

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät Elektro- und Informationstechnik
Studiengang Elektrotechnik – Elektro- und Informationstechnik

Masterthesis

VON

David Erb

Referent:	Prof. Dr. Marianne Katz
Korreferent:	Prof. Dr. rer. nat. Klaus Wolfrum
Arbeitsplatz:	
Betreuer am Arbeitsplatz:	
Zeitraum:	18.04.2017 – 18.10.2017

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Masterthesis ohne unzulässige fremde Hilfe selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Stuttgart, den 23. August 2017

Inhaltsverzeichnis

Inhaltsverzeichnis	II
1 Einleitung	1
1.1 Motivaton	1
1.2 Aufgabenstellung	1
2 Grundlagen	2
2.1 GIGABOX	2
2.1.1 Überblick	2
2.1.2 Einsatz	2
2.1.3 Hardware der GIGABOX FD	4
2.1.4 Software der GIGABOX FD	5
2.2 Pawn	7
2.3 Softwareentwicklungsprozess	8
2.3.1 V-Modell	8
2.3.2 Qualitätskriterien an Software	8
2.4 Entwicklungsframework	11
2.4.1 .NET Framework	11
2.4.2 Windows Presentation Framework(WPF)	12
3 Anforderungsanalyse	14
3.1 Übersicht	14

3.2	Ermittlung und Bewertung von funktionalen Anforderungen . . .	16
3.2.1	Erstellung von Projekten mit Baumstruktur	16
3.2.2	Einstellung der Sprache	16
3.2.3	Texteditor zur Erstellung von PAWN-Skripten	17
3.2.4	Konfigurieren einer GIGABOX ohne PAWN-Kenntnisse .	20
3.2.5	Verbundene GIGABOXEN auflisten	20
3.2.6	PAWN-Compiler	20
3.2.7	Beschreiben/Auslesen des Flash-Speichers der GIGABOX	21
3.2.8	Kommunikation mit einer GIGABOX	21
3.2.9	Kommunikation mit dem Benutzer	22
3.2.10	Steuern und Beobachten von Ein-/Ausgängen und Timern	22
3.2.11	Debugging	22
3.2.12	Bereitstellung von Dokumentationen	23
3.2.13	Individualisierbare Oberfläche	23
3.3	Ermittlung von nichtfunktionalen Anforderungen	24
4	Ist-Zustand	25
4.1	Überblick	25
4.2	Elemente des configurAIDER	25
4.2.1	Rahmenfenster	25
4.2.2	Fenster „Verfügbare Geräte“	25
4.2.3	Fenster „Konsole“	27
4.2.4	Fenster „Programmlog“	28
4.2.5	Fenster „scriptEDITOR“	29
4.2.6	Fenster „multiEDITOR“	30
4.3	Funktionalität	32
4.3.1	Einstellung der Sprache	35
4.3.2	Bereitstellung von Dokumentation zur GIGABOX, con- figurAIDER und PAWN	35

4.3.3	Verbindung zu einer GIGABOX herstellen	35
4.3.4	PAWN-Code kompilieren	36
4.3.5	PAWN-Code entwickeln	36
4.3.6	Übertragen von Skript-Applikationen	37
4.3.7	Kommunikation mit verbundener GIGABOX	38
4.3.8	Ausgabe von Ereignismeldungen	38
4.3.9	Ergebnis	38
5	Evaluation	39
5.1	Funktionalität und Usability des Ist-Zustands	39
5.1.1	Einstellung der Sprache	39
5.1.2	Bereitstellung von Dokumentation zur GIGABOX, configurAIDER und PAWN	39
5.1.3	Verbindung zu einer GIGABOX herstellen	40
5.1.4	PAWN-Code kompilieren	40
5.1.5	PAWN-Code entwickeln	40
5.1.6	Übertragen von Skript-Applikationen	41
5.1.7	Kommunikation mit verbundener GIGABOX	42
5.1.8	Ausgabe von Ereignismeldungen des configurAIDERS	42
5.2	Differenz zwischen Anforderungen und Ist-Zustand	42
5.3	Quellcode des configurAIDERS	47
5.4	Ergebnis	47
6	Konzeption	48
6.1	Übersicht	48
6.2	Erweiterung des bestehenden configurAIDERS um neue Features	48
6.3	Neuentwicklung des configurAIDERS	49
6.4	Ergebnis	50

7 Design	51
7.1 Entwurf der Benutzeroberfläche	51
7.2 Entwurf der Softwarearchitektur	51
7.2.1 Entwurf der Kontextabgrenzung	52
7.2.2 Entwurf der Laufzeitsicht	53
8 Realisierung	55
8.1 Andockbare Fenster	55
8.2 Text Editor	55
8.3 Projektexplorer	55
8.4 Konsole	56
8.5 Kommunikation mit GIGABOX	56
9 Fazit und Ausblick	57
Abbildungsverzeichnis	58
Tabellenverzeichnis	59
Literaturverzeichnis	60

Kapitel 1

Einleitung

1.1 Motivaton

configurAIDER wurde im Rahmen von verschiedenen studentischen Arbeiten erstellt. Aktuell ergeben sich folgende Probleme:

- nicht testbar, da das Sollverhalten nicht immer bekannt ist
- nicht wartbar, da sich keiner im Code auskennt und keine saubere Dokumentation vorhanden ist
- schwer erweiterbar, da sich keiner im Code auskennt und keine saubere Dokumentation vorhanden ist

-> Kommt erst später rein

Einführung in das Thema...

1.2 Aufgabenstellung

Kapitel 2

Grundlagen

2.1 GIGABOX

2.1.1 Überblick

GIGABOX ist eine Produktfamilie von Steuergeräten der Firma GIGATRONIK. GIGABOX-Steuergeräte werden hauptsächlich in der Fahrzeugentwicklung eingesetzt, um prototypische Kommunikationslösungen im Fahrzeug schnell realisieren zu können. Beispielhafte Anwendungen sind der Einsatz als Gateway und die Ansteuerung von Treiber-ICs im Automobil.

GIGABOX-Steuergeräte werden in unterschiedlicher Hardwareausstattung und Funktionalität angeboten. Typischerweise verfügen sie über diverse Kommunikationsschnittstellen die im Automotive-Bereich benötigt werden wie CAN und LIN. Außerdem stehen digitale und analoge Ein- bzw. Ausgänge zur Verfügung. Das neueste Modell der GIGABOX, die GIGABOX FD, befindet sich aktuell noch in der Entwicklung. Im Folgenden wird ausschließlich auf die GIGABOX FD näher eingegangen, da nur diese für die vorliegende Arbeit von Bedeutung ist.

2.1.2 Anwendungen

Das Use-Case-Diagramm in Abbildung 2.1 zeigt drei typische Einsatzszenarien der GIGABOX.

1. Einsatz als Gateway zur Modifikation von Busbotschaften

Beispiel:

Ein Fahrzeughersteller entwickelt das Nachfolgemodell einer PKW-Modellreihe. Einige Busbotschaften, die die Steuergeräte untereinander austauschen, wurden für das neue Modell verändert oder wurden neu hinzugefügt. Der Großteil der Busbotschaften sind allerdings identisch wie beim alten Modell. Die Funktion einer neu entwickelten Klimaanlage soll nun im realen neuen Fahrzeugmodell überprüft werden. Allerdings ist das neue Modell noch nicht so weit entwickelt, dass es eine sinnvolle Testumgebung darstellen kann. Die neue Klimaanlage wird deshalb im alten Fahrzeugmodell getestet. Die für das Klimasteuergerät relevanten Busbotschaften werden dann mithilfe einer GIGABOX so manipuliert, dass sie den Botschaften entsprechen, die im neuen Fahrzeugmodell am Klimasteuergerät ankommen.

2. Funktionalität des Fahrzeugs erweitern

- Rapid Prototyping: Eine neue Funktion soll möglichst schnell und unkompliziert getestet werden

Beispiel:

Blinker-LEDs sollen nicht mehr gleichzeitig angesteuert werden, sondern nacheinander von innen nach außen mit kurzer zeitlicher Verzögerung.

- Serieneinsatz: Eine neue zusätzliche Funktion soll realisiert werden bei Fahrzeugumbau

Beispiel:

Eine Mercedes E-Klasse soll zu einem Leichenwagen umgebaut werden, der elektrisch verstellbare Fensterjalousien besitzt. Mithilfe einer GIGABOX können die elektrisch verstellbaren Fensterjalousien realisiert werden.

3. Restbussimulation:

Eine GIGABOX kann Steuergeräte innerhalb eines Bussystems simulieren. Steuergeräte können damit getestet werden, ohne das komplette Bussystem aufbauen zu müssen.

Restbussimulationen können auch mit PC-Tools wie zum Beispiel „CANoe“ von Vector Informatik GmbH und passendem Bus-Interface durchgeführt werden. Die Kosten für Softwarelizenzen und benötigter Hardware übersteigen allerdings die Kosten einer konfigurierten GIGABOX und bedingen einen komplizierteren Systemaufbau.

Quelle: <http://www.samtec.de/hauptmenu/loesungen-fuer/restbussimulation.html>

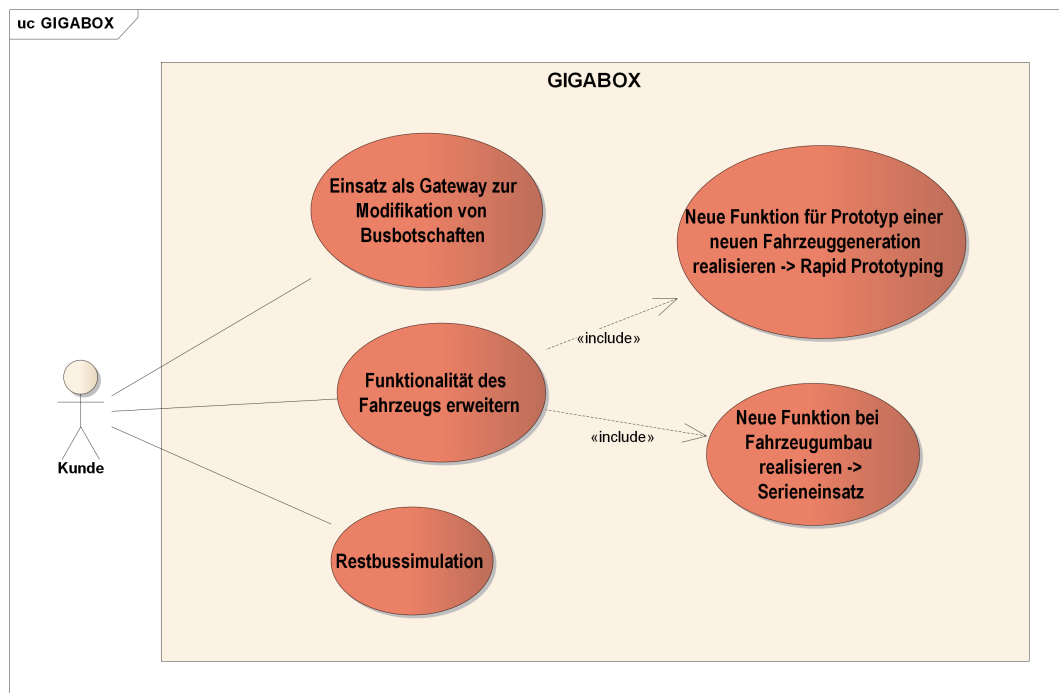


Abbildung 2.1 – Use-Case-Diagramm GIGABOX

2.1.3 Hardware der GIGABOX FD

Die GIGABOX FD ist modular aufgebaut bestehend aus einer Basisplatine und diversen Applikationsplatinen, mit denen zusätzliche Funktionalitäten zur Verfügung gestellt werden können. Auf der Basisplatine befindet sich ein Spannungsregler, ein 32-bit Mikrocontroller ARM Cortex M4 STM32F427IIT6, Kommunikationsschnittstellen sowie analoge und digitale Ein- und Ausgänge. Der Spannungsregler wird eingesetzt, um die Batteriespannung des Fahrzeugs auf die Versorgungsspannung des Mikrocontrollers herunterzuregeln.

Der Mikrocontroller wird mit einer Taktfrequenz von 168MHz betrieben und stellt 2MB Flash Speicher sowie 192 kB SRAM + 64 kB core coupled memory (CCM) zur Verfügung.

Auf der GIGABOX FD sind als Kommunikationsschnittstellen 2 CAN-, 2 CAN FD- und 2 LIN-Kanäle implementiert. Zusätzlich wird eine USB-Schnittstelle bereitgestellt, die mit einem UART-to-USB-Wandler realisiert wird. Es sind 4 analoge Eingänge vorhanden um Sensorwerte einzulesen wie z. B. von einem Temperatursensor. Mit den 8 digitalen Eingängen können z. B. Schalterzustände eingelesen werden. Als Ausgänge stehen 8 Halbbrückenausgänge und 2 Highsideausgänge zur Verfügung (Wann werden Halbbrücken- wann Highside

benutzt?).

Einen Überblick über die eingesetzte Hardware bietet Abbildung 2.2 (Von Shruti, Abb neu erstellen wegen schlechter Bildqualität).

Der Einsatz von einer Applikationsplatine ermöglicht die Funktionserweiterung der GIGABOX FD um beispielsweise zusätzliche Kommunikationsschnittstellen wie Bluetooth bereitzustellen.

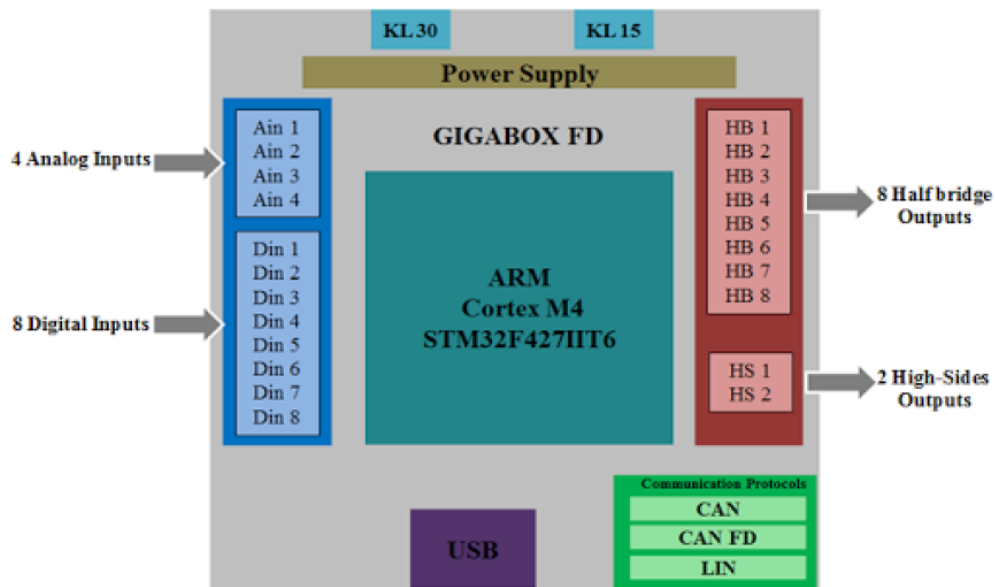


Abbildung 2.2 – Hardwareüberblick GIGABOX FD

2.1.4 Software der GIGABOX FD

Abbildung 2.3 gibt einen Überblick über die Softwarearchitektur der GIGABOX FD.

