Masterarbeit

AUTOMATISIERTE SPS-PROJEKTERSTELLUNG FÜR DIE ENTWICKLUNGSUMGEBUNG TWINCAT 3

ausgeführt am

CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Ing. Gernot Katzjäger, BSc
2010322011

betreut und begutachtet von
DI (FH) Gernot Hofer

Graz, im Jänner 2022

.....
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

Meiner Familie, meinem Chef, meinen Kolleginnen und Kollegen und meinem Betreuer möchte ich für die moralische Unterstützung und für Rat und Tat danken.

KURZFASSUNG

Die Komplexität von automatisierten Systemen, die durch eine speicherprogrammierbare Steuerung (SPS) gesteuert werden, wie es Prüfstände in der Automobilindustrie sein können, nimmt weiter zu. Die herkömmliche Planung, Programmierung und Bewertung dieser Systeme ist auf lange Sicht nicht kosteneffizient. Daher ist es für den Erfolg eines Unternehmens wichtig, die Effizienz des Entwicklungs- und Fertigungsprozesses zu steigern.

Ziel dieser Masterarbeit ist es daher, eine Anwendung zu entwickeln, die den Prozess solcher Systeme stärker automatisiert. Dies bedeutet, dass eine SPS-Software oder ein Validierungsplan für ein System nahezu automatisiert erstellt wird. Um dieses Ziel zu erreichen, ist die programmierte Anwendung in der Lage, Daten aus einer externen Quelle zu importieren und konvertieren. Die importierte Datei wird aus einem aktuellen Schaltplanprojekt eines solchen automatisierten Systems erstellt. Die Anwendung wandelt diese Daten in nützliche Informationen um, die in der entwickelten Anwendung verarbeitet werden, um den Entwicklungsprozess zu automatisieren.

Zusammengefasst erlaubt das entwickelte Programm die Erstellung einer Hardwarekonfiguration, das Anlegen lokaler und globaler Variablen, die Erstellung von definierten Quellcodezeilen in der Entwicklungsumgebung TwinCAT3 sowie die Erstellung eines Bewertungsplanes.

Je nach Komplexität des Systems ist das manuelle Eingreifen von Entwickler*innen in die Projekterstellung möglich bzw. notwendig. Allerdings macht diese Anwendung den Entwicklungsprozess von Anlagen, welche von einer SPS gesteuert werden, effizienter und führt zu einer Senkung der Fehleranfälligkeit.

ABSTRACT

The complexity of automated systems controlled by programmable logic controllers (PLCs), such as test benches in the automotive industry, continues to grow. The cost efficiency by conventional planning, programming and the evaluation of these systems is not sustainable in the long term. As a consequence it is important to increase the development and manufacturing processes´ efficency for the success of a company.

The aim of this master thesis is to develop an application which offers the opportunities either to create a software or an evaluation plan for an automated system. In order to achieve this, the programmed application is fed by data from external sources, which are files that provide the application with useful information of an automated system.

The developed program allows the creation of a hardware configuration, the creation of local and global variables as well as the creation of program modules with defined source code lines in the development environment TwinCAT3 as well as the generating an evaluation plan.

Depending on the complexity of the system, manual intervention of a developer during the project creation process could be needed. However, this application leads to a higher efficiency for the developing process of systems which are controlled by PLCs and is less prone to failure.

INHALTSVERZEICHNIS

1 EINLEITUNG

In den nachfolgenden Punkten wird die aktuelle Situation im Unternehmen dargestellt und die Zielsetzung dieser Arbeit erläutert. Ebenfalls wird die Aufteilung dieser Arbeit, in einen theoretischen und in einen praktischen Teil, näher beschrieben.

1.1 Aktuelle Situation

Aktuell wird die Erstellung von SPS-Projekten inkl. dessen internen Modulen, bzw. die Erstellung von bestimmten Matrizen, in der Abbildung 1-1 dargestellten Reihenfolge durchgeführt:

Die Erstellung eines SPS-Programmes benötigt auf konventionellem Wege einen enormen Zeitaufwand.

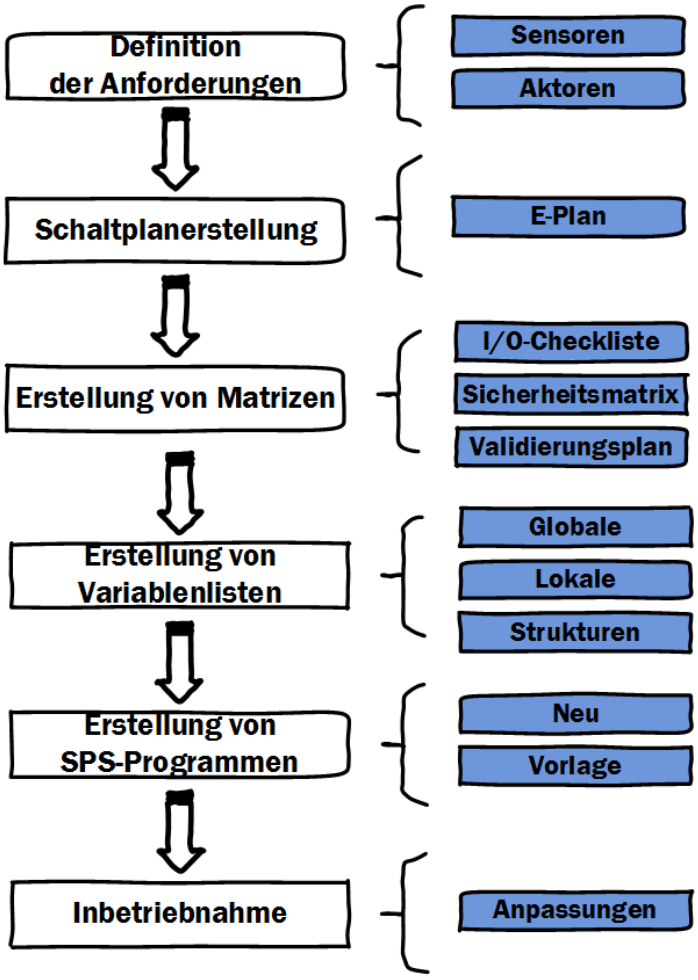

Abbildung 1-1: Ausgangssituation zur Erstellung von SPS-Projekten, Quelle: Eigene Darstellung

Ebenfalls lassen sich Fehler beim Übertragen von Variablennamen, sowie Werte aus der I/O-Checkliste oder den Schaltplänen nicht gänzlich vermeiden. Um eine möglichst fehlerfreie und standardisierte Codeerstellung zu gewährleisten, sollte diese weitestgehend automatisiert ablaufen. Dafür erforderlich ist, dass alle relevanten Daten aus einem Schaltplan in eine externe Datei übertragen werden können. Mittels Import dieser Datei in ein auf .Net-basierendem Programm, soll die Möglichkeit geschaffen werden, SPS-Programme aber auch Excel-Dateien, wie I/O-Checklisten, automatisiert erstellen zu können.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist die Ausarbeitung eines Grundkonzeptes, welches die Möglichkeiten beschreibt bzw. die Strukturen definiert, um eine automatisierte SPS-Projekterstellung und Erstellung der entsprechenden Matrizen aus einer externen Datei mittels eines C#-Programmes, zu realisieren. Diese externe Datei wird im EPLAN Electric P8 durch eine Export-Funktion erzeugt und entspricht dem AML-Datenformat.

Als Schnittstelle zwischen einer C#-Anwendung und der Entwicklungsumgebung TwinCAT3 bietet die Fa. Beckhoff eine Applikation namens „Automation Interface“ an. Mit dieser ist es möglich, Variablenlisten aber auch Funktionen bzw. Funktionsbausteine mit definiertem Quellcode, welcher der IEC-61131-3 entspricht, zu erstellen. Bei den zu erstellenden Excel-Dateien handelt es sich um Checklisten für eine Validierung von SPS-Hardwarekanälen, welche in der entsprechenden Software verwendet werden.

1.3 Gliederung der Arbeit

Wie in der Einleitung schon erwähnt, wird diese Arbeit in einen theoretischen und in einen praktischen Teil gegliedert. Auf die Umfänge dieser beiden Teile wird in den nachfolgenden Punkten 1.3.1 und 1.3.2 näher eingegangen.

1.3.1 Theoretische Grundlagen

Der theoretische Teil gibt Aufschluss über die Möglichkeiten und deren Umsetzung, um ein SPS-Projekt, inklusive spezifischer Eigenschaften und Funktionen, in TwinCAT 3 mittels eines C#-Programmes zu erstellen. Ebenfalls befasst sich der theoretische Teil mit den Vorgehensmodellen zur Softwareentwicklung, um einen effizienten und qualitativ hochwertigen Entwicklungsprozess durchführen zu können. Für die Durchführung der Softwareentwicklung ist es hilfreich, wenn die zu implementierenden Systeme vorher modelliert werden. Daher befasst sich der theoretische Teil auch mit den Eigenschaften von UML-Diagrammen und deren Anwendung.

1.3.2 Praktische Umsetzung

Für einen effizienten Ablauf des Entwicklungsprozesses wird eine Evaluierung durchgeführt, um ein passendes Vorgehensmodell zur Softwareentwicklung auszuwählen, welches in dieser Arbeit angewendet wird. Der praktische Teil umfasst ebenfalls die Erstellung und Umsetzung von Konzepten und Modellen, wie zum Beispiel eines UML-Diagrammes. Vereint werden diese Konzepte und Modelle mit der Entwicklung eines C#-Programmes, welches den Namen KS-SPS-Project-Creator (KS-SPC) trägt. Die Entwicklung dieses Programmes basiert auf den erstellten UML-Diagrammen.

1.4 Projektstruktur

Zu Beginn des Projektes wird eine einfache Projektstruktur, dargestellt in Abbildung 1-2, festgelegt, um auch ohne Anwendung von aufwendigen Projektmanagementsystemen einen Ablauf und eine Gliederung der Arbeit zu generieren. Die schriftliche Arbeit besteht aus zwei Teilbereichen, der theoretischen Grundlagenforschung und der praktischen Umsetzung. Nach der Auswahl eines geeigneten Vorgehensmodells zur Softwareentwicklung werden die relevanten Themengebiete zur Erstellung einer C#-Anwendung, welche in der Lage ist, SPS-Projekte inkl. den notwendigen untergeordneten Modulen in TwinCAT 3-Anwendungen zu erstellen und anzupassen. Im Zuge der praktischen Umsetzung werden Konzepte, welche z.B. in Form von UML-Diagrammen oder textueller Beschreibung ebenfalls erarbeitet werden, in eine C#-Anwendung implementiert. Dies wird ab Kapitel 3 genauer beschrieben.

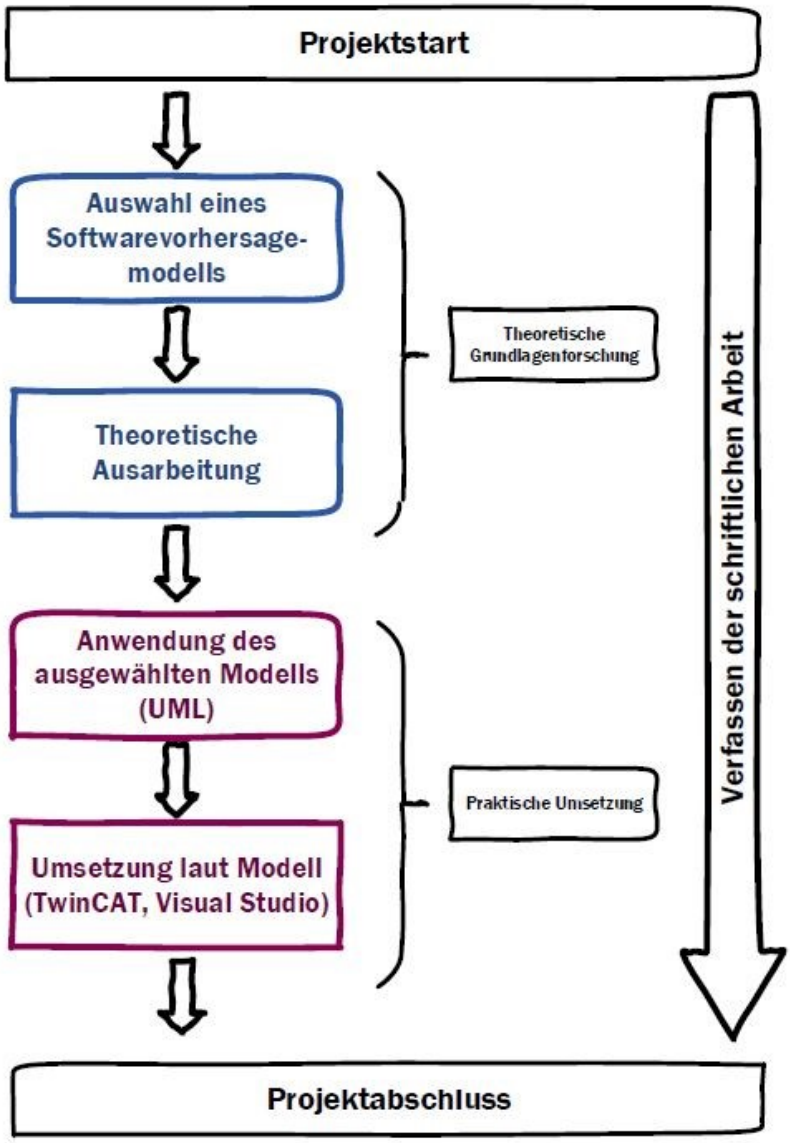

Abbildung 1-2: Projektstruktur, Quelle: Eigene Darstellung

2 ANGEWANDTE SOFTWARE

Für die Umsetzung der zu entwickelnden Applikation kommen bereits bestehende Programme zum Einsatz. Diese sind in den nachfolgenden Unterpunkten aufgelistet und näher beschrieben.

2.1 Visual Studio

Visual Studio (VS) ist eine von Microsoft ins Leben gerufene Entwicklungsumgebung für die Erstellung von Software. Als Programmiersprachen können zum Beispiel C++ oder C# angewandt werden.

2.2 TwinCAT 3

TwinCAT 3 (TC3) ist eine auf Visual Basic basierende Entwicklungsumgebung von Beckhoff für die Programmierung von speicherprogrammierbaren Steuerungen.

2.3 Automation Interface

Das Automation Interface (AI) wurde von Beckhoff entwickelt und dient als Schnittstelle zwischen dem Visual Studio und TwinCAT 3. Mittels AI kann durch Aufrufen von spezifischen Funktionen in Visual Studio, auf das gewählte TwinCAT 3 Projekt zugegriffen und angepasst werden.

2.4 Microsoft Excel

Microsoft Excel ist ein von Microsoft entwickeltes Tabellenkalkulationsprogramm.

2.5 Microsoft Visio

Microsoft Visio ist ein von Microsoft entwickeltes Visualisierungsprogramm. In dieser Arbeit werden alle UML-Diagramme mittels MS-Visio erstellt.

2.6 EPLAN Electric P8

Mittels dieser Software wird die elektrotechnische Dokumentation für Anlangen in der Automatisierungstechnik ermöglicht. Diese umfasst die Erstellung der elektrischen Stromlaufpläne, welche alle Komponenten und deren Beziehungen zueinander darstellt.

3 SOFTWAREENTWICKLUNG

Als Softwareentwicklung wird die Entwicklung von Softwaresystemen bezeichnet, welche im Allgemeinen die Interaktion von Klassen, Methoden und Eigenschaften innerhalb eines Programmes bzw. auch die Beziehungen zwischen Programmen beschreibt, um eine oder mehrere definierte Aufgaben zu lösen. Somit umfasst die Softwareentwicklung den gesamten Entwicklungsprozesses, von der Idee bis hin zur Inbetriebnahme- und Testphase der gewünschten Softwarelösung. [1]

3.1 Softwareentwicklungsprozess

Um den Prozess der Softwareentwicklung möglichst effizient zu gestalten, wird ein geeignetes Verfahren gewählt. Gängige Verfahren sind das V-Modell, Wasserfallmodell, Spiralmodell, Scrum oder Kanban. Diese unterscheiden sich im Ablauf des Prozesses bzw. dahingehend, welche Beziehungen zwischen Stakeholder und Entwickler*innen aufgebaut werden.

3.2 Zur Auswahl stehende Modelle

Für die Auswahl eines geeigneten Modells werden folgende analysiert und bewertet:

- Wasserfallmodell
- V-Modell
- Spiralmodell
- Scrum

In Kapitel 3.4 wird eines dieser Verfahren aufgrund spezieller Kriterien ausgewählt und gemäß seiner Eigenschaften ein Vorgehensmodell konzipiert.

3.3 Analyse von gängigen Verfahren

Um ein geeignetes Vorgehensmodell zur Softwareentwicklung auszuwählen, werden 4 gängige Modelle, begutachtet und anschließend evaluiert, um eine geeignete Entscheidung zu treffen. Die Modelle sind unter anderem in Richtlinien, wie beispielsweise das V-Modell in der VDI 2206, definiert.

[1] Vgl. Brandt-Pook & Kollmeier (2015), S. 1 ff.

3.3.1 Wasserfallmodell

Das in Abbildung 3-1 dargestellte Schema zeigt einen möglichen Aufbau eines Wasserfallmodells als Vorgehensmodell zur Softwareentwicklung.

Bei diesem Vorgehensmodell handelt es sich um ein Modell, welches seine Phasen in eine lineare Struktur gliedert und diese Phasen sequenziell abgearbeitet werden. Das Modell startet mit einer Idee bzw. der Analyse der Idee und schließt mit der Betriebsphase ab. Bei dieser Vorgehensweise wird jede Phase konsequent abgeschlossen bevor eine neue Phase starten kann.[2]

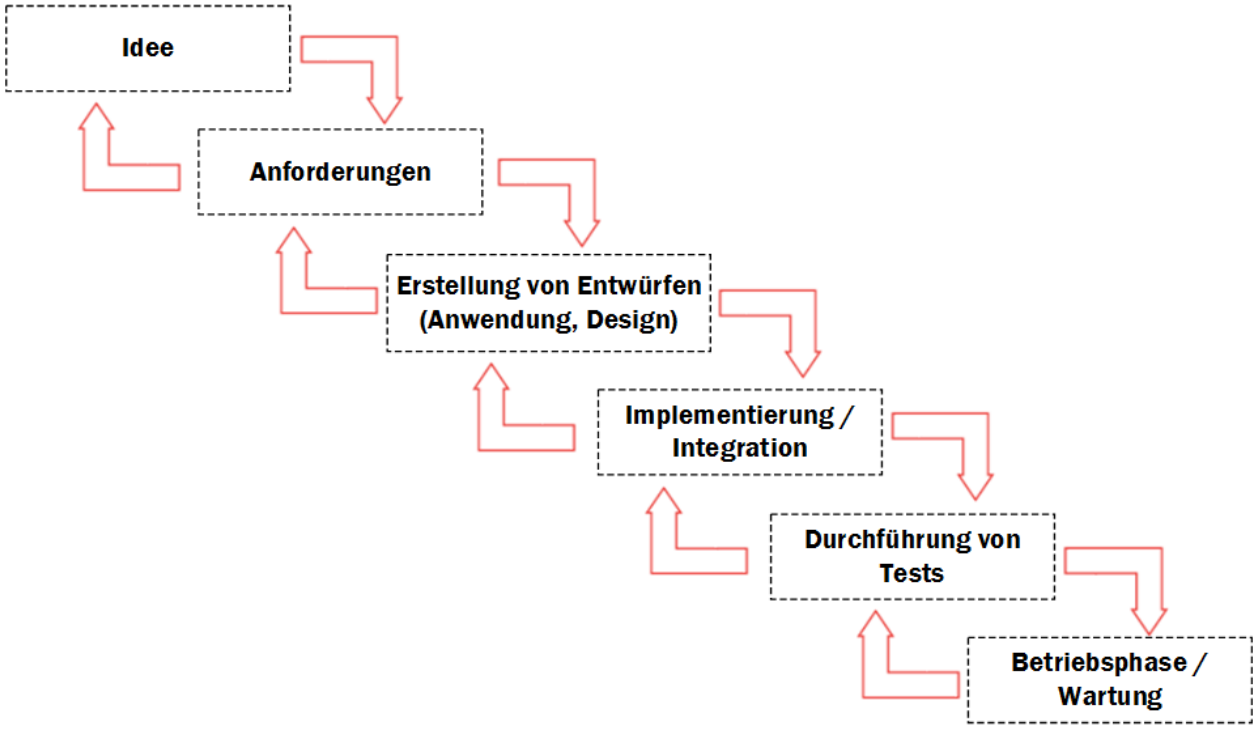

Abbildung 3-1: Beispiel eines Wasserfallmodells, Quelle: Brandt-Pook & Kollmeier (2015) S. 23 (leicht modifiziert).

Vorteile:

Der Vorteil des Wasserfallmodells ist, dass dieses einer klaren Struktur folgt, woraus sich fest definierte Phasen ergeben. Ebenfalls zeigt sich in der Abbildung 3-1, dass dieses Modell sehr übersichtlich und relativ einfach zu verstehen ist.

Nachteile:

Durch den einfachen Aufbau des Wasserfallmodells ergeben sich jedoch auch Nachteile. Komplexe Projekte können zwar mittels dieses Modells umgesetzt werden, jedoch erweisen sich andere Vorgehensmodelle als wesentlich effizienter.

[2] Vgl. Brandt-Pook & Kollmeier (2015), S. 23 f.

3.3.2 V-Model

Das in Abbildung 3-2 dargestellte Schema zeigt einen möglichen Aufbau eines V-Modell als Vorgehensmodell zur Softwareentwicklung.

Das V-Modell ist ein Vorgehensmodell, welches ebenfalls, wie das Wasserfallmodell, einen linearen Ablauf der einzelnen Phasen beschreibt. Nach Abschluss jeder Phase werden mittels Prüfmaßnahmen die Funktionsfähigkeit der einzelnen Komponenten bewertet. Aus sämtlichen Modulen ergeben durch die Integration die Komponenten, welche in der Integrationsphase getestet werden.[3]

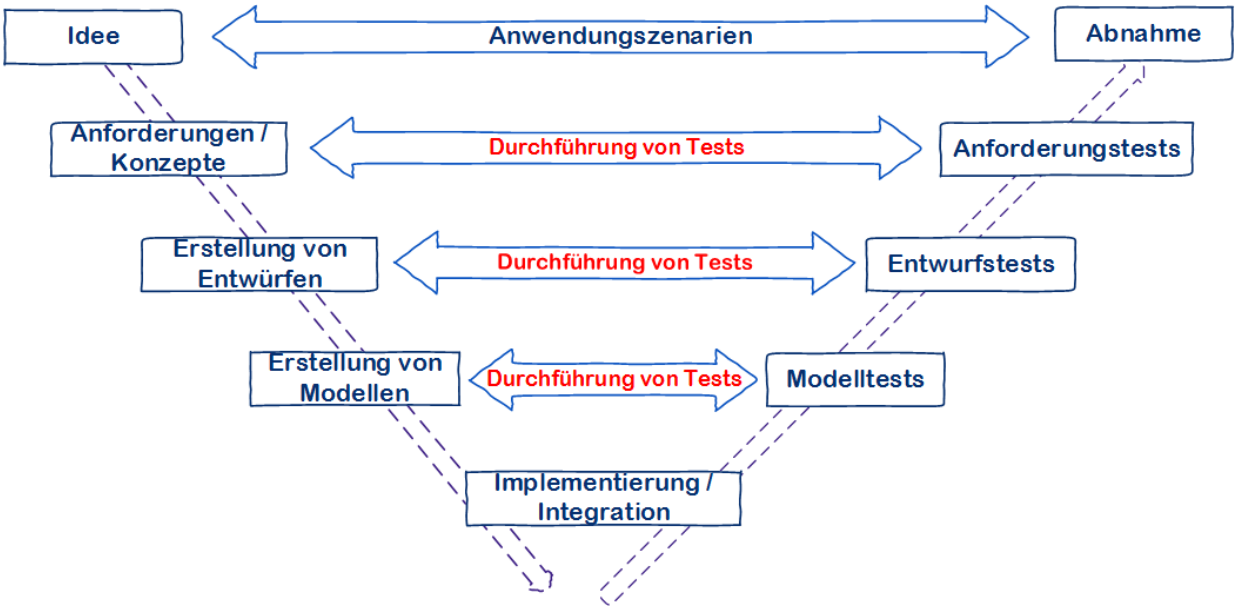

Abbildung 3-2: Beispiel eines V-Modells, Quelle: Kleuker (2018), S. 34 (leicht modifiziert).

Vorteile

Da ein Durchlaufen von Testfällen noch während jeder Phase stattfindet, können so Projektrisiken systematisch kontrolliert und dadurch minimiert werden. Ebenfalls können, beim Eintreten von nicht erwünschten Ergebnissen, für die einzelnen Testfälle, Korrekturmaßnahmen eingeleitet werden.

Nachteile

Durch ein ständiges Durchlaufen von Testfällen in den einzelnen Phasen ist das V-Modell ein sehr zeitaufwendiges und kostspieliges Vorgehensmodell.

[3] Vgl. Kleuker (2018), S. 33 f.

3.3.3 Spiralmodell

Das in Abbildung 3-3 dargestellte Schema zeigt einen möglichen Aufbau eines Spiralmodells als Vorgehensmodell zur Softwareentwicklung.

Das Spiralmodell ist in 4 Zyklen unterteilt.

- Die Zieldefinition, in welcher die Vereinbarung der Ziele bzw. Strategien sowie mögliche Alternativen stattfindet.
- Die Bewertung der Risiken, welche mittels einer Risikoanalyse oder Erstellung eines Prototyps realisiert werden kann.
- Die Umsetzungen, welche zum Beispiel die Erstellung eines Konzeptes, Designs oder Durchführung von Tests, beinhalten.
- Die Planungsphase, in welcher die Planung des nächsten Durchlaufes stattfindet.4

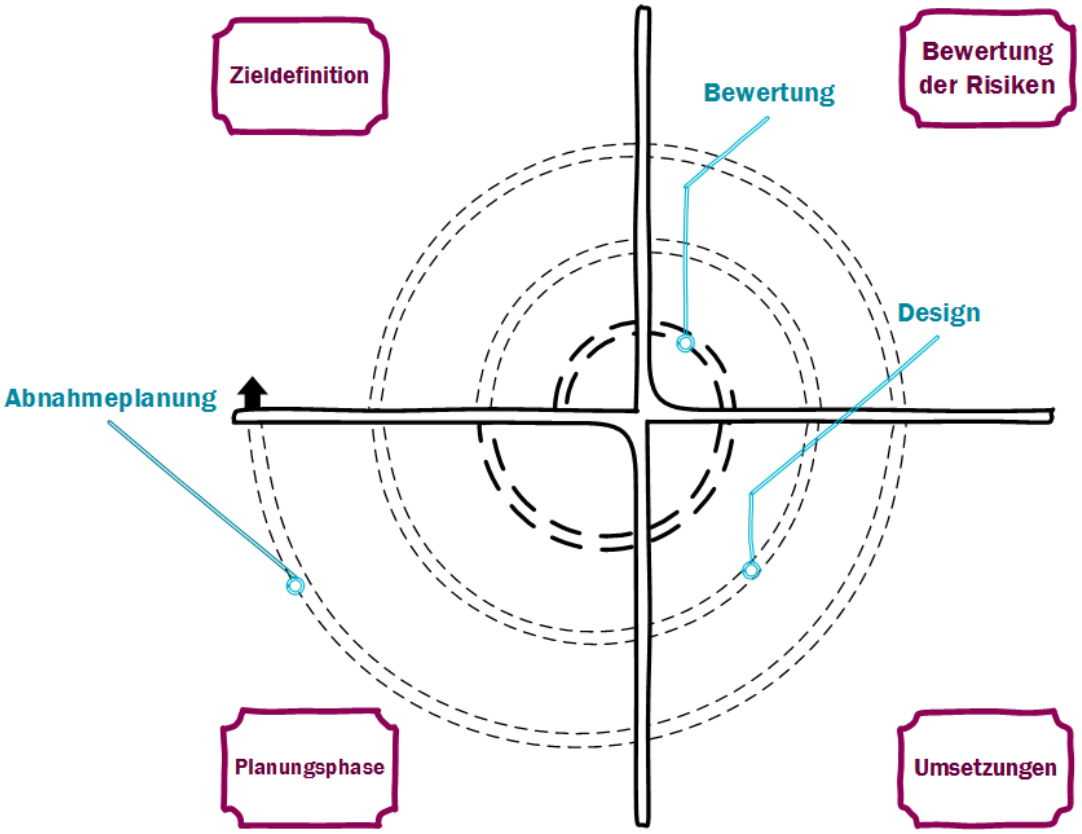

Abbildung 3-3: Beispiel eines Spiralmodels, Quelle: Brandt-Pook & Kollmeier (2015), S. 26 (leicht modifiziert).

Vorteile

Die Einbindung der Stakeholder ist schon zu Beginn eines Projektes möglich, da Phasen wiederholt durchlaufen werden. Ebenfalls findet nach jedem Durchlauf eine Bewertung der Planung und deren Risiken statt.

Nachteile

Um Risiken richtig einschätzen zu können, ist ein hohes Maß an Wissen im Bereich Risikomanagement notwendig. Durch das wiederholte Durchlaufen der Phasen wird die Projektdauer signifikant verlängert.

[4] Vgl. Brandt-Pook & Kollmeier (2015), S. 25 ff.

3.3.4 Scrum

Das in Abbildung 3-4 dargestellte Schema zeigt einen möglichen Aufbau eines Scrum-Modells als Vorgehensmodell zur Softwareentwicklung.

Wird Scrum als Vorgehensmodell verwendet, wird die gesamte Projektlaufzeit in sogenannte Sprints aufgeteilt. Jeder dieser Sprints wird ebenfalls in Phasen aufgeteilt, welche sequentiell abgearbeitet werden. Anforderungen werden zu Beginn nicht zur Gänze definiert, da dies in den Sprints erfolgt. Dadurch ist es möglich, dass auch während des Entwicklungsprozesses Spezifikationen der Stakeholder ermittelt bzw. angepasst werden. Somit wird ausgeschlossen, dass Anforderungen nicht mehr den aktuellen Bedürfnissen der Stakeholder entsprechen. Ebenfalls werden vorherige Sprints analysiert und bewertet. Nach Ende eines Sprints soll ein überprüfbares Ergebnis vorhanden sein.[5]

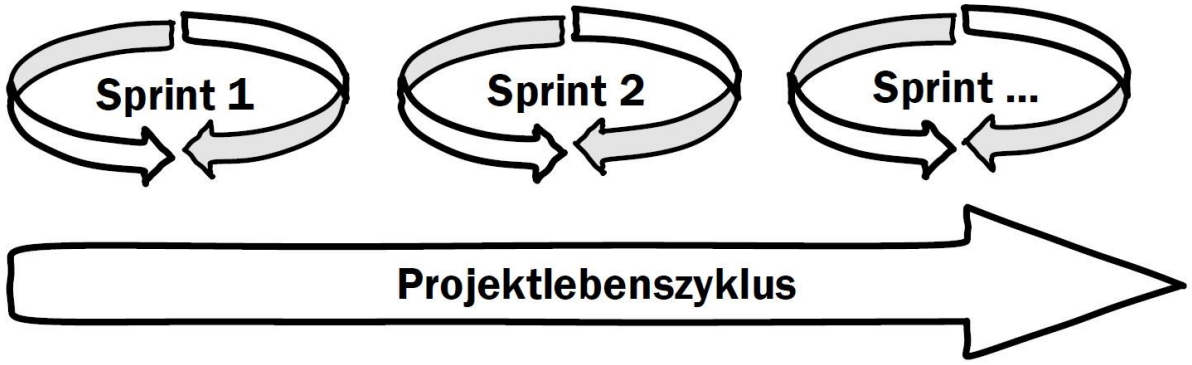

Abbildung 3-4: Beispiel eines Scrum-Models, Quelle: Goll & Hommel (2015), S.87 (leicht modifiziert)

Vorteile

Durch die Aufteilung des Projektes in viele kleine Teilprojekte, den sogenannten Sprints, wird die Möglichkeit eines agileren Einschreitens bei notwendigen Korrekturmaßnahmen geschaffen. Diese Teilprojekte können einzelnen Teams zugewiesen werden. Die interne Organisation der Sprints obliegt allein den jeweiligen Sprint-Teams, was zu einer Erhöhung der Effizienz und Flexibilität in den jeweiligen Sprints führt.

Durch Methoden, wie zum Beispiel dem Planning Poker, ist es möglich mittels eines spielerisches Verfahrens die Aufwendungen der verschiedenen Sprints durch die Teammitglieder zu definieren.[6]

Nachteile

Die Aufteilung des Projektes in viele kleine Teilprojekte bringt jedoch auch den Nachteil den Gesamtüberblick des Projektes zu verlieren. Ebenfalls benötigt die Erstellung und Einteilung der notwendigen Sprints einen sehr hohen Managementaufwand.

[5] Vgl. Goll & Hommel (2015), S. 76 f.

[6] Vgl. Hummel (2011), S. 17 f.

3.4 Auswahl eines Softwareentwicklungsverfahrens

Um ein geeignetes Vorgehensmodell auswählen zu können, müssen die in Punkt 3.3 genannten Modelle evaluiert und einer Bewertung unterzogen werden. Dies geschieht in den nachfolgenden Punkten.

3.4.1 Auswahlkriterien

Folgende Aspekte wurden für die Bewertung und für die Auswahl eines geeigneten Softwareentwicklungsverfahrens in Betracht gezogen:

- Komplexität
  Definiert die Komplexität, somit auch die Einschätzung der zu erwarteten Problemstellungen, eines Projektes
- Umfang
  Der Umfang spiegelt den zu erwarteten Gesamtaufwand für den Softwareentwicklungsprozess des Projektes wider
- Flexibilität
  Unter der Flexibilität wird die individuelle Anpassung der einzelnen Phasen im laufenden Projekt verstanden
- Managementaufwand
  Gibt an, welcher Managementaufwand für die Durchführung eines Projektes aufgewendet werden soll bzw. welcher notwendig sein wird
- Risiken
  Die Risiken eines Projektes ergeben sich aus den zu erwartenden Problemlösungsaufgaben

3.4.2 Erkenntnisse

Um sich für ein geeignetes Vorgehensmodell zu entscheiden, ist dieses Projekt, nach den in Punkt 3.4.1 genannten Kriterien, bewertet. Die Einstufung erfolgt anhand des Schulnotensystems (1-5).

Handelt es sich zum Beispiel um ein Projekt, in welchem ein relativ geringer Managementaufwand betrieben werden soll, kann dieses mit 1 oder 2 definiert werden.

Hinsichtlich des in dieser Arbeit umzusetzenden Softwareprojektes, kann von einer niedrigen bis mittleren Komplexität ausgegangen werden, was zu einer Einstufung des Komplexitätsgrades mit der Note 3 führt. Der Projektumfang und dessen Managementaufwand ist ebenfalls eher gering anzusehen. Auch zu erwartende Risiken können beinahe ausgeschlossen und daher mit der Note 2 bewertet werden. Außerdem ist ein hoher Grad an Flexibilität in diesem Projekt nicht erforderlich.

Nach diesen Aspekten wurden die Bewertungen der Kriterien der KS-SPS-Project-Creator Anwendung getroffen und in der nachfolgenden Tabelle 3-1 dargestellt.

Komplexität	Umfang	Flexibilität	Managementaufwand	Risiken
3	2	2	1	2

Tabelle 3-1: Bewertung der Kriterien, Quelle: Eigene Darstellung

3.4.3 Ergebnis

Das nachfolgende Bewertungsschema in der Abbildung 3-5 zeigt den Bewertungsprozesses und welche Aspekte zur Wahl des geeigneten Modells führen. Bewertet werden die in Kapitel 3.3 erwähnten Vorgehensmodelle, auf Basis der analysierten Kenndaten hinsichtlich auf dessen Anwendung für die Umsetzung des Softwareprojektes in dieser Arbeit.

Ein Modell gilt dann als ideal, wenn die Einschätzungen der Kriterien zum passenden Modell denselben oder abweichend vom Wert ± 1 der Einstufung ergeben.

Bewertungsschema

Kriterien	Modelle				KS-SPS-Project-Creator	Zutreffende Modelle
	Wasserfallmodell	Sprialmodell	V-Modell	Scrum		
Komplexität	2	4	5	5	3	Wasserfallmodell, Spiralmodell
Umfang	3	4	5	5	2	Wasserfallmodell
Flexibilität	2	3	4	4	2	Wasserfallmodell, Spiralmodell
Mangagementaufwand	2	4	3	5	1	Wasserfallmodell
Risiken	2	4	5	3	2	Wasserfallmodell, Scrum

Abbildung 3-5: Bewertungsschema, Quelle: Eigene Darstellung

Anhand des durchgeführten Bewertungsprozesses in Abbildung 3-5 leitet sich das Wasserfallmodell als geeignetes Modell zur Softwareentwicklung für die KS-SPS-Project-Creator Anwendung ab.

3.4.4 Phasen

Folgende Phasen wurden im Laufe des Projektes aufgrund der Wahl des Wasserfallmodells durchlaufen:

- Beschreibung der Idee sowie die Durchführung einer Systemanalyse
- Aufzählung aller Anforderungen
- Erstellung der Entwürfe/Designs
- Umsetzung der Implementierung
- Durchführung von Tests
- Durchlaufen der Betriebsphase

In Kapitel 7 dieser Arbeit findet der eigentliche Softwareentwicklungsprozess sowie die Anwendung des Wasserfallmodells statt. Alle oben genannten Phasen werden dazu sukzessiv durchlaufen. Jeder einzelne Schritt dient der Umsetzung spezieller Arbeitsaufträge. Diese sind lt. Modell zwar nur grob definiert, jedoch in viele Einzelpakete unterteilt.

4 AUTOMATION INTERFACE (AI) ALS SCHNITTSTELLE

Um mittels einer .Net-Anwendung Konfigurationen bzw. Objekte inkl. Quellcode in einem TwinCAT 3 (TC3)-Projekt zu erstellen, bietet Beckhoff als Schnittstelle das sogenannte Automation Interface an. Dieses stellt eine direkte Verbindung zwischen der jeweiligen Anwendung und dem TC3 Projekt her.

4.1 Verwendung des AI in einer .Net-Anwendung

Durch die Verwendung des Automation Interfaces in einer .Net-Anwendung können zum Beispiel Methoden der Visual Studio-API aufgerufen werden, um einen Visual Studio Container zu erstellen und diesen anschließend in der TwinCAT 3 Entwicklungsumgebung auszuführen. Die Visual Studio DTE bietet darüber hinaus weitere Funktionen, wie z.B. den Zugriff auf das Fehlerausgabefenster[7].

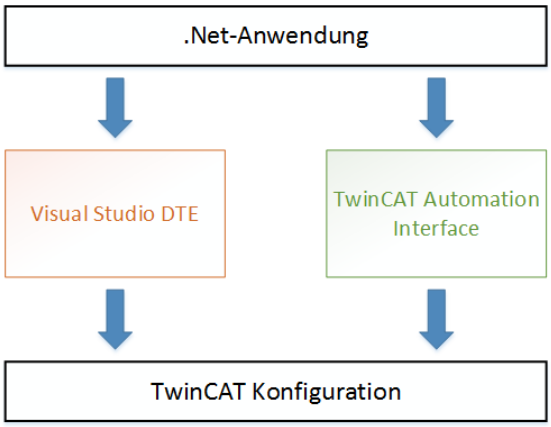

Abbildung 4-1: Schnittstelle Automation Interface, Quelle: Beckhoff Automation (2020), S. 20 (leicht modifiziert).

4.2 Funktionen

Das Automation Interface stellt Klassen und Methoden zur Verfügung, welche es ermöglichen, durch ein Aufrufen von Befehlen in einer .Net-Anwendung, diese in ein TwinCAT 3-Projekt zu übermitteln und auszuführen. Diese werden in zwei Ebenen unterteilt.

