, können diese Identifikatoren von der Spielplattform-Software abgefragt werden. Die Abfrage dient der Spielplattform einerseits zur Erkennung der angeschlossenen Hardwarekomponenten und andererseits als Manipulationsschutz. Es soll vermieden werden, dass manipulierte Komponenten in den Automaten eingebaut werden. Der Betreiber muss neue Hardware, die eben anhand spezieller Identifikatoren erkannt wird, immer in höheren Zugriffsebenen der Spielautomaten-Konfiguration freigeben. Der Emulator muss aus den genannten Gründen in der Lage sein, die zuvor ermittelten Münzprüfer und Münzauszahlseinheiten anhand dieser speziellen Identifikationsmerkmale zu emulieren.

Folglich wurden in Tabelle 2 alle Münzprüfer eruiert, für die das Unternehmen IGT Austria GmbH in den AWP-Märkten Italien und Spanien Unterstützung in der Software anbietet:

Hersteller	Modellbezeichnung
Alberici S.p.A.	AL05
Alberici S.p.A.	AL06V
Alberici S.p.A.	AL55V
Alberici S.p.A.	AL66FG
Alberici S.p.A.	AL66V
Alex Elettronica srl	S10
Azkoyen Payment Technologies	Modular X DSP X6-D2S
Azkoyen Payment Technologies	Modular X6
Azkoyen Payment Technologies	L66S
Azkoyen Payment Technologies	X66S
Azkoyen Payment Technologies	X6-Pina
Crane Payment Innovations - NRI	G-13
Crane Payment Innovations - NRI	v ² colibri
Crane Payment Innovations - NRI	v ² eagle
Money Controls	SR3
Microhard Vending Projects	MOD H10
SUZOHAPP-Comestero	RM5
SUZOHAPP-Comestero	RM5HD

Tabelle 2: Liste der unterstützten Münzprüfer, Quelle: Eigene Darstellung.

Darüber werden alle verwendeten Münz auszahleinheiten derselbigen Märkte in Tabelle 3 angeführt:

Hersteller	Modellbezeichnung
Alberici S.p.A.	Hopper CD F7
Alberici S.p.A.	HopperOne
Alberici S.p.A.	HopperOne S11
Alberici S.p.A.	Hopper Kid
Asahi Seiko Co., Ltd.	SA-595
Asahi Seiko Co., Ltd.	SA-595LC
Asahi Seiko Co., Ltd.	SCH-700
Azkoyen Payment Technologies	Hopper T3
Azkoyen Payment Technologies	Hopper U-II
Azkoyen Payment Technologies	Hopper U3
Azkoyen Payment Technologies	Hopper U Plus
Azkoyen Payment Technologies	Hopper Rode U
Crane Payment Innovations - NRI	currenza h ²
HIMECS	Mini-Hopper
Money Controls	Universal Hopper MK IV
Money Controls	SBB M1
Money Controls	SCH2
Microhard Vending Projects	Hopper X3
Microhard Vending Projects	Hopper X5
Paytec	Hopper HPRO
Paytec	Mini-Hopper F1PRO
SUZOHAPP	Cube Hopper Duo
SUZOHAPP	Cube Hopper MKII
SUZOHAPP	Evolution
SUZOHAPP	Flow Hopper
VNE	Hopper Cubotto
VNE	Hopper Rotativo Fast

Tabelle 3: Liste der unterstützten Münz auszahleinheiten, Quelle: Eigene Darstellung.

3.2.2.2 ccTalk-Hardware-Schnittstelle

Die elektrische Anbindung der in Kapitel 3.2.2.1 ermittelten Komponenten des Münzverarbeitungssystems an die Spielplattform erfolgt über das ccTalk-Bussystem. Mit ccTalk wurde vom Unternehmen Crane Payment Solutions (ehemals Money Controls bzw. Coin Controls) ein Standard geschaffen, der die Möglichkeit bietet, die verschiedensten Bezahlssystemkomponenten über ein und dasselbe Protokoll anzusteuern.²⁸

Das ccTalk-Bussystem ist sehr simpel aufgebaut und benötigt lediglich drei Leitungen – eine Leitung für die Versorgungsspannung, eine Datenleitung und eine Leitung für das Massepotential. Es handelt sich somit um eine serielle und bidirektionale Verbindung mit einer einzigen Datenleitung für beide

²⁸ Vgl. Crane Payment Solutions (2013), Part 1 S. 10.

Kommunikationsrichtungen. Folge dessen werden die Daten im Halbduplex-Betrieb übertragen. Weiters basiert das ccTalk-Protokoll auf dem asynchronen Übertragungsprotokoll RS-232 und dessen Protokollrahmen mit all seinen Start- und Stopbit Bedingungen. Die gewöhnliche Datenübertragungsrate beträgt 9600 Baud. Der Unterschied zu RS-232 besteht in der Kombination von Sende- und Empfangsleitung zu einer gemeinsamen bidirektionalen Datenleitung sowie der Anpassung der spezifizierten RS-232 Spannungspegel auf einfache +5 V oder +12 V Logikpegel.²⁹

Der grundlegende Systemaufbau besteht typischerweise aus einem Hostcontroller (Master) und einem oder mehreren Peripheriegeräten (Slaves) denen unterschiedliche Adressen zugewiesen sind. Der Hostcontroller ist z.B. ein PC, Mikrocontroller oder wie im vorliegenden Fall eine Spielplattform, und die Peripheriegeräte werden durch Bezahlsystemkomponenten, wie Münzprüfer, Münzauszahlungen, Banknotenprüfer oder ähnliches dargestellt.³⁰

Während die meisten Peripheriegeräte die elektronische Schaltung zur Anbindung an das ccTalk-Bussystem bereits integriert haben, muss für die Anbindung an herkömmliche RS-232 Schnittstellen oder UARTs (Universal Asynchronous Receiver Transmitter) von PCs oder anderen Plattformen zunächst die in Abbildung 18 dargestellte Schaltung realisiert werden. Für die Anbindung an eine genormte RS-232 Schnittstelle muss die Schaltung um einen entsprechenden Pegelumsetzer erweitert werden.

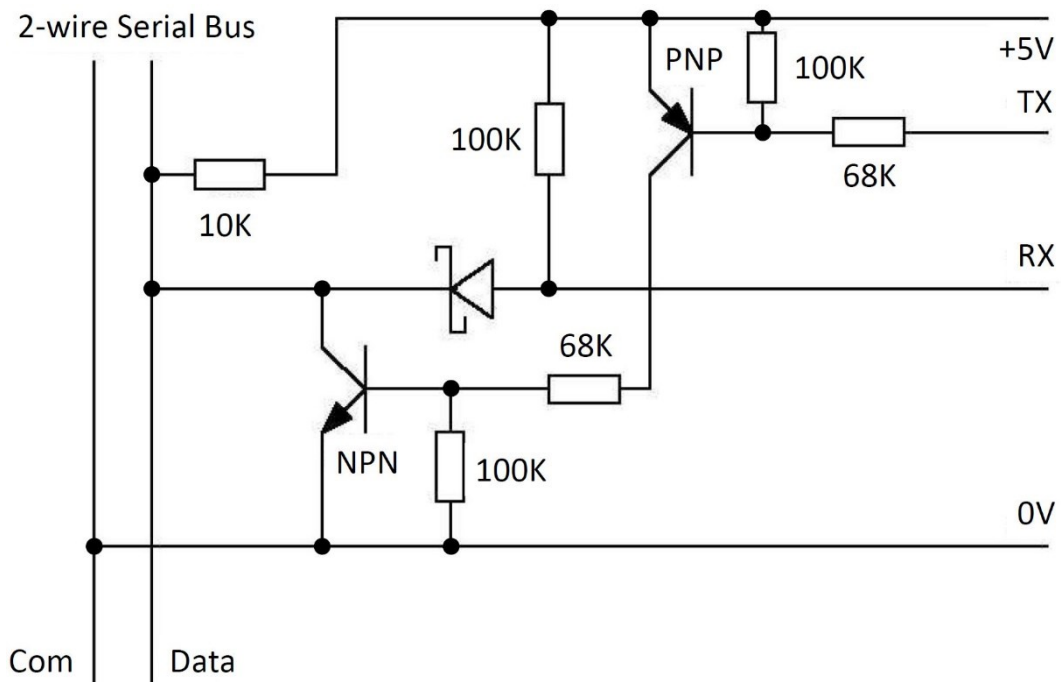


Abbildung 18: Empfohlene ccTalk-Standard-Schnittstelle, Quelle: Crane Payment Solutions (2013), Part 3 S. 82 (leicht modifiziert).

Die Grundsaltung besteht aus einem PNP-Transistor an der Sendeleitung (TX) der UART-Treiberstufe, einem NPN-Transistor in Open-Kollektor-Schaltung zur Ansteuerung der Datenleitung, sowie einer Schottky-Diode an der Empfangsleitung (RX) des UARTs.

²⁹ Vgl. Crane Payment Solutions (2013), Part 4 S. 4 f.

³⁰ Vgl. Crane Payment Solutions (2013), Part 1 S. 10.

Im Ausgangszustand, d.h. wenn keine Daten gesendet werden, liegen an der Sendeleitung +5 V an. Der PNP- als auch der NPN-Transistor befinden sich somit im Sperrzustand. Daraus kann man schließen, dass auch an der Datenleitung des ccTalk-Busses ein Ruhepegel von +5 V und somit ein logischer High-Pegel anliegt.

Werden Daten vom Gerät an den Bus gesendet, erfolgt eine Pegeländerung der Sendeleitung von +5 V auf 0 V. Der PNP-Transistor wird leitend und dies hat zur Folge, dass auch der NPN-Transistor durchschaltet. Über den Open-Kollektor-Eingang wird die Datenleitung des Busses schließlich auf Massepotential gezogen und am ccTalk-Bus liegt ein logischer Low-Pegel an.

Ankommende Empfangssignale von anderen Busteilnehmern werden ebenfalls durch Low-Pegel am Bus signalisiert. Die Empfangsleitung des UARTs wird somit über die Schottky-Diode auf Low-Pegel gezogen. In der praktischen Anwendung resultiert daraus der Nachteil, dass alle an den Bus gesendeten Daten direkt wieder durch die Empfangsleitung eingelesen werden. Um die ankommenden Daten in der Software richtig interpretieren zu können, ist eine Filterung des Echos notwendig. Im Zuge der Softwareentwicklungsphase wird die Vorgehensweise hierfür noch genauer erläutert.

3.2.3 Softwareseitige Analyse

Aufbauend auf den Erkenntnissen der hardwareseitigen Analysen muss im Zuge der Systemanalyse auch die Software genauer betrachtet werden. Im Fokus der softwareseitigen Analyse steht die Analyse des seriellen ccTalk-Kommunikationsprotokolls aus der entsprechenden Protokoll-Dokumentation. Darüber hinaus sollen Untersuchungen der Spielplattform-Software zeigen, welche Befehle aus der Spezifikation tatsächlich im Software-Stack implementiert sind.

3.2.3.1 ccTalk-Protokoll

Wie auch für andere Übertragungsprotokolle existiert für das ccTalk-Protokoll eine entsprechende Protokollbeschreibung bzw. Dokumentation. Die ccTalk-Dokumentation besteht aus einer allgemeinen Spezifikation, die sich in vier Teile gliedert und Folgendes beinhaltet:

- **Teil 1** – Historisches und allgemeine Protokollbeschreibung
- **Teil 2** – Detaillierte Auflistung aller Befehle
- **Teil 3** – Anhänge, Tabellen und Schaltungsbeispiele
- **Teil 4** – FAQ (Frequently Asked Questions), Design-Richtlinien und sonstige Informationen

Bei ccTalk handelt es sich grundsätzlich um einen lizenzfreien Übertragungsstandard zur Kommunikation zwischen verschiedenen Bezahlssystemkomponenten und dem Host. Wie bereits im vorherigen Kapitel erwähnt, basiert das Protokoll auf den Grundsätzen von RS-232 und ist ausschließlich für Single-Master-Applikationen angedacht. Somit existiert im Bussystem immer nur ein einziger Master bzw. Hostcontroller, der alle anderen an den Bus angeschlossenen Peripheriegeräte in einem definierten Zeitintervall abfragt (pollt). Polling bezeichnet in der Informatik eine einfache zyklische Abfrage.³¹

³¹ Vgl. Crane Payment Solutions (2013), Part 1 S. 10 ff.

Der Protokollstapel von ccTalk basiert auf dem ISO/OSI-Referenzmodell und besteht aus vier verschiedenen Schichten. Im Zuge einer Nachrichtenübertragung wird den Nutzdaten zuerst ein entsprechender Overhead hinzugefügt, um die Daten in ein standardisiertes Format zu bringen. Anschließend wird für das aufbereitete Datenpaket eine entsprechende Checksumme berechnet und das Paket verschlüsselt. Im letzten Schritt werden die binären Daten an den Bus gesendet. Beim Empfänger des Datenpaketes erfolgt die Interpretierung der Daten in umgekehrter Abfolge.³²

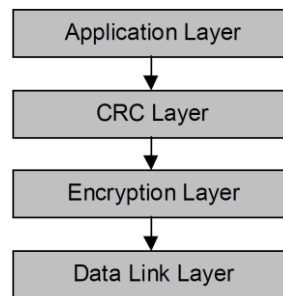


Abbildung 19: ccTalk-Protokollstapel, Quelle: Crane Payment Solutions (2013), Part 1 S. 26.

Im Zuge dieser Arbeit wird auf die CRC- und Verschlüsselungsschicht nur teilweise eingegangen, weil für die Entwicklung des Emulators vorerst nur die einfache Checksummen-Berechnung ohne zusätzliche Verschlüsselung von Relevanz ist.

Allgemeine Nachrichtenstruktur

Das ccTalk-Protokoll weist eine byteorientierte Nachrichtenstruktur auf und unterstützt eine variable Länge der zu übertragenden Nutzdaten. Die byteorientierte Struktur limitiert das System bei der logischen Übertragung somit auf 256 Werte (0 bis 255 dezimal). Für das genaue Format der Daten gibt es aber keinerlei Einschränkungen. Deshalb eignet sich das Protokoll ideal zur Übertragung von Bitmasken, BCD-Codes (Binary Code Decimal) oder von ASCII-Zeichen und -Zeichenfolgen (American Standard Code for Information Interchange). Diese finden beispielsweise Anwendung bei der Identifikation der Peripheriegeräte durch Abfrage von Hersteller, Modellbezeichnung oder Equipment-Kategorie.³³

Die Kommunikation zwischen Hostcontroller und Peripheriegerät besteht in jeder Polling-Sequenz aus zwei aufeinanderfolgenden Datenpaketen. Das erste Datenpaket beinhaltet die Daten der Polling-Anfrage vom Host an die jeweilige Peripherie, und das zweite Datenpaket besteht aus der Antwort der Peripherie zurück an den Host.³⁴

In Tabelle 4 wird die Struktur eines einfachen Datenpaketes mit zu übertragenden Nutzdaten schematisch dargestellt. Der Overhead dieses Datenpaketes besteht aus der Zieladresse des Peripheriegeräts, der Anzahl der zu übertragenden Datenbytes, der Quelladresse des Hosts und einem entsprechenden Header, der einen ccTalk-Befehl darstellt. Anschließend erfolgt die Übertragung der entsprechenden Nutzdaten, die mit 255 begrenzt ist, gefolgt von einer einfachen Checksumme.

³² Vgl. Crane Payment Solutions (2013), Part 1 S. 26.

³³ Vgl. Crane Payment Solutions (2013), Part 1 S. 11 f.

³⁴ Vgl. Crane Payment Solutions (2013), Part 1 S. 24 f.

Zieladresse	Anzahl an Datenbytes	Quelladresse	Header (Befehl)	Datenbyte 1	...	Datenbyte n	Checksumme
-------------	----------------------	--------------	-----------------	-------------	-----	-------------	------------

Tabelle 4: Datenpaket mit zu übertragenden Nutzdaten, Quelle: Eigene Darstellung.

In seiner einfachsten Form besteht ein Datenpaket aus lediglich fünf Bytes, die keine Nutzdaten enthalten (siehe Tabelle 5). Dieses Datenpaket wird durch den Wert 0 an der Stelle der Datenbytes signalisiert. Hierbei handelt es sich beispielsweise um einen einfachen Poll vom Hostcontroller oder um eine ACK/NAK-Nachricht (engl. Acknowledge oder Not Acknowledge) von einem der Peripheriegeräte.

Zieladresse	0	Quelladresse	Header (Befehl)	Checksumme
-------------	---	--------------	-----------------	------------

Tabelle 5: Datenpaket ohne Nutzdaten, Quelle: Eigene Darstellung.

Adressbereiche

Der Overhead eines ccTalk-Datenpaketes enthält die Ziel- und Quelladresse. Die Zieladresse kann dabei einen Wert von 0 bis 255 annehmen. Theoretisch erlaubt das Bussystem somit eine maximale Teilnehmeranzahl von bis zu 256 Geräten am Bus. Allerdings ist die Adresse 0 für die sogenannte Broadcast-Nachricht reserviert. In diesem speziellen Fall wird ein Datenpaket an die Zieladresse 0 gesendet, worauf alle Teilnehmer am Bus antworten. Da es leicht zu Kollisionen im System kommen kann, ist die Verwendung der Broadcast-Nachricht mit Vorsicht zu behandeln. Des Weiteren ist die Adresse 1 für den Hostcontroller bzw. den Master im Bussystem reserviert. Für die Peripheriegeräte bzw. Slaves stehen somit die übrigen 254 Adressen (Adresse 2 bis 255) zur Verfügung. Im Gegensatz zur Zieladresse ist für die Quelladresse nur der Adressbereich von 1 bis 255 definiert.³⁵

Für Bussysteme mit mehreren Slaves, sogenannte Multi-Drop-Netzwerke, sind in der Spezifikation für jede Art bzw. Equipment-Kategorie von Peripheriegerät bestimmte Adressbereiche vordefiniert. Die folgende Tabelle 6 zeigt einen Auszug einiger dieser Adressbereiche:³⁶

Equipment-Kategorie	Adresse	Zusätzlicher Adressbereich
Broadcast-Message	0	-
Hostcontroller	1	-
Coin Acceptor	2	11 bis 17
Payout (Hopper)	3	4 bis 10
Bill Validator	40	-
Card Reader	50	-
Ticket Printer	110	-

Tabelle 6: Definierte Adressbereiche aus der Spezifikation, Quelle: In Anlehnung an Crane Payment Solutions (2013), Part 3 S. 61.

³⁵ Vgl. Crane Payment Solutions (2013), Part 1 S. 26.

³⁶ Vgl. Crane Payment Solutions (2013), Part 3 S. 61.

Header

Die Befehle bzw. Kommandos werden in der ccTalk-Spezifikation als Header bezeichnet. Im Datenpaket steht für den Header exakt ein Byte zur Verfügung und umfasst somit ebenfalls einen Bereich von 0 bis 255. Den einzelnen Headern sind im Protokoll spezielle Tätigkeiten zugeordnet, welche in Teil 2 der allgemeinen ccTalk-Spezifikation eingesehen werden können. Hat der Header den Wert 0, dann handelt es sich z.B. immer um ein Antwortpaket des Slaves auf einen vorangegangenen Poll vom Master.³⁷

Eine genaue Beschreibung der Header, die für den zu entwickelnden Emulator von Bedeutung sind, folgt im weiteren Verlauf dieser Arbeit.

Checksumme

Die Checksumme steht am Ende jedes Datenpaketes und dient zur Erkennung von Übertragungsfehlern in der Kommunikation. In der ccTalk-Spezifikation unterscheidet man zwischen den folgenden beiden Varianten:³⁸

- **Einfache Checksumme**

Die einfache Checksumme ist eine Nullsummen-Checksumme und basiert auf einer 8-Bit-Addition (Modulo 256). Hierfür werden die Dezimalwerte aller Bytes des jeweiligen Datenpaketes zuerst aufsummiert und dann durch 256 dividiert. Der Rest aus der Division muss immer 0 ergeben. Ist der Rest ungleich 0, weist dies auf Fehler in der Übertragung hin. Mit der einfachen 8-Bit-Checksumme lassen sich alle Einzelbitfehler und fast alle Doppelbitfehler erkennen. Eine noch bessere Doppelbitfehler-Erkennung lässt sich durch einfache 16-Bit-Checksummen erzielen.

Die folgende Tabelle 7 zeigt ein 8-Bit-Checksummen-Beispiel anhand einer ACK-Nachricht. In der ersten Zeile ergibt die Summe aller Datenbytes inklusive der berechneten Checksumme den Wert 256. Dividiert man diesen durch 256, so ergibt der Rest 0 und die Übertragung ist in Ordnung. Baut man allerdings, wie in der zweiten Zeile zu sehen ist, einen Übertragungsfehler bei z.B. der Quelladresse ein, dann ergibt die Summe einen Wert, der nicht mehr ohne Rest durch 256 teilbar ist. Dies weist schließlich auf einen Fehler in der Übertragung hin und wird durch das Protokoll erkannt.

Datenpaket							
Zieladr.	Anzahl an Datenbytes	Quelladr.	Header	Checksumme	Summe	Modulo 256	Rest
1	0	2	0	253	= 256	/ 256	0
1	0	5	0	253	= 259	/ 256	≠ 0 (Error)

Tabelle 7: Einfache Checksummen-Berechnung, Quelle: Eigene Darstellung

³⁷ Vgl. Crane Payment Solutions (2013), Part 1 S. 29.

³⁸ Vgl. Crane Payment Solutions (2013), Part 1 S. 29 f.

- **CRC-Checksumme (Cyclic Redundancy Check)**

Bei den verwendeten CRC-Checksummen handelt es sich um CRC-16, CRC-16-CCIT oder CRC-32-Berechnungen, deren Basis eine mathematische Polynomdivision bildet. Mit diesen Varianten der Checksummen lassen sich alle Einzelbitfehler und Doppelbitfehler, sowie alle ungeraden Fehleranzahlen erkennen. Allerdings wird für die Berechnung von CRC-Checksummen mehr Rechenleistung benötigt als bei einfachen Additions-Checksummen.

Da für die meisten Situationen einfache 8-Bit-Checksummen ausreichend sind und auch für den zu entwickelnden Emulator ausschließlich auf die einfache Form der Checksummen-Berechnung zurückgegriffen werden muss, werden die CRC-Checksummen hier nicht näher erläutert.

Was genau nun beim Auftreten von Übertragungsfehlern passieren soll und wie die Fehlerbehandlung aussieht ist im ccTalk-Protokoll nicht eindeutig festgelegt. Manche Protokolle verwenden Funktionen zur Neuübertragung der Daten, die direkt in die Transportschicht integriert sind. Um ccTalk jedoch einfach zu halten, wurde bewusst darauf verzichtet und die Möglichkeit zur Fehlerbehandlung besteht somit nur in höheren Ebenen des Protokolls. Je nach Anforderungen der spezifischen Anwendung wird daher die Implementierung einfacher Algorithmen zur Fehlerbehandlung dringend empfohlen, um die Robustheit des Systems zu verbessern.³⁹

Datenübertragungsbeispiele

Zum besseren Verständnis über die Funktionsweise des ccTalk-Protokolls wird im Folgenden die Kommunikation zwischen Master und Slave anhand einiger Beispiele aus der Praxis kurz näher erläutert.

Tabelle 8 zeigt einen Adress-Poll an die Broadcast-Adresse 0. Alle Peripheriegeräte bzw. Slaves am Bus antworten mit ihrer eigenen Adresse, wobei die Antworten nur aus den wirklichen Nutzdaten bestehen und keinen Overhead enthalten. Im vorliegenden Beispiel gibt es am Bus also drei Teilnehmer mit den Slave-Adressen 2, 3 und 4.

Spielplattform (Master)					Alle Slaves		
Zieladresse	Anzahl an Datenbytes	Quelladresse	Header	Checksumme	Datenbytes		
0	0	1	253	2	2	3	4

Tabelle 8: Broadcast Adress-Poll (Header 253), Quelle: Eigene Darstellung.

Tabelle 9 stellt einen Reset-Poll an einen Münzprüfer mit der Zieladresse 2 dar. Der Münzprüfer antwortet mit einer einfachen ACK-Nachricht, womit dieser die Ausführung des Resets bestätigt.

Spielplattform (Master)					Münzprüfer (Slave)				
Zieladresse	Anzahl an Datenbytes	Quelladresse	Header	Checksumme	Zieladresse	Anzahl an Datenbytes	Quelladresse	Header	Checksumme
2	0	1	1	252	1	0	2	0	253

Tabelle 9: Reset-Poll (Header 1), Quelle: Eigene Darstellung.

³⁹ Vgl. Crane Payment Solutions (2013), Part 1 S. 34 f.

Tabelle 10 zeigt die Abfrage der Hersteller ID eines Münzprüfers mit der Zieladresse 2. Der Münzprüfer antwortet auf die Abfrage mit drei Datenbytes, welche eine ASCII-Zeichenfolge darstellen. Der Hersteller dieses Münzprüfers wird somit mit „CMG“ bezeichnet.

Spielplattform (Master)					Münzprüfer (Slave)							
Zieladresse	Anzahl an Datenbytes	Quelladresse	Header	Checksumme	Zieladresse	Anzahl an Datenbytes	Quelladresse	Header	Datenbytes			Checksumme
2	0	1	246	7	1	3	2	0	67 ('C')	77 ('M')	71 ('G')	35

Tabelle 10: Abfrage der Hersteller ID (Header 246), Quelle: Eigene Darstellung.

Tabelle 11 stellt die Kommunikation zwischen der Spielplattform und einer Münzauszahlereinheit mit der Zieladresse 4 schematisch dar. Die Spielplattform schickt dabei den Header 164 gefolgt von einem Datenbyte mit dem Wert 165. Dieser Befehl dient zur Aktivierung der Münzauszahlereinheit. Diese antwortet zur Bestätigung des Polls mit der einfachen ACK-Nachricht.

Spielplattform (Master)						Münzauszahlereinheit (Slave)				
Zieladresse	Anzahl an Datenbytes	Quelladresse	Header	Datenbytes	Checksumme	Zieladresse	Anzahl an Datenbytes	Quelladresse	Header	Checksumme
4	1	1	164	165	177	1	0	4	0	251

Tabelle 11: Aktivierung der Münzauszahlereinheit (Header 164), Quelle: Eigene Darstellung.

3.2.3.2 Spielplattform-Software

Die Spielplattform-Software wird auf der Hardware-Plattform des jeweiligen Spielautomaten ausgeführt. Diese besteht grundsätzlich aus dem Betriebssystem, der Foundation und aus der Spielesoftware. Die Spielesoftware bildet die Hauptapplikation des Spielautomaten und enthält somit je nach Modell-, Markt- und Kundenanforderung das eigentliche Spielepaket. In diesem Teil der Software ist somit das gesamte Spiel mit allen Spielabläufen, Grafiken, Spielstatistiken und allen zugehörigen mathematischen Algorithmen zur Berechnung von Zufallszahlen implementiert. Den Grundstein für die Spielesoftware bildet hingegen die Foundation. Diese beinhaltet die komplette Implementierung, welche zur Ansteuerung aller Hardwarekomponenten im Spielautomaten benötigt wird. Hierzu zählen beispielsweise Software-Module zur Ansteuerung der gesamten Beleuchtung, die Ansteuerung der Buttons und Türschalter und die Treiberimplementierung von Komponenten des Bezahlsystems. Darüber hinaus befinden sich in der Foundation alle notwendigen Schnittstellen zur Spielesoftware und die Implementierung von Protokollen, die zur Anbindung des Spielautomaten an Backoffice-Systeme benötigt werden. Die Basis für die Foundation und die Spielesoftware bildet ein herkömmliches Linux- oder Windows-basiertes Betriebssystem, welches schließlich für die Verwaltung der gesamten Systemressourcen der Hardware-Plattform zuständig ist.

Implementierung der ccTalk-Header

Für die Entwicklung des Emulators des automatisierten Münzverarbeitungssystems gilt es einzig zu verstehen, wie das ccTalk-Protokoll in der Foundation implementiert ist und welche ccTalk-Header von der Software abgefragt werden. Die Foundation für die gegebenen Märkte ist in der Programmiersprache C++ entwickelt. Dies macht es sehr einfach den Quellcode der Foundation auf die Implementierung des ccTalk-Treibers zu analysieren. Dieses Device API (Application Programming Interface) besteht aus dem Bus-

Treiber, dem Core Command Set, dem Treiber für Münzprüfer und Auszahlereinheit und einigen weiteren Bibliotheken für z.B. die Berechnung von Checksummen oder ähnlichem. Um herauszufinden, auf welche Header der Emulator reagieren muss, ist lediglich das Core Command Set des Treiberbausteines zu analysieren.

Die ccTalk-Dokumentation definiert bis zu 256 verschiedene ccTalk-Befehle bzw. Header. Zur Ansteuerung des elektronischen Münzprüfers und der Münzauszahlereinheiten dient aber nur ein Bruchteil davon. Durch die Analyse des Core Command Sets im Treiber-Quellcode der Foundation stellt sich heraus, dass auch von diesen Headern nur einige tatsächlich im Code implementiert sind (siehe Abbildung 20).

```
24 // Headers for CCTalk
25 static constexpr std::uint8_t RESET_DEVICE = 1;
26 static constexpr std::uint8_t REQUEST_COMMS_REVISION = 4;
27 static constexpr std::uint8_t REQUEST_CIPHER_KEY_HEADER = 160;
28 static constexpr std::uint8_t TEST_HOPPER = 163;
29 static constexpr std::uint8_t ENABLE_HOPPER = 164;
30 static constexpr std::uint8_t REQUEST_HOPPER_STATUS = 166;
31 static constexpr std::uint8_t DISPENSE_HOPPER_COINS = 167;
32 static constexpr std::uint8_t REQUEST_HOPPER_DISPENSE_COUNT = 168;
33 static constexpr std::uint8_t EMERGENCY_STOP = 172;
34 static constexpr std::uint8_t REQUEST_COIN_ID = 184;
35 static constexpr std::uint8_t REQUEST_BUILD_CODE = 192;
36 static constexpr std::uint8_t REQUEST_SORTER_PATHS = 209;
37 static constexpr std::uint8_t MODIFY_SORTER_PATHS = 210;
38 static constexpr std::uint8_t MODIFY_SORTER_OVERRIDE_STATUS = 222;
39 static constexpr std::uint8_t REQUEST_SORTER_OVERRIDE_STATUS = 221;
40 static constexpr std::uint8_t MODIFY_MASTER_INHIBIT_STATUS = 228;
41 static constexpr std::uint8_t READ_BUFFERED_COIN_EVENTS = 229;
42 static constexpr std::uint8_t REQUEST_INHIBIT_STATUS = 230;
43 static constexpr std::uint8_t MODIFY_INHIBIT_STATUS = 231;
44 static constexpr std::uint8_t READ_OPTO_STATES = 236;
45 static constexpr std::uint8_t REQUEST_SOFTWARE_REVISION = 241;
46 static constexpr std::uint8_t REQUEST_SERIAL_NUMBER = 242;
47 static constexpr std::uint8_t REQUEST_PRODUCT_CODE = 244;
48 static constexpr std::uint8_t REQUEST_EQUIPMENT_CATEGORY_ID = 245;
49 static constexpr std::uint8_t REQUEST_MANUFACTURER_ID = 246;
50 static constexpr std::uint8_t REQUEST_POLLING_PRIORITY = 249;
51 static constexpr std::uint8_t SIMPLE_POLL = 254;
```

Abbildung 20: Auszug der verwendeten ccTalk-Header aus dem Foundation Quellcode, Quelle: Eigene Darstellung.