Der Hardware Abstraction Layer (HAL) ist eine Softwareschicht, die übergeordnete Schichten von der Hardware abstrahiert. Übergeordnete Schichten sind damit unabhängig von der eingesetzten Hardware. Dies bietet den Vorteil, dass bei Hardwareänderungen nur der HAL angepasst werden muss, nicht aber die übergeordneten Schichten. Auf dem HAL sind die Treiber des Mikrocontrollers und der zugehörigen Peripherie implementiert. Die Treiber stellen ein Application Programming Interface (API) bereit, die Zugriff auf den Speicher oder die Peripherie des Mikrocontrollers wie z.B. Analog-Digital-Converter (ADC) ermöglicht.

Vom Middleware Layer aus kann über die API des Hardware Abstraction Layers auf die Hardware zugegriffen werden. Auf dieser Schicht ist ein USB-HID Treiber implementiert, der benötigt wird, damit die GIGABOX FD bei Verbindung mit einem Computer über USB als Human Interface Device (HID) erkannt wird. Außerdem befindet sich dort das Transportprotokoll ISO-TP für CAN-Bus. Das Protokoll wird benötigt, um Botschaften zu verschicken, die die maximale Nutzdatengröße von 8 Byte eines CAN-Frames überschreiten. Um die Pawn virtuelle Maschine (VM) an den HAL anzubinden, befindet sich auf dem Middleware Layer zusätzlich die PAWN Driver Abstraction.

Auf dem Application Layer läuft das Betriebssystem FreeRTOS, ein Echtzeitbetriebssystem für eingebettete Systeme. Innerhalb von FreeRTOS ist die Pawn VM, der Bootloader und eine Konsole implementiert. In der Pawn-Umgebung wird eine Pawn Skriptapplikation ausgeführt, die vom Endanwender auf den Flash-Speicher der GIGABOX FD aufgespielt werden kann. Das Skript ermöglicht die Ansteuerung bzw. Abfrage der Kommunikationsschnittstellen sowie das Steuern bzw. Beobachten der digitalen/analogen Ein-/Ausgänge der GIGABOX FD.

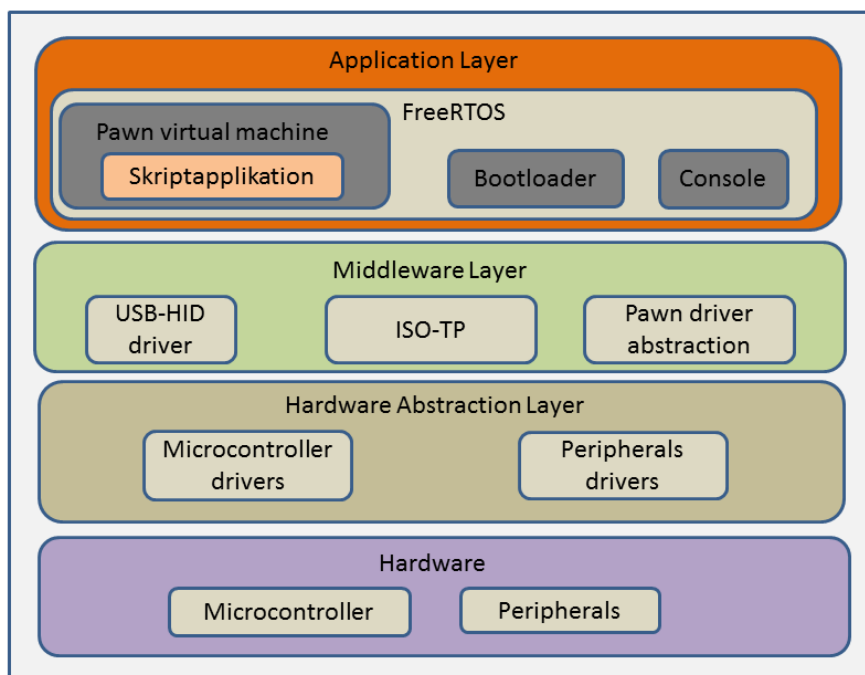


Abbildung 2.3 – Hardwareüberblick GIGABOX FD

Beschreibung Funktionsweise GIGABOX aus Softwaresicht. Skript-Applikation und Bootapplikation. Virtuelle Maschine läuft in Skript-Applikation. Auf virtueller Maschine wird Bytecode ausgeführt. Der Bytecode wurde vom PAWN-Compiler (in configuraIDER) aus einer Skriptdatei erzeugt. Quelle: Bachelorarbeit Christian Eissler Abschn. 4.1 (**Arbeit über GIGABOX FD -> Funktionsweise bei den alten GIGABOXEN gleich?**)

Eventgetriebene Programmierung -> bei Eintreffen einer CAN-Botschaft wird Event ausgelöst

2.2 Pawn

Pawn ist eine Skriptsprache mit einer C ähnlichen Syntax sowie eine zugehörige virtuelle Maschine, auf der kompilierter Pawn Sourcecode ausgeführt werden kann. Das PAWN-Paket, das Compiler und virtuelle Maschine beinhaltet, ist kostenlos verfügbar und wurde unter der Apache Lizenz 2.0 veröffentlicht. Die Sprache wurde abgeleitet von der Sprache „Small-C“, die wiederum eine Teilmenge von C darstellt und für Mikrocontroller entwickelt wurde. Pawn wurde erstmalig 1998 öffentlich zugänglich gemacht, damals noch unter dem Namen „SMALL“ und wird seitdem regelmäßig weiterentwickelt.

Pawn Sourcecode kann mithilfe des PAWN-Compilers in Bytecode kompiliert werden. Der Bytecode wird anschließend auf der zugehörigen virtuellen Maschine interpretiert. Dies bedeutet, dass für jede Bytecode-Anweisung definierte Funktionen aus einer C-Klassenbibliothek aufgerufen werden. Die Ausführung von Bytecode auf einer virtuellen Maschine bietet im Vergleich zur Ausführung von nativem Code auf der Zielhardware den Vorteil, dass der Code unabhängig von der eingesetzten Hardware lauffähig ist. Der Code kann folglich plattformunabhängig eingesetzt werden. Die virtuelle Maschine muss dagegen an die eingesetzte Plattform angepasst werden. Dieser Vorteil muss allerdings mit Geschwindigkeitseinbußen bei der Programmausführung bezahlt werden. Bei der Entwicklung von PAWN wurde besonders auf Einfachheit, Schnelligkeit und Robustheit Wert gelegt. Es wurde entwickelt zur Anwendung auf Mikrocontroller mit geringer Rechenleistung und Speicherkapazität. Die Sprache benötigt deshalb wenig Overhead.

[1], [2]

Siehe Pawn_Language_Guide.pdf, Foreword

Pawn_Implementation_Guide.pdf, The Compiler, The abstract machine

2.3 Softwareentwicklungsprozess

2.3.1 V-Modell

2.3.2 Qualitätskriterien an Software

Um Softwarequalität definierbar zu machen, wurden diverse Software-Qualitätsmodelle entwickelt. Die ISO/IEC 25010 definiert unter anderem das Product Quality Model. Dieses Modell definiert acht Qualitätsmerkmale für Software, die im folgenden erläutert werden:

- *Functional Suitability:*

„Das Qualitätsmerkmal der funktionellen Eignung fordert, dass ein Software-System die von ihm erwartete Funktionalität in angemessener und konkreter Art und Weise und unter Berücksichtigung der festgelegten Randbedingungen und Eigenschaften umsetzt. Dabei geht es um Fragestellungen nach der Angemessenheit der Funktionalitäten für den vorgegebenen Einsatzbereich des Systems.“ [3]

- *Performance Efficiency*

„Das Merkmal der Performanz und Effizienz betrachtet das Leistungsniveau einer Anwendung in Abhängigkeit der eingesetzten Betriebsmittel und der geforderten Bedingungen. Dabei geht es insbesondere um das Zeitverhalten der Anwendung (Anwendungszeit, Durchsatz) sowie das Verbrauchsverhalten in Bezug auf Speicher, Prozessorzeit und andere Ressourcen.“ [3]

- *Compatibility*

„Das Merkmal der Kompatibilität betrachtet einerseits die explizite Zusammenarbeit des Systems mit anderen Systemen und andererseits die impliziten Auswirkungen des Systems auf andere Systeme, die auf der gleichen Hardware laufen.“ [3]

- *Usability*

„Das Qualitätsmerkmal der Benutzerfreundlichkeit beschäftigt sich damit, wie gut ein System seine Nutzer dabei unterstützt ihre Ziele effizient und effektiv zu erreichen. Zu den zentralen Teilmerkmalen zählt hier beispielsweise der Aufwand für die Erlernung des Systems sowie für die Bedienung. Ein weiteres Teilmerkmal ist die Attraktivität des Systems für die Nutzer.“ [3]

- *Reliability*

„Das Qualitätsmerkmal der Zuverlässigkeit beschreibt die Fähigkeit des Systems sein Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu bewahren. Zu den Teilmerkmalen zählen hier beispielsweise die Robustheit gegenüber fehlerhaften Eingaben, die Stabilität beim Umgang mit fehlerhaften Systemzuständen und die Wiederherstellbarkeit bei Systemausfällen.“ [3]

- *Security*

„Das Merkmal umfasst alle in Bezug auf Datensicherheit relevanten Teilmerkmale wie die Datenintegrität, Authentizität und Vertraulichkeit.“ [3]

- *Maintainability*

„Das Merkmal der Wartbarkeit, teilweise auch als Änderbarkeit bezeichnet, beschäftigt sich vorrangig mit der internen Qualität eines Systems. Zentrale Teilmerkmale hier sind der Aufwand zur Analyse eines Fehlers oder Fehlverhaltens, der Aufwand zur Durchführung von Modifikationen mit entsprechender Testbarkeit sowie die Stabilität des Systems gegenüber unerwarteten Seiteneffekten.“ [3]

- *Portability*

„Die Portabilität als weiteres Qualitätsmerkmal beschäftigt sich mit dem Aufwand, um ein System strukturell zu verändern, auf eine andere Umgebung zu verlagern oder einzelne Systemkomponenten auszutauschen. Als Teilmerkmale nennt der Standard die Anpassbarkeit eines Systems an unterschiedliche Umgebungen, den Aufwand zur Installation und die Austauschbarkeit von Komponenten.“ [3]

Abbildung 2.4 zeigt eine Übersicht über die acht Qualitätsmerkmale und löst jedes Merkmal feingranular weiter auf.

Um das Qualitätsmerkmal „Usability“ zu verfeinern, kann die Norm EN ISO 9241-110 (Grundsätze der Dialoggestaltung) herangezogen werden. Diese beschreibt, welche Eigenschaften Dialoge erfüllen sollen, um eine gute Usability zu erreichen und werden im folgenden aufgezählt:

- **Aufgabenangemessenheit:** „Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen.“

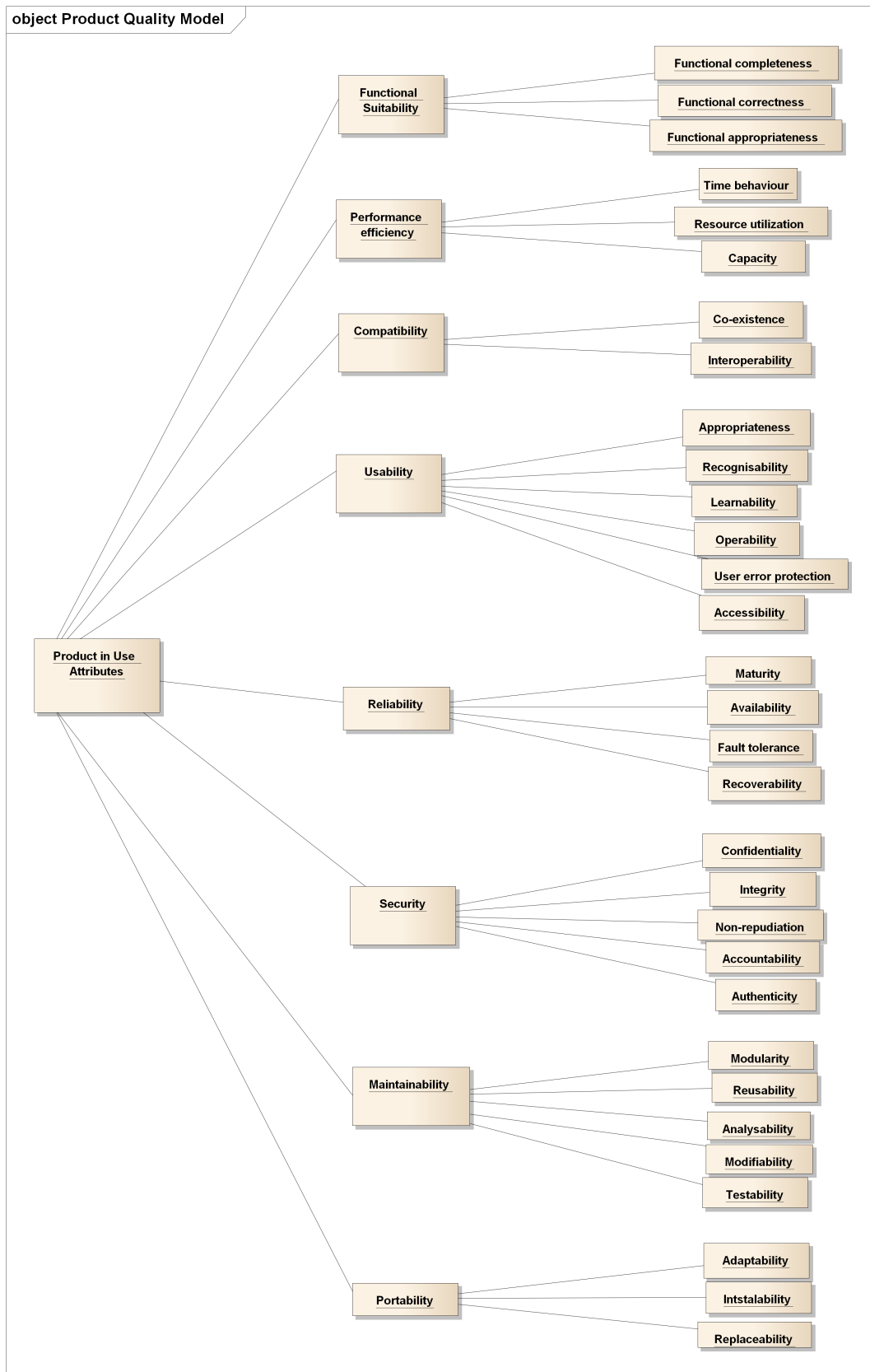


Abbildung 2.4 – Qualitätsmerkmale des Product Quality Model nach ISO/IEC 25010

- Selbstbeschreibungsfähigkeit: „Ein Dialog ist in dem Maße selbstbeschreibungsfähig, in dem für den Benutzer zu jeder Zeit offensichtlich ist, in welchem Dialog, an welcher Stelle er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können.“
- Lernförderlichkeit
- Steuerbarkeit
- Erwartungskonformität: „Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z.B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den allgemein anerkannten Konventionen.“
- Individualisierbarkeit
- Fehlertolerant: „Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand durch den Benutzer erreicht werden kann.“

Quelle: Wikipedia -> überprüfen ob andere Quelle verfügbar ist, am Besten die Norm direkt

2.4 Entwicklungsframework

2.4.1 .NET Framework

Es wurden im folgenden Abschnitt die Quellen [4],[5] verwendet.