4.2.1 Schnittstelle der Ebene 1

Die Ebene 1 bietet zwei Hauptklassen für das Referenzieren und Navigieren in einem TC3-Projekt. Diese sind in der nachfolgenden Tabelle 4-1 aufgelistet.

Hauptklasse	Beschreibung	Verfügbar seit
ITcSysManager	Basisklasse für die Erstellung und Parametrisierung einer TwinCAT-Konfiguration	TwinCAT 2.11
ITcSmTreeItem	Stellt ein Tree Item innerhalb einer TwinCAT-Konfiguration dar	TwinCAT 2.11

Tabelle 4-1: Schnittstellenebene 1, Quelle: Beckhoff Automation (2020), S 113 (leicht modifiziert).

[7]Vgl. Beckhoff Automation (2021), S. 19 f.

4.2.2 Schnittstelle der Ebene 2

In der Ebene 2 findet die Unterteilung in sogenannten Helferklassen statt, welche in Verbindung mit den Hauptklassen in Ebene 1 verwendet werden. In der nachfolgenden Tabelle 4-2 sind drei dieser Helferklassen aufgelistet.

Helferklasse	Beschreibung	Verfügbar seit
ITcPlcLibraryManager	Definiert Methoden und Eigenschaften für die SPS-Bibliotheksverwaltung	TwinCAT 3.1
ITcPlcPou	Definiert Methoden und Eigenschaften für den Umgang mit SPS-POUs	TwinCAT 3.1
ITcPlcDeclaration	Definiert Methoden zum Lesen/Schreiben des Deklarationsbereichs einer SPS-POU	TwinCAT 3.1
…	…	…

Tabelle 4-2: Schnittstellenebene 2, Quelle: Beckhoff Automation (2020), S. 114 (leicht modifiziert).

4.2.3 Auflistung relevanter Methoden

Die ITcSysManger-Klasse kann als zentrale Schnittstelle im Automation Interface angesehen werden. Die nachfolgende Tabelle 4-3 listet einige Methoden auf, welche in dieser Arbeit angewendet wurden:

ITcSysManager-Methoden
NewConfiguration()
OpenConfiguration
SaveConfiguration()
LookupTreeItem()
LinkVariables()
UnlinkVariables()
…

Tabelle 4-3: Schnittstellen TwinCAT 3 XAE, Quelle: Beckhoff Automation (2020), S. 114 (leicht modifiziert).

4.3 Einbindung in C#

Mit den in Punkt 4.2 genannten Funktionen ist es möglich, TwinCAT 3 Befehle, mittels einer C#-Anwendung, auszuführen. Die Einbindung dieser Befehle erfordert jedoch eine Kompatibilität mit der am System installierten TwinCAT-Version.

4.3.1 Anwendung von AI-Funktionen

Um Funktionen des AI in einer C#-Anwendung aufrufen zu können, müssen folgende Kriterien erfüllt werden:

- Kompatibilität mit der installierten TC-Version muss gegeben sein (TC-Version > 3.1)
- Erstellung eines Verweises zum entsprechenden COM-Objekt, abhängig von der verwendeten TC3-Version

Typbibliotheksversionen

Die nachfolgende Tabelle 4-4 zeigt, welche Typbibliothek zur Erstellung eines Verweises zum entsprechenden COM-Objekt verwendet werden:

Typbibliotheksname	Typbibliotheksversion	TwinCAT-Version
Beckhoff TwinCAT XAE Base 3.1	3.1	TwinCAT 3.1 Build 4020.0 und höher

Tabelle 4-4: Typbibliotheksversionen, Quelle: Beckhoff Automation (2020), S. 18 (leicht modifzierit)

4.3.2 Objektorientierte Programmierung

Um den Quellcodeaufbau der Anwendung strukturiert umzusetzen, werden die Eigenschaften der objektorientierten Programmierung angewendet. Dies bedeutet, dass innerhalb eines Programmes, nach Strukturen von sogenannten Klassen, Objekte erzeugt werden. Weitere wesentliche Bereiche der objektorientierten Programmierung sind die Kapselung, Vererbung und der Polymorphismus.

Kapselung, Vererbung und Polymorphismus

Mittels der Kapselung kann der Zugriff der Daten kontrolliert werden. Um den Zugang auf die inneren Daten zu erhalten, werden Schnittstellen bereitgestellt. Bei der Anwendung des Polymorphismus werden Methoden, welche sich in einer Basisklasse befinden, auf sogenannten Subklassen abgeleitet und implementiert. Erst zur Laufzeit wird bei Operationsaufruf entschieden, welche Subklasse aufgerufen wird. Der Gedanke der Vererbung ist jener, dass Gemeinsamkeiten in übergeordneten Klassen definiert werden und die Eigenschaften in der Subklasse verwendet werden können. So muss eine Änderung nur an einem Ort stattfinden. [8]

[8] Vgl. Schäfer (2010), S. 70 ff.

4.3.3 Klassen

Wie schon erwähnt, sind sogenannte Klassen ein essentieller Bestandteil in der objektorientierten Programmierung. Bei der Erstellung von Klassen ist es sinnvoll auf Gemeinsamkeiten von Daten, wie Attribute und Methoden, hinsichtlich der bekannten Objekte zu achten. Daher muss zu Beginn bei der objektorientierten Programmierung eines Systems eine Identifikation der Objekte und dessen Gemeinsamkeiten stattfinden. Die anschließende Zuordnung dieser Objekte zu Klassen wird als Klassifizierung bezeichnet. [9]

Für die Erstellung des KS-SPC werden vereinzelt Klassen aus Beispielprojekten von Beckhoff, welche in C# geschrieben wurden, eingesetzt. Größtenteils werden jedoch neue Klassen entwickelt und in die zu entwickelnde Anwendung implementiert. Somit können Aufgaben in Teilbereiche aufgeteilt und separat in einer Klasse behandelt werden, um zur Lösung der Aufgabenstellung in einer übergeordneten Klasse zusammengeführt und deren Synergien genutzt werden.

4.3.3.1 Verwendung von vorhandenen Klassen

Beckhoff stellt diverse Klassen, welche schon in Beispielprogrammen implementiert wurden, für die Erstellung von TwinCAT-Projekte zur Verfügung. Diese Klassen bieten zum Beispiel die Möglichkeit der Implementierung eines Com-Nachrichtenfilter, der zur Lösung von Blockaden eingesetzt werden kann. Beckhoff bietet diese Beispielprojekte auf ihrer Website zum Download an.

Folgende Klasse, welche in der Tabelle 4-5 aufgelistet ist, wird aus einem Beispielprogramm von Beckhoff übernommen:

Klasse	Beckhoff-Programm	Funktion
MessageFilter	Scripting Container	Com-Nachrichtenfilterung

Tabelle 4-5: Übernommene Klasse von Beckhoff: Quelle: Eigene Darstellung

Die in der Tabelle 4-5 gelistete Klasse ist im Programmcode des KS-SPS-Project-Creators entsprechend kommentiert.

4.3.3.2 Erstellung neuer Klassen

Da sich die zu entwickelnde Anwendung, welche in Folge dieser Arbeit erstellt wird, von den von Beckhoff zum Download bereitgestellten Programmen unterscheidet, müssen die vorhandenen Klassen von Beckhoff bei Bedarf integriert, sowie neue Klassen erstellt werden. Die Funktionen der Klassen werden teilweise mit UML-Diagrammen im Zuge dieser Arbeit erstellt und in die C#-Anwendung implementiert.

[9] Vgl. Lahres, Rayman, & Strich (2021), S. 100 ff.

5 SCHNITTSTELLEN

Die nachfolgenden Punkte behandeln sämtliche Schnittstellen, welche zur Bereitstellung der Informationen an die zu entwickelnde Anwendung dienen. Mittels des Konstruktionsprogrammes EPLAN Electric P8 werden die Informationen über einen Datenexport zur Verfügung gestellt. Das dafür angewendete Dateiformat ist das Automation Markup Language (AML). Zusätzlich werden Informationen, durch den Im- und Export von Dateien im PLCopenXML-Format, an die Anwendung übertragen.

Das folgende Schema in der Abbildung 5-1 zeigt die oben genannten Schnittstellen und dessen Beziehungen zueinander.

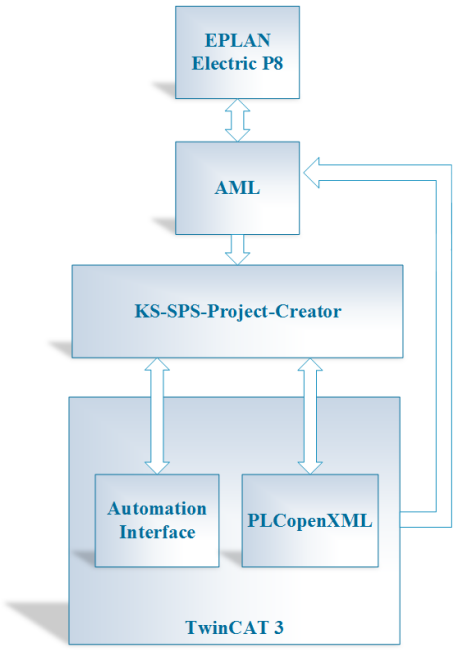

Abbildung 5-1: Schnittstellenbeschreibung, Quelle: Eigene Darstellung

5.1 EPLAN

Prinzipiell bietet EPLAN verschiedene Anwendungsbereiche, wie zum Beispiel für fluidtechnische Anlagen in der Hydraulik oder für Konstruktionen und Dokumentationen von Kabelbäumen, an. Diese Arbeit beschäftigt sich mit dem Export von EPLAN-Daten, welche mit der CAE-Softwarelösung EPLAN Electric P8 projektiert wurden.

5.1.1 EPLAN Electric P8

EPLAN Electric P8 wird verwendet, um Schaltkreise von Anlagen grafisch darzustellen und zu dokumentieren. Um eine AML-Exportdatei für die Entwicklungsumgebung TwinCAT 3 generieren zu können, ist mindestens die Version 2.9 von EPLAN Electric P8 notwendig.

5.1.2 Exporteinstellungen

Um eine Export-Datei in EPLAN erzeugen zu können, muss man den folgenden Programmpfad wählen:

Projektdaten → SPS → Daten exportieren

Mit der Auswahl „Daten exportieren“ öffnet sich das eigentliche Exportmenü, siehe Abbildung 5-2, in welchem die Einstellungen betreffend der zu exportierenden Datei getroffen werden.

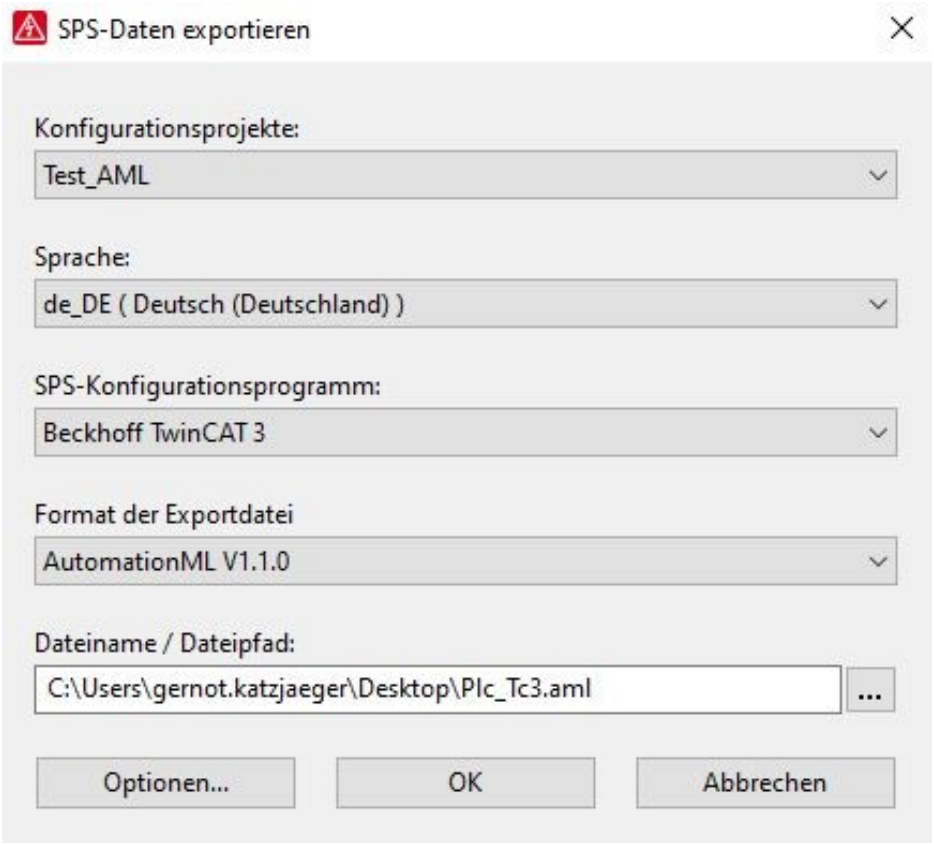

Abbildung 5-2: Exporteinstellungen in EPLAN, Quelle: Eigene Darstellung

Es besteht die Möglichkeit innerhalb eines EPLAN-Projektes mehrere Konfigurationsprojekte zu verwalten und für einen Export zu verwenden. Daher ist es notwendig, alle gewünschten SPS-Teilnehmer, welche sich in einer Exportdatei befinden sollen, demselben Konfigurationsprojekt zuzuweisen. Die Zuweisung des Konfigurationsprojektes erfolgt, wie später in Punkt 5.1.2.2 dargestellt, im SPS-Strukturbaukasten. Bei der Auswahl des SPS-Konfigurationsprogrammes hat man die Möglichkeit, zwischen verschiedenen Entwicklungsumgebungen, wie TwinCAT 2, TwinCAT 3, TIA-Portal oder noch weiteren namenhaften Herstellern zu wählen. Dies beeinflusst die Struktur und in manchen Fällen den Dateityp der exportierten Datei. Da in dieser Arbeit TwinCAT 3 als Zielsystem zur Betrachtung und Umsetzung angewendet wird, muss auch hier TwinCAT 3 als SPS-Konfigurationsprogramm ausgewählt werden. Als Format der Exportdatei wird AutomationML V1.1.0 gewählt.

5.1.2.1 Meta-Daten

Meta-Daten sind Daten, welche große Datenmengen in Form von Stichworten beschreiben. Die Meta-Daten eines Buches sind zum Beispiel sein Autor oder Titel des Buches. Meta-Daten finden ebenfalls im Exportdienst von EPLAN ihre Verwendung. So werden nur die gewünschten und relevanten Daten aller SPS-Teilnehmer in EPLAN definiert, welche für einen späteren Export in Frage kommen. Diese Definition der Meta-Daten wird im nachfolgenden Punkt 5.1.2.2 näher beschrieben.

5.1.2.2 SPS-Strukturbaukasten

Für die Erstellung eines zweckmäßigen Export-Files, welches später für den Import in die .Net-Anwendung dient, ist es zwingend notwendig, den SPS-Strukturbaukasten in EPLAN bestimmungsgemäß mit den zutreffenden Metadaten zu befüllen.

In der nachfolgenden Abbildung 5-3 sind jene Felder mit einer roten Markierung gekennzeichnet, welche zwingend für eine Erstellung einer AML-Datei korrekt befüllt werden müssen.

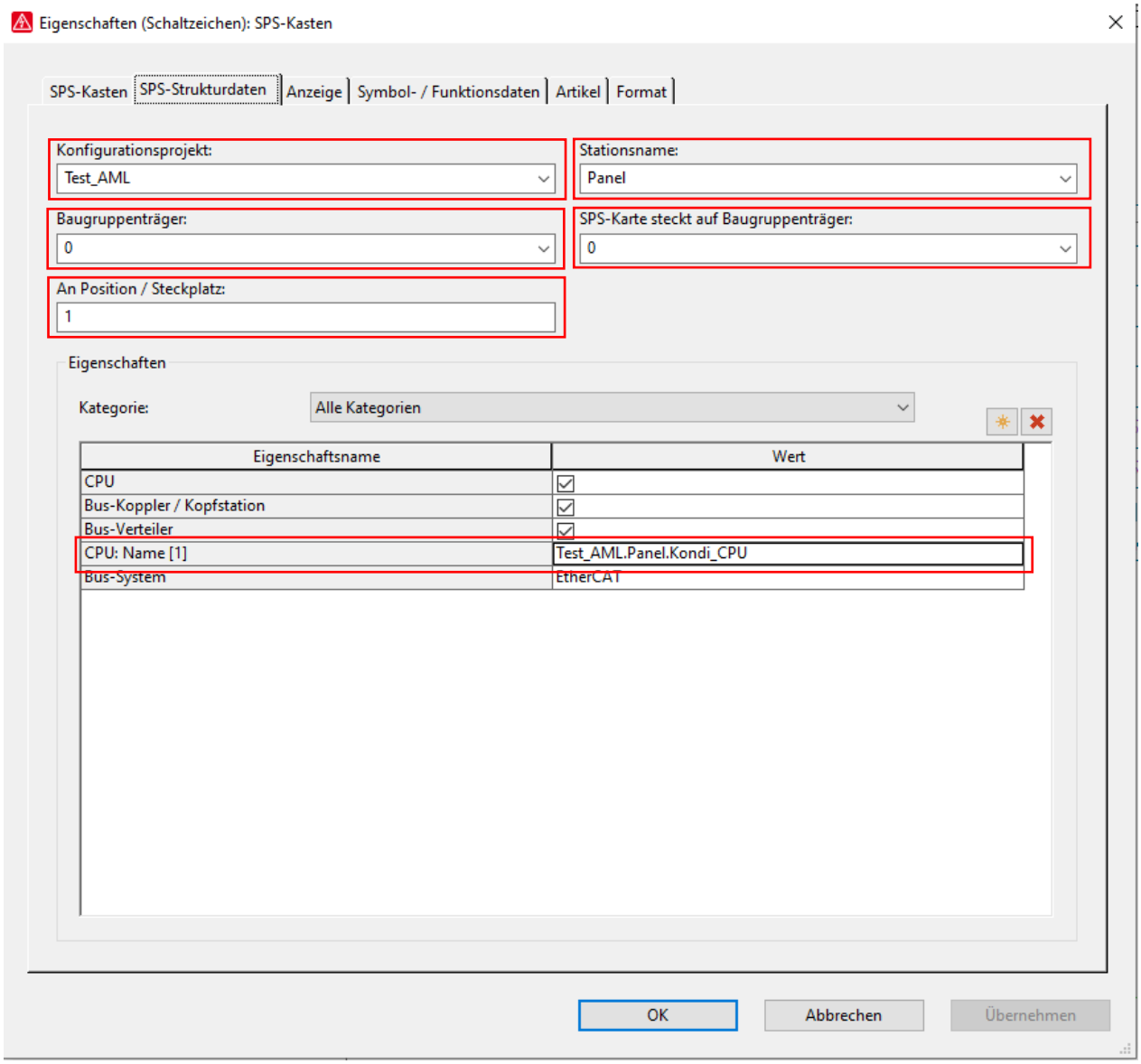

Abbildung 5-3: SPS-Strukturbaukasten, Quelle: Eigene Darstellung

Wie schon erwähnt, handelt es sich in Abbildung 5-3 bei den roten markierten Feldern um Eingabefelder von sogenannten Metadaten. Die Zweckmäßigkeit dieser Felder wird in den nachfolgenden Punkten näher beschrieben:

- Konfigurationsprojekt
  Jeder zu exportierende Teilnehmer muss demselben Konfigurationsobjekt zugewiesen werden. Der Name eines Konfigurationsprojekts ist somit eindeutig. Jeder Teilnehmer kann nur einem Konfigurationsprojekt zugeordnet werden.
- Baugruppenträger
  Der Baugruppenträger gibt an, an welcher Baugruppe sich der SPS-Teilnehmer befindet. Jeder Baugruppenträger kann somit nur einmal in einem Konfigurationsprojekt vorkommen, jedoch mehrere Baugruppenträger in einem EPLAN-Projekt definiert werden. Jede Station (z.B. Cx, Kopper usw.) beginnt mit einem Baugruppenträger.
- An Position / Steckplatz
  Die Nummer des Steckplatzes gibt an, an welchem Steckplatz sich der EtherCAT-Teilnehmer befindet. Jede Nummer eines Steckplatzes kann ebenfalls nur einmal je Konfigurationsprojekt vergeben werden, jedoch vielfach im gesamten EPLAN-Projekt.
- Stationsname
  Der Stationsname ist der Name der zentralen Steuerung und gibt an, zu welcher Station der jeweilige Teilnehmer zugehörig ist. Jedes Konfigurationsprojekt kann jedoch mehrere Stationen besitzen.
- SPS-Karte steckt auf Baugruppe
  Diese Zuweisung gibt zusätzlich an, auf welcher Baugruppe die jeweilige SPS-Karte verortet ist.
- CPU: Name [1]
  Der Name der CPU ergibt sich aus 3 Teilbereichen:
  i. Name des Konfigurationsprojekts
  ii. Bezeichnung des Stationsnamens
  iii. Benennung der CPU

5.2 Automation Markup Language (AML)

Mithilfe von AML ist es möglich, Anlagen vollständig mit den enthaltenen Komponenten, durch einen auf XML-basierten Standard detailliert zu beschrieben. Somit können unterschiedliche Systeme, welche in der Lage sind dieses Format zu verarbeiten, die darin enthaltenen Informationen für Ihre Zwecke nutzen. Da AML in Engineering-Phasen angewendet wird, werden bewusst unvollständige oder fehlerhafte Beschreibungen zugelassen. Um die Effizienz zu steigern, versucht AML jedoch überflüssige Informationen gezielt zu vermeiden. [10]

Für den bidirektionalen Datenaustausch zwischen einem Electronic Computer Aided Design (ECAD)-Tool und einem SPS-Projekt wird in dieser Arbeit AML als Datenaustauschformat angewendet. Mittels EPLAN Electric P8 ist es möglich, Daten in ein solches Format zu exportieren. Ebenfalls bietet TwinCAT 3 die Möglichkeit, dieses Datenformat zu importieren. Um einen hochwertigen Datenexport aus der EPLAN-Software generieren zu können, werden spezifische Eigenschaften, wie z.B. die Betriebsmittelkennzeichnungen, SPS-Kartennamen oder Namen der Baugruppen, verwendet.

5.3 PLCopenXML

PLCopenXML ist ein Datenaustauschformat, welches einerseits für den Datenaustausch zwischen verschiedenen Entwicklungsumgebungen angewendet aber auch bei anderen Anwendungsfällen eingesetzt werden kann. Mittels PLCopenXML ist es zum Beispiel möglich, Programmorganisationsbausteine (Programming Organization Unit, POU) aus einer Entwicklungsumgebung zu exportieren und diese ebenfalls wieder zu importieren. Dies ist plattformunabhängig, was ein paralleles Arbeiten auf mehreren Plattformen ermöglicht. Der Aufbau einer PLCopenXML basiert auf dem Grundkonzept einer XML-Struktur. Ebenfalls unterstützt dieses Format die Weitergabe der Programmiersprachen nach der IEC 61131-3. Der Austausch beschränkt sich nicht nur auf die POU-Ebene, sondern ist ebenfalls auf der Projekt- und Funktionsebene anwendbar. Neben dem Austausch basierend auf den oben genannten Ebenen, ist es ebenfalls möglich, mit PLCopenXML grafische Layouts zu speichern. [11]

[10] Vgl. Drath (2010), S. 25 f.

[11] Vgl. Drath (2010) S. 153 ff.

6 UML (UNIFIED-MODELING-LANGUAGE)

Bei UML handelt es sich um eine grafische Modellierungssprache, welche es ermöglicht Softwaresysteme vereinfacht, mittels Verwendung von Symbolbildern, darstellen zu können. Je nach Anwendungsfall werden unterschiedliche UML-Diagrammtypen angewendet. Die folgende nachfolgende Auflistung nennt einige der gängigen und effizientesten Diagrammtypen:

- Anwendungsfalldiagramm
- Klassendiagramm
- Sequenzdiagramm
- Kommunikationsdiagramm
- Aktivitätsdiagramm
- Zustandsdiagramm
- Komponentendiagramm
- Verteilungsdiagramm

Die nachfolgende Tabelle 6-1 veranschaulicht, in welchen Entwicklungsphasen die oben genannten UML-Diagrammtypen eingesetzt werden können bzw. in welchen der Einsatz die notwendigen Resultate bringt.

UML Diagramm	Entwicklungsphase	Resultat
Anwendungsfalldiagramm	Idee/Systemanalyse, Anforderungen	Verhalten des Systems darstellen
Klassendiagramm	Entwurf/Design, Implementierung	Modellierung von Klassenstrukturen
Sequenzdiagramm	Idee/Systemanalyse, Anforderungen, Entwurf/Design	Definieren von Sequenzabläufen
Kommunikationsdiagramme	Idee/Systemanalyse, Anforderungen, Entwurf/Design	Modellierung des Nachrichtenaustausches
Aktivitätsdiagramm	Idee/Systemanalyse, Entwurf/Design, Test, Implementierung	Verhalten des Systems darstellen
Zustandsdiagramm	Idee/Systemanalyse, Entwurf/Design	Aktionen des Systems darstellen
Komponentendiagramm	Entwurf/Design, Test	Spezifizierung des Systems
Verteilungsdiagramm	Entwurf/Design	Spezifizierung des Systems

Tabelle 6-1: UML-Diagrammtypen für die Entwicklungsphasen, Quelle: Eigene Darstellung

Nachfolgend wird auf jene Diagrammtypen näher eingegangen, die für den Softwareentwicklungsprozess in dieser Arbeit verwendet werden. Um diese Arbeit effizient und zielführend zu gestalten, wurden folgende UML-Diagramme ausgewählt:

- Anwendungsfalldiagramme
- Klassendiagramme
- Aktivitätsdiagramme
- Sequenzdiagramme

6.1 Anwendungsfalldiagramme

Ein Anwendungsfalldiagramm beschreibt spezielle Anwendungs- sowie Benutzungsfälle auf einem hohen Abstraktionsniveau. Durch die Modellierung eines solchen Diagrammes wird ersichtlich, welche Anwendungsfälle vorkommen und wie eine Umsetzung im System erfolgen könnte. Die Verwendung von Notationselementen kann bei Anwendungsfalldiagrammen gering gehalten werden. Diese Eigenschaften machen Anwendungsfalldiagramme zu einem effizienten Werkzeug in den Softwareentwicklungsphasen, wie bei Ideen- und Systemanalysen oder für das Definieren der Anforderungen.[12]

6.1.1 Bestandteile eines Anwendungsfalldiagramms

Der Ablauf, der durch ein Anwendungsfalldiagramm beschrieben werden soll, wird im Wesentlichen durch die in den nachfolgenden Punkten aufgelisteten Elemente dargestellt und steht ebenfalls im MS-Visio zur Verfügung. Teilweise wurden diese Elemente zur Erstellung von Anwendungsfalldiagrammen in dieser Arbeit angewendet.

6.1.1.1 Akteur

Akteure sind die Schnittstellen zwischen externen Benutzer*innen oder Systemen zum eigentlichen modellierenden System. Sie können zum Beispiel Bediener*innen eines Prozesses darstellen, welcher für den Start eines Prozesses eine Eingabe tätigen muss. [13]

6.1.1.2 Anwendungsfall

Anwendungsfälle sind Aktionen, welche zur Lösung der jeweiligen Aufgabenstellung dienen. Anwendungsfälle können zum Beispiel Objekte oder Aufgaben beinhalten. Diese können zu Beginn oder am Ende mit Akteuren aber auch untereinander verknüpft sein. Die Verknüpfung und Anzahl der Anwendungsfälle hängt je nach Komplexität und Umfang der jeweiligen darzustellenden Anwendung ab.[14]

6.1.1.3 Teilsystem

Teilsysteme umfassen alle Anwendungsfälle, welche mit den Akteuren oder untereinander interagieren. Die Anzahl der Teilsysteme hängt, wie bei den Anwendungsfällen von Komplexität und Umfang der jeweiligen darzustellenden Anwendung ab.

6.1.1.4 Assoziation

Eine Assoziation kann eine Beziehung zwischen einem Akteur und einem Anwendungsfall oder zwischen Anwendungsfällen untereinander darstellen. Die Beziehungen können mit einer Multiplizität angegeben werden. Diese gibt an, wie oft und in welche Richtung diese Beziehungen aufgerufen werden können. [15]

---

[12] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 246.

[13] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 250.

[14] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 252.

[15] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 254.

6.1.2 Beispiel anhand von Transportmöglichkeiten

Zur näheren Erklärung eines Anwendungsfalldiagrammes wird in der unten gezeigten Abbildung 6-1 ein derartiges UML-Diagramm für den Personenverkehr modelliert und dargestellt.

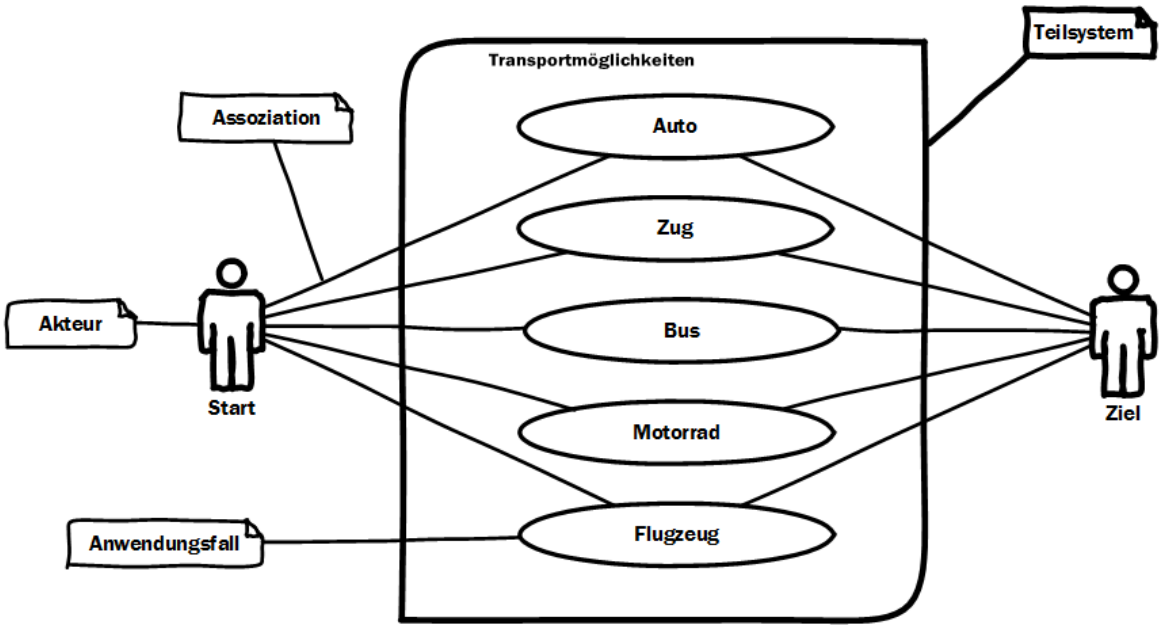

Abbildung 6-1: Beispiel eines Anwendungsfalldiagrammes, Quelle: Eigene Darstellung

Durch dieses Diagramm wird ersichtlich, welche Möglichkeiten es für die Ausführung einer Anwendung gibt oder welche notwendig sind und welche Beziehungen diese zueinander aufweisen. Dieses Beispiel soll zeigen, welche Möglichkeiten einer Person zur Verfügung stehen, um ein Ziel zu erreichen. Natürlich können solche Diagramme weitaus komplexer ausgeführt werden.

6.2 Klassendiagramme

Für die Softwareentwicklung sind Klassendiagramme ein sehr effizientes Werkzeug, um Strukturen oder Architekturen von Systemen und deren Beziehungen zwischen den Methoden und Funktionen innerhalb einer Klasse, sowie deren Schnittstellen nach außen, darstellen zu können. Zu den Aufgaben einer Klasse gehört zum Beispiel die Kapselung von Attributen und Methoden oder die Typisierung von Objekten. Somit schaffen Klassendiagramme nicht nur eine übersichtlichere Darstellung der Struktur des zu entwickelnden Programmes, sondern ermöglichen ebenso eine Zeit- und Fehlerminimierung während der Implementierungsphase. [16]

Ein wesentlicher Vorteil von Klassendiagrammen ist jener, dass die modellierte Struktur inkl. aller definierter Eigenschaften direkt in eine Entwicklungsumgebung implementierbar ist. Visual Studio bietet zum Beispiel die Möglichkeit, Klassendiagramme zu erstellen und erzeugt daraus die entsprechenden Quellcodes

[16] Vgl. Rumpe (2012), S. 34.

6.2.1 Bestandteile eines Klassendiagrammes

Klassendiagramme bestehen im Wesentlichen aus mehreren Klassenbausteinen inkl. bestimmter Attribute und deren Assoziationen miteinander. In den nachfolgenden Punkten werden diese Elemente näher beschrieben und sind Teil der UML-Bibliothek des MS-Visios. Teilweise wurden diese Elemente zur Erstellung von Klassendiagrammen in dieser Arbeit angewendet.[17]

6.2.1.1 Klassen

Eine Klasse beinhaltet alle Eigenschaften für Objekte, wie zum Beispiel Methoden oder Funktionen, welche mit der definierten Struktur der jeweiligen Klasse erzeugt werden können. Ebenfalls werden jeder Klasse spezifische Attribute zugewiesen, welche das Verhalten einer Klasse bestimmen. [18]

6.2.1.2 Mitglieder

Mitglieder sind die Attribute der Klasse, wie zum Beispiel eine Variable oder eine Methode. Folgende Eigenschaften können für Mitglieder definiert werden:

- Mitglieds-Name
  Über den Namen erfolgt der Zugriff auf das jeweilige Mitglied. Prinzipiell gibt es keine Einschränkungen für die Wahl des Namens. Da manche Programmiersprachen jedoch bei Namen durch deren Syntax beschränkt sind, empfiehlt es sich Kleinbuchstaben oder Umlaute zu verwenden. [19]
- Mitglieds-Typ
  Durch die Definition des Typus wird der Datentyp oder eine hinterlegte Struktur die jeweiligen Attribute zugewiesen. Diese können den allgemein bekannten Datentypen entsprechen. Mögliche Datentypen sind zum Beispiel char, int, string, real, boolean oder bei einer Struktur beispielsweise eine Aufzählung von Automarken.[20]
- Multiplizität
  Der Wert der Multiplizität definiert die erlaubte Anzahl an Elementen eines Attributes, welche zur Laufzeit existieren darf. [21]
- Zugriffsmodifikatoren
  Die Attribute oder Methoden in einer Klasse können durch Zugriffsmodifikator den jeweiligen Zugriffsebenen zugewiesen werden. Diese sind in der nachfolgenden Tabelle 6-2 dargestellt.[22]

Vorzeichen	Bedeutung
(+)	public
(-)	private
(#)	protected
(~)	Package

Tabelle 6-2: Vorzeichendefinition, Quelle: Unhelkar (2018), S. 137 (leicht modifiziert).

[17] Vgl. Rumpe (2012), S. 35.

[18] Vgl. Jan van Randen, Bercker, & Fieml (2016), S. 5.

[19] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 46.

[20] Vgl. Jan van Randen, Bercker, & Fieml (2016), S. 6 f.

[21] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 47 f.

[22] Vgl. Unhelkar (2018), S. 137 f.

6.2.1.3 Schnittstellen

Durch die Definition der Schnittstellen können die Beziehungen von Klassen nach außen definiert und dargestellt werden. [23]

6.2.1.4 Assoziationen

Assoziationen beschreiben die Beziehung zwischen zwei Klassen. Diese können mit der Angabe einer Multiplizität näher beschrieben werden. [24]

6.2.1.5 Vererbung

Wird eine Vererbung zwischen zwei Klassen dargestellt, so signalisiert dies, welche Beziehung diese beiden Klassen zueinander aufweisen. Ebenfalls wird ersichtlich, welche das erbende bzw. vererbendete Element ist. [25]

6.2.2 Beispiel anhand einer Teigmischanlage

Um die Umsetzung von UML-Klassendiagrammen in dieser Arbeit vorab zu simulieren, wird als Mustervorlage eine Teigfertigungsanlage durch ein Klassendiagramm modelliert. Dieses Diagramm ist in der Abbildung 6-2 dargestellt und zeigt, welche Methoden und Variablen für die Programmierung einer Software für eine Teigmischanlage notwendig sind und in welcher Beziehung diese zueinander stehen.

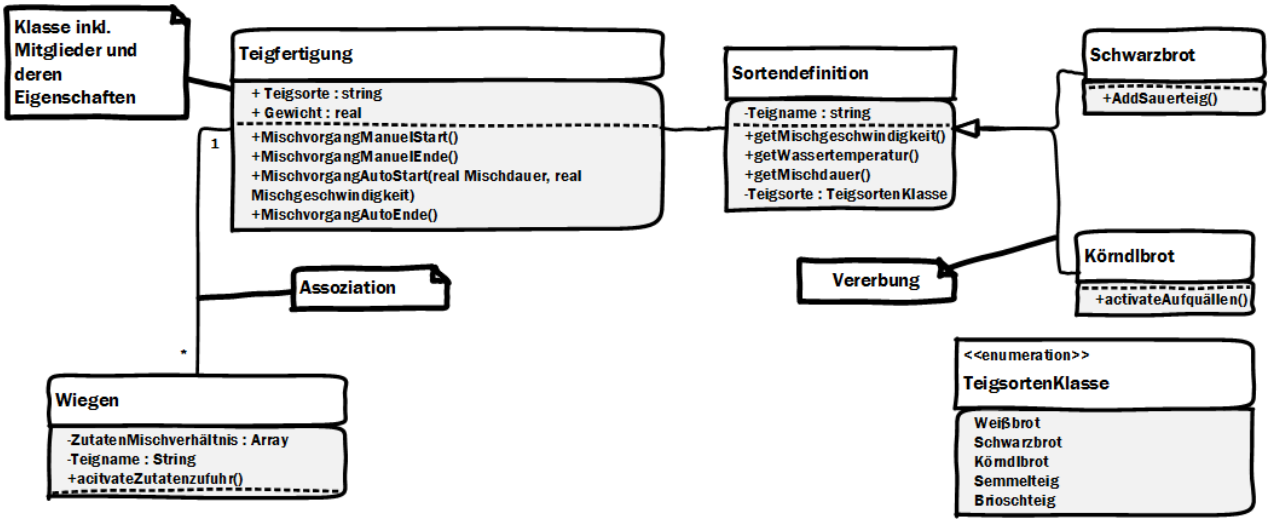

Abbildung 6-2: Beispiel eines Klassendiagrammes, Quelle: Eigene Darstellung

Die Hauptfunktionen der Teigfertigung befinden sich in der Klasse „Teigfertigung“. Die Klasse der Sortendefinition hat die Möglichkeit, je nach Sortenwahl über get-Methoden die spezifischen Werte der Mischdauer, Wassertemperatur und der Mischgeschwindigkeit zu setzen. Durch die Klasse „Wiegen“ erfolgt die Definition der Zutatenzufuhr. In der Aufzählung „TeigsortenKlasse“ befinden sich alle Teigsorten,

[23] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 122 f.

[24] Vgl. Jan van Randen, Bercker, & Fieml (2016), S. 9.

[25] Vgl. Jan van Randen, Bercker, & Fieml (2016), S. 16 f.

welche für die Auswahl zur Verfügung stehen. Da die Brotsorten „Schwarzbrot“ und „Körndlbrot“ spezifische Methoden für die Produktion benötigen, stehen in den Klassen „Schwarzbrot“ und „Körndlbrot“ diese Methoden zur Verfügung.