Im Zuge der Analyse stellt sich allerdings ebenfalls heraus, dass in diesem Command Set einige Header gelistet sind, die ohne andere nicht angeführte Header aus der Spezifikation nicht sinnvoll sind. Durch Gespräche mit anderen Entwicklern stellt zeigt sich, dass manche der Header ohnedies nicht abgefragt werden und nur aus historischen Gründen noch im Software-Stack vorhanden sind. Der Vollständigkeit halber werden den 26 Headern aus Abbildung 20, somit 5 fehlende Header aus der Spezifikation hinzugefügt. All diese Header lassen sich nun in generische Header, d.h. Befehle, die für jegliche Art von Peripheriegerät eingesetzt werden können, und spezifische Header, die für Münzprüfer bzw. Münzauszahlereinheiten eingesetzt werden, einteilen.

Im Folgenden erfolgt die Einteilung der Header in die soeben erwähnten Gruppen mit einer kurzen Beschreibung aus Teil 2 der ccTalk-Dokumentation:⁴⁰

Allgemeine Header

Header	Bezeichnung (engl.)	Kurzbeschreibung
1	Reset Device	Löst einen Soft-Reset auf der Seite des Slaves aus. Was genau beim Reset passiert, hängt von der Art des jeweiligen Geräts ab. Meist bringt es das Gerät in einen Startzustand und löscht sämtliche Zählerstände. Die Antwort des Slaves auf einen Reset erfolgt mit der ACK-Nachricht.
4	Request Comms Revision	Dient zur Abfrage von Release Nummer und Revision der ccTalk-Spezifikation, die auf dem Gerät implementiert ist.
192	Request Build Code	Dient zur Abfrage des Build Codes, welcher in einer ASCII-Zeichenfolge geschickt wird. Spezielle Formvorschriften gibt es keine.
236	Read Opto States	Fragt den Status von verschiedenen Lichtschranken des Gerätes ab. Dies können beispielsweise Füllstandssensoren bei Auszähleinheiten sein. Die genaue Definition ist allerdings der Gerätebeschreibung zu entnehmen. Empfangen wird eine Bitmaske, die aus 8 Bit besteht.
241	Request Software Revision	Wird zur Abfrage der Software Revision des Slaves verwendet. Die Antwort beinhaltet eine ASCII-Zeichenfolge, für die es keine speziellen Formvorschriften gibt.
242	Request Serial Number	Fordert die Seriennummer des Slaves an. Die Antwort ist ein genormter Code, der aus drei Datenbytes besteht und aus dem sich die Seriennummer errechnet.
244	Request Product Code	Fragt den Product Code, welcher z.B. die Modellbezeichnung des Peripheriegeräts beinhaltet, ab. Formvorschriften gibt es hierfür keine, aber es gibt bei manchen Geräten spezielle Codes zur Identifikation unterschiedlicher ccTalk-Standards.
245	Request Equipment Category ID	Dieser Befehl dient zur Abfrage der Kategorie des jeweiligen Peripheriegeräts. Die verschiedenen Kategorien sind in der Spezifikation gelistet (z.B. Payout, Coin Acceptor etc.).
246	Request Manufacturer ID	Fragt den genauen Namen des Herstellers in Form einer ASCII-Zeichenfolge ab (z.B. Azkoyen, CMG etc.).

⁴⁰ Vgl. Crane Payment Solutions (2013), Part 2 S. 7 ff.

253	Request Address	Ruft die genaue Slave-Adresse vom Gerät ab. Die Antwort ist das entsprechende Adressbyte.
254	Simple Poll	Dieser Befehl wird verwendet, um zu kontrollieren, ob das adressierte Peripheriegerät am Bus angeschlossen ist und funktioniert.

Tabelle 12: Auflistung der allgemeinen ccTalk-Header, Quelle: Eigene Darstellung.

Spezifische Header für Münzprüfer

Header	Bezeichnung (engl.)	Kurzbeschreibung
184	Request Coin ID	Fordert die Coin ID vom jeweiligen Münzkanal des Münzprüfers an. Bei der Coin ID handelt es sich um eine Zeichenfolge für die Identifikation der jeweiligen Münze. Im Regelfall besteht diese aus dem Währungskurzzeichen und dem Münzwert bezogen auf die kleinste verfügbare Einheit (z.B. 2 Euro-Münze = "EU200A").
188	Request Default Sorter Path	Dieser Befehl fragt den Default-Sortierpfad des Münzprüfers bzw. des Sortierers ab. Ist die Bitmaske für den Sorter Override Status (Header 221, 222) gesetzt, dann werden die Münzen in den hier definierten Pfad sortiert. Diese Funktion wird z.B. genutzt, wenn Münzboxen voll sind und die Münzen deshalb in alternative Pfade sortiert werden müssen.
189	Modify Default Sorter Path	Mit diesem Befehl kann der Default-Sortierpfad modifiziert werden.
209	Request Sorter Paths	Dieser Befehl fordert den Sortierpfad für den jeweiligen Münzkanal des Münzprüfers bzw. des Sortierers an. Jeder Münzwertigkeit lassen sich somit bestimmte Sortierpfade zuordnen. ACHTUNG: In der Spezifikation gibt es unterschiedliche Formate (Single oder Multiple Paths).
210	Modify Sorter Paths	Mit diesem Befehl lassen sich die Sortierpfade modifizieren.
221	Request Sorter Override Status	Dieser Befehl fragt die Bitmaske (8 Bit) für den Override Status des Münzprüfers ab. Jedes der 8 Bits repräsentiert einen Sortierpfad. Ist das jeweilige Bit gesetzt, dann wird in die definierten Sortierpfade (Header 209, 210) sortiert. Ändert sich das Bit für einen Sortierpfad von 1 auf 0, erfolgt die Sortierung der Münzen dieses Pfads in den Default-Sortierpfad (Header 188, 189).

222	Modify Sorter Override Status	Mit diesem Befehl lässt sich die Bitmaske für den Sorter Override Status modifizieren.
227	Request Master Inhibit Status	Fragt den Master-Blockier-Status des Münzprüfers ab. Ist dieser gesetzt, dann werden alle eingeworfenen Münzen vom Münzprüfer blockiert.
228	Modify Master Inhibit Status	Mit diesem Befehl lässt sich der Master-Blockier-Status des Münzprüfers setzen und zurücksetzen.
229	Read Buffered Credit or Error Codes	Hierbei handelt es sich um den wichtigsten Befehl des Münzprüfers. Mit ihm lässt sich der Ereignisdatenspeicher, in welchem die eingeworfenen Münzen gespeichert werden, auslesen. Weiters beinhaltet der Buffer auch auftretende Fehler, die durch einen speziellen Fehlercode dargestellt werden.
230	Request Inhibit Status	Dieser Befehl fordert den Blockier-Status einzelner Münzkanäle des Münzprüfers ab. Ist der Blockier-Status für einen bestimmten Kanal gesetzt, dann werden die eingeworfenen Münzen dieses Kanals blockiert.
231	Modify Inhibit Status	Mit diesem Befehl lässt sich der Blockier-Status einzelner Münzkanäle des Münzprüfers setzen und zurücksetzen.
249	Request Polling Priority	Fordert die Zykluszeit an, mit der der Münzprüfer vom Host gepollt werden muss.

Tabelle 13: Auflistung der spezifischen ccTalk-Header für Münzprüfer, Quelle: Eigene Darstellung.

Spezifische Header für Münz auszahleinheiten

Header	Bezeichnung (engl.)	Kurzbeschreibung
160	Request Cipher Key	Dieser Befehl fordert den Verschlüsselungscode von der Münz auszahleinheit an. Dieser wird allerdings nur dann benötigt, wenn für die Auszahlung von Münzen ein Verschlüsselungs-Algorithmus verwendet wird. Im vorliegenden Fall wird dieser nicht benötigt.
163	Test Hopper	Der Test Hopper Befehl wird gesendet, um der Münz auszahleinheit zu signalisieren, dass diese einen Selbsttest durchführen soll. Wie dieser Test genau funktioniert, hängt vom jeweiligen Modell ab. Als Antwort auf diesen Befehl schickt die Münz auszahleinheit die Statusregister, welche unterschiedliche Betriebs- und Error-Flags enthalten.

164	Enable Hopper	Dieser Befehl muss einmalig vor der Münzauszahlung gesendet werden, um die jeweilige Münzauszahleinheit zu aktivieren. Zur Aktivierung muss ein Datenbyte mit dem Wert 165 gesendet werden. Alle anderen Werte ungleich 165 oder ein Soft-Reset deaktivieren die Auszahleinheit wieder.
166	Request Hopper Status	Mit diesem Befehl lässt sich der Status von Münzauszahlungen abfragen. Die Auszahleinheit schickt als Antwort darauf 4 Datenbytes: Event Counter, Anzahl der noch auszubezahlenden Münzen, Anzahl der zuletzt ausbezahlten Münzen, Anzahl der noch nicht ausbezahlten Münzen.
167	Dispense Hopper Coins	Dies ist der wichtigste Befehl für die Kommunikation mit der Münzauszahleinheit. Mit ihm wird die Auszahlung von bis zu 255 Münzen gestartet. Während bei der unverschlüsselten Version für die Auszahlung nur die Seriennummer und die Anzahl der auszubezahlenden Münzen an die Auszahleinheit gesendet werden muss, muss bei verschlüsselten Versionen auch der Verschlüsselungscode (Header 160) gesendet werden.
168	Request Hopper Dispense Count	Dieser Befehl dient zur Abfrage der Anzahl aller Münzen, die ausbezahlt wurden. Es handelt sich dabei um einen 24 Bit Dezimalzähler, der durch das Senden eines Soft-Resets wiederum zurückgesetzt werden kann.
172	Emergency Stop	Mit diesem Befehl ist es möglich eine gerade ablaufende Münzauszahlungs-Sequenz sofort zu unterbrechen. Die Antwort der Auszahleinheit besteht aus einem Datenbyte, welches die noch auszubezahlenden Münzen in dieser Sequenz beinhaltet.

Tabelle 14: Auflistung der spezifischen ccTalk-Header für Münzauszahleinheiten, Quelle: Eigene Darstellung.

3.3 Definition der Anforderungen an den Emulator

Ziel von Kapitel 3 ist es, eine geeignete Anforderungsspezifikation für den zu entwickelnden Emulator zu erstellen. Mithilfe des zuvor erarbeiteten theoretischen Wissens über die Haupttätigkeiten des Requirements-Engineerings (siehe Kapitel 3.1.2) und die richtige Gliederung der Anforderungen (siehe Kapitel 3.1.3) gilt es im nächsten Schritt die wesentlichen Anforderungen in einer geeigneten Form festzuhalten. Zu diesem Zweck werden im Folgenden die wichtigsten Erkenntnisse aus der Systemanalyse (siehe Kapitel 3.2) erläutert. Die anschließende Definition der Anforderungen stützt sich auf diese Erkenntnisse und auf Gespräche mit den verantwortlichen Stakeholdern.

3.3.1 Erkenntnisse aus der Systemanalyse

Die hardware- und softwareseitige Analyse hat Aufschluss über den Aufbau und die genaue Funktionsweise des automatisierten Münzverarbeitungssystems gegeben. Fehlende und unklare Informationen werden in enger Zusammenarbeit mit den Stakeholdern ergänzt. Im Folgenden sind die wichtigsten Erkenntnisse aus der Analyse zusammengefasst:

- **Zielgruppe**

Die Zielgruppe des Emulators bilden ausschließlich Mitarbeiter*innen der Hardware- und Software R&D, sowie der SQA-Abteilung von IGT Austria GmbH.

- **Systemkomponenten**

Das Gesamtsystem für die Münzverarbeitung im Spielautomaten besteht immer aus einem Münzprüfer und einem oder mehreren Münzauszahlseinheiten. Hierbei können unterschiedliche Modelle zum Einsatz kommen, die von den Markt- und Kundenanforderungen abhängen.

- **Anbindung der Hardware**

Die einzelnen Komponenten sind an den sogenannten ccTalk-Bus angeschlossen. Für die elektrische Anbindung an eine herkömmliche serielle RS-232 Schnittstelle ist eine spezielle elektronische Schaltung zu realisieren.

- **ccTalk-Protokoll**

Das Protokoll arbeitet nach dem Master/Slave-Prinzip und ist relativ einfach aufgebaut. Die Slaves werden dabei in einem definierten Zeitintervall vom Master gepollt. Für die relevanten Märkte ist vorerst keine Verschlüsselung vorgesehen und nur die einfache 8 Bit Checksummen-Berechnung findet Anwendung. Weiters ist in der Spielplattform-Software nur ein Teil der ccTalk-Header aus der Spezifikation implementiert.

3.3.2 Anforderungen an den Emulator

Bezugnehmend auf die zweite Haupttätigkeit des Requirements-Engineerings in Kapitel 3.1.2, das Dokumentieren, stehen unterschiedliche Ansätze für die Dokumentation zur Verfügung. Zur besseren Verständlichkeit fällt die Wahl für die hier auszuarbeitende Anforderungsspezifikation auf die Formulierung

in natürlicher Sprache. Somit wird versucht die wesentlichen Anforderungen richtig zu strukturieren und in möglichst kurzen und aussagekräftigen Sätzen zu konkretisieren.

Diese ausgearbeiteten Anforderungen wurden mehrfachen Reviews mit den Stakeholdern des Unternehmens IGT Austria GmbH unterzogen und anschließend von allen Seiten akzeptiert. In den nachfolgenden Unterkapiteln findet sich ein Auszug der grundlegenden Anforderungen aus der erstellten Anforderungsspezifikation.

3.3.2.1 Funktionale Anforderungen

ID	Beschreibung der Anforderung
F1	Der Emulator muss den elektronischen Münzprüfer und einen oder mehrere Münzauszahleinheiten im Spielautomaten emulieren.
F2	Der Emulator muss dem Spieler*in bzw. dem Tester*in die Möglichkeit bieten, den Münzeinwurf von mindestens 8 verschiedenen Münzwertigkeiten über Buttons oder eine andere entsprechende Benutzerschnittstelle zu simulieren.
F3	Der Emulator muss dem Spieler*in bzw. dem Tester*in die Möglichkeit bieten, ausbezahlte Münzen durch die Münzauszahleinheiten in geeigneter Form zu visualisieren.
F4	Der Emulator muss die Möglichkeit bieten, verschiedene Modelle von elektronischen Münzprüfern und Münzauszahleinheiten zu konfigurieren (siehe Tabelle 2, Tabelle 3). <u>Anmerkung:</u> Die Identifikation des jeweiligen Modells erfolgt durch die Abfrage bestimmter modellspezifischer ccTalk-Header (4, 192, 241, 242, 244, 245, 246, 253). Es muss dem Emulator daher möglich sein, die Parameter für jeden dieser Header in der Konfiguration zu ändern.

Tabelle 15: Funktionale Anforderungen, Quelle: Eigene Darstellung.

3.3.2.2 Qualitätsanforderungen

ID	Beschreibung der Anforderung
Q1	Der Emulator muss sich im Einsatz wie die reale Hard- und Software des automatisierten Münzverarbeitungssystems verhalten und darf keine Störungen oder Fehlermeldungen in der Software des Spielautomaten hervorrufen.
Q2	Die Hardware des Emulators soll eine kompakte Einheit bilden, um sie leicht in den Spielautomaten integrieren zu können.
Q3	Die Software des Emulators soll modular aufgebaut sein, so dass die Wartungsfreundlichkeit und Erweiterbarkeit gewährleistet ist. <u>Anmerkung:</u> Der Emulator könnte zukünftig um die Funktionen eines Ticketprinters oder Banknotenprüfers erweitert werden.

Q4	Die Benutzerschnittstelle des Emulators soll mit Hinblick auf Benutzerfreundlichkeit entworfen werden.
Q5	Die Änderung der Konfiguration muss einer Person mit technischem Verständnis nach einer kurzen Einschulung zumutbar sein.

Tabelle 16: Qualitätsanforderungen, Quelle: Eigene Darstellung.

3.3.2.3 Randbedingungen

ID	Beschreibung der Anforderung
R1	Die Zielplattform des Emulators muss an das ccTalk-Bussystem des Spielautomaten angeschlossen werden. <u>Anmerkung:</u> Für die Anbindung ist die Hardware-Schnittstelle gemäß Abbildung 18 zu realisieren.
R2	Der Emulator muss fähig sein über das in Kapitel 3.2.3.1 erwähnte ccTalk-Protokoll mit der Spielplattform-Software des Spielautomaten zu kommunizieren. <u>Anmerkung:</u> Es sind nur die notwendigen Header zu implementieren, die von der Spielplattform-Software auch tatsächlich abgefragt werden (siehe Tabelle 12, Tabelle 13, Tabelle 14).
R3	Der Emulator muss fähig sein auf die Polling Rate von 2 Hz (500 ms Zykluszeit) durch die Plattform des Spielautomaten zu reagieren.
R4	Die Kommunikation mittels ccTalk-Protokoll erfolgt unverschlüsselt.
R5	Zur Erkennung von Übertragungsfehlern wird nur die einfache 8 Bit Checksummen-Berechnung eingesetzt.

Tabelle 17: Randbedingungen, Quelle: Eigene Darstellung.

4 AUFBAU DER HARDWARE

In diesem Kapitel wird das Hauptaugenmerk auf die Hardware-Plattform gelegt, welche die Basis für den Emulator bildet. Zunächst wird diesbezüglich erläutert auf welchen Aspekten die Auswahl der Zielplattform beruht und worauf letztendlich die konkrete Wahl fällt. Anschließend wird präzisiert, wie die Anbindung der zuvor gewählten Zielplattform an das bestehende ccTalk-Bussystem im Spielautomaten erfolgt und welche zusätzliche Peripherie eingesetzt werden könnte.

4.1 Wahl der Zielplattform für den Emulator

In der Regel steht zu Beginn der Systementwicklung immer die Frage nach einer geeigneten Plattform im Mittelpunkt. Diese soll einerseits möglichst effizient und zuverlässig sein und die definierten Anforderungen an das System erfüllen. Andererseits sind aber auch Faktoren wie die Beschaffbarkeit, der Langzeitsupport, die Time-to-Market und schlussendlich der Preis in die Wahl mit einzubeziehen. Im Vergleich zu vergangenen Zeiten, in denen noch enorm viel Wissen und Arbeitsaufwand für die Entwicklung eigener Hardwarelösungen betrieben wurde, ist der Markt heute gesättigt mit fertigen COTS-Lösungen (engl. Commercial Off-the-Shelf). Dazu zählen verschiedene Formen von Mikrocontroller- und Prototyping-Boards, SBCs (Single-Board Computer), SoCs (System-on-a-Chip) und SoMs (System-on-a-Module), die es für die unterschiedlichsten Anwendungsgebiete gibt. Aber gerade dieses breite Spektrum an Auswahlmöglichkeiten macht es nicht immer einfacher die richtige Wahl zu treffen.

Grundsätzlich halten sich die Anforderungen an die Hardware für den zu entwickelnden Emulator in Grenzen und wegen der geringen Stückzahl ist ebenso der Faktor Preis vernachlässigbar. Die Plattform soll allerdings durch eine möglichst kompakte Bauweise überzeugen, so dass sie schnell und einfach in den Spielautomaten ein- und wieder ausgebaut werden kann. Weiters soll eine unkomplizierte Inbetriebnahme ohne längere Einarbeitungszeiten es ermöglichen, schnell mit der eigentlichen Softwareentwicklung starten zu können, um den Entwicklungsprozess zu beschleunigen. Die Unterstützung von einer der bekannten höheren Programmiersprachen wie C, C++ oder C# ist somit anzudenken. Was die technischen Merkmale der Plattform betrifft, so muss diese über genug Speicherkapazität für den zu implementierenden Programmcode verfügen und genug Rechenleistung bereitstellen, um die ankommenden Polls des Spielautomaten abzarbeiten. Darüber hinaus muss die Plattform mindestens über eine UART-Schnittstelle zur seriellen Buskommunikation verfügen. Um die Flexibilität für die Wahl eines geeigneten Lösungsansatzes zu steigern, sollen außerdem mindestens acht GPIOs (General Purpose Input/Output) zur Verfügung stehen, so dass der Münzeinwurf optional über mechanische Buttons getriggert werden könnte. Zusätzlich soll die Anbindung eines Grafikdisplays die Visualisierung von Münzauszahlungen ermöglichen. Alle diese genannten Aspekte werden aber ohnedies von fast allen gängigen Plattformen am Markt erfüllt, weshalb die Entscheidung nicht leichter zu fällen ist.

In Anbetracht der soeben genannten Aspekte fallen in die engere Auswahl zunächst herkömmliche ARM- oder AVR-basierte Mikrocontroller-Boards, die mit Open-Source-Plattformen wie Arduino oder Mbed außerdem eine benutzerfreundliche Entwicklungsumgebung anbieten. Die große Anzahl an verfügbaren Entwicklungsboards, die in den verschiedensten Ausführungen angeboten werden, ermöglichen somit einen umgehenden Start mit der Softwareentwicklung in den Programmiersprachen C oder C++. Der

Nachteil einer Mikrocontroller-Lösung besteht allerdings in der geringen Flexibilität in Bezug auf unterschiedliche Faktoren. So ist beispielsweise die Portierung von Programmcode auf andere Plattformen teilweise mit größerem Entwicklungsaufwand verbunden, für Grafikdisplays müssen meist spezielle Treiber implementiert werden und für die Triggerung des Münzeinwurfs stehen ausschließlich die GPIOs zur Verfügung.

Resultierend aus den soeben genannten Kehrseiten von Mikrocontroller-Lösungen fällt die endgültige Wahl der Zielplattform auf den Raspberry Pi (RPI) in der Version 4 B. Der bekannte Einplatinencomputer stellt einen vollwertigen Mini-Computer dar und hat zugleich sämtliche Schnittstellen wie die benötigten GPIOs und den UART, als auch zwei HDMI-Videoausgänge (High Definition Multimedia Interface) zur Anbindung von allen gängigen Displays integriert (siehe Abbildung 21). Die Verwaltung aller notwendigen Systemressourcen wird dabei von dem Linux-basierten Raspberry Pi Betriebssystem übernommen. Die auf Grund des Multithreading nur bedingte Echtzeitfähigkeit des RPI ist für die Anwendung des Emulators zu vernachlässigen. Die Zykluszeit für Abfragen des Spielautomaten, auf welche der Emulator reagieren muss, beträgt 500 ms und ist somit unkritisch für die vorgesehene Anwendung. Der entscheidende Auslöser für diese Wahl der Zielplattform ist allerdings die Unterstützung der von Microsoft entwickelten .NET Software-Plattform. Die innerhalb diese Software-Plattform enthaltenen Pakete .NET Core/.NET 5 unterstützen sämtliche Betriebssysteme und Hardware-Architekturen und machen eine plattformübergreifende Entwicklung möglich. Eine genauere Beschreibung der Software-Plattform und der plattformübergreifenden Entwicklung sowie deren Vorteile folgt in Kapitel 5.1.

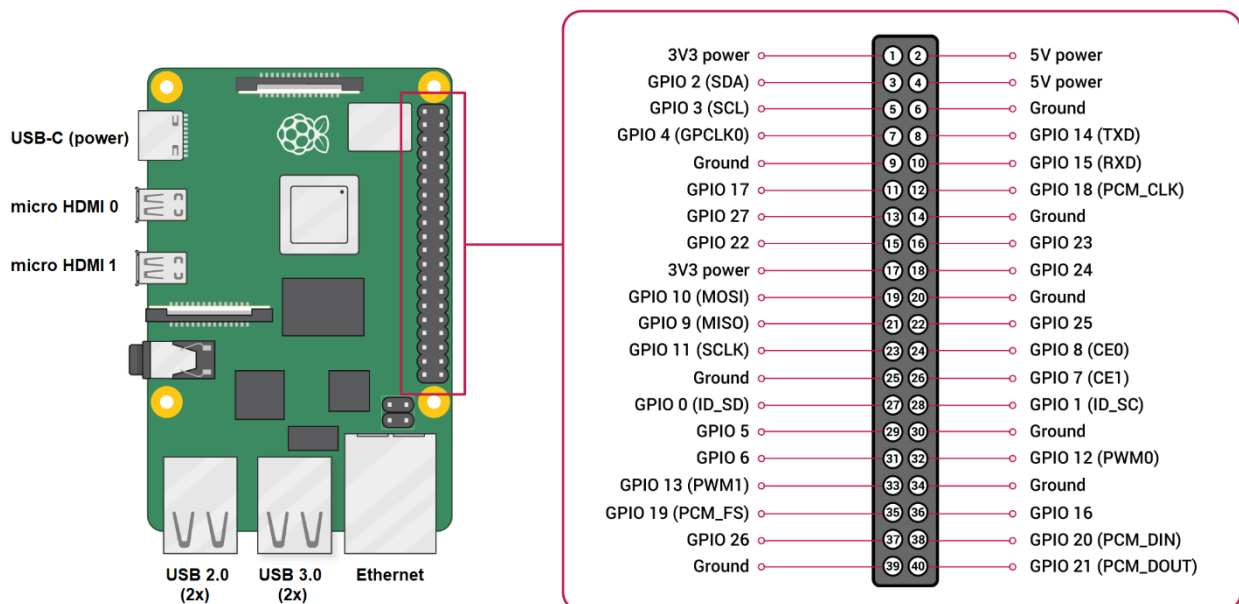


Abbildung 21: Raspberry Pi 4 B Schnittstellen, Quelle: Raspberry Pi Foundation (o.J.), Online-Quelle [24.02.2021], (leicht modifiziert).

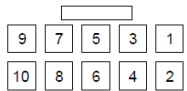
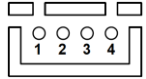
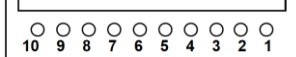
4.2 Bussystemanbindung

Zur Anbindung des Emulators an die Spielplattform des Spielautomaten muss die ausgewählte Zielplattform anstelle der bestehenden Komponenten des Münzverarbeitungssystems an das ccTalk-Bussystem angeschlossen werden. In der Praxis werden dafür zunächst die elektrischen Verbindungen zum Münzprüfer und zu den Münzauszahlseinheiten im stromlosen Zustand des Spielautomaten unterbrochen. Anschließend wird der Emulator elektrisch an den bestehenden Bus angeschlossen. Welche Steckverbindungen hierfür nötig sind und wie die Realisierung der Hardware-Schnittstelle im Detail aussieht, wird nachfolgend beschrieben.

4.2.1 Normung der Steckverbindungen

Die Normung der Steckverbindungen ist keine verpflichtende Anforderung aus der ccTalk-Spezifikation. Dadurch soll den einzelnen Herstellern von Peripheriegeräten genug Entscheidungsspielraum gelassen werden, um je nach Anwendungsgebiet und Anforderung selbst zwischen passenden Stecksystemen zu wählen. Allerdings versucht man durch ein eigenes Kapitel in der Spezifikation mit einer Art Quasistandard die Anzahl an unterschiedlichen Steckertypen zu reduzieren. Dieser Quasistandard definiert bis zu neun unterschiedliche ccTalk-Standard-Steckverbindungen samt Herstellerangaben und genauer Belegung.⁴¹

In der Spielautomaten-Industrie handelt es sich meist nicht um erschwerte Umgebungsbedingungen, weshalb auch auf die in der Spezifikation erwähnten Steckverbindungen zurückgegriffen wird. Diese Normung bringt den Betreibern der Spielautomaten den großen Vorteil, dass die Peripheriegeräte ohne weiteres untereinander ausgetauscht werden können. Bei den elektronischen Münzprüfern haben sich erfahrungsgemäß Steckverbindungen der Type 7 und bei den Münzauszahlseinheiten Steckverbindungen von Type 5 und Type 8 durchgesetzt (siehe Tabelle 18). Allerdings gibt es auch hier teilweise geringfügige Abweichungen was die Steckerbelegung betrifft. Für die Anbindung des Emulators an den Bus sollten aber zumindest diese drei SteckerAusprägungen vorgesehen werden.

Type	Herstellerbezeichnung	Steckerbelegung*	
5	Molex 70246 Series (P/N 70246-1021)	Pin 1: /DATA Pin 4,8: GND Pin 9,10: +Vs	
7	JST XH Series (P/N B 4B-XH-A)	Pin 1: +Vs Pin 3: GND Pin 4: /DATA	
8	AMP MTA-100 Series (P/N 1-640456-0)	Pin 4,5: +Vs Pin 6,7: GND Pin 8: /DATA	

*Ansicht von der Steckervorderseite

Tabelle 18: ccTalk-Standard-Steckverbindungen, Quelle: In Anlehnung an Crane Payment Solutions (2013), Part 1 S. 18 ff.

⁴¹ Vgl. Crane Payment Solutions (2013), Part 1 S. 18 ff.

4.2.2 Realisierung der Hardware-Schnittstelle

Der RPI verfügt nur über den benötigten UART, nicht aber über die cc-Talk-Schnittstelle. Für die Anbindung des Emulators an das ccTalk-Bussystem muss daher ein geeignetes Erweiterungsboard bzw. Shield entwickelt werden, dessen Basis die ccTalk-Schaltung (siehe Abbildung 18) aus Kapitel 3.2.2.2 darstellt. Die Entwicklung des Schaltplans erfolgt unter Zuhilfenahme der Software Autodesk EAGLE. Das Erweiterungsboard soll etwas kleinere Abmessungen als der RPI aufweisen und wird direkt an die 40-polige Stiftleiste des RPIs angeschlossen. An Pin 8 und 9 der Stiftleiste befinden sich die Signale für die Sende- und Empfangsleitung des UARTs (TX und RX). Da die Ein- und Ausgänge des RPIs gemäß der elektrischen Spezifikation nicht 5-V-tolerant sind, müssen die Pegel vom ccTalk-Bus zunächst in kompatible 3,3-V-Logikpegel umgesetzt werden. Hierzu werden zwei einfache bidirektionale Pegelumsetzer im Schaltplan des Erweiterungsboards (siehe Abbildung 22) realisiert. Diese bestehen je aus einem N-KANAL-MOSFET (engl. Metal-Oxide Semiconductor Field-Effect Transistor) und zwei Pull-Up-Widerständen. Nach der Pegelumsetzung erfolgt die eigentliche Schnittstellenumwandlung der Sende- und Empfangsleitung vom UART in die gemeinsame bidirektionale Datenleitung, welche die Anbindung an den Bus ermöglicht. Zu guter Letzt werden im Schaltplan die drei verschiedenen Steckertypen aus Tabelle 18 vorgesehen. Dies soll die einfache Installation der Zielplattform im Spielautomaten möglich machen, weil die bestehenden Komponenten des Münzverarbeitungssystems einfach gegen den Emulator ersetzt werden können.