Das .NET Framework ist eine von Microsoft entwickelte Entwicklungsplattform für Anwendersoftware, die erstmals im Januar 2002 in der Version 1.0 erhältlich war. Es stellt die Umsetzung des Common Language Infrastructure (CLI) Standards dar, der sprach- und plattformunabhängige Anwendungsentwicklung spezifiziert. Wesentliche Bestandteile des .NET Frameworks sind die Laufzeitumgebung Common Language Runtime (CLR) und die Framework Class Library (FCL). Die CLR verwaltet Speicher, Thread- und Codeausführung, Überprüfung der Codesicherheit und Kompilierung. Von der CLR verwalteter Code wird dabei Managed Code genannt, außerhalb der CLR laufender Code Unmanaged Code. Es wird eine Vielzahl von Programmiersprachen unterstützt, unter anderem C#, C++ und Visual Basic .NET. Beim Kompilieren der verschiedenen Hochsprachen wird der Code zunächst in eine

Zwischensprache übersetzt, der Common Intermediate Language (CIL). Aus dem daraus entstehenden Intermediate Language Code (IL-Code) wird dann von einem Just-In-Time-Compiler (JIT-Compiler) zur Laufzeit Maschinencode erzeugt (Abbildung 2.5).

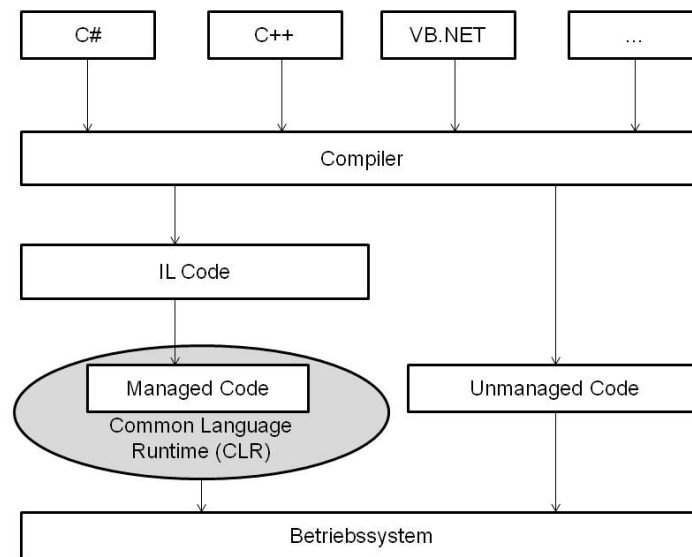


Abbildung 2.5 – Managed Code und Unmanaged Code

Die FCL ist eine objektorientierte Klassenbibliothek mit einer Auflistung wiederverwendbarer Typen, von denen eigener Code abgeleitet werden kann.

2.4.2 Windows Presentation Framework(WPF)

Mit Einführung des .NET Frameworks 3.0 im Jahr 2006 wurde Windows Presentation Foundation (WPF) veröffentlicht, ein Framework zur Erstellung von grafischen Benutzeroberflächen. WPF stellt den Nachfolger für das seit Version 1.0 im .NET Framework enthaltene Windows Forms. Es können Desktop- sowie Webanwendungen erstellt werden.

Eines der zentralen Konzepte von WPF ist die Verwendung der XML-basierten Beschreibungssprache Extensible Application Markup Language (XAML) zur Beschreibung der Benutzeroberfläche. XAML ermöglicht eine übersichtlichere und kompaktere Beschreibung der Benutzeroberfläche als die Erstellung in C#. Die Logik zu einer erstellten Benutzeroberfläche wird in C# programmiert. Dies kann in der zum XAML-File gehörenden Codebehind-Datei erfolgen oder

in einem eigenen C#-Sourcefile, das über DataBindings oder Commands an die XAML-Datei angebunden werden kann.

Außerdem werden die Leistungsressourcen der Grafikkarten mit 3D-Beschleunigern besser ausgenutzt im Vergleich zu Windows Forms, da zum Rendern der GUI DirectX benutzt wird anstatt Graphics Device Interface+ (GDI+). Mithilfe von DirectX kann WPF grafische Elemente selbst zeichnen anstatt durch dass sie durch das Betriebssystem gezeichnet werden. Ein weiterer Vorteil von WPF ist das vektorbasierte Zeichnen der Anwendungsinhalte. Dies ermöglicht eine beliebige Skalierung der Inhalte ohne Verpixelung.

Kapitel 3

Anforderungsanalyse

3.1 Übersicht

In diesem Kapitel soll eine Anforderungsanalyse für eine Entwicklungsumgebung für GIGABOX-Steuergeräte durchgeführt werden. In Abbildung 2.1 wurde gezeigt, für welche Anwendungen GIGABOXEN eingesetzt werden. Die Anforderungen an die Entwicklungsumgebung wurden so gestellt, dass sie Softwareentwicklern für GIGABOX-Funktionen die Umsetzung dieser Anwendungen ermöglicht und möglichst große Unterstützung bei der Codeentwicklung bietet.

Zuerst wurden Gespräche mit Entwicklern geführt, vorhandener Programmcode aus realisierten Projekten analysiert und selbstständig Codeentwicklung mit dem bestehenden configurAIDER betrieben um herauszufinden, welche Use-Cases eine IDE zur Entwicklung von Funktionen für GIGABOX-Steuergeräte abdecken soll. Aus dieser Analyse entstanden die in Abbildung 3.1 veranschaulichten Use-Cases.

Von den ermittelten Use-Cases wurden direkt die funktionalen Anforderungen abgeleitet, die im folgenden Abschnitt näher beleuchtet werden.

Auf Basis der Qualitätsmerkmale für die Anforderungsspezifikation im Standard IEEE 830-1998 werden die ermittelten funktionalen Anforderungen eingeteilt in drei Prioritätsklassen:

- Essential: Das Software-System kann nicht akzeptiert werden, wenn diese Anforderung nicht in der vereinbarten Form geliefert wird.
- Conditional: Diese Anforderungen erweitern das Software-System in geeigneter Form. Wenn sie fehlen, würden sie den Einsatz des Systems jedoch nicht gefährden.

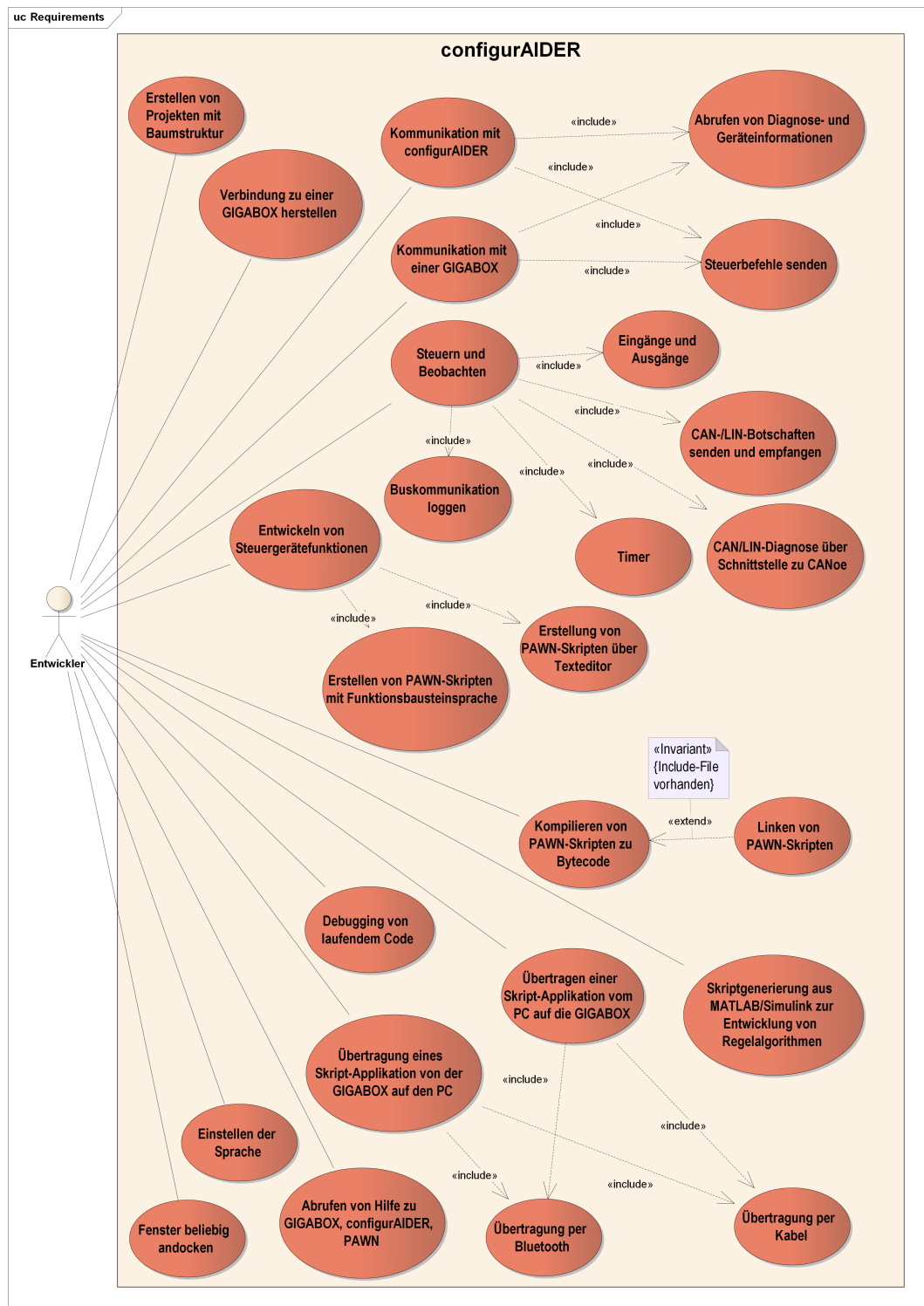


Abbildung 3.1 – Use-Case-Diagramm funktionale Anforderungen

- Optional: Diese Anforderungen sind nicht unbedingt notwendig, für die Anwender jedoch eine angenehme Erweiterung (nice to have).

Zur Ermittlung der nichtfunktionalen Anforderungen wurden bekannte Vorgehensmodelle und Normen der Softwareentwicklung herangezogen. Die hergeleiteten nichtfunktionalen Anforderungen sollen bei der Entwicklung beachtet werden, um eine hohe Qualität der zu entwickelnden Software sicherzustellen. Auf eine Bewertung wird verzichtet.

Quelle: [3]

3.2 Ermittlung und Bewertung von funktionalen Anforderungen

3.2.1 Erstellung von Projekten mit Baumstruktur

Beschreibung:

Innerhalb eines Projektes können verschiedene GIGABOX-Modelle angelegt werden. Für jedes angelegte GIGABOX-Modell können PAWN-Skripte erstellt werden mit der zum Modell passenden Dateiendung `.gt**.p`. Für jedes angelegte GIGABOX-Modell können Include-Files hinzugefügt werden. Der Benutzer erhält einen strukturierten Überblick über alle seine GIGABOX-Projekte und alle für ein Projekt entwickelten Skripte. Das Öffnen eines Skriptes erfolgt komfortabel über Doppelklick auf ein Skript in der Projektstruktur. Lästiges navigieren durch die Dateistruktur des PCs beim Öffnen eines Skriptes entfällt + der Benutzer muss den Ablageort im Dateisystem nicht wissen.

Bewertung:

Erhöht den Benutzerkomfort signifikant, deshalb Einstufung in Prioritätsklasse „Conditional“.

3.2.2 Einstellung der Sprache

Beschreibung:

Die Sprache der Benutzeroberfläche kann angepasst werden. Es soll Deutsch und Englisch zur Verfügung stehen.

Bewertung:

Die Darstellung der Benutzeroberfläche in der Muttersprache des Benutzers erhöht die Usability, da es die Selbstbeschreibungsfähigkeit und Individualisierbarkeit fördert. Englisch muss auf jeden Fall zur Verfügung stehen, da es international am weitesten verbreitete Sprache ist und auch in Deutschland von einem Großteil der Menschen verstanden wird. Die Darstellung der Benutzeroberfläche in Englisch wird deshalb in Prioritätsklasse „Essential“ eingeteilt. Die Darstellung in Deutsch wird in Klasse „Optional“ eingeteilt, da die Usability dadurch nur mäßig erhöht wird. Das liegt daran, dass der potenzielle Benutzerkreis des Tools Entwickler sind, denen größtenteils der Umgang mit englischen GUIs vertraut ist.

3.2.3 Texteditor zur Erstellung von PAWN-Skripten

- Schreiben von PAWN-Code

Beschreibung:

Es steht ein Textfeld zur Verfügung, in dem PAWN-Code editiert werden kann.

Bewertung:

Klasse „Essential“, da als Grundfunktionalität einer IDE einzustufen.

- Copy/Paste

Beschreibung:

Codefragmente können kopiert und an anderer Stelle eingefügt werden.

Bewertung:

Macht die Codeentwicklung bedeutend effektiver, da dem Benutzer Tipparbeit erspart bleibt. Weiterer Vorteil ist die Fehlervermeidung durch Tippfehler. Deshalb Einstufung in Klasse „Conditional“.

- Suchen/Ersetzen

Beschreibung:

Ein Skript kann nach Zeichenketten durchsucht werden. Die gefundenen Ergebnisse werden farblich hervorgehoben und es kann zu den Ergebnissen gesprungen werden. Alle gefundenen Ergebnisse können ersetzt werden durch eine neue Zeichenkette.