6.3 Aktivitätsdiagramme

Mithilfe von Aktivitätsdiagrammen kann das spezifische Verhalten von Systemen modelliert werden. Aktivitätsdiagramme können ebenfalls in beinahe allen Phasen des Softwareentwicklungsprozesses eingesetzt werden. Die Modellierung von alternativen Abläufen oder parallelen Aktivitäten, ist mittels eines Aktivitätsdiagrammes auf eine relativ einfache und verständliche Weise möglich.[26]

6.3.1 Bestandteile eines Aktivitätsdiagrammes

Um eine effiziente Modellierung eines Aktivitätsdiagrammes umsetzen zu können, werden verschiedene Elemente eingesetzt. In den nachfolgenden Punkten werden die wesentlichen Elemente zur Erstellung eines Aktivitätsdiagrammes aufgelistet und kurz beschrieben. Diese Elemente stehen im MS-Visio zur Verfügung und wurden teilweise zur Erstellung von Aktivitätsdiagrammen in dieser Arbeit angewendet.

6.3.1.1 Aktion

Basierend auf einer Aktion wird der eigentliche Prozess eines Systems gestartet. Ein Aktivitätsdiagramm kann je nach Umfang und Aufgabenstellung über mehrere Aktionen verfügen. [27]

6.3.1.2 Entscheidung

Mithilfe der Entscheidungselemente werden Bedingungen abgefragt und anhand derer Ergebnisse der weitere Verlauf der Aktivität bestimmt. [28]

6.3.1.3 Zusammenführungsknoten

Mittels solcher Knoten ist es möglich, Zusammenführungen mehrerer Signalflüsse darzustellen.

6.3.1.4 Anfangsknoten

Anfangsknoten stellen den Start einer Aktivität dar.

6.3.1.5 Abschlussknoten

Abschlussknoten signalisieren das Ende einer Aktivität.

6.3.1.6 Verbindungsknoten

Verbindungsknoten führen mehrere parallele Signalflüsse zu einem gesammelten Signalfluss zusammen.

6.3.1.7 Verzweigungen

Verzweigungen teilen einen Signalfluss in mehrere separate Signalflüsse auf.

---

[26] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 268 f.

[27] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 272 f.

[28] Vgl. Kleuker (2018), S. 10 f.

6.3.2 Beispiel anhand einer Benutzerkontoverwaltung

In der nachfolgenden Abbildung 6-3 wird der Ablauf des Zugriffes auf ein Benutzerkonto anhand eines UML-Aktivitätsdiagrammes beschrieben und dargestellt. Um in den Benutzerbereich zu gelangen, müssen Eingaben, wie der Username und das Passwort, getätigt werden. Basierend auf den Funktionen zur Überprüfung des Usernamens und Passwortes wird eine Validierung der Gültigkeit durchgeführt und in den nachfolgenden Entscheidungen mittels dieser Ergebnisse die Zugriffserlaubnis erteilt. Kommt es zu keiner Übereinstimmung des Usernamens und des Passwortes mit den gespeicherten Userdaten, kann die Eingabe der Userdaten wiederholt werden. Mit der Erteilung der Zutrittserlaubnis erfolgt der Sprung in den personalisierten Benutzerbereich.

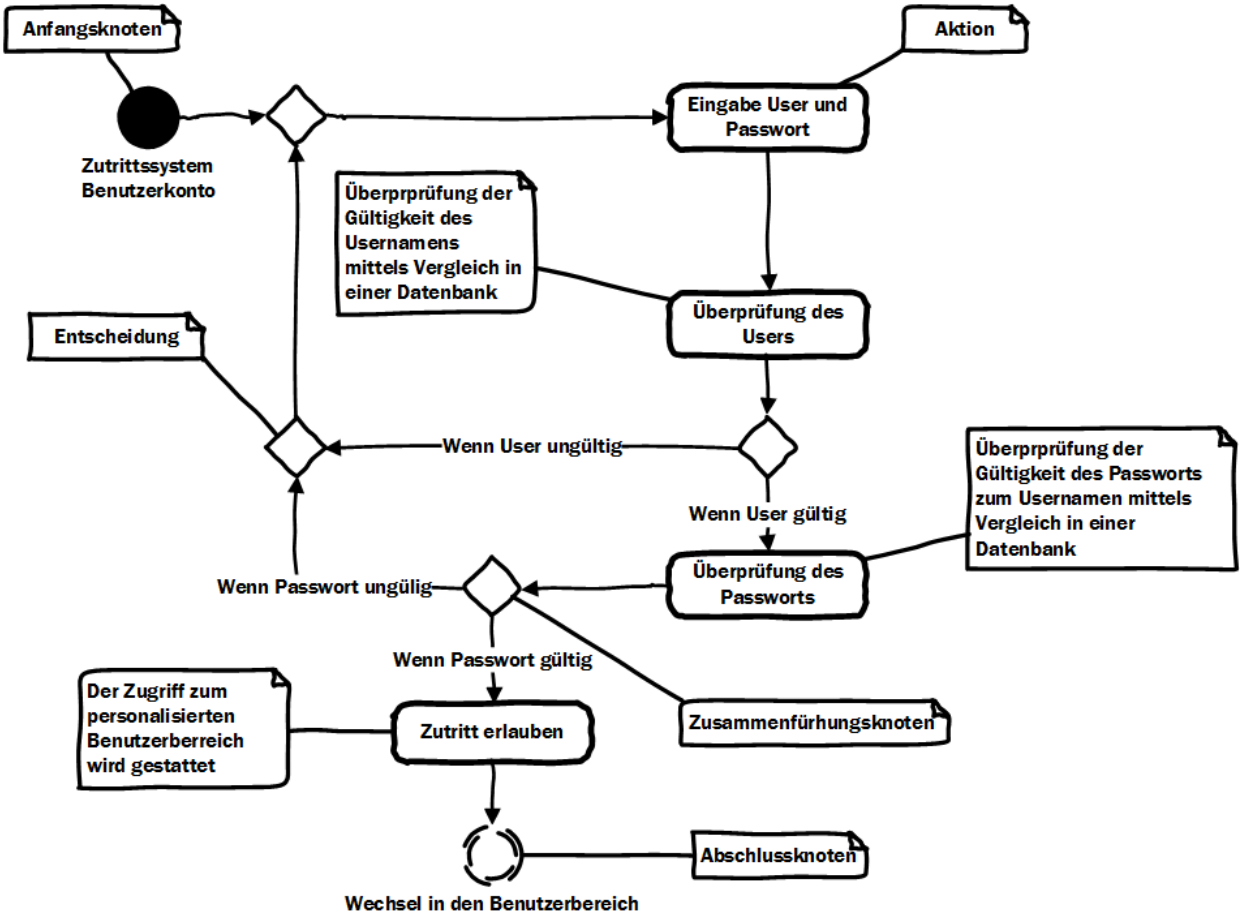

Abbildung 6-3: Beispiel eines Aktivitätsdiagrammes, Quelle: Eigene Darstellung

6.4 Sequenzdiagramme

Sequenzdiagramme weisen Parallelen zu den Aktivitätsdiagrammen auf und werden für die Darstellung von sequentiellen Abläufen angewendet. Dies bedeutet, dass durch die Anwendung von Sequenzdiagrammen aufgezeigt werden kann, welche Methoden von Objekten in welcher Reihenfolge ausgeführt werden. Bei einem Methodenaufruf wird in einem asynchronen sowie in einen synchronen Aufruf unterschieden. Erfolgt der Methodenaufruf synchron, so wartet diese Methode bis ein Ergebnis rückgemeldet wird. Konträr zum synchronen Aufruf muss der asynchrone Aufruf nicht auf eine Rückmeldung warten. Mithilfe dieser Diagramme können Abläufe sehr effizient und verständlich dargestellt werden. Die Anwendung von Sequenzdiagrammen ist durch die oben genannten Eigenschaften fast in jeder Phase der Softwareentwicklung möglich.[29]

6.4.1 Bestandteile eines Sequenzdiagrammes

Um ein Sequenzdiagramm darstellen zu können, bietet MS-Visio diverse Elemente zur Umsetzung an, welche auch zur Erstellung von Sequenzdiagrammen in dieser Arbeit angewendet werden. Diese Elemente werden in den nachfolgenden Punkten näher beschrieben.

6.4.1.1 Lebenslinie

Eine Lebenslinie stellt Darsteller*innen in der Sequenz dar. Alle Darsteller*innen werden horizontal nebeneinander im Sequenzdiagramm dargestellt. Lebenslinien werden in folgende zwei Gruppen unterteilt[30]:

- Akteurlebenslinie
  Stellt externe Teilnehmer*innen, wie zum Beispiel Kunden*innen als Darsteller*innen dar.
- Objektlebenslinie
  Stellt ein Objekt oder Komponenten innerhalb des Systems, wie zum Beispiel ein Bezahlsystem dar.

6.4.1.2 Aktivierung

Die Aktivierung erfolgt mit einem Balken an der vertikalen Lebenslinie und stellt die aktive bzw. passive Zeit einer Interaktion dar.[31]

6.4.1.3 Nachrichten

Durch einen Methodenaufruf, einer sogenannten Nachricht, wird das Ausführen einer Handlung von einer Lebenslinie zur Anderen signalisiert. Bei asynchronen Methodenaufrufen erfolgt eine Weiterführung der nachfolgenden Sequenzen ohne auf eine Rückmeldung der gesendeten Nachricht zu warten. Wird ein synchroner Aufruf darstellt, so ist eine Rückmeldung, eine sogenannte Antwortnachricht, notwendig um weitere Sequenzen abzuarbeiten. Eine solche Nachricht kann zum Beispiel die Eingabe von Zugangsdaten eines Kunden auf einer Kundenwebsite darstellen. [32]

---

[29] Vgl. Jan van Randen, Bercker, & Fieml (2016), S. 91 f.

[30] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 414 ff.

[31] Vgl. Rumpe (2012), S. 68 f.

[32] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 417 ff.

6.4.1.4 Kombiniertes Fragment

In einem kombinierten Fragment können Bedingungen abgefragt und basierend auf diesen Ergebnissen Methoden ausgeführt werden. Ein kombiniertes Fragment kann durch die Definition des Interaktions-Operators, wie zum Beispiel „alt“ für Alternative oder „opt“ für Option, näher spezifiziert werden.[33]

6.4.2 Beispiel anhand eines Sequenzdiagrammes

In der nachfolgenden Abbildung 6-4 wird ein Zutrittssystem einer Firma anhand eines UML-Sequenzdiagrammes modelliert und dargestellt. Die Sequenz wird durch den Akteur, also den Mitarbeiter*innen, eingeleitet, welcher den Ausweis zur Freigabe an das Zutrittspanel hält. Infolgedessen wird eine Bedingung abgefragt, welche die Gültigkeit des Ausweises überprüft und anhand dieses Ergebnisses das Öffnen der Eingangstür durchführt oder verschlossen hält. Bei einer genaueren Betrachtungsweise wird deutlich, dass die Aktivierung der Lebenslinie der Mittarbeiter*innen nur für den Zeitraum während des Zuführens der Mitarbeiterkarte aktiv ist. Zutrittspanel und Server verfügen über den gleichen Aktivierungszeitrahmen da die Verarbeitung und Ausgabe synchron abläuft. Das Öffnen der Eingangstür ist hingegen nur aktiv, wenn die Freigabe des Zutrittspanels erfolgt.

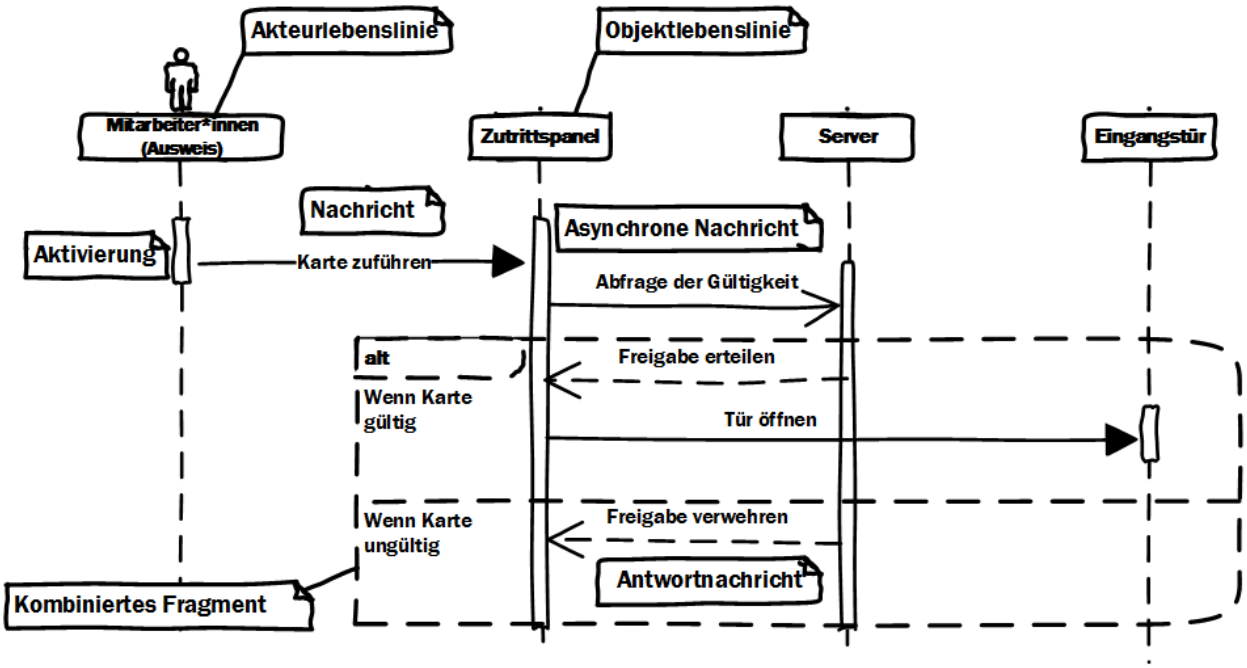

Abbildung 6-4: Beispiel eines Sequenzdiagrammes, Quelle: Eigene Darstellung

[33] Vgl. Kecher, Salvanos, & Hoffmann-Elbern (2018), S. 431 ff.

7 ANWENDUNG DES AUSGEWÄHLTEN VORGEHENSMODELLS

In den nachfolgenden Punkten findet die systematische Abarbeitung der einzelnen Phasen des Wasserfallmodells zur Entwicklung der .Net-Anwendung statt. In jeder dieser nachfolgenden Phasen werden verschiedene Werkzeuge zur Softwareentwicklung angewendet und näher beschrieben. Der Prozess startet mit der Systemanalyse, in welcher alle Ideen und Wünsche an das System gesammelt werden und endet mit der Betriebsphase, in welcher der Prototyp der Anwendung seine eigentliche Aufgabe aufnimmt.

7.1 Idee/Systemanalyse

Die Idee dieser Masterarbeit ist es, mittels eines Programmes eine automatisierte Erstellung eines SPS-Projektes zu ermöglichen. Ebenfalls soll es möglich sein, I/O-Checklisten in Excel zu befüllen. Um dies zu veranschaulichen, ist die Grundidee eines UML-Sequenzdiagrammes modelliert und in der nachfolgenden Abbildung 7-1 dargestellt.

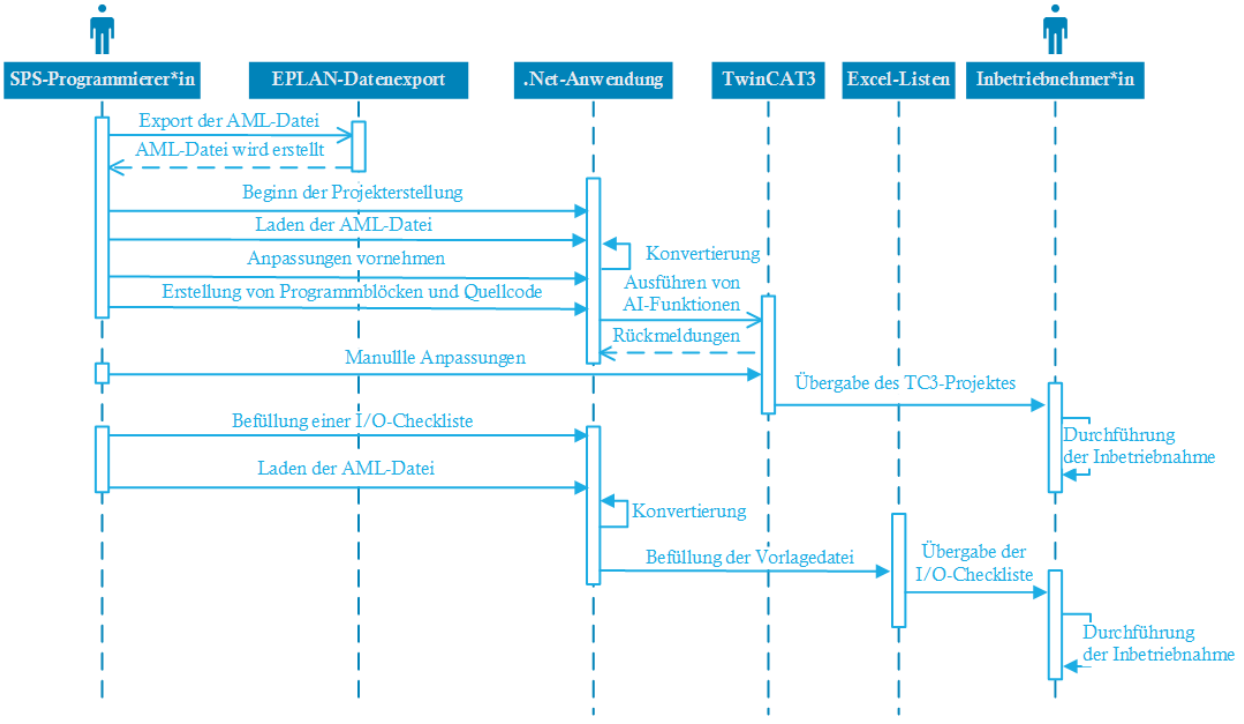

Abbildung 7-1: Schema der Idee, Quelle: Eigene Darstellung

Das oben dargestellte Diagramm zeigt, dass der*die SPS-Entwickler*in für die Erstellung der Software und der I/O-Checklisten zuständig ist. Sobald die Erstellung abgeschlossen ist, werden diese an den*die Inbetriebnehmer*in übergeben. Dieser führt dann die Inbetriebnahme anhand der übergebenen Software und der I/O-Checkliste durch.

Um die in Abbildung 7-1 dargestellte Idee zu realisieren, ist eine detaillierte Spezifikation weiterer Anwendungen und Abläufe notwendig. In den nachfolgenden Punkten werden die Hauptaufgaben mittels UML-Diagrammen modelliert und dargestellt.

7.1.1 Bereichszuweisung

Das folgende Diagramm in Abbildung 7-2 stellt die Beziehungen und Aufgaben der Entwickler*innen und der Konstrukteure*innen dar. Zu beachten ist, dass für die Festlegung der Strukturen sowohl der*die Softwareentwickler*in als auch der*die Hardwareplaner*in Beziehungen aufweisen und somit Daten austauschen müssen.

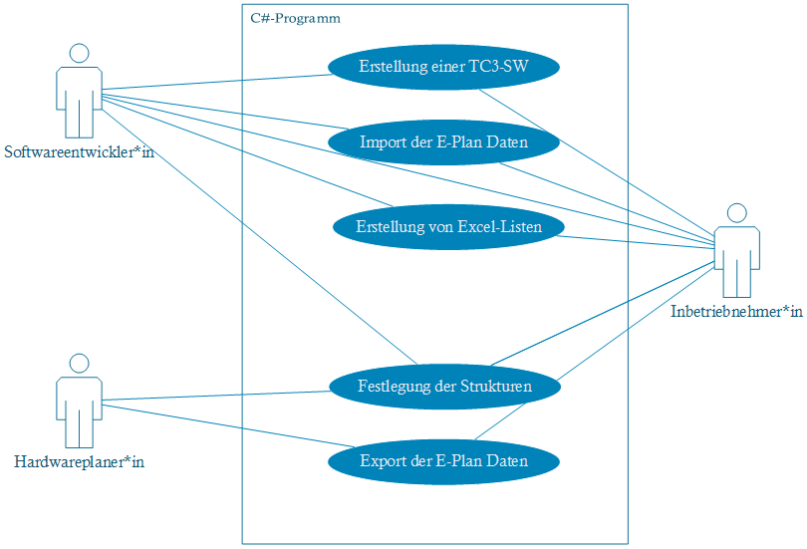

Abbildung 7-2: Anwendungsdiagramm der Idee, Quelle: Eigene Darstellung

7.1.2 TC3-Konfigurationserstellung

Ziel ist es, die Erstellung einer Hardwarekonfiguration im TwinCAT 3 zu realisieren. Durch die Verwendung des Automation Interfaces wird dies möglich gemacht. Es bedarf jedoch eines definierten Anwendungsablaufes. So muss zuerst die Entwicklungsumgebung TwinCAT 3 geöffnet und ein neues Projekt angelegt werden. Ebenfalls muss die AML-Datei geladen und geprüft werden. Gegebenenfalls können vor einer Erstellung auch Anpassungen vorgenommen werden. Die nachfolgende Abbildung 7-3 soll dies veranschaulichen:

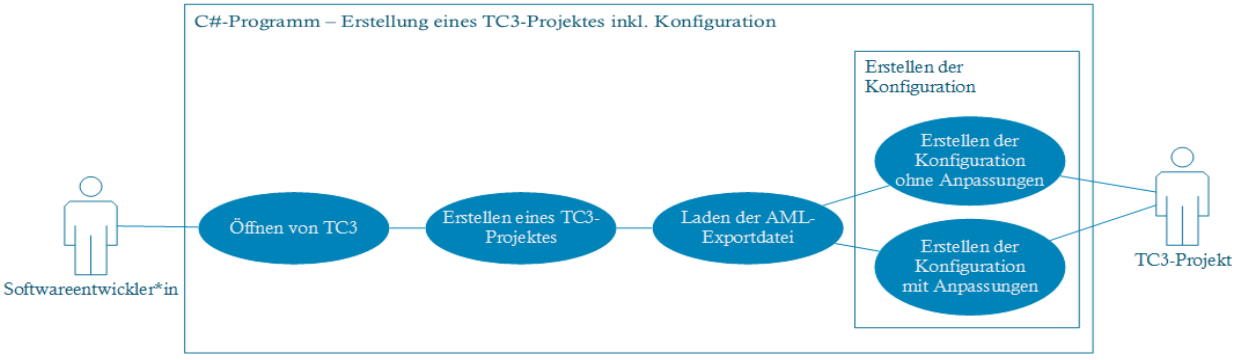

Abbildung 7-3: Anwendungsfalldiagramm Projekt- und Konfigurationserstellung, Quelle: Eigene Darstellung

7.1.3 Variablen- POUs- und Quellcodeerstellung

Um eine effiziente Erstellung eines SPS-Programmes zu ermöglich, soll die Erstellung von Variablenlisten, Programmorganisationseinheiten (POUs) aber auch die Erstellung von Quellcodes teilweise automatisiert durchgeführt werden. Bevor ein Unterprogramm erstellt werden kann, müssen die jeweiligen Eigenschaften definiert werden, wie zum Beispiel die Deklaration der Variablen, oder die logische Verknüpfung von Variablen. Die Funktionalität dieses Systems wird in der nachfolgenden Abbildung 7-4, anhand eines UML-Anwendungsfalldiagrammes dargestellt:

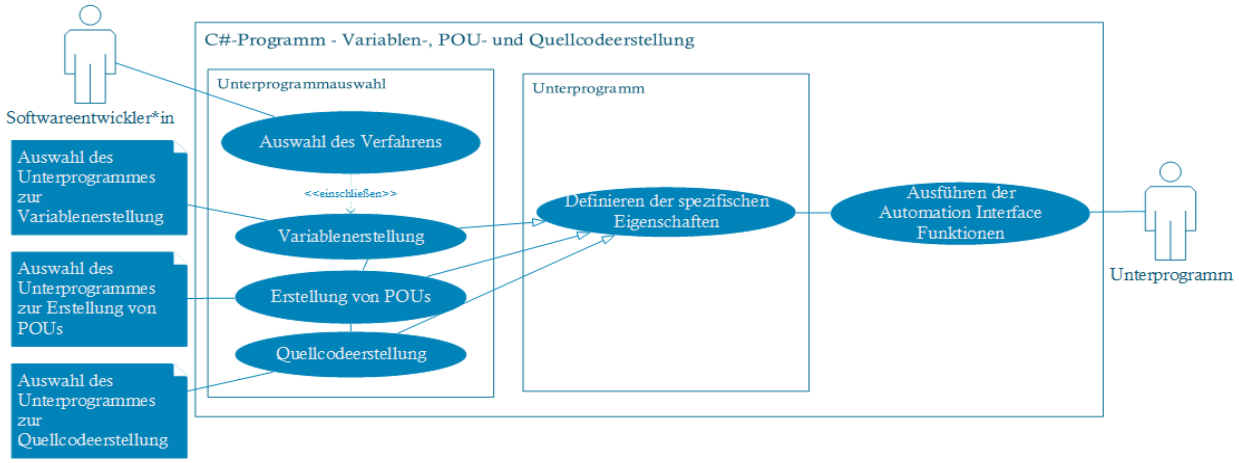

Abbildung 7-4: Anwendungsdiagramm Quellcodeerstellung, Quelle: Eigene Darstellung

7.1.4 Befüllung einer I/O-Checkliste

Für die Umsetzung einer automatisierten Befüllung einer Excel-Liste mit sämtlichen relevanten Hardwaredaten eines SPS-Projektes, müssen diverse Anwendungsfälle sukzessiv abgearbeitet werden. Zu Beginn muss die entsprechende AML-Datei in der .Net-Anwendung geladen werden. Der Import erzeugt eine Auflistung der aktuellen Hardwarekonfiguration. Die Beschreibung sämtlicher Module und deren Kanäle können anschließend in eine Excel-Datei übertragen werden. Das nachfolgenden UML-Anwendungsfalldiagramm in Abbildung 7-5 stellt dieses Zusammenhänge dar.

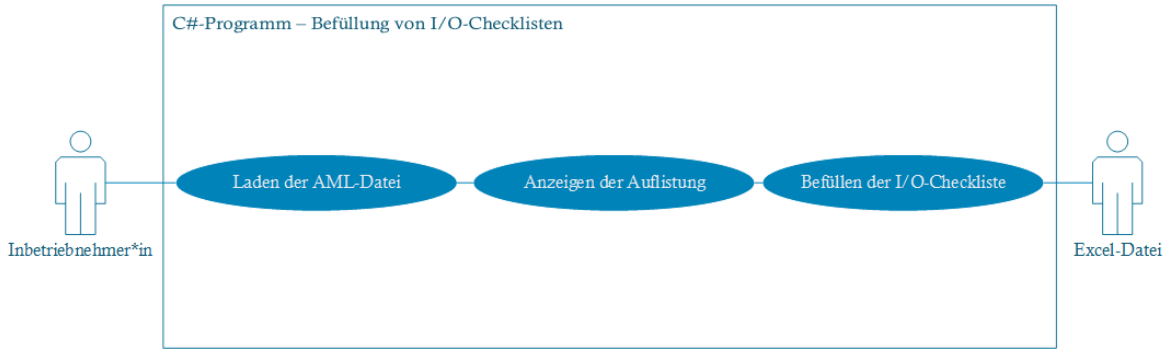

Abbildung 7-5: Anwendungsdiagramm Befüllung von I/O-Checklisten, Quelle: Eigene Darstellung

7.2 Anforderungen

In der zweiten Phase der Softwareentwicklung erfolgt die Definition der notwendigen Anforderungen an verschiedene Bereiche. Für folgende Bereiche werden Anforderungen gestellt und in den nachfolgenden Punkten näher spezifiziert:

- Anforderungen an die .Net-Anwendung
- Anforderungen an EPLAN
- Anforderungen an TwinCAT 3
- Anforderungen für eine automatisierte Quellcodegenerierung

7.2.1 Anforderungen an die .Net-Anwendung (KS-SPS-Project-Creator)

Die Abbildung 7-6 soll veranschaulichen, welche Anforderungen an die zu programmierende Anwendung gestellt werden. Dieses Schema zeigt, dass die C#-Anwendung der zentrale Mittelpunkt der automatisierten Projekterstellung ist und stellt sämtliche externe Schnittstellen dar. Die .Net-Anwendung ist in der Lage, eine AML-Datei einzulesen und diese für eine Weiterverarbeitung der Daten zu konvertieren. Mittels dieser konvertierten Daten wird die Erstellung einer Hardwarekonfiguration ermöglicht, welche vor Erstellung noch angepasst werden kann. Ebenfalls können durch die konvertierten Daten globale Variablenlisten erstellt werden, die für eine Verlinkung der Kanäle der jeweiligen Hardwaremodule genutzt werden können. Die Anwendung ermöglicht ebenfalls einen Export der Hardwarekonfiguration inkl. aller Kanäle und deren Kanalnamen in ein Excel-File.

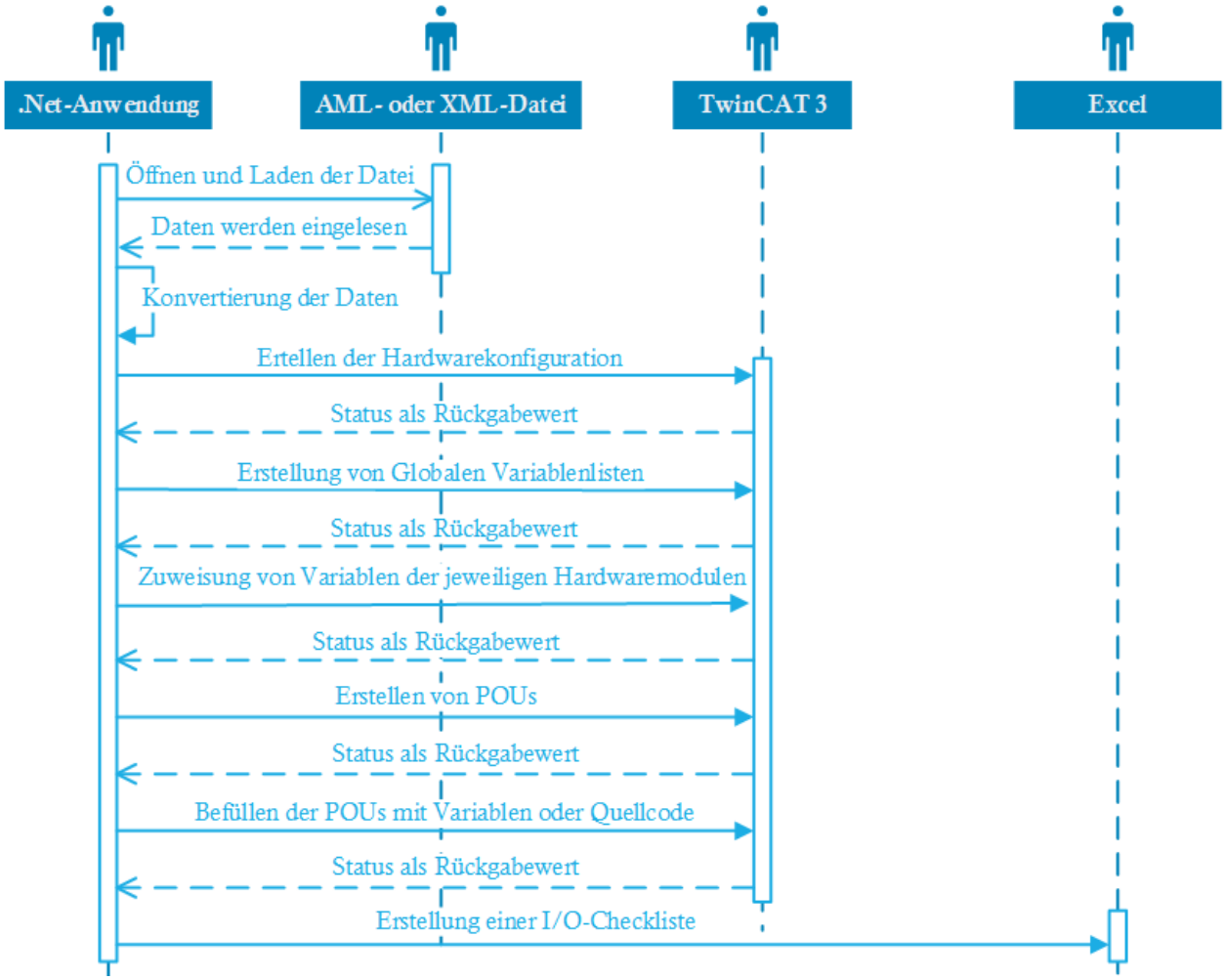

Abbildung 7-6: Sequenzdiagramm der Anforderungen an das C#-Programm, Quelle: Eigene Darstellung

7.2.2 Anforderungen an EPLAN Electric P8

In Kapitel 5.1 wurde bereits auf die EPLAN-Software als Schnittstelle zum KS-SPC näher eingegangen. Das folgende UML-Sequenzdiagramm, welches in Abbildung 7-7 dargestellt ist, zeigt detailliert die Anforderungen und Abläufe die an EPLAN Electric P8 sowie an dessen Schnittstellen erforderlich sind beziehungsweise gestellt werden. Für eine effiziente und fehlerfreie Erstellung eines zu exportierenden EPLAN-Projektes ist es essenziell, eine Artikeldatenbank für alle verwendbaren Komponenten zu führen und diese mit sämtlichen Herstellerdaten zu befüllen. Diese werden beinahe von allen namenhaften Herstellern als Importdatei bereits zur Verfügung gestellt. Um gewünschte Daten mittels eines Exports zu erhalten, müssen SPS-Komponenten diverse Meta-Daten, wie der zugehörigen Baugruppe oder dem Steckplatz auf dieser Baugruppe, zugewiesen werden. Zu beachten ist ebenfalls, dass ein Export für die Entwicklungsumgebung TwinCAT 3 in AML-Format nur ab einer EPLAN Electric P8-Version höher 2.9 zur Verfügung steht.

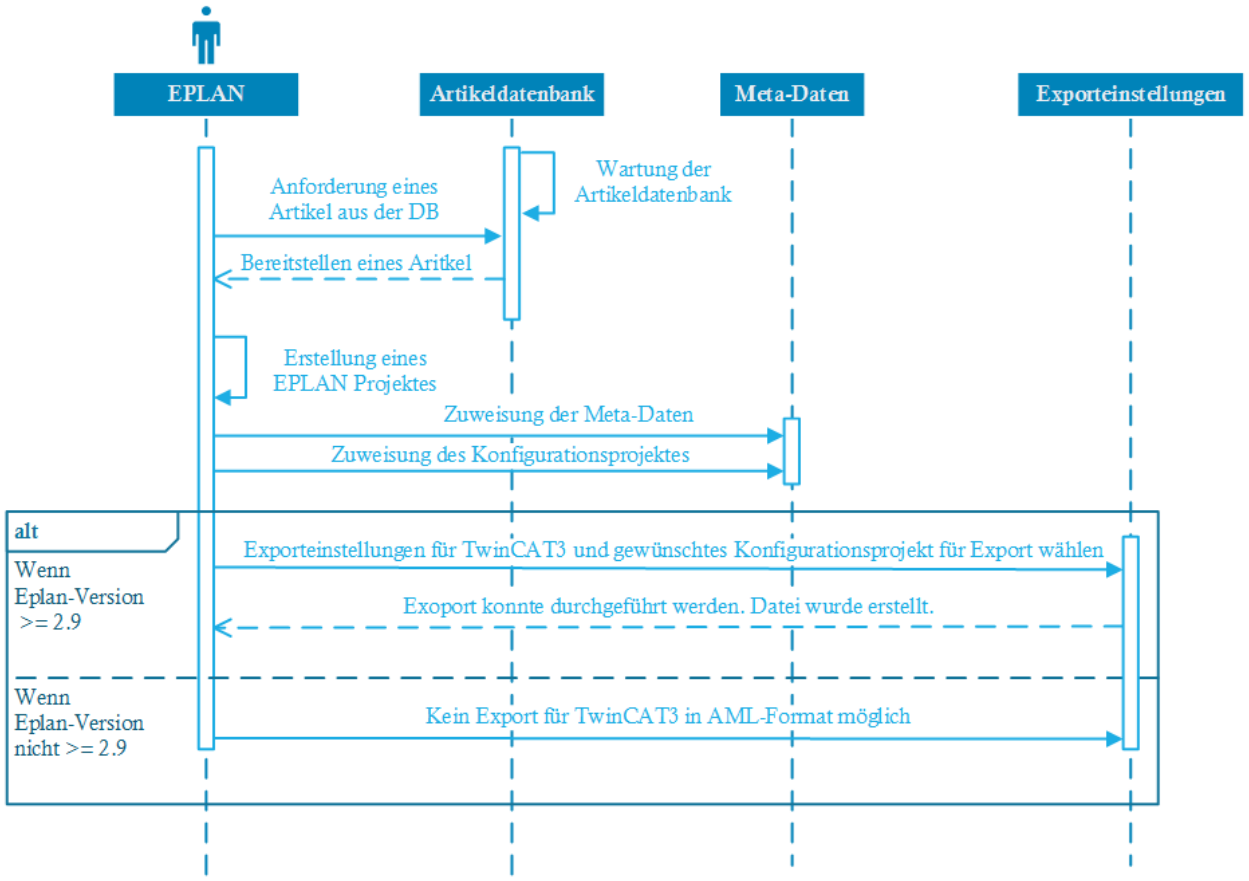

Abbildung 7-7: Sequenzdiagramm der Anforderungen an EPLAN, Quelle: Eigene Darstellung

Da die Konstruktion der EPLAN-Dateien meist von separaten Abteilungen durchgeführt wird, ist es essenziell, dass die Anforderungen an EPLAN von der SPS-Entwicklungsabteilung an die EPLAN-Konstruktionsabteilung klar kommuniziert werden.

7.2.3 Anforderungen an TwinCAT 3

TwinCAT 3 bietet grundsätzlich mehrere Möglichkeiten, um automatisiert Hardwarekonfigurationen, aber auch weitere SPS-Projektspezifikationen, wie zum Beispiel Variablenlisten, zu erstellen. So ist es möglich, die exportierte AML-Datei, welche auch für die Projekterstellung im KS-SPC genutzt wird, manuell über die AML-DataExchange-Funktion in TwinCAT 3 zu importieren. TwinCAT bietet hier auch die Möglichkeit, diverse Eigenschaften dieser Daten vor Erstellung der eigentlichen Projektdaten anzupassen. Der Unterschied bei der Verwendung des Automation Interfaces ist jener, dass kein Import einer Datei in das TwinCAT-Projekt stattfindet, sondern Befehle durch die KS-SPS-Project-Creator Anwendung ausgeführt werden, welche direkt im TwinCAT-Projekt Methoden und Funktionen aufrufen, die wiederum als Beispiel, ein EtherCAT-Modul erzeugen oder einen Funktionsbaustein erstellen und diesen mit bestimmten Quellcodes befüllen. Dargestellt sind diese Anforderungen im nachfolgenden UML-Sequenzdiagramm in der Abbildung 7-8.