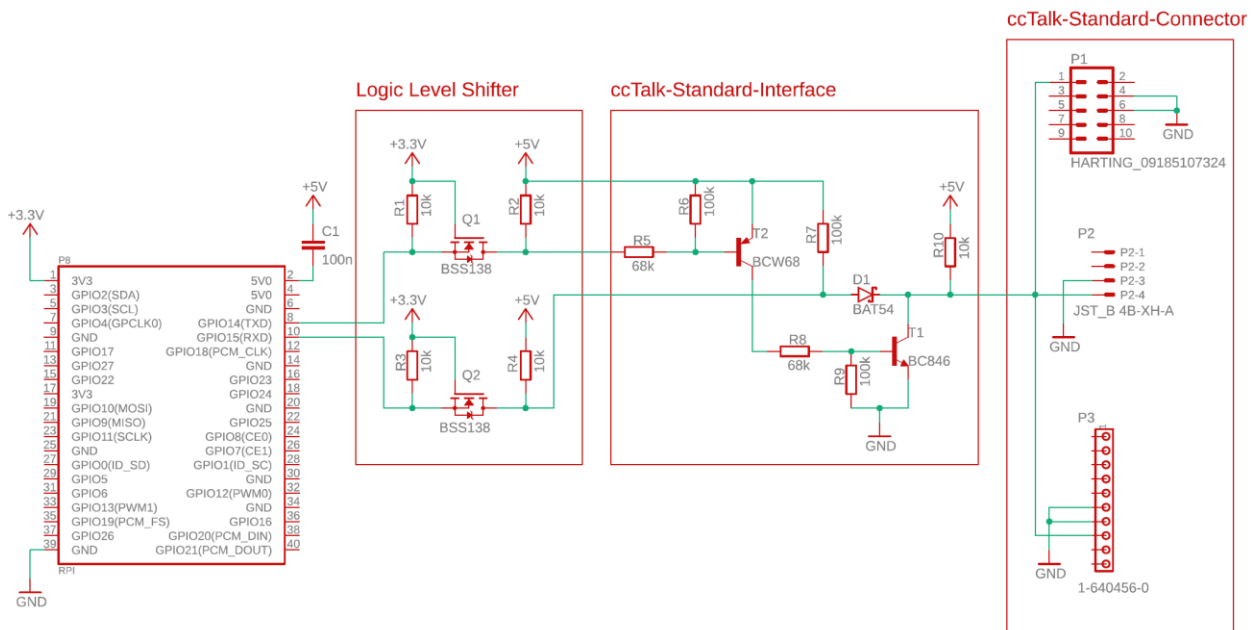


Abbildung 22: Schaltplan des RPI-Erweiterungsboards für die ccTalk-Anbindung, Quelle: Eigene Darstellung.

Um zeiteffizient zu arbeiten, wird für die Entwicklungsphase des Emulators vorerst darauf verzichtet ein eigenes PCB-Layout (engl. Printed Circuit Board) zu erstellen. Der Aufbau eines ersten Prototyps erfolgt somit ausschließlich in diskreter Form auf einem zugekauften Prototyping-Board mit bedrahteten elektronischen Bauteilen. Die nachfolgende Abbildung 23 zeigt den RPI mit dem Prototyp des ccTalk-Erweiterungsboards in einem zugekauften Kunststoffgehäuse. Fürs Erste wird nur einer der gelisteten ccTalk-Standard-Stecker bestückt.

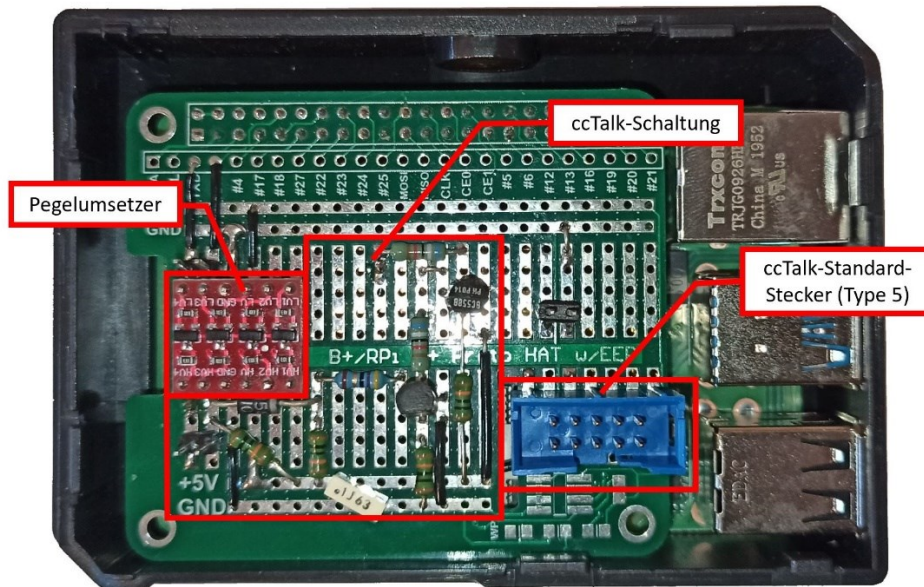


Abbildung 23: Prototyp des RPI mit dem ccTalk-Erweiterungsboard, Quelle: Eigene Darstellung.

4.3 Ein- und Ausgabegeräte

Die Benutzerschnittstellen zwischen dem Emulator und dem Endbenutzer bilden in der Regel immer bestimmte Ein- und Ausgabegeräte. In Bezug auf den Emulator sollen die Eingabegeräte den Trigger für den Münzeinwurf darstellen und die Ausgabegeräte sollen Auszahlungen durch den Spielautomaten visualisieren.

Vorgreifend auf die Softwareentwicklung, welche in Kapitel 5 thematisiert wird, soll an diesem Punkt erwähnt werden, dass es sich bei der angestrebten Lösung um eine einfache Konsolenanwendung handelt. Im Entwicklungsstadium soll die Steuerung des Emulators daher einzig in Form textbasierter Kommandos über eine handelsübliche Tastatur erfolgen. Darüber hinaus soll sich auch die Visualisierung von Auszahlungen durch die Emulation der Münzauszahlseinheiten auf Standard-Computer-Peripherie in der Konsolenanwendung beschränken.

Für die spätere Kleinserie, bei welcher die Software nur noch am RPI ausgeführt wird, soll der Emulator eine kompakte Einheit formen. Auf Grund der kleinen Bauform und der Möglichkeit zur Stapelung mehrerer Hardware-Module hat der RPI die besten Voraussetzungen dafür. Eine beispielhafte Lösung einer solchen Einheit wäre demnach ein modulares Produktkonzept, bei dem das Grundmodul der RPI selbst bildet. Darauf könnte das ccTalk-Erweiterungsmodul, welches im vorherigen Kapitel erläutert wurde, gesteckt werden. Auf einem sich darüber befindenden Eingabemodul, das mit mehreren Tastern zur Ansteuerung der GPIOs bestückt ist, werden die Münzeinwürfe getriggert. An oberster Ebene soll schließlich ein kleines Display-Modul, welches in unterschiedlichsten Ausführungen zugekauft werden kann, montiert werden und die Inhalte der Konsolenanwendung an die Endnutzer*innen ausgeben.

5 KONZEPTIONIERUNG UND IMPLEMENTIERUNG DER SOFTWARE

Der nächste Abschnitt dieser Arbeit wendet sich der Konzeptionierung und Implementierung der Software für den Emulator zu. Dazu werden zunächst erste Überlegungen in Bezug auf das zu verwendende Software-Framework angestellt. Anschließend folgt eine kurze Einführung in die wichtigsten Systemkomponenten der Softwarearchitektur und dessen Entwurfsmuster (engl. Design Pattern), welches die Grundlage für die eigentliche Entwicklung schafft. Der größte Teilbereich des Kapitels konzentriert sich auf die Entwicklung der Emulator-Software, worin der Aufbau und die einzelnen Module der Software im Detail beschrieben werden.

5.1 Erste Überlegungen

Zu Beginn der Entwicklung stehen unterschiedliche Ansätze und Überlegungen für einen geeigneten Lösungsweg im Raum. Die Aspekte der Hardware, als auch die der Software, stehen hierbei in einer gewissen Wechselwirkung zueinander. So ermöglicht eine spezifische Hardware deren Programmierung bekanntlich nur in einer oder mehreren bestimmten Programmiersprachen. Während sich die meisten Mikrocontroller nur in Maschinensprache (engl. Assembler), C oder C++ programmieren lassen, kann man auf SBCs meist vollwertige Betriebssysteme ausführen, welche wiederum die Basis für die Programmierung in diversen Hochsprachen und verschiedensten Software-Frameworks bilden.

Eines der bekanntesten Software-Frameworks ist die von Microsoft entwickelte .NET-Plattform, welche eine Rahmenstruktur für die objektorientierte Softwareentwicklung bereitstellt. Die Zeiten, in denen .NET ausschließlich unter Windows Betriebssystemen eingesetzt werden konnte, gehören seit der Einführung von .NET Core allerdings der Geschichte an. Diese Variante ermöglicht die Unterstützung sämtlicher Betriebssysteme und Hardware-Architekturen. Dazu zählt neben vielen anderen schließlich auch die Linux-Distribution des Raspberry Pi Betriebssystems. Nicht zuletzt die Sicherstellung der einfachen Portierbarkeit von Programmcode, als auch die Möglichkeit der Programmierung in der bekannten Programmiersprache C# machen somit das .NET Core bzw. in der neuesten Version .NET 5 genannte Software-Framework zum absoluten Favoriten für die Emulator-Software.

5.1.1 Das .NET 5-Framework

.NET 5 ist ein quelloffenes Software-Framework, welches auf der seit vielen Jahren beliebten .NET-Plattform basiert. Es ist das Ergebnis der Zusammenführung des bisher bekannten .NET Framework und .NET Core zu einer einheitlichen Plattform und wurde im November 2020 offiziell von Microsoft freigegeben. Das Standard .NET Framework inkludiert die Common Language Runtime (CLR) zur Ausführung des Programmcodes und die Base Class Library (BCL), welche die entsprechenden Klassen zur Erstellung der Applikationen beinhaltet. Es wurde 2002 speziell für Windows entwickelt und ermöglicht beispielsweise die Erstellung von Windows Forms- oder Active Server Page-Applikationen (ASP), nicht aber plattformübergreifender Anwendungen. Da das moderne Zeitalter von mobilen Geräten, Cloud Computing, Internet of Things (IoT) etc. immer mehr Flexibilität in Bezug auf unterschiedliche Plattformen fordert, hat Microsoft im Jahr 2015 das .NET Core-Framework entwickelt. Dazu wurden alle an Windows geknüpften Teile des .NET Frameworks entfernt und die Runtime in CoreCLR und die Klassenbibliothek in

CoreFX umbenannt, da nur mehr nicht an Windows geknüpften Kernelemente vorhanden sind. Das Ergebnis ist ein plattformunabhängiges Software-Framework das zugleich mit Windows, macOS und Linux kompatibel ist. Die allgemeine Architektur des nun wiederum vereinheitlichten .NET 5-Frameworks wird in der folgenden Abbildung 24 veranschaulicht.⁴²

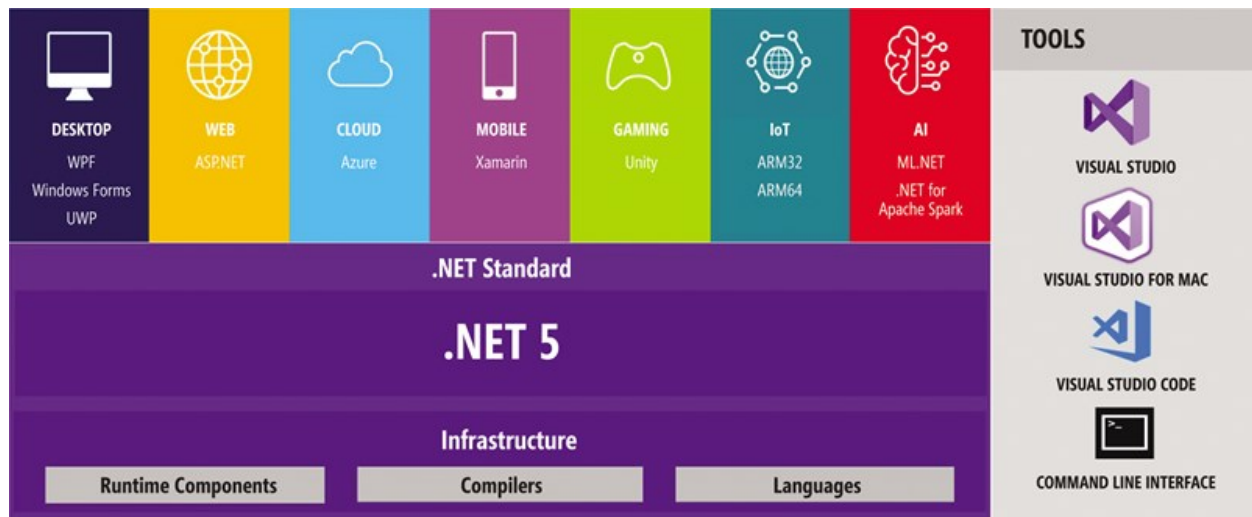


Abbildung 24: Die vereinheitlichte .NET 5-Architektur, Quelle: Microsoft Corporation (2019), Online-Quelle [24.02.2021].

5.1.2 Plattformübergreifende Entwicklung

Wenn es um die Frage nach einfacher Portierbarkeit von Programmcode geht, dann lautet das Stichwort „Plattformübergreifende Entwicklung (engl. Cross-Plattform Development)“. Genau diesen entscheidenden Vorteil bringen das .NET Core- sowie .NET 5-Framework mit sich. Einerseits kann so die Entwicklung des Programmcodes auf einer Alternativplattform, anstelle der eigentlichen Zielplattform, erfolgen. Andererseits ist es aber auch im Nachhinein noch möglich den erstellten Programmcode einfach auf andere mit dem Software-Framework kompatible Zielplattformen zu portieren.

Zur Unterstützung des Software-Frameworks auf der jeweiligen Plattform ist es fürs erste wichtig, dass die entsprechende Laufzeitbibliothek (engl. Runtime Library) und das Software Development Kit (SDK) installiert sind. Die Runtime Library enthält alle Bibliotheken und Softwarefunktionen, die benötigt werden, um ein bereits kompiliertes Programm auf der Zielplattform auszuführen. Zusätzlich wird auf der Entwicklungsplattform das SDK benötigt, um den geschriebenen Code interpretieren und kompilieren zu können. Alle diese Pakete sind für die jeweilige Zielplattform auf der Webseite von Microsoft kostenlos zum Download verfügbar.

Für die Entwicklung des Programmcodes werden von Microsoft verschiedene Entwicklungsumgebungen (engl. Integrated Development Environment, IDE) zur Verfügung gestellt. Unter Windows kann somit wie in gewohnter Weise in der Visual Studio IDE entwickelt werden. Auch für das macOS steht bereits seit 2017 eine eigene Visual Studio Version zur Verfügung. Um direkt unter Linux entwickeln zu können, empfiehlt es sich hingegen die Visual Studio Code IDE zu verwenden. Visual Studio Code ist

⁴² Vgl. Price (2019), S. 7 ff.

plattformübergreifend und kann unter Windows, macOS und Linux installiert werden. Eine weitere Möglichkeit besteht in der Entwicklung mit dem .NET Kommandozeilenfenster (engl. Command Line Interface, CLI). Programmcode kann somit auch in jedem herkömmlichen Texteditor geschrieben werden. Mit den entsprechenden CLI-Kommandos kann dieser Code anschließend kompiliert und ausgeführt werden. Da die meisten IDEs jedoch viele nützliche Add-ons für die Quellcodeformatierung, die Programmstrukturierung und das Debugging bereitstellen, ist dies die bevorzugte Variante bei der Entwicklung des Programmcodes.

Im Zuge des vorliegenden Entwicklungsprozesses wird zunächst der Ansatz gewählt, bei dem direkt auf der Zielplattform programmiert werden soll. Aus diesem Grund wird zunächst das SDK und die Runtime Library für .NET 5, als auch die Visual Studio Code IDE auf dem RPI installiert. Bei der ersten Erstellung einer einfachen „Hello World!“ Applikation kristallisieren sich allerdings einige Problembereiche heraus. Einerseits ist es nicht möglich die C#-Erweiterung in Visual Studio Code ordnungsgemäß zu installieren bzw. auszuführen. Infolgedessen fehlt bei der Erstellung des Programmcodes jegliche Art von Unterstützung zur automatischen Vervollständigung des Quellcodes (IntelliSense), was die Programmierung ungemein erschwert. Da schließlich auch das direkte Debugging am RPI Probleme verursacht, wird ein anderer Ansatz für die Entwicklung weiterverfolgt. Bei diesem Ansatz erfolgt die Entwicklung des Programmcodes auf einem herkömmlichen Windows PC, auf dem Visual Studio 2019 installiert ist. Vorerst soll somit die gesamte Entwicklung der Emulator-Software unter Windows erfolgen. Nötige Funktionen wie IntelliSense, Debugging, Ansteuerung des seriellen Ports und Kompilierung funktionieren ohne Probleme. Einzig die GPIOs können an einem herkömmlichen PC nicht angesteuert werden. In späteren Entwicklungsstadien soll die Emulator-Software schließlich auch auf der Zielplattform ausgeführt werden. Dazu muss das Projektverzeichnis mittels Netzwerkverbindung und File Transfer Protocol (FTP) auf den RPI kopiert werden. Anschließend wird der Programmcode mittels *dotnet publish* Kommando über das CLI direkt am RPI kompiliert und mit *dotnet run* ausgeführt. Visual Studio 2019 bietet außerdem die Möglichkeit mit der Funktion „An den Prozess anhängen“ den Debugger zur Evaluierung des Programms zu starten und sich an den ausgeführten Prozess des RPIs anzuhängen. Somit stehen in Visual Studio 2019, das über das verschlüsselte Secure Shell (SSH) Netzwerkprotokoll mit dem RPI kommuniziert, alle Funktionen zum Debuggen zur Verfügung.

Aus der zuvor beschriebenen Herangehensweise lässt sich schlussfolgern, dass sich eine plattformübergreifende Entwicklung mit dem .NET 5-Framework auch in der Praxis sehr einfach und unkompliziert umsetzen lässt. Die Entwicklung des Quellcodes für den Emulator erfolgt somit größtenteils unter Windows auf einem Standard-PC und wird in weiterer Folge auf den RPI portiert und ausgeführt.

5.1.3 ASP-Webanwendung vs. Konsolen Applikation

Weitere Überlegungen führen zur Auswahl zwischen einer ASP.NET Core-Webanwendung und einer klassischen Konsolen-Applikation. Anfangs überwiegen die Vorteile der Webanwendung, weshalb erste Evaluierungen in diese Richtung durchgeführt werden. Eine Webanwendung soll die Darstellung von Buttons und Textfeldern zur Ein- und Auszahlungsemulation von Münzen, sowie eine geeignete Konfigurationsseite in Form einer grafischen Benutzerschnittstelle (Graphical User Interface, GUI)

ermöglichen. Hinzu kommt, dass der Emulator von sämtlichen Endgeräten im Netzwerk, dazu zählen auch mobile Geräte wie Smartphones oder Tablets, bedient werden könnte.

Trotz dieses Vorteils, zeigen die ersten Recherchen bezüglich einer Umsetzung mit ASP.NET Core, dass dieser Ansatz mit verhältnismäßig langen Einarbeitungszeiten verknüpft wäre. Je nach tatsächlich anzuwendender Architektur zählt dazu z.B. die Bekanntmachung mit Mustern wie dem Model-View-Controller (MVC), Razor Pages oder Blazor. Bei diesen drei Begrifflichkeiten handelt es sich um spezielle Entwurfsmuster, die auf dem Webframework ASP.NET Core basieren. Ein weiteres Problem bei der Einarbeitung stellt die Echtzeitkommunikation zwischen dem Webserver auf der Zielplattform und den Clients dar. Während der Münzeinwurf problemlos über Buttons von den Clients getriggert werden könnte, führt die Darstellung von Auszahlungen auf der Seite der Clients zu einigen Schwierigkeiten. Um Aktualisierungen, die am Server stattfinden auch auf den Clients anzuzeigen, muss in der Regel auch die Webseite des Clients neu geladen werden. Zur Behebung dieser Problematik gibt es die SignalR-Bibliothek, die Echtzeitfunktionen in ASP.NET Core bereitstellt. Dazu zählt auch die nötige Funktion zur Durchführung eines serverseitigen Broadcasts (serverseitige Übertragung einer Nachricht an alle Netzwerkteilnehmer). Informationen können damit an alle Clients gesendet werden, ohne dass die gerade dargestellten Webseiten neu geladen werden müssten. Kenntnisse in der Skriptsprache JavaScript sind für SignalR allerdings essenziell und Einarbeitungszeiten unvermeidbar.

Zur Minimierung der allgemeinen Einarbeitungszeit in neue Architekturen, Entwurfsmuster sowie Skriptsprachen, wird von der Entwicklung in ASP.NET Core vorerst abgesehen. Stattdessen soll der erste Prototyp des Emulators in einer einfachen .NET Core Konsolen-Applikation umgesetzt werden. Sollte eine modifizierte Version des Emulators zu einem späteren Zeitpunkt in Form einer Webanwendung realisiert werden, können Teile des Programmcodes wieder verwendet werden.

5.2 Basissystemkomponenten in der Softwarearchitektur

Das erarbeitete Softwarekonzept baut auf einer einfachen .NET Core Konsolen-Applikation auf. In dieser Konsolen-Applikation wird als Grundgerüst ein Event-Aggregator-Entwurfsmuster zur Kommunikation zwischen den verschiedenen Software-Modulen implementiert. Die Frage nach dem Aufbau und der Funktionsweise dieses Entwurfsmusters, sowie einigen anderen verwendete Add-ons, die für den Aufbau der Software von belangen sind, werden im Folgenden näher erläutert.

5.2.1 Das Event-Aggregator-Entwurfsmuster

Der Event Aggregator ist ein nützliches Entwurfsmuster, um zwischen verschiedenen Teilbereichen einer Software ereignisorientiert kommunizieren zu können. In einer komplexen Softwarestruktur, die aus vielen unterschiedlichen Objekten besteht, dient dieses Muster zur Entkopplung der einzelnen Bereiche, in dem es die Abhängigkeiten zueinander auf ein Minimum reduziert. Dies hat einerseits den Vorteil, dass einzelne Module innerhalb der Software eigenständig entwickelt, getestet und modifiziert werden können. Andererseits wird durch den modularen Aufbau die geforderte Wart- und Erweiterbarkeit gewährleistet.

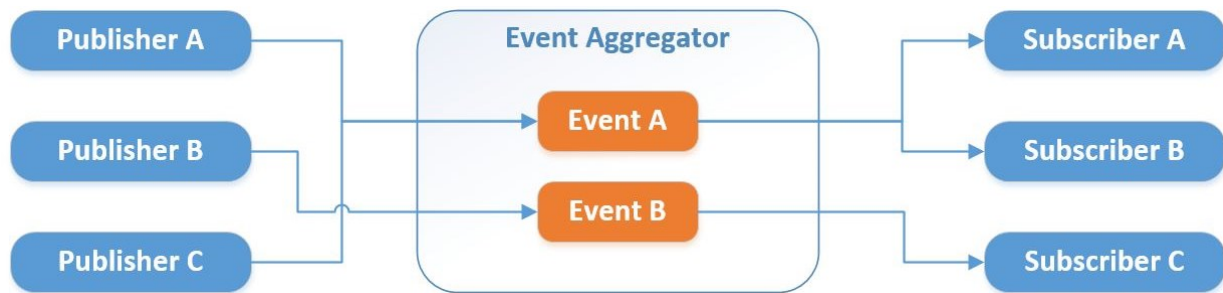


Abbildung 25: Schematische Darstellung des Event-Aggregator-Entwurfsmusters, Quelle: Eigene Darstellung.

Wie sich anhand von Abbildung 25 erkennen lässt, besteht das Muster aus den folgenden Komponenten:

- **Publisher (Herausgeber)**

Das sind Datenquellen, die Nachrichten senden oder Ereignisse auslösen.

- **Subscriber (Abonnet)**

Das sind die Empfänger der Nachrichten oder Ereignisse, die von den Publishern ausgelöst werden.

- **Event Aggregator**

Dieser agiert als zentraler Nachrichtendienst zwischen den Publishern und Subscribern

- **Events**

Diese dienen als gemeinsame Kommunikationsbasis zwischen Publishern und Subscribern.

Eine der wichtigsten Eigenschaften des vorliegenden Entwurfsmusters liegt in der losen Kopplung zwischen den Publishern und den Subscribern in einer komplexen Softwareumgebung, die aus vielen Komponenten besteht. Das bedeutet so viel wie, dass sich Publisher und Subscriber gegenseitig nicht kennen und völlig unabhängig voneinander agieren können. Beide kennen einzig und allein den Event Aggregator und das jeweilige Event, welches ausgelöst oder empfangen werden soll. Der Event Aggregator lässt hierbei auch Mehrpunktverbindungen zu. Mehrere Publisher können somit ein und dasselbe Event auslösen. Umgekehrt können mehrere Subscriber auf ein und dasselbe Event reagieren. Das grundlegende theoretische Prinzip der Kommunikation funktioniert, indem ein Publisher ein beliebiges Event an den Event Aggregator publiziert. Alle Subscriber, die sich schließlich für dieses Event beim Event Aggregator subskribiert haben, reagieren darauf. Wie dieses Entwurfsmuster in der Emulator-Software genau abgebildet ist, wird in Kapitel 5.3.3 beschrieben.

5.2.2 NuGet

NuGet ist Microsofts nützliches Add-on für moderne Entwicklungsplattformen, dass die Verteilung von quelloffenem Code erleichtert. Im Wesentlichen besteht es aus einem Softwarekomponentenverzeichnis (engl. Repository), das .NET-Komponenten, aber auch einige JavaScript-Bibliotheken für Entwickler frei zur Verfügung stellt. Der benötigte Client für die NuGet-Paketverwaltung ist bereits in der Visual Studio IDE inkludiert. Die so bezeichneten NuGet-Pakete mit der Dateinamenserweiterung *.nupkg* können somit direkt in der IDE heruntergeladen und installiert werden. Seit der Einführung von .NET Core hat NuGet enorm an

Bedeutung gewonnen, weil Microsoft dieses Tool als Hauptdistributionskanal für Komponenten des Frameworks selbst nutzt. Diese werden nun nicht mehr über das Windows-Installer-Setup der IDE installiert, sondern in Form von Paketen mit dem NuGet-Paket-Manager nur für die gerade zu erstellende Applikation heruntergeladen und installiert.⁴³

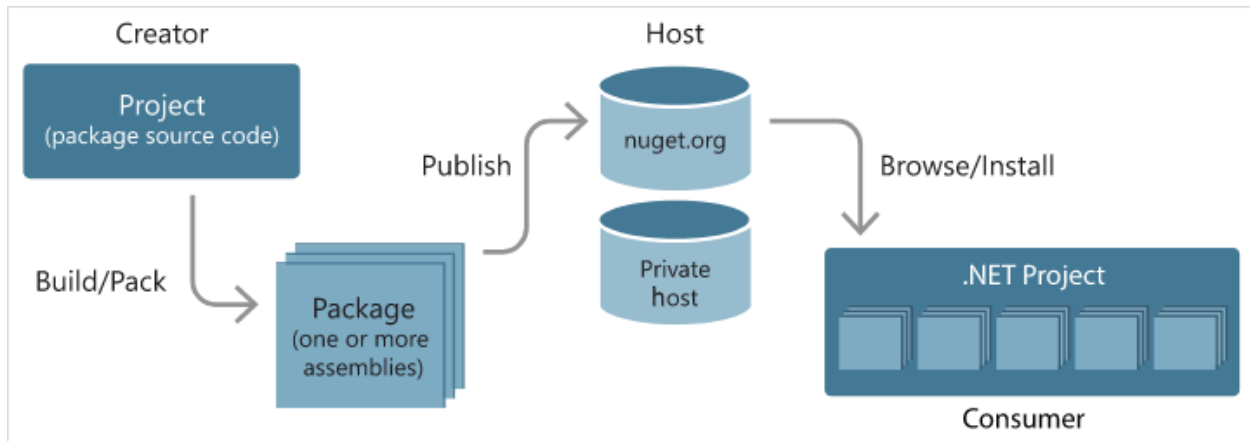


Abbildung 26: NuGet Paketfluss zwischen Ersteller, Host und Endnutzer, Quelle: Microsoft Corporation (2019), Online-Quelle [24.02.2021].

Für die Entwicklung der Emulator-Software wird NuGet beispielsweise verwendet um die entsprechenden Softwarekomponenten wie NLOG, JSON und Standardbibliotheken, wie die Serial-Port-Klasse, zu installieren.

5.2.3 JSON

Als JSON (JavaScript Object Notation) bezeichnet man ein standardisiertes, textbasiertes Datenformat. Es dient zur Darstellung von strukturierten Daten und ist, wie es der Name bereits vermuten lässt, an die Programmiersprache JavaScript angelehnt. Auch wenn die Syntax der von JavaScript ähnelt, ist JSON aber völlig unabhängig sowohl von dieser als auch von anderen Programmiersprachen. JSON ist somit ein reines Datenformat und beinhaltet nur Eigenschaften, aber keine Methoden. Dennoch wird JSON heute von vielen IDEs unterstützt und ermöglicht einen Lese- und Schreibzugriff auf die Daten.⁴⁴

In der Praxis wird JSON als Format zum Datenaustausch verwendet. Um die Daten in einem einheitlichen Format darzustellen und Datenhierarchien zu erreichen, sind in JSON die folgenden Strukturen verfügbar:⁴⁵

- **Objekte**

Objekte bestehen aus einer Menge untergeordneter Schlüssel-Wert-Paaren, die sich in einer geschwungenen Klammer befinden. Ein Schlüssel-Wert-Paar innerhalb der Klammern besteht jeweils aus Schlüssel und dazugehörigem Wert, die durch einen Doppelpunkt getrennt sind. Mehrere Schlüssel-Wert-Paare werden schließlich durch ein einfaches Komma voneinander getrennt.

⁴³ Vgl. IT-Visions (2019), Online-Quelle [24.02.2021].

⁴⁴ Vgl. Mozilla Developer Network (2020), Online-Quelle [24.02.2021].

⁴⁵ Vgl. json.org (o.J.), Online-Quelle [24.02.2021].

- **Arrays**

Arrays sind geordnete Listen von Werten, die sich innerhalb einer eckigen Klammer befinden. Die darin enthaltenen Werte werden wiederum durch ein Komma getrennt.

- **Werte**

Werte können durch Objekte, Arrays, Zeichenketten oder in booleschen Variablen repräsentiert werden, die wiederum ineinander verschachtelt sein können.

- **Zeichenketten**

Zeichenketten bestehen aus Unicode-Zeichen, die jeweils von doppelten Hochkomma-Zeichen eingeschlossen sind. Diese können auch Escape-Sequenzen mit einer besonderen Bedeutung beinhalten. Prinzipiell sind JSON-Zeichenketten denen in C oder Java aber sehr ähnlich.

In der vorliegenden Arbeit wird das JSON-Format zum Austausch der Konfigurationsdaten eingesetzt. Wie diese genaue Struktur aussieht, wird in Kapitel 5.3.4.1 beschrieben.