Bewertung:

Das Auffinden von Variablen und Methoden im Skript wird erheblich erleichtert. Bei Umbenennung von Variablen oder Methoden wird viel

Zeit eingespart und es werden Tippfehler vermieden. Einteilung in Klasse „Conditional“.

- Letzte Schritte rückgängig machen

Beschreibung:

Die letzten vom Benutzer vorgenommenen Anweisungen können rückgängig gemacht werden.

Bewertung:

Fördert die Fehlertoleranz. Einteilung in Klasse „Conditional“

- Paralleles Öffnen von Skripten

Beschreibung:

Mehrere Skripte lassen sich parallel über Tabs öffnen.

Bewertung:

Erhöht die Usability (effektives arbeiten). Es kann einfacher Code zwischen verschiedenen Skripten kopiert werden. Klasse „Conditional“

- Anzeige aller instanziierten Variablen im Skript

Beschreibung:

Alle instanziierten Variablen werden aufgelistet. Doppelklick auf einen Variablennamen in der Liste markiert im Skript alle Verwendungen der Variable. Per Drag&Drop kann ein Variablenname aus der Liste in das Skript eingefügt werden.

Bewertung:

Benutzer hat alle instanziierten Variablen im Blick und kann leicht zu Verwendungsstellen im Skript navigieren. Das Einfügen per Drag&Drop bietet eine komfortable Möglichkeit der Variablenverwendung und verhindert Fehleingaben durch Tippfehler. Einteilung in Klasse „Optional“.

- Anzeige aller implementierten Funktionen zur Navigation im Skript

Beschreibung:

Alle implementierten Funktionen werden gelistet. Doppelklick auf eine Funktion führt zu einem Sprung zur Funktionsimplementierung. Per Drag&Drop kann ein Funktionsname in das Skript gezogen werden, ein Aufruf der Funktion wird in das Skript eingefügt. Es können alle Stellen im Skript markiert werden, an denen die gewählte Funktion aufgerufen wird.

Bewertung:

Benutzer hat alle implementierten Funktionen im Blick und kann leicht zu Verwendungsstellen im Skript navigieren. Komfortable Möglichkeit der Funktionsverwendung, Verhindert Fehleingaben durch Tippfehler. Einteilung in Klasse „Optional“.

- Bei der Codeentwicklung unterstützende Funktionen

Beschreibung:

Es sollen Funktionen integriert werden, die schnelleres und effektiveres Schreiben von Code ermöglichen. Bsp: Autovervollständigung von Variablen, Funktionen und Anweisungen bei der Tastatureingabe

Bewertung:

Einteilung in Klasse „Conditional“.

- Routingeditor

Beschreibung:

Beschreibung für CAN hier. Auch für LIN gibt es Dokumente, die Botschaften definieren. Die Dokumente könnten geparkt werden und darauf ein PAWN-Skript generiert werden, das die Botschaften als Variablen enthält.

Aus einer Datenbankdatei (.dbc/.arxml) können dort definierte Bussignale geladen werden. Die Signale können in verschiedenen CAN-Botschaften enthalten sein. Aus den Bussignalen kann im Routingeditor eine eigene CAN-Busbotschaft konfiguriert werden mit einer vom Benutzer definierten ID. Dazu können die Signale innerhalb der Nutzdatenfeldes einer PDU frei angeordnet werden.

Aus der im Routing-Editor konfigurierten Botschaft kann PAWN-Code generiert werden und in die Funktion OnCanRxEvent() eines geöffneten Skriptes im Texteditor eingefügt werden.

Bewertung:

Beim Programmieren der GIGABOX tritt des öfteren der Use-Case auf, dass aus Signalen, die in verschiedenen auf dem Bus ankommenden Botschaften enthalten sind, eine neue Botschaft erstellt und verschickt werden soll.

Dabei werden oft Bitverschiebungen benötigt, deren Programmierung viel Zeit- und Denkaufwand bedeuten. Der Routingeditor ermöglicht die grafische Konfiguration von Busbotschaften und erleichtern die Umsetzung dieser Routingaufgaben. Einteilung in Klasse „Optional“.

3.2.4 Konfigurieren einer GIGABOX ohne PAWN-Kenntnisse

Beschreibung:

Grundlegende Funktionen einer GIGABOX sollen ohne Kenntnisse von PAWN konfiguriert werden können. Dazu bietet sich die Funktionsbausteinsprache an.

Bewertung:

Erleichtert es Kunden ohne Programmierkenntnisse, selbständig GIGABOX-Funktionalitäten zu konfigurieren. Erweitert möglicherweise den Kreis an potentiellen Käufern der GIGABOX. Im Vergleich zur direkten Codeentwicklung können allerdings nur eingeschränkte Funktionalitäten realisiert werden. Erfahrene Softwareentwickler werden aufgrund dieser Tatsache PAWN-Code direkt entwickeln. Einteilung in Klasse „Optional“.

3.2.5 Verbundene GIGABOXEN auflisten

Beschreibung:

Alle GIGABOXEN, die mit dem PC per USB oder Bluetooth verbunden sind, werden aufgelistet. Es werden Geräteinformationen wie z.B. Modellname, Seriennummer bereitgestellt. Der Benutzer kann aus der Liste eine GIGABOX anwählen, mit der kommuniziert werden soll.

Bewertung:

Verbindung zu GIGABOX per USB herstellen wird als Grundfunktionalität eingestuft, Klasse „Essential“. Verbindung per Bluetooth wird in Klasse „Optional“ eingestuft.

3.2.6 PAWN-Compiler

Beschreibung:

PAWN-Skripte können in Bytecode übersetzt werden. Wenn Include-Files verwendet werden, soll ein Linker den Bytecode des Skriptes und des Include-Files nach dem Übersetzen zusammenfügen.

Bewertung:

Das Kompilieren ist eine Grundfunktionalität und wird deshalb in die Klasse „Essential“ eingestuft.

3.2.7 Beschreiben/Auslesen des Flash-Speichers der GIGABOX

Beschreibung:

Skript-Applikationen können vom PC auf den Flash-Speicher einer GIGABOX übertragen werden. Sich bereits auf dem Flash-Speicher befindliche Skripte können auf den PC geladen und angezeigt werden. Die Übertragung soll jeweils per Kabel über die USB-Schnittstelle und per Bluetooth möglich sein.

Bewertung:

Die Übertragung von Skripten vom PC auf eine GIGABOX über die USB-Schnittstelle wird als Grundfunktionalität bewertet und in Klasse „Essential“ eingeteilt. Die Übertragung von der GIGABOX auf den PC über die USB-Schnittstelle ist für den Entwickler sehr nützlich, aber nicht unbedingt notwendig und wird deshalb als Klasse „Conditional“ eingestuft. Bluetooth-Übertragung wird in Klasse „Optional“ eingestuft, da es wenige Use-Cases gibt, bei denen eine Funkübertragung Vorteile gegenüber der Kabelübertragung bietet. Ein möglicher Use-Case wäre eine schwer zugängliche GIGABOX im Fahrzeug, auf die ein neues Skript aufgespielt werden soll. Eine Übertragung per Bluetooth würde einen aufwändigen Ausbau der GIGABOX aus dem Fahrzeug ersparen.

3.2.8 Kommunikation mit einer GIGABOX

Beschreibung:

Benutzer soll Befehle an eine GIGABOX senden können, z. B. einen Reset der GIGABOX veranlassen, den Bootloader aufrufen. Außerdem sollen Diagnose- und Geräteinformationen abgerufen werden können. Die Datenübertragung soll per Kabel über die USB-Schnittstelle und per Bluetooth möglich sein.

Bewertung:

Das Senden von Befehlen und Abrufen von Diagnose- und Geräteinformationen per Kabel über die USB-Schnittstelle wird in Klasse „Essential“ eingestuft. Die Datenübertragung per Bluetooth wird in Klasse „Optional“ eingestuft, da es wenige Use-Cases gibt, bei denen eine Funkübertragung Vorteile gegenüber der Kabelübertragung bietet. Ein möglicher Use-Case wäre eine schwer zugängliche GIGABOX im Fahrzeug, auf die ein neues Skript aufgespielt werden soll. Eine Übertragung per Bluetooth würde einen aufwändigen Ausbau der GIGABOX aus dem Fahrzeug ersparen.

3.2.9 Kommunikation mit dem Benutzer

Beschreibung:

Benutzer soll Befehle über die Konsole an die IDE senden können, z. B. das Kompilieren eines Skriptes veranlassen. Außerdem sollen Diagnose- und Programminformationen der IDE abgerufen werden können.

Bewertung:

Einstufung in Klasse „Essential“.

3.2.10 Steuern und Beobachten von Ein-/Ausgängen und Timern

Beschreibung:

Die Zustände der digitalen und analogen Ausgänge sowie der internen Timer sollen über die Oberfläche der Entwicklungsumgebung gesteuert und beobachtet werden können. Die Zustände der digitalen und analogen Eingänge sollen über die Oberfläche der Entwicklungsumgebung beobachtet werden können. Außerdem sollen vom Benutzer konfigurierte CAN- und LIN-Botschaften über die IDE gesendet werden können. Dabei soll einmaliges und zyklisches Senden von Botschaften möglich sein. Auf dem Bus liegende Botschaften einer definierten ID (CAN) bzw. für eine definierte Adresse (LIN) sollen beobachtet werden können.

Zusätzlich soll die Möglichkeit bestehen, Buskommunikation innerhalb eines definierten Zeitfensters mitzuloggen. Alle Busbotschaften, die innerhalb dieses Zeitfensters auf dem Bus liegen, sollen dabei mit Zeitstempel gelistet werden.

Bewertung:

Die beschriebenen Funktionen sind für den Nutzer hilfreich bei der Codeentwicklung, aber nicht unbedingt notwendig. Damit können die Funktionen in die Prioritätsklasse „Optional“ eingeteilt werden.

3.2.11 Debugging

Beschreibung:

Es sollen Breakpoints gesetzt werden können. Wenn ein Breakpoint erreicht wird, wird die Codeausführung auf der virtuellen Maschine der GIGABOX angehalten. Da der Codeausführung eventbasiert ist, werden die gesetzten Break-

points nur erreicht, wenn das Event ausgelöst wird, in dem sich der Break-point befindet. Events, die auslösen während die Codeausführung angehalten ist, müssen ignoriert werden. -> führt zu Problemen, da Timer ablaufen aber nicht neu aufgezogen werden -> Timerevents müssten trotzdem im Hintergrund auslösen und neu aufgezogen werden. Aber Anweisungen in den Event außer TimerSet() werden ignoriert.

CAN und LIN-Events werden ignoriert.

Der Benutzer kann zeilenweise zu den folgenden Anweisungen springen.

Bewertung:

Die beschriebenen Funktionen sind für den Nutzer hilfreich bei der Codeentwicklung, aber nicht unbedingt notwendig. Damit können die Funktionen in die Prioritätsklasse „Optional“ eingeteilt werden.

3.2.12 Bereitstellung von Dokumentationen

Beschreibung:

Die IDE soll Hilfedokumentation zu den GIGABOX-Modellen, zum configurAIDER und zu PAWN bereitstellen.

Bewertung:

Ein direkter Zugriff von der IDE aus auf die Hilfedokumentation erhöht die Usability. Dies bietet dem Benutzer den Vorteil, dass er die Oberfläche der IDE nicht verlassen muss um Dokumentationen aufzurufen. Einteilung in Klasse „Conditional“.

3.2.13 Individualisierbare Oberfläche

Beschreibung:

Die Fenster innerhalb der IDE sind frei andockbar. Somit kann der Benutzer die Oberfläche individuell nach seinen Wünschen aufbauen.

Bewertung:

Steigert die Usability (Individualisierbarkeit), ist aber keine unbedingt notwendige Funktion und wird deshalb als „Optional“ eingestuft.

3.3 Ermittlung von nichtfunktionalen Anforderungen

- Entwicklung des Tools mit WPF unter Verwendung MVVM-Pattern
- Entwicklung für Windows 32bit + 64bit als Zielbetriebssystem
- Entwicklung nach V-Modell
- Qualitätskriterien an Software nach ISO 9126 (aus Wiki)/ siehe auch Kapitel 6 Moderne Softwarearchitektur
 - Wartbarkeit
 - * Analysierbarkeit:
Coding-Richtlinien sollen eingehalten werden: Verständliche Kommentare, einheitliche Namensgebung etc
 - * Modifizierbarkeit:
SOLID-Prinzipien einhalten, um möglichst wenig Abhängigkeiten unter den Klassen zu erreichen. Dadurch treten bei Modifikationen innerhalb einer Klasse weniger schwer vorhersehbare Fehler auf
 - * Testbarkeit:
Keine Ahnung auf was bei der Entwicklung geachtet werden soll um gute Testbarkeit zu erreichen
 - Benutzbarkeit (genauer definiert in Norm EN ISO 9241-110)
 - Effizienz
 - * Akzeptable Zeit bis Tool gestartet ist und verwendbar ist
 - * Unmittelbare und flüssige Reaktion auf Benutzereingaben
 - Übertragbarkeit
 - * Anpassbarkeit: Möglichkeit, Software an verschiedene Umgebungen anzupassen
Verschiedene Betriebssysteme? 32/64Bit?
 - Zuverlässigkeit
 - * Reife: Geringe Versagenshäufigkeit durch Fehlerzustände: Ausreichendes Testing
 - * Wiederherstellbarkeit: Fähigkeit, bei einem Versagen das Leistungsniveau wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen. Zu berücksichtigen sind die dafür benötigte Zeit und der benötigte Aufwand.

Kapitel 4

Ist-Zustand

4.1 Überblick

In diesem Kapitel wird der Ist-Zustand des configurAIDERS beschrieben. Es wird die Benutzeroberfläche vorgestellt und die Funktionalität beschrieben. Im Anschluss wird die Funktionalität und die Usability bewertet. Abbildung 4.1 veranschaulicht die Module, die im configureAIDER enthalten sind.

4.2 Elemente des configurAIDER

4.2.1 Rahmenfenster

Der configurAIDER besteht aus einem Rahmenfenster, in das weitere Fenster eingebettet werden können. Das Rahmenfenster besitzt im oberen Fensterbereich eine Menüleiste in horizontaler Ausrichtung und ist in einem Hintergrund gehalten, der Firmenlogo und -farbe zeigt. Die Menüeinträge ändern sich dynamisch in Abhängigkeit der vom Benutzer geöffneten Fenster. Beim Start des configurAIDERS ist das Fenster „Verfügbare Geräte“ standardmäßig in das Rahmenfenster eingebettet. Eingebettete Fenster können nicht über die Grenzen des Rahmenfensters hinausgezogen werden.