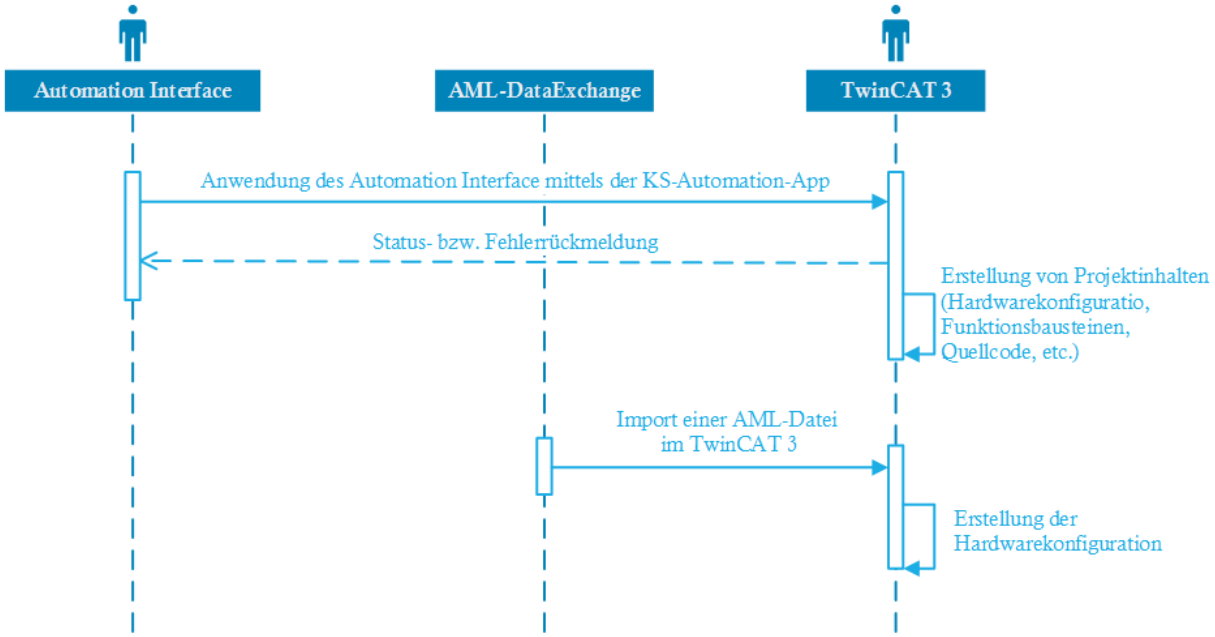

Abbildung 7-8: Sequenzdiagramm der Anforderungen an TwinCAT 3, Quelle: Eigene Darstellung

7.2.4 Anforderungen für die Erstellung von Variablen

Eine Variable ist ein abstrakter Platzhalter, welcher durch logische oder mathematische Funktionen mit Werten befüllt werden kann. In der Softwareentwicklung für SPS-Systeme wird im Allgemeinen unter folgenden Arten von Variablen unterschieden:

- Globale Variablen – Auf diese Variablen ist der Zugriff aus jedem Baustein des Projektes erlaubt
- Lokale Variablen – Auf diese Variablen kann nur innerhalb einer Funktion zugegriffen werden

Ebenfalls muss eine Zuweisung des benötigten Datentyps erfolgen. Der zu wählende Datentyp hängt von der Art des Anwendungsfalls und der logischen oder mathematischen Aufgabenstellung ab.

In der nachfolgenden Tabelle 7-1 sind gängige Datentypen als Beispiel dargestellt:

Type	Keyword	Größe [Bits]	Wertebereich
Integer	INT	16	Ganzzahl (-32768 bis 32767)
Double Integer	DINT	32	Ganzzahl (-2^{31}bis 2^{31} -1)
Word	WORD	16	Wort-Werte (0 bis 65535)
Double Word	DWORD	32	Doppelwort-Werte (0 bis 2^{32} -1)
String	STRING		1 Byte pro Zeichen + 1 zusätzliches Byte bei TwinCAT
Booleschen	BOOL	1	true oder false
Byte	Byte	8	Byte-Wert (0 bis 255 oder -128 bis 127)
Character	CHAR	8	ASCII-Code
Reelle Zahlen	REAL	32	Reelle Zahlen (IEEE-Gleitpunktzahl)
TIME	TIME	32	IEC-Zeitdauer (1ms Schritten)
Date	DATE	32	IEC-Datum (1 Tages Schritte)
Time of Day	Time_of_Day	32	Uhrzeit (1ms Schritten)
Array	ARRAY		Gruppe eines Datentyps
Structure	STRUCT		Gruppe kombinierter Datentypen

Tabelle 7-1: Auflistung gängiger Datentypen, Quelle: Eigene Darstellung

Da die EPLAN Electric P8 die Möglichkeit bietet, auch die Beschriftungen der einzelnen Kanäle der digitalen und analogen Hardwaremodule in eine Datei zu exportieren, werden diese Daten genutzt, um Variablenlisten für die Hardwarekonfiguration zu erstellen. Bei diesen Variablen handelt es sich meist um Boolesche-, Integer- oder REAL-Datentypen, da digitale und analoge Werte mit diesen darstellbar sind. Ebenfalls besteht teilweise jedoch auch die Möglichkeit, ganze Strukturen mit einem Hardwaremodul zu verknüpfen.

7.2.4.1 Erstellung von globalen Variablen

Globale Variablenlisten sind Sammlungen von Variablen, auf welche der Zugriff von jedem Ort des Programmes zugelassen wird. Wie auch bei der Hardwarekonfigurationserstellung, ist es auch bei der Erstellung von globalen Variablenlisten möglich, Informationen aus einer AML-Datei zu laden, da diese Namen bereits im EPLAN definiert werden können und bei einem Daten-Export im EPLAN in diese AML-Datei geladen werden.

Im nachfolgenden UML-Sequenzdiagramm, welches in der Abbildung 7-9 dargestellt wird, werden die Anforderungen und Abläufe für die Erstellung von globalen Variablen grafisch dargestellt:

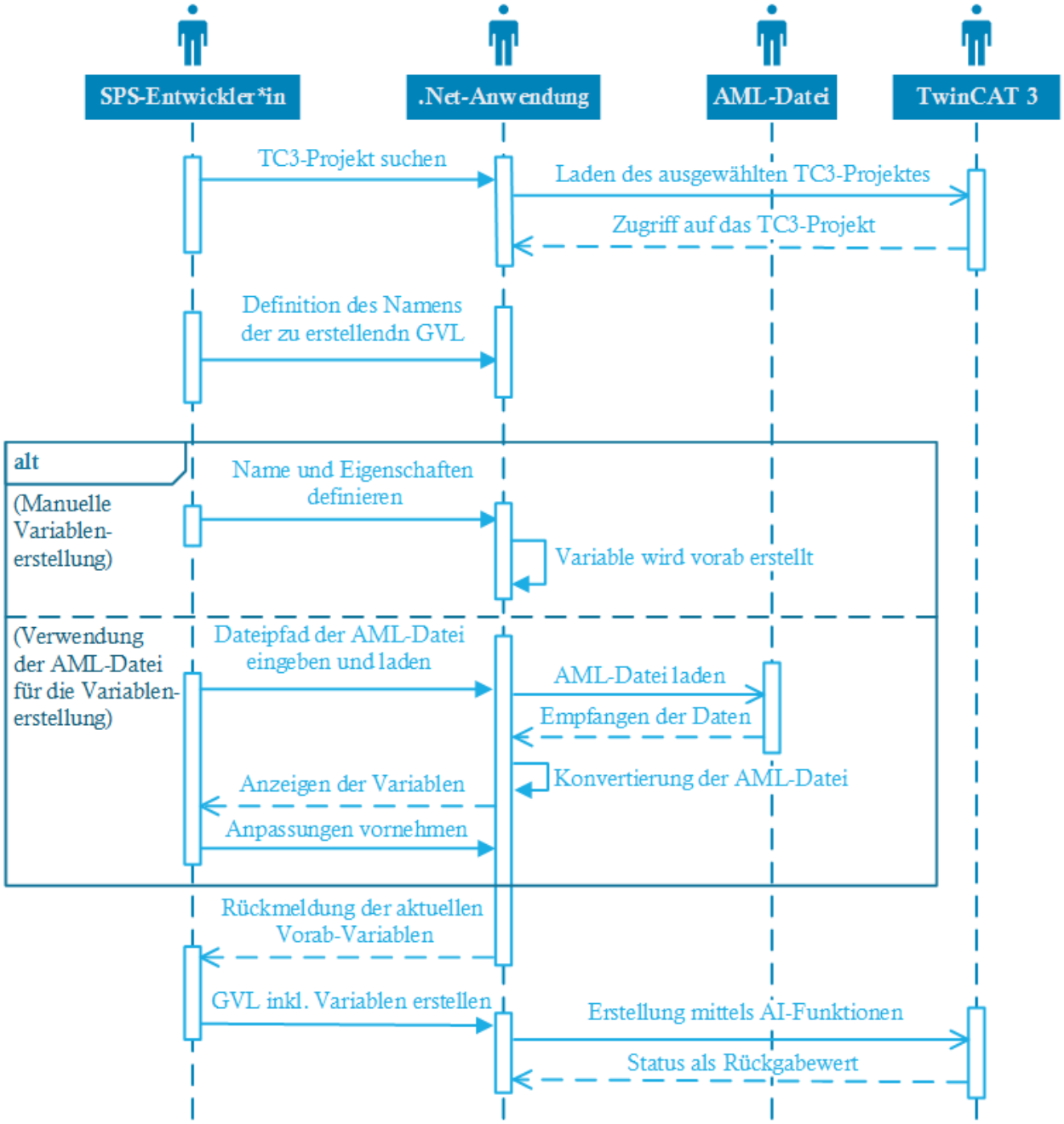

Abbildung 7-9: Sequenzdiagramm der Anforderungen an die Erstellung von globalen Variablen, Quelle: Eigene Darstellung

7.2.4.2 Erstellung von lokalen Variablen

Lokale Variablen werden in den jeweiligen Funktionen bzw. Funktionsbausteinen, durch die in der Abbildung 7-10 dargestellten Vorgehensweise, deklariert. Somit kann kein direkter Zugriff außerhalb des Bausteines erfolgen. Die jeweiligen SPS-Entwickler*innen müssen daher den Namen der gewünschten POU angeben, in welcher die Variablen erstellt werden. Für die finale Erstellung muss vorab eine Liste befüllt werden, die alle Eigenschaften der zu erstellenden Variablen enthält. In dieser Liste ist es möglich, Variablen hinzuzufügen und diese auch wieder zu löschen. Mittels einer Funktion, welche die entsprechenden Automation Interface Befehle zur Erstellung der Variablen ausführt, wird diese Liste in das TwinCAT 3-Projekt und die ausgewählte Programmorganisationseinheit übertragen (vergl. Kapitel 7.3.1).

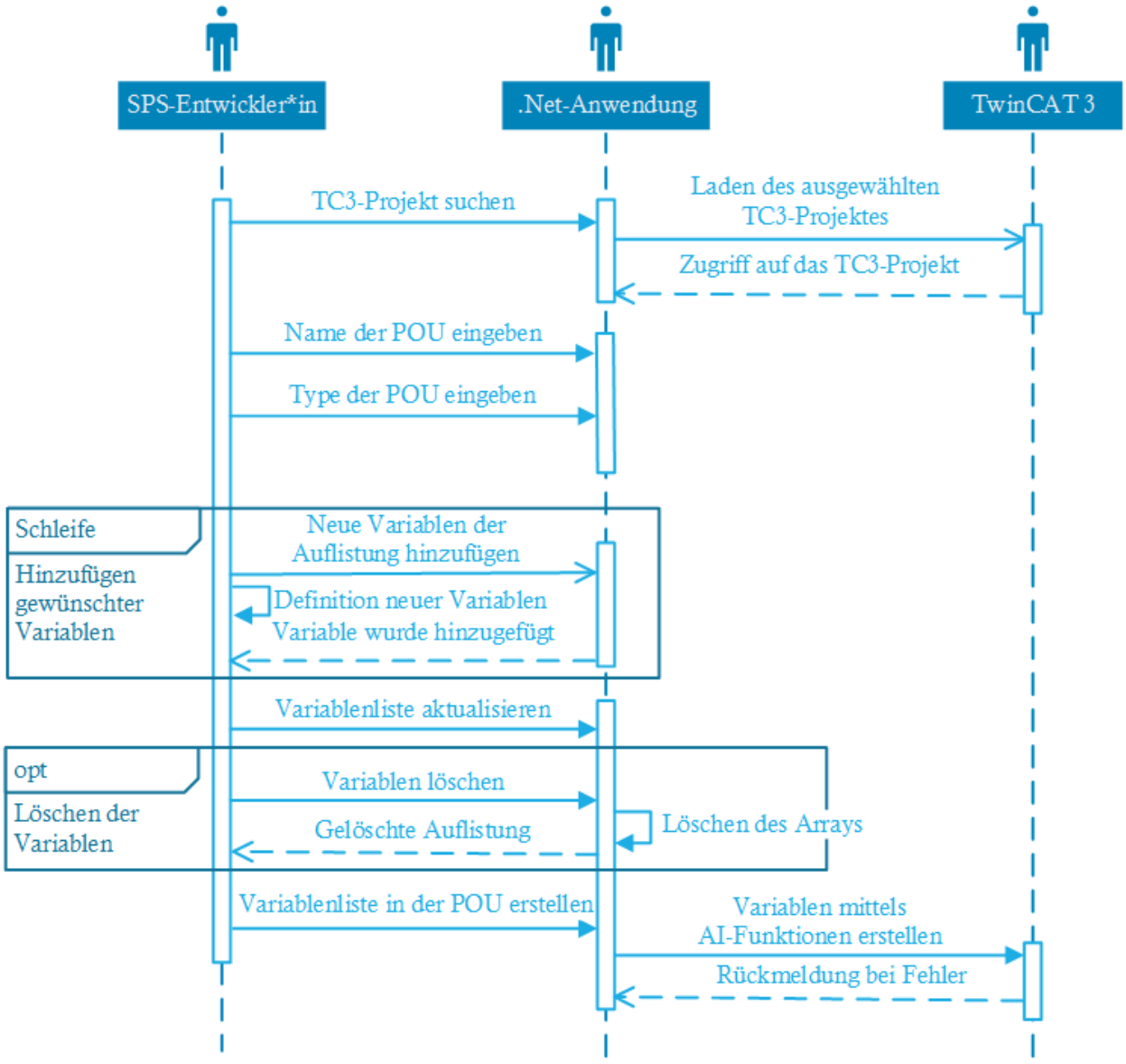

Abbildung 7-10: Sequenzdiagramm der Anforderungen an die Erstellung von lokalen Variablen, Quelle: Eigene Darstellung

7.2.5 Anforderungen für die Variablenzuordnung

Für die Zuordnung von Variablen zu deren I/O-Konnektoren werden globale Variablen verwendet, da sie die dafür notwendige Erreichbarkeit aufweisen. Diese müssen in einer globalen Variablenliste mit deren Typ und als Input- oder Output-Variable definiert sein.

Um die Zuweisung über den KS-SPC im TwinCAT3 durchführen zu können, muss zuerst ein bestehendes TwinCAT3-Projekt geladen werden, um Zugriff auf diese zu erhalten. Anschließend muss eine PLCopenXML-Datei der gewünschten Variablenliste dieses Projektes geladen werden. Mit dem Laden dieser Datei wird eine Konvertierung durchgeführt, um sämtliche verfügbare Variablen in einer Auflistung darzustellen.

Ist die Konvertierung abgeschlossen, kann die zu verlinkende Variable ausgewählt werden. Um die Verlinkung durchführen zu können, muss ebenfalls der Name des gewünschten Moduls, sowie dessen Typ und der Kanal definiert werden. Das nachfolgende UML-Sequenzdiagramm, in der Abbildung 7-11, stellt dies bildlich dar.

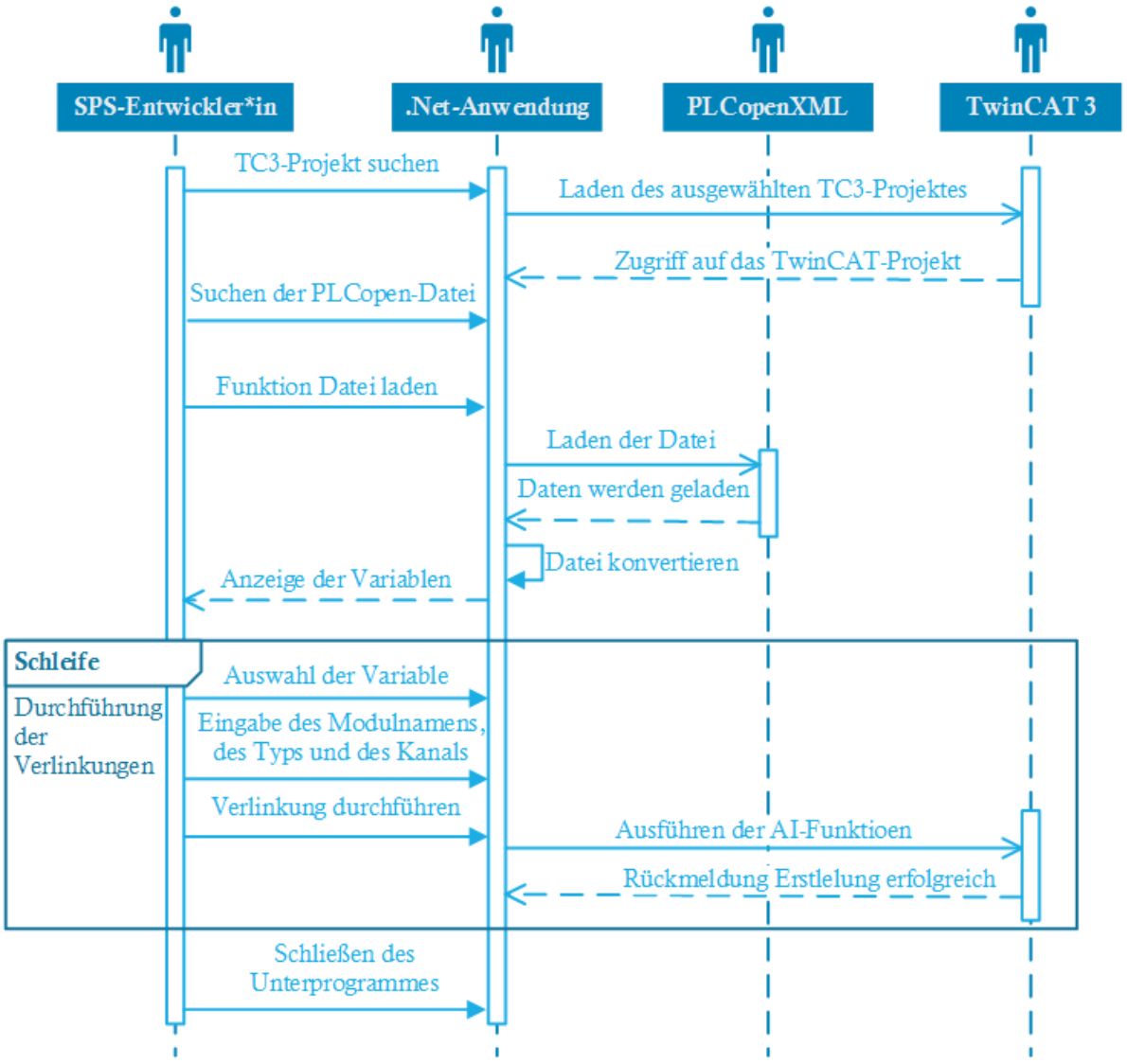

Abbildung 7-11: Sequenzdiagramm der Anforderungen an die Variablenzuordnung, Quelle: Eigene Darstellung

7.2.6 Anforderungen für die Erstellung von POUs

Eine POU dient zur Abarbeitung der programmierten Aufgabenstellungen. In dieser befindet sich der eigentliche Quellcode, welcher alle Abläufe zur Lösung der Aufgabenstellung, wie zum Beispiel Bedingungen, Schleifen oder Abfragen, in textueller Form beschreibt. Dieser wird, sofern sich der jeweilige Baustein in einer Taskklasse befindet, zyklisch abgearbeitet. Die Zuweisung der Taskklasse erfolgt manuell im TwinCAT-Projekt.

Taskklassen dienen zur Festlegung der Reihenfolge der abzuarbeitenden POUs. In Taskklassen wird die Zykluszeit bestimmt, die das Zeitintervall der aufeinanderfolgenden Aufrufe definiert. Wird zum Beispiel eine Taskklasse mit einer Zykluszeit von 10ms definiert, so wird diese Taskklasse mit all ihren zugewiesenen Funktionen und Funktionsbausteinen alle 10ms aufgerufen. Das entspricht einem Takt von 100Hz.

Für die Erstellung von Programmeinheiten müssen folgende Eigenschaften vergeben werden:

- Name der Programmorganisationseinheit
- Programmiersprache

  Durch die Auswahl der Programmiersprache wird definiert, welche Syntax für die Programmierung in diesem Baustein zur Anwendung kommt. Zur Auswahl stehen, laut IEC 61131-3 definierten Programmiersprachen folgende:[34]
  - ST (Structured Text)
  - FBD (Function Block Diagram)
  - ID (Ladder Diagram)
  - IL (Instruction List)
- Art der Programmorganisationseinheit

  Durch die Definition der Art der Programmorganisationseinheit wird entschieden, ob es sich bei der zu erstellenden POU um einen Funktionsbaustein, ein Programm oder um eine Funktion handeln soll. Dies hat Einfluss auf die Speicherfähigkeit der in der POU verwendeten Variablen. [35]

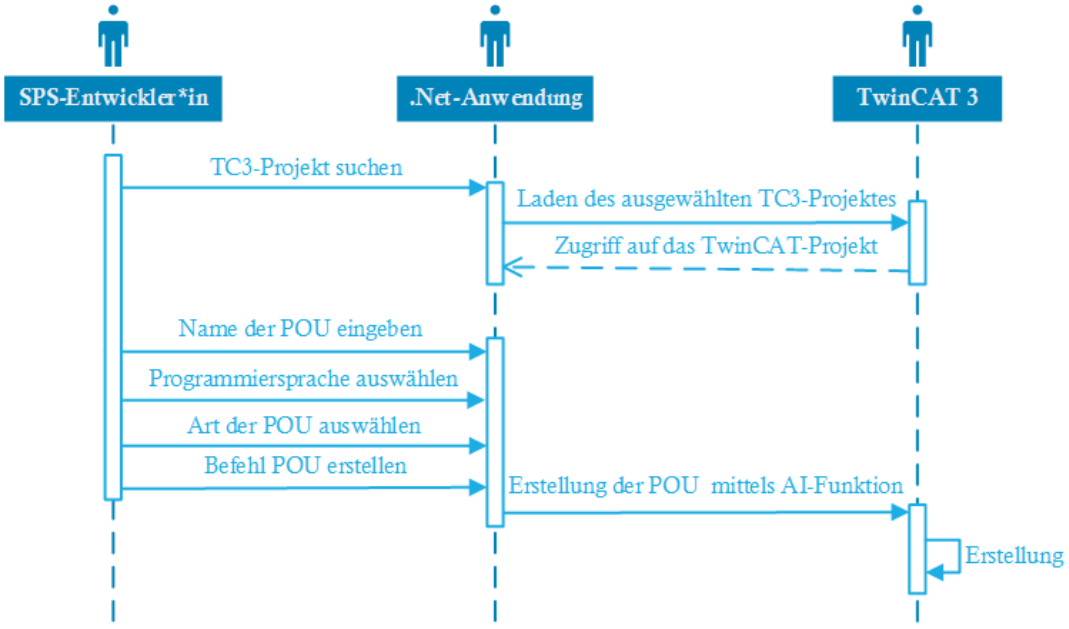

Abbildung 7-12: Sequenzdiagramm der Anforderungen an die Programmbausteinerstellung, Quelle: Eigene Darstellung

[34] Vgl. John & Tiegelkamp (2008), S. 103 f.

[35] Vgl. John & Tiegelkamp (2008), S. 21 f.

7.2.7 Anforderungen an die automatisierte Codeerstellung

Durch eine automatisierte Quellcodeerstellung soll ermöglicht werden, einfache logische Verknüpfungen zwischen Ein- und Ausgangs-Variablen sowie Lokalen-Variablen in einem SPS-Programm herzustellen. Der Grad der Komplexität einer automatisierten Quellcodeerstellung ist von der Anforderung der jeweiligen Aufgabenstellung abhängig. In dieser Arbeit beschränkt sich die automatisierte Quellcodeerstellung auf logische Verknüpfungen von Variablen. Als Programmiersprache für die automatisierte Erstellung von Quellcode wird Strukturierter Text (ST) angewendet.

7.2.7.1 Möglichkeiten einer automatisierten Codeerstellung

Eine automatisierte Codeerstellung kann durch benutzerdefinierte Einstellungen von einfachen logischen Verknüpfungen bis hin zu komplexen Steuer- und Regelungsaufgaben umgesetzt werden. In dieser Arbeit wurde die Erstellung von logischen UND- sowie ODER-Verknüpfungen realisiert. Dies dient jedoch als Basis für die Entwicklung weiterer Modelle. Für das Einlesen der dafür notwendigen Variablen wurde der Import einer PLCopenXML-Datei gewählt.

7.2.7.2 Ergänzungen durch manuelle Codeaddition

Da bei erhöhter Komplexität eine vollständige, automatisierte Codeerstellung nicht immer garantiert werden kann, ist eine manuelle Bearbeitung in TwinCAT 3 jederzeit möglich. Wird ein Code manuell geändert, ist es jedoch möglich, diese mittels AML wieder in das EPLAN-Projekt zu übertragen.

7.2.7.3 Modellierung

Für die Erstellung von Quellcodes in einem POU oder das Importieren eines bestehenden POU muss zuerst ein TwinCAT3-Projekt geladen werden, um auf diesem den Zugriff für die Anwendung der Automation Interface Funktionen zu erhalten.

Je nachdem, ob der Quellcode manuell erstellt oder importiert werden soll, muss die dementsprechende Funktion ausgeführt werden. Das entscheidet, ob die PLCopenXML-Datei konvertiert und die Variablendaten gelesen werden oder ob sie als gesamte POU importiert wird.

Wird die Codeerstellung manuell durchgeführt, so erfolgt basierend auf der Konvertierung der PLCopenXML-Datei eine Trennung und Auflistung der Variablen. Diese Variablen können anschließend logischen Verknüpfungen zugewiesen werden.

Ist die Zuweisung abgeschlossen oder ist keine Zuweisung aufgrund des Imports einer PLCopenXML notwendig, kann der Quellcode im TwinCAT erstellt werden.

Die nachfolgende Abbildung 7-13 soll anhand eines UML-Sequenzdiagrammes die Anforderungen für eine automatisierte Quellcodeerstellung darstellen.

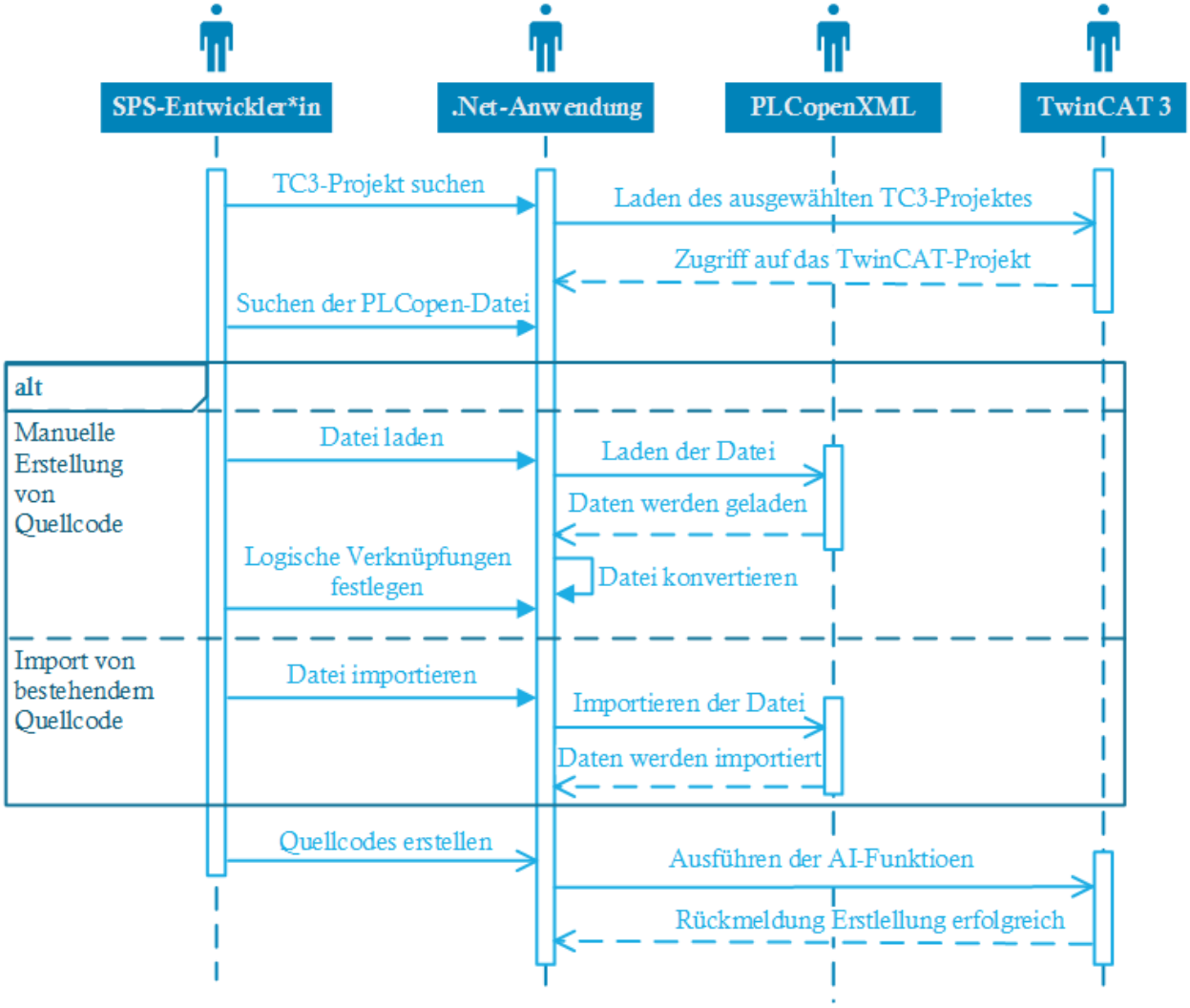

Abbildung 7-13: Sequenzdiagramm der Anforderungen an die automatisierte Quellcodeerstellung, Quelle: Eigene Darstellung

7.2.8 Anforderungen für die Befüllung einer I/O-Checkliste

Zur Erstellung einer Checkliste, zur Überprüfung der Eingangs- und Ausgangssignale während einer Inbetriebnahme, kann ebenfalls die AML-Exportdatei, welche mittels EPLAN-Export erstellt wurde, in die KS-SPS-Project-Creator Anwendung importiert werden, und so eine automatische Befüllung einer Vorlage-Checkliste durch die jeweiligen Bediener*innen ausgeführt werden. Der*die Inbetriebnehmer*in muss die entsprechende AML-Datei laden, um anschließend mithilfe der Importfunktion im KS-SPC die aktuelle Hardwarekonfiguration zu importieren. Die Konvertierung findet automatisiert in der .Net-Applikation statt. Die konvertierten Daten, welche aus den Hardwareinformationen bestehen, werden dem Bediener in einer Auflistung angezeigt. Anpassungen können im Unterprogramm zur Erstellung der Hardwarekonfiguration vorgenommen werden. Dargestellt sind diese Abläufe im nachfolgenden UML-Sequenzdiagramm.

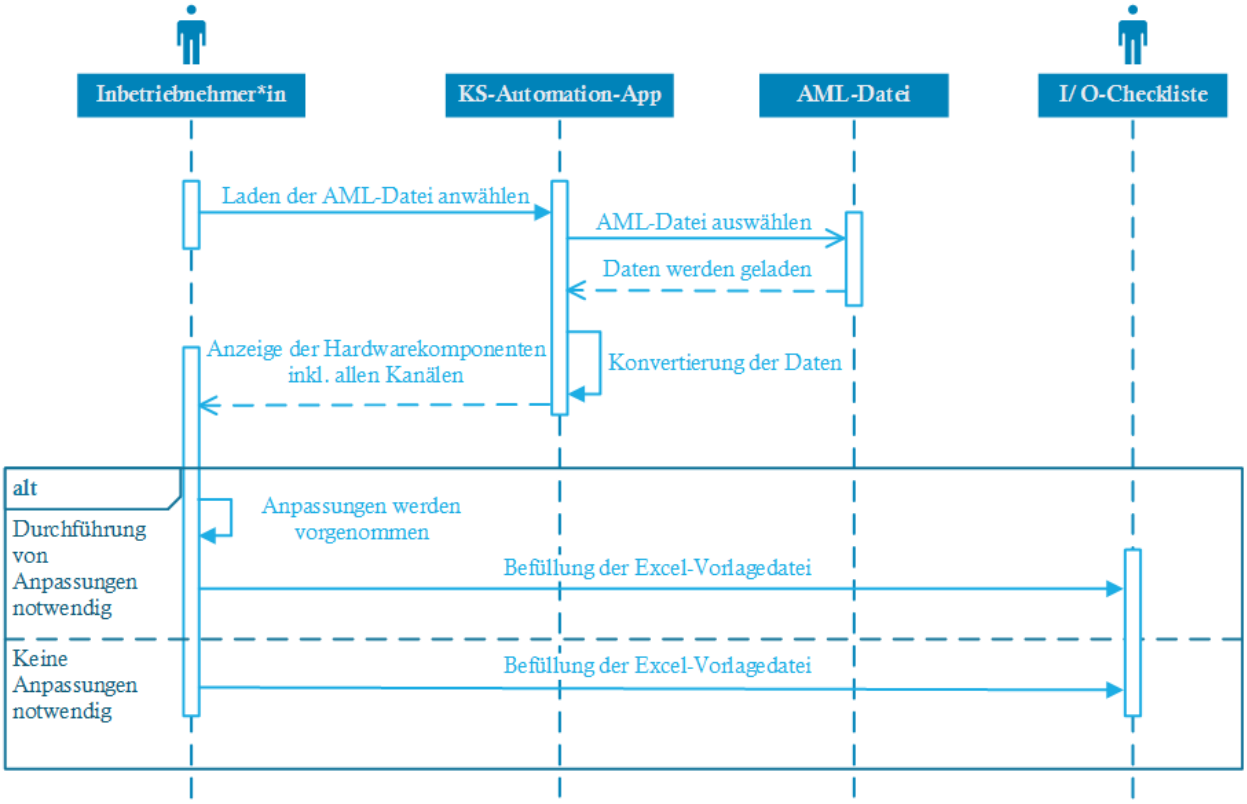

Abbildung 7-14: Sequenzdiagramm der Anforderungen an die Befüllung von I/O-Checklisten, Quelle: Eigene Darstellung

Für die Befüllung einer I/O-Checkliste ist eine Vorlage erstellt worden, welche automatisiert befüllt wird. Der Aufbau dieser Struktur ist in der nachfolgenden Abbildung 7-15 dargestellt.

	A	B	C	D	E	F	G	H	I	J	K	L	M	P
1	IBN - Checkliste - I/O und Funktion													
2	Auftraggeber:	BMW												
3	Auftragnehmer:	Kristl, Seibt & Co Ges.m.b.H.												
4	Anlage:	Vorlage												
5	Projektnummer:	Vorlage												
6	Prüfstand	Vorlage												
7	Überprüfer:	Katzjäger												
8	Auftragsnummer													
9	Seriennummer:													
10	Verteiler	BMK	Klemme		Kanal	Datentyp	Bezeichnung		elekt. BMK	Vorgang	Status	Bemerkung		Datum
11	I/O-SPS													
12	Digitale Eingänge													
13		-120KF1	EL1904		1	BOOL	Schnellstopp KS OMM CH1		-1AF1	Not-Halt betätigen und SDI kontrollieren	OK	SDI		03.08.2020
14														

Abbildung 7-15: Struktur der I/O-Checkliste, Quelle: Eigene Darstellung

7.3 Erstellung der Entwürfe / Design-Erstellung

Die Erstellung der Entwürfe bzw. die Designerstellung bezieht sich auf die dritte Phase der Softwareentwicklung bei Verwendung des Wasserfallmodells und wird in den nachfolgenden Punkten umgesetzt. Auch in dieser Phase werden UML-Diagramme zur Modellierung und Darstellung der zur erstellenden Entwürfe verwendet.

7.3.1 Modellierung der grafischen Benutzeroberfläche

Mittels der grafischen Benutzeroberfläche, auch GUI (Graphical User Interface) genannt, können sämtliche Befehle im KS-SPC ausgeführt werden. Für die Erstellung einer solchen GUI werden im Visual Studio Windows Forms-Anwendungen erstellt und diese grafisch designt. Die nachfolgenden Modellierungen dienen daher zur Einteilung und Bestimmung der Design-Umfänge und der Beziehungen untereinander.

7.3.1.1 Erstellen des Hauptmenüs

Das Hauptmenü wird über einen Menüstreifen realisiert, in welchem alle weiteren Unterprogramme anwählbar sind. Das nachfolgende Diagramm in Abbildung 7-16 stellt alle möglichen anwählbaren Unterprogramme dar und beschreibt die dadurch entstehenden Funktionen.

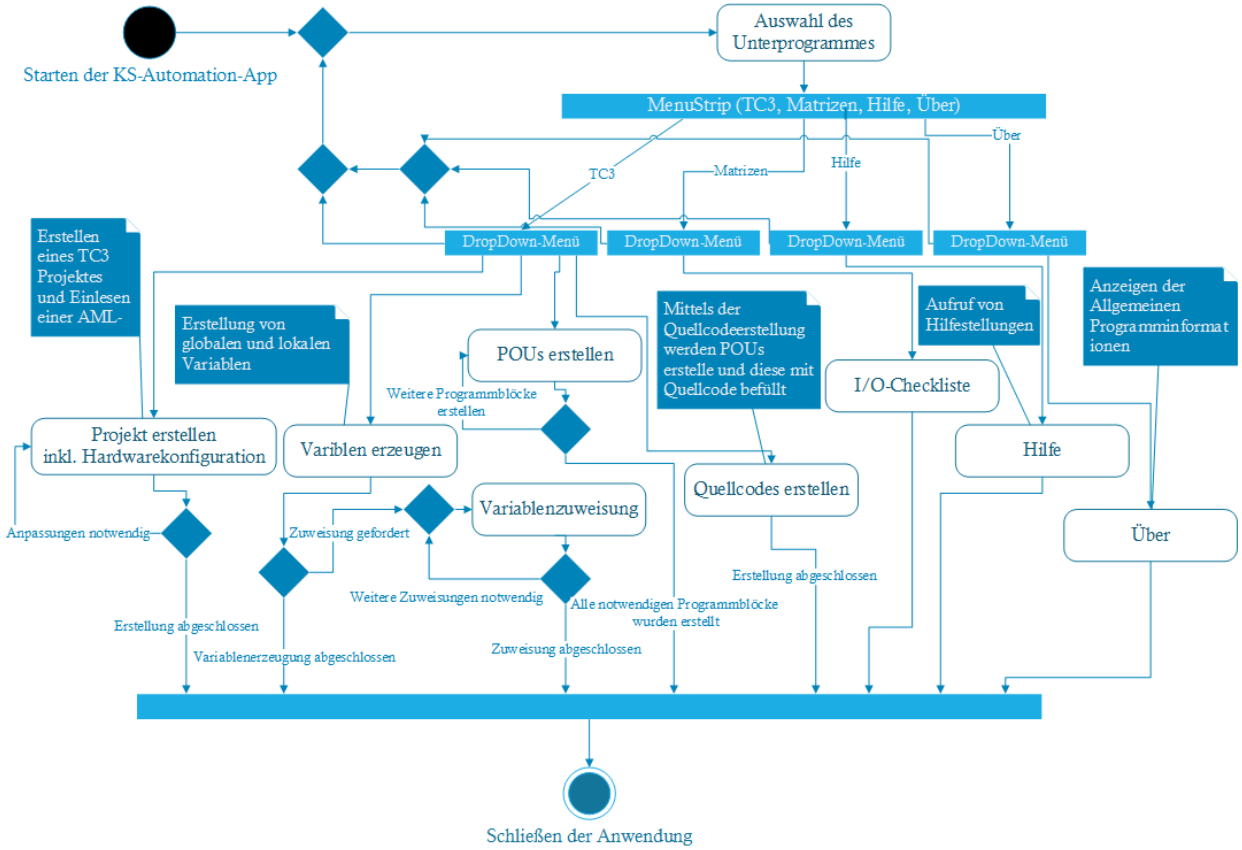

Abbildung 7-16: Aktivitätsdiagramm - Design des Hauptmenüs, Quelle: Eigene Darstellung

7.3.1.2 Erstellen eines TC3-Projektes inkl. Konfiguration

Mit der Anwahl des Unterprogrammes „Projekt erstellen“ bietet der KS-SPC die Möglichkeit, TwinCAT 3 zu öffnen und anschließend ein leeres Projekt zu erstellen. Dieses Projekt wird dann genutzt, um ein Vorlageprojekt, ein sogenanntes Template, zu laden. In diesem Vorlageprojekt ist bereits ein SPS-Projekt angelegt. Dieses Projekt kann dann mit einer Hardwarekonfiguration, welche zum Beispiel aus einer AML-Datei eingelesen oder auch selbst erstellt wird, befüllt werden. Wenn jedoch eine manuelle Erstellung der weiteren Projektdaten gewünscht ist, kann hier das Unterprogramm geschlossen und weitere Eingaben manuell im TwinCAT vorgenommen werden.