5.2.4 NLog

NLog ist eine quelloffene Plattform zur automatischen Protokollierung von Zuständen, Ereignissen und Fehlern in einer Software. Dies dient der Nachvollziehbarkeit und der Fehleranalyse eines Softwareprozesses. Die Bibliothek von NLog wurde gänzlich in C# entwickelt und steht somit für fast alle Teile der .NET-Plattform im NuGet-Paket-Manager zur Verfügung. Die Protokollierung mittels NLog erfolgt in Form von Logdateien, Datenbanken, Kommandozeilenfenstern oder anderen Systemen. Außerdem bietet NLog durch die vielen verschiedenen Templates für Zeitstempel, Zähler, Nachrichtenformatierung usw. eine gute Form der strukturierten Protokollierung.

Bei der Emulator-Software werden alle Polling-Sequenzen sowie die Münzauszahlungen mit Hilfe der NLog-Plattform protokolliert. Einerseits hilft dies bei der Fehlersuche und andererseits können die Logdateien beim späteren Einsatz des Emulators von den Tester*innen der Software-Qualitätssicherung für Analysezwecke verwendet werden. Der Einsatz von NLog in der Emulator-Software wird in Kapitel 5.3.9 genauer beschrieben.

5.2.5 Serial-Port-Klasse

Die Serial-Port-Klasse ist im Namespace *System.IO.Ports* von .NET enthalten, welcher auch alle anderen Klassen zum Steuern von seriellen Anschlüssen inkludiert. Bei .NET Core Applikationen muss der entsprechende Namespace zunächst über den NuGet-Paket-Manager installiert werden. Anschließend stellt die Serial-Port-Klasse das nötige Framework für die synchrone und ereignisgesteuerte Ein- und Ausgabe, den Zugriff auf alle Pin- und Unterbrechungszustände sowie alle anderen Treibereigenschaften des seriellen Anschlusses bereit.⁴⁶

⁴⁶ Microsoft Corporation (o.J.), Online-Quelle [24.02.2021].

Die Klasse enthält somit alle Eigenschaften zur Konfiguration des seriellen Anschlusses, als auch die nötigen Methoden zum Senden und Empfangen von seriellen Daten. Sie bildet deshalb die Basis für den seriellen Bustreiber des Emulators, der in Kapitel 5.3.5 erläutert wird.

5.2.6 ccTalk-Bibliothek

Die ccTalk-Bibliothek ist ein bereits existierendes API, das auf der allgemeinen ccTalk-Spezifikation in der Version 4.6 basiert. Das API wurde in C# entwickelt und steht für das .NET Framework 4.0 zur Verfügung. Es kann auf der Webseite von GitHub, welche die populärste Plattform für die Verteilung von quelloffenen Softwareprojekten darstellt, kostenlos heruntergeladen werden.

Das API ist grundsätzlich dazu gedacht, um beispielsweise mittels einer Windows Forms Applikation ccTalk-basierte Peripheriegeräte anzusteuern. Ausführbare quelloffene Projekte von anderen Entwicklern gibt es derzeit nur für die Ansteuerung von elektronischen Münzprüfern. Der Kommunikationsfluss in Bezug auf den zu entwickelnden Emulator funktioniert verständlicherweise aber genau in die andere Richtung. So muss die Emulator-Software in der Lage sein auf Abfragen zu reagieren und nicht umgekehrt, wie es bei der Verwendung des APIs der Fall ist. Daraus lässt sich schlussfolgern, dass nur Codeausschnitte des APIs für den vorliegenden Softwareentwicklungsprozess verwendet werden können. Dazu zählt z.B. die Berechnung der Checksummen sowie die Aufbereitung des ccTalk-Datenpaketes in der spezifizierten Protokoll-Struktur.

5.3 Entwicklung der Emulator-Software

Nachdem in den vorherigen Kapiteln viele Grundlagen für die Entwicklung der Anwendungssoftware des Emulators geschaffen wurden, soll der Fokus in diesem Kapitel auf die Softwareentwicklung selbst gelegt werden. Im Folgenden wird daher zuerst auf den Aufbau der Softwarearchitektur und dessen Komponenten näher eingegangen, bevor anschließend eine ausführliche Erläuterung der Implementierung der einzelnen Komponenten bzw. Module folgt.

5.3.1 Aufbau der Softwarearchitektur

Die Softwarearchitektur charakterisiert die grundlegenden Systemkomponenten der Emulator-Software und zeigt wie diese untereinander Zusammenwirken. Wie in den Anforderungen an den Emulator (siehe Kapitel 3.3.2.2) definiert, ist die Architektur der Emulator-Software modular aufgebaut, um vor allem die Aspekte der Erweiterbarkeit und Wartungsfreundlichkeit zu gewährleisten.

Die im Zuge dieser Arbeit erarbeitete Softwarearchitektur wird in Abbildung 27 veranschaulicht. Die zentrale Komponente der Architektur bildet der Event Aggregator, dessen Grundlagen bereits in Kapitel 5.2.1 näher erläutert wurden. In der vorliegenden Anwendung dient dieser zur Entkopplung des Bustreibers, Geräte-Managers und Konsolen-Managers, um untereinander auftretende Abhängigkeiten zu verhindern. Der Bustreiber agiert als serielle Treiberstufe und ist somit für die gesamte Kommunikation mit dem seriellen ccTalk-Bussystem verantwortlich. Der Geräte-Manager schafft die Grundlage für die Emulation, indem es die empfangenen Daten vom Bustreiber aufbereitet und dem zu emulierenden Gerät zuweist. Die mit dem Geräte-Manager in Verbindung stehende ccTalk-Bibliothek enthält neben der Implementierung aller

notwendigen ccTalk-Header, auch alle anderen Softwareabläufe zur Aufbereitung der Datenpakete und Berechnung der Checksummen. Der Konsolen-Manager dient zur Verarbeitung von Ein- und Ausgabeereignissen, die über das Kommandozeilenfenster von Endnutzer*innen des Emulators erfolgen. Eine weitere wichtige Komponente der Architektur stellt der Konfigurations-Manager dar. Dieser dient dazu, um die benötigten Konfigurationsdaten im JSON-Format (siehe Kapitel 5.2.3) auszulesen und schließlich in geeigneter Form aufzubereiten. Dazu zählen beispielsweise die Einstellungen für die serielle Kommunikation sowie alle Parameter der zu emulierenden Geräte.

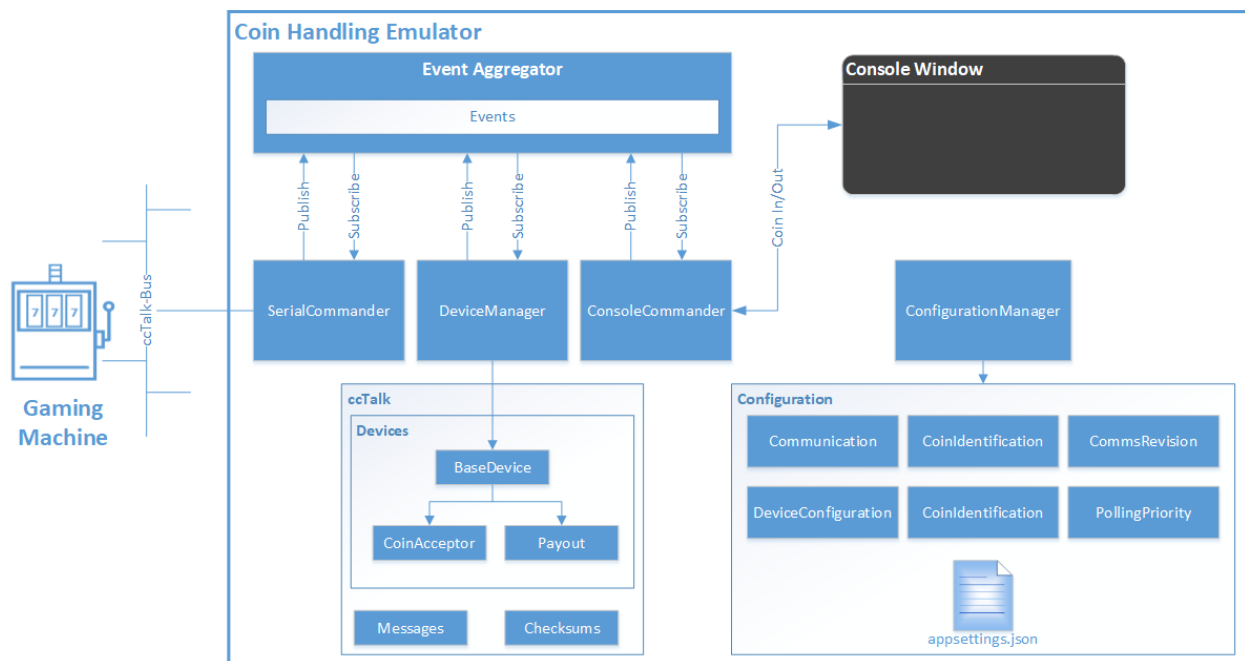


Abbildung 27: Übersicht der Architektur des Emulators, Quelle: Eigene Darstellung

Wie sich anhand des soeben beschriebenen Architekturabbilds gut erkennen lässt, wird hier genug Freiraum für zukünftige Erweiterungen durch zusätzliche Peripheriegeräte geschaffen. Neben Münzprüfer und Auszahlungseinheit können der Geräte-Manager und die ccTalk-Bibliothek einfach um weitere Geräte wie z.B. Ticketprinter oder Banknotenprüfer erweitert werden.

5.3.2 Die Startup-Klasse

Die *Startup*-Klasse stellt den Haupteinstiegspunkt der Applikation dar und beinhaltet somit auch die *Main*-Methode. In dieser wird zunächst der Geräte-Manager instanziiert und initialisiert, um die nötigen Daten aus der *appsettings.json* Konfigurationsdatei zu laden. Mit Hilfe der entsprechenden NLog-Konfigurationsdaten erfolgt anschließend die Konfiguration des Logging-Services. Bevor nun alle relevanten Module für den Emulator gestartet werden, wird über die *Mutex*-Klasse geprüft, ob bereits andere Instanzen der Applikation ausgeführt werden. Dies soll sicherstellen, dass die Applikation nicht mehrfach gestartet werden kann.

Im nächsten Schritt erfolgt nun durch den Aufruf der Methode *InitializeCoinEmulator* die Initialisierung der einzelnen Module für den Emulator. Wie im Codeausschnitt von Quelltext 1 zu erkennen ist, werden in dieser Methode dazu die entsprechenden Objekte zu den Klassen *EventAggregator*, *ConsoleCommander*, *SerialCommander* und *DeviceManager* instanziiert und zusätzlich die entsprechenden Methoden zum

Starten der Module aufgerufen. Die Klasseninstanz von der Klasse *EventAggregator* wird dabei den Instanzen aller anderen Klassen mit übergeben, so dass sich diese als Subscriber abonnieren können. Da innerhalb der Klassen *ConsoleCommander* und *SerialCommander* auch Schleifen implementiert sind, wird den Instanzen zusätzlich ein Abbruchtoken übergeben. Dieser dient in der Thread-Programmierung zum kooperativen Abbruch von asynchronen oder länger andauernden synchronen Abläufen.

```

Private static void InitializeCoinEmulator()
{
    // Hier fehlen Teile des Codes...

    try
    {
        var eventAggregator = new EventAggregator.EventAggregator();

        var consoleCommander = new ConsoleCommander(Tokensource, eventAggregator);
        consoleCommander.StartInputHandler(cancelToken);

        var serialCommander = new SerialCommander(Tokensource, eventAggregator);
        serialCommander.StartSerialCommunicationHandler(cancelToken);

        var deviceManager = new DeviceManager(eventAggregator);
        deviceManager.ConfigureDevices();

        // Wait for finish...
        cancelToken.WaitHandle.WaitOne();
    }
    catch
    {
        // Hier fehlen Teile des Codes...
    }
}

```

Quelltext 1: Codeausschnitt der Methode *InitializeCoinEmulator*, Quelle: Eigene Darstellung.

5.3.3 Event Aggregator API

Wie bereits erwähnt dient das Event-Aggregator-Entwurfsmuster in der vorliegenden Applikation zur losen Kopplung der Klassen *SerialCommander*, *DeviceManager* und *ConsoleCommander*. Wie der allgemeinen Darstellung in Abbildung 28 entnommen werden kann, agiert jede dieser Klassen hierbei entweder als Publisher, d.h. als Komponente die Events publiziert, oder als Subscriber, also als Komponente, die Events abonniert.

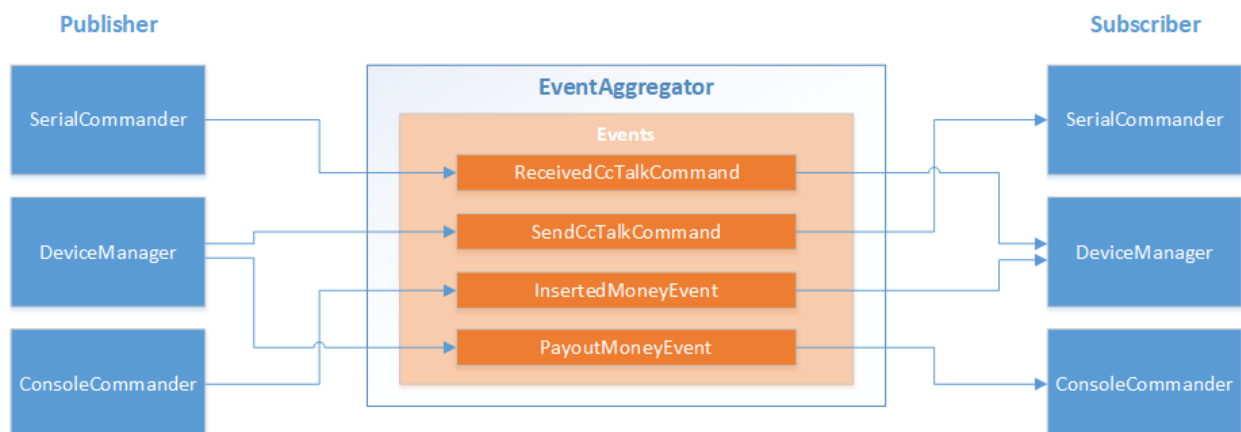


Abbildung 28: Schematischer Aufbau des Event Aggregators für die Emulator-Software, Quelle: Eigene Darstellung.

Als Basis für die Kommunikation zwischen den einzelnen Klassen, stehen vier verschiedene Events zur Verfügung, die jeweils publiziert und subskribiert werden können: *ReceivedCcTalkCommand*, *SendCcTalkCommand*, *InsertedMoneyEvent* und *PayoutMoneyEvent*.

Die Klasse *SerialCommander* publiziert beispielsweise das Event *ReceivedCcTalkCommand* sobald Daten vom Bus empfangen werden. Die Klasse *DeviceManager*, die sich für dieses Event subskribiert hat, reagiert schließlich mit entsprechender Softwareroutine auf das Event. Von der Klasse *DeviceManager* können hingegen zwei unterschiedliche Events publiziert werden. Einerseits das Event *SendCcTalkCommand*, das von der Klasse *SerialCommander* subskribiert wurde und Daten an den Bus sendet. Und andererseits das *PayoutMoneyEvent*, welches durch die Klasse *ConsoleCommander* eine Auszahlung im Kommandozeilenfenster anzeigt. Die dritte und letzte Variante tritt auf, wenn die Klasse *ConsoleCommander* das *InsertedMoneyEvent* herausgibt, welches wiederum von der Klasse *DeviceManager* abonniert ist.

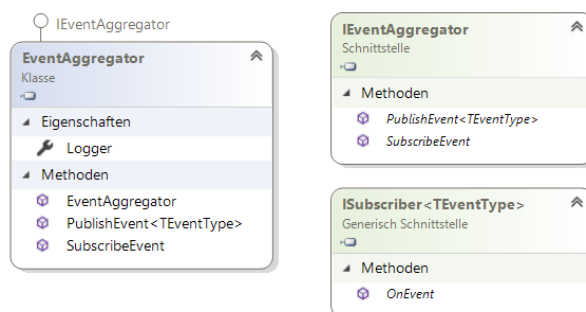


Abbildung 29: Klassendiagramm des Event Aggregators, Quelle: Eigene Darstellung.

In der Software-Implementierung besteht das Event Aggregator API aus Interfaces und einer Klasse (siehe Abbildung 29). Das Interface *IEventAggregator* dient als externe Schnittstelle und deklariert die Methoden zum Publizieren und Subsribieren der Events. In der Klasse *EventAggregator* werden diese schließlich implementiert. Die Methode *SubscribeEvent* wird verwendet, um alle Subscriber-Klassen sowie die zugehörigen Events einem Dictionary hinzuzufügen, das alle Subscriber in der Applikation registriert. Die generische Methode *PublishEvent*, die den entsprechenden Typparameter *TEventType* angibt, implementiert den Ablauf zum Publizieren der Events. Die vorhandenen Events sind als Datensätze mit dem Schlüsselwort *record* im entsprechenden Namespace deklariert (siehe Quelltext 2). Bei Datensätzen handelt es sich um Verweistypen, deren Eigenschaften automatisch vom Compiler generiert werden.

```

public record ReceivedCcTalkCommand(byte[] Payload);
public record SendCcTalkCommand(byte[] Payload);
public record InsertedMoneyEvent(byte Address, int MoneyValue);
public record PayoutMoneyEvent(byte Address, string CoinValue, byte coins);
    
```

Quelltext 2: Deklaration der Events, Quelle: Eigene Darstellung.

Eine weitere Schnittstelle stellt das generische Interface *ISubscriber* dar, das als Typparameter ebenfalls *TEventType* angibt. Dieses Interface sorgt dafür, dass die Methode *OnEvent* seitens der Subscriber

ausimplementiert wird. Wird nun von einem Publisher die Methode *PublishEvent* aufgerufen, dann wird von den jeweiligen Subscribern die Implementierung der zugehörigen Methode *OnEvent* aufgerufen.

5.3.4 Konfigurations-Manager

Der Konfigurations-Manager hat die Aufgabe beim Starten des Emulators die richtige Softwarekonfiguration aus der Konfigurationsdatei zu laden. Die Konfiguration enthält zum einen die Einstellungen für das NLog API sowie die Einstellungen für die serielle Kommunikation. Darüber hinaus sind in der Konfiguration die gesamten Parameter des Münzprüfers und der Münzauszahlseinheiten enthalten. Darunter befinden sich auch all diejenigen modellspezifischen Identifikatoren, die in den funktionalen Anforderungen definiert sind (siehe Kapitel 3.3.2.1).

5.3.4.1 JSON-Konfigurationsdatei

Das Format der Konfigurationsdatei entspricht dem JSON-Format, dass bereits in Kapitel 5.2.3 kurz vorgestellt wurde. Das Format folgt somit einem strukturierten Aufbau der Daten und kann wie gefordert (siehe Kapitel 3.3.2.2) relativ einfach geändert werden. Folgende Einstellungen und Parameter sind schließlich in der *appsettings.json* Datei enthalten:

- **Logging- und NLog-Settings**

Die Logging-Settings enthalten zunächst den Parameter für den *LogLevel*, der den Grad der Protokollierung angibt. Für Entwicklungszwecke ist dieser mit *Debug* initialisiert. Weiters befinden sich alle NLog relevanten Einstellungen in der Konfiguration. Dazu gehören z.B. der Pfad für die Logdatei, die Limitierung von Lognachrichten innerhalb eines definierten Zeitintervalls oder farbliche Hervorhebungen von Nachrichten bestimmter *LogLevel* im Kommandozeilenfenster.

- **Communication Settings**

In den Communication Settings befinden sich die Einstellungen, die für die Konfiguration der seriellen Schnittstelle von Bedeutung sind und leicht veränderbar sein sollten. Dazu zählt einerseits die standardmäßige Bezeichnung der seriellen Schnittstelle auf der jeweiligen Zielplattform sowie die Baudrate für die Kommunikation.

```
"CommunicationSettings":{  
  "SerialPortWindows": "Com2",  
  "SerialPortLinux": "/dev/ttyS0",  
  "BaudRate": 9600  
},
```

Quelltext 3: CommunicationSettings-Objekt in der *appsettings.json* Datei, Quelle: Eigene Darstellung.

- **Device Configurations**

Die Device Configurations beinhalten den Teil der Konfigurationsdatei, der am öftesten geändert werden muss. Er definiert die komplette Konfiguration für die zu emulierenden Peripheriegeräte des Münzverarbeitungssystems. Den Haupt-Identifikator stellt die Adresse dar, weil die zu emulierenden Geräte anhand dieser mit dem ccTalk-Protokoll angesprochen werden. Der nachfolgende Quelltext 4 zeigt einen Ausschnitt aus der *appsettings.json* Datei. Auf der linken Seite

wird die Konfiguration eines Münzprüfers mit der Adresse 2 und auf der rechten Seite die Konfiguration von zwei Münzauszahleinheiten mit den Adressen 3 und 4 dargestellt. Unterhalb der Adressen befinden sich die entsprechenden Schlüsse-Wert-Paare für die modellspezifischen Parameter der einzelnen Geräte. Je nach Modell von Münzprüfer und Münzauszahleinheit (siehe Tabelle 2 und Tabelle 3), welches von der Emulator-Software emuliert werden soll, müssen die einzelnen Parameter in der Konfigurationsdatei entsprechend angepasst werden.

```

"Address": "2",
"EquipmentCategoryId": "Coin Acceptor",
"ManufacturerId": "CMG",
"ProductCode": "RM5HDC",
"SoftwareRevision": "10/03/2011",
"SerialNumber": "67093",
"BuildCode": "RM5 HD 4.0",
"CoinChannels": 16,
"MultipleSorterPaths": false,
"PollingPriority": {
  "unit": "x10ms",
  "value": "50"
},
"CommsRevision": {
  "Release": "1",
  "MajorRevision": "4",
  "MinorRevision": "6"
},
"CoinConfigurations": [
  {
    "CoinPosition": "1",
    "CoinIdentification": {
      "CountryCode": "EU",
      "Value": "200",
      "MintCode": "A"
    }
  }
],
...

```

```

"Address": "3",
"EquipmentCategoryId": "Payout",
"ManufacturerId": "Azkoyen",
"ProductCode": "SCH2",
"SoftwareRevision": "72",
"SerialNumber": "0",
"BuildCode": "Payout",
"CipherKey": [0, 0, 0, 0, 0, 0, 0, 0],
"MoneyControlsVersion": "SCH2",
"PayoutCoinIdentification": "EU200A",
"CommsRevision": {
  "Release": "1",
  "MajorRevision": "4",
  "MinorRevision": "6"
},
"Address": "4",
"EquipmentCategoryId": "Payout",
"ManufacturerId": "Azkoyen",
"ProductCode": "SCH2",
"SoftwareRevision": "72",
"SerialNumber": "0",
"BuildCode": "Payout",
"CipherKey": [0, 0, 0, 0, 0, 0, 0, 0],
"MoneyControlsVersion": "SCH2",
"PayoutCoinIdentification": "EU200A",
"CommsRevision": {
  "Release": "1",
  "MajorRevision": "4",
  "MinorRevision": "6"
},

```

Quelltext 4: *DeviceConfigurations* in der *appsettings.json* Datei, Quelle: Eigene Darstellung.

5.3.4.2 Laden der Systemkonfiguration

Der relevante Teil der Emulator-Software, der für das Laden der Systemkonfiguration verantwortlich ist, besteht aus der Klasse *ConfigurationManager* und dem zugehörigen Interface *IConfigurationManager* (siehe Abbildung 30).

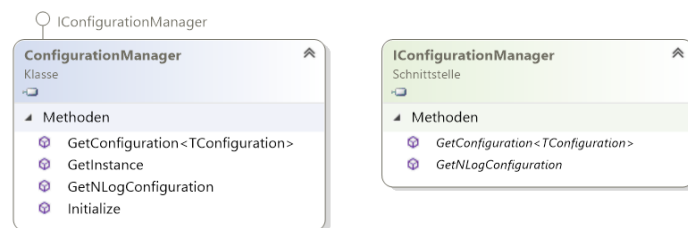


Abbildung 30: Klassendiagramm des Konfigurations-Managers, Quelle: Eigene Darstellung.

Das Interface deklariert die Methoden *GetConfiguration* und *GetNLogConfiguration*, welche wiederum in der zugehörigen Klasse ausimplementiert sind. Wird die Klasse *ConfigurationManager* instanziiert und die Methode *Initialize* aufgerufen, dann werden zunächst alle Schlüssel-Wert-Paare aus der *appsettings.json* Datei gelesen. Diese werden in NLog-Konfiguration und Gerätekonfiguration aufgesplittet. Hierzu wird ein

von .NET verfügbares API zur Konfigurationsunterstützung eingesetzt. Die Methode *GetInstance* wird anschließend dazu verwendet, um in anderen Klassen auf die entsprechende Klasseninstanz des Konfigurationsmanagers zu verweisen.

Während die Methode *GetNLogConfiguration* ausschließlich zum Laden der Logging-Settings für das NLog API benötigt wird, dient die Methode *GetConfiguration*, die den Typparameter *TConfiguration* angibt, dazu, um die entsprechende Sektion der Gerätekonfiguration zurückzugeben. Diese besteht aus der Device Configuration und den Communication Settings. Die Parameter der Gerätekonfiguration werden somit in den anderen Teilbereichen der Software, wie dem Bustreiber, Geräte-Manager oder Konsolen-Manager, nutzbar gemacht.

Device Configuration

Die Device Configuration wird in den Klassen *DeviceManager* und *ConsoleCommander* benötigt. Die Basis für die Device Configuration in den Klassen *DeviceManager* und *ConsoleCommander* bildet eine generische Liste, die als Typparameter die Klasse *DeviceConfiguration* angibt (siehe Quelltext 5).

```
public class DeviceConfiguration
{
    public byte Address { get; set; }
    public string EquipmentCategoryId { get; set; }
    public string ManufacturerId { get; set; }
    public string ProductCode { get; set; }
    public string SoftwareRevision { get; set; }
    public string SerialNumber { get; set; }
    public string BuildCode { get; set; }
    public string PayoutCoinIdentification { get; set; }
    public string MoneyControlsVersion { get; set; }
    public int CoinChannels { get; set; }
    public bool MultipleSorterPaths { get; set; }
    public byte[] CipherKey { get; set; }
    public CoinConfigurations CoinConfigurations { get; set; }
    public CommsRevision CommsRevision { get; set; }
    public PollingPriority PollingPriority { get; set; }
}
```

Quelltext 5: Implementierung der Klasse *DeviceConfiguration*, Quelle: Eigen Darstellung.

Mit dem Ausdruck in Quelltext 6 wird das Objekt *deviceSettings* erzeugt und mit den entsprechenden Schlüssel-Wert-Paaren aus der Konfiguration initialisiert.

```
var deviceSettings =
    ConfigurationManager.GetInstance().GetConfiguration<DeviceConfigurations>();
```

Quelltext 6: Ausdruck zum Erzeugen des Objekts *deviceSettings*, Quelle: Eigene Darstellung.

Die zur Laufzeit geladene Konfiguration wird schließlich in Abbildung 31 auf der nächsten Seite dieser Arbeit veranschaulicht.

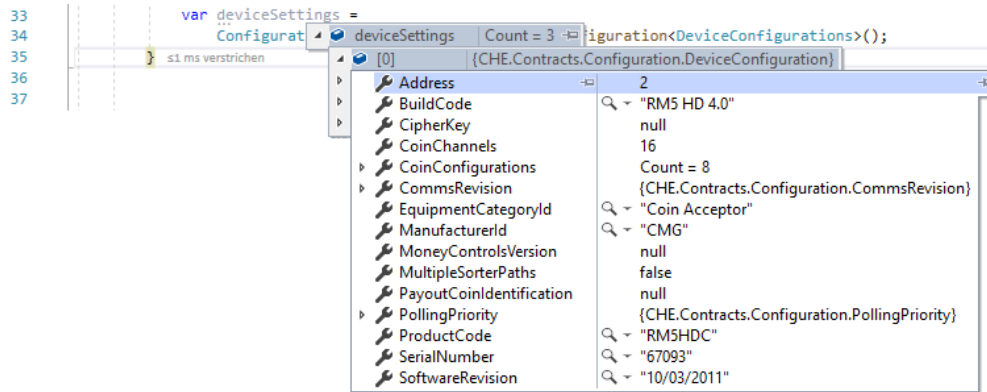


Abbildung 31: Gerätekonfiguration zur Laufzeit, Quelle: Eigene Darstellung.

Zur Aufbereitung einiger Daten aus der Konfiguration werden zusätzliche Helferklassen eingesetzt, welchen hier aber keine weitere Beachtung geschenkt wird. Diese Helferklassen werden beispielsweise verwendet, um Parameter aus der Konfigurationsdatei, wie die `ccTalk`-Revision oder die Münz-Konfiguration in Form von Listen, darzustellen.

Communication Settings

Die Einstellungen für die serielle Kommunikation werden ausschließlich von der Klasse `SerialCommander` zur Initialisierung der seriellen Schnittstelle verwendet. Die Basis für die Konfiguration bildet die Klasse `CommunicationSettings` (siehe Quelltext 7).

```
public class CommunicationSettings
{
    public string SerialPortWindows { get; set; }
    public string SerialPortLinux { get; set; }
    public int BaudRate { get; set; }
}
```

Quelltext 7: Implementierung der Klasse `CommunicationSettings`, Quelle: Eigene Darstellung.

Die Klasseninstanz wird innerhalb des Konstruktors der Klasse `SerialCommander` erzeugt und deren Eigenschaften anschließend mit den Werten aus der Konfiguration initialisiert (siehe Quelltext 8).

```
var communicationSettings =
    ConfigurationManager.GetInstance().GetConfiguration<CommunicationSettings>();
```

Quelltext 8: Ausdruck zum Erzeugen des Objekts `communicationSettings`, Quelle: Eigene Darstellung.

In Abbildung 32 werden die Eigenschaften der instanziierten Klasse zur Laufzeit dargestellt. In weiterer Folge werden die geladenen Parameter zur Konfiguration der Serial-Port-Klasse verwendet.

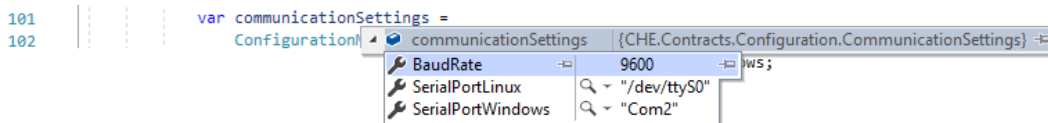


Abbildung 32: Serielle Schnittstellenkonfiguration zur Laufzeit, Quelle: Eigene Darstellung.

5.3.5 Serieller Bustreiber

Die Klasse *SerialCommander* bildet die Basis für die Anbindung an den ccTalk-Bus und übernimmt somit die komplette serielle Kommunikation. Zu den Hauptfunktionen zählt daher das Empfangen von Daten des Masters im Bussystem, sowie das Senden der Antwortdaten auf eine Anfrage zurück zum Master. Hierbei ist es insbesondere wichtig ein möglich auftretendes Echo von gesendeten Daten, welches auf Grund der zugrundeliegenden Hardware-Topologie auftritt, herauszufiltern.