4.2.2 Fenster „Verfügbare Geräte“

Im Fenster „Verfügbare Geräte“ werden alle GIGABOXen angezeigt, die an der USB-Schnittstelle des PC angeschlossen sind. Es wird der Modellname, die

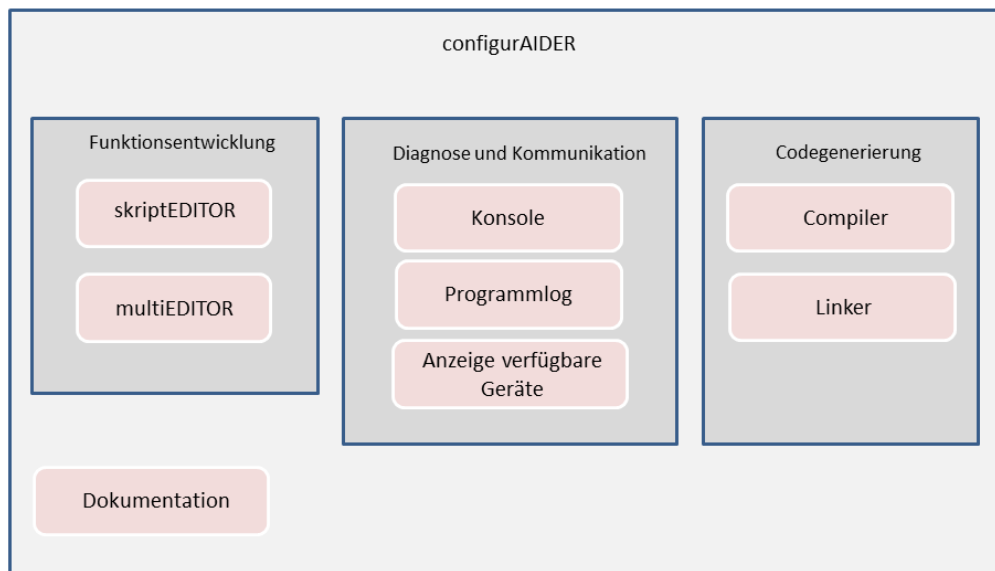


Abbildung 4.1 – Übersicht configurAIDER

Seriennummer und der Verbindungsstatus von GIGABOXen ausgegeben, die am PC erkannt wurden. Über dieses Fenster kann eine GIGABOX angewählt werden, mit der eine Kommunikation gestartet werden soll. Wenn keine reale GIGABOX zur Verfügung steht, kann eine GIGABOX simuliert werden. Die Auswahl eines zu simulierenden Gerätes erfolgt über ein eigenes Dialogfenster „Simuliertes Gerät“ (siehe Abbildung 4.2).

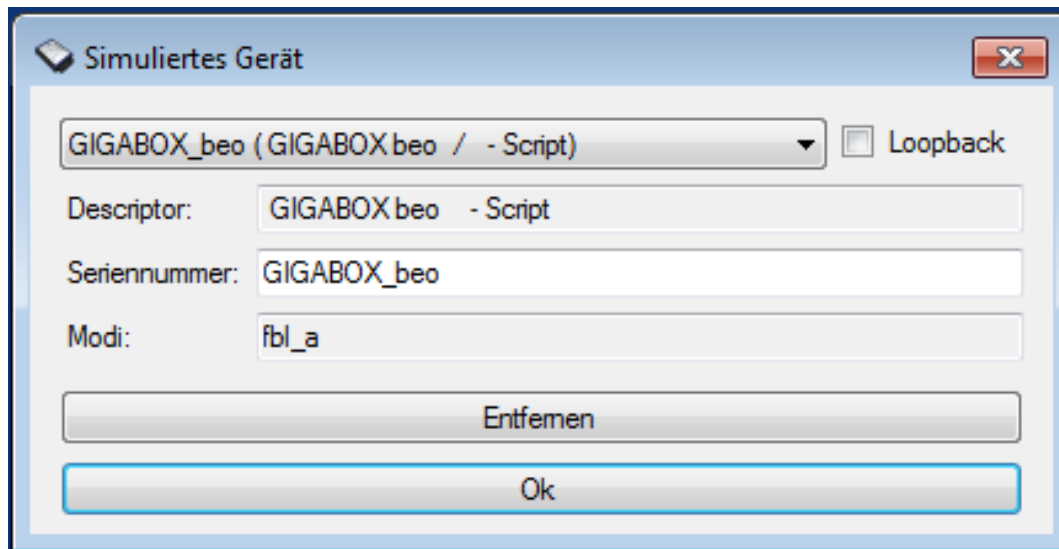


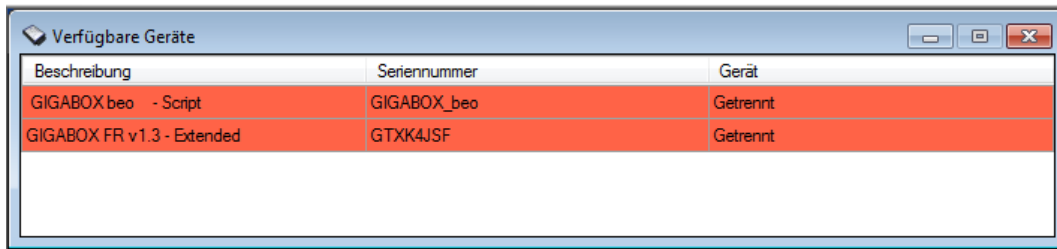
Abbildung 4.2 – Fenster „Simuliertes Gerät“

Die simulierte GIGABOX wird wie eine reale GIGABOX gelistet.

Nach der Anwahl eines GIGABOX-Modelles wird versucht, eine Verbindung zu dem Gerät herzustellen (wird das tatsächlich hardwaremäßig versucht oder ist die Anzeige nur dazu da, dem Benutzer zu signalisieren, welches Gerät er konfiguriert). Bei erfolgreicher Verbindungsherstellung wird das Gerät grün hinterlegt und es öffnet sich ein Fenster zur Editorauswahl. Dort kann ausgewählt werden, ob die Konsole, der scriptEDITOR oder der multiEDITOR geöffnet werden soll. Bei einigen GIGABOX-Modellen steht kein multiEDITOR zur Verfügung.

4.2.3 Fenster „Konsole“

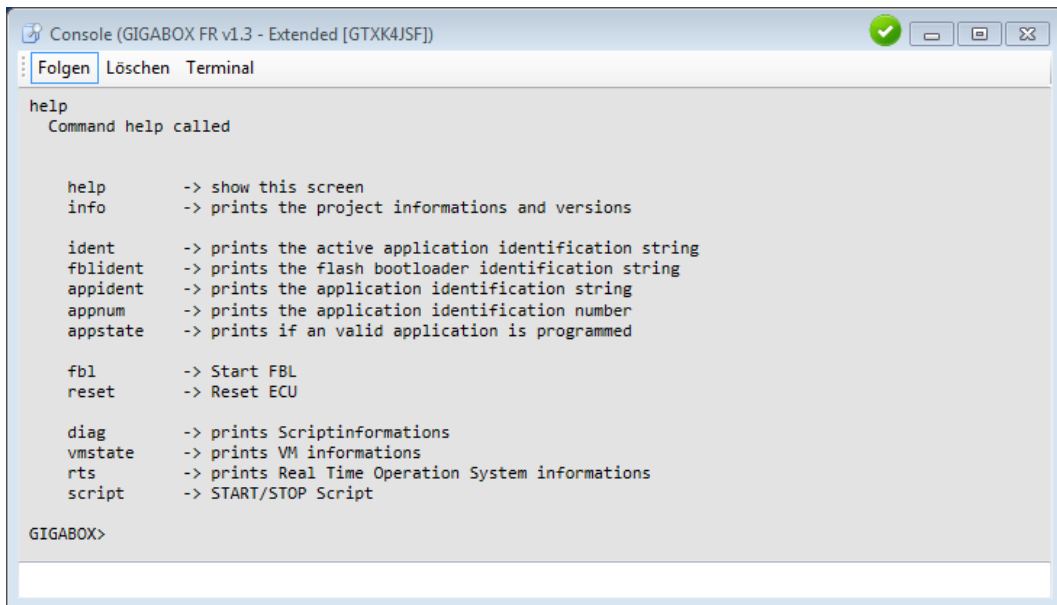
Die Konsole besteht aus einer Textbox zur Eingabe von Befehlen und einer Textbox zur Ausgabe von Informationen. Sie ermöglicht Kommunikation mit einer GIGABOX. Die Konsole kann z.B eingesetzt werden zur Abfrage von Diagnoseinformationen und um einen Reset zu veranlassen.



Beschreibung	Seriennummer	Gerät
GIGABOX beo - Script	GIGABOX_beo	Getrennt
GIGABOX FR v1.3 - Extended	GTXXK4JSF	Getrennt

Abbildung 4.3 – Fenster „Verfügbare Geräte“

Über die Schaltflächen „Folgen“ und „Terminal“ kann gewählt werden, wie die Textein- und ausgabe erfolgen soll. Wenn „Folgen“ ausgewählt wird, erfolgt die Texteingabe in einem von der Textausgabe getrennten Fenster. Wenn „Terminal“ ausgewählt wird, erfolgt die Textein- und ausgabe in einem gemeinsamen Fenster.



```

Console (GIGABOX FR v1.3 - Extended [GTXXK4JSF])
Folgen Löschen Terminal

help
Command help called

help      -> show this screen
info      -> prints the project informations and versions

ident     -> prints the active application identification string
fbldent   -> prints the flash bootloader identification string
appident  -> prints the application identification string
appnum    -> prints the application identification number
appstate  -> prints if an valid application is programmed

fbl       -> Start FBL
reset     -> Reset ECU

diag      -> prints Scriptinformations
vmstate   -> prints VM informations
rts       -> prints Real Time Operation System informations
script    -> START/STOP Script

GIGABOX>

```

Abbildung 4.4 – Fenster „Konsole“

4.2.4 Fenster „Programmlog“

Das „Programmlog“-Fenster besteht aus einer Textbox, über die tabellarisch Erfolgs- und Fehlermeldungen über die ausgeführten Aktionen des configurAI-DERs ausgegeben werden.

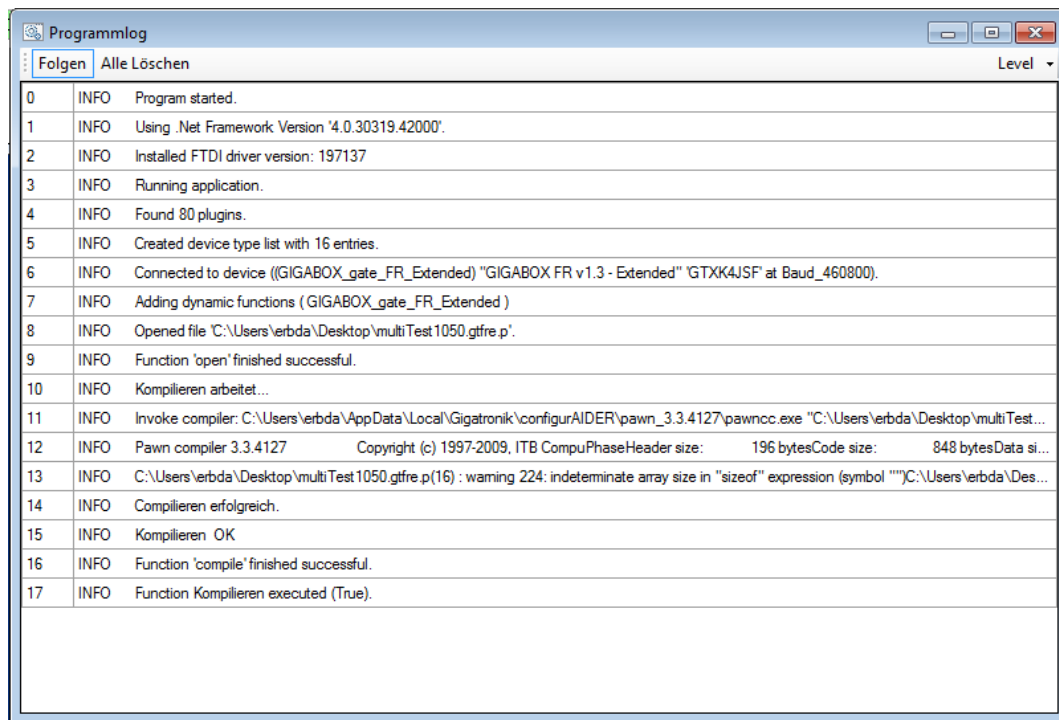


Abbildung 4.5 – Fenster „Programmlog“

4.2.5 Fenster „scriptEDITOR“

Der scriptEDITOR ist ein Texteditor zum Erstellen und Editieren von Funktionen in PAWN. Um den scriptEDITOR aufzurufen, muss zuerst eine Verbindung zu einer realen oder simulierten GIGABOX hergestellt werden über das Fenster „Verfügbare Geräte“. Anschließend öffnet sich automatisch ein Fenster zur Editorauswahl. Dort kann der scriptEDITOR angewählt werden. Mit dem scriptEDITOR erstellte Skriptdateien haben die Dateiendung .gt**.p. Die Sterne sind zu ersetzen mit einem Kürzel, das abhängig ist vom verbundenen GIGABOX-Modell. Für die unterschiedlichen GIGABOX-Modelle werden also Skripte mit verschiedenen Dateiendungen erstellt, da die Modelle unterschiedliche Funktionen implementieren. Bsp: GIGABOX gate FR extended: .gtfre.p GIGABOX gate XL: .gtxl.p GIGABOX gate gateway: .gtfrg.p

In der obersten Fensterzeile ist eine Buttonleiste angeordnet mit den folgenden Buttons:

- Neu: Öffnet ein neues Skript
- Öffnen: Öffnet ein vorhandenes Skript

- Speichern: Speichert das geöffnete Skript unter einem bereits festgelegt Dateinamen und Pfad
- Speichern unter: Speichert das geöffnete Skript unter einem anzugebenden Dateinamen und Pfad
- Kompilieren: Interpreter erzeugt Bytecode aus dem PAWN-Quellcode
- Herunterladen: Interpreter erzeugt Bytecode aus dem PAWN-Quellcode. Die PAWN-Skriptdatei wird als ZIP-Datei komprimiert. Anschließend wird der Bytecode und das gezippte Skript auf die virtuelle Maschine der GIGABOX übertragen.
- Heraufladen: Die als ZIP-Datei komprimierte Skriptdatei auf dem Steuergerät wird auf den PC übertragen, entzippt und im Texteditor geöffnet.
- Reset: Führt einen Reset der GIGABOX durch
- Log: Öffnet Konsole

Aufzählung der Buttons zu detailreich Mittig ist der Texteditor angeordnet. Hier kann ein Skript angezeigt und bearbeitet werden. Es stehen bei der Entwicklung typische Unterstützungswerkzeuge wie Codefolding zur Verfügung. **(Aufzählung der unterstützenden Werkzeuge wie Autovervollständigung etc.)**

Unten befindet sich ein Fenster zur Ausgabe von Fehlern, Warnungen und Erfolgsmeldungen.