Bevor eine Konfiguration aus einer AML-Datei erstellt wird, kann diese ebenfalls noch angepasst werden. Die Anpassungen können direkt in der dafür vorgesehenen Auflistung der Hardwarekomponenten der importierten AML-Datei durchgeführt werden. Das in der Abbildung 7-17 gezeigte Diagramm stellt diesen Ablauf in Form eines UML-Aktivitätsdiagrammes dar.

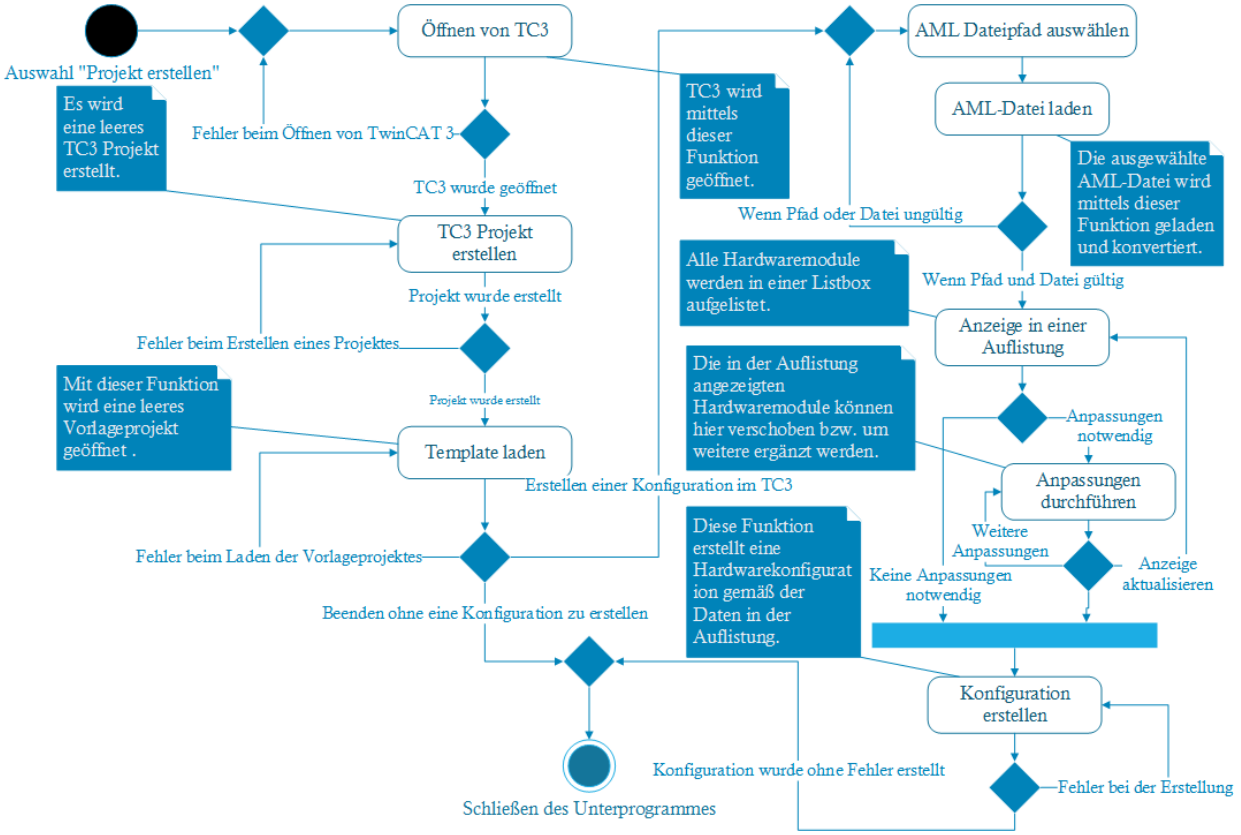

Abbildung 7-17: Aktivitätsdiagramm - Design der Projekterstellung, Quelle: Eigene Darstellung

Tritt während der Erstellung einer Konfiguration ein Fehler auf, wird diese nicht erstellt und eine Fehlermeldung ausgeworfen. Wurde die Konfiguration erstellt, so ist der Vorgang abgeschlossen und das Unterprogramm kann geschlossen werden. Das Einfügen weiterer Hardwarekomponenten ist durch ein weiteres Unterprogramm möglich.

7.3.1.3 Erstellen von globalen Variablenlisten und Variablen

Für die Erstellung einer globalen Variablenliste (GVL) muss ein bestehendes TwinCAT-Projekt geladen werden. Für eine automatisierte Erstellung einer GVL muss die gewünschte AML-Datei ausgewählt und geladen werden, welche alle Informationen für die Variablendeklaration zur Verfügung stellt. Diese AML-Datei wird anschließend konvertiert, um eine Ermittlung der Variablennamen durchzuführen. Diese Variablennamen werden anschließend in einer Auflistung im KS-SPC angezeigt. Die angezeigten Variablen können bei Bedarf angepasst oder verschoben werden.

Ist keine AML-Datei vorhanden oder soll die Erstellung der Variablenlisten manuell durchgeführt werden, ist dies auch durch den KS-SPS-Project-Creator möglich bzw. kann dies natürlich auch im TwinCAT 3 vorgenommen werden. Hierfür muss die Auflistung über die Funktion „Variable hinzufügen“ ergänzt werden. Hierfür ist es notwendig den gewünschten Namen sowie den Datentyp und Art der Variable, Input oder Output, zu definieren. Ebenfalls ist es möglich, Variablen aus dieser Liste mittels der Funktion „Variable löschen“ zu löschen.

Die nachfolgende Abbildung 7-19 zeigt die beschriebenen Abläufe zur Erstellung einer globalen Variablenliste anhand eines UML-Aktivitätsdiagrammes.

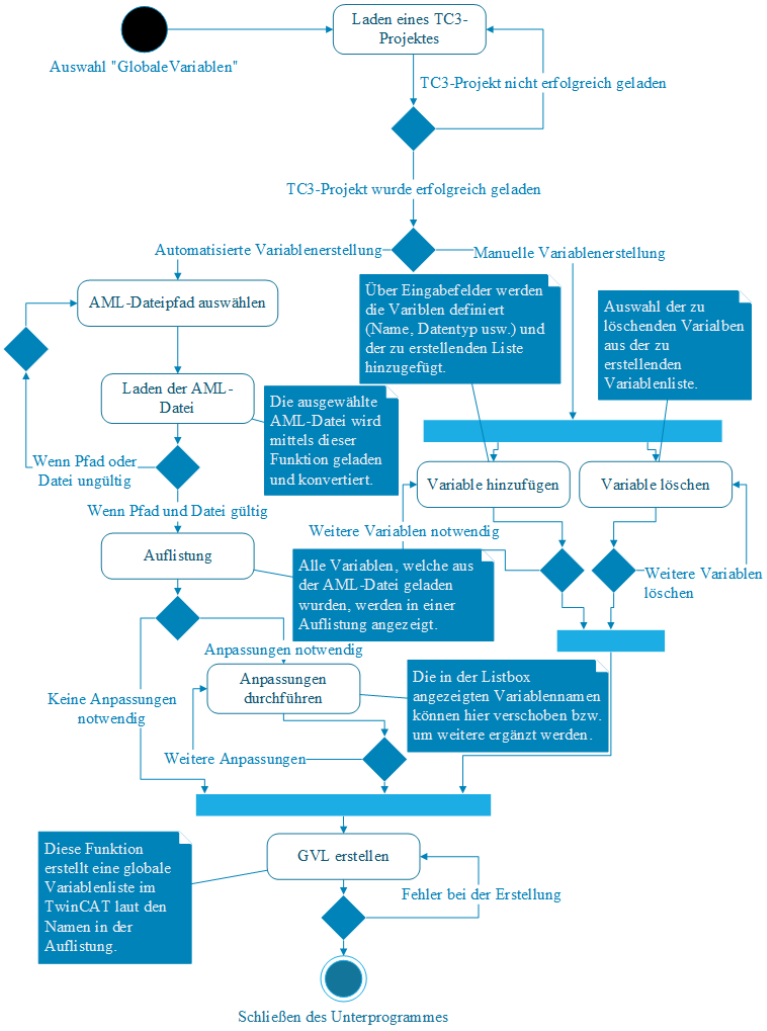

Abbildung 7-19: Aktivitätsdiagramm - Design der Erstellung von GVL`s, Quelle: Eigene Darstellung

7.3.1.4 Zuordnung von SPS-Ein-/Ausgangsvariablen und deren I/O-Gegenstücken

Um den Status von SPS-Hardwaremodulen, wie zum Beispiel den aktuellen Wert einer analogen Eingangsbaugruppe in einem SPS-Programm verarbeiten zu können, ist es notwendig, die einzelnen Kanäle dieser Baugruppen mit Variablen zu verknüpfen. Es ist zwar möglich, durch Adressen direkt auf diese Kanäle in einem SPS-Programm zuzugreifen, jedoch ermöglicht die Verwendung von Variablen eine wesentlich strukturiertere und nachvollziehbarere Programmierung. Die zu verlinkenden Variablen müssen als globale Variablen und entweder als Input- oder Output-Variable deklariert sein.

Die nachfolgenden Abbildungen zeigen jeweils eine Variablendeklaration für Eingänge und für Ausgänge in einer globalen Variablenliste in der Entwicklungsumgebung TwinCAT 3.

Eingänge:

```
(*EL1004*)
di_Kanal1    AT    %I*    :    BOOL;    (*Digitaler Eingang Kanal 1*)
```

Abbildung 7-20: Variablendeklaration einer Eingangsvariable, Quelle: Eigene Darstellung

Ausgänge:

```
(* EL2008*)
do_Kanal1    AT    %Q*    :    BOOL;    (*Digitaler Ausgang Kanal 1*)
```

Abbildung 7-21: Variablendeklaration einer Ausgangsvariable, Quelle: Eigene Darstellung

Somit müssen folgende Eingabefelder für die Erstellung von Variablen vorhanden sein:

- Name der Variable (in Abbildung 7-20 und Abbildung 7-21 rot markiert)
  Der Name der Variable kann beliebig vergeben werden. Laut der KS Engineers Programmierrichtlinie müssen diese Namen jedoch am Beginn den Variablentyp enthalten. [36]

- Datentyp (in Abbildung 7-20 und Abbildung 7-21 grün markiert)
  In diesem Fall ist der Datentyp „BOOL“ definiert, was bedeutet, dass es sich um eine Booleschen Variable handelt und diese nur den Zustand „true“ oder „false“ annehmen kann. [37]

- Variablentyp (in Abbildung 7-20 und Abbildung 7-21 blau markiert)
  Der Variablentyp deklariert eine Variable entweder als Input oder Output. Dies wird durch die Ergänzungen „%Q*“ oder „%I*“ realisiert.
  - %I* → Definition einer Input-Variable
  - %Q* → Definition einer Output-Variable [38]

Jedes Hardwaremodul verfügt über Kanäle, an welche Sensoren oder Aktoren angeschlossen werden können. Diese Kanäle müssen mit den oben genannten Variablen verknüpft werden. Diese Verknüpfung kann durch den KS-SPC mithilfe der Anwendung des Automation Interfaces oder aber auch manuell im TwinCAT 3 durchgeführt werden.

---

[36] Vgl. KS Engineers (2021), S 1.

[37] Vgl. John & Tiegelkamp (2008), S. 72.

[38] Vgl. Beckhoff Automation (2022), Kapitel Globale Variablen.

Das nachfolgende Diagramm in der Abbildung 7-22 stellt sämtliche Abläufe, welche für die Zuordnung von Variablen zu deren I/O-Gegenstücken notwendig sind in Form eines UML-Aktivitätsdiagramms dar. Die Zuordnung der Variablen kann natürlich auch manuell im TwinCAT 3-Projekt erfolgen und parallel der Zuordnung mithilfe des KS-SPS-Project-Creators durchgeführt werden.

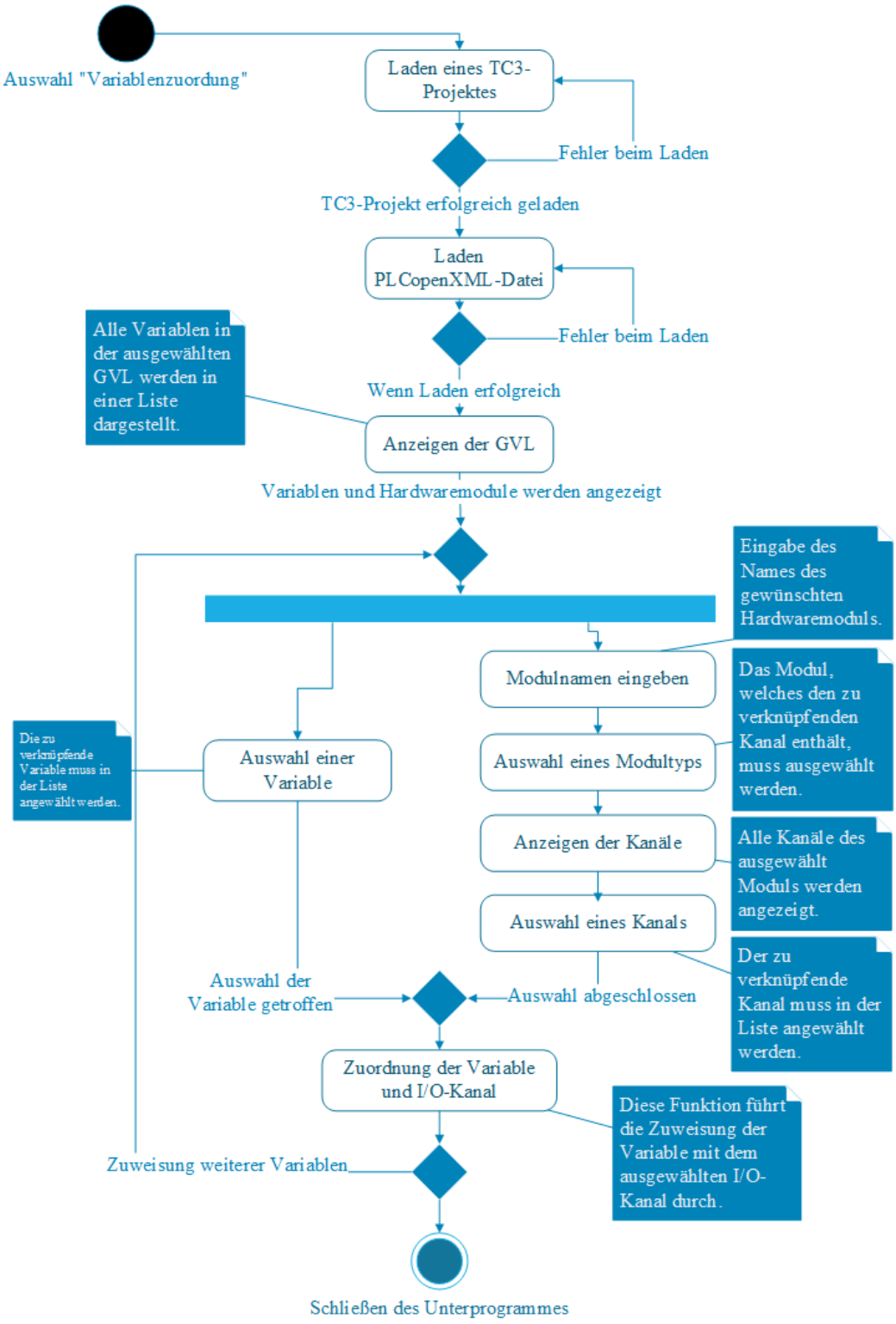

Abbildung 7-22: Aktivitätsdiagramm - Design der Variablenzuordnung, Quelle: Eigene Darstellung

7.3.1.5 Erstellung von POUs

Die Erstellung von POUs, wie etwa Programme, Funktionen oder Funktionsbausteine, ist über eine grafische Benutzeroberfläche in der KS-SPS-Project-Creator Anwendung möglich. Zuerst muss ein bestehendes TwinCAT3-Projekt geladen werden, in welchem die zu erstellenden POUs integriert werden.

Um eine POU erstellen zu können, müssen dessen Eigenschaften, wie der Name, die Programmiersprache und der Typ definiert werden. Dies wird über Textfelder bzw. über DropDown-Menüs realisiert. Sind diese Felder mit verwertbaren Daten gefüllt, so kann die Erstellung eines POUs erfolgen.

Die nachfolgende Abbildung 7-23 modelliert die notwendigen Abläufe zur Erstellung eines neuen POUs und weist nochmals auf die manuelle Zuweisung einer Taskklasse hin.

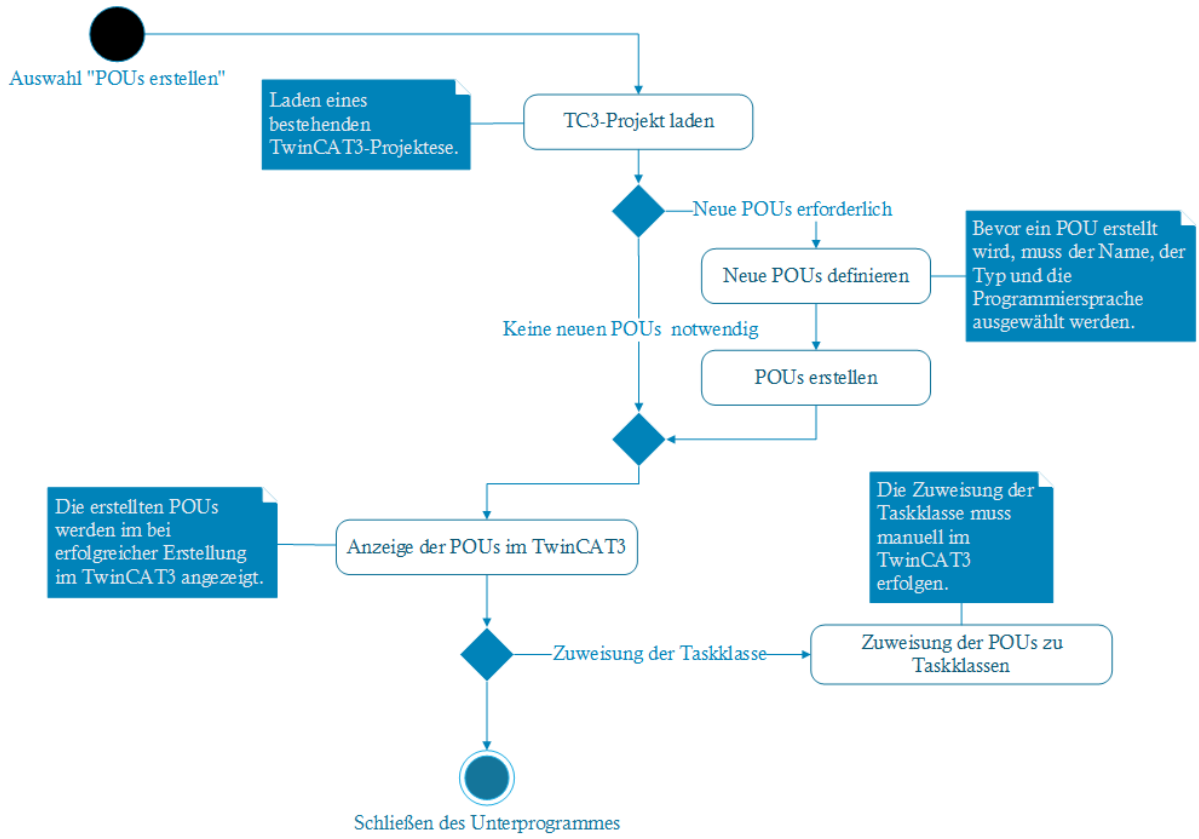

Abbildung 7-23: Aktivitätsdiagramm - Design der Programmbausteinerstellung, Quelle: Eigene Darstellung

Das Erstellen von POUs wird durch den KS-SPC durch die Verwendung des Automation Interfaces ermöglicht. Dies kann natürlich ebenfalls auch manuell in der Entwicklungsumgebung TwinCAT 3 durchgeführt werden.

7.3.1.6 Erstellung von lokalen Variablen

Lokale Variablen werden nicht etwa, wie globale Variablen in einer globalen Variablenliste, von welcher aus der Zugriff auf jede POU erlaubt ist, erstellt, sondern werden im jeweiligen POU deklariert und erzeugt. Somit muss zu Beginn ein gewünschter POU ausgewählt werden, in welchem die Variablenerstellung stattfinden soll. Lokale Variablen können nur manuell durch Eingabe der notwendigen Eigenschaften, wie Datentyp, Namen und dem Variablentyp der Variablenliste hinzugefügt werden. Ebenfalls besteht die Möglichkeit, die hinzugefügten Variablen, bevor diese im TwinCAT 3-Projekt erzeugt werden, zu löschen. Mit der Funktion „Variablen im TC3 erstellen“ werden die Variablen im ausgewählten POU erstellt. Dargestellt ist diese Vorgehensweise in der nachfolgenden Abbildung 7-24 durch ein UML-Aktivitätsdiagramm (vergl. Kapitel 7.2.4).

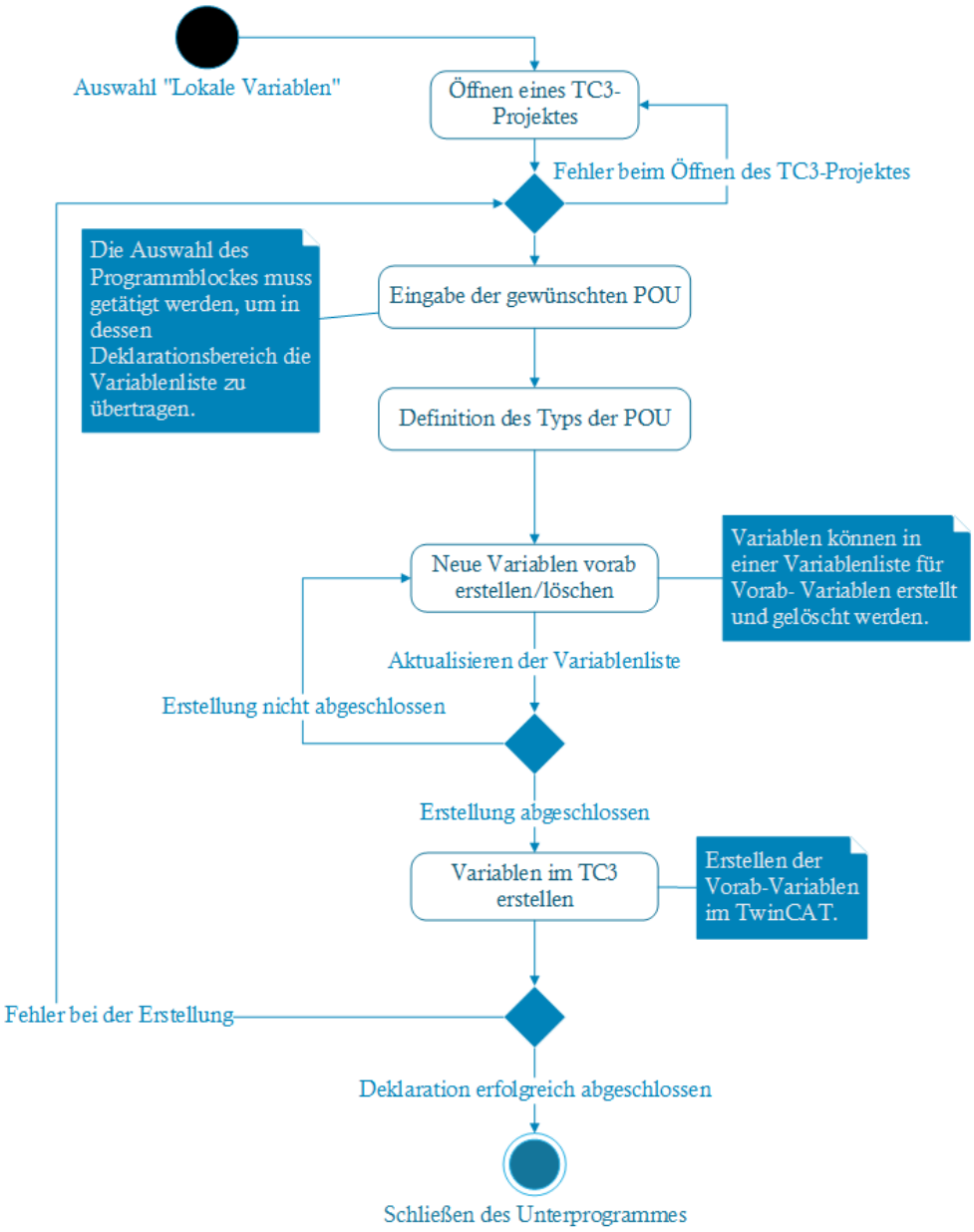

Abbildung 7-24: Aktivitätsdiagramm - Design für die Erstellung von lokalen Variablen, Quelle: Eigene Darstellung

7.3.1.7 Erstellung von automatisierten Quellcodes

Die Erstellung von automatisierten Quellcodes ist ebenfalls, wie zum Beispiel bei der Erstellung der Hardwarekonfiguration oder von POUs, über eine grafische Benutzeroberfläche möglich. Um einen Quellcode zu erzeugen, muss zuerst das gewünschte TwinCAT3-Porjekt geladen werden. Anschließend muss für eine automatisierte Codeerstellung aber auch für den Import einer Vorlage-POU die dazugehörige PLCopenXML-Datei geladen werden.

Je nach Anwendungsfall erfolgt die Trennung und Auflistung der Variablen. Handelt es sich um eine manuelle Codeerstellung können die Variablen logisch miteinander verknüpft werden. Die erstellten Verknüpfungen werden in einer Vorabliste angezeigt und können ebenfalls wieder gelöscht werden.

Sind sämtliche Zuweisungen abgeschlossen, kann daraus ein Quellcode in TwinCAT3 erzeugt werden. Diese Abläufe sind im nachfolgenden UML-Aktivitätsdiagramm in der Abbildung 7-25 grafisch dargestellt.

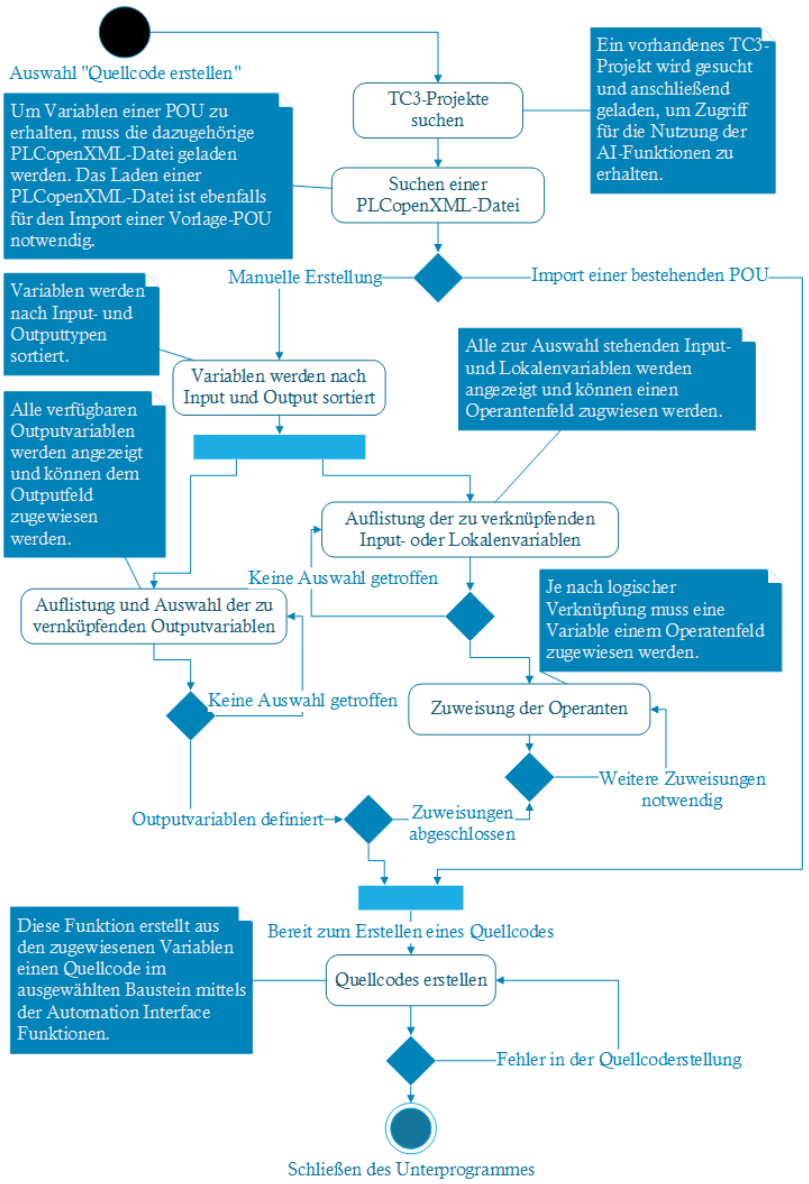

Abbildung 7-25: Aktivitätsdiagramm - Design der automatisierten Quellcodeerstellung, Quelle: Eigene Darstellung

7.3.1.8 Befüllen einer Vorlagedatei für eine I/O-Checkliste im Excel-Format

Das nachfolgende UML-Aktivitätsdiagramm in der Abbildung 7-26 soll veranschaulichen, wie der grafische Aufbau für die Befüllung einer Excel-Datei für eine I/O-Checkliste umgesetzt ist. Die Daten, welche für die Befüllung einer Excel-Datei mittels Import einer AML-Datei herangezogen werden, werden in einem EPLAN-Projekt definiert und bereitgestellt. Wird die AML erfolgreich geladen, werden alle Module inkl. Kanäle in einer Auflistung dargestellt. Bevor die Übertragung dieser Liste in eine Excel-Datei durchgeführt werden kann, muss zuerst eine passende Datei mittels Suchfunktion ausgewählt werden. Anschließend kann mithilfe der Funktion „Befüllen der Datei“ die Datenübertragung eingeleitet werden. Um eine korrekte Darstellung der Daten zu erhalten, muss wie in Punkt 7.2.8 beschrieben, eine formatierte Vorlagedatei verwendet werden.

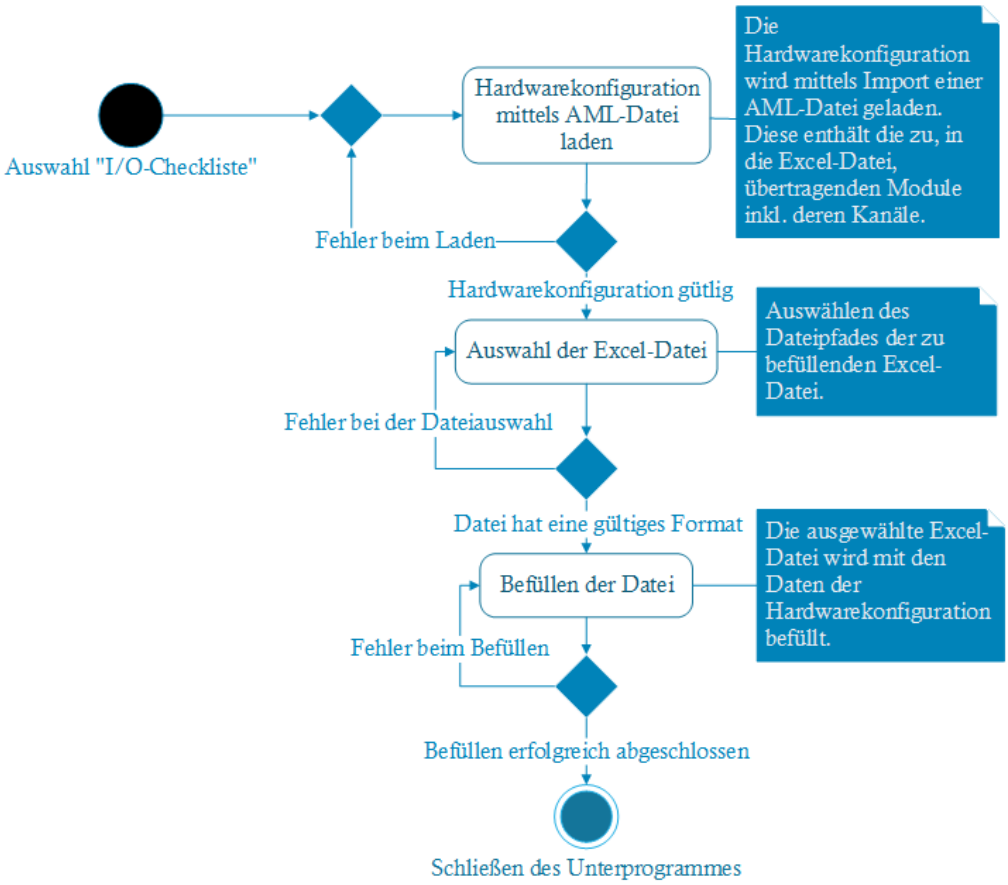

Abbildung 7-26: Aktivitätsdiagramm - Design für die Befüllung einer Excel-Datei, Quelle: Eigene Darstellung

7.3.2 Modellierung der benötigten Funktionen

Die nachfolgenden Punkte befassen sich mit der Modellierung des Strukturaufbaus der C#-Anwendung. Dazu eignen sich sogenannte Strukturdiagramme wie zum Beispiel das Klassendiagramm, welches in Punkt 6.2 bereits näher beschrieben wurde.

7.3.2.1 Erstellung eines TC3-Projektes inkl. Hardware-Konfiguration

Das UML-Klassendiagramm in der Abbildung 7-27 stellt die Beziehung der Klassen zur Erstellung eines TwinCAT 3 Projektes inklusive der Hardwarekonfiguration dar. Die Klasse MessageFilter ist eine von der Fa. Beckhoff bereitgestellte Klasse, mit welcher eine COM-Nachrichtenfilterung implementiert wird.

Um ein TC3-Projekt zu laden und Zugriff auf die Konfiguration zu erhalten, werden die Methoden in der Klasse „loadTCProject“ angewendet. Die Klasse „LoadAmlFile“ stellt die Methoden zur Konvertierung der AML-Datei in ein String-Array bereit, welche sämtliche Hardwarekonfigurationsdaten des zu erstellenden TwinCAT-Programmes enthalten. Um Funktionen des Automation Interface auszuführen wird die Klasse „setAiFunctions“ aufgerufen. Mithilfe der Klasse „EditModul“ können neue Hardwaremodule hinzugefügt werden.

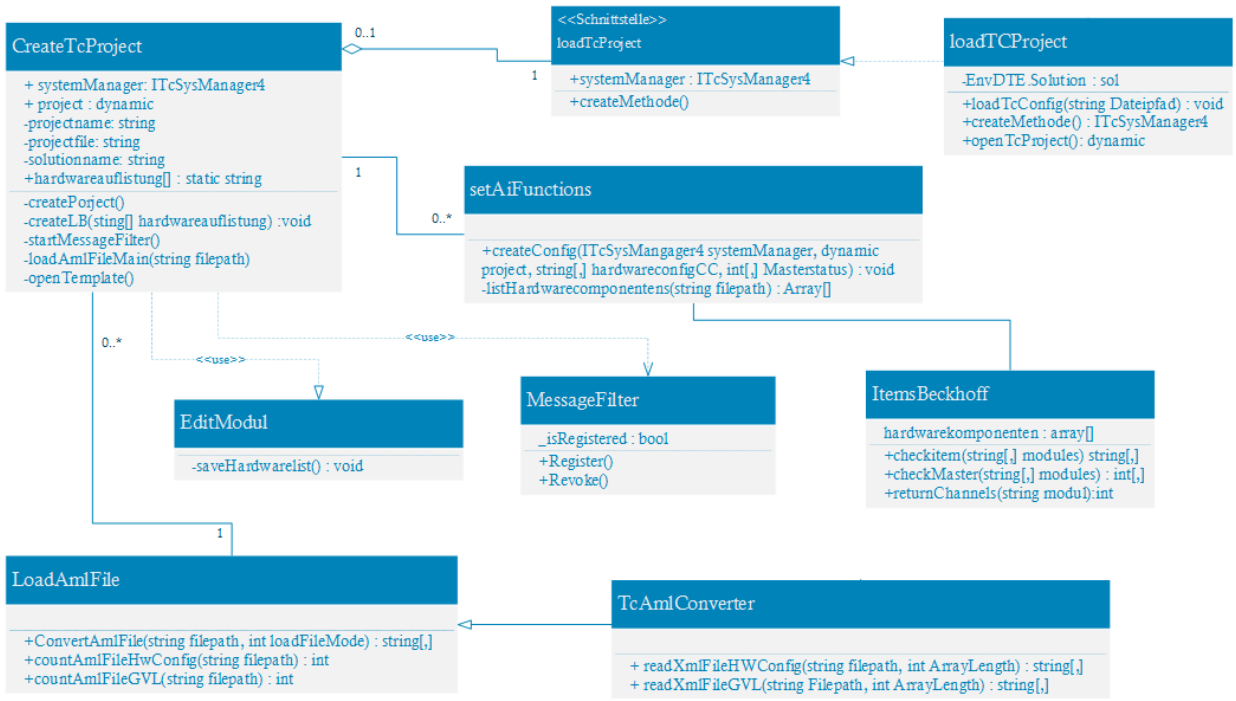

Abbildung 7-27: Klassendiagramm-Erstellung einer TC3 Konfiguration, Quelle: Eigene Darstellung

7.3.2.2 Erstellung von globalen Variablenlisten und Variablen

In der nachfolgenden Abbildung 7-28 wird die Modellierung durch ein UML-Klassendiagramm zur Erstellung von globalen Variablenlisten und deren Variablen aufgezeigt. Wie auch bei der Erstellung einer Hardwarekonfiguration, muss das Laden eines TC3-Projektes durch die Methoden der Klasse „loadTCProject“ erfolgen.