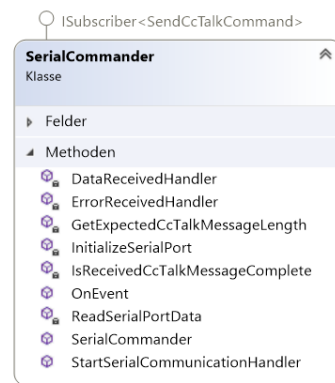


Abbildung 33: Klassendiagramm des seriellen Bustreibers, Quelle: Eigene Darstellung.

Der prinzipielle Aufbau der Klasse *SerialCommander* lässt sich anhand des Klassendiagramms in Abbildung 33 erkennen. Mit dem Aufruf der Methode *StartSerialCommunicationHandler* in der *Startup*-Klasse (siehe Kapitel 5.3.2) wird zunächst ein Objekt vom Typ *SerialPort* erstellt und mit der Methode *InitializeSerialPort* initialisiert. Anschließend werden für die Ereignisse *DataReceived* und *ErrorReceived* aus der *Serial-Port*-Klasse die gleichbenannten Ereignishandler *DataReceivedHandler* und *ErrorReceivedHandler* gestartet, um auf empfangene Daten und auftretende Fehler zu reagieren. Die Auslösung des jeweiligen Ereignisses erfolgt automatisch vom Betriebssystem, wenn Daten über den seriellen Port empfangen werden. Wird nun beispielsweise das Ereignis *DataReceived* durch ankommende Daten ausgelöst, erfolgt das Einlesen der Daten durch den Aufruf der Methode *ReadSerialPortData*. Die Methoden *IsReceivedCcTalkMessageComplete* und *GetExpectedCcTalkMessageLength* sind einfache Helfermethoden, um festzustellen, ob das eingelesene Datenpaket vollständig ist.

Wie sich anhand des Klassendiagramms noch erkennen lässt, subscribiert die Klasse *SerialCommander* das Event *SendCcTalkCommand* vom Event Aggregator. Wird dieses Event nun von einem der Publisher publiziert, dann wird die Methode *OnEvent* aufgerufen. *OnEvent* von der Klasse *SerialCommander* wird z.B. von den ccTalk-Device-Klassen publiziert, um auf Polling-Anfragen vom Master zu antworten. Die mit der Event-Methode übergebenen Daten werden schließlich an den Bus gesendet.

5.3.5.1 ReadSerialPortData-Methode

Mit der Methode *ReadSerialPortData* werden die empfangenen Daten des *DataReceived* Ereignisses eingelesen und in weiterer Folge an den Event Aggregator publiziert. Wie sich anhand von Quelltext 9 erkennen lässt, wird dazu in der Methodendefinition zunächst eine generische Liste *receiveBuffer* erzeugt, die den Type *byte* angibt und als Empfangsbuffer agiert. Die Liste wird unmittelbar mit dem ersten empfangene Byte initialisiert. Der nachfolgende Bytestream wird anschließend Byte für Byte in einer

Schleife eingelesen. Nach jedem eingelesenen Byte wird über die *IsReceivedCcTalkMessageComplete*-Methode geprüft, ob das empfangene ccTalk-Datenpaket vollständig ist. Sobald alle Bytes eingelesen wurden und das Datenpaket vollständig ist, wird der Inhalt vom receiveBuffer der implizit deklarierten Variable *receivedMessage* zugewiesen. Anschließend wird verifiziert, ob es sich beim empfangenen Datenpaket um das Echo eines gerade gesendeten Datenpaketes des Emulators handelt. Die Vorgehensweise wird in Unterkapitel 5.3.5.3 noch im Detail beschrieben. Handelt es sich um kein Echo, dann wird das Event *ReceivedCcTalkCommand* vom *eventAggregator* Objekt publiziert. Dem Event wird dabei als Parameter das empfangene Datenpaket *receivedMessage* übergeben. Auf das Event wird schließlich vom Subscriber *DeviceManager*, welcher in Kapitel 5.3.6 beschrieben wird, reagiert.

```

var receiveBuffer = new List<byte>
{
    Convert.ToByte(connection.ReadByte())
};

while (true)
{
    try
    {
        receiveBuffer.Add(Convert.ToByte(connection.ReadByte()));
        if(IsCcTalkRespondComplete(receiveBuffer))
        {
            var receivedMessage = receiveBuffer.ToArray();
            lock (LockSerialSender)
            {
                bool messageRemoved = false;
                for(int echoMessageIndex = echoMessages.Count -1;
                    echoMessageIndex >= 0; echoMessageIndex--)
                {
                    if(echoMessages[echoMessageIndex].SequenceEqual(receivedMessage))
                    {
                        echoMessages.RemoveAt(echoMessageIndex);
                        receiveBuffer.Clear();
                        messageRemoved = true;
                        break;
                    }
                }
                if(messageRemoved)
                {
                    continue;
                }
            }
            eventAggregator.PublishEvent(new ReceivedCcTalkCommand(receivedMessage));
            break;
        }
    }
    catch (TimeoutException ex)
    {
        // Hier fehlen Teile des Codes...
    }
}

```

Quelltext 9: Codeausschnitt der Methode *ReadSerialPortData*, Quelle: Eigene Darstellung.

5.3.5.2 OnEvent-Methode (SendCcTalkCommand)

Die Methode *OnEvent* dient im Kontext der Klasse *SerialCommander* dazu, um bei Aufruf Daten über die serielle Schnittstelle an das ccTalk-Bussystem zu senden. Da die Klasse *SerialCommander* das Event *SendCcTalkCommand* vom Event Aggregator subskribiert hat, wird die Methode *OnEvent* aufgerufen, sobald das Event vom Publisher publiziert wird. Publisher ist in diesem Fall die Klasse *DeviceManager* bzw. die ccTalk-Bibliothek, welche die Antworten auf die Polling-Sequenzen vom Master senden möchte.

Wie in Quelltext 10 zu erkennen ist, wird der *OnEvent*-Methode als Parameter der Datensatz des Events *SendCcTalkCommand* angegeben. Hiermit werden bei Methodenaufruf zunächst auch die zu übertragenden Rohdaten übergeben. Anschließend wird mit der *lock*-Anweisung das serielle Sender-Objekt für andere Threads gesperrt. Nach Überprüfung darauf, ob die serielle Anschlussverbindung geöffnet ist, werden die zu übertragenden Daten für die Filterung des Echos zwischengespeichert. Danach folgt die eigentlich Übertragung der Daten und das Leeren des Ausgabepuffers.

```
public void OnEvent(SendCcTalkCommand receivedRawData)
{
    lock (LockSerialSender)
    {
        if (!connection.IsOpen)
            connection.Open();
        logger?.Debug("Sending ccTalk message..");
        echoMessages.Add(receivedRawData.Payload);
        connection.Write(receivedRawData.Payload, 0, receivedRawData.Payload.Length);
        connection.DiscardOutBuffer();
    }
}
```

Quelltext 10: Implementierung der Methode *OnEvent(SendCcTalkCommand)*, Quelle: Eigene Darstellung.

5.3.5.3 Echo-Filterung

Wie bereits in den vorherigen Kapiteln erwähnt, macht die Hardware-Topologie der Busanbindung es notwendig, die empfangenen Daten vom Bus auf ein gegebenenfalls auftretendes Echo zu überprüfen. Das Echo tritt schaltungsbedingt nach der Ausgabe von Daten über die serielle Schnittstelle unmittelbar am Eingang auf.

Zur Filterung dieses Echos werden vor dem Senden alle Datenpakete in der Liste *echoMessages* gespeichert (siehe Quelltext 10). Werden anschließend Daten vom Bus über die Methode *ReadSerialPortData* empfangen, dann wird in einer Schleife geprüft, ob das empfangene Datenpaket in der Liste enthalten ist (siehe Quelltext 9). Ist das Datenpaket darin enthalten, handelt es sich tatsächlich um ein Echo. Somit werden die Daten aus der Liste entfernt und die Ausführung der Schleife mit der *break*-Anweisung unterbrochen. Mit der nachfolgenden *if*-Anweisung wird erkannt, dass es sich bei den soeben empfangen Daten um ein Echo handelte. Die *continue*-Anweisung lässt den Befehlszähler (engl. Program Counter) in die nächste Iteration springen. Die Daten werden somit nicht vom Event Aggregator publiziert.

5.3.6 Geräte-Manager

Die Klasse *DeviceManager* übernimmt die Zuteilung und Verarbeitung der empfangenen Daten über den ccTalk-Bus sowie Eingaben über das Kommandozeilenfenster an die zu emulierenden Peripheriegeräte, die in der Konfiguration hinterlegt sind. Wie in Abbildung 34 zu sehen ist, agiert die Klasse somit als Subscriber der Events *ReceivedCcTalkCommand* und *InsertedMoneyEvent*.

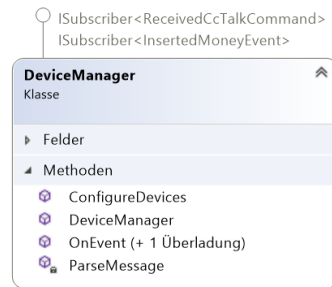


Abbildung 34: Klassendiagramm des Geräte-Managers, Quelle: Eigene Darstellung.

Der Aufbau der Klasse beinhaltet fürs erste den parametrisierten Konstruktor *DeviceManager*, in dem die Klasse beim Event Aggregator als Subscriber registriert wird. Dieser wird automatisch beim Erstellen der Klasseninstanz in der *Startup*-Klasse aufgerufen. Die Methode *ConfigureDevices*, deren Aufruf ebenfalls in der *Startup*-Klasse erfolgt, dient zum Laden und Erstellen der Gerätekonfiguration. Da die Klasse *DeviceManager* als Subscriber von mehreren Events registriert ist, ist die Methode *OnEvent* überladen. Um die empfangen Daten schließlich in das einheitliche ccTalk-Format zu konvertieren, wird die Hilfsmethode *ParseMessage* eingesetzt. In den folgenden Unterkapiteln wird die Konfiguration der Geräte sowie die überladene *OnEvent* Methode näher beschrieben.

5.3.6.1 Konfigurieren der Geräte

Für die Zuteilung der empfangen Datenpakete zu den jeweiligen Peripheriegeräten wird jeder Eintrag der Gerätekonfiguration einer entsprechenden Equipment-Kategorie zugeordnet. Dies geschieht durch die Implementierung des folgenden Codes in Quelltext 11.

```

public void ConfigureDevices()
{
    var deviceSettings =
        ConfigurationManager.GetInstance().GetConfiguration<DeviceConfigurations>();

    foreach (var setting in deviceSettings)
    {
        switch (setting.EquipmentCategoryId)
        {
            case "Coin Acceptor":
                coinAcceptorDevices.Add(new CoinAcceptorDevice(setting, eventAggregator));
                break;
            case "Payout":
                payoutDevices.Add(new PayoutDevice(setting, eventAggregator));
                break;
            default:
                logger?.Warn($"The device type <{setting.EquipmentCategoryId}> is not supported.");
                break;
        }
    }
}
  
```

Quelltext 11: Implementierung der Methode *ConfigureDevices*, Quelle: Eigene Darstellung.

Zunächst wird die Instanz der Gerätekonfiguration, die bereits mit dem Geräte-Manager (siehe Kapitel 5.3.4) erstellt und geladen wurde, dem implizit deklarierten Typ *deviceSettings* zugewiesen. Im Anschluss wird mit der *foreach*-Anweisung jedes Element in *deviceSettings*, abhängig von dessen Equipment-Kategorie, Einträgen der zuvor erstellten Listen *coinAcceptorDevices* oder *payOutDevices* zugewiesen. Die Listen entsprechen den entsprechenden Typen aus der ccTalk-Bibliothek (*CoinAcceptorDevice* oder *PayoutDevice*), wobei jeder Listeneintrag ein Objekt der jeweiligen Klasse darstellt. Die Objekte werden beim Hinzufügen der Listeneinträge erzeugt und mit den jeweiligen Elementen aus *deviceSettings* in deren Konstruktor initialisiert. Näheres dazu findet sich in Kapitel 5.3.7

5.3.6.2 OnEvent-Methode (ReceivedCcTalkCommand)

Die *OnEvent*-Methode, welche bei Auftreten des Events *ReceivedCcTalkCommand* vom Subscriber *DeviceManager* aufgerufen wird, übergibt diesem zunächst die empfangenen Rohdaten vom *SerialCommander*. Danach wird mit einer Helferklasse zur Berechnung der Checksumme aus der ccTalk-Bibliothek die Korrektheit der Daten geprüft. Sind die empfangenen Rohdaten korrekt, dann werden diese mit der Methode *ParseMessage* in das ccTalk-Format übergeführt. Zu guter Letzt erfolgt mit einer *if-else*-Anweisung die Zuordnung der empfangenen ccTalk-Nachricht zum jeweiligen emulierten Gerät. Zuerst wird abgefragt, ob es sich um eine Broadcast-Nachricht vom Master handelt (siehe Kapitel 3.2.3.1). Trifft dies zu, dann wird für jedes Objekt von Münzprüfer oder Münzauszahlungseinheit die Methode *HandleCcTalkMessage* aufgerufen und als Parameter, die empfangene ccTalk-Nachricht übergeben. Wie der Ablauf innerhalb dieser aufgerufenen Methode aussieht, wird in Kapitel 5.3.7 beschrieben.

```
public void OnEvent(ReceivedCcTalkCommand receivedRawData)
{
    lock (LockDeviceManagerLock)
    {
        logger?.Debug($"Bus input message received with <{receivedRawData.Payload.Length}> bytes.");

        // Checksum verification of the received data
        var chHandler = new Checksum();

        if (!chHandler.Check(receivedRawData.Payload, 0, receivedRawData.Payload.Length))
        {
            throw new NotSupportedException("Checksum check of received ccTalk message failed");
        }

        // Parse received message into the ccTalk message format
        var receivedMessage = ParseMessage(receivedRawData.Payload, receivedRawData.Payload.Length);

        // Assignment of the received message to the appropriate ccTalk device API class object
        if (receivedMessage.DestinationAddress.IsBroadcastMessage())
        {
            coinAcceptorDevices.ForEach(device => device.HandleCcTalkMessage(receivedMessage));
            payoutDevices.ForEach(device => device.HandleCcTalkMessage(receivedMessage));
        }
        else
        {
            coinAcceptorDevices.SingleOrDefault(device =>
                device.Address == receivedMessage.DestinationAddress)?
                .HandleCcTalkMessage(receivedMessage);
            payoutDevices.SingleOrDefault(device =>
                device.Address == receivedMessage.DestinationAddress)?
                .HandleCcTalkMessage(receivedMessage);
        }
    }
}
```

Quelltext 12: Implementierung der Methode *OnEvent(ReceivedCcTalkCommand)*, Quelle: Eigene Darstellung.

5.3.6.3 OnEvent-Methode (InsertedMoneyEvent)

Eine weitere Ausprägung der überladenen *OnEvent*-Methode wird seitens Subscriber aufgerufen, sobald von der Klasse *ConsoleCommander* ein Münzeinwurf-Ereignis über die Tastatureingabe im Kommandozeilenfenster erkannt wird. Der Methode werden dabei die eingegeben Rohdaten übergeben. Innerhalb der Methodendefinition wird vom jeweiligen Objekt des emulierten Münzprüfers die Methode *MoneyInserted* aufgerufen, wenn die Adresse der empfangenen ccTalk-Nachricht mit der Adresse des Münzprüfer-Objekts übereinstimmt. Der Methode werden schließlich die eingegeben Daten übergeben, um diese anschließend in den Ereignisdatenspeicher des emulierten Münzprüfers zu schreiben.

```
public void OnEvent(InsertedMoneyEvent receivedRawData)
{
    coinAcceptorDevices.SingleOrDefault(device => device.Address == receivedRawData.Address)?
        .MoneyInserted(receivedRawData);
}
```

Quelltext 13: Implementierung der Methode *OnEvent(InsertedMoneyEvent)*, Quelle: Eigene Darstellung.

5.3.7 ccTalk-Bibliothek

5.3.7.1 Aufbau der Bibliothek

Die ccTalk-Bibliothek schafft die Grundlage für die Emulation von Komponenten des automatisierten Münzverarbeitungssystems. Der grundlegende Aufbau der Bibliothek wird in Abbildung 35 veranschaulicht.

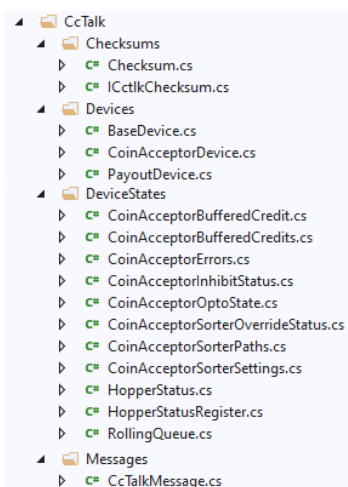


Abbildung 35: Projektmappen-Verzeichnis der ccTalk-Bibliothek, Quelle: Eigene Darstellung.

Einige Teile dieser Bibliothek stammen von einem bestehenden API aus dem Internet (siehe Kapitel 5.2.6). Dazu zählt beispielsweise die Klasse *CcTalkMessage*, worin der gesamte Aufbau des Datenpaketes aus der Spezifikation definiert ist, sowie das Interface und die Klasse im Verzeichnis *Checksums*, die für die Berechnung der einfachen Checksummen zuständig sind.

Die Klassen in den anderen Verzeichnissen sind hingegen unabhängig von bestehenden APIs. Im Verzeichnis *DeviceStates* befinden sich alle Klassen, die notwendig sind, um die entsprechende Statusregister, Einstellungen und den Ereignisdatenspeicher für Münzprüfer und Münzauszahlungen programmtechnisch nachzubilden. Darüber hinaus enthält das Verzeichnis *Devices* die Klassen für die zu emulierenden Geräten.

5.3.7.2 Emulation der Peripheriegeräte

Der Teil aus der ccTalk-Bibliothek, der für die Emulation der Peripheriegeräte und die Zuweisung der einzelnen Polling-Sequenzen zuständig ist, besteht aus den drei Klassen, die in Abbildung 36 dargestellt werden: *BaseDevice*, *CoinAcceptorDevice* und *PayoutDevice*.

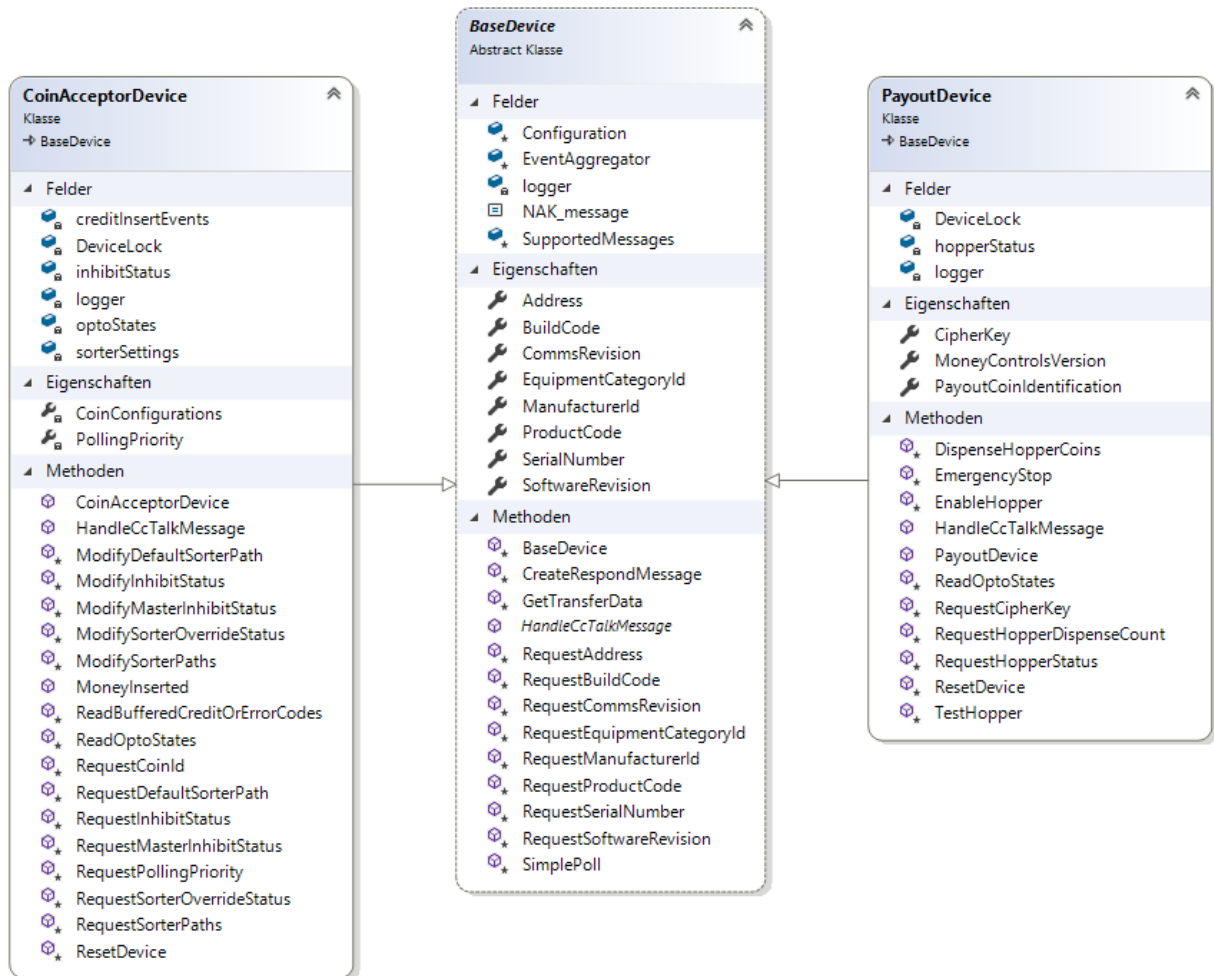


Abbildung 36: Klassendiagramm der ccTalk-Device-Klassen, Quelle: Eigene Darstellung.

Bei der Klasse *BaseDevice* handelt es sich um eine abstrakte Klasse, welche die Basisklasse für die abgeleiteten Device-Klassen *CoinAcceptorDevice* und *PayoutDevice* darstellt. Die Basisklasse implementiert hierbei die Routinen für alle generischen ccTalk-Header, die von jedem Peripheriegerät, ungeachtet dessen, um welche Art von Gerät es sich handelt, unterstützt werden müssen. Die gerätespezifischen ccTalk-Header sind hingegen in den abgeleiteten Klassen des jeweiligen Gerätes implementiert. Diese Struktur soll eine einfache Erweiterung um zusätzliche Geräte ermöglichen, wie es in den Anforderungen (siehe Kapitel 3.3.2.2) definiert ist.

Alle drei Klassen bestehen aus Feldern, Eigenschaften und Methoden. Während die Felder der abgeleiteten Klassen vorwiegend Objekte von Statusregistern oder Ähnlichem darstellen, werden den Feldern *Configuration* und *EventAggregator* in der Basisklasse die Klasseninstanzen des Event Aggregators und der Gerätekonfiguration im Konstruktor zugewiesen. Zusätzlich wird in der Basisklasse das Dictionary *SupportedMessages* deklariert und im Konstruktor mit den unterstützten ccTalk-Headern

und zugehörigen Methoden initialisiert. In den Konstruktoren der abgeleiteten Klassen werden zusätzliche Elemente für die gerätespezifischen Header hinzugefügt. Das Dictionary dient bei ankommenden Polls zur Abfrage, ob der gerade empfangene ccTalk-Header vom Emulator unterstützt wird oder nicht. Wie in Abbildung 37 dargestellt, dient als Schlüssel dem Dictionary der Header und als Wert sind die zugehörigen Methodenaufrufe zur Aufbereitung des Antwortpaketes hinterlegt. Weiters werden die Eigenschaften aller drei Klassen bei der Instanziierung für jedes Objekt mit der übergebene Instanz der Gerätekonfiguration initialisiert.

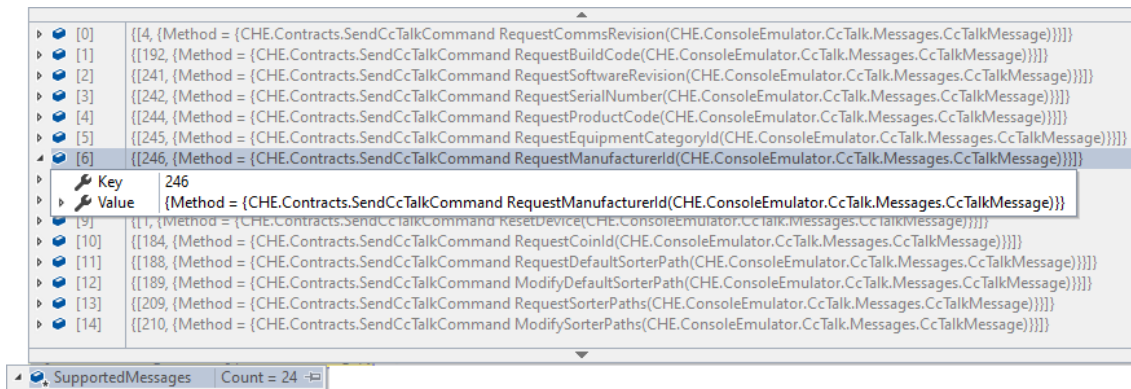


Abbildung 37: Befülltes Dictionary *SupportedMessages* zur Laufzeit, Quelle: Eigene Darstellung.

5.3.7.3 Verarbeitung von ankommenden Polls

Die Methode *HandleCcTalkMessage*, die bei empfangenen Daten vom Geräte-Manager aufgerufen wird (siehe Quelltext 12), ist in der Basisklasse *BaseDevice* lediglich als abstrakte Methode deklariert. Die Implementierung dieser Methode wird somit von den abgeleiteten Device-Klassen bereitgestellt und in Quelltext 14 veranschaulicht.

```
public override void HandleCcTalkMessage(CcTalkMessage receivedMessage)
{
    // Verifies if the header of the received message is supported by the assigned device emulator
    if (SupportedMessages.ContainsKey(receivedMessage.Header))
    {
        // Prepares the respond for the received ccTalk message poll
        logger?.Debug($"Prepare respond for ccTalk Message with header <{receivedMessage.Header}>...");
        var sendCommand = SupportedMessages[receivedMessage.Header](receivedMessage);

        // Publishes the respond for the received ccTalk message poll
        logger?.Debug($"Publish respond message with <{sendCommand.Payload.Length}> bytes of data.");
        EventAggregator.PublishEvent(sendCommand);
    }
    else
    {
        logger?.Info($"ccTalk Message request for header {receivedMessage.Header} is not supported.");
    }
}
```

Quelltext 14: Implementierung der Methode *HandleCcTalkMessage*, Quelle: Eigene Darstellung.

Als Parameter wird der Methode *HandleCcTalkMessage* das empfangene und aufbereitete Datenpaket übergeben. Anhand des Headers in diesem Datenpaket wird verifiziert, ob dieser im Dictionary *SupportedMessages* enthalten ist und somit vom Emulator unterstützt wird. Nachfolgend wird über diesen Header im Dictionary die zugehörige Methode zur Aufbereitung des Antwortpaketes aufgerufen und der Variable *sendCommand* zugewiesen. Alle diese Methoden sind als Verweistyp *SendCcTalkCommand*

deklariert und übergeben als Rückgabewert direkt das Antwortpaket für den jeweiligen Header. Darüber hinaus sind die Methoden mit dem *protected*-Zugriffsmodifizierer deklariert, so dass der Aufruf von Methoden der Basisklasse auch von abgeleiteten Klassen erfolgen kann.

```
protected SendCcTalkCommand RequestManufacturerId(CcTalkMessage message) // Header 246
{
    return new(GetTransferData(message.SourceAddress,Address,Encoding.ASCII.GetBytes(ManufacturerId)));
}
```

Quelltext 15: Implementierung der Methode *RequestManufacturerId*, Quelle: Eigene Darstellung.

Die Implementierung in Quelltext 15 zeigt z.B. die Methode *RequestManufacturerId*, die für Polling-Sequenzen mit dem ccTalk-Header 246 aufgerufen wird. Den Rückgabewert dieser Methode stellt die *ManufacturerId* vom zugehörigen Objekt der jeweiligen Device-Klasse dar. Mit der Methode *GetTransferData* wird diese in das Antwortpaket verpackt. Zu guter Letzt erfolgt über die Methode *HandleCcTalkMessage* die Publizierung des Events an den Event Aggregator. Der zugehörige Subscriber, in diesem Fall die Klasse *SerialCommander*, sendet die Daten schließlich über die serielle Schnittstelle an den Bus und somit zurück zum Master.

5.3.7.4 Verarbeitung von Münzeinwurf-Ereignissen

Wie bereits in Kapitel 5.3.6.3 beschrieben wurde, wird bei einem Münzeinwurf-Ereignis die *OnEvent*-Methode für das *InsertedMoneyEvent* aufgerufen. In dieser wiederum erfolgt der Aufruf der Methode *MoneyInserted* von der jeweiligen Klasseninstanz der Klasse *CoinAcceptorDevice*. Dieser wird als Parameter der eingegebenen Münzwert der Konsole übergeben. Anschließend wird in mehrstufigen Abfragen geprüft, ob etwaige Münzkanäle gesperrt sind oder ob es sich um nicht unterstützte Münzwertigkeiten handelt. Wird die Eingabe hingegen akzeptiert, wird diese in den Ereignisdatenspeicher *creditInsertEvents* vom Typ *CoinAcceptorBufferedCredits* geschrieben (siehe Abbildung 38). Dieser bietet Speicherplatz für bis zu fünf Ereignisse nach dem FIFO-Prinzip (First In – First Out). Folgt nun die Polling-Sequenz mit dem ccTalk-Header 229 vom Master, wird dieser Ereignisdatenspeicher ausgelesen.

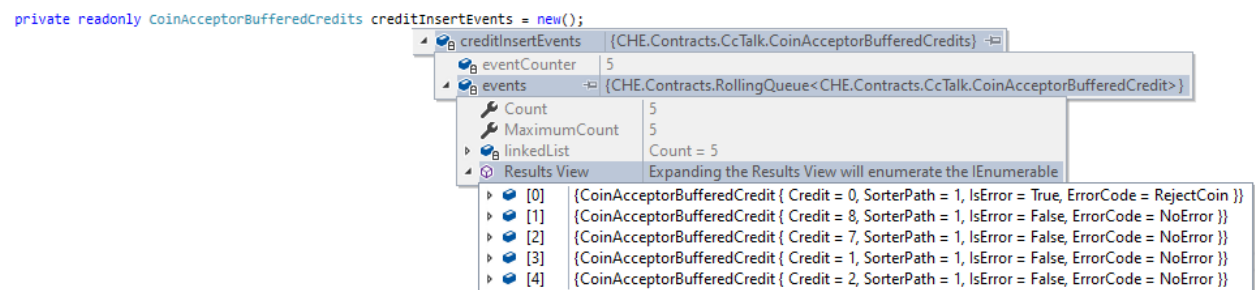


Abbildung 38: Ereignisdatenspeicher *creditInsertEvents* zur Laufzeit, Quelle: Eigene Darstellung.