4.2.6 Fenster „multiEDITOR“

Der „multiEDITOR“ ist ein Editor zur Funktionskonfiguration mit WENN-DANN-Anweisungen in Tabellen. Er soll es Benutzern mit geringer Programmiererfahrung ermöglichen, einfache Funktionen für die GIGABOX zu entwickeln. Aus den konfigurierten Funktionstabellen kann anschließend automatisch ein PAWN-Skript generiert werden. Das erstellte PAWN-Skript kann dann kompiliert und der Bytecode auf den Flash-Speicher der GIGABOX übertragen werden.

In der obersten Fensterzeile ist eine Buttonleiste angeordnet mit den folgenden Buttons:

- Neu: Öffnet ein neues Skript
- Öffnen: Öffnet ein vorhandenes Skript

```

1 // Declare variables.
2 new ida_CAN_1[] = { 1, 2 }
3 new ida_CAN_2[] = { 1 }
4 new timeout[] = { 1 }
5 new timeout_counter[] = { 1 }
6 static #undef can_0_1_0 = 0
7 static #undef can_0_2_1 = 0
8
9
10
11 // Handle startup event.
12 public OnStartup()
13 {
14     TimerSet( 0, 1 )
15     CAN_Init(CAN_1, CAN_BAUD_500KBIT_65PERC, CAN_GetArbitrationMask(ida_CAN_1, sizeof(ida_CAN_1)), CAN_GetAcceptanceMask(ida_CAN_1, sizeof(ida_CAN_1)));
16     CAN_Init(CAN_2, CAN_BAUD_500KBIT_65PERC, CAN_GetArbitrationMask(ida_CAN_2, sizeof(ida_CAN_2)), CAN_GetAcceptanceMask(ida_CAN_2, sizeof(ida_CAN_2)));
17 }
18
19 // Handle timeouts.
20 OnTimeoutTimer()
21 {
22     for ( new i_var = 0; i_var < sizeof ( timeout_counter ); i_var++ )
23     {
24         --timeout_counter[i_var];
25         if ( timeout_counter[i_var] == 0 )
26         {
27             // Reset Timeout Counter
28             timeout_counter[i_var] = timeout[i_var]
29             switch (i_var)
30             {
31                 // Timeout Action Code
32
33                 default:
34                 {
35                     break
36                 }
37             }
38         }
39     }
40 }
41
42 // Handle single timer event.
43 public OnTimerFromTimer1 timerNumber 1
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Pawn compiler 3.3.4127 Copyright (c) 1997-2009, ITS CompuPhase
 Header size: 196 bytes
 Code size: 848 bytes
 Data size: 16 bytes
 Stack/heap size: 256 bytes; estimated max. use=14 cells (56 bytes)
 Total requirements: 1316 bytes
 3 Warnings.
 C:\Users\verba\Desktop\multitest1050.gtfre.p(16) : warning 224: indeterminate array size in "sizeof" expression (symbol "")
 C:\Users\verba\Desktop\multitest1050.gtfre.p(16) : warning 224: indeterminate array size in "sizeof" expression (symbol "")
 C:\Users\verba\Desktop\multitest1050.gtfre.p(23) : warning 224: indeterminate array size in "sizeof" expression (symbol "")
 Compilieren erfolgreich.
 Kompilieren OK

Abbildung 4.6 – Fenster „scriptEDITOR“

- Speichern: Speichert das geöffnete Skript unter einem bereits festgelegt Dateinamen und Pfad
- Speichern unter: Speichert das geöffnete Skript unter einem anzugebenden Dateinamen und Pfad
- Generieren: Aus den mittels Tabelle konfigurierten Funktionen wird ein PAWN-Skript generiert
- Kompilieren: Interpreter erzeugt Bytecode aus dem PAWN-Quellcode
- Herunterladen: Aus den mittels Tabelle konfigurierten Funktionen wird ein PAWN-Skript generiert. Interpreter erzeugt Bytecode aus dem PAWN-Quellcode. Die PAWN-Skriptdatei wird als ZIP-Datei komprimiert. Anschließend wird der Bytecode und das gezippte Skript auf die virtuelle Maschine der GIGABOX übertragen.
- Heraufladen: Die als ZIP-Datei komprimierte Skriptdatei auf dem Steuergerät wird auf den PC übertragen und entzippt. PAWN-Code wird rückgewandelt in Tabelle mit WENN-DANN-SONST-Anweisungen. Die funktioniert nur, wenn der von der GIGABOX hochgeladenen Code ursprünglich mithilfe des multiEDITORS erzeugt wurde.
- DBC: Es können Vector-DBC Dateien oder AUTOSAR .arxml Dateien erstellt oder geöffnet werden. Dabei handelt es sich um Datenbanken, in denen CAN-Botschaften und Signale definiert sind
- Reset: Führt einen Reset der GIGABOX durch
- Log: Öffnet Konsole

Aufzählung der Buttons zu detailreich

Unter der Buttonleiste befindet sich eine Tabelle, in der CAN-Botschaften und Signale erstellt und bearbeitet werden können. Botschaften und Signale, die aus Datenbank-Dateien (.dbc oder .arxml) geladen wurden, werden hier angezeigt.

Darunter befindet sich eine Tabelle zur Konfiguration von Funktionen mithilfe von WENN-DANN-SONST-Anweisungen.

4.3 Funktionalität

Abbildung 4.8 gibt einen Überblick über die verfügbaren Funktionen des configurAIDERS.

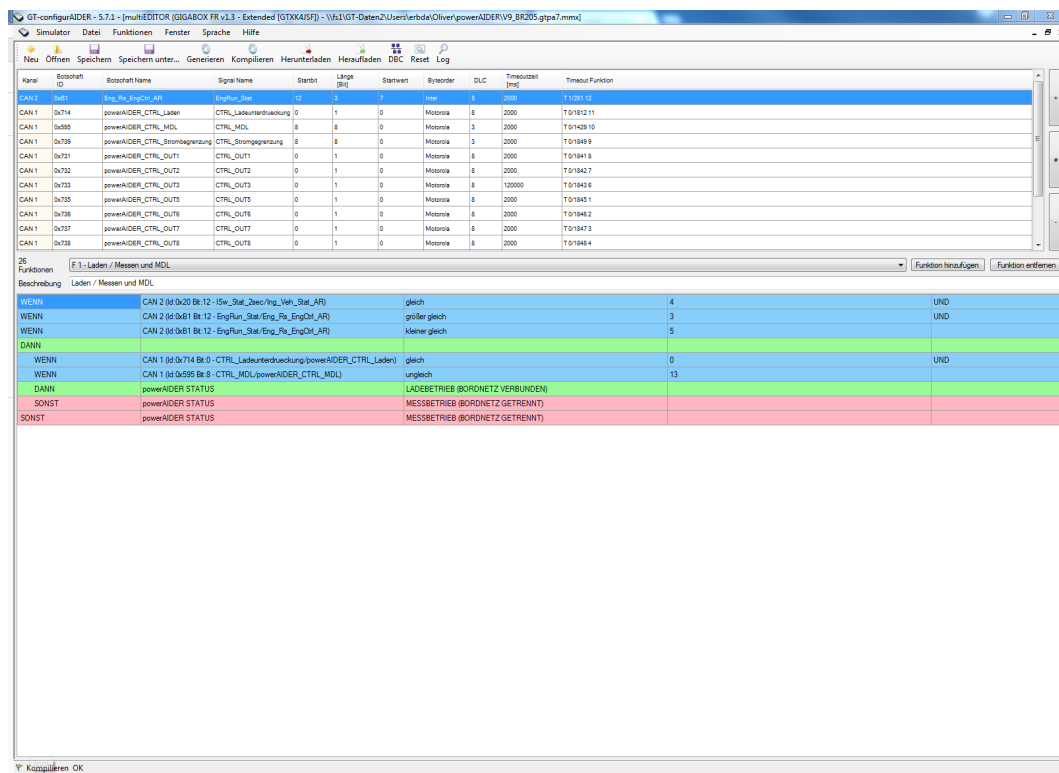


Abbildung 4.7 – Fenster „multiEDITOR“



Abbildung 4.8 – Überblick Funktionen des configurAIDERS

4.3.1 Einstellung der Sprache

Die Sprache der Fensterdialoge des configurAIDERS kann in der Menüleiste des Rahmenfenster unter dem Eintrag „Sprache“ eingestellt werden. Es steht Deutsch und Englisch als Sprache zur Verfügung. Nach der Sprache von Deutsch auf Englisch wird ein Dialog eingeblendet, dass die Sprachänderung erst nach einem Neustart übernommen wird. Nach ausgeführtem Neustart werden nicht alle Menüeinträge auf Englisch dargestellt. So werden beispielsweise im scriptEDITOR die Menüeinträge unter „Edit“ weiterhin komplett in Deutsch dargestellt. Die Funktionstabellen im multiEDITOR werden auch weiterhin in Deutsch dargestellt.

4.3.2 Bereitstellung von Dokumentation zur GIGABOX, configurAIDER und PAWN

Über den Menüeintrag „Hilfe“ können PDF-Dokumente zu den verschiedenen GIGABOX-Modellen, configurAIDER und zur Skriptsprache PAWN abgerufen werden. Für den configurAIDER steht ein allgemeines Handbuch, Hinweise zur Installation und jeweils für scriptEDITOR und multiEDITOR ein eigenes Handbuch zur Verfügung. Der Menüeintrag „linEDITOR“ enthält nur einen ausgegrauten Eintrag. Zu den GIGABOX-Modellen stehen Handbücher, Flyer und Informationen zu Firmwareupdates zur Verfügung. Die Dokumentationen werden nicht durchgängig für alle GIGABOX-Modelle zur Verfügung gestellt. So wird für die GIGABOX gate xl kein Handbuch angeboten, für die GIGABOX gate powerAIDER kein Flyer.

4.3.3 Verbindung zu einer GIGABOX herstellen

GIGABOXEN, die an der USB-Schnittstelle des PCs erkannt werden sowie simulierte GIGABOXEN werden im Fenster „Verfügbare Geräte“ (Abbildung 4.3) gelistet. Hier kann die Verbindung zu einem Gerät hergestellt werden. Falls keine reale GIGABOX zur Verfügung steht, kann eine GIGABOX simuliert werden.

Die verschiedenen Modelle besitzen unterschiedliche Hardwarefunktionen (unterschiedliche Anzahl von Ein-/Ausgängen, Bussysteme), die zu entwickeln den Funktionen müssen daran angepasst werden. Dem configurAIDER muss deshalb vor der Funktionsentwicklung bekannt sein, für welches Modell Code entwickelt werden soll.

Es wurden verschiedene Fehler entdeckt, die im Folgenden aufgelistet werden.

1. Das Fenster „Verfügbare Geräte“ wird vom Benutzer geschlossen. Bei einem Wiederaufruf des Fensters kann kein neues Gerät mehr simuliert werden. Lösung: configurAIDER neu starten
2. Im Fenster „Verfügbare Geräte“ sind mehrere Geräte gelistet. Bei dem Versuch, die Verbindung zu einem Gerät herzustellen über Rechtsklick auf das Gerät und „Verbinden“, wird keine Verbindung zum angewählten Gerät hergestellt sondern zum ersten Gerät in der Liste. Lösung: Verbindung herstellen über Doppelklick auf Gerät
3. Die Simulation von Geräten schlägt gelegentlich fehl, es wird über das Programmlog-Fenster ein Fehler ausgegeben (ERROR: Internal device table obscured!) Lösung: Neuer Versuch der Herstellung einer Verbindung

4.3.4 PAWN-Code kompilieren

Aus scriptEDITOR und multiEDITOR heraus kann mithilfe des Compilers PAWN-Code in Bytecode übersetzt werden. Der erzeugte Bytecode kann auf der virtuellen Maschine der GIGABOX ausgeführt werden.

4.3.5 PAWN-Code entwickeln

PAWN-Skripte schreiben

PAWN-Skripte können im scriptEDITOR geschrieben werden. Erstellte Skriptdateien besitzen die Dateiendung .gt*.p. Die Sterne sind zu ersetzen mit einem Kürzel, das abhängig ist vom verbundenen GIGABOX-Modell. Für die unterschiedlichen GIGABOX-Modelle werden also Skripte mit verschiedenen Dateiendungen erstellt, da die Modelle unterschiedliche Funktionen implementieren. Bsp: GIGABOX gate FR extended: .gtfre.p GIGABOX gate XL: .gtxl.p GIGABOX gate gateway: .gtfrg.p

Um den Benutzer bei der Codeentwicklung zu unterstützen, sind aus anderen IDEs bekannte Funktionen integriert. Einige ausgewählte Funktionen sind hier beispielhaft angeführt:

- Codeabschnitte sind auf- und zuklappbar
- Unterschiedliche Farbgebung (Anweisungen in blau, Kommentare in grün etc)

- Eventfunktionen sind als Rumpf vorimplementiert
- Markierung der Zeile, in der sich der Cursor aktuell befindet

Im Vergleich zu weit verbreiteten IDEs wie Visual Studio ist der Umfang an unterstützenden Funktionen aber klein gehalten.

PAWN-Skripte aus Funktionstabellen generieren

Im multiEDITOR können tabellarisch Funktionen konfiguriert werden mit WENN-DANN-SONST-Anweisungen. Aus den konfigurierten Funktionstabellen kann anschließend automatisch ein PAWN-Skript generiert werden. Das erstellte PAWN-Skript kann dann kompiliert und der Bytecode auf den Flash-Speicher der GIGABOX übertragen werden.

Für jede Funktion wird eine neue Tabelle erstellt. Jede Tabelle besteht mindestens aus 3 Zeilen mit den Anweisungen WENN, DANN und SONST. Es können mehrere WENN-Anweisungszeilen erstellt werden, die mit dem Bedingungsoperator UND oder ODER verknüpft werden müssen. Außerdem können mehrere DANN und SONST-Anweisungen erstellt werden, die jeweils miteinander über den UND-Operator verknüpft werden. Unter jeweils jede DANN und SONST-Anweisung kann eine neue, untergeordnete WENN-DANN-SONST-Anweisung eingefügt werden. Damit sind verschachtelte Anweisungen möglich.