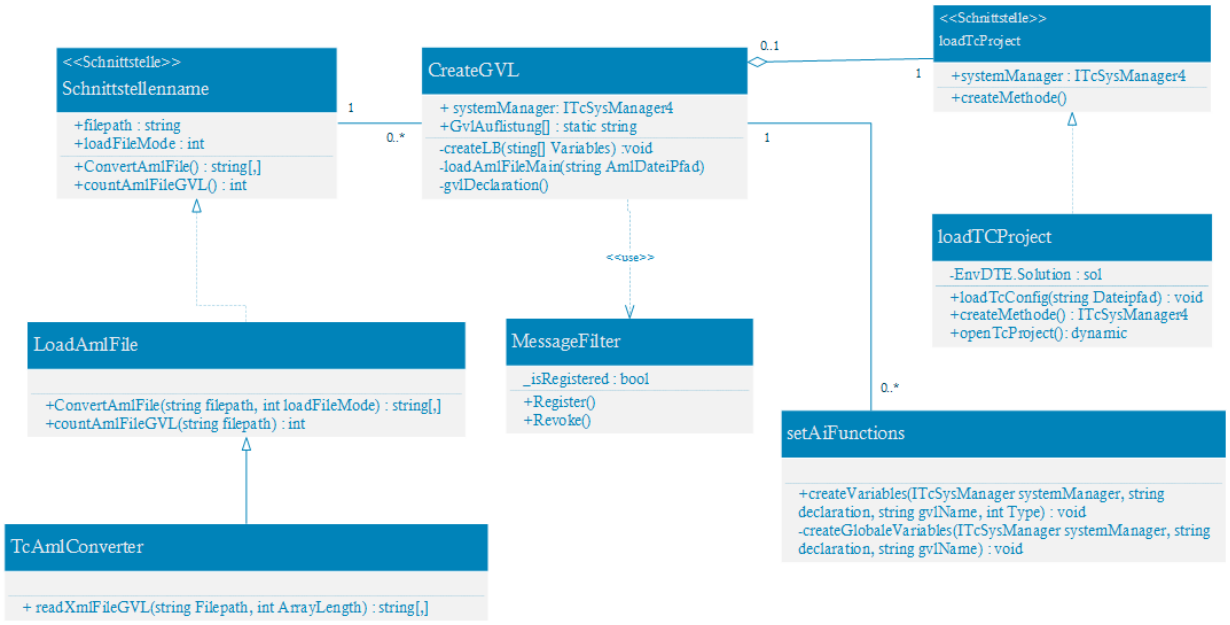

Abbildung 7-28: Klassendiagramm-Erstellung von GVL`s und GV`s, Quelle: Eigene Darstellung

7.3.2.3 Erstellung von lokalen Variablen

Der Unterschied bei der Erstellung von lokalen zu globalen Variablen ist jener, dass eine Vorab-Erstellung von Variablen durch eine AML-Datei nicht möglich ist. Variablen müssen manuell der Vorabliste hinzugefügt werden. Diese Methode wird durch die Klasse „addLocalVariables“ zur Verfügung gestellt.

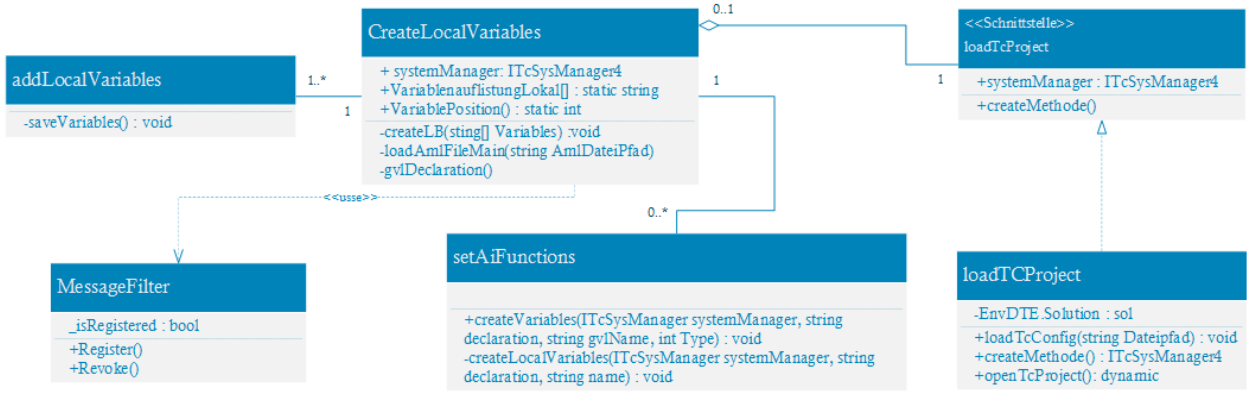

Abbildung 7-29: Klassendiagramm-Erstellung von lokalen Variablen, Quelle: Eigene Darstellung

7.3.2.4 Zuordnung von Variablen zu deren I/O-Gegenstücken

Um globale Variablen einem Hardwaremodul zuzuordnen, muss eine globale Variablenliste eines bestehenden TwinCAT-Projektes durch den PLCopenXML-Export erstellt werden. Diese kann dann über eine Methode der Klasse „LoadXMLFile“ geladen und konvertiert werden. Mit der Methode „setVariableReference“ in der Klasse „setAiFunctions“ können anschließend die gewünschten Verknüpfungen hergestellt werden. Im nachfolgenden UML-Klassendiagramm in der Abbildung 7-30 sind die beschriebenen Klassen grafisch dargestellt:

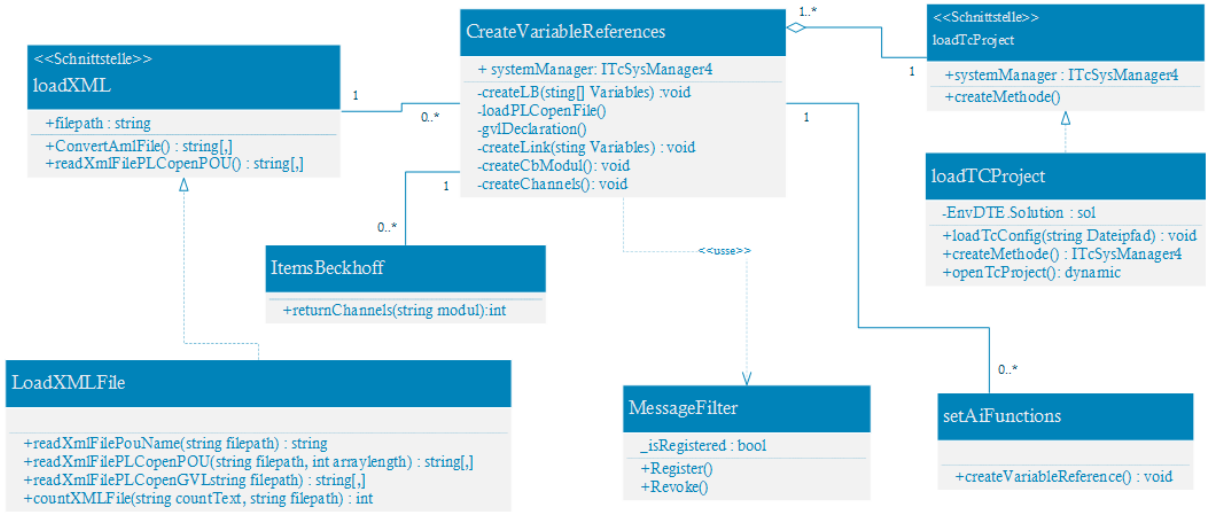

Abbildung 7-30: Klassendiagramm-Erstellung der Variablenzuordnung, Quelle: Eigene Darstellung

7.3.2.5 Erstellung von POUs

In der Abbildung 7-31 sind die Strukturen und Beziehungen der implementierten Klassen für die Erstellung von POUs in einem UML-Klassendiagramm grafisch dargestellt. Durch die „setAiFunctions“-Klasse kann die benötigte Methode zur Erstellung einer POU angewendet werden.

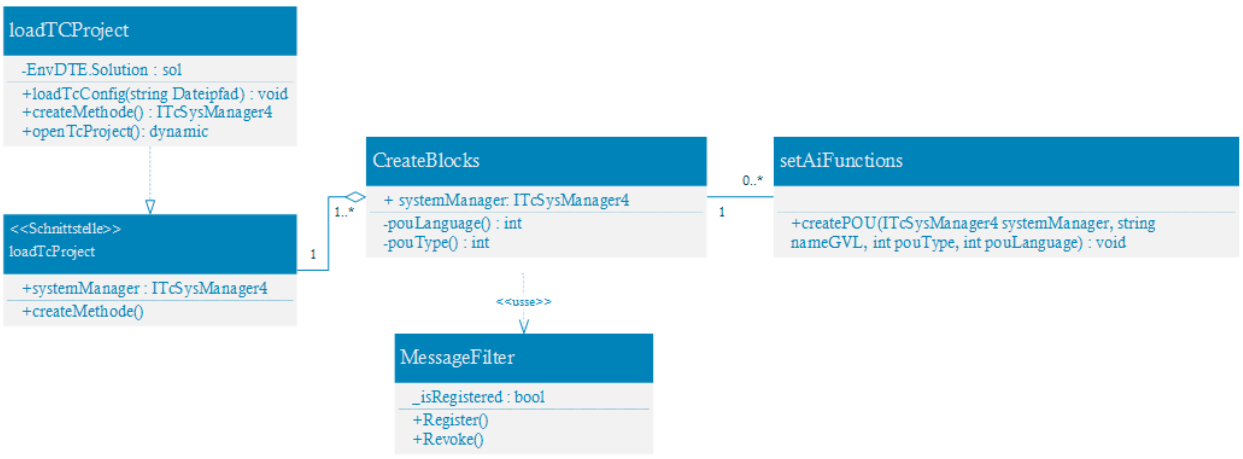

Abbildung 7-31: Klassendiagramm-Erstellung von POUs, Quelle: Eigene Darstellung

7.3.2.6 Erstellen von automatisierten Quellecodes

Für die Erstellung von automatisierten Quellcodes werden die in der Abbildung 7-32 beschriebenen Klassen und deren Methoden benötigt. Beispielsweise kann mit der Methode „createSourcecode“ in der Klasse „CreateSourcecode“ eine Verknüpfung zwischen den Variablen durchgeführt werden. Das Erstellen dieser Verknüpfung im TwinCAT3 kann durch das Anwenden der Methode „createSourcecode“ in der Klasse „setAiFunctions“ durchgeführt werden.

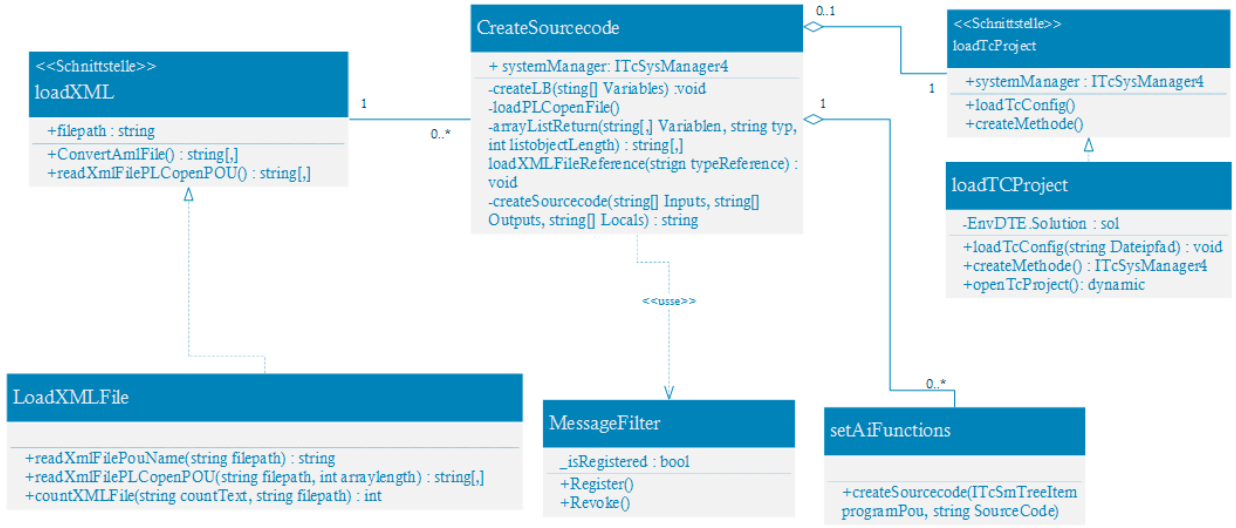

Abbildung 7-32: Klassendiagramm-Erstellung von automatisierten Quellcodes, Quelle: Eigene Darstellung

7.3.2.7 Befüllen einer Vorlagedatei in Excel-Format

Das nachfolgende UML-Klassendiagramm in der Abbildung 7-33 zeigt die implementierten Klassen und deren Methoden für die Befüllung einer Vorlagedatei in Excel-Format. Durch die Klasse „LoadAmlFile“ werden die Methoden zum Laden und Konvertieren einer AML-Datei bereitgestellt. Mit der Methode „fillExcelFile“ in der Klasse „CreateIoChecklist“ wird die Excel-Datei mit Daten vervollständigt.

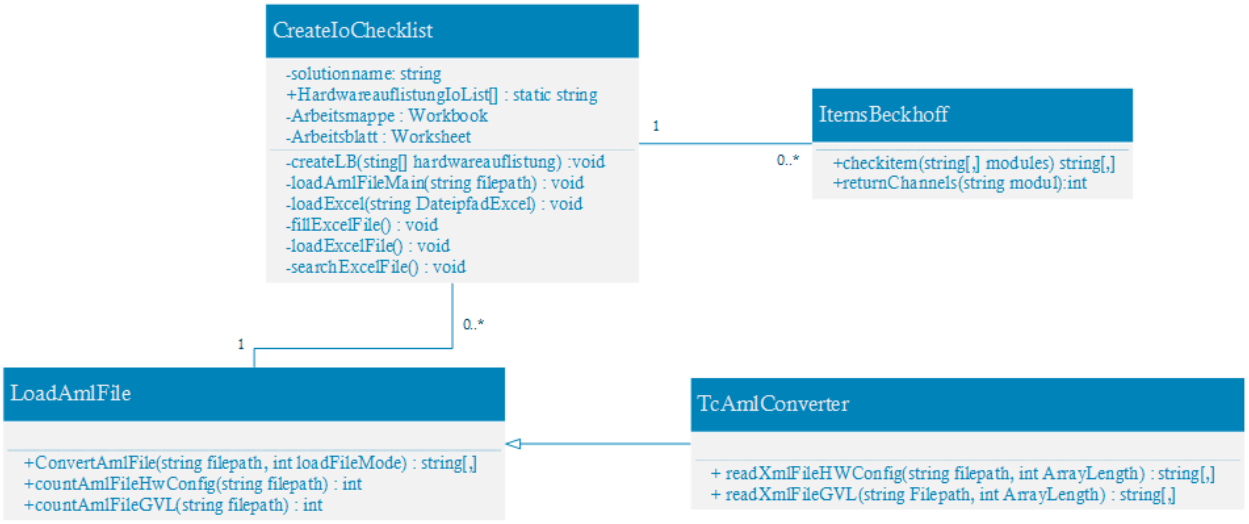

Abbildung 7-33: Klassendiagramm-Erstellung von I/O-Checklisten, Quelle: Eigene Darstellung

7.4 Implementierung

Die Phase der Implementierung behandelt die Übertragung der in der Anforderung- und Designphase erstellten Modelle in die Entwicklungsumgebung Visual Studio zur Erstellung des KS-SPS-Project-Creators. Die Umsetzung erfolgt in der Programmiersprache C# und mithilfe der Verwendung von Windows Forms Anwendungen. In den nachfolgenden Punkten wird die Implementierung der erstellten Konzepte näher beschrieben.

7.4.1 Graphical User Interface (GUI)

Die Bedienung der KS-SPS-Project-Creator Anwendung ist mittels einer grafischen Benutzeroberfläche, welche im englischen Graphical User Interface (GUI) genannt wird, möglich. So können sämtliche Funktionen mittels Mausklick ausgewählt werden bzw. werden alle relevanten Daten in den dafür vorgesehenen Listboxen angezeigt.

7.4.1.1 Implementierung der GUI des Hauptmenüs

Das Hauptmenü dient, wie schon im Punkt 7.3.1.1 beschrieben, zur Auswahl weiterer Unterprogramme, welche über die jeweiligen DropDown-Menüs anwählbar sind. Damit der Menürahmen in jedem Unterprogramm zur Verfügung steht, wird nur der innere Rahmen, der in der Abbildung 7-34 schwarz markiert ist, für das jeweilige Unterprogramm neu geladen. Die Menüleiste bleibt somit erhalten. Ist bereits ein Unterprogramm geladen, so wird diese beim Laden eines weiteren Unterprogrammes geschlossen. Dies verhindert Fehler bei der Erstellung mehrerer Instanzen, welche über das Automation Interface auf die Entwicklungsumgebung TwinCAT zugreifen.

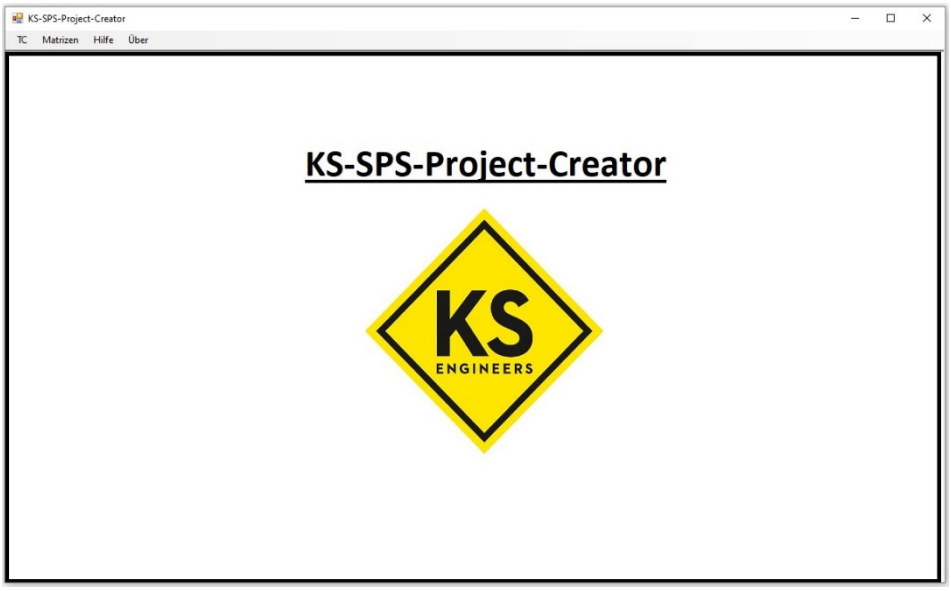

Abbildung 7-34: GUI-Hauptfenster, Quelle: Eigene Darstellung

7.4.1.2 Implementierung der GUI zur Erzeugung der Hardwarekonfiguration

Die grafische Oberfläche für die Erstellung einer Hardwarekonfiguration in TwinCAT 3, die in der Abbildung 7-35 dargestellt ist, wurde gemäß dem modellierten Konzept in Punkt 7.3.1.2 mittels einer Windows-Forms-Anwendung umgesetzt. Über die Eingabe eines Projektnamens wird der Name des zu erstellenden Projektes definiert, welcher im Zuge der Projekterstellung dem jeweiligen TwinCAT 3-Projekt zugewiesen wird. Auf der linken Seite befinden sich die Buttons zum Öffnen der TwinCAT 3 Entwicklungsumgebung, zum Erstellen eines neuen leeren Projektes, und zum Laden eines Vorlageprojektes mittels „Laden eines Template“. Als Vorlageprojekt wurde der Pfad zum Standardvorlageprojekt von TwinCAT in der Software hinterlegt. Diese ist bei jeder TwinCAT Installation unter dem gleichen Pfad aufrufbar.

Auf der rechten Seite der Oberfläche befinden sich die Bedienfelder für das Laden der AML-Datei sowie für die Anpassung der Positionen der Teilnehmer. Die Auswahl einer AML-Datei kann mittels des „Suche der AML-Datei“ Buttons durchgeführt werden. Dieser öffnet ein Suchfenster, in welchem die Datei über den Explorer von Windows gesucht werden kann. Mittels des Buttons „Hardwarekonfiguration erstellen“ wird die Hardwarekonfiguration, auf Basis der in der Mitte der Oberfläche dargestellten Auflistung, erstellt.

Bei dem Auflistungsfeld in der Mitte handelt es sich um ein ListBox-Feld, welches, wie schon erwähnt, alle durch die AML-Datei eingelesenen Teilnehmer anzeigt. Dieses wird mithilfe des Buttons „Laden der AML-Datei“, sofern eine gültige AML-Datei ausgewählt wurde, befüllt. Ebenfalls besteht die Möglichkeit einen Teilnehmer zu markieren und diesen über die Funktion „Teilnehmer bearbeiten“ anzupassen. Die Funktion öffnet ein weiteres Eingabefeld, in welchem der Name und der Teilnehmertyp manuell einzugeben sind.

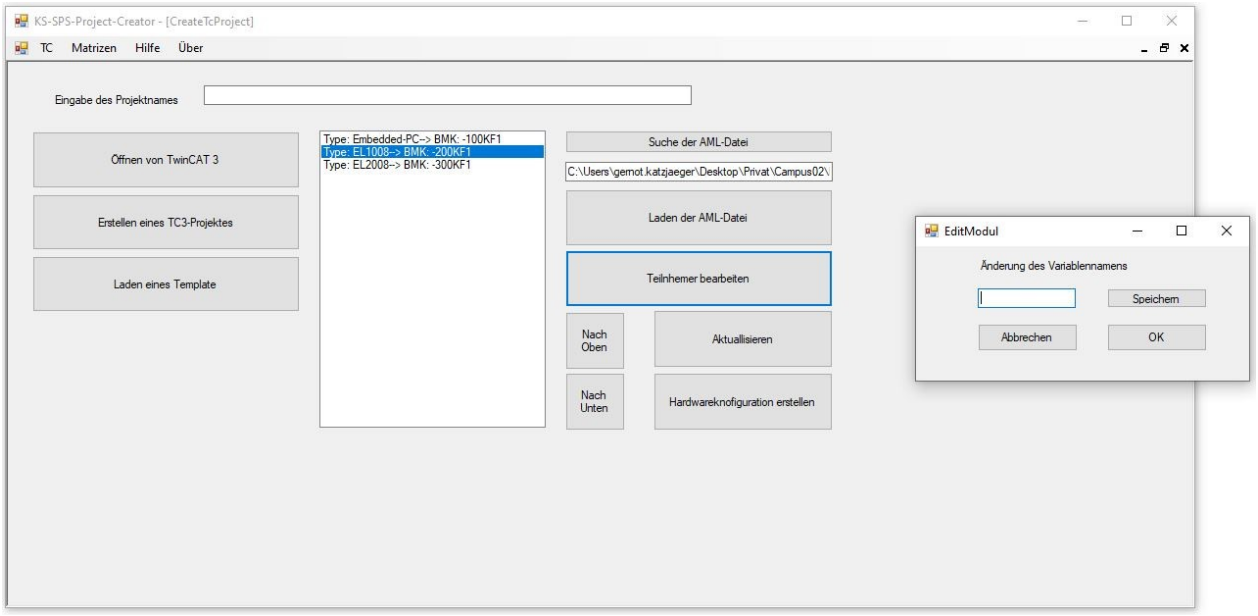

Abbildung 7-35: GUI - Konfigurationserstellung, Quelle: Eigene Darstellung

7.4.1.3 Implementierung der GUI zur Erzeugung von globalen Variablenlisten und Variablen

Die Erstellung von globalen Variablenlisten und deren Variablen kann in einem bestehenden TwinCAT 3-Projekt durchgeführt werden. Daher muss zu Beginn ein bestehendes Projekt geladen werden. Die grafische Benutzeroberfläche, die für die Durchführung der dafür notwendigen Funktionen dient, ist in der Abbildung 7-36 dargestellt.

Um einen gewissen Grad an Automatisierung zu erreichen, kann eine bestehende AML-Datei mittels „AML-Dateipfad suchen“ geladen werden. Die geladenen Variablen werden in der Auflistung dargestellt. Bevor diese Variablen in einer neuen Variablenliste erzeugt werden, können neue hinzugefügt, bestehende angepasst oder gelöscht werden. Anschließend muss ein Name für die neu zu erstellende globale Variablenliste definiert werden.

Über den Button „GVL in TC3 laden“ werden die Automation Interface Befehle für die Erstellung der Variablenliste sowie der definierten Variablen ausgeführt. Wird eine Variablenliste mit bereits vorhandenen Namen erstellt, wird dies durch eine Fehlermeldung signalisiert.

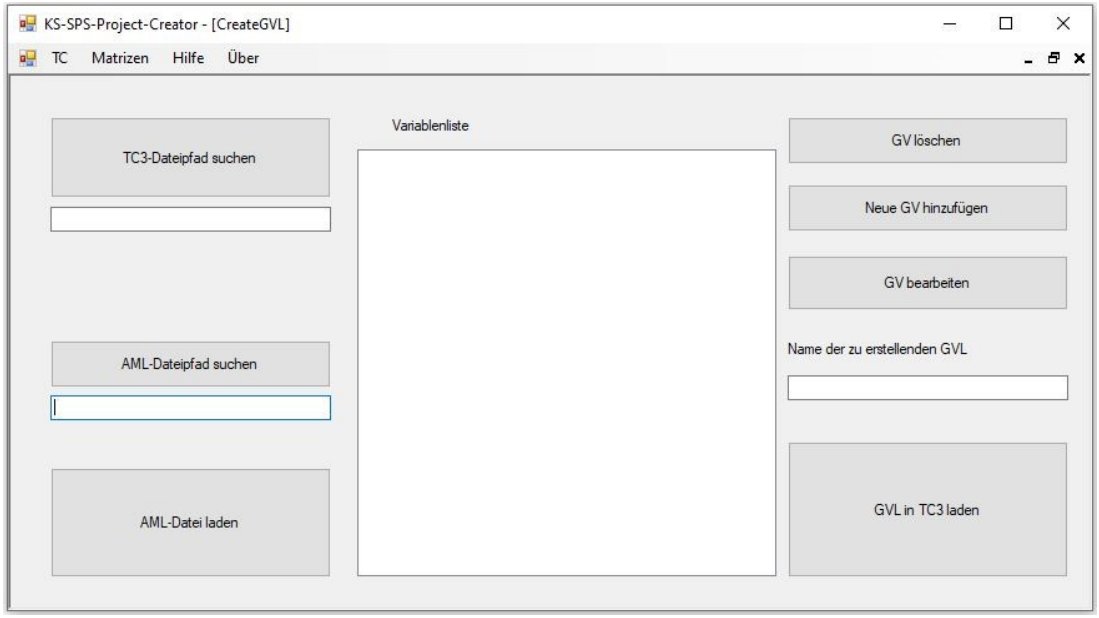

Abbildung 7-36: GUI - Erstellung von globalen Variablenlisten und deren Variablen, Quelle: Eigene Darstellung

7.4.1.4 Implementierung der GUI für die Zuordnung der Variablen zu deren I/O-Gegenstücken

Globale Variablen besitzen die Eigenschaft, dass auf diese uneingeschränkt in einem Programm zugegriffen werden kann. Somit können diese für die Verknüpfung von I/O-Modulen genutzt werden, was mittels der grafischen Benutzeroberfläche, dargestellt in der Abbildung 7-37, ermöglicht werden soll. Für die Auswahl der zu verknüpfenden Variablen müssen diese über eine PLCopenXML-Datei geladen werden. Alle gültigen Variablen werden in der Variablenliste geladen und angezeigt.

Um eine Variable der Variablenliste mit einem Kanal eines Hardwaremoduls zu verlinken, muss dieser ebenfalls ausgewählt werden. Dies ist durch Eingabe des Modulnamens möglich. Zusätzlich müssen der Typ des Moduls und der gewünschte Kanal, welcher mit der Variable verknüpft werden soll, ausgewählt werden.

Sind die gewünschte Variable und das Modul inkl. dessen Typ und Kanal ausgewählt, kann die Verlinkung mithilfe der Funktion „Verlinkung erstellen“ durchgeführt werden. Die Kanäle eines Moduls können nach der Modulauswahl geladen werden.

Mit dem Button „Alle Verlinkungen löschen“ können die bestehenden Verlinkungen gelöscht werden. Somit wäre es möglich, eine neue Verlinkung zu erstellen.

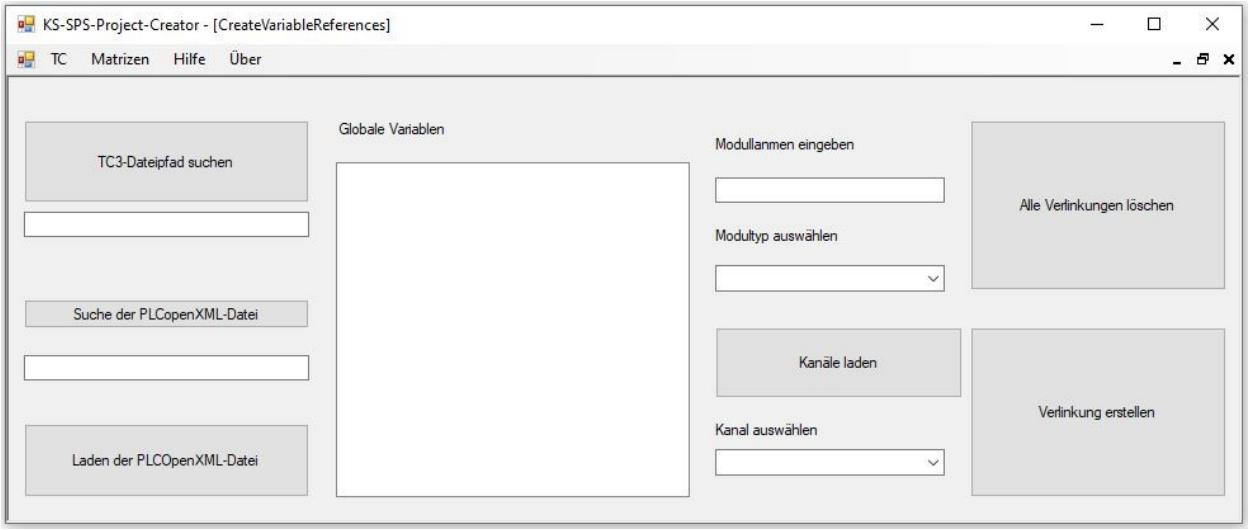

Abbildung 7-37: GUI - Variablenzuordnung, Quelle: Eigene Darstellung

7.4.1.5 Implementierung der GUI für die Erstellung von POUs

Um mithilfe des KS-SPS-Project-Creators POUs zu erstellen, wurde die in der Abbildung 7-38 grafische Benutzeroberfläche erzeugt. Diese ermöglicht es ein bestehendes TwinCAT3-Projekt über den Explorer zu suchen und dieses anschließend zu öffnen.

Ist das Projekt geladen, so kann der Name des Programmbausteines, die Programmiersprache und der Typ des zu erstellenden POU eingegeben bzw. ausgewählt werden. Sind diese Daten befüllt, kann mittels der Funktion „Neuen Programmbaustein erstellen“ die Erstellung eines neuen POU, eingeleitet werden.

Im DropDown-Menü für die Programmiersprache stehen folgende Sprachen zur Verfügung:

- IL (Instruction List)
- FBD (Function Block Diagram)
- ST (Structured Text)
- LD (Ladder Diagram)

Im DropDown-Menü für den Programmblock stehen folgende Typen zur Verfügung:

- Funktion
- Funktionsbaustein
- Programm

Durch die Einleitung der Erstellung werden die ausgewählten und eingegebenen Daten mittels Automation Interface Funktionen an das TwinCAT-Projekt übergeben und die notwendigen Funktionen für die Erstellung ausgeführt.

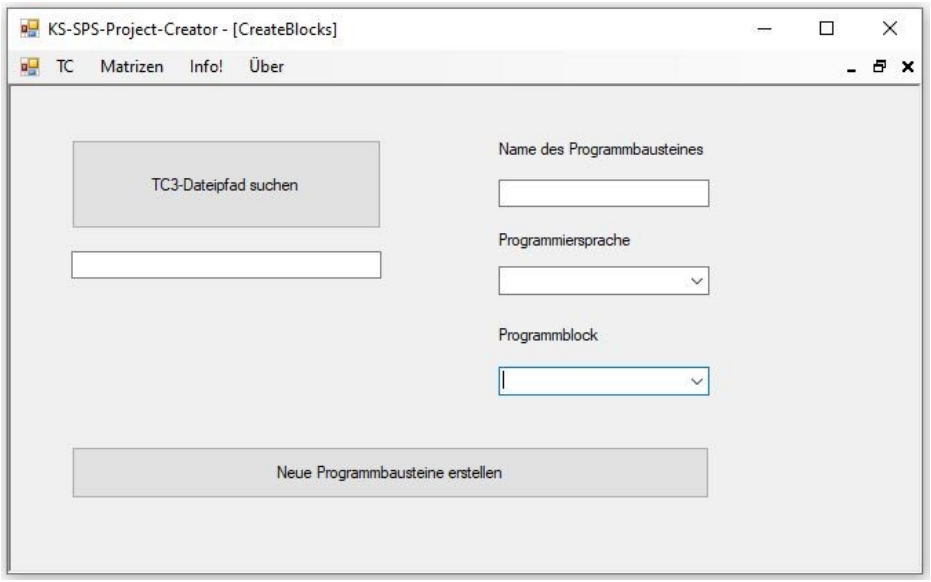

Abbildung 7-38: GUI - Programmbausteinerstellung, Quelle: Eigene Darstellung

7.4.1.6 Implementierung der GUI zur Erzeugung von lokalen Variablen

Lokale Variablen sind, anders als bei den globalen Variablen, einem POU zugeordnet und werden in diesem erstellt. Somit kann der Name einer bestehenden POU in die grafische Oberfläche, welche in der Abbildung 7-39 dargestellt ist, eingegeben werden (siehe auch Kapitel 7.2.4).

Bevor die Erstellung der Variablen durchgeführt werden kann, werden die zu erstellenden Variablen in der Variablenliste definiert. Mit der Funktion „Neue Variable hinzufügen, und „Variable löschen“ kann diese Auflistung bearbeitet werden. Wird eine neue Variable hinzugefügt, so muss der Name, der Dateityp und der Variablentyp definiert werden. Durch das Dropdown-Menü „Type der Variable“ stehen folgende drei Auswahlmöglichkeiten zur Verfügung:

- Input
- Output
- Lokal

Die Vorab-Variablenliste zeigt die aktuellen Variablen und dessen Datentypen an, welche für die Erstellung im TwinCAT-Projekt bereitstehen. Ist die Erstellung der Vorab-Variablenliste abgeschlossen, so kann diese mit der Funktion „Variablen der Auflistung im TC3 erstellen“ im ausgewählten Programmbaustein erstellt werden.

Um die Variablen in den gewünschten POU zu übertragen, muss der Name des POUs sowie dessen Typ ausgewählt werden. Die Auswahl des Typs kann im Dropdown-Menü „Variablentyp“ erfolgen. Dieser Vorgehensweise ermöglicht die Auswahl der Typen „Funktion“, „Programm“ und „Funktionsbaustein“.

Werden diese Eigenschaften nicht definiert, so gibt die Anwendung eine Fehlermeldung aus bzw. kann der POU nicht korrekt im TwinCAT abgearbeitet werden.

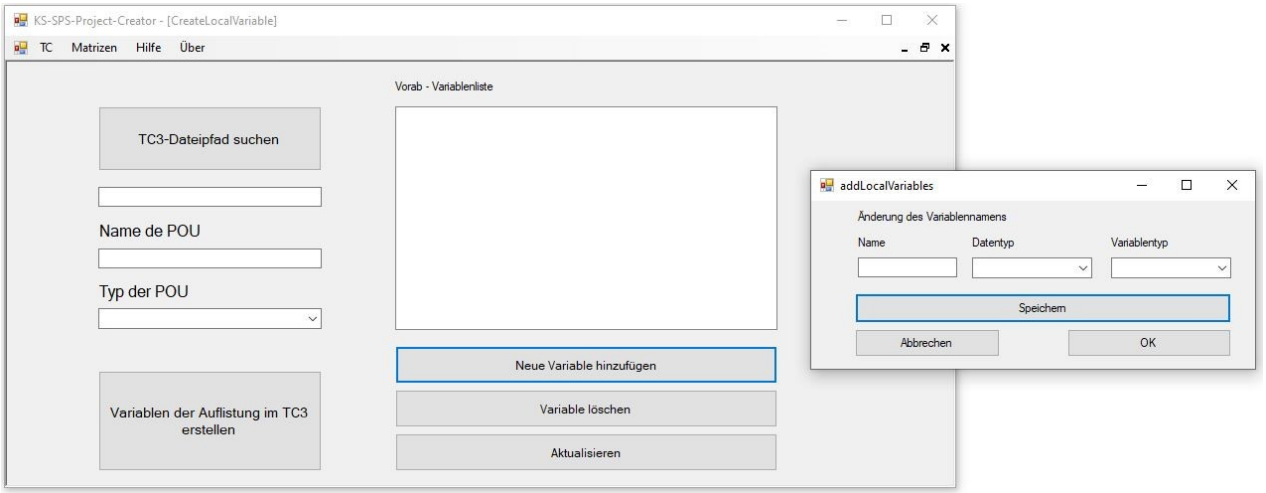

Abbildung 7-39: GUI - Erstellung von lokalen Variablen, Quelle: Eigene Darstellung

7.4.1.7 Implementierung der GUI für die Erstellung von automatisierten Quellcodes

Für die Erstellung von Quellcodes werden Input-Variablen, welche für die Abfrage der Bedingungen genutzt werden, sowie Output-Variablen, welche für die Speicherung des Ergebnisses verwendet werden, benötigt. Die GUI der Applikation, die in der Abbildung 7-40 dargestellt ist, bietet die Möglichkeit „UND“ und „ODER“ Verknüpfungen zu erstellen.

Die Abfragen werden in POUs erstellt, somit muss zu Beginn der Quellcodeerstellung der gewünschte Programmbaustein durch Import der zugehörigen PLCopenXML-Datei gewählt werden. Wird dieser geladen, so werden die vorhandenen Variablen, sofern diese existieren, zwischen Input-Variablen, Output- und Lokale-Variablen getrennt und in separaten Listboxen aufgelistet.

Um eine Verknüpfung zu definieren, müssen mindestens zwei Variablen angewählt werden. Mit den Funktionen „UND-Verknüpfung ausführen“ oder „ODER-Verknüpfung ausführen“ wird die jeweilige Verknüpfung zwischen den ausgewählten Variablen erstellt.

Um eine erfolgreiche logische Verknüpfung von 2 Variablen durchführen zu können, muss ebenfalls eine Output-Variable definiert sein. Dieser wird bei erfolgreicher Abfrage der Wert „true“ zugewiesen. Ebenfalls wird automatisiert eine Bedingung erstellt, die dieser Variablen den Wert „false“ zuweist, wenn die Abfrage nicht erfolgreich durchgeführt werden konnte.

Sind alle Verknüpfungen definiert, so kann basierend auf der Funktion „Quellcode im TC erstellen“ der Quellcode im ausgewählten Programmbaustein erzeugt werden.