5.3.7.5 Verarbeitung von Münzauszahlungs-Ereignissen

Für die Verarbeitung von Münzauszahlungen sind die jeweiligen Klasseninstanzen der Klasse *PayoutDevice* zuständig. Die Emulation der Münzauszahlung erfolgt über ein empfangenes Datenpaket vom Master mit dem ccTalk-Header 167. Die Implementierung der zugehörige Methode der Emulator-Software wird in Quelltext 16 veranschaulicht.

```
protected SendCcTalkCommand DispenseHopperCoins(CcTalkMessage message) // Header 167
{
    var configuredSerialNumber = BitConverter.GetBytes(int.Parse(SerialNumber)).Take(3).ToArray();

    byte[] receivedSerial = new byte[3];
    for (int i = 0; i < receivedSerial.Length; i++)
        receivedSerial[i] = message.Data[i];

    // Verifies if hopper is enabled and received s/n matches the device s/n
    if(hopperStatus.IsHopperEnabled() && receivedSerial.SequenceEqual(configuredSerialNumber))
    {
        hopperStatus.EventCounter++;
        hopperStatus.DispenseCounter += message.Data[3];
        hopperStatus.LastPayout_coinsPaid = message.Data[3];

        // Publishes the number of coins to be paid out to the console commander
        EventAggregator.PublishEvent(new PayoutMoneyEvent(Configuration.Address,
            Configuration.PayoutCoinIdentification,
            message.Data[3], "PAYOUT IN PROGRESS..."));

        return new(GetTransferData(message.SourceAddress, Address));
    }
    else
        return new(GetTransferData(message.SourceAddress, Address, new byte[] { NAK_message }));
}
```

Quelltext 16: Implementierung der Methode *DispenseHopperCoins*, Quelle: Eigene Darstellung.

Bei Aufruf der Methode *DispenseHopperCoins* wird die Seriennummern aus der Gerätekonfiguration, sowie die empfangen Seriennummer, zunächst in ein einheitliches Format konvertiert, so dass ein Vergleich durchgeführt werden kann. Anschließend wird in einer *if*-Anweisung verifiziert, ob die Münz auszahleinheit aktiv ist. Ist dies nicht der Fall, muss vom Master der ccTalk-Header 164 an des jeweilige Gerät gesendet werden. Des Weiteren wird abgefragt, ob die empfangene Seriennummer mit der Seriennummer der emulierten Münz auszahleinheit übereinstimmt. Zum Schutz vor Manipulationen muss der Master vor einer Auszahlung die Seriennummer des Geräts mit ccTalk-Header 242 abfragen und beim Auszahlungsbefehl mit übertragen. Ist die Emulation der Münz auszahleinheit aktiviert und stimmt die Seriennummer überein, werden im Folgeschritt diverse Zählerstände inkrementiert. Zum Schluss wird das *PayoutMoneyEvent* publiziert und vom zugehörigen Subscriber, in diesem Fall der Konsolen-Manager, subskribiert. Dieser ruft schließlich die zugehörige *OnEvent*-Methode auf und visualisiert die gerade durchgeführte Auszahlung in der Konsole.

5.3.8 Konsolen-Manager

Der Konsolen-Manager ist ein weiteres wichtiges Modul der Emulator-Software. Er hat die Aufgabe, die Befehle, die über das Kommandozeilenfenster eingegeben werden, zu verarbeiten und an andere Module der Software zu übermitteln, so dass diese auf den jeweiligen Befehl reagieren können. Weiters übernimmt der Konsolen-Manager die Ausgabe von Nachrichten, die von anderen Software-Modulen erzeugt werden, über das Kommandozeilenfenster.

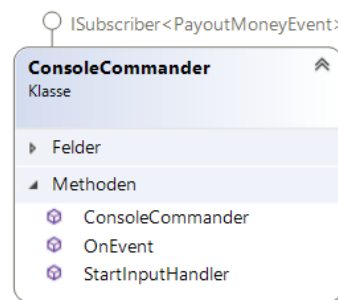


Abbildung 39: Klassendiagramm des Konsolen-Managers, Quelle: Eigene Darstellung.

Der prinzipielle Aufbau der Klasse `ConsoleCommander` wird anhand des Klassendiagramms in Abbildung 39 veranschaulicht. Wie man zunächst erkennen kann, agiert auch die Klasse `ConsoleCommander` als Subscriber, in dem er das `PayoutMoneyEvent` vom Event Aggregator subskribiert. Die entsprechende Ereignisroutine ist in der zugehörigen Methode `OnEvent` implementiert und dient zur Anzeige von Münzauszahlungen im Kommandozeilenfenster. Die Methode `StartInputHandler` dient hingegen zum Einlesen von eingegebenen Kommandos über die Tastatur. Die zugehörigen Methodendefinitionen werden in den folgenden Unterkapiteln kurz näher erläutert.

5.3.8.1 Eingabehandler

Der Eingabehandler wird mit dem Aufruf der Methode `StartInputHandler` unmittelbar nach Instanziierung der Klasse `ConsoleCommander` in der `Startup`-Klasse gestartet. Innerhalb der Methode wird zunächst der Task, der zum Erkennen und Verarbeiten von Eingaben über das Kommandozeilenfenster zuständig ist, dem Threadpool hinzugefügt. Die Implementierung dieses Tasks wird im Codeausschnitt des nachfolgenden Quelltext 17 dargestellt. Anfangs wird eine Schleife gestartet, in welcher die eingegebenen Kommandos im Kommandozeilenfenster eingelesen und verarbeitet werden. Zum Beenden der Schleife dient der zuvor erzeugte Abbruchtoken als Bedingung. Innerhalb der Schleife wird die eingegebene Zeichenfolge eingelesen und in ein vordefiniertes Format konvertiert. Die Separierung der Zeichenfolge in Kommando, Zieladresse und Eingabewert erfolgt durch ein Komma. Anschließend wird anhand des eingegebenen Kommandos mit Hilfe der `switch`-Anweisung das entsprechende Event an den Event Aggregator publiziert. Vorerst wird hier nur das `InsertedMoneyEvent` implementiert, welches vom Geräte-Manager subskribiert wird und die Münzeinwurf-Ereignisse in den Ereignisdatenspeicher des emulierten Münzprüfers schreibt. Zukünftig könnten hier sehr einfach zusätzliche Kommandos, wie z.B. zum Simulieren von Fehlern, hinzugefügt werden.

```
// Start loop for user input commands
logger?.Debug("Console input handler started...");
while (!token.IsCancellationRequested)
{
    // Read user input and convert it into a usable format
    var rawUserInput = Console.ReadLine();

    if (string.IsNullOrEmpty(rawUserInput))
        continue;

    string[] splitInput = rawUserInput.Split(',');
    string command = splitInput[0];
    byte.TryParse(splitInput[1], out var destinationAddress);
    int.TryParse(splitInput[2], out var value);

    // Assign entered command to the appropriate event
    switch(command)
    {
        case "cin":
            logger?.Debug($"Executing user command <{command}>");
            eventAggregator?.PublishEvent(new InsertedMoneyEvent(destinationAddress, value));
            break;

        default:
            logger?.Debug($"Invalid Command <{command}>");
            Console.WriteLine($"<=== Invalid command entered <{command}> ===>");
            break;
    }
}
}
```

Quelltext 17: Codeausschnitt der Methode *StartInputHandler*, Quelle: Eigene Darstellung.

In Abbildung 40 wird veranschaulicht, wie die Eingabe der Kommandos im Kommandozeilenfenster erfolgt. Wie bereits zuvor erwähnt, erfolgt die Eingabe im Format *Kommando,Zieladresse,Eingabewert*. Das Kommando *cin* (*coin in*) signalisiert dabei einen Münzeinwurf, die Zieladresse den angesprochenen Münzprüfer und der Eingabewert die Wertigkeit des emulierten Münzeinwurfs. Dieser wird in Form der Denomination, welche die kleinste Stückelung einer Währung angibt, eingegeben (z.B. 1 = 1 Eurocent, 5 = 5 Eurocents, 100 = 100 Eurocents = 1 Euro). Die ersten drei Eingaben zeigen akzeptierte Münzeinwurf-Ereignisse. Die anderen Eingaben werden auf Grund unterschiedlicher Ursachen nicht akzeptiert.

Abbildung 40: Münzeinwurf im Kommandozeilenfenster, Quelle: Eigene Darstellung.

5.3.8.2 OnEvent-Methode (PayoutMoneyEvent)

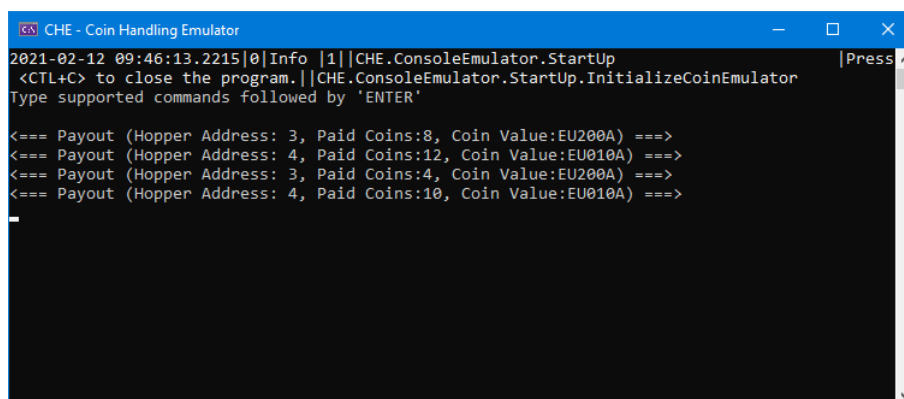
Da der Konsolen-Manager auch als Subscriber agiert, wird hier die Methode *OnEvent* für das *PayoutMoneyEvent* implementiert (siehe Quelltext 18). Diese Ereignisroutine wird ausgelöst, wenn die Klasse *PayoutDevice* eine Auszahlung von Münzen publiziert. Dazu wird das *ConsoleCommander* Objekt durch die *lock*-Anweisung für andere Threads gesperrt. Anschließend werden die auszubehandelnden Münzen der jeweiligen Münzauszahlungseinheit im Kommandozeilenfenster ausgegeben. Als Parameter werden der Methode die Quelladresse der Auszahlungseinheit, sowie die Anzahl und die Wertigkeit der Münzen übergeben.

```
public void OnEvent(PayoutMoneyEvent receivedRawData)
{
    lock (LockConsoleWriter)
    {
        logger?.Debug($"Payout money event received (Hopper Address <{receivedRawData.Address}>.");

        // Visualize paid out coins in the command-line interface
        Console.WriteLine($"<=== Payout (Hopper Address: {receivedRawData.Address},
                            Paid Coins:{receivedRawData.Coins},
                            Coin Value:{receivedRawData.CoinValue}) ===>");
    }
}
```

Quelltext 18: Implementierung der Methode *OnEvent(PayoutMoneyEvent)*, Quelle: Eigene Darstellung.

Wie diese Auszahlungen im Kommandozeilenfenster dargestellt werden, zeigt Abbildung 41. In vier Schritten erfolgt hier die Auszahlung einer unterschiedliche Anzahl von Münzen von unterschiedlichen Auszahlungseinheiten, die der Emulator emuliert. Die Auszahlung der ersten Zeile stammt beispielsweise von der Auszahlungseinheit mit Quelladresse 3 und zahlt 8 Münzen mit einer Wertigkeit von EU200A (200 Eurocents = 2 Euro) aus.



```
CHE - Coin Handling Emulator
2021-02-12 09:46:13.2215|0|Info |1||CHE.ConsoleEmulator.StartUp
<CTL+C> to close the program.||CHE.ConsoleEmulator.StartUp.InitializeCoinEmulator
Type supported commands followed by 'ENTER'

<=== Payout (Hopper Address: 3, Paid Coins:8, Coin Value:EU200A) ===>
<=== Payout (Hopper Address: 4, Paid Coins:12, Coin Value:EU010A) ===>
<=== Payout (Hopper Address: 3, Paid Coins:4, Coin Value:EU200A) ===>
<=== Payout (Hopper Address: 4, Paid Coins:10, Coin Value:EU010A) ===>
```

Abbildung 41: Münzauszahlung im Kommandozeilenfenster, Quelle: Eigene Darstellung.

5.3.9 Logging-Service

Zur Protokollierung von Ereignissen wird die in Kapitel 5.2.4 beschriebene Plattform NLog verwendet. Da NLog ein fertiges API bereitstellt, das über den NuGet-Paket-Manager installiert werden kann, ist dessen Einsatz mit relativ geringem Aufwand möglich. Nach der Installation und einbinden des Namespaces in den Programmcode der *Startup*-Klasse, muss NLog lediglich richtig konfiguriert werden (siehe Quelltext 19). Hierfür muss zunächst die Klasseninstanz vom Interface *ILogger* deklariert werden. Anschließend erfolgt

die Konfiguration mit Hilfe der Klasse *LogManager*, die auf die entsprechende Instanz *Logger* angewendet wird. Anschließend kann die Protokollierung unter Angabe des jeweiligen Log-Levels erfolgen.

```
/// <summary>
/// Gets the logging service instance.
/// </summary>
/// <value>The logging service instance.</value>
private static ILogger? Logger { get; set; }

// Configure Logger:
LogManager.Configuration = configuration.GetNLogConfiguration();
Logger = LogManager.GetCurrentClassLogger();
Logger.Debug("Starting Coin Handling Emulator...");
```

Quelltext 19: Codeausschnitt zur Konfiguration von NLog, Quelle: Eigene Darstellung.

Um schließlich auch in anderen Modulen der Emulator-Software protokollieren zu können, muss in jedem Modul die Logging-Server-Instanz deklariert und zugewiesen werden. Dies erfolgt in den Konstruktoren der jeweiligen Klassen.

Die Protokollierung wird neben der Ausgabe im Kommandozeilenfenster auch in einer entsprechenden Logdatei gespeichert. Die Logdatei *CHE.log* befindet sich im zugehörigen Verzeichnis der kompilierten Emulator-Software. Wie man anhand vom Abbildung 42 erkennen kann, werden alle Ereignisse mit Zeitstempel, Log-Level und Auflistung der jeweiligen Software-Komponente, von der das Ereignis stammt, in dieser Logdatei gespeichert.

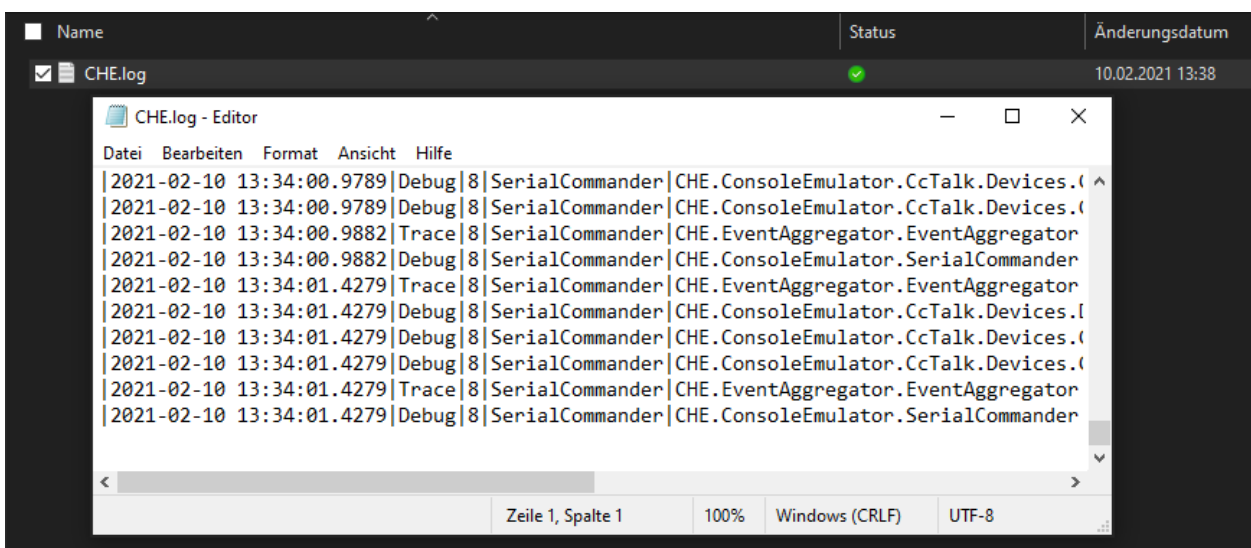


Abbildung 42: Logdatei im Projektverzeichnis, Quelle: Eigene Darstellung.

5.4 Darstellung der programmtechnischen Abläufe bei Events

Um das Verständnis für das Zusammenwirken der einzelnen Software-Module zu verbessern, wird im Nachfolgenden der Programmablauf anhand unterschiedlicher Aktivitätsdiagrammen kurz erläutert.

5.4.1 ReceivedCcTalkCommand und SendCcTalkCommand

In Abbildung 43 wird der grundsätzliche Ablauf beim Empfangen und Senden von ccTalk-Polls dargestellt. Sendet die Spielplattform-Software eine Anfrage, dann wird das Datenpaket vom Bustreiber eingelesen und das Event *ReceivedCcTalkCommand* publiziert. Der Subscriber Geräte-Manager ordnet die Anfrage der Geräteemulation zu, welche das Antwortpaket aufbereitet und anschließend die Antwort durch das Event *SendCcTalkCommand* publiziert. Der Subscriber Bustreiber sendet die Daten an den ccTalk-Bus.

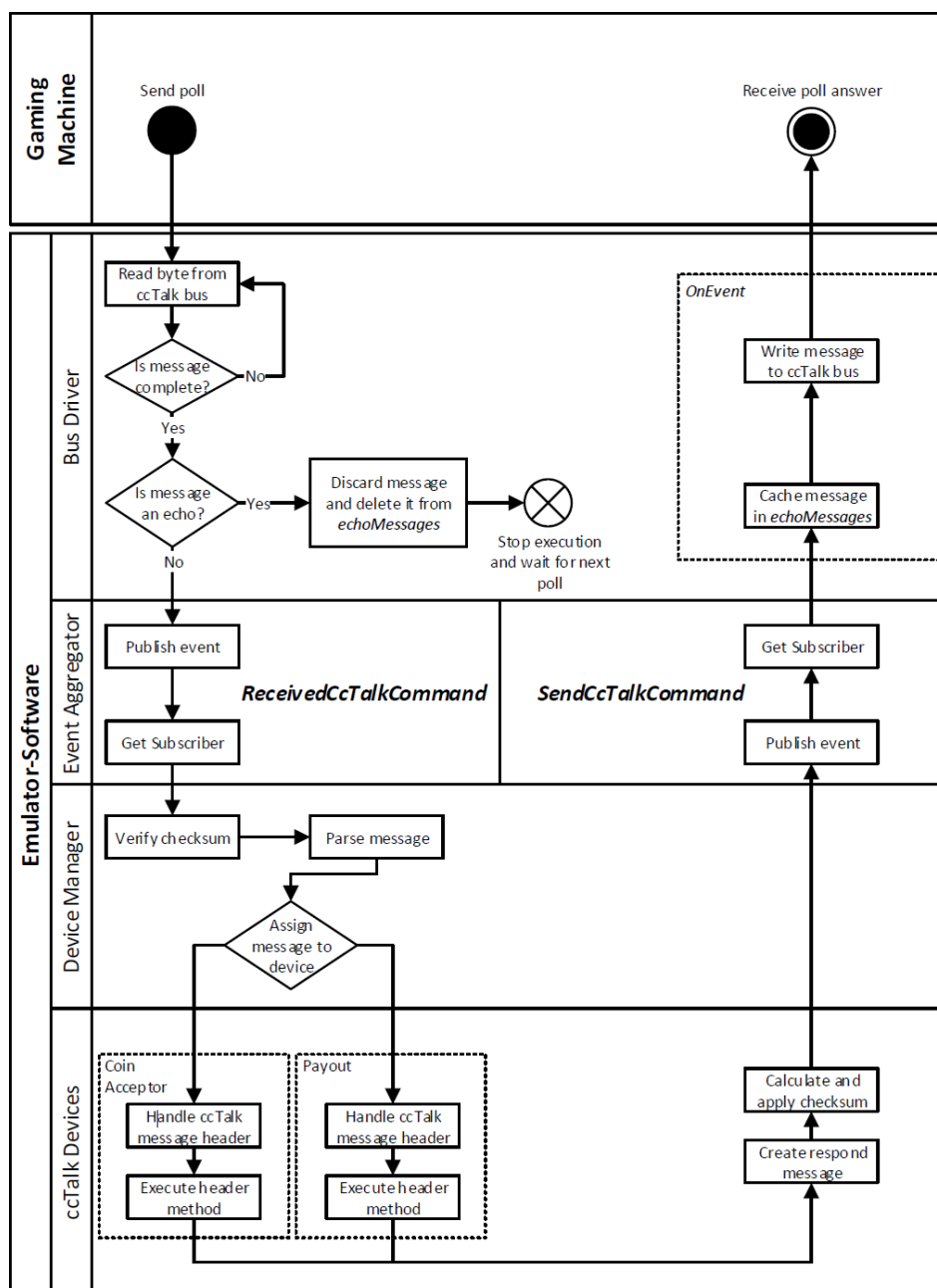


Abbildung 43: Aktivitätsdiagramm für die Events *ReceivedCcTalkCommand* und *SendCcTalkCommand*, Quelle: Eigene Darstellung.

5.4.2 InsertedMoneyEvent

Das in Abbildung 44 dargestellte Aktivitätsdiagramm beschreibt den programmtechnischen Ablauf beim Emulieren eines Münzeinwurf-Ereignisses durch das Kommandozeilenfenster. Nach der manuellen Eingabe des entsprechenden Kommandos durch den Benutzer, wird dieses vom Konsolen-Manager eingelesen, interpretiert und das *InsertedMoneyEvent* publiziert. Anschließend wird die *OnEvent*-Methode des Geräte-Managers ausgeführt, wodurch das *cin* Kommando der Münzprüfer-Emulation zugeordnet wird. Die Emulation prüft anschließend, ob der Master-Blockier-Status gesetzt ist oder einzelne Münzkanäle gesperrt sind. Ist dies nicht der Fall, dann wird der akzeptierte Münzwert in den Ereignisdatenspeicher des emulierten Münzprüfers gespeichert. Im Zuge der nächsten Polling-Sequenz durch die Spielplattform-Software wird der Speicher mittels ccTalk-Header 229 abgefragt.

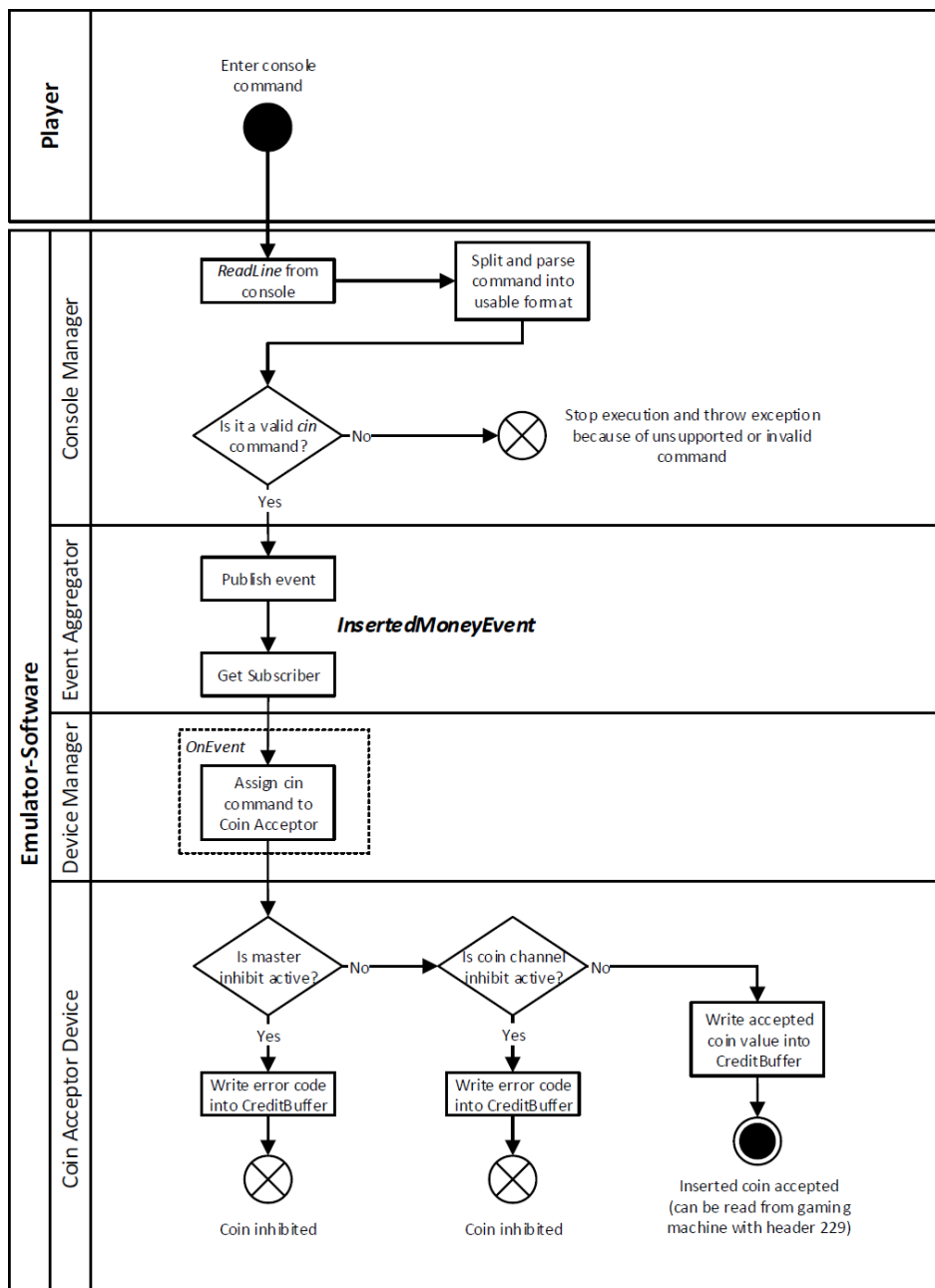


Abbildung 44: Aktivitätsdiagramm für das *InsertedMoneyEvent*, Quelle: Eigene Darstellung.

5.4.3 PayoutMoneyEvent

Die Durchführung von Münzauszahlungs-Ereignissen durch die Emulation einer der Münzauszahlereinheiten wird in Abbildung 45 veranschaulicht. Wird von der Spielplattform-Software der Auszahlungsbefehl mit ccTalk-Header 167 empfangen, dann wird die entsprechende Geräteemulation der Auszahlereinheit aufgerufen. Auf die Beschreibung vorangehender Abläufe zum Empfangen und Senden von Polls wird hier verzichtet, weil diese bereits in Kapitel 5.4.1 erläutert wurden. Ausgangspunkt ist somit der empfangene Auszahlungsbefehl, der folglich die Payout-Routine der Geräteemulation aufruft. In dieser Routine wird geprüft, ob Münzauszahlereinheit aktiviert ist und die empfangene Seriennummer mit der Nummer der Emulation übereinstimmt. Anschließend werden interne Zählervariablen inkrementiert und das *PayoutMoneyEvent* publiziert. Der Konsolen-Manager, der diese Event subskribiert, führt die *OnEvent*-Methode aus und visualisiert die Auszahlung im Kommandozeilenfenster, wo sie vom Benutzer eingesehen werden kann.

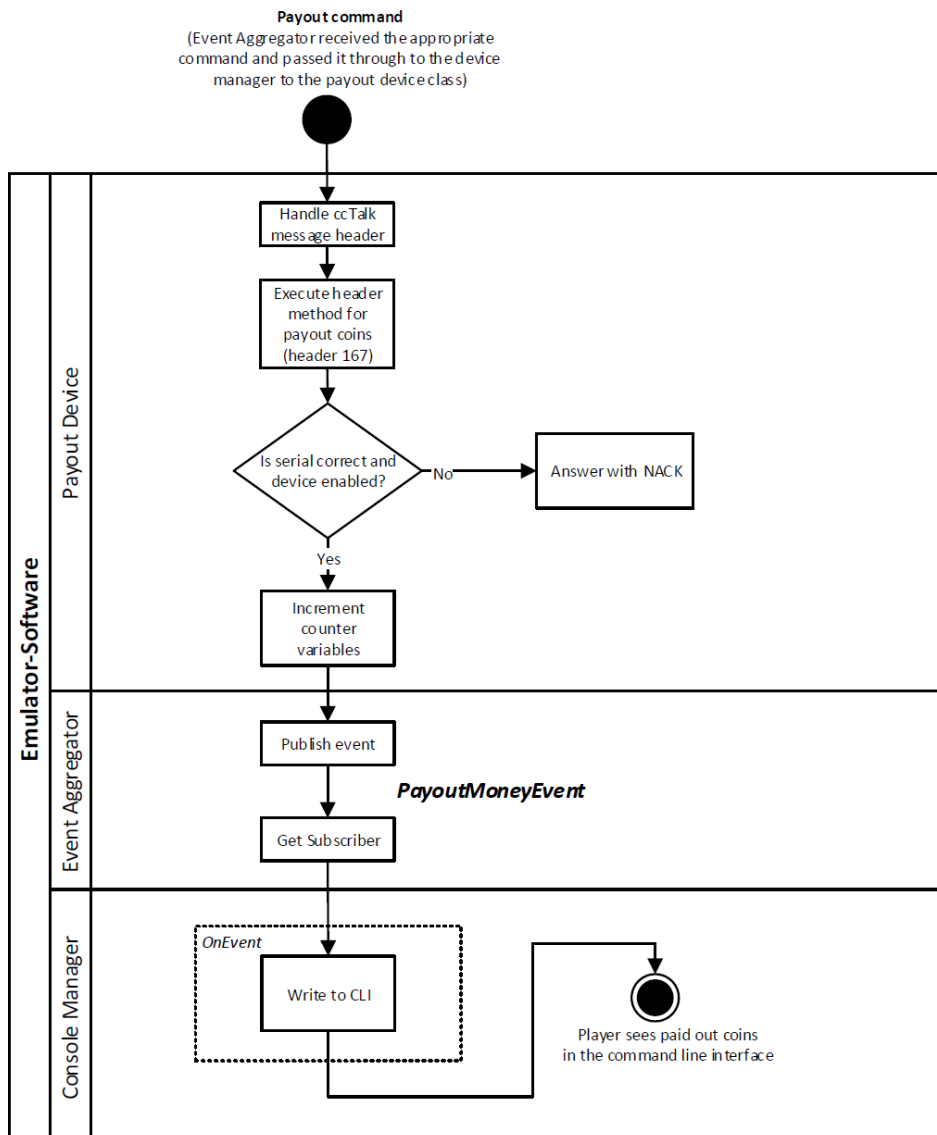


Abbildung 45: Aktivitätsdiagramm für das PayoutMoneyEvent, Quelle: Eigene Darstellung.

6 EVALUIERUNG DES PROTOTYPS

Um erste Eindrücke zur Umsetzung des Emulators zu erlangen, gilt es diesen möglichst frühzeitig auf seine Funktion zu überprüfen. Basierend auf den Erkenntnissen dieser Überprüfung, kann bei auftretenden Problemen oder Fehlern bereits in frühen Entwicklungsphasen gegengesteuert werden.

Im Fokus dieses Kapitels steht daher der gesamte Evaluierungsprozess für das zu entwickelnde System. Hierzu wird anfänglich auf die entwicklungsbegleitenden Tests eingegangen, welche kontinuierlich während des gesamten Entwicklungsprozesses durchgeführt werden, um einzelne Funktionalitäten bereits vorab zu überprüfen. In weiterer Folge wird der fertig entwickelte Prototyp der Emulator-Software in seinem definierten Einsatzgebiet, dem Spielautomaten, getestet. Zur Schlussfolgerung werden die gewonnenen Erkenntnisse aus der Evaluierung schließlich zusammengefasst.

6.1 Entwicklungsbegleitendes Testen

Zur Erlangung von Kenntnissen über die Funktionsweise einzelner Module, Teilbereiche oder Methoden aus der Software, werden im Zuge des gesamten Entwicklungsprozesses kontinuierliche Tests und Systemevaluierungen vorgenommen. Da durch diese permanenten Tests die Risiken in Bezug auf kritische Fehler und Probleme minimiert werden, ergibt sich der Vorteil in einer wesentlich höheren Produktqualität und zuverlässigeren Entwicklung durch das ständig erlangte Feedback.

Die Durchführung dieser Tests erfolgt unter Verwendung eines simplen Prüfaufbaus, der es ermöglicht, die serielle Kommunikation mit der Emulator-Software zu simulieren, ohne dass ein Spielautomat dafür benötigt wird. Dies ermöglicht die Abfrage der unterschiedlichen Befehle aus der ccTalk-Spezifikation vom Emulator. Wie genau dieser Prüfaufbau realisiert wird, welche Zusatzprogramme benötigt werden und wie die Tests einzelner Befehle im Detail aussehen, wird in den nachfolgenden Kapiteln beschrieben.

6.1.1 Realisierung des Prüfaufbaus

Der Prüfaufbau, der für die entwicklungsbegleitenden Tests zum Einsatz kommt, wird in Abbildung 46 veranschaulicht. Dieser besteht aus der Entwicklungsplattform, auf welcher der Programmcode in der Visual Studio IDE entwickelt wird. Darüber hinaus werden für diese Art von Funktionstests auf der Entwicklungsplattform einige Zusatzprogramme (siehe 6.1.2) ausgeführt. Mit der Hilfe von drei seriellen Schnittstellen-Konvertern, die über USB verbunden sind, werden drei virtuelle COM-Ports vom Betriebssystem erzeugt. Unter Verwendung der in Kapitel 3.2.2.2 beschriebenen Schaltung, wandeln die jeweils nachgeschalteten ccTalk-Schnittstellen-Konverter die seriellen RS-232 Signale in ccTalk-kompatible Signale um. Zur Erzeugung der benötigten Busspannung, die auch die Schnittstellen-Konverter versorgt, wird einer der Konverter mit einem DC-Netzteil versorgt.

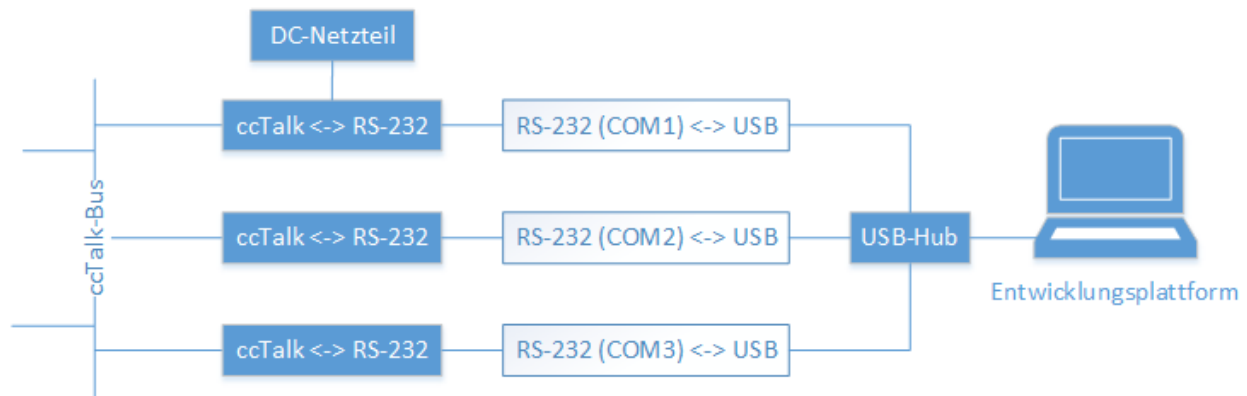


Abbildung 46: Prüfaufbau für entwicklungsbegleitende Tests mit der Entwicklungsplattform, Quelle: Eigene Darstellung.

Mit Hilfe dieses Prüfaufbaus lässt sich sowohl das Empfangen und Verarbeiten, als auch das Senden von Antworten auf einzelnen ccTalk-Header testen. Die Signalkonvertierung mit Hilfe der ccTalk-Schnittstellen-Konverter ermöglicht außerdem die Nachahmung einer realen Bus-Umgebung. Diese unterstützt vor allem bei der Entwicklung des seriellen Treiberbausteins, weil hier auch zeitliche Abhängigkeiten und Echos auf gesendete Datenpakete wie im Realfall auftreten.

Je nach vorliegender Testsituation werden zwei oder drei der virtuellen COM-Ports für die Tests verwendet. Im Regelfall dient ein COM-Port als serielle Schnittstelle für die Emulator-Software und ein COM-Port als Schnittstelle für Zusatzprogramme von Herstellern, die normalerweise dazu gedacht sind, um reale Peripheriegeräte anzusteuern. Der dritte COM-Port wird letztendlich zur Aufzeichnung des Datenverkehrs zwischen den anderen beiden Ports für Analysezwecke eingesetzt.

6.1.2 Eingesetzte Zusatzprogramme

Im Zuge der Tests kommen einige Zusatzprogramme zum Einsatz. Dies können einerseits Programme von Herstellern sein, die eigentlich dazu gedacht sind, um ihre Peripheriegeräte, wie Münzprüfer oder Münzauszahlungen für Testzwecke zu bedienen. Andererseits sind aber auch Terminal-Programme für die serielle Kommunikation hilfreich, um manuelle Tests einzelner Befehle durchzuführen.

Für die hier durchgeführten Tests werden vorwiegend die folgenden beiden Programme eingesetzt:

SeciCCtalk

Die freie Software SeciCCtalk wurde von der Comestero Group, einem Hersteller von Bezahlssystem-Komponenten der heute zu SUZOHAPP zählt, entwickelt. Es handelt sich dabei um eine Software zum Ansteuern und Testen von elektronischen Münzprüfern, die eine Vielzahl der relevanten Header für Münzprüfer aus der ccTalk-Spezifikation implementiert hat.

Die Benutzeroberfläche der Software wird in Abbildung 47 gezeigt. Unter der Registerkarte *Group 1* lassen sich alle gängigen modellspezifischen Identifikatoren von Münzprüfern und auch der Ereignisdatenspeicher abfragen. Die Registerkarte *Group 2* ermöglicht hingegen die Zuweisung von bestimmten Münzkanälen zur Sortiereinheit, das Auslesen des Währungscode und das Sperren einzelner Münzkanäle.

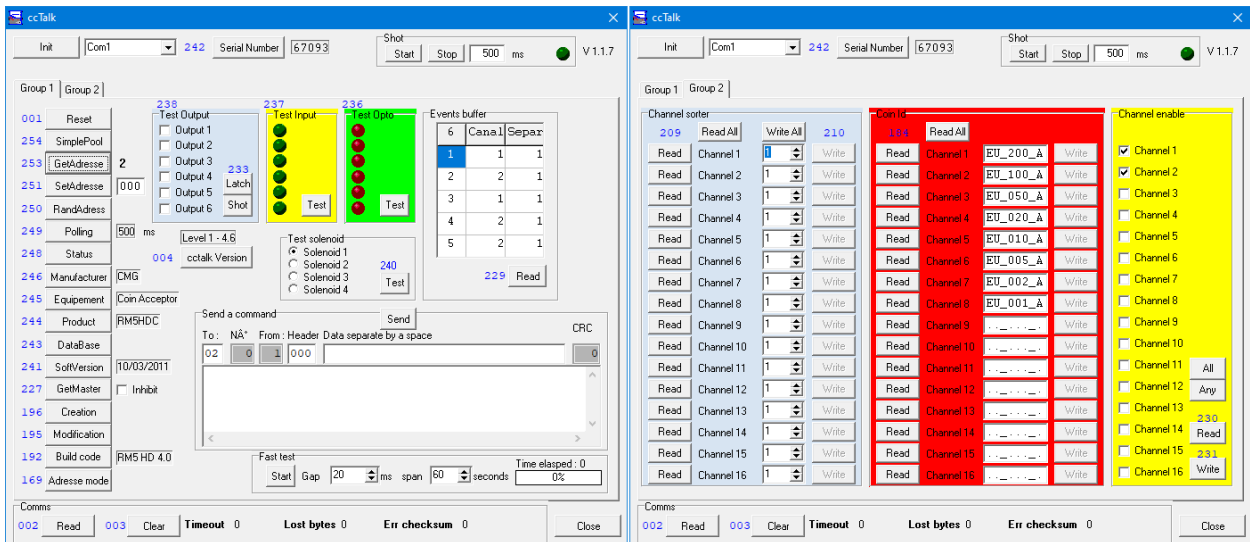


Abbildung 47: Benutzeroberfläche der Software SeciCctalk, Quelle: Eigene Darstellung.

Bei der Entwicklung des Software-Moduls für den Geräte-Manager und dem Münzprüfer-Modul aus der ccTalk-Bibliothek (siehe Kapitel 5.3.6 und 5.3.7) stellt SeciCctalk eine wesentliche Erleichterung dar. Insbesondere Datenpakete mit einer großen Anzahl an zu übertragenden Datenbytes können mit der Software einfach und in strukturierter Form dargestellt und analysiert werden. Dazu zählen der Ereignisdatenspeicher oder die Darstellung von Bitmustern in Statusregistern, wie z.B. die zum Sperren einzelner Münzkanäle. Weiters ermöglicht die Software die zyklische Abfrage des Ereignisdatenspeichers in einem zuvor definierten Zeitintervall, der in der Regel 500 ms beträgt. Somit kann der emulierte Münzprüfer unter möglichst realen Bedingungen getestet werden.

HTerm

Eine weitere wichtige Software, die bereits während der Entwicklung des Emulators eingesetzt wurde, ist HTerm. Bei HTerm, dessen Benutzeroberfläche in Abbildung 48 dargestellt wird, handelt es sich um ein gängiges Terminal-Programm für serielle Datenkommunikation. Über den eingestellten virtuellen COM-Port lassen sich somit Daten über die serielle Schnittstelle empfangen und senden.

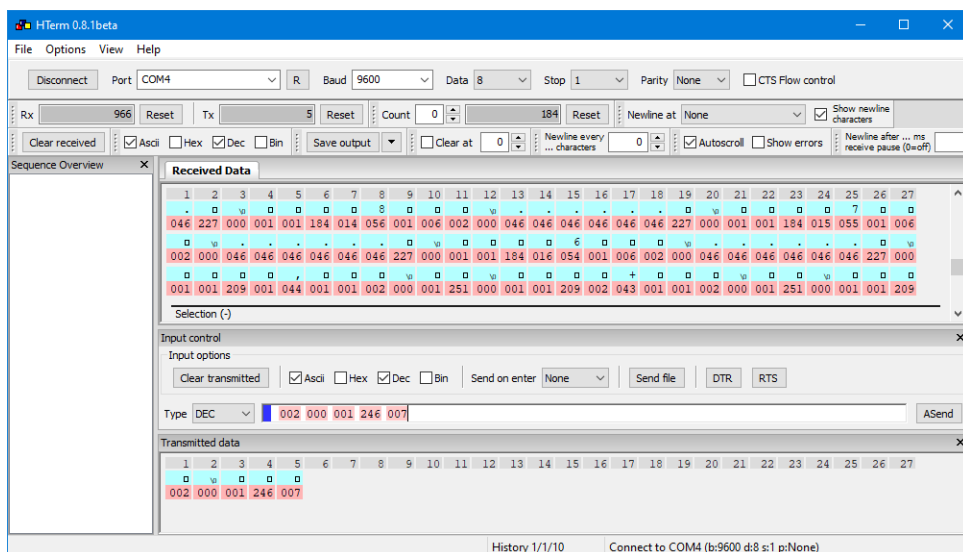


Abbildung 48: Benutzeroberfläche der Software HTerm, Quelle: Eigene Darstellung.

HTerm wird während des Entwicklungsprozesses bevorzugt zum Senden einzelner Polling-Sequenzen an die Emulator-Software verwendet. In der Visual Studio IDE können anschließend mit dem Debugger einzelne Ereignisroutinen im Programmcode durch das Setzen von Haltepunkten verifiziert werden. Da für die Ansteuerung der emulierten Münzanzahlmodulen keine passende Test-Software zur Verfügung steht, wird für die Entwicklung des Münzanzahlmoduls ausschließlich HTerm eingesetzt, um die entsprechenden ccTalk-Header während der Entwicklung zu testen. Weiters wird die Software zur Aufzeichnung der seriellen Datenkommunikation zwischen der Emulator-Software und der SeciCCtalk Software oder im späteren Entwicklungsverlauf zwischen der Emulator-Software und der Spielplattform-Software eingesetzt. So lassen sich beim Auftreten von Fehlern in der Kommunikation einzelne Datenpakete einfach analysieren.

6.1.3 Funktionstest einzelner ccTalk-Header

Der Verlauf der Softwareentwicklung macht es notwendig, die entsprechenden Ereignisroutinen auf einzelne Polls, die gerade in der Software implementiert werden, auf ihre Funktion zu überprüfen. Unter Beachtung des Protokollrahmens (siehe 3.2.3.1) werden demnach einzelne Header manuell an die ausgeführte Emulator-Software gesendet und die empfangenen Antworten verifiziert. Die Softwarefunktion zum automatischen Senden von Datenpaketen in gewissen Zeitintervallen ermöglicht zusätzlich eine Art von Stresstest, wodurch die zeitliche Anforderung geprüft werden kann (siehe Kapitel 3.3.2.3). Die Eingabe der Adressen, Datenbytes und Header erfolgt dabei im Dezimalformat. Checksummen müssen manuell berechnet werden. Wie die Eingabe in der Software erfolgt, wird in Abbildung 49 veranschaulicht.

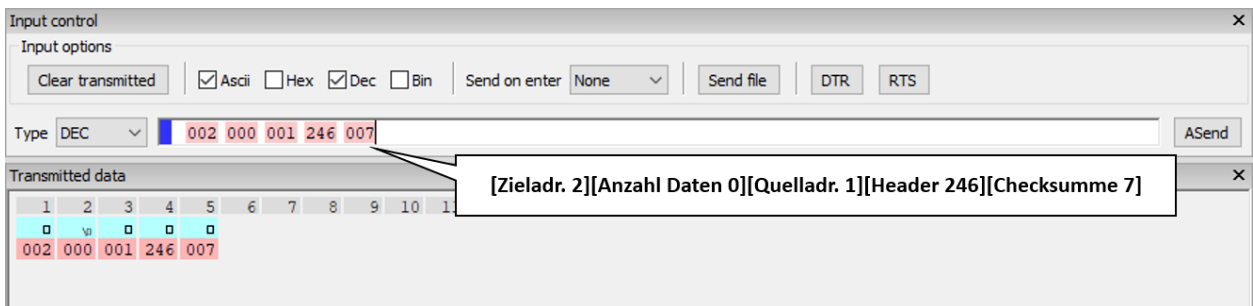


Abbildung 49: Senden eines Datenpaketes in der Software HTerm , Quelle: Eigene Darstellung.

Die Empfangenen Daten vom Bus zeigt hingegen Abbildung 50. Hierbei ist anzumerken, dass die ersten Bytes immer die Echos der gesendeten Datenpakete sind, und die blau hinterlegten Bytes die Antworten.

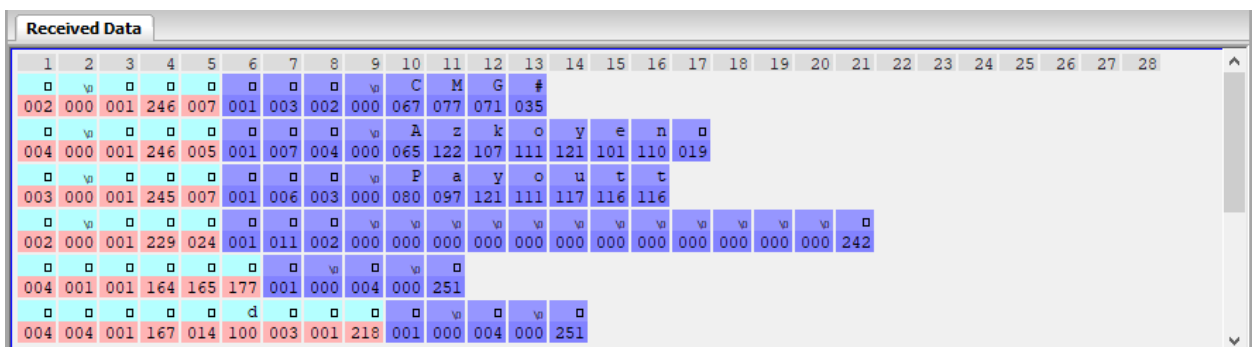


Abbildung 50: Empfangene Datenpakete in der Software HTerm, Quelle: Eigene Darstellung.

6.2 Evaluierung des Emulators im Spielautomaten

Mit der Fertigstellung eines funktionsfähigen Prototyps, der bereits einige entwicklungsbegleitende Tests aus Kapitel 6.1 erfolgreich absolviert hat, wird ein Punkt erreicht, an dem erste Funktionstests unter realen Einsatzbedingungen durchgeführt werden können. Wie die Durchführung dieser Tests aussieht und worauf es im Detail ankommt, wird im Folgenden ausführlich erläutert.

6.2.1 Funktionstests auf der Entwicklungsplattform

Im ersten Schritt erfolgen die Funktionstests unter Ausführung der entwickelten Emulator-Software auf der Entwicklungsplattform. Der Prüfaufbau, der in Abbildung 51 ersichtlich ist, wird hierfür im Gegensatz zum vorherigen Aufbau entsprechend angepasst. Die Kommunikation zum Bus erfolgt jetzt nur noch über zwei virtuelle COM-Ports mit nachgeschalteten ccTalk-Schnittstellen-Konvertern. Ein Port dient der Kommunikation mit der Emulator-Software und ein weiterer Port steht optional für die Analyse der Bus-Kommunikation zur Verfügung. Weiters werden die elektrischen Verbindungen zu den bestehenden Komponenten des Münzverarbeitungssystems im Spielautomaten, wie der Münzprüfer und die Münzauszahlungen, elektrisch getrennt. Anschließend wird stattdessen die Entwicklungsplattform über die den jeweiligen Schnittstellen-Konverter mit der Spielplattform verbunden. Dass somit die grundlegende Kommunikation, zwischen der Emulator- und der Spielplattform-Software gewährleistet ist, wird durch die entsprechende Protokollierung einzelner Polling-Sequenzen im CLI und auch in der Logdatei ersichtlich.

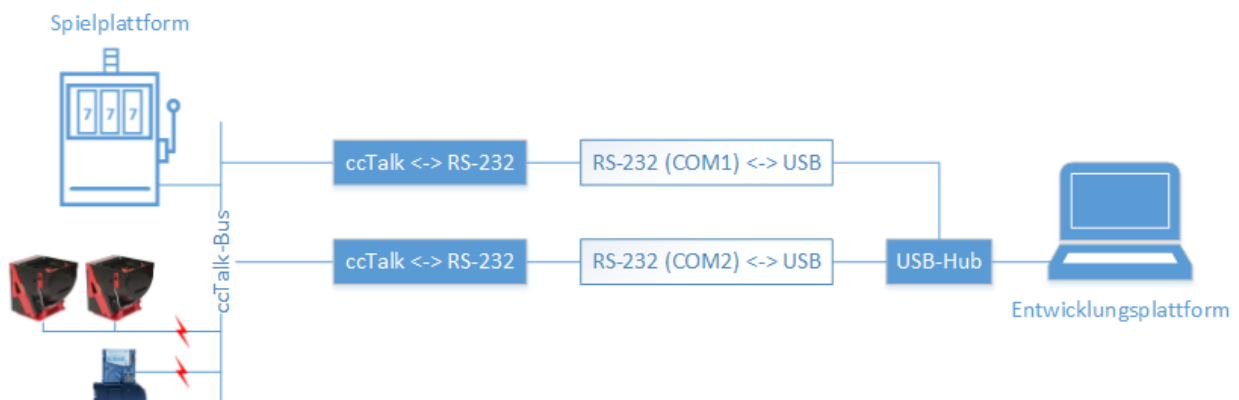


Abbildung 51: Prüfaufbau für den Funktionstest mit der Entwicklungsplattform, Quelle: Eigene Darstellung.

Wie man anhand der Bildschirmanzeige in Abbildung 52 erkennen kann, wird die Unterbrechung der elektrischen Verbindungen zum Münzprüfer und zu den Münzauszahlungen mit den IDs 2 bis 4 sofort von der Spielplattform-Software erkannt. Dies hat zur Folge, dass der Spielautomat unmittelbar in den Out-of-Service-Modus wechselt. Wird anschließend die Emulator-Software auf der Entwicklungsplattform gestartet und dessen modellspezifische Identifikatoren weichen von den vorherigen Identifikatoren der realen Komponenten ab, wird dies ebenfalls signalisiert. Würde der Emulator hingegen exakt dieselben modellspezifischen Identifikatoren emulieren wie die zuvor abgesteckten Komponenten aufweisen, dann würde die Spielplattform-Software ohne jegliche Art von Neukonfiguration vom Out-of-Service-Modus wieder in den Spielbetrieb wechseln.

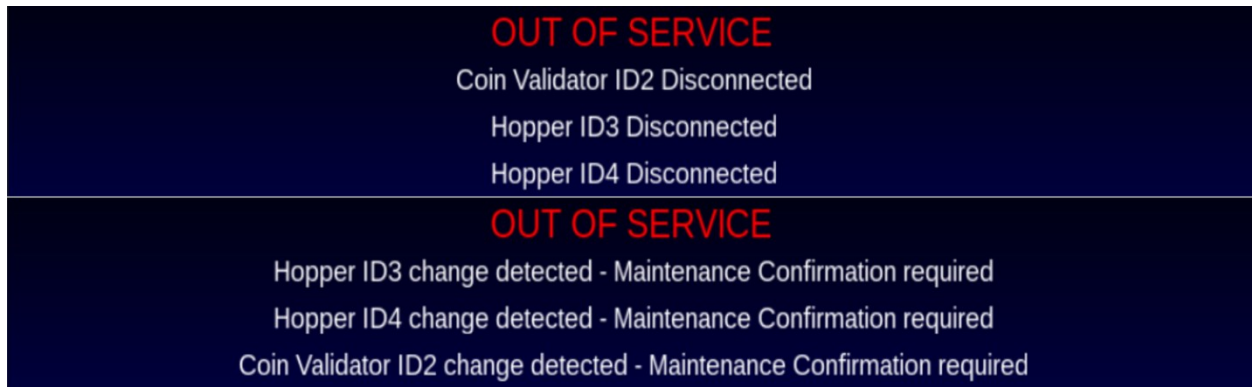


Abbildung 52: Spielautomaten Fehlermeldungen beim Anbinden des Emulators, Quelle: Eigene Darstellung.

Wenn die Parameter der Emulation von denen der realen Komponenten abweichen, dann muss diese neue Konfiguration in der Software des Spielautomaten freigegeben werden. Dies dient unter anderem als Manipulationsschutz, weil somit die willkürliche Installation neuer Bezahlssystemkomponenten verhindert wird.

Für die Freigabe der neu verbauten Komponenten bzw. des Emulators ist es zunächst notwendig sich mit einem Passwort am Spielautomaten als Administrator zu identifizieren. Anschließend muss im Wartungsmodus jede Änderung einer Komponente gesondert freigegeben werden. In der folgenden Abbildung 53 werden die im Spielautomaten installierten Bezahlssystemkomponenten angezeigt. Die obere Tabelle zeigt die gelisteten Komponenten vor Freigabe der neuen Peripherie im Wartungsmodus. Dabei handelt es sich um die reale Konfiguration vor Einbau des Emulators. Die untere Tabelle zeigt demgegenüber die emulierten Peripheriegeräte nach der erfolgten Freigabe in der Spielplattform-Software des Spielautomaten.

Installed Money Devices							Before
ID	Type	Producer	Model	FW	S/N	Available	
02	Validator	AZKOYEN	X6-I	X6-I ccTalk V18.0 41177071-18 1	2771718	No	
03	Hopper	Azkoyen	Payout	55.50 Payout 1.4.3.0	882709	No	
04	Hopper	Azkoyen	Payout	55.50 Payout 1.4.3.0	882712	No	

Installed Money Devices							After
ID	Type	Producer	Model	FW	S/N	Available	
02	Validator	CMG	RM5HDC	10/03/2011 RM5 HD 4.0 1.4.6.0	67093	Yes	
03	Hopper	Azkoyen	UPLUS	55.50 Payout 1.4.6.0	111111	Yes	
04	Hopper	Azkoyen	UPLUS	55.50 Payout 1.4.6.0	222222	Yes	

Abbildung 53: Installierte Bezahlssystemkomponenten vor und nach dem Anbinden des Emulators, Quelle: Eigene Darstellung.

Wie sich anhand der Verfügbarkeit der emulierten Geräten erkennen lässt, ist die grundlegende Kommunikation mit der ausgeführten Emulator-Software somit gewährleistet. Darüber hinaus werden auch alle modellspezifischen Identifikatoren aus der Emulator-Konfiguration korrekt von der Spielautomaten-Software dargestellt. Nach Verlassen des Wartungsmodus wechselt der Spielautomat automatisch in den Spielbetrieb. Der angebundene Emulator verhält sich somit, wie es in den Anforderungen an den Emulator gefordert wird (siehe Kapitel 3.3.2.2), wie echte Hardware und es werden keinerlei Fehlermeldungen von der Spielplattform-Software hervorgerufen.

Aus den zuvor genannten Gegebenheiten lässt sich schlussfolgern, dass Münzeinwurf-Ereignisse vom Spielautomaten erkannt werden müssten, und auch durch den Spielautomaten ausgelöste Auszahlungen im Kommandozeilenfenster des Emulators visualisiert werden sollten. Im nächsten Schritt wird daher die Emulation von Ein- und Auszahlungen getestet. Bei der verwendeten Testkonfiguration des Spielautomaten handelt es sich um eine Version für den AWP-Markt Italien, weshalb die Ein- und Auszahlung von Münzen limitiert ist. Die Software akzeptiert somit ausschließlich 1 Euro- und 2 Euro-Münzen und es können nicht mehr als 2 Euro an Krediten auf einmal aufgebucht werden.

Die nachfolgende Abbildung 54 zeigt den Test anhand eines Screenshots der Spielplattform-Software am Spielautomaten sowie einem Abbild vom Kommandozeilenfenster der ausgeführten Emulator-Software auf der Entwicklungsplattform. Wie sich daraus erkennen lässt, können mit dem Kommando *cin* bis zu 2 Euro an Krediten auf den Automaten aufgebucht werden. Wie zuvor erwähnt, werden nur 1 Euro- und 2-Euro-Münzen akzeptiert. Ist der maximale Kreditstand erreicht, werden auch diese Münzkanäle in der Emulation des Münzprüfers gesperrt. Erst wenn die Kredite durch Spielen wieder aufgebraucht sind, werden die Münzkanäle wieder freigegeben, und es können erneut Kredite aufgebucht werden.

Darüber hinaus wird auch die Auszahlung über die emulierten Auszahleinheiten getestet. Befinden sich aufgebuchte oder gewonnene Kredite auf dem Spielautomaten, dessen Auszahlung sich durch die jeweiligen Münzwertigkeiten der beiden Auszahleinheiten durchführen lässt, kann der *Payout*-Button betätigt werden. Die ausbezahlten Münzen werden folglich im Kommandozeilenfenster des Emulators visualisiert.



Abbildung 54: Spielplattform-Software und CLI des Emulators während der laufenden Emulation, Quelle: Eigene Darstellung.

6.2.2 Funktionstests auf der Zielplattform

Mit den ersten Ergebnissen der Tests auf der Entwicklungsplattform, gilt es in der nächsten Stufe die Emulator-Software auf die Zielplattform zur portieren. Dazu wird vornan der Prüfaufbau geringfügig adaptiert (siehe Abbildung 55). Anstelle einer der Schnittstellen-Konverter zur Entwicklungsplattform wird das ccTalk-Bussystem mit dem in Kapitel 4.2.2 erarbeiteten Erweiterungsboard des RPI verbunden. Die andere Verbindung zur Entwicklungsplattform wird hingegen für das Debuggen bzw. die Analyse der Datenkommunikation aufrechterhalten.

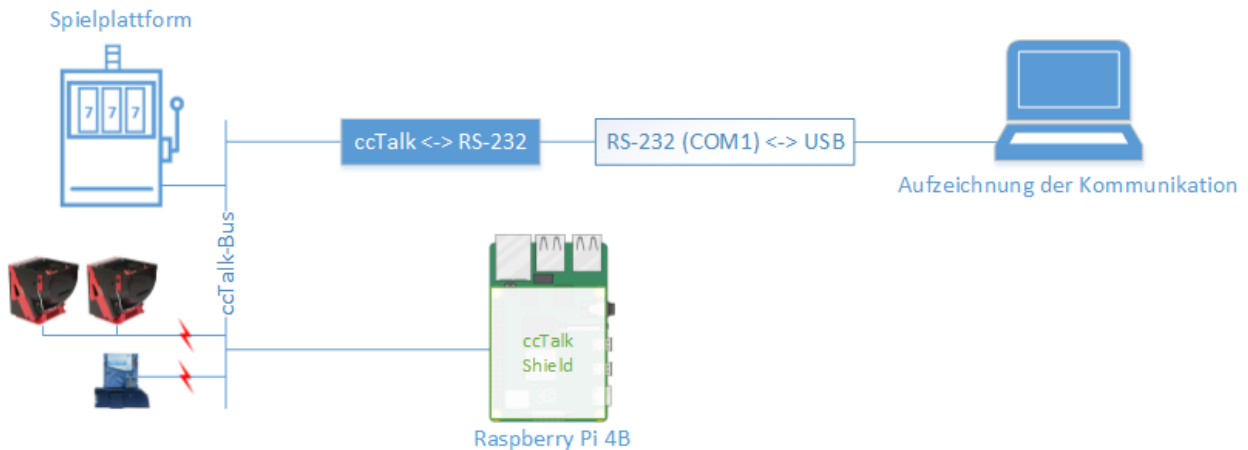


Abbildung 55: Prüfaufbau für den Funktionstest mit der Zielplattform, Quelle: Eigene Darstellung.