In den Spalten der WENN-Anweisungszeile können digitale und analoge Eingänge, digitale Ausgänge sowie eingehende Busbotschaften auf definierte Werte geprüft werden. Wenn die dort definierten Werte angenommen werden, wird die DANN-Anweisungszeile ausgeführt. Hier können digitale Ausgänge durchgeschaltet oder ausgeschaltet werden. Das Absetzen von Busbotschaften ist nicht möglich.

Wenn die in der WENN-Anweisungszeile definierten Werte nicht angenommen werden, wird die SONST-Anweisungszeile ausgeführt. Auch hier können digitale Ausgänge durchgeschaltet oder ausgeschaltet werden.

Bewertung

4.3.6 Übertragen von Skript-Applikationen

Es kann eine Skript-Applikation auf die GIGABOX übertragen werden. Die Skriptdatei wird als ZIP-Datei komprimiert und zusammen mit dem Bytecode auf den Flash-Speicher geschrieben. Wenn ein Skript mithilfe des multiEDITORS generiert wurde, wird zusätzlich zu der Skriptdatei (.gt*.p) auch

die multiEDITOR-Konfigurationsdatei (.gt**.mmx) (+.tmp -> was steht da drin?) mit in die ZIP-Datei gepackt.

Ein auf der GIGABOX ausgeführtes Skript kann auch von der GIGABOX auf den PC geladen werden. Dabei wird die auf der GIGABOX gespeicherte ZIP-Datei auf den PC geladen, entpackt und im scriptEDITOR angezeigt. Wenn das Skript mit dem multiEDITOR erzeugt wurde, liegt der ZIP-Datei zusätzlich die multiEDITOR-Konfigurationsdatei (.gt**.mmx) bei und kann folglich auch im multiEDITOR geöffnet werden.

4.3.7 Kommunikaton mit verbundener GIGABOX

Über die Konsole können Befehle an eine reale GIGABOX gesendet werden, z.B. Befehl zum Reset, Aufrufen des Bootloaders. Dort können auch Informationen der GIGABOX abgerufen und ausgegeben werden, z.B. Diagnoseinformationen über Bussysteme oder Timer.

Wenn die Schaltfläche „Terminal“ aktiviert ist, kann eine erfolgte Texteingabe nicht mehr korrigiert werden. Bei deaktivierter Schaltfläche ist die Korrektur möglich.

4.3.8 Ausgabe von Ereignismeldungen

Es kann über das Programmlog-Fenster nachverfolgt werden, welche Aktionen vom configurAIDER durchgeführt wurden. Wichtige Erfolgs- und Fehlermeldungen werden dort ausgegeben. Bsp: Erfolgreich kompiliert.

4.3.9 Ergebnis

Kapitel 5

Evaluation

5.1 Funktionalität und Usability des Ist-Zustands

Es sollen die Funktionen des configurAIDERS bewertet werden hinsichtlich Umfang, Umsetzung und Fehlerzuständen. Zusätzlich soll die Usability nach EN ISO 9241-110 (siehe Abschnitt 2.3.2) bewertet werden.

5.1.1 Einstellung der Sprache

Es kann Deutsch und Englisch als Sprache eingestellt werden. Hier wäre zu überlegen, ob noch weitere Spracheinstellungen zur Verfügung gestellt werden sollten, falls es eine größere Anzahl an Benutzern aus nicht deutsch- oder englischsprachigen Ländern gibt. Die englische Spracheinstellung ist nicht konsistent umgesetzt, hier sollte darauf geachtet werden, dass durchgängig alle Bedienelemente englisch beschriftet werden.

5.1.2 Bereitstellung von Dokumentation zur GIGABOX, configurAIDER und PAWN

Über den Menüeintrag „Hilfe“ können PDF-Dokumente zu den verschiedenen GIGABOX-Modellen, configurAIDER und zur Skriptsprache PAWN abgerufen werden. Für den configurAIDER steht ein allgemeines Handbuch, Hinweise zur Installation und jeweils für scriptEDITOR und multiEDITOR ein eigenes Handbuch zur Verfügung. Hier ist zu kritisieren, dass der Menüeintrag „linEDITOR“ nur einen ausgegrauten Eintrag enthält und somit unnötig

ist. Zu den GIGABOX-Modellen stehen Handbücher, Flyer und Informationen zu Firmwareupdates zur Verfügung. Leider werden die Dokumentationen nicht durchgängig für alle GIGABOX-Modelle zur Verfügung gestellt. So wird für die GIGABOX gate xl kein Handbuch angeboten, für die GIGABOX gate powerAIDER kein Flyer.

5.1.3 Verbindung zu einer GIGABOX herstellen

Für den Benutzer ist nicht zu unterscheiden, ob ein Gerät in der Liste real oder simuliert ist. Simulierte Geräte werden in der Liste nicht hervorgehoben oder markiert. Die gewünschte Aufgabenangemessenheit ist für dieses Fenster damit nicht gegeben, da der Benutzer die Auswahl einer GIGABOX nicht effizient erledigen kann.

Die Auswahl eines zu simulierenden Gerätes erfolgt über ein eigenes Dialogfenster „Simuliertes Gerät“ (siehe Abbildung 4.2). Für den Benutzer ist in diesem Dialog nicht ersichtlich, was die Checkbox „Loopback“ bewirkt. Die Selbsterklärungsfähigkeit des Dialogs ist nicht ausreichend.

Bei der Auswahl eines zu simulierenden Gerätes im Fenster „Simuliertes Gerät“ wird das angewählte Gerät der Liste im Fenster „Verfügbare Geräte“ hinzugefügt, bevor die Auswahl mit „Ok“ bestätigt wird. Diese Reaktion entspricht nicht den üblichen, dem Benutzer bekannten Konventionen von Software und ist damit nicht erwartungskonform.

Aufgrund von diverse Fehlerzuständen, die beobachtet wurden, kann die Verbindung zu einem Gerät nicht zuverlässig und fehlerfrei hergestellt werden. Das Tool produziert Ergebnisse, die vom Benutzer so nicht erwartet werden und erfordert lästiges Neustarten des Programmes.

5.1.4 PAWN-Code kompilieren

Die Kompilierung funktioniert ohne Probleme.

5.1.5 PAWN-Code entwickeln

PAWN-Skripte schreiben

Keine Möglichkeit, letzte Aktion rückgängig zu machen -> schlechte Steuerbarkeit

Keine Erkennung von Syntaxfehlern -> schlechte Selbstbeschreibungsfähigkeit

Buttons Herunterladen/Heraufladen sind missverständlich -> schlechte Selbstbeschreibungsfähigkeit

Nur ein Skript kann geöffnet werden, nicht mehrere gleichzeitig -> Aufgabenangemessenheit

Autovervollständigung nur teilweise umgesetzt -> Aufgabenangemessenheit

Codeblöcke können nicht auskommentiert werden über Buttons/Tastenkombination -> Aufgabenangemessenheit

Die Buttons „Herunterladen“ und „Heraufladen“ in der oberen Buttonleiste sind nicht klar verständlich benannt. Dem Benutzer wird nicht klar, in welche Richtung der Datenaustausch erfolgt. Der Button „Log“ öffnet die Konsole, die Erwartung wäre, dass sich das Programmlog-Fenster öffnet. Die Selbstbeschreibungsfähigkeit und Erwartungskonformität der Buttonleiste ist negativ zu bewerten.

PAWN-Skripte aus Funktionstabellen generieren

Im Signaleditor ist Bit/CAN-Bit Unterschied nicht klar ersichtlich -> Selbstbeschreibungsfähigkeit

Es können keine CAN-Nachrichten gesendet werden -> eingeschränkte Funktionalität

Keine Reaktion auf LIN-Nachrichten möglich/ kein Senden von LIN-Nachrichten möglich

Keine Möglichkeit, SWITCH CASE-Anweisung einzufügen. Deshalb müssen verschachtelte WENN DANN-Anweisungen erstellt werden, die schnell unübersichtlich werden

Es ist nicht ersichtlich, dass über den DBC-Button auf .arxml-Dateien geladen werden können

Bei Klick auf DBC-Button öffnet sich Fenster zum Laden von .dbc und .armxl-Dateien. In diesem Fenster befindet sich eine Tabelle zur Anzeige der geladenen Signale. Bei Auswahl einer .dbc-Datei bleibt die Tabelle allerdings leer. Die im ausgewählten DBC definierten Signale werden dort nicht angezeigt

5.1.6 Übertragen von Skript-Applikationen

Das Übertragen von Skript-Applikationen funktioniert ohne Probleme.

5.1.7 Kommunikaton mit verbundener GIGABOX

Wenn die Schaltfläche „Terminal“ aktiviert ist, kann eine erfolgte Texteingabe nicht mehr mit der Rücktaste korrigiert werden. Bei deaktivierter Schaltfläche ist die Korrektur möglich. Die Konsole mangelt es folglich an Fehlertoleranz, was einer schlechten Usability entspricht.

5.1.8 Ausgabe von Ereignismeldungen des configurAI-DErs

Die Ausgabe von Ereignismeldungen über das Programmlog-Fenster ist zufriedenstellend.

5.2 Differenz zwischen Anforderungen und Ist-Zustand

	Anforderungen	Priorität	Ist-Zustand
Erstellung von Projekten mit Baumstruktur	Innerhalb eines Projektes können verschiedene GIGABOX-Modelle angelegt werden . Für jedes angelegte GIGABOX-Modell können PAWN-Skripte erstellt werden mit der zum Modell passenden Dateiendung. Für jedes angelegte GIGABOX-Modell können Include-Files hinzugefügt werden.	Conditional	Nicht realisiert
Einstellung der Sprache	Deutsch	Optional	Realisiert
	Englisch	Essential	Realisiert, aber nicht konsistent umgesetzt
Oberfläche individualisierbar	Frei andockbare Fenster	Optional	Fenster sind frei verschiebbar, aber nicht andockbar
Verbindung zu einer GIGABOX herstellen	Alle GIGABOXEN, die mit dem PC per USB verbunden sind, werden aufgelistet. Es werden Geräteinformationen wie z.B. Modellname, Seriennummer bereitgestellt. Der Benutzer kann aus der Liste eine GIGABOX anwählen, mit der kommuniziert werden soll.	Essential	Realisiert im Fenster „Verfügbare Geräte“. Es treten Fehler auf.
	Alle GIGABOXEN, die mit dem PC per Bluetooth verbunden sind, werden aufgelistet. Es werden Geräteinformationen wie z.B. Modellname, Seriennummer bereitgestellt. Der Benutzer kann aus der Liste eine GIGABOX anwählen, mit der kommuniziert werden soll.	Optional	Nicht realisiert

PAWN-Compiler	Compiler zum Übersetzen von PAWN-Code in Bytecode. Linker um Bytecode zusammenzufügen.	Essential	Realisiert
Beschreiben/Auslesen des Flash-Speichers der GIGABOX	Übertragen der Skript-Applikation vom PC auf den Flash-Speicher der GIGABOX per USB	Essential	Realisiert
	Übertragen der Skript-Applikation vom PC auf den Flash-Speicher der GIGABOX per Bluetooth	Optional	Nicht realisiert
	Übertragen der Skript-Applikation vom Flash-Speicher der GIGABOX auf den PC per USB	Conditional	Realisiert
	Übertragen der Skript-Applikation vom Flash-Speicher der GIGABOX auf den PC per Bluetooth	Optional	Nicht realisiert

Kommunikation mit einer GIGABOX	Benutzer kann Geräte- und Diagnoseinformationen der GIGABOX abrufen per USB	Essential	Realisiert im Fenster "Konsole"