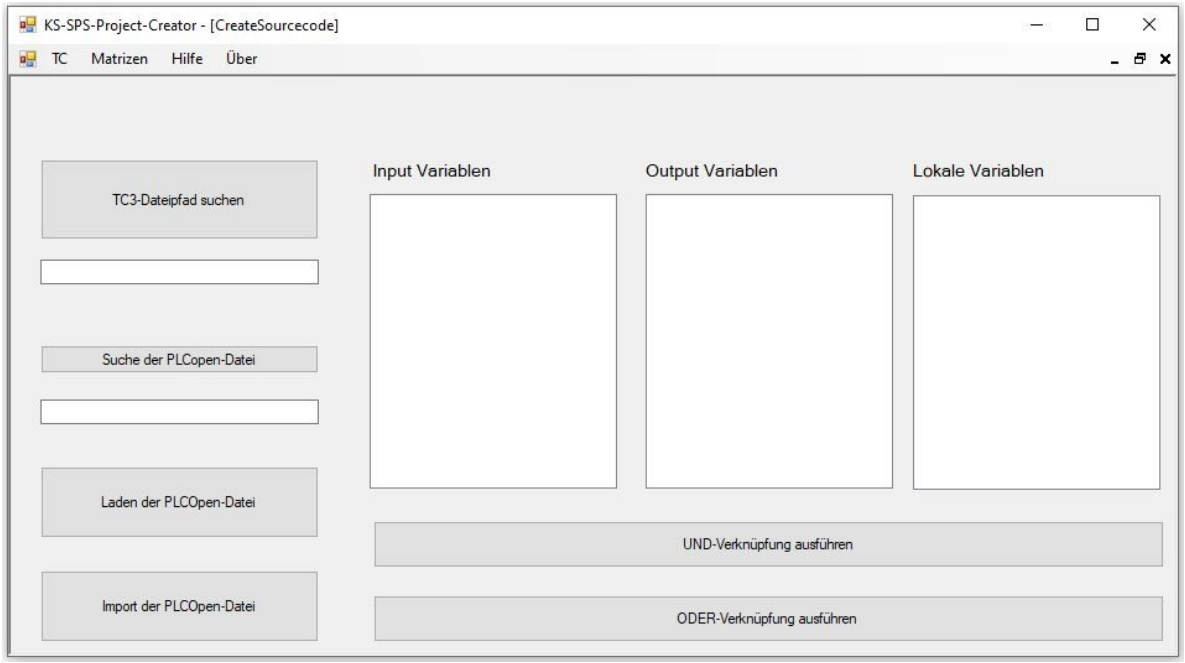

Abbildung 7-40: GUI - Quellcodeerstellung, Quelle: Eigene Darstellung

7.4.1.8 Implementierung der GUI für die Befüllung einer Vorlagedatei in Excel-Format

Die automatisierte Befüllung einer Excel-Vorlagedatei mit den Hardwareinformationen einer im EPLAN konstruierten SPS-Steuerung wird mittels Laden und Konvertieren einer AML-Datei realisiert. Daher muss die zu importierende AML-Datei anhand der Funktion „AML-Datei suchen“ ausgewählt werden. Anschließend werden durch die Funktion „Hardwarekonfiguration aus AML-Datei laden“ sämtliche relevanten Hardwareinformationen aus der gewählten AML-Datei geladen und in der Auflistung angezeigt.

Um die Befüllung einer Vorlagedatei erfolgreich durchführen zu können, muss eine gültige Datei über die Funktion „Suche nach der zu befüllenden Excel-Datei“ ausgewählt werden. Ist eine Auflistung vorhanden, kann nach erfolgreicher Auswahl einer Excel-Datei ein Befüllen der ausgewählten Datei stattfinden. Dargestellt wird die Durchführung der grafischen Implementierung, der GUI zur Befüllung einer Vorlagedatei, in der nachfolgenden Abbildung 7-41.

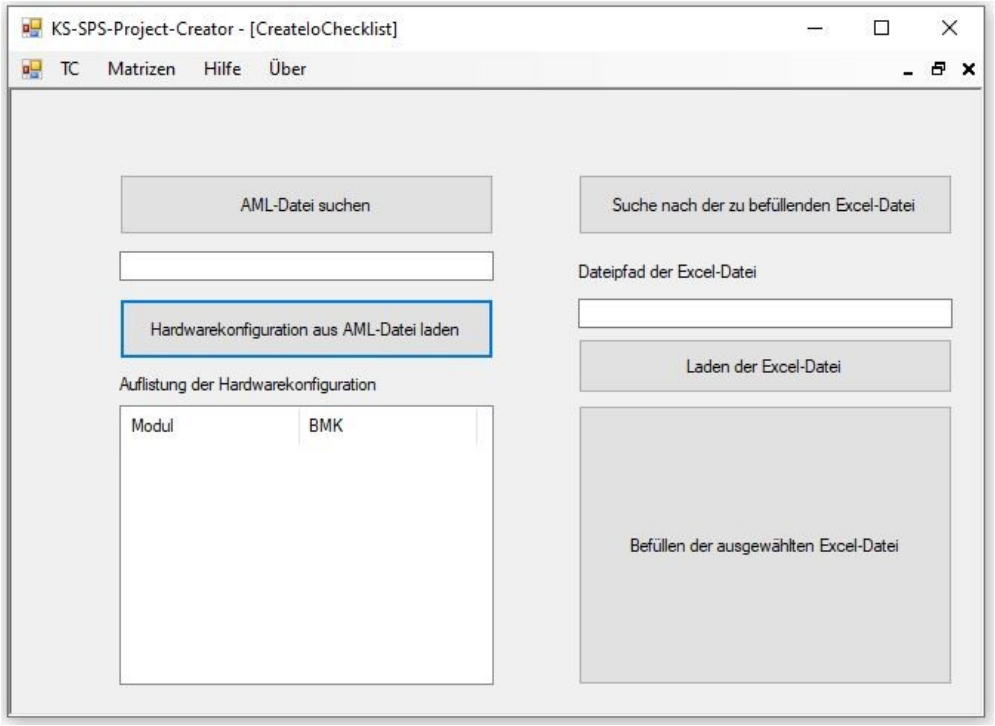

Abbildung 7-41: GUI - Befüllung einer Excel-Vorlagedatei, Quelle: Eigene Darstellung

7.4.2 Methoden und Funktionen

Die nachfolgenden Implementierungen beziehen sich auf die Umsetzung der benötigten Funktionen in Form von Quellcodes in Visual Studio. Dafür wurden teilweise die konzipierten UML-Klassendiagramme als Vorlage verwendet aber auch Teilbereiche des Quellcodes “frei“ entwickelt.

7.4.2.1 Implementierung der TC3-Projekterstellung

Für die Erstellung eines TwinCAT 3 Projektes und die Erstellung der gewünschten Hardwarekonfiguration wurde die Implementierung anhand des Konzeptes in Punkt 7.3.2.1, welches mittels eines UML-Klassendiagrammes erstellt wurde, durchgeführt. Die wesentlichen Funktionen dieses Konzeptes sind das Laden einer AML-Datei und dessen Konvertierung und Filterung der Daten, sowie die Erstellung und Anpassung einer Auflistung der konvertierten und gefilterten Daten. Aus dieser angepassten Auflistung wird die Hardwarekonfiguration, durch die Funktion „Create Config“ im geöffneten TwinCAT 3-Projekt erstellt. Diese Funktionen werden anschließend in der Quellcodebeschreibung näher erläutert.

Die im nachfolgenden Quelltext beschriebene Methode leitet die eigentliche Erstellung einer Hardwarekonfiguration ein. Diese wird mithilfe eines Button-One-Click Ereignisses aufgerufen. Die Klasse „ItemsBeckhoff“ prüft die Hardwarekonfiguration auf vorhandene Master. Ebenfalls findet eine Prüfung der Gültigkeit der von der AML-Datei eingelesenen EtherCAT-Module statt. Die gültige Hardwareauflistung wird im String-Array „hwconfig“ gespeichert. Mittels der Methode „createConfig“ wird die Hardwarekonfiguration im TwinCAT 3 auf Basis der gültigen Hardwareauflistung erstellt.

```
/*============================================================================================
Funktion für das Erstellen einer Hardwarekonfiguration in TwinCAT 3
============================================================================================ */

        private void btnCreateCofnig_Click(object sender, EventArgs e)
        {
            ItemsBeckhoff items = new ItemsBeckhoff();

            //Gültige Hardwareitems aus dem String-Array "Hardwareauflistung" ermitteln
            string[,] hwconfig = items.checkitem(Hardwareauflistung);

            //Anzahl der Master aus dem String-Array "Hardwareauflistung" ermitteln
            int[,] AnzahlMasters = items.checkMaster(Hardwareauflistung);

            CreateConfigAI cc = new CreateConfigAI();
            cc.createConfig(systemManager, project, hwconfig, AnzahlMasters);

        }
```

Quelltext 1: Hauptmethode zu Erstellung der Hardwarekonfiguration, Quelle: Eigene Darstellung

Um durch Konvertierung einer AML-Datei in Visual Studio die erforderlichen Daten für die Erstellung einer Hardwarekonfiguration im TwinCAT 3 zu erhalten, muss das Programm die notwendigen Abfragen durchführen. Da eine AML-Datei dem XML-Format entspricht, können die bekannten XML-Klassen im Visual Studio angewendet werden. Um jedes Modul und dessen Betriebsmittelkennzeichen (BMK) aus einer AML-Datei zu erhalten, ist es notwendig, jeden Wert des „TypeName“ und „ProductDesignation IEC“ abzufragen und diese auszuwerten. In der nachfolgenden Abbildung 7-42 wird ein Auszug einer AML-TC3-Exportdatei dargestellt.

```
73      <Attribute Name="TypeName" AttributeDataType="xs:string">
74       <Value>EL1008</Value>
75      </Attribute>
76      <Attribute Name="TypeIdentifier" AttributeDataType="xs:string">
77       <Value>OrderNumber:EL1008</Value>
78      </Attribute>
79      <Attribute Name="ProductDesignation IEC" AttributeDataType="xs:string">
80       <Value>-200KF1</Value>
81      </Attribute>
```

Abbildung 7-42: XML-File für Hardwareerstellung, Quelle: Eigene Darstellung

Der nachfolgende Quelltext dient zur Konvertierung der AML-Datei in verwendbare Daten, welche als Ausgabe sämtliche Hardwareelemente in einem 2-Dimensionalen Array speichern.

```
/*=============================================================================================
Konvertierung der AML-Datei zur Erstellung der Hardwarekonfiguration
============================================================================================= */
 public string[,] readXmlFileHwConfig(string FilePath, int ArrayGroesse, int AmlLoadMode)
        {
            int arraystelle = 0;
            String[,] AmlDaten_roh = new String[ArrayGroesse,2];
            XmlReader AmlFile = XmlReader.Create(FilePath);
            bool BmkAviable = false;
            if(arraystelle < ArrayGroesse)
            {
             while (AmlFile.Read())
              {
               if (AmlFile.IsStartElement()) // Start eines Elements im AML-File
                  {
                   AmlFile.MoveToFirstAttribute();
                  if (AmlFile.Value == "TypeName" && arraystelle < (ArrayGroesse - 1))
                     {
                      AmlFile.ReadToFollowing("Value");
                      AmlDaten_roh[arraystelle, 0] = AmlFile.ReadElementContentAsString();
                      arraystelle = arraystelle + 1;
                      BmkAviable = true;}
                             if (BmkAviable == true)
                                    {
                             if (AmlFile.Value == "ProductDesignation IEC")
                             {   AmlFile.ReadToFollowing("Value");
              AmlDaten_roh[arraystelle - 1, 1] = AmlFile.ReadElementContentAsString();
                             BmkAviable = false;
                             }}}}}
            return AmlDaten_roh; // Array mit den aufbereiteten AML-Daten als
Rückgabewert

        }
```

Quelltext 2: Konvertierung der AML-Datei, Quelle: Eigene Darstellung

Der unten dargestellte Programmausschnitt befasst sich mit der Erstellung der Hardwarekonfiguration in einem TwinCAT 3-Projekt. Diese wird auf Basis der eingelesenen AML-Datei und der durchgeführten Anpassungen erstellt. Als Übergabevariable der Auflistung der zu erstellenden Hardware an diese Methode dient das 2-Dimensionalen String-Arrays „hardwareconfigCC“. Die eigentliche Erstellung der EtherCAT-Teilnehmer im TwinCAT 3 wird mittels Ausführung von Automation Interface Befehlen, wie den „CreateChild-Befehl“, durchgeführt.

```
/*============================================================================================
 Funktion für das Erstellen der Hardwarekonfiguration in TwinCAT 3
============================================================================================*/

public void createConfig(ITcSysManager4 systemManager, dynamic project, string[,]
hardwareconfigCC, int[,] MasterStatus)
        {
            bool MasterVorhanden = false;
            MessageFilter.Register();
            systemManager = (ITcSysManager4)project.Object;
            ITcSmTreeItem devices = systemManager.LookupTreeItem("TIID");
            if(MasterStatus[0,0] == 1)
            {
                ITcSmTreeItem ethercatMaster = devices.CreateChild("EtherCAT Master",
111, null, null);

                MasterVorhanden = true;
                if (MasterVorhanden == true)
                {
                     ethercatMaster.CreateChild("EK1200", 9099, "", "EK1200-5000-0000");
                    ITcSmTreeItem ek1200 = systemManager.LookupTreeItem("TIID^EtherCAT
                    Master^EK1200");
                    try
                    {
                        for (int i = 0; i < hardwareconfigCC.Length /3; i++)
                        ek1200.CreateChild(hardwareconfigCC[i, 0],
Convert.ToInt32(hardwareconfigCC[i, 1]), "", hardwareconfigCC[i, 2]);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show(Convert.ToString(ex));
                    }
                }
            }
            else
            {
                  MessageBox.Show("Kein Master vorhanden", "Fehlermeldung",
                  MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            MessageFilter.Revoke();

        }
```

Quelltext 3: Erstellung der EtherCAT-Teilnehmer, Quelle: Eigene Darstellung

7.4.2.2 Implementierung der Erstellung von globalen Variablenlisten und Variablen

Eine wesentliche Funktion bei der Erstellung einer globalen Variablenliste ist die Erstellung einer Auflistung der zu deklarierenden Variablen. Ebenfalls wird der Anfang und das Ende des Deklarationsbereiches einer globalen Variablenliste mit „VAR_GLOBAL“ für den Start und mit „END_VAR“ für das Ende gekennzeichnet.

Der nachfolgend dargestellte Auszug des Programmcodes zeigt die Aufteilung der Input- und Outputvariablen. Je nach Zuteilung wird der Parameter „%I*“, für eine Inputvariable oder „%Q*“, für eine Outputvariable, dem String hinzugefügt.

Sind alle verfügbaren Variablen, welche sich im Array „GvlAuflistung“ befinden, die Zuweisungsschleife durchlaufen und der String-Variable „gvlDecl“ hinzugefügt, so wird diese String-Variable als Return-Wert zurückgegeben.

```
/*============================================================================================
Erstellen der globalen Variablenliste
============================================================================================*/

        private string gvlDeclaration()
        {
            int arrayLength = GvlAuflistung.Length / 3;
            string gvlDeclStart = @"VAR_GLOBAL";
            string gvlDeclEnd = "\n END_VAR";
            string gvlDeclMiddle = "";
            string gvlDeclMiddleDef = "";
            string gvlDecl="";

            for (int a =0; a < arrayLength; a++)// Erstellen des Deklarationstextes
            {
                if(GvlAuflistung[a,1] == "Input")
                {
                  gvlDeclMiddleDef = "\n" + GvlAuflistung[a, 0] + " AT %I*: " +
                  GvlAuflistung[a, 2] + ";\n";
                }
                if (GvlAuflistung[a, 1] == "Output")
                {
                    gvlDeclMiddleDef = "\n" + GvlAuflistung[a, 0] + " AT %Q*: " +
                    GvlAuflistung[a, 2] + ";\n";
                }
                gvlDeclMiddle = gvlDeclMiddle + gvlDeclMiddleDef;
            }
            gvlDecl = gvlDeclStart + gvlDeclMiddle + gvlDeclEnd;
            return gvlDecl;

        }
```

Quelltext 4: String-Array der Variablenliste, Quelle: Eigene Darstellung

7.4.2.3 Implementierung der Erstellung von lokalen Variablen

Für die Erstellung von lokalen Variablen ist es essenziell, diese nach Variablentyp zu trennen. So werden Input- und Output-Variablen separat deklariert. Da jede Erstellung der Variablenliste den Deklarationsbereich der POU löscht und mit den neuen Informationen befüllt, muss als Übergabeparameter auch der Typ der POU definiert und übertragen werden. Je nach Typ wird dieser zu Beginn des Deklarationsbereiches definiert. Im nachfolgenden Quelltext sind diese Funktionen genauer beschrieben.

```
/*============================================================================================
Definition und Deklaration der lokalen Variablen
============================================================================================*

private string gvlDeclaration(string pouName)
        {
            int arrayLength = VariablenStelle;
            string gvlDeclPOU = "";
            string gvlDeclStart =@"\n VAR";
            string gvlDeclEnd = "\n END_VAR";
            string gvlDeclMiddle = "";
            string gvlDeclMiddleDef = "";
            string gvlDecl = "";

            switch(cbPouTyp.SelectedItem.ToString())//Definition der POU
            {
                case "Funktionsbaustein":
                    gvlDeclPOU = "FUNCTION_BLOCK";
                    break;
                case "Funktion":
                    gvlDeclPOU = "FUNCTION";
                    break;
                case "Programm":
                    gvlDeclPOU = "PROGRAM";
                    break;
            }

            for (int a = 0; a < arrayLength; a++) // Erstellen des Deklarationstextes
            {
                switch (VariablenauflistungLokal[a,2])
                {
                    case "Input":
                        gvlDeclStart = @"VAR_Input";
                        break;
                    case "Output":
                        gvlDeclStart = @"VAR_Output";
                        break;
                    case "Local":
                        gvlDeclStart = @"VAR";
                        break;
                }
   gvlDeclMiddleDef ="\n"+ gvlDeclStart + "\n" + VariablenauflistungLokal[a, 0] + " : " +
   VariablenauflistungLokal[a, 1] + ";\n"+ gvlDeclEnd + "\n";
   gvlDeclMiddle = gvlDeclMiddle + gvlDeclMiddleDef;

            }
```

Quelltext 5: Deklaration von lokalen Variablen, Quelle: Eigene Darstellung

7.4.2.4 Implementierung der Variablenzuordnung

Für die Implementierung der Variablenzuordnung ist eine Aufteilung der Variablen, welche aus der PLCopenXML-Datei eingelesen werden, notwendig. Der unten angeführte Quelltext stellt die Aufteilung der durch die PLCopenXML-Datei geladenen Variablen in In- und Outputtypen dar. Gespeichert werden die aufgeteilten Variablen in einem 2-Dimensionalen String-Array, welches als Rückgabewert der Methode ausgegeben wird.

```
/*============================================================================================
Aufteilung der Variablen in In- und Outputvariablen
============================================================================================*/

        public string[,] readXmlFilePLCopenGVL(string FilePath)
        {
            int arraygroesse = countXMLFile("variable",FilePath);
            String[,] PLCopenXmlData = new String[arraygroesse,2];
            XmlReader PLCopenXmlFile = XmlReader.Create(FilePath);
            int i  = 0;
            while (PLCopenXmlFile.Read())
            {
                if (PLCopenXmlFile.IsStartElement()) // Abfrage, wenn ein neues Element
im AML-File beginnt
                {
                    if (PLCopenXmlFile.Name.ToString() == "variable")
                    {
                        PLCopenXmlData[i,0] =
PLCopenXmlFile.GetAttribute("name").ToString();
                        if(PLCopenXmlFile.GetAttribute("address").ToString() != null)
                        switch(PLCopenXmlFile.GetAttribute("address").ToString())
                            {
                                case "%I*":
                                    PLCopenXmlData[i, 1] = "Input";
                                    break;
                                case "%Q*":
                                    PLCopenXmlData[i, 1] = "Output";
                                    break;
                            }
                        i = i + 1;
                    }
                }
            }
            return PLCopenXmlData; // Array mit den gefilterten AML-Daten als
Rückgabewert

        }
```

Quelltext 6: Variablenaufteilung, Quelle: Eigene Darstellung

7.4.2.5 Implementierung der Erstellung von POUs in TC3

Für die Erstellung von POUs müssen sowohl die Programmiersprache als auch der Typ des POUs definiert werden. Die Switch-Anweisungen in den nachfolgenden Quelltexten führen Abfragen durch, um den Wert der gewünschten Programmiersprache oder Typs in der jeweiligen Methode zurückzugeben. Der Rückgabewert entspricht einem Zahlenwert, da das Automation Interface als Übergabewert für die Methoden zur Erstellung von POUs einen numerischen Wert erwartet.

```
/*============================================================================================
Definieren der Programmiersprache
============================================================================================ */
        private int pouLanguage()
        {string cb = cbIecLanguage.SelectedItem.ToString();
            int type = 0;
            switch (cb)
            {
                case "IL":
                    type = 2; //"Instruciton List"(AWL-Anweinsungsliste)
                    break;
                case "ST":
                    type = 1; //"Structured Text" (AS-Ablaufsprache)
                    break;
                case "FBD":
                    type = 4; //"Function Block Diagramm" (FBS-Funktionsbaustein-Sprache)
                    break;
                case "LD":
                    type = 6; //"Ladder Diagram" (KOP-Kontaktplan)
                    break;
                case "":
                    type = 1; //"Structured Text" als Default-Wert
                    break;
            }
            return type;}
```

Quelltext 7: Definieren der Programmiersprache, Quelle: Eigene Darstellung

```
/*============================================================================================
Definieren des POU-Typs
============================================================================================ */
        private int pouType()
        {string cb = cbBlocks.SelectedItem.ToString();
            int type = 0;

            switch (cb)
            {
                case "Funktionsbaustein":
                    type = 604;
                    break;
                case "Funktion":
                    type = 603;
                    break;
                case "Programm":
                    type = 602;
                    break;
                case "":
                    type = 604; //Funktionsbaustein als Default-Wert
                    break;
            }
            return type;}
```

Quelltext 8: Definieren des POU-Typs, Quelle: Eigene Darstellung

7.4.2.6 Implementierung der automatisierten Codegenerierung

Um Verknüpfungen zwischen den gewünschten Variablen herzustellen, können diese in den dafür vorgesehenen Auflistungen ausgewählt werden. Durch Abfragen des Index der markierten Zellen, werden die dazugehörigen Informationen in Arrays abgerufen. Die eigentlichen Quellcodes werden in einem String gespeichert und als Rückgabewert der Methode zurückgegeben. Dargestellt ist diese Funktion im nachfolgenden Quelltext:

```
/*============================================================================================
Erstellen einer UND- oder ODER-Verknüpfung zwischen Variablen
============================================================================================*/

   private string createSourcecode(string[] Inputs, string[] Output, string[] Locals, int
type)
        {
            int selectetdInputs = lbChooseIV.SelectedIndex;
            int selectedOutput = lbChooseOV.SelectedIndex;
            int selectedLocals = lbLocalVar.SelectedIndex;
            int numberVarsIn = 0;
            int numberVarsOut = 0;
            int numberVars = 0;
            string condition = "";
            string returnSourcecode = "";
            string typeReference = "";

            if (type == 1)
                typeReference = " AND ";// Definition einer AND-Verknüpfung
            if (type == 2)
                typeReference = " OR ";// Definition einer OR-Verknüpfung
            if (selectetdInputs > -1)
                numberVarsIn = 1;
            if (selectedLocals > -1)
                numberVarsOut = 1;

            numberVars = numberVarsIn + numberVarsOut;
            if (numberVars >1)
            condition = Inputs[selectetdInputs] +" = true" + typeReference +
Locals[selectedLocals] + " = true";// Variablen zuweisen
            else
            {
                if (selectetdInputs > 0)
                    condition = Inputs[selectetdInputs] + "= true";
                if (selectedLocals > 0)
                    condition = Locals[selectedLocals] + "= true";
            }

            returnSourcecode = "IF (" + condition + ") THEN \n" + Output[selectedOutput]
+ " := true; " + "\n END_IF; ";
            return returnSourcecode;
        }
```

Quelltext 9: Erstellen einer Verknüpfung, Quelle: Eigene Darstellung

7.4.2.7 Implementierung der Befüllung von Excel-Listen

Um auf eine Excel-Datei in Visual Studio zugreifen zu können muss der Namespace „Microsoft.Office.Interop.Excel“ eingebunden werden. Anschließend ist es möglich, durch Instanziierung eines neuen Objektes Funktionen auszuführen, mit denen auf eine Excel-Datei lesend und schreibend zugegriffen werden kann.

Der dargestellte Quelltext zeigt den schreibenden Zugriff auf eine Excel-Datei. Zwei verschachtelte for-Schleifen führen die Befüllung der Datei mit den Hardwaremodulen und deren Kanäle durch. Für eine korrekte Befüllung ist es wichtig, die Startpositionen der Zellen zu ermitteln und diese zu definieren.

```
/*============================================================================================
Befüllung der I/O-Checkliste
============================================================================================*/

        private void btnSaveToExcelFile_Click(object sender, EventArgs e)
        {

            int listobject = 0;
            listobject = HardwareauflistungIoList.Length / 2;
            int zelle = 12;

            for (int i = 0; i < listobject; i++)
            {
    if (HardwareauflistungIoList[i, 1] != null && HardwareauflistungIoList[i, 0] != null)
                {
                    (Arbeitsblatt.Cells[i+ zelle, 2] as Excel.Range).Value2 =
                     HardwareauflistungIoList[i, 1];

                              (Arbeitsblatt.Cells[i + zelle, 3] as Excel.Range).Value2 =
                               HardwareauflistungIoList[i, 0];
                    ItemsBeckhoff channels = new ItemsBeckhoff();
                    if(channels.returnChannels(HardwareauflistungIoList[i, 0])>0)
                    {
        for(int r = 0; r< channels.returnChannels(HardwareauflistungIoList[i, 0]); r++)
                        {
 (Arbeitsblatt.Cells[i + zelle, 4] as Excel.Range).Value2 = "Channel "+ (1+r).ToString();
                            zelle = zelle + 1;
                        }
                    }
                }
                else
                {
                    (Arbeitsblatt.Cells[i+12, 2] as Excel.Range).Value2 = "";
                    (Arbeitsblatt.Cells[i + 12, 3] as Excel.Range).Value2 = "";
                }
            }
            Arbeitsmappe.Save();
            Arbeitsmappe.Close();

        }
```

Quelltext 10: Funktion für die Befüllung einer Excel-Datei, Quelle: Eigene Darstellung

7.5 Testphase

Um im Vorfeld die Funktionalität des Programmes zu testen und mögliche Fehler ausschließen zu können, wird in den nachfolgenden Punkten die Planung und der Entwurf eines zu testenden Systems durchgeführt. Anschließend werden für dieses System spezifische Testfälle entwickelt, welche anhand praktischer Versuche evaluiert werden.

7.5.1 Planung und Entwurf

Für die Planung solcher Testfälle ist es wichtig, diese so zu konstruieren, dass sie ein aussagekräftiges Ergebnis liefern. Dazu wird die Hardwarekonfiguration, welche zur Erstellung eines EPLAN Electric P8-Projektes herangezogen wird, definiert. Ebenfalls liefert diese Hardwarekonfiguration die Basis für eine Erstellung von globalen Variablen sowie für die Variablenzuordnung und die Erstellung einer Hardwarekonfiguration in TwinCAT 3.

7.5.1.1 Definition der Hardwarekonfiguration

Die Definition der Hardware wird so gestaltet, dass die Durchführung eines Datenexports mittels EPLAN alle notwendigen Daten enthält, um diese auch mittels Import in die KS-SPS-Project-Creator Anwendung verwenden zu können. Die Konfiguration wird in der nachfolgenden Tabelle 7-2 dargestellt.

Modultyp	Beckhoff-Bezeichung	Eigenschaften	BMK
CPU	CX5120	Embedded-PC	100KF1
Digitales Eingangsmodul	EL1008	8xDI	150KF1
Digitales Ausgangsmodul	EL2008	8xDO	200KF1

Tabelle 7-2: Definition der Hardwarekonfiguration, Quelle: Eigene Darstellung

7.5.1.2 Erstellung eines E-Plan-Projektes

Die nachfolgende Abbildung 7-43 zeigt die Erstellung der Hardwarekonfiguration, bestehend aus einer CPU und jeweils einen digitalen Eingangs- und Ausgangsmodul, in EPLAN.

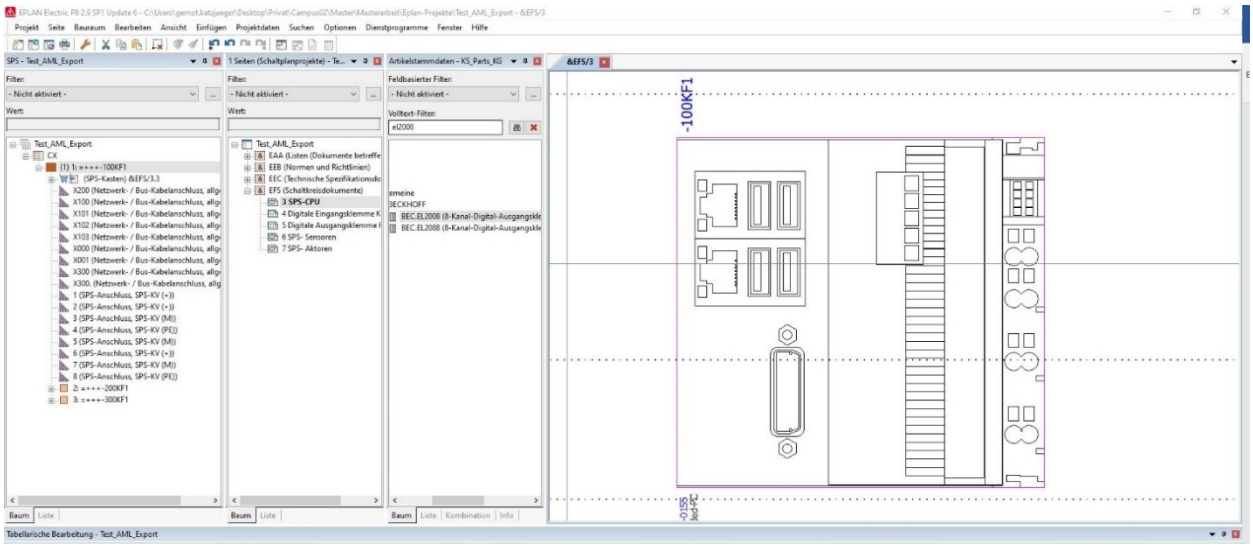

Abbildung 7-43: Erstellung der Hardwarekonfiguration in EPLAN, Quelle: Eigene Darstellung

7.5.2 Definition der Testfälle

Testfälle dienen der Validierung und der Fehlerermittlung des zu prüfenden Systems. In den nachfolgenden Punkten werden Testfälle definiert, welche bei erfolgreicher Durchführung eine qualitativ höherwertige Betriebsphase ermöglichen sollen.

Folgende Testfälle werden im Zuge der Testphase definiert und durchgeführt:

- Export einer AML-Datei aus EPLAN
- Laden und Erstellen einer Hardwarekonfiguration in TwinCAT 3
- Erstellung einer globalen Variablenliste und einer globalen Variable
- Erstellung eines Funktionsbausteins
- Erstellung einer lokalen Variable
- Befüllen einer Excel-Vorlagedatei

7.5.3 Durchführung von Tests

Die Durchführung von Testfällen ist ein relevanter Schritt in der Softwareentwicklung, da in dieser Phase ermittelte Mängel und Fehler einfach behoben werden können. In den nachfolgenden Punkten werden die oben erwähnten Testfälle durchgeführt. Eine Evaluierung der Testfälle wird abschließend in jedem Testfall separat durchgeführt.

7.5.3.1 Testfall: Export einer AML-Datei aus EPLAN

Dieser Testfall stellt sicher, dass sämtliche SPS-spezifische Einstellungen, welche in Punkt 5.1.2 näher erläutert wurden, korrekt definiert wurden. Die Durchführung sowie das Ergebnis des Testfalls sind in der Abbildung 7-44 ersichtlich.

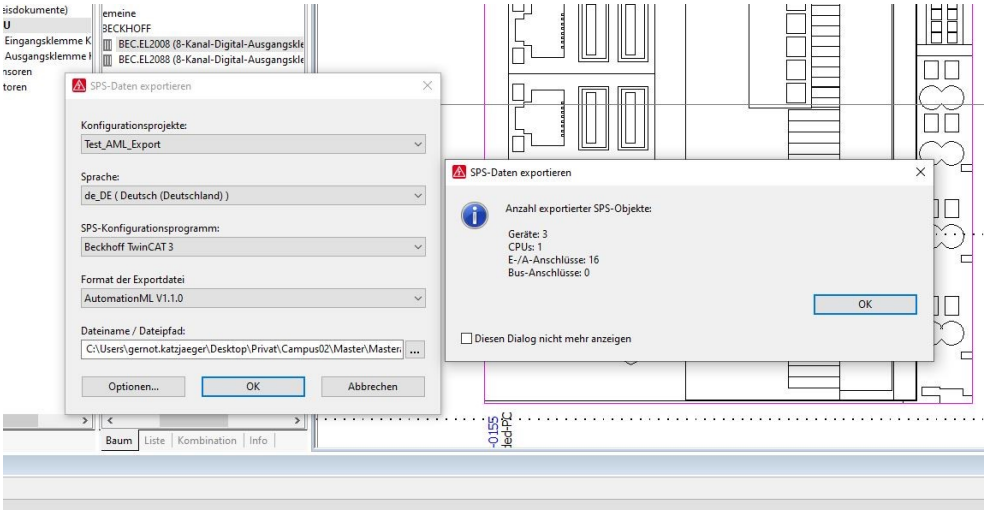

Abbildung 7-44: Testfall - AML-Export, Quelle: Eigene Darstellung

Einen Aufschluss über das Testergebnis gibt das in der Abbildung 7-44 dargestellte Meldungsfenster. Dieses zeigt, dass drei Geräte, eine CPU und 16 E/A-Anschlüsse exportiert wurden. Dies entspricht exakt der Anzahl der in Tabelle 7-2 definierten Hardwaremodule und deren Kanälen. Somit kann dieser Test als erfolgreich angesehen werden.

7.5.3.2 Testfall: Laden und Erstellen einer Hardwarekonfiguration in TwinCAT 3

Für die Erstellung der Hardwarekonfiguration für diesen Testfall, wurde die in Punkt 7.5.3.1 exportierte AML-Datei für den Import verwendet. Auf der linken Seite der Abbildung 7-45 ist die erstellte TwinCAT 3-Hardwarekonfiguration dargestellt. Mittig in dieser Darstellung befindet sich die KS-SPS-Project-Creator Anwendung, welche die importierte und geänderte Hardwarekonfiguration zeigt.

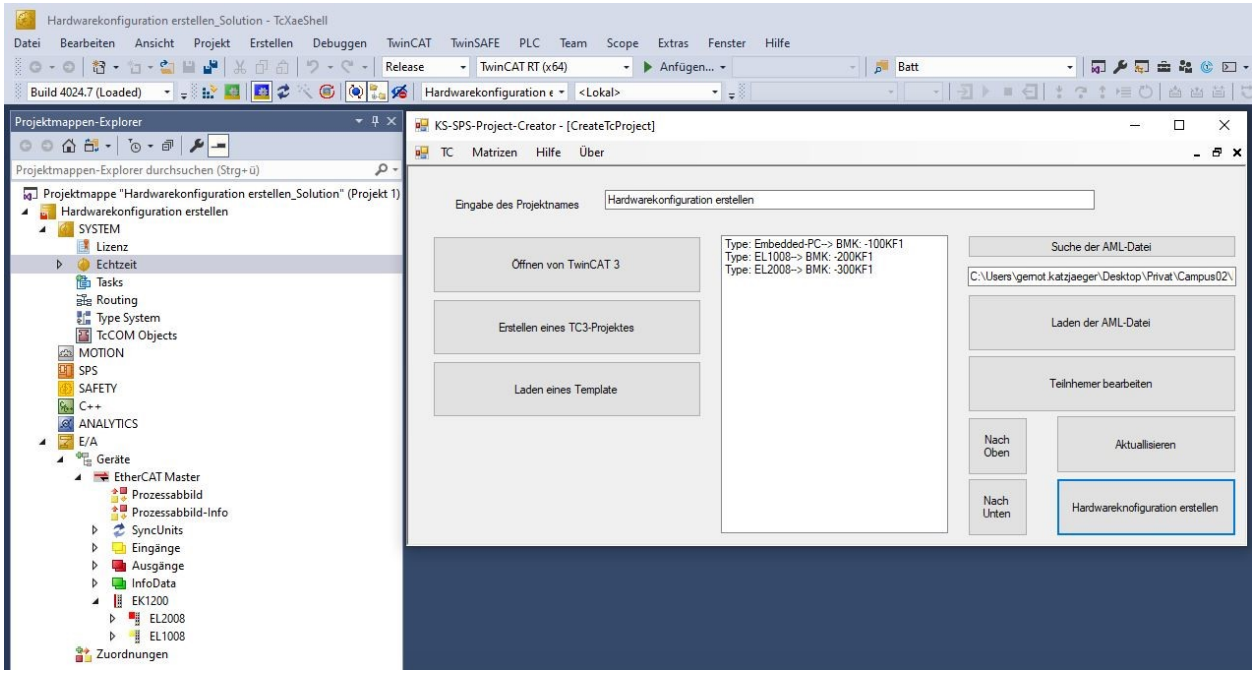

Abbildung 7-45: Testfall - Erstellung der Hardwarekonfiguration, Quelle: Eigene Darstellung

In diesem Testfall werden die mittels einer AML-Datei importierten Hardwaremodule in einem TwinCAT3-Projekt erstellt. Ersichtlich ist dies in der oben dargestellten Abbildung. Auf der linken Seite, im TwinCAT 3 Projektmappen-Explorer, wird ein EtherCAT-Master gelistet. Dieser verfügt über einen EK1200-Koppler, der den Embedded-PC darstellt, sowie über eine EL2008 und eine EL1008. Da die Hardwarekonfiguration der Auflistung der importieren AML-Datei entspricht, kann dieser Testfall als erfolgreich angesehen werden.

7.5.3.3 Testfall: Erstellung einer globalen Variablenliste und globalen Variablen

Um den Testfall für die automatisierte Erstellung einer globalen Variablenliste durchführen zu können, wird die bereits im Testfall 7.5.3.1 erstellte AML-Datei als Datenquelle verwendet. Die nachfolgende Tabelle 7-3 zeigt die im Vorfeld definierten Variablen, welche mittels EPLAN-Export in die AML-Datei übertragen wurden:

Variablenart	Bezeichnung	Datentyp
Input	Naeherungssensor_BT1	BOOL
Input	Naeherungssensor_BT1	BOOL
Output	Schuetz_STO_Kanal_1	BOOL
Output	Schuetz_STO_Kanal_2	BOOL

Tabelle 7-3: Definierte Variablen, Quelle: Eigene Darstellung

Mittels Importfunktion im KS-SPS-Project-Creators ist es möglich, sämtliche in EPLAN verknüpfte Variablen zu importieren und in eine globale Variablenliste zu übertragen. Die Durchführung dieses Testfalls ist in der nachfolgenden Abbildung 7-46 dargestellt.

