

# **BACHELORARBEIT**

im Studiengang Mechatronik / Robotik

## **Erstellung eines Codegenerators für standardisierte Transportsysteme**

Ausgeführt von: Florian Hackl  
Personenkennzeichen: 1010330054

BegutachterIn: DI(FH) Günther Poszvek

Wien, 26.1.2013

## Eidesstattliche Erklärung

„Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Ich versichere, dass die abgegebene Version jener im Uploadtool entspricht.“

---

Ort, Datum

---

Unterschrift

## Kurzfassung

Um standardisierte Umlaufanlagen in Fertigteilbetonwerken zu konfigurieren sind hunderte von SPS Quellcodezeilen notwendig.

Das manuelle erstellen dieser ist Zeitaufwendig und oft fehlerbehaftet. Vorhandene Lösungen mit Tabellenkalkulationsprogrammen sind schlecht wartbar, und äußerst unflexibel.

Der Lösungsansatz ist eine Umsetzung mit einer objektorientierten Programmiersprache, um einen hohen Modularitätsgrad zu erreichen. Durch Automatisierung der Codegeneration kann die Effizienz gesteigert werden. Durch Analyse des Ausgangscodes, und sinnvoller Kapselung kann eine Software mit benutzerfreundlichem Interface erstellt werden.

Das erstellte Programm automatisiert diesen Vorgang so weit, dass die Vorbereitungszeiten für die Umlaufsteuerungen drastisch reduziert, und die Fehleranfälligkeit auffallend dezimiert werden konnte. Der Arbeitsaufwand wurde auf ca. 30% des Ausgangssumms gesenkt.

Schlüsselwörter: Codegeneration,

## Abstract

To configure standardized transport systems in concrete plants, hundreds of SPS code lines are a necessity. Manually creating these is a time-expensive and error prone process. Existing solutions, using spreadsheet technology are hard to maintain, and not very flexible. The approach taken in this publication is the use of object oriented programming techniques to produce a high level of modularity.

By automating the process of code generation, the efficiency can be drastically improved. By analyzing existing SPS source code, and employing encapsulation methodology, it is possible to create software which is user friendly and efficient.

The developed application automates this process in such a fashion, that the workload could be reduced to approximately 30%.

**Keywords:** code generation, model driven software, transport systems

# Inhaltsverzeichnis

1	Einleitung .....	7
1.1	Motivation .....	7
1.2	Aufgabenstellung .....	7
1.3	Vorhandene Lösung .....	8
2	Generative Programmierung .....	8
2.1	Definition .....	8
2.2	Grundlagen .....	9
3	Objektorientierung .....	9
3.1	Konzepte .....	9
3.1.1	Eigenschaften .....	9
3.1.2	Funktionen .....	10
3.1.3	Instanziierung .....	10
3.1.4	Polymorphie .....	10
3.2	Datenmodelle .....	10
3.3	Abstraktionsmöglichkeiten .....	10
4	Standardisierte Transportsysteme .....	11
4.1	Übersicht .....	11
4.2	Stationen .....	12
4.3	Zwischenstationen .....	12
4.4	Quertransportwagen .....	12
4.5	Bedienpulte .....	13
5	Entwicklung des Benutzerinterfaces .....	14
5.1	Lösungsansatz .....	14
5.2	Benutzerfreundlichkeit und Workflow .....	15
5.2.1	Eingabe der Daten .....	15
5.2.2	Eingangszuweisungen .....	17
5.2.3	Ausgabe .....	18
5.3	Angewandte Software und Technologien .....	19
5.3.1	C# / .net .....	19

5.3.2	Common Interface Language .....	20
5.3.3	Visual Studio 2010.....	20
5.3.4	XML .....	21
5.3.5	Extention Methods .....	21
5.4	Fehlertextgeneration .....	21
5.4.1	Übersetzermodul .....	22
5.4.2	SQL Export .....	22
5.4.3	atvise SCADA .....	23
5.5	Routenplanung .....	23
5.6	Überblick über das Daten / Objektmodell .....	24
6	Codegenerator für Persistenten Zielcode .....	26
6.1	Separation von Generator und Interface .....	26
6.2	Wartungs- und Erweiterungsmöglichkeiten .....	26
6.3	STEP7 Syntax .....	26
7	Diskussion .....	27
7.1	Zusammenfassung .....	27
7.2	Vor- und Nachteile .....	28
7.3	Ausblick .....	28
8	Literaturverzeichnis .....	30
9	Abbildungsverzeichnis .....	31
	Abkürzungsverzeichnis .....	32

# 1 Einleitung

Seit es programmierbare Maschinen gibt, wird danach gestrebt, die Softwareentwicklung dafür möglichst unkompliziert zu gestalten. Daraus entwickelten sich eine Vielzahl an Programmiersprachen und Entwicklungsumgebungen, um dem Softwaredesigner repetitive Arbeiten abzunehmen, sodass sich auf wesentliche Merkmale konzentriert werden kann.