Nach der Realisierung des Prüfaufbaus wird die Emulator-Software auf der Entwicklungsplattform kompiliert. Dies geschieht unter Eingabe des Befehls `dotnet publish -r linux-arm --self-contained true` im Kommandozeilenfenster. Der Befehl `linux-arm` aus der Befehlskette definiert die Linux-basierte Zielplattform. Eine genaue Beschreibung der Befehlskette kann auf der Microsoft Webseite eingesehen werden. Anschließend wird die fertige Kompilation auf die Zielplattform kopiert, mit dem Befehl `chmod +x` die entsprechenden Zugriffsrechte für die Programmausführung gesetzt und mit `./ConsoleEmulator` ausgeführt. Die IDE auf der Entwicklungsplattform ermöglicht in Folge die Verbindung mit dem ausgeführten Prozess auf dem RPI mittels SSH-Verbindung zum Debuggen (siehe Abbildung 56).

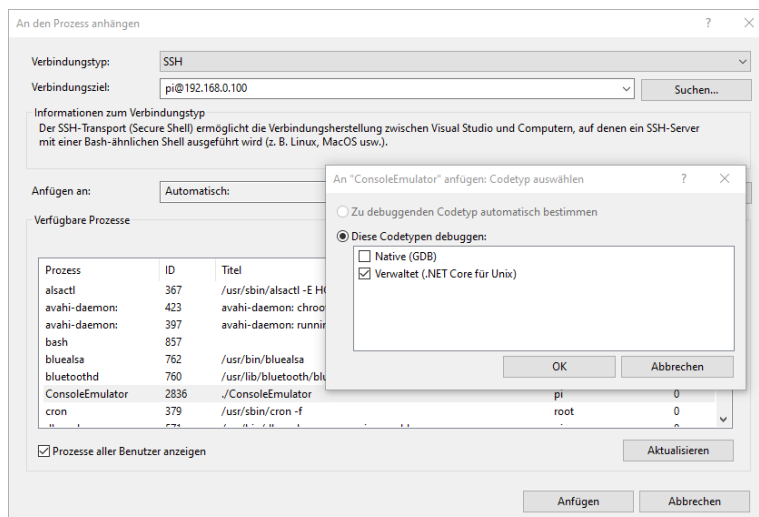


Abbildung 56: Anhängen an den ausgeführten Prozess des RPI, Quelle: Eigene Darstellung.

Um hoffentlich eine Übereinstimmung mit den durchwegs positiven Testresultaten aus den Kapiteln 6.1 und 6.2 zu erhalten, wird der RPI mit der darauf ausgeführten Kompilation der Emulator-Software an den Spielautomaten angebunden. Leider hat sich bereits direkt bei der Anbindung herausgestellt, dass keines der emulierten Geräte von der Spielesoftware erkannt wird. Darüber hinaus sind im CLI und in der Logdatei des Emulators immer wieder Exceptions mit Bezug auf die Checksummen-Berechnung sowie Polling-Sequenzen mit nicht unterstützten ccTalk-Headern gelistet.

Für die Fehler-Ursachen-Analyse wird anstelle des Spielautomaten nun wieder die Entwicklungsplattform an den Bus angeschlossen. Wie in Kapitel 6.1.3 beschrieben, können mit dem Terminal-Programm so einzelne Polling-Sequenzen mit dem Debugger analysiert werden. Diese Analysen ergeben, dass Abfragen bzw. Polls, deren Antwortpakete nur eine geringe Anzahl an zu übertragenden Datenbytes aufweisen, ohne jegliche Fehler funktionieren. Enthält ein Antwortpaket mehr als drei Datenbytes, dann tritt die erwähnte Checksummen Exception in der Emulator-Software auf. Wird dementsprechend ein und dieselbe Abfrage in Form einer Einzelprozessdurchschritt-Verarbeitung mit dem Debugger durchgeführt, treten keinerlei Fehler auf.

Auf Grund der genannten Fakten liegt es nahe, dass es beim Einlesen und Senden des seriellen Bytestreams zu zeitkritischen Abweichungen zwischen der Entwicklungsplattform und dem RPI kommt. Es sollten daher weitere Analysen in diese Richtung durchgeführt werden, um die Fehlerursache festzustellen und zu beseitigen.

6.3 Erkenntnisse aus den Tests

Die in diesem Kapitel durchgeführte Evaluierung des zu entwickelnden Emulators hat Aufschluss über dessen Verhalten im Einsatz gebracht. Die Evaluierung erfolgte hierfür in einem mehrphasigen Funktionstest, der von entwicklungsbegleitenden Tests bis hin zu Tests unter realen Einsatzbedingungen reichte. Aus diesen Tests wurden schließlich die folgenden Erkenntnisse gewonnen:

- Mit der Hilfe eines geeigneten Prüfaufbaus und geeigneter Zusatzprogramme lassen sich bereits in frühen Entwicklungsstadien zielgerichtete Funktionstests an der Software durchführen, ohne dass die endgültige Zielplattform benötigt wird. Die grundlegenden Softwareprozeduren, als auch alle implementierten ccTalk-Header konnten somit erfolgreich überprüft werden.
- Der positive Nutzen der entwicklungsbegleitenden Tests bestätigt sich im nachfolgenden Funktionstest der Software auf der Entwicklungsplattform, die an einen realen Spielautomaten angebunden wurde. Die vom Emulator emulierten Peripheriegeräte wurden von der Spielesoftware ohne Schwierigkeiten erkannt, und es war möglich Kredite aufzubuchen, als auch auszubezahlen.
- Die wertvollsten Erkenntnisse wurden allerdings aus der letzten Testphase mit der RPI Zielplattform gewonnen. Es hat sich herausgestellt, dass sich trotz dem plattformübergreifendem .NET Software-Framework gravierende Unterschiede in der Programmausführung ergeben können. Obwohl sich bei der Ausführung unter Windows keinerlei Auffälligkeiten gezeigt haben, kam es beim Empfangen und Senden des Bytestreams unter Linux zu Problemen, die vermutlich auf zeitliche Abhängigkeiten in Verbindung mit der seriellen Schnittstelle zurückzuführen sind.

7 ZUSAMMENFASSUNG UND AUSBLICK

Der letzte Abschnitt dieser Masterarbeit umfasst die Schlussbetrachtung, in der sowohl die wesentlichen Inhalte der Arbeit als auch die daraus resultierenden Ergebnisse kurz zusammengefasst werden. Nach anschließendem Fazit zu den Ergebnissen wird schließlich ein kurzer Ausblick in die Zukunft gewährt.

7.1 Zusammenfassung der Ergebnisse

Die Zielsetzung der vorliegenden Arbeit lag in der Systementwicklung eines Emulators, der die realen Komponenten des automatisierten Münzverarbeitungssystems in einem Spielautomaten emuliert. Den Benutzer*innen sollte zudem ermöglicht werden, Ein- und Auszahlungen von Münzen zu simulieren. Der Emulator soll vor allem die Software-Qualitätssicherung bei durchgeführten Softwaretestprozessen unterstützen

Zu Beginn der Arbeit wurden nach einem kurzen Einblick in relevante Märkte der Spieleindustrie alle notwendigen Informationen über den grundlegenden Aufbau und die Funktionsweise des Münzverarbeitungssystems zusammengetragen. Diese Informationen haben insbesondere Aufschluss über die wesentlichen Systemkomponenten und deren Funktionsweise gegeben. Im weiteren Verlauf wurden die grundlegenden Prinzipien und Methodiken des Requirements-Engineering erarbeitet und angewandt. Nach der Analysephase, in der schließlich detailreiche Untersuchungen der Spielautomaten-Software und des ccTalk-Protokolls, das zur Ansteuerung der Systemkomponenten eingesetzt wird, durchgeführt wurden, folgte zusammen mit den Stakeholdern die konkrete Definition und Abstimmung der Anforderungen an den Emulator.

In der darauffolgenden Hard- und Softwareentwicklungsphase wurden auf Basis der definierten Anforderungen erste Überlegungen in Bezug auf eine passende Zielplattform und eine geeignete Softwarearchitektur angestellt. Im Fokus dieser Überlegungen standen Faktoren wie Einarbeitungszeit, Integrierbarkeit, Portierbarkeit von Programmcode und Wartungsfreundlichkeit. Das Resultat ist ein RPI Einplatinencomputer und das neueste .NET 5 Software-Framework, welches plattformübergreifende Entwicklung in der Hochsprache C# und einfache Portierbarkeit zwischen unterschiedlichen Hardware-Plattformen gewährleistet. Zur Anbindung des RPIs an den ccTalk-Bus des Spielautomaten wurde ein Erweiterungsboard entworfen. In weiterer Folge wurde auf Basis des gewählten Software-Frameworks eine geeignete Architektur für die nachfolgende Implementierung der Emulator-Software entworfen. Das dafür definierte Entwurfsmuster dient zur Entkopplung der einzelnen Softwaremodule und minimiert die Abhängigkeiten untereinander, was zudem die Faktoren Erweiterbarkeit und Wartungsfreundlichkeit sicherstellt. Im nächsten Schritt erfolgte die Implementierung der zuvor entworfenen Softwarearchitektur zunächst auf der Entwicklungsplattform. Unter Zuhilfenahme von Prüfaufbauten und Zusatzprogrammen wurden im Zuge dessen immer wieder entwicklungsbegleitende Funktionstests einzelner Softwareroutinen und Methoden durchgeführt, um Fehler bereits während der Entwicklung zu identifizieren.

Letztendlich wurden erste Evaluierungen am entwickelten Prototyp des Emulators durchgeführt. Diese reichten von den bereits erwähnten entwicklungsbegleitenden Tests bis hin zu Funktionstests unter realen Einsatzbedingungen im Spielautomaten. Mit den letzteren Tests konnten die grundsätzlichen Funktionen aus der Anforderungsspezifikation des zu entwickelnden Systems im Einsatz bewiesen werden. Allerdings

wurden etwaige Abweichungen hinsichtlich der Funktionsweise bei Ausführung der Emulator-Software auf der Zielplattform festgestellt, weshalb es weiteren Analysen in Anschluss an die hier dokumentierte Arbeit bedarf.

7.2 Fazit

Mit Hilfe dieser Masterarbeit konnte schließlich ein erster funktionsfähiger Emulator Prototyp entwickelt werden, der die zuvor festgelegten Anforderungen aller Stakeholder erfüllt. Wie aus den praktischen Funktionstests hervorgeht, werden bei Ausführung des Emulators auf der Entwicklungsplattform keinerlei Fehlermeldungen von der Spielplattform-Software hervorgerufen. Von den Endnutzer*innen des Emulators können somit einfach Ein- und Auszahlungen am Spielautomaten, ohne jeglichen Einsatz echter Bezahlssystemkomponenten oder Münzen, vorgenommen werden. Da die Ausführung der Emulator-Software auf der RPI Zielplattform vorerst allerdings zu einigen Problemen geführt hat, wurde die Erkenntnis erlangt, dass im Zuge plattformübergreifender Entwicklung Funktionstests mit jeder Art von Zielplattform essenziell sind.

7.3 Ausblick

Die Entwicklung eines ersten funktionsfähigen Prototyps für das automatisierte Münzverarbeitungssystem in Spielautomaten konnte mit dieser Arbeit umgesetzt werden. Der Prototyp weist allerdings noch einige Fehler auf, die es in Anschluss an diese Arbeit zu beseitigen gilt. In weiterer Folge soll ein zweiter Prototyp in Form einer Kleinserie für die Software-Qualitätssicherung aufgelegt werden. Teil dieser Serie sollen sowohl ein fertiges Platinen-Layout für das Erweiterungsboard als auch ein kleines Display und ein kompaktes Gehäuse sein.

Langfristig gilt es die stetige Erweiterung und Verbesserung von Methoden, die zur Emulation der realen Systemkomponenten dienen, zu forcieren, um eine noch realitätsgetreuere Emulation sicherzustellen. Dazu zählen beispielsweise die Simulation von Fehlermeldungen des Münzprüfers oder zeitliche Verzögerungen bei Auszahlungen, die in der Realität durch mechanische Rotationsbewegungen der Drehscheibe bei den Münzauszahleinheiten resultieren. Darüber hinaus könnten Erweiterungen des Emulators um Funktionen von Banknotenprüfern oder Ticketprintern durch zusätzliche Module in der Software umgesetzt werden.

LITERATURVERZEICHNIS

Gedruckte Werke (6)

Gebhardt, Ihno; Grüsser-Sinopoli, Sabine Miriam (Hrsg.) (2008): *Glücksspiel in Deutschland: Ökonomie, Recht, Sucht*, De Gruyter Rechtswissenschaften Verlag, Berlin

Haberbosch, Birgit Friederike (Hrsg.) (1994): *Schöne alte Automaten. Waren-, Spiel- und Musikautomaten*, Battenberg Verlag, Augsburg

Harris, Julian (2012): *Gaming Law: Jurisdictional Comparisons*, First Edition, Sweet & Maxwell Verlag, U.K.

Rupp, Chris; die SOPHISTen (2014): *Requirements Engineering und -Management*, 6. Auflage, Carl Hanser Verlag, München

Pohl, Klaus; Rupp, Chris (2015): *Basiswissen Requirements Engineering*, 4. Auflage, dpunkt Verlag, Heidelberg

Price, Mark J.; (2019): *C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development*, Fourth Edition, Packt Publishing Ltd., U.K.

Online-Quellen (18)

International Game Technology (o.J.): *IGT® Logo*

https://www.igt.com/IGTSitecore/BrandAssets/images/LOGO_IGT_RGB_4COL-BLUE.png

[Stand: 24.02.2021]

Nähter, Ulrike (o.J.): *Zur Geschichte des Glücksspiels*

<https://www.uni-hohenheim.de/fileadmin/einrichtungen/gluecksspiel/Forschungsarbeiten/Naether.pdf>

[Stand: 18.09.2020]

Honeywell International Inc. (o.J.): *Application Note: Coin Acceptor*

<http://notes-application.abcelectronique.com/231/231-48796.pdf> [Stand: 24.02.2020]

Rieck, Manuel; Krüger Nils (2016): *Projektarbeit P4 „Die Münze“*

https://www.arte.tu-berlin.de/fileadmin/fg301/Archivierungsprojekt/P4-Dokumentation_Muenze__Krueger__Rieck.pdf [Stand: 02.09.2020]

Amtsblatt der Europäischen Union (Hrsg.) (2014): *VERORDNUNG (EU) Nr. 729/2014 DES RATES vom 24. Juni 2014 über die Stückelungen und technischen Merkmale der für den Umlauf bestimmten Euro-Münzen (Neufassung)*

<https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32014R0729&from=de> [Stand: 24.02.2021]

Crane Payment Innovations (2020): *CPI - One Global Brand*

<https://www.cranepi.com/en/about-us/brand-history> [Stand: 24.02.2021]

Mars Money Systems (o.J.): *Die elektronische Münzprüfung mit einem neuen Kontrollgerät von Mars Money Systems*

https://coinop.mally.eu/flyer/mars_ms130.pdf [Stand: 24.02.2021]

National Rejectors, Inc. GmbH (2007): *Elektronischer Münzprüfer G-13.mft ccTalk (ab Version /4) Bedienungsanleitung*

https://del-service.de/pdfxx/NRI/G_13_mft_cctalk_Handbuch.pdf [Stand: 24.02.2021]

SUZOHAPP (o.J.): *SEPARATORE DI MONETE*

https://oem.suzohapp.com/wp-content/uploads/2019/09/Manuale_Separatori_IT.pdf [Stand: 24.02.2021]

AZKOYEN Medios de Pago, S.A (2011): *HOPPER U-II*

<https://www.casino-software.de/download/hopper-azkoyen-u2-manual.pdf> [Stand: 24.02.2021]

RIS der Republik Österreich (2021): *Gesamte Rechtsvorschrift für Glücksspielgesetz*

<https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10004611>
[Stand: 24.02.2021]

Raspberry Pi Foundation (o.J.): *GPIO-Pinout-Diagram-2*

<https://www.raspberrypi.org/documentation/usage/gpio/> [Stand: 24.02.2021].

Microsoft Corporation (2019): *.NET erneut vereint: Microsofts Pläne für.NET 5*

<https://docs.microsoft.com/de-de/archive/msdn-magazine/2019/july/csharp-net-reunified-microsoft%E2%80%99s-plans-for-net-5> [Stand: 24.02.2021]

IT-Visions; Dr. Schwichtenberg, Holger (Hrsg.) (2019): *Was ist NuGet Library Package Manager?*

<https://www.dotnetframework.de/%7BAD7C7CFF6-31A7-48E5-AB2B-CD3904BE431B%7D.aspx> [Stand: 24.02.2021]

Microsoft Corporation (2019): *Abbildung NuGet Paketfluss*

<https://docs.microsoft.com/de-de/nuget/media/nuget-roles.png> [Stand: 24.02.2021]

Mozilla Developer Network (2020): *Arbeiten mit JSON*

<https://developer.mozilla.org/de/docs/Learn/JavaScript/Objects/JSON> [Stand: 24.02.2021]

json.org (o.J.): *Einführung in JSON*

<https://www.json.org/json-de.html> [Stand: 24.02.2021]

Microsoft Corporation (o.J.): *System.IO.Ports Namespace*

<https://docs.microsoft.com/de-de/dotnet/api/system.io.ports?view=dotnet-plat-ext-5.0> [Stand: 24.02.2021]

Wissenschaftliche Arbeiten (2)

Bongardt, Johannkarl (1983): *Anbieter und Nachfragerbeziehungen im Münzhandel – Eine markttheoretische Analyse*, Dissertation zur Erlangung des Grades eines Doktors der Staatswissenschaften, Johannes-Gutenberg-Universität, Mainz

Adameck, Markus (2004): *Entwicklung von Methoden der Prägebildaufnahme zur Merkmalsextraktion und Klassifikation von Münzen*, Paderborn

Normen und Standards (4)

Crane Payment Solutions (Hrsg.) (2013): *ccTalk Serial Communication Protocol - Generic Specification - Issue 4.7 - Part1*

Crane Payment Solutions (Hrsg.) (2013): *ccTalk Serial Communication Protocol - Generic Specification - Issue 4.7 – Part2*

Crane Payment Solutions (Hrsg.) (2013): *ccTalk Serial Communication Protocol - Generic Specification - Issue 4.7 – Part3*

Crane Payment Solutions (Hrsg.) (2013): *ccTalk Serial Communication Protocol - Generic Specification - Issue 4.7 – Part4*

ABBILDUNGSVERZEICHNIS

Abbildung 1: IGT Logo, Quelle: International Game Technology (o.J.), Online-Quelle [24.02.2021].	1
Abbildung 2: Automatisiertes Münzverarbeitungssystem im Spielautomatenmodell delite™, Quelle: Eigene Darstellung.	6
Abbildung 3: Einzelkomponenten des Münzverarbeitungssystems, Quelle: Eigene Darstellung.	7
Abbildung 4: Münzpfad bei Einwurf von Falschgeld, Quelle: Eigene Darstellung.	8
Abbildung 5: Münzpfad bei Einwurf einer 2 Euro-Münze, Quelle: Eigene Darstellung.	8
Abbildung 6: Münzpfad bei Einwurf einer 20 Cent-Münze, Quelle: Eigene Darstellung.	9
Abbildung 7: Münzpfad bei Einwurf von 1 Euro-, 50 Cent- oder 10 Cent-Münzen, Quelle: Eigene Darstellung.	9
Abbildung 8: Auszahlung von 2 Euro-Münzen, Quelle: Eigene Darstellung.	10
Abbildung 9: Auszahlung von 20 Cent-Münzen, Quelle: Eigene Darstellung.	10
Abbildung 10: Aufbau des Münzprüfers SUZOHAPP-Comestero RM5 Modell G, Quelle: Eigene Darstellung.	14
Abbildung 11: SUZOHAPP-Comestero Münzprüfer und Münzsortierer, Quelle: SUZOHAPP (o.J.), Online-Quelle [24.02.2021].	16
Abbildung 12: Weichenstellungen des SUZOHAPP-Comestero SPS31LCC3 Sortierers (Rückseite), Quelle: Eigene Darstellung.	16
Abbildung 13: Hopper U-II mit kleinster Münzschale, Quelle: AZKOYEN Medios de Pago, S.A (2011), Online-Quelle [24.02.2021].	17
Abbildung 14: Münzauszahleinheiten mit demontiertem Trichter, Quelle: Eigene Darstellung.	18
Abbildung 15: Haupttätigkeiten des Requirements-Engineerings, Quelle: Rupp/SOPHISTen (2014), S. 14.	20
Abbildung 16: System- und Kontextgrenze eines Systems, Quelle: In Anlehnung an Pohl/Rupp (2015), S. 15 f.	20
Abbildung 17: Systemkontext des zu entwickelnden Emulators, Quelle: Eigene Darstellung.	24
Abbildung 18: Empfohlene ccTalk-Standard-Schnittstelle, Quelle: Crane Payment Solutions (2013), Part 3 S. 82 (leicht modifiziert).	27
Abbildung 19: ccTalk-Protokollstapel, Quelle: Crane Payment Solutions (2013), Part 1 S. 26.	29
Abbildung 20: Auszug der verwendeten ccTalk-Header aus dem Foundation Quellcode, Quelle: Eigene Darstellung.	34
Abbildung 21: Raspberry Pi 4 B Schnittstellen, Quelle: Raspberry Pi Foundation (o.J.), Online-Quelle [24.02.2021], (leicht modifiziert).	43

Abbildung 22: Schaltplan des RPI-Erweiterungsboards für die ccTalk-Anbindung, Quelle: Eigene Darstellung.	45
Abbildung 23: Prototyp des RPI mit dem ccTalk-Erweiterungsboard, Quelle: Eigene Darstellung.	46
Abbildung 24: Die vereinheitlichte .NET 5-Architektur, Quelle: Microsoft Corporation (2019), Online-Quelle [24.02.2021].	48
Abbildung 25: Schematische Darstellung des Event-Aggregator-Entwurfsmusters, Quelle: Eigene Darstellung.	51
Abbildung 26: NuGet Paketfluss zwischen Ersteller, Host und Endnutzer, Quelle: Microsoft Corporation (2019), Online-Quelle [24.02.2021].	52
Abbildung 27: Übersicht der Architektur des Emulators, Quelle: Eigene Darstellung.	55
Abbildung 28: Schematischer Aufbau des Event Aggregators für die Emulator-Software, Quelle: Eigene Darstellung.	56
Abbildung 29: Klassendiagramm des Event Aggregators, Quelle: Eigene Darstellung.	57
Abbildung 30: Klassendiagramm des Konfigurations-Managers, Quelle: Eigene Darstellung.	59
Abbildung 31: Gerätekonfiguration zur Laufzeit, Quelle: Eigene Darstellung.	61
Abbildung 32: Serielle Schnittstellenkonfiguration zur Laufzeit, Quelle: Eigene Darstellung.	61
Abbildung 33: Klassendiagramm des seriellen Bustreibers, Quelle: Eigene Darstellung.	62
Abbildung 34: Klassendiagramm des Geräte-Managers, Quelle: Eigene Darstellung.	65
Abbildung 35: Projektmappen-Verzeichnis der ccTalk-Bibliothek, Quelle: Eigene Darstellung.	67
Abbildung 36: Klassendiagramm der ccTalk-Device-Klassen, Quelle: Eigene Darstellung.	68
Abbildung 37: Befülltes Dictionary <i>SupportedMessages</i> zur Laufzeit, Quelle: Eigene Darstellung.	69
Abbildung 38: Ereignisdatenspeicher <i>creditInsertEvents</i> zur Laufzeit, Quelle: Eigene Darstellung.	70
Abbildung 39: Klassendiagramm des Konsolen-Managers, Quelle: Eigene Darstellung.	72
Abbildung 40: Münzeinwurf im Kommandozeilenfenster, Quelle: Eigene Darstellung.	73
Abbildung 41: Münzauszahlung im Kommandozeilenfenster, Quelle: Eigene Darstellung.	74
Abbildung 42: Logdatei im Projektverzeichnis, Quelle: Eigene Darstellung.	75
Abbildung 43: Aktivitätsdiagramm für die Events <i>ReceivedCcTalkCommand</i> und <i>SendCcTalkCommand</i> , Quelle: Eigene Darstellung.	76
Abbildung 44: Aktivitätsdiagramm für das <i>InsertedMoneyEvent</i> , Quelle: Eigene Darstellung.	77
Abbildung 45: Aktivitätsdiagramm für das <i>PayoutMoneyEvent</i> , Quelle: Eigene Darstellung.	78
Abbildung 46: Prüfaufbau für entwicklungsbegleitende Tests mit der Entwicklungsplattform, Quelle: Eigene Darstellung.	80
Abbildung 47: Benutzeroberfläche der Software SeciCCtalk, Quelle: Eigene Darstellung.	81

Abbildung 48: Benutzeroberfläche der Software HTerm, Quelle: Eigene Darstellung..... 81

Abbildung 49: Senden eines Datenpaketes in der Software HTerm , Quelle: Eigene Darstellung..... 82

Abbildung 50: Empfangene Datenpakete in der Software *HTerm*, Quelle: Eigene Darstellung. 82

Abbildung 51: Prüfaufbau für den Funktionstest mit der Entwicklungsplattform, Quelle: Eigene Darstellung. 83

Abbildung 52: Spielautomaten Fehlermeldungen beim Anbinden des Emulators, Quelle: Eigene Darstellung. 84

Abbildung 53: Installierte Bezahlssystemkomponenten vor und nach dem Anbinden des Emulators, Quelle: Eigene Darstellung. 84

Abbildung 54: Spielplattform-Software und CLI des Emulators während der laufenden Emulation, Quelle: Eigene Darstellung. 85

Abbildung 55: Prüfaufbau für den Funktionstest mit der Zielplattform, Quelle: Eigene Darstellung. 86

Abbildung 56: Anhängen an den ausgeführten Prozess des RPI, Quelle: Eigene Darstellung. 86

TABELLENVERZEICHNIS

Tabelle 1: Technische Merkmale der Euro-Münzen, Quelle: In Anlehnung an VERORDNUNG (EU) Nr. 729/2014 (2014), Online-Quelle [24.02.2021].	13
Tabelle 2: Liste der unterstützten Münzprüfer, Quelle: Eigene Darstellung.	25
Tabelle 3: Liste der unterstützten Münzauszahlungen, Quelle: Eigene Darstellung.	26
Tabelle 4: Datenpaket mit zu übertragenden Nutzdaten, Quelle: Eigene Darstellung.	30
Tabelle 5: Datenpaket ohne Nutzdaten, Quelle: Eigene Darstellung.	30
Tabelle 6: Definierte Adressbereiche aus der Spezifikation, Quelle: In Anlehnung an Crane Payment Solutions (2013), Part 3 S. 61.	30
Tabelle 7: Einfache Checksummen-Berechnung, Quelle: Eigene Darstellung.	31
Tabelle 8: Broadcast Adress-Poll (Header 253), Quelle: Eigene Darstellung.	32
Tabelle 9: Reset-Poll (Header 1), Quelle: Eigene Darstellung.	32
Tabelle 10: Abfrage der Hersteller ID (Header 246), Quelle: Eigene Darstellung.	33
Tabelle 11: Aktivierung der Münzauszahlung (Header 164), Quelle: Eigene Darstellung.	33
Tabelle 12: Auflistung der allgemeinen ccTalk-Header, Quelle: Eigene Darstellung.	36
Tabelle 13: Auflistung der spezifischen ccTalk-Header für Münzprüfer, Quelle: Eigene Darstellung.	37
Tabelle 14: Auflistung der spezifischen ccTalk-Header für Münzauszahlungen, Quelle: Eigene Darstellung.	38
Tabelle 15: Funktionale Anforderungen, Quelle: Eigene Darstellung.	40
Tabelle 16: Qualitätsanforderungen, Quelle: Eigene Darstellung.	41
Tabelle 17: Randbedingungen, Quelle: Eigene Darstellung.	41
Tabelle 18: ccTalk-Standard-Steckverbindungen, Quelle: In Anlehnung an Crane Payment Solutions (2013), Part 1 S. 18 ff.	44

QUELLTEXTVERZEICHNIS

Quelltext 1: Codeausschnitt der Methode <i>InitializeCoinEmulator</i> , Quelle: Eigene Darstellung.....	56
Quelltext 2: Deklaration der Events, Quelle: Eigene Darstellung.....	57
Quelltext 3: CommunicationSettings-Objekt in der <i>appsettings.json</i> Datei, Quelle: Eigene Darstellung...	58
Quelltext 4: <i>DeviceConfigurations</i> in der <i>appsettings.json</i> Datei, Quelle: Eigene Darstellung.	59
Quelltext 5: Implementierung der Klasse <i>DeviceConfiguration</i> , Quelle: Eigen Darstellung.....	60
Quelltext 6: Ausdruck zum Erzeugen des Objekts <i>deviceSettings</i> , Quelle: Eigene Darstellung.	60
Quelltext 7: Implementierung der Klasse <i>CommunicationSettings</i> , Quelle: Eigene Darstellung.	61
Quelltext 8: Ausdruck zum Erzeugen des Objekts <i>communcationSettings</i> , Quelle: Eigene Darstellung. .	61
Quelltext 9: Codeausschnitt der Methode <i>ReadSerialPortData</i> , Quelle: Eigene Darstellung.	63
Quelltext 10: Implementierung der Methode <i>OnEvent(SendCcTalkCommand)</i> ,Quelle: Eigene Darstellung.	64
Quelltext 11: Implementierung der Methode <i>ConfigureDevices</i> , Quelle: Eigene Darstellung.	65
Quelltext 12: Implementierung der Methode <i>OnEvent(ReceivedCcTalkCommand)</i> , Quelle: Eigene Darstellung.	66
Quelltext 13: Implementierung der Methode <i>OnEvent(InsertedMoneyEvent)</i> , Quelle: Eigene Darstellung.	67
Quelltext 14: Implementierung der Methode <i>HandleCcTalkMessage</i> , Quelle: Eigene Darstellung.....	69
Quelltext 15: Implementierung der Methode <i>RequestManufacturerId</i> , Quelle: Eigene Darstellung.	70
Quelltext 16: Implementierung der Methode <i>DispenseHopperCoins</i> , Quelle: Eigene Darstellung.....	71
Quelltext 17: Codeausschnitt der Methode <i>StartInputHandler</i> , Quelle: Eigene Darstellung.....	73
Quelltext 18: Implementierung der Methode <i>OnEvent(PayoutMoneyEvent)</i> , Quelle: Eigene Darstellung.	74
Quelltext 19: Codeausschnitt zur Konfiguration von NLog, Quelle: Eigene Darstellung.	75

ABKÜRZUNGSVERZEICHNIS

ACK	Acknowledge
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AWP	Amusement With Prize
BCD	Binary Coded Decimal
BCL	Base Class Library
CLI	Command Line Interface
CLT	Common Language Runtime
COTS	Commercial Off-the-Shelf
FAQ	Frequently Asked Questions
FTP	File transfer Protocol
GPIO	General Purpose Input/Output
GSpG	Glücksspielgesetz
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NAK	Not Acknowledge
RS-232	Recommended Standard 232
SBC	Single-Board Computer
SoC	System-on-a-Chip
SoM	System-on-a-Module
SSH	Secure Shell
UART	Universal Asynchronous Receiver Transmitter
VLT	Video Lottery Terminals