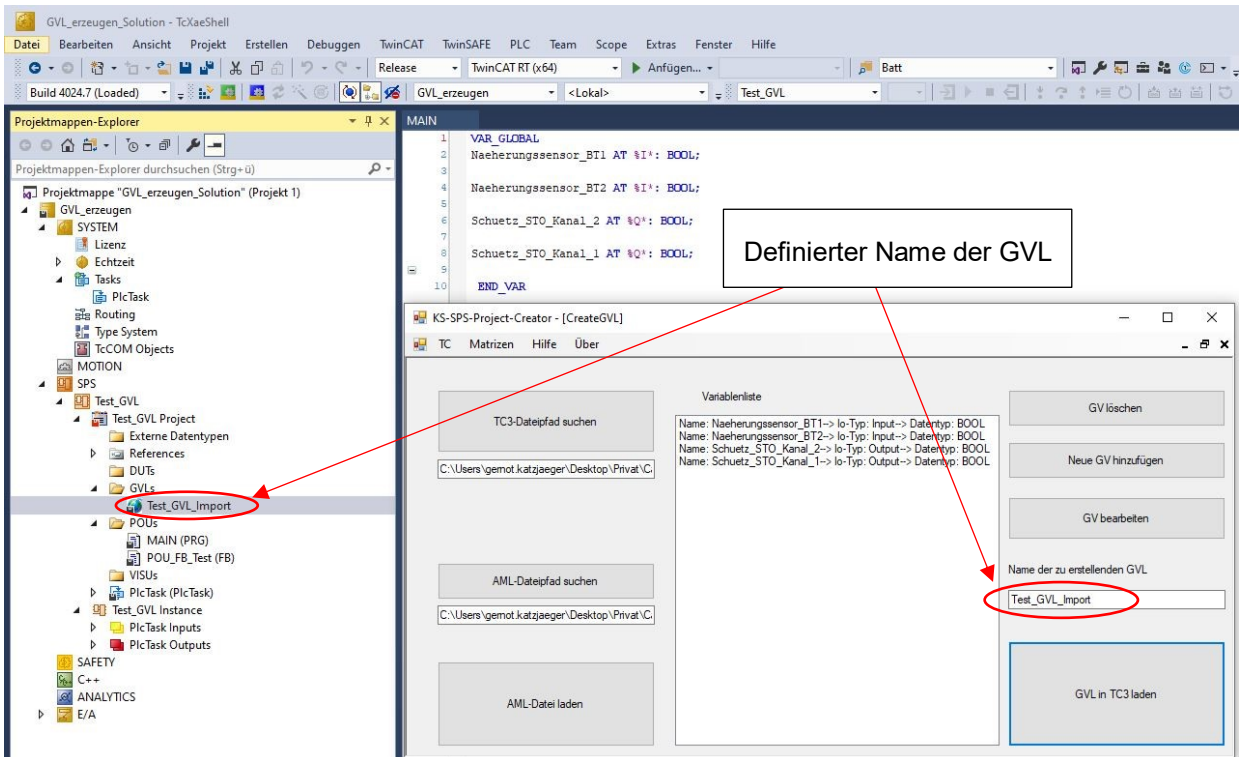

Abbildung 7-46: Testfall - Erstellung einer globalen Variablenliste, Quelle: Eigene Darstellung

Der Testfall ist erfolgreich durchgeführt, da sämtliche, in der Variablenliste vorkommenden Variablen, in das TwinCAT3-Projekt übertragen wurden. Ebenfalls wurde der definierte Name für die GVL vergeben. Dieser Test führte jedoch zur Erkenntnis, dass die Zuweisung von verschiedenen Konfigurationsprojekten in EPLAN-Projekt eine wesentliche Rolle für die Erstellung von verschiedenen globalen Variablenlisten dienen kann. Mittels Zuweisung von spezifischen Konfigurationsprojekten zu den gewünschten Modulen, ist es möglich, mehrere AML-Dateien eines EPLAN Projektes zu exportieren, welche die gewünschten Daten beinhalten. So kann man zum Beispiel, eine AML-Datei für alle digitalen Eingangsmodule erstellen.

7.5.3.4 Testfall: Erstellung eines Funktionsbausteines

Um den Testfall für die Erstellung einer POU durchzuführen, wurde das im Punkt 7.5.3.2 erstellte Projekt nochmals geladen. Der Speicherpfad der zu erstellenden POUs verweist standardmäßig auf den POUs-Ordner. Somit muss dieser nicht ausgewählt werden.

Für die Erstellung des POU, wurden die in der nachfolgenden Tabelle 7-4 beschriebenen Eigenschaften ausgewählt:

Eigenschaft	Definition
Name der POU	Test_POU
Art der POU	Funktionsbaustein
Programmiersprache	ST (Structured Text)

Tabelle 7-4: POU-Eigenschaften, Quelle: Eigene Darstellung

Die Abbildung 7-47 zeigt die Erstellung eines Funktionsbausteines mittels des KS-SPS-Project-Creators.

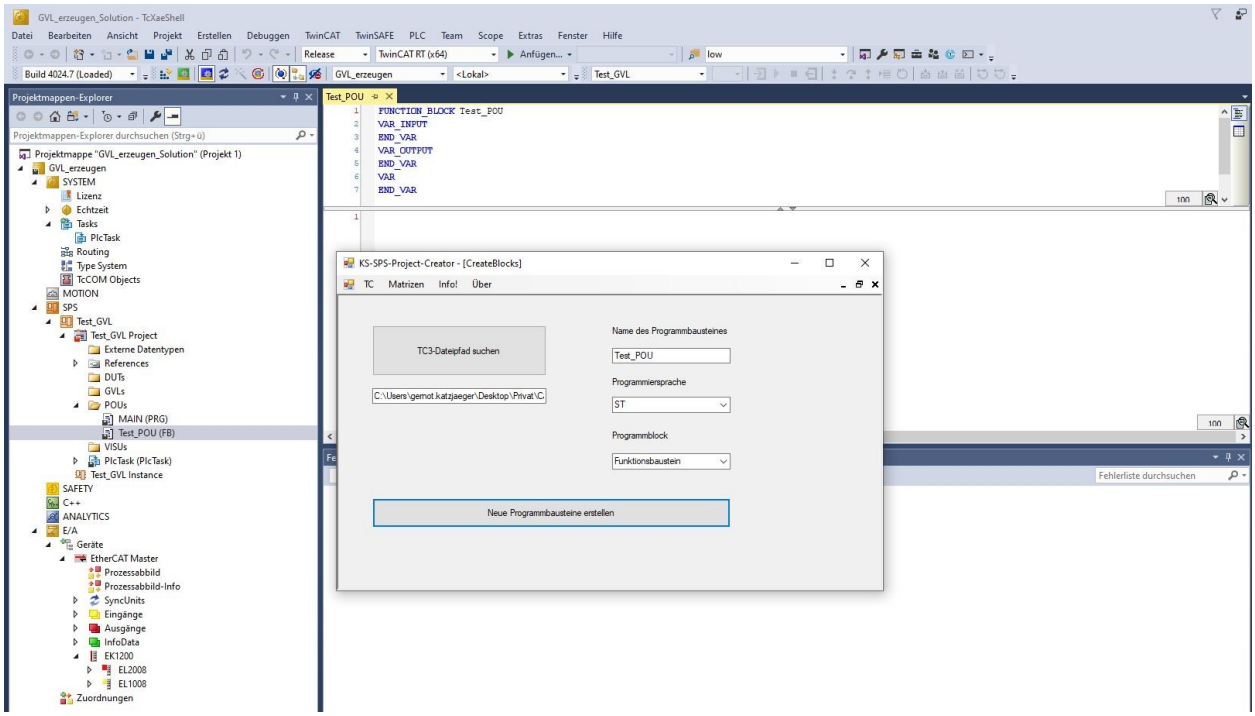

Abbildung 7-47: Testfall - Erstellung eines Funktionsbausteines, Quelle: Eigene Darstellung

Der Testfall kann als erfolgreich angesehen werden, da eine POU erstellt wurde, welche den definierten Eigenschaften entspricht.

7.5.3.5 Testfall: Erstellung einer lokalen Variable

Für die Erstellung von lokalen Variablen wurden das bereits in den vorigen Testfällen verwendete TwinCAT-Projekt nochmals erweitert und. angepasst. Für diesen Testfall wurde ein neuer Funktionsbaustein erstellt, in dessen Deklarationsbereich die Variablenliste erstellt wird.

Für die Durchführung dieses Testfalles wurden folgende Variablen, welche in der Tabelle 7-5 aufgelistet sind, definiert:

Name	Datentyp	Variablentyp
Test_BOOL	BOOL	Input
Test_INT	INT	Output

Tabelle 7-5: Definition der lokalen Variablen, Quelle: Eigene Darstellung

Die nachfolgende Abbildung 7-48 zeigt die Durchführung und das Ergebnis des Testfalls.

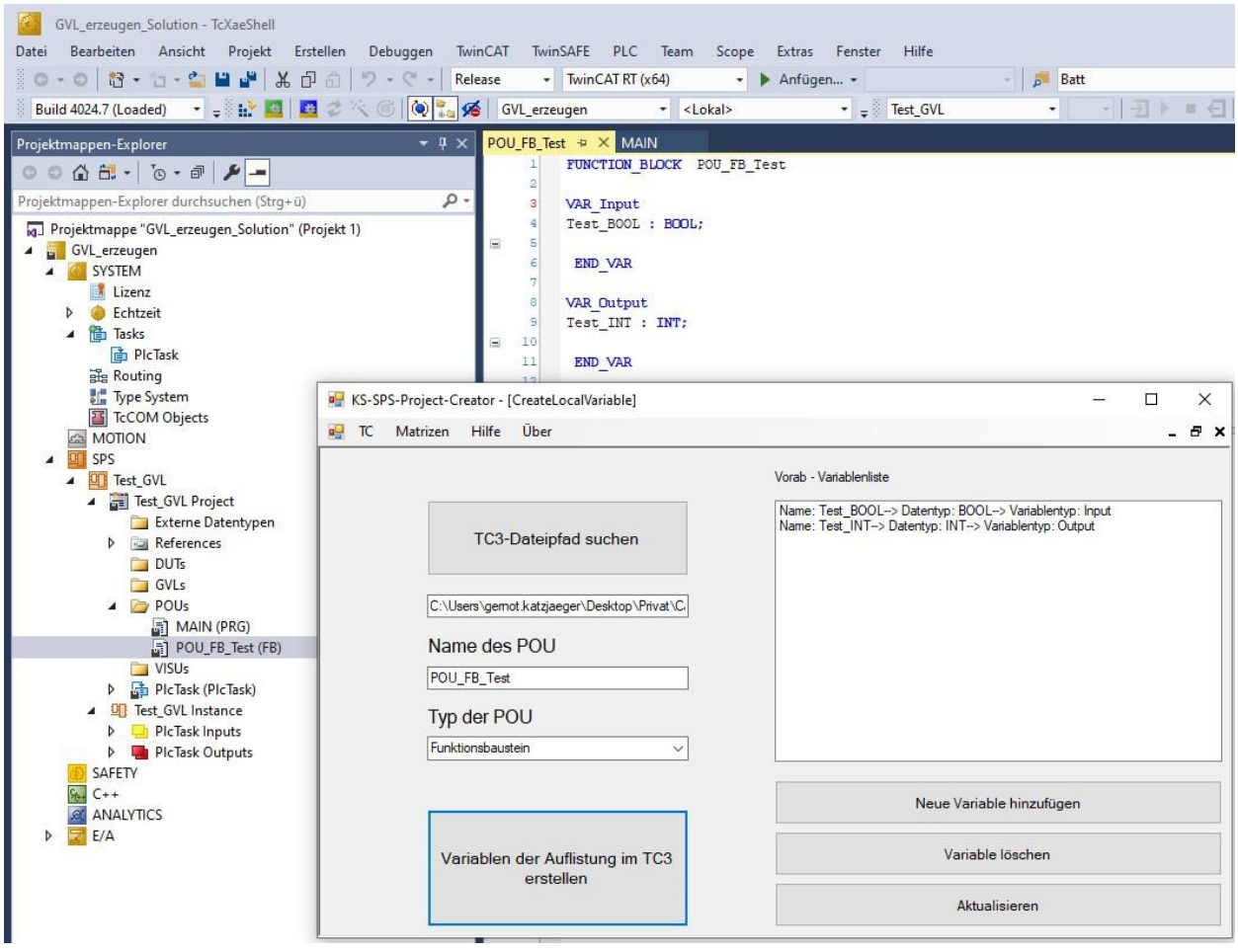

Abbildung 7-48: Testfall - Erstellung von lokalen Variablen, Quelle: Eigene Darstellung

Das positive Ergebnis des Testfalles ist durch die oben dargestellte Abbildung ersichtlich. Diese zeigt die Erstellung sowohl einer Input-Variable als auch die einer Output-Variable. Die Input-Variable wurde, wie auch in der Tabelle 7-5 definiert, als boolescher Datentyp deklariert. Der Output-Variablen wurde der Datentyp „Integer“ zugewiesen.

7.5.3.6 Testfall: Erstellung von automatisierten Quellcodes

Die Erstellung von automatiserten Quellcodes fordert, dass in Programmorganisationseinheiten Variablen deklariert sind. Zu diesem Zweck wurden mithilfe des KS-SPS-Project-Creators Variablen in einer POU erstellt und ein PLCopenXML-Export im TwinCAT 3 durchgeführt. Die nachfolgende Tabelle 7-6 zeigt eine Auflistung der erstellten Variablen.

Name	Datentyp	Variablentyp
di_Sensor_1	BOOL	Input
di_Sensor_2	BOOL	Input
do_Schuetz_1	BOOL	Output
lokaleVar_1	BOOL	Lokal
lokaleVar_2	BOOL	Lokal

Tabelle 7-6: Definition der Variablen für eine Quellcodeerstellung, Quelle: Eigene Darstellung

Anschließend wurde die erstellte XML-Datei im KS-SPS-Project-Creator eingelesen und konvertiert. Die Aufteilung der Variablen ist in der Abbildung 7-49 dargestellt.

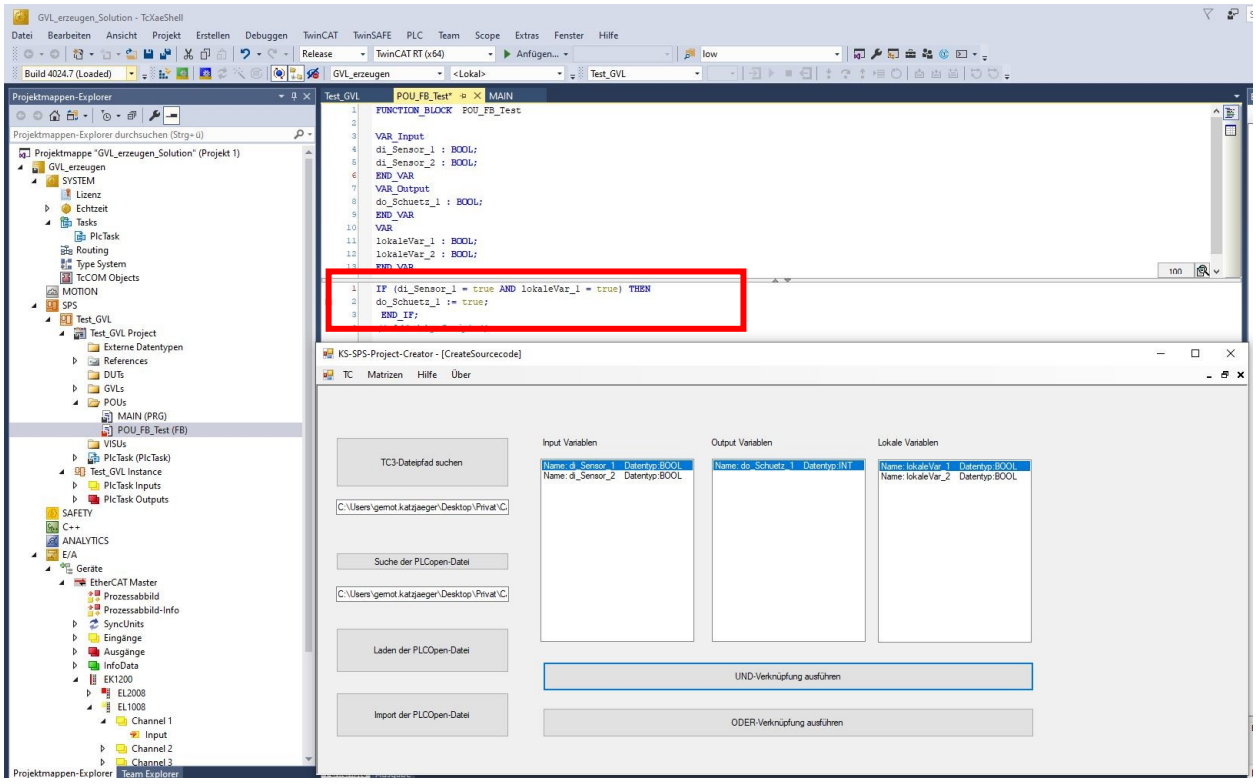

Abbildung 7-49: Testfall - Erstellung von lokalen Variablen, Quelle: Eigene Darstellung

Durch die Funktion „UND-Verknüpfung ausführen“ wurde eine IF-Anweisung mit den ausgewählten Variablen erstellt. Nachvollziehbar ist dies in der Abbildung 7-49 im rot markierten Bereich. Somit kann dieser Testfall als erfolgreich erachtet werden.

7.5.3.7 Testfall: Zuweisung von Variablen zu I/O-Modulen

Für die Verlinkung einer globalen Variable mit einem Kanal eines Hardwaremoduls wird eine globale Variablenliste mithilfe des KS-SPS-Project-Creators erstellt. Anschließend wird in TwinCAT 3 ein PLCopenXML-Export durchgeführt. Um nun eine Verlinkung von bestehenden globalen Variablen mit Kanälen von Hardwaremodulen durchzuführen wird diese XML-Datei im KS-SPS-Project-Creator geladen und konvertiert. In der Abbildung 7-50 wird die Auflistung der eingelesenen Variablen dargestellt.

Für diesen Testfall wird eine Input-Variable auf den Channel 1 einer digitalen Eingangskarte, EL1008, verlinkt. Dies ist ebenfalls in der nachfolgenden Abbildung ersichtlich, da die erste Variable angewählt und somit blau hinterlegt ist. Ebenfalls ist der Modulname, sowie der Typ und der Kanal ausgewählt.

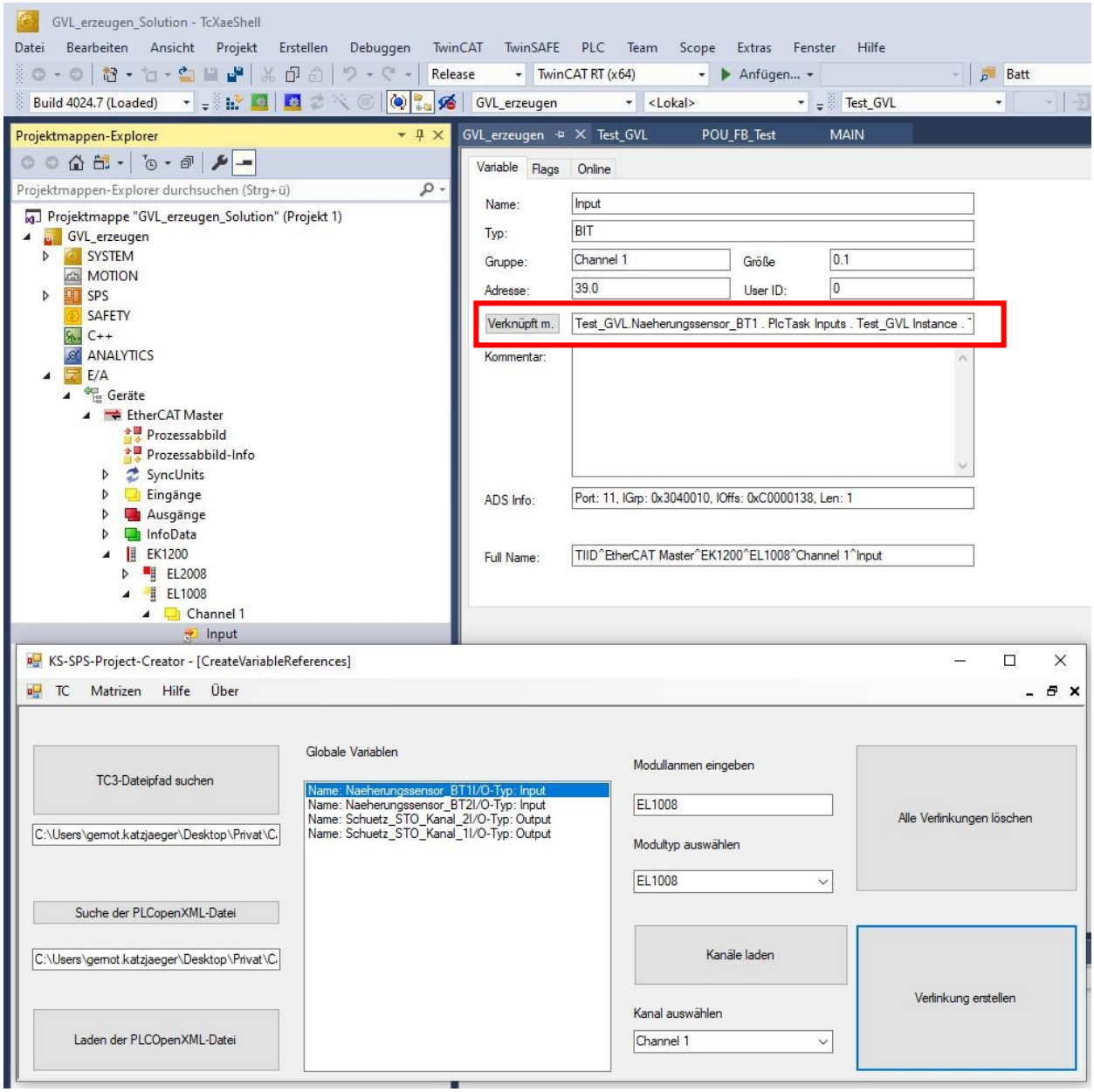

Abbildung 7-50: Testfall – Zuordnung von Variablen zu I/O-Modulen, Quelle: Eigene Darstellung

Der rot-markierte Bereich in der Abbildung 7-50 zeigt den Verknüpfungszustand des Hardwarekanals in der Entwicklungsumgebung TwinCAT 3. Der Testfall kann als erfolgreich angesehen werden, da die im KS-SPS-Project-Create ausgewählte Variable, mit dem ausgewählten Kanal, verlinkt wurde.

7.5.3.8 Testfall: Befüllen einer Excel-Vorlagedatei

Um eine Excel-Datei mit einer Hardwarekonfiguration zu befüllen, muss eine AML-Datei mit den benötigten Informationen über die gewünschte Hardwarekonfiguration geladen und konvertiert werden. Das Ergebnis der Konvertierung ist in der Auflistung der Hardwarekonfiguration in der Abbildung 7-51 dargestellt. Anschließend wird eine Ziel-Datei, welche ein Excel-Format aufweisen muss, ausgewählt. Durch die Funktion „Befüllen der ausgewählten Excel-Datei“ wird die Übertragung der Daten durchgeführt.

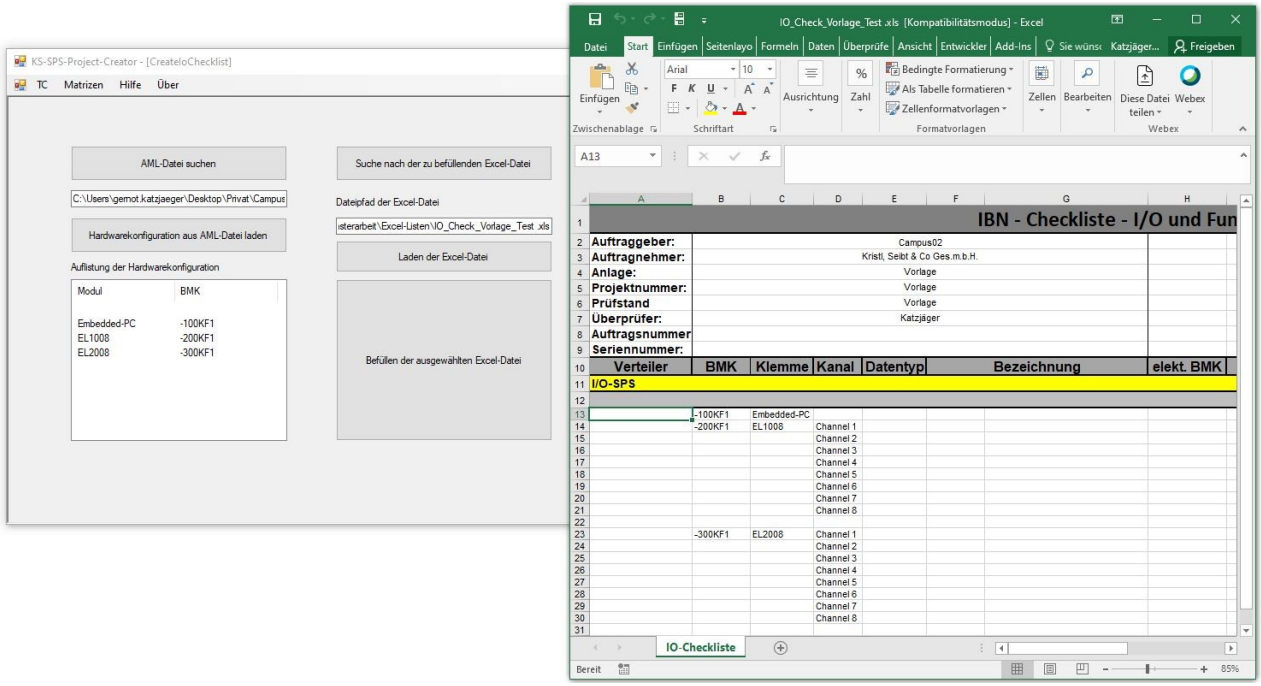

Abbildung 7-51: Testfall – Zuordnung von Variablen zu I/O-Modulen, Quelle: Eigene Darstellung

Die Abbildung 7-51 zeigt die erfolgreiche Befüllung der Excel-Vorlagedatei laut der Auflistung der Hardwarekonfiguration, welche durch die AML-Datei im KS-SPS-Project-Creators geladen und konvertieret wurde. Daher kann dieser Testfall als erfolgreich angesehen werden.

7.6 Betriebsphase

Mit der Betriebsphase schließt sich die Anwendung des Softwareentwicklungsmodells bzw. leitet die abschließenden Prozesse für die Übergabe und Weiterentwicklung der Applikation ein, da Fehler und Weiterentwicklungen einer Software nie ausgeschlossen werden können. Auf diese Punkte wird nachfolgend näher eingegangen.

7.6.1 Übergabe

Die Übergabe des KS-SPS-Project-Creatros, inklusive dessen Quellcode, erfolgt an die jeweiligen SPS Entwicklungsabteilungen, da so ein schneller und einfacher Einblick in den Ablauf der Funktionen geschaffen wird.

7.6.2 Installation und Schulung

Um eine Zugänglichkeit sämtlicher Softwaretools mit den gültigen Revisionsständen für alle Entwickler*innen zu schaffen, wird eine Ablage am Firmenserver realisiert.

Im Anschluss an diese Arbeit wird ein Schulungsdokument erstellt, welches alle Arbeitsschritte der jeweiligen Funktionen detailliert beschreibt.

7.6.3 Fehlerkorrektur und Optimierung

Da, wie schon erwähnt, Fehler nie vollständig ausgeschlossen werden können bzw. für spätere Revisionsstände bereits Optimierungen geplant sind, muss hier eine Dokumentation stattfinden. Da bei KS Engineers bereits das Management Tool Jira eingesetzt wird, soll diese Möglichkeit geschaffen werden.

7.6.4 Weiterentwicklung

Wie auch bei der Fehlerkorrektur und bei der Optimierung soll die Weiterentwicklung der KS-SPS-Project-Creator Anwendung auch über Jira gelöst werden. Da jedoch bereits während dieser Arbeit Potential für diverse Weiterentwicklungen erkannt wurde, werden diese in den nachfolgenden Punkten aufgeschlüsselt. Deren Umsetzung kann somit im Anschluss dieser Arbeit erfolgen.

Die nachfolgende Tabelle 7-7 stellt eine Auflistung der bereits definierten Weiterentwicklungen dar.

Weiterentwicklung	Art	Beschreibung
Erstellung der Hardwarekonfiguration	Erweiterung der Modulliste	Die Modulliste ist eine im KS-SPS-Project-Creator hinterlegte Auflistung, welche einige Beckhoff Module beinhaltet. Diese Auflistung muss entweder auf sämtliche von Beckhoff bereitgestellten Module erweitert werden, oder diese Daten über eine externe Datei oder Datenbank beziehen.

Erstellung der Hardwarekonfiguration	Auswahl des Revisionsstandes	Aktuell wurde in dieser Auflistung ein Default-Wert für den Revisionsstand eines Moduls hinterlegt. Die Anpassung an den exakten Revisionsstand kann mittels Online-Abgleich der tatsächlich verbauten Hardware vorgenommen werden. Um dies ebenfalls zu Automatisieren soll eine Eingabe des Revisionsstandes umgesetzt werden.
Erstellung der globalen Variablenlisten und Variablen	Eingabe des Speicherpfades der GVL	Aktuell wurde ein Pfad zum Speicherort der GVL`s im TwinCAT-Projekt in der .Net-Anwendung definiert. Die Ablage einer GVL wird in Projekten meist spezifisch gewählt. Um eine erhöhte Flexibilität zu erreichen, soll dieser Pfad ebenfalls im KS-SPC manuell definierbar sein.
Erstellung der globalen Variablenlisten und Variablen	Vorhandene GVL`s Einlesen und Bearbeiten	Wird aktuell eine GVL erstellt, so werden deren vorhandene Daten gelöscht und mit neuen Daten gefüllt. Um bestehende GVL`s jedoch anzupassen, muss eine Möglichkeit geschaffen werden, diese zu importieren und deren vorhandenen Informationen in einem Zwischenspeicher abzulegen. Neue Informationen und die des Zwischenspeichers müssten zusammengeführt und in die neu zu erstellende Variablenliste geladen werden.
Erstellung von POUs	Eingabe des Speicherpfades	Wie auch bei der Erstellung von globalen Variablenlisten und Variablen, soll auch hier die Möglichkeit geschaffen werden, den Speicherpfad durch eine Eingabe zu definieren.
Erstellung von POUs	Erstellung durch eine Vorlagenliste	Um die Effizienz zu steigern und Fehler zu verhindern, soll eine Erstellung der POUs durch den Import einer Vorlagedatei, welche aus einem Excel- oder XML-Dateiformat besteht, realisiert werden.
Erstellung von automatisierten Quellcodes	Weitere Funktionen	Aktuell ist es möglich, eine UND- oder ODER-Verknüpfung zu erstellen. Diese Funktionen werden um weitere, wie zum Beispiel das Erstellen von Switch-Anweisungen, ergänzt.

Befüllung von Excel-Listen	Anpassung der importierten Daten	Es soll die Möglichkeit geschaffen werden, um Anpassungen der Hardwaremodule inkl. deren Eigenschaften, mithilfe der Anwendung noch vor einer Datenübertragung in das Excel durchführen zu können.

Tabelle 7-7: Weiterentwicklungen, Quelle: Eigene Darstellung

8 ERGEBNISSE UND AUSBLICK

In den nachfolgenden Punkten wird auf die Ergebnisse, welche im Zuge dieser Arbeit ermittelt wurden, näher eingegangen. Ebenfalls wird aufgeschlüsselt, welche weitere Vorgehensweisen, für eine effiziente Weiterentwicklung der Applikation aber auch für die Zusammenarbeit mit den Konstruktionsabteilungen umgesetzt werden sollte.

8.1 EPLAN Electric P8

Auch wenn EPLAN Electric P8 im Zuge dieser Arbeit nur für den Datenexport genutzt wurde, ergaben sich wesentliche Erkenntnisse hinsichtlich der Erstellung einer AML- Exportdatei. Ebenfalls ist es relevant, für eine Verbesserung der Projekterstellung weitere Maßnahmen und Anpassungen (gemäß Kapitel 8.1.1) in EPLAN durchzuführen.

8.1.1 Ergebnisse

Der Datenexport einer AML-Datei erwies sich als zentrale Schnittstelle zwischen dem KS-SPC und einem EPLAN-Projekt. Somit ermöglicht er eine Einsparung der Entwicklungszeit und eine Minimierung der manuellen Übertragungsfehler. Essenziell für die Erstellung von AML-Exportdateien durch EPLAN ist es, die Meta-Daten jedes Moduls nach einem vordefinierten Schema zu befüllen. So ist die Erstellung und Zuweisung von Konfigurationsprojekten relevant, wenn die Auftrennung von Modulen, zum Beispiel Eingangs- und Ausgangsmodule, durch Erstellung mehrerer AML-Dateien erfolgen soll.

8.1.2 Ausblick

Es wurde festgestellt, dass eine effiziente Erstellung eines SPS-Projektes nur erreicht wird, wenn die aus dem EPLAN exportierte AML-Datei über ausreichend und korrekt definierte Meta-Daten verfügt. Diesbezüglich werden weitere firmeninterne Abstimmungen zwischen der SPS-Entwicklungsabteilung und der EPLAN-Konstruktionsabteilung stattfinden.

8.2 Programmerstellung in Visual Studio

Die Entwicklungsumgebung Visual Studio diente in dieser Arbeit als zentrale Anwendung zur Erstellung des KS-SPS-Project-Creators. Die Implementierung der notwendigen Funktionen aber auch der grafischen Benutzeroberfläche wurde mittels dieses Tools durchgeführt.

8.2.1 Ergebnisse

Sämtliche in dieser Arbeit erstellten UML-Diagramme, wurden für die Erstellung der Anwendung in das Visual Studio-Projekt übernommen. Dies bezieht sich sowohl auf die Erstellung der jeweiligen Klassen, Methoden und Funktionen als auch auf die grafische Benutzeroberfläche. Ebenfalls wurden die Funktionen, welche auf die Entwicklungsumgebung TwinCAT3 zugreifen, in dieses Visual-Studio Projekt implementiert.

8.2.2 Ausblick

Weitere notwendige oder nützliche Funktionen wurde bereits in Punkt 7.6.4 ermittelt bzw. werden durch die Betriebsphase noch ersichtlich. Für deren Erstellung müssen die bestehenden UML-Diagramme angepasst bzw. neue Konzepte entwickelt werden.

8.3 TwinCAT3

TwinCAT3 wird in dieser Arbeit als Entwicklungsumgebung für die Entwicklung von SPS-Softwarelösungen eingesetzt. Diese Entwicklungsumgebung für speicherprogrammierbare Steuerungen bietet mit seiner Schnittstelle namens Automation Interface eine sehr effiziente Methode, um in diese Applikation mittels eines externen Programmes eingreifen zu können.

8.3.1 Ergebnisse

Es wurden sämtliche Funktionen, welche in den Anforderungen definiert wurden, mittels der Verwendung des Automation Interface implementiert. An Stellen, an welchen ein manueller Zugriff auf das TwinCAT-Projekt notwendig ist, konnte dies ebenfalls umgesetzt werden.

8.3.2 Ausblick

Wie schon in Punkt 7.6.4 erwähnt, werden weitere Funktionen folgen, somit ist auch die Einbindung weiterer Automation Interface Funktionen notwendig.

8.4 Automatisierte Quellcodeerstellung

Die automatisierte Quellcodeerstellung wurde in dieser Arbeit nur dahingehend miteinbezogen, um die Erstellung von logischen UND- und ODER-Verknüpfungen zu gewährleisten. Die Erstellung von automatisierten Quellcodes kann jedoch weitaus komplexere Formen annehmen.

8.4.1 Ergebnisse

Die Erstellung von logischen Verknüpfungen zwischen Eingangs- oder Lokalen-Variablen sowie Ausgangsvariablen wurde umgesetzt. Wird eine solche Verknüpfung erstellt, wird diese mittels der Programmiersprache „Strukturierter Text“ in den Implementierungsbereich der jeweiligen POU geschrieben. Diese Umsetzung bildet die Basis für die Umsetzung von komplexeren logischen Systemen.

8.4.2 Ausblick

Um komplexere Systeme anhand einer automatisierten Quellcodeerstellung darstellen zu können, müssen weitere Modelle für die spezifischen Anwendungsfälle entwickelt werden. Das Grundkonzept wurde in dieser Arbeit erstellt und bietet die Basis für sämtliche weitere Anwendungen.

8.5 Durchführung der Testfälle

Ein wesentlicher Bestandteil des Softwareentwicklungsprozesses war die abschließende Durchführung von vordefinierten Testfällen. Diese wurden für die Evaluierung des beschriebenen Modelles und dessen Umsetzung in der programmierten Anwendung durchlaufen.

8.5.1 Ergebnisse

Sämtliche durchgeführten Testfälle konnten mit einem positiven Testergebnis abgeschlossen werden. Ebenfalls wurden anhand der Testergebnisse erste Weiterentwicklungen erkannt und in Punkt 7.6.4 definiert.

8.5.2 Ausblick

Wie schon erwähnt, werden Erweiterungen folgen, die wiederum die Definition weiterer Testfälle erfordert. Da es durch die definierten Testfälle zu aussagekräftigen Ergebnissen kam, sollen auch zukünftige Testfälle nach dem in dieser Arbeit angewandten Prinzip gestaltet und durchgeführt werden.

8.6 Erweiterung auf TIA-Portal

Total Integrated Automation Portal (TIA-Portal) ist die Bezeichnung für die Entwicklungsumgebung für speicherprogrammierbare Steuerungen des Herstellers Siemens. Da KS Engineers sowohl Beckhoff- als auch Siemens- Produkte im Einsatz hat, ist eine Erweiterung der Anwendung auf Siemens notwendig.

Notwendige Anpassungen

Ein Vorteil von objektorientierter Programmierung ist jener, dass mit deren Hilfe der Aufbau der internen Struktur nur durch geringe Anpassungen und Programmänderungen eine Erweiterung auf Siemens möglich macht. Dies bedeutet, dass bereits bestehende Klassen, welche für die AML-Dateikonvertierung oder für die Erstellung von Programmorganisationsbausteinen notwendig sind, auf die notwendigen Bedienungen und Anforderungen von Siemens angepasst oder erweitert werden können.

LITERATURVERZEICHNIS

Gedruckte Werke

Brandt-Pook, H., & Kollmeier, R. (2015). *Softwareentwicklung kompakt und verständlich* (Bd. 2). Bielefeld, Deutschland: Springer Fachmedien Wiesbaden GmbH.

Drath, D.-I. R. (2010). *Datenaustausch in der Anlagenplanung mit AutomationML.* Deutschland: Springer.

Goll, J., & Hommel, D. (2015). *Mit Scrum zum gewünschten System.* Esslingen, Deutschland: Springer Fachmedien Wiesbaden GmbH.

Hummel, P. D. (2011). *Aufwandsschätzungen in der Software- und Systementwicklung kompakt.* Heidelberg: Spektrum Akademischer Verlag.

Jan van Randen, H., Bercker, C., & Fieml, J. (2016). *Einführung in UML - Analyse und Entwurf von Software.* Essen, Deutschland: Springer Vieweg.

John, K., & Tiegelkamp, M. (2008). *SPS-Programmierung mit IEC 61131-3* (Bd. 4). Heidelberg, Deutschland: Springer-Verlag Berlin Heidelberg.

Kecher, C., Salvanos, A., & Hoffmann-Elbern, R. (2018). *UML 2.5 Das umfassende Handbuch* (Bd. 6). Bonn: Rheinwerk Verlag GmbH.

Kleuker, S. (2018). *Grundkurs Software-Engineering mit UML* (Bd. 4). Osnabrück, Deutschland: Springer Fachmedien Wiesbaden GmbH.

KS Engineers. (15. November 2021). Interne Programmierrichtlinie . Graz, Steiermark, Österreich.

Lahres, B., Rayman, G., & Strich, S. (2021). *Objektorientierte Programmierung - Das umfassende Handbuch.* Bonn, Deutschland: Rheinwerk Verlag GmbH.

Rumpe, B. (2012). *Agile Modellierung mit UML* (Bd. 2). Aachen, Deutschland: Springer-Verlag Berlin Heidelberg.

Schäfer, W. (2010). *Softwareentwicklung - Einstieg für Anspruchsvolle.* Deutschland: Pearson Studium, ein Imprint der Pearson Education Deutschland GmbH.

Unhelkar, B. (2018). *Software Engineering with UML.* USA, Florida : Auerbach Publications, CRC Press, Taylor & Francis Group .

Online Quellen

Beckhoff Automation. (9. Dezember 2020). *Automation Interface.* Abgerufen am 15. Juni 2021 von Beckhoff Automation: https://download.beckhoff.com/download/document/automation/twincat3/Automation_Interface_DE.pdf

Beckhoff Automation. o.J. *Globale Variablen*. Abgerufen am 15. November 2021 von Beckhoff Information System: https://infosys.beckhoff.com/index.php?content=../content/1031/tcplccontrol/html/tcplcctrl_resglobvar.htm

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

AI	Automation Interface
GUI	Graphical User Interface
VS	Visual Studio
TC3	TwinCAT 3
API	Application Programming Interface
UML	Unified Modeling Language
POU	Programming Organization Unit
GVL	Globale Variablen Listen
ST	Strukturierter Text
KS-SPC	KS-SPS-Project-Creator
DTE	Development Tools Environment
ECAD	Electronic Computer Aided Design
BMK	Betriebsmittelkennzeichen
CPU	Central Processing Unit