Durch Analyse von bereits vorhandenen Codesegmenten kann ein Bedienerfreundliches Programm erstellt werden, welches die Codegeneration zu einem großen Teil automatisiert.

## 1.1 Motivation

Als Motivation zur Erstellung eines Codegenerators liegt der Gedanke zu Grunde, dass sich in der Softwareerstellung für Transportsysteme in der Betonindustrie, besonders in den Initialisierungsabschnitten der Software Prozeduren und Codeblöcke sehr ähnlich und selbstwiderholend sind. Mitarbeiter waren dadurch verleitet, diese Blöcke aus alten Projekten, bzw. Abschnitte aus demselben Projekt mittels Copy & Paste zu duplizieren und anzupassen. Dadurch entstandene Kopierfehler sind schwer zu finden und auszubessern. Die bisherige Lösung wurde mittels Visual Basic Makros in einer Excel Tabelle gelöst. Diese Lösung ist jedoch nicht besonders Stabil, und kaum wartbar, da diese Methodik in den Fähigkeiten wesentlich beschränkter ist, als ein dediziertes Programm.

## 1.2 Aufgabenstellung

Die Firma SAA Software Engineering beschäftigt sich mit der Entwicklung von Software im Bereich von Steuerungstechnik und Leittechnik. Hauptkunde ist die Betonfertigteilindustrie sowie Firmen im Bereich von Wand und Deckenherstellung aus Holz und Ziegel.

Da Betonwerk-Umlaufsysteme vom Ablauf in vielen Fällen große Ähnlichkeiten aufweisen, jedoch in der elektrischen Implementation nicht kohärent sind, ergibt sich für Steuerungsrechner und die dazugehörigen SPS-Systeme in jedem Einzelfall eine unterschiedliche Konfiguration. Manuelle Programmierung dieser Systeme ist relativ Fehleranfällig und Zeitaufwändig.

Aus diesem Grund soll ein Programm in Microsoft C# erstellt werden, welches ein einfaches Userinterface zur Verfügung stellt, um diese Konfigurationen schnell und unkompliziert zu erstellen. Durch einen Modulare und objektorientierten Aufbau soll eine einfache Erweiterung und Wartbarkeit gegeben werden, die mit der bisherigen Umsetzung nicht gegeben ist.

Die bisherige Lösung für dieses Problem waren mehrere Tabellen in Microsoft Excel, mit Makros in Microsoft Visual Basic.

[illegible]

Abbildung 1: Screenshot (SAA Engineering, 2012)

Die Bestehende Lösung erforderte sehr viele manuelle Eingaben, und bot keine automatische Fehlerkorrektur. Da die Makros der Grundfunktionen über mehrere Jahre hinweg ständig erweitert wurden, und ohne modulares Konzept entstanden, war eine schlechte, bis gar keine Wartbarkeit gegeben.

Des Weiteren ist bei dieser Umsetzung zwingend eine Installation von Microsoft Excel Voraussetzung, welches die Portabilität der Anwendung weiter einschränkt.

## 2 Generative Programmierung

*„Meta-programming is the act of writing programs that generate other programs“*

(Karsai, 2004)

Generative Programmierung entspricht einer Quellcodegenerierung ohne direkte Quellcodeingabe.

## 2.1 Definition

Generative Programmierung ermöglicht es, automatisiert Softwarequellcode aus Logischen Konzepten zu generieren, um schnell und einfach spezifizierte Software für ähnliche Produktgruppen zu erstellen.

## 2.2 Grundlagen

Als grundlegendes Schema ist die Codegeneration als unabhängige Schicht von der Dateneingabe zu sehen. Anstatt Anweisungen für einen Compiler zu schreiben, hat der Bediener primär die Aufgabe, Datenmodelle zu beschreiben. Die Umsetzung zu maschinenlesbaren Code erfolgt automatisiert. Dadurch ist es möglich, einen Großteil von menschlichen Fehlern wie Tippfehlern, Kopierfehlern und „vergessen etwas zu implementieren“ auszuschließen. Als weiterer Vorteil zeigt sich, dass der Bediener nicht zwingend Kenntnisse in der Programmiersprache haben muss, um Programmteile zu erstellen.

## 3 Objektorientierung

Objektorientierung im Sinne von Programmiersprachen bezeichnet ein Programmierparadigma, bei dem Daten und Funktionen möglichst eng in einem Objekt gekapselt werden.

Prinzipiell hat jedes dieser Objekte Attribute und Methoden, und ist in der Lage mit anderen Objekten zu kommunizieren.

Im Vergleich zu nicht-objektorientierten Programmiersprachen, wo Daten und Funktionen strikt getrennt werden, sind in diesem Schema die zugehörigen Methoden und Daten pro Objekt zusammengefasst. (Sharp, 2010)

### 3.1 Konzepte

Im Folgenden Kapitel wird auf grundlegende Eigenschaften einer objektorientierten Sprache eingegangen.

#### 3.1.1 Eigenschaften

Als Eigenschaften kann man den „Datenteil“ eines Objektes verstehen. Hierbei ist anzumerken, dass Objekte sowohl öffentliche, als auch private Eigenschaften haben können. Öffentliche Eigenschaften sind von anderen Objekten einsehbar, und unter Umständen auch veränderbar, Private hingegen nur für das Objekt selbst.

Ein abstrahiertes „Auto“-Objekt könnte z.B. Die öffentliche Eigenschaft „Lackfarbe“ besitzen. Eine interne (private) Eigenschaft in diesem Beispiel wäre die Tankfüllung, die von außen nicht erkennbar ist.

### 3.1.2 Funktionen

Wie aus fast jeder Programmiersprache bekannte Funktionen sind auch in das Objekt gekapselt, und können aufgerufen werden.

### 3.1.3 Instanziierung

Einmal als Klasse beschrieben, können mehrere Instanzen dieser gleichzeitig existieren. Dieser Vorgang wird Instanziierung genannt. Zu dem vorhergehenden „Auto“ Beispiel, wären unterschiedliche Instanzen der Klasse „Auto“ möglich, eine mit der Eigenschaft „Lackfarbe Rot“ und einmal mit der Eigenschaft „Lackfarbe Grün“.

### 3.1.4 Polymorphie

*„The basic idea behind inheritance is that new data types can be defined as extensions of previously defined types, rather than having to be defined from scratch“*

(Meyer, 1986)

Eine weitere Eigenheit der objektorientierten Programmierung ist die sogenannte „Vererbung“, mittels der es möglich ist, von vorhandenen Klassen, neue Klassen zu erstellen, die einige, oder alle Eigenschaften der Stammklasse erben.

Dies ermöglicht eine hierarchische Struktur in der Objekterstellung.

## 3.2 Datenmodelle

Durch die objektorientierte Betrachtungsweise, ist es leicht möglich, reale Objekte im Softwareraum nachzubilden.

Da reale Gegenstände auch über Eigenschaften, bzw. Funktionen verfügen, ist eine Abstrahierung möglich.

## 3.3 Abstraktionsmöglichkeiten

Im speziellen Fall des Umlaufsystems wurden hier Grundklassen erstellt, welche Eigenschaften beinhalten, die auf alle Teile eines Umlaufsystems angewendet werden können, wie Kurzbezeichnung, Name und Index.

Abgeleitete Klassen wie die Pultklasse, erweitern diesen Umfang um Felder wie Ein- und Ausgänge, Stationszugehörigkeit und Funktionsumfang (Taster, Lampen, Schalter, etc.)

## 4 Standardisierte Transportsysteme

In der Industrie werden diverse Transportsysteme verwendet, die sich in vielen Grundprinzipien und Bausteinen ähneln. Die hier beispielhaft beschriebene Anlage ist in Größe und Komplexität durchschnittlich, und daher als Anschauungsobjekt geeignet.

### 4.1 Übersicht

Die Betonfertigteile (unter anderem Betonwände, Doppelwände und Decken) werden auf einem Palettensystem in einer Umlaufanlage gefertigt und transportiert. Jede Palettenstation ist dabei mit einigen Sensoren ausgestattet, um den Ablauf überprüfbar zu machen. Des Weiteren sind in den Produktionshallen diverse Bedienpulte vorhanden, mittels denen der Ablauf und die Maschinen auch manuell steuerbar sind.

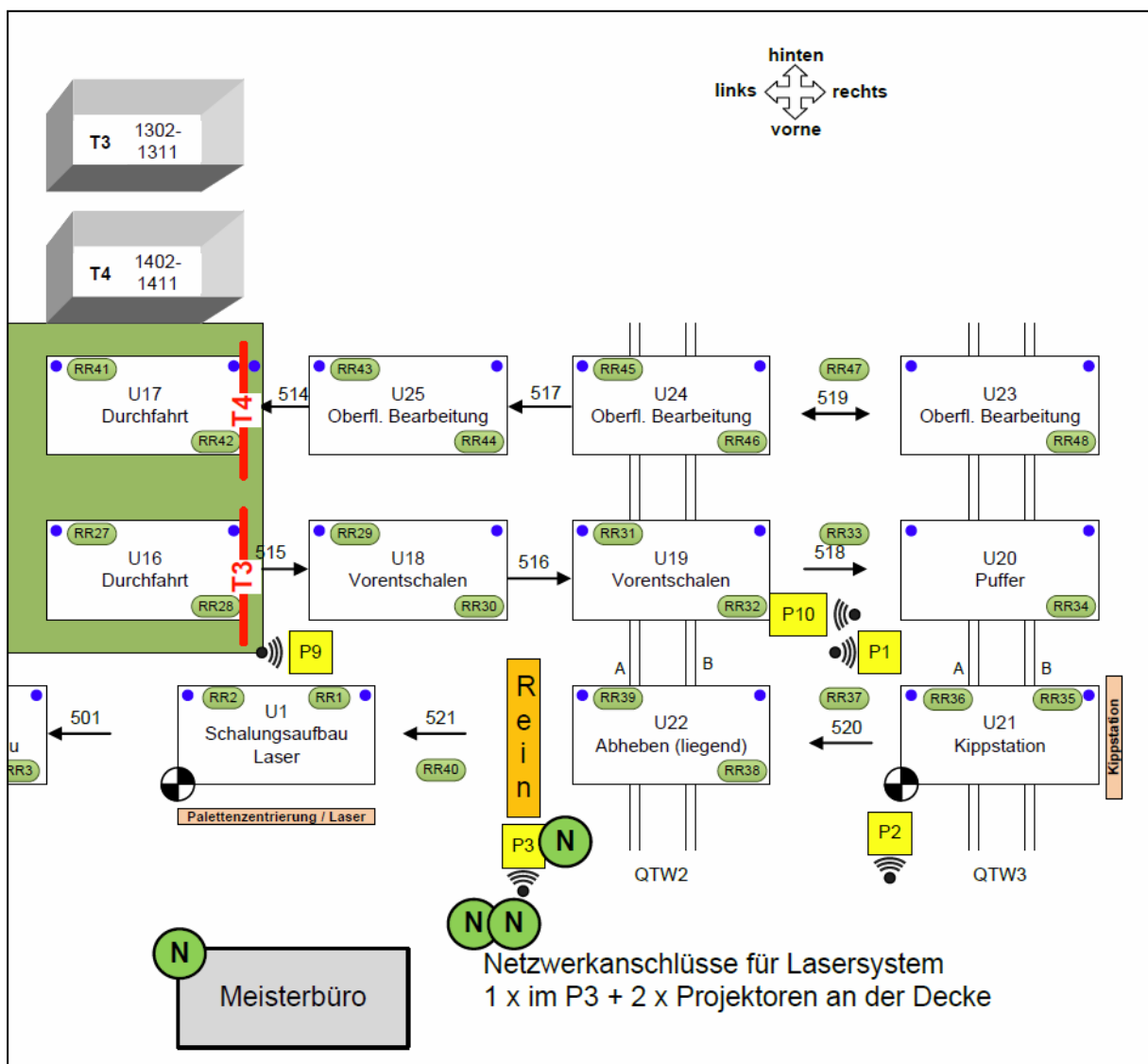


Abbildung 2: Layout einer Umlaufanlage (teilweise) (SAA Engineering, 2012)

## 4.2 Stationen

Als Station ist ein stationärer Platz, auf dem ein Bearbeitungsschritt maschinell oder manuell durchgeführt werden kann bezeichnet.

Jede Station verfügt über Sensorik zur Palettenerkennung, und Aktoren (Reibräder) mittels denen die Palette transportiert werden kann. (In der Abbildung 2 mit U bezeichnet.)



Abbildung 3: Betonverteilerstation (SAA Engineering, 2012)

## 4.3 Zwischenstationen

Ein Längstransport beschreibt einen Virtuellen zustand zwischen zwei Stationen, und ist damit der Weg zwischen diesen zu verstehen. (In der Abbildung 2 mit 500-520 Bezeichnet)

## 4.4 Quertransportwagen

Ein Quertransportwagen (QTW) ist eine mechanische Einrichtung, um Paletten quer zur Standardtransportrichtung zwischen Stationen welche parallel angeordnet sind zu transportieren. (In der Abbildung 2 mit QTW Bezeichnet.)

Ein QTW setzt dieses um, in dem er unter die Palette fährt, diese anhebt und seitwärts transportiert, und danach auf einer anderen Längstransportbahn absetzt.



Abbildung 4: Längstransportbahnen (Blau) und Quertransportwagen (Gelb) (SAA Engineering, 2012)

## 4.5 Bedienpulte

Bedienpulte stehen den Arbeitern im Werk als Benutzerinterface zur Verfügung, und sind hauptsächlich durch Taster und Kontrolllampen umgesetzt. Diese Pulte setzen Funktionen für eine oder mehrere Stationen um. (In der Abbildung 2 mit P bezeichnet)

Ein substantieller Teil der Aufgabenstellung beschäftigt sich mit den Logischen Verknüpfungen der Ein- und Ausgänge dieser Pulte, mit der restlichen Programmlogik für den Palettenumlauf.