	Benutzer kann Geräte- und Diagnoseinformationen der GIGABOX abrufen per Bluetooth	Optional	Nicht realisiert
	Benutzer kann Befehle an eine GIGABOX senden per USB	Essential	Realisiert im Fenster "Konsole"
	Benutzer kann Befehle an eine GIGABOX senden per Bluetooth	Optional	Nicht realisiert
Kommunikation mit dem Benutzer	Benutzer kann Diagnose- und Programminformationen der IDE abrufen	Essential	Realisiert im Fenster "Programmlog" sowie im Ausgabefenster des scriptEDITOR und multiEDITOR
	Benutzer kann über Konsole Befehle an IDE senden	Essential	Nicht realisiert
Steuern- und Beobachten von Ein- und Ausgängen und Timern	Steuern und Beobachten der Zustände der digitalen und analogen Ausgänge über GUI der IDE. Beobachten der Zustände der digitalen und analogen Eingänge über GUI der IDE.	Optional	Nicht realisiert
	Steuern und Beobachten der internen Timer über GUI der IDE.	Optional	Nicht realisiert
	CAN-/LIN-Botschaften senden (einmalig und zyklisch) über GUI der IDE.	Optional	Nicht realisiert
	Auf dem Bus liegende Botschaften einer definierten ID (CAN) bzw. für eine definierte Adresse (LIN) sollen beobachtet werden können.	Optional	Nicht realisiert
	Buskommunikation kann innerhalb eines definierten Zeitfensters mitgeloggt werden.	Optional	Nicht realisiert
Debugging	Setzen von Breakpoints im Code	Optional	Nicht realisiert
	Zeilenweise Steuerung der Anweisungsausführung	Optional	Nicht realisiert
Bereitstellung von Dokumentationen	Dokumentation zur IDE, GIGABOX-Modellen und PAWN	Conditional	Realisiert

Texteditor zur Erstellung von PAWN-Skripten	Schreiben von PAWN-Code	Essential	Realisiert
	Paralleles Öffnen von Skripten über Tabs	Conditional	Nicht realisiert
	Copy/Paste	Conditional	Realisiert
	Suchen/Ersetzen	Conditional	Realisiert
	Möglichkeit, letzte Schritte rückgängig zu machen	Conditional	Nicht realisiert
	Anzeige aller implementierten Funktionen zur Navigation im Skript	Optional	Nicht realisiert
	Anzeige aller instanziierten Variablen. Einfügen des Variablennamens per Drag&Drop	Optional	Nicht realisiert
	Routing-Editor	Optional	Nicht realisiert
	Funktionen, die bei der Codeentwicklung unterstützen, z.B. Autovollständigung	Conditional	Realisiert, aber Funktionsumfang ist mäßig
Konfigurieren einer GIGABOX ohne PAWN-Kenntnisse	Realisierung von Funktionen mit Funktionsbausteinsprache	Optional	Nicht mit Funktionsbausteinsprache realisiert sondern mit Anweisungstabellen

5.3 Quellcode des configurAIDERS

Was wurde mit WinForms gemacht, was mit WPF?

Entstand aus mehreren studentischen Arbeiten

Diagramm mit bestehender Architektur erstellen

5.4 Ergebnis

Kapitel 6

Konzeption

6.1 Übersicht

Ziel des Kapitels ist es, Konzepte aufzuzeigen, wie die in der Anforderungsanalyse erarbeiteten Anforderungen umgesetzt werden können.

6.2 Erweiterung des bestehenden configurAIDERS um neue Features

Der bestehende Code des configurAIDERS wird erweitert um neue Features, die bei der Bewertung der Anforderungen in Abschnitt ?? mit (+) oder (++) bewertet wurden. Die an bereits bestehenden Features in Abschnitt 4.3 kritisierten Punkte sollen verbessert werden.

Neue Funktionen:

- Projektverwaltung: Es können Projekte angelegt werden, in denen unterschiedliche Modelle der GIGABOX projiziert sind und die zugehörigen Skripte+Include-Files
- Texteditor wird erweitert um:
 1. Möglichkeit, letzte Schritte rückgängig zu machen
 2. Anzeige aller implementierten Funktionen zur Navigation im Skript
 3. Bei der Codeentwicklung unterstützende Funktionen

4. Routing-Editor

- Frei andockbare Fenster
- Steuern und Beobachten von DIN, AIN, DOUT, SWITCH

Vorteil:

- Das Tool erhält damit einen erhöhten Funktionsumfang, als wichtig bewertete funktionale Anforderungen können umgesetzt werden.

Nachteil:

- Nichtfunktionale Anforderungen wie Qualitätskriterien an Software nach ISO9126 (Wartbarkeit, Usability, Effizienz, Übertragbarkeit, Zuverlässigkeit) können nicht erfüllt werden, sondern werden sich verschlechtern durch eine weitere Funktionserweiterung des Tools
- Viel Einarbeitung in bestehenden Code nötig

6.3 Neuentwicklung des configurAIDERS

Das Tool wird von Grund auf neu entwickelt. Es sollen die funktionalen und nichtfunktionalen Anforderungen aus Kapitel 3 umgesetzt werden. Im Rahmen dieser Arbeit sollen vorerst nur funktionalen Anforderungen umgesetzt werden, die in die Klasse „Essential“ oder „Conditional“ eingeordnet wurden. Allerdings sollen alle nichtfunktionalen Anforderungen erfüllt werden.

Vorteil:

- Tool wird einfacher erweiterbar als bisheriger Stand. Dies ermöglicht es, zukünftig einfacher neue Features zu integrieren.
- Besser wartbar: Bugs können in Zukunft besser behoben werden, da Verhalten des neuen Codes besser bekannt ist
- Usability kann verbessert werden
- Tool konsistent mit WPF realisiert unter aktueller .NET Version 4.6
- Keine lange Einarbeitung in alten Code nötig

Nachteil:

- In Entwurf der neuen Softwarearchitektur muss Zeit investiert werden
- Tool bietet keinen direkten Mehrwert in Form von höherem Funktionsumfang
- Insgesamt höherer Aufwand um die gestellten funktionalen Anforderungen zu erreichen

6.4 Ergebnis

Es wird entschieden, das Konzept der Neuentwicklung des `configurAIDERS` umzusetzen.

Kapitel 7

Design

7.1 Entwurf der Benutzeroberfläche

Entwurf mit Pencil hier einfügen

Erläuterung des Entwurfes mit Bezugnahme auf ISO, die gute Usability definiert

7.2 Entwurf der Softwarearchitektur

[6]

Eine Softwarearchitektur definiert die Komponenten eines Systems, beschreibt deren wesentliche Merkmale und charakterisiert die Beziehungen dieser Komponenten. Sie beschreibt den statischen Aufbau einer Software im Sinne eines Bauplans und den dynamischen Ablauf einer Software im Sinne eines Ablaufplans.

Die in diesem Kapitel vorgestellte Software-Architektur des configurAIDERS wurde entworfen auf Basis der funktionalen und nichtfunktionalen Anforderungen an Software in Kapitel 3. Wie im vorigen Kapitel „Konzeption“ erläutert, werden nur Funktionen umgesetzt, die als „Essential“ oder „Conditional“ priorisiert wurden. **Evtl. nochmal die Funktionen aufzählen (textuell oder als UseCaseDiagramm), die implementiert werden sollen**

7.2.1 Entwurf der Kontextabgrenzung

Die Kontextabgrenzung zeigt das System als Blackbox und stellt das System in Kontext zu seiner Umgebung dar. Abbildung 7.1 zeigt die Kontextabgrenzung des configurAIDERS als UML Deployment Diagramm. Es werden alle den configurAIDER umgebenden Systeme dargestellt, die zur Erfüllung der in der Anforderungsanalyse hergeleiteten Use-Cases (nur die als „Essential“ oder „Conditional“ klassifizierten) benötigt werden. Dazu zählt der Benutzer, eine GIGABOX und ein Datenspeicher.

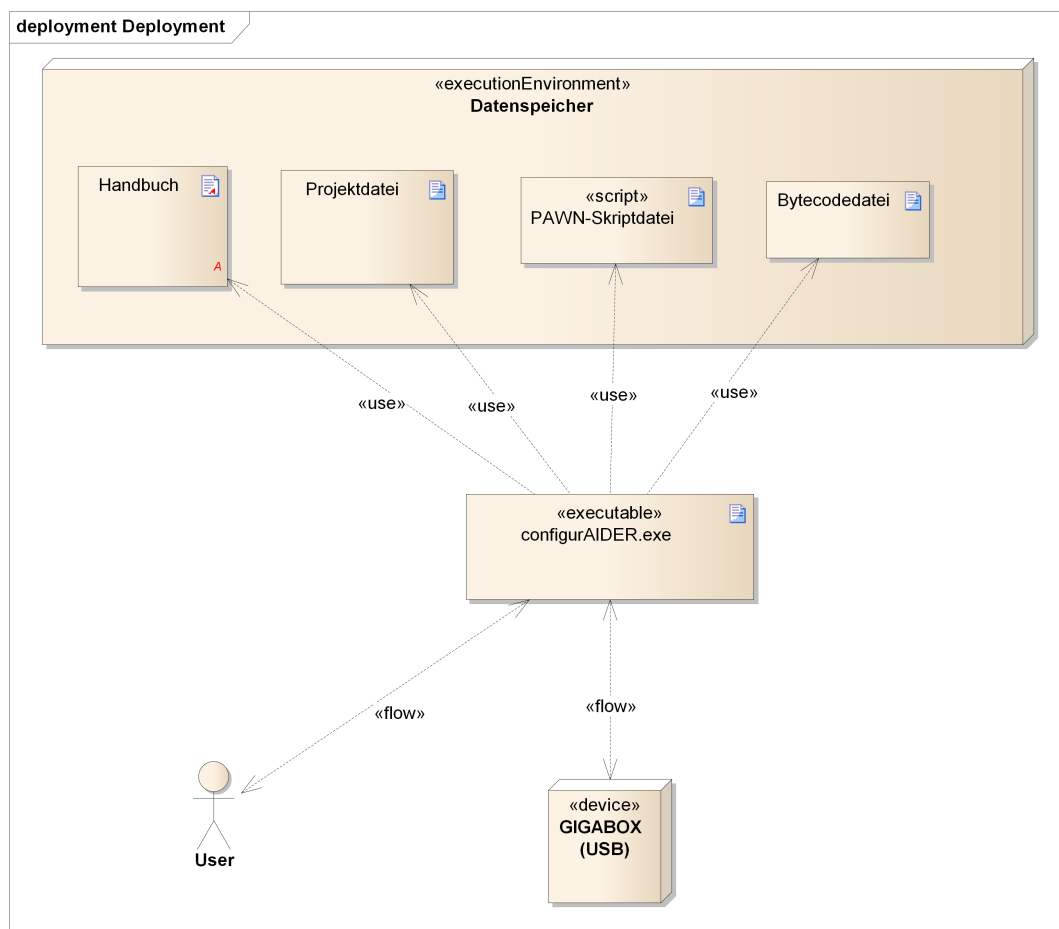


Abbildung 7.1 – Darstellung der Kontextabgrenzung des Systems im UML-Deployment Diagramm

7.2.2 Entwurf der Laufzeitsicht

Die Laufzeitsicht veranschaulicht, wie die Komponenten eines Systems zur Laufzeit zusammenarbeiten. Für jede Komponente des configurAIDERS wurde eine Laufzeitsicht mithilfe von UML Sequenzdiagrammen erstellt. Dort kann für jede Komponente abgelesen werden, mit welchen anderen Komponenten sich kommuniziert im Rahmen der wichtigsten Use-Cases.

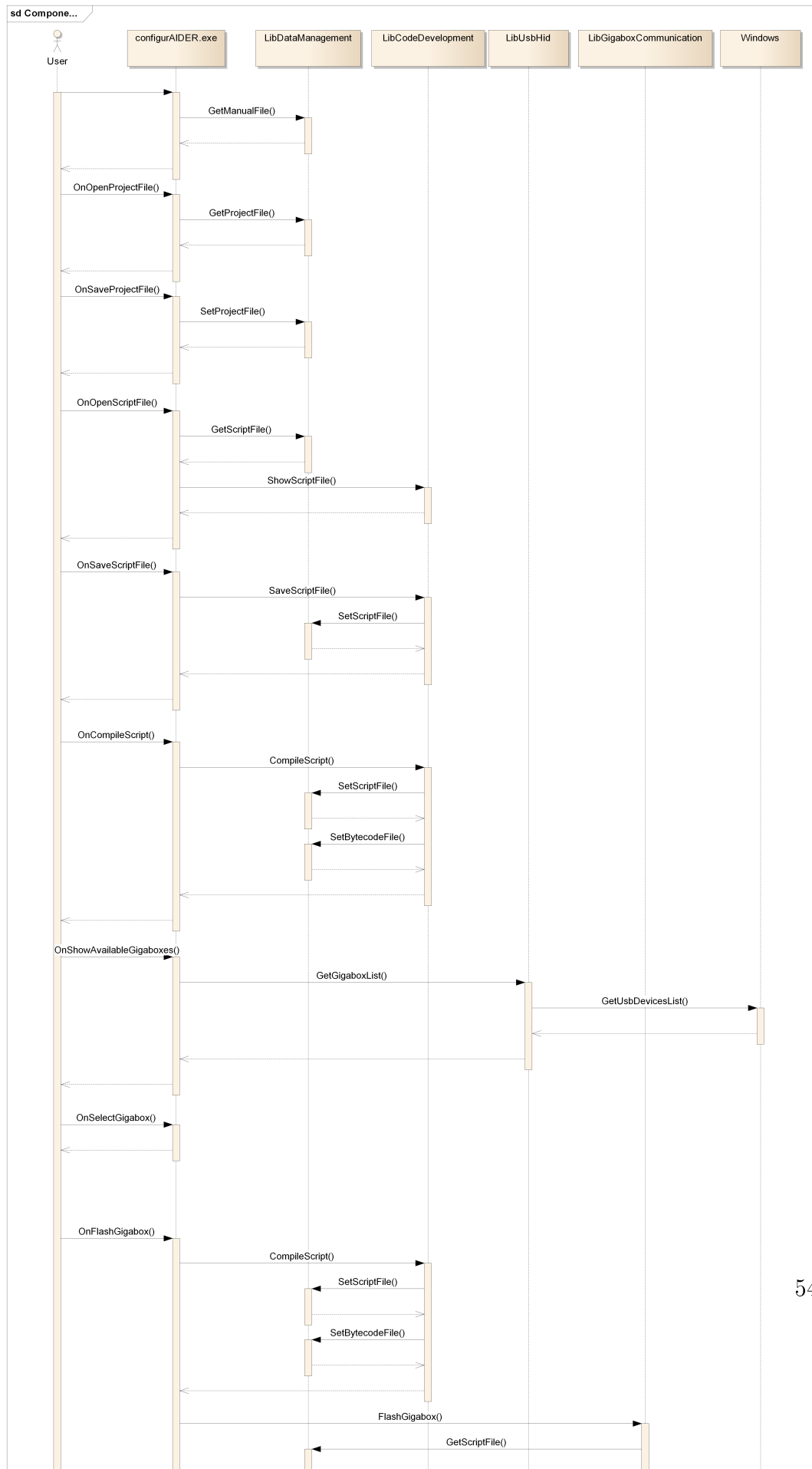
Abbildung XX zeigt die Laufzeitsicht des configurAIDERS als UML Sequenzdiagramm. Für die wichtigsten Use-Cases werden die Kommunikationsabläufe zwischen den verschiedenen Komponenten dargestellt.

Was ist Softwarearchitektur? IEEE 1471-2000

Architekturmuster: Layer-Architektur: MVVM Bild mit MVVM Model, 3 Schichten

Architektur des configurAIDERS

Module: Rahmenfenster Projektverwaltung



Kapitel 8

Realisierung

8.1 Andockbare Fenster

Realisierung mit AvalonDock

Beschreibung der bereitgestellten Funktionen

Beschreibung der Verwendung/ Codeanbindung von AvalonDock

8.2 Text Editor

Realisierung mit AvalonEdit

Beschreibung der Funktionen - Syntaxhighlighting, Zeilennummern, Autovervollständigung,...

8.3 Projektexplorer

Realisierung mit TreeView

Beschreibung der Funktionen

8.4 Konsole

8.5 Kommunikation mit GIGABOX

Beschreibung der API zu GIGABOX

Kapitel 9

Fazit und Ausblick

Abbildungsverzeichnis

2.1	Use-Case-Diagramm GIGABOX	4
2.2	Hardwareüberblick GIGABOX FD	5
2.3	Hardwareüberblick GIGABOX FD	6
2.4	Qualitätsmerkmale des Product Quality Model nach ISO/IEC 25010	10
2.5	Managed Code und Unmanaged Code	12
3.1	Use-Case-Diagramm funktionale Anforderungen	15
4.1	Übersicht configurAIDER	26
4.2	Fenster „Simuliertes Gerät“	27
4.3	Fenster „Verfügbare Geräte“	28
4.4	Fenster „Konsole“	28
4.5	Fenster „Programmlog“	29
4.6	Fenster „scriptEDITOR“	31
4.7	Fenster „multiEDITOR“	33
4.8	Überblick Funktionen des configurAIDERS	34
7.1	Darstellung der Kontextabgrenzung des Systems im UML-Deployment Diagramm	52
7.2	Darstellung der Laufzeitsicht des Systems im UML Sequenzdia- gramm	54

Tabellenverzeichnis

Literaturverzeichnis

- [1] ITB CompuPhase. Pawn Implementer´s Guide , 09.2016.
- [2] ITB CompuPhase. Pawn Language Guide , 01.2016.
- [3] Gerd Beneken Ulrike Hammerschall. *Software Requirements*. Pearson Deutschland GmbH, 2013.
- [4] K. Löffelmann D. Louis, S. Strasser. *Microsoft Visual C# 2005 - Das Entwicklerbuch*. Hrsg.: Microsoft Press Deutschland, 2006.
- [5] Microsoft. Übersicht über .NET Framework. [https://msdn.microsoft.com/de-de/library/vstudio/zw4w595w\(v=vs.11\)](https://msdn.microsoft.com/de-de/library/vstudio/zw4w595w(v=vs.11)), Stand: 13.10.2015.
- [6] Gernot Starke. *Effektive Software Architekturen*. Hanser Verlag, 2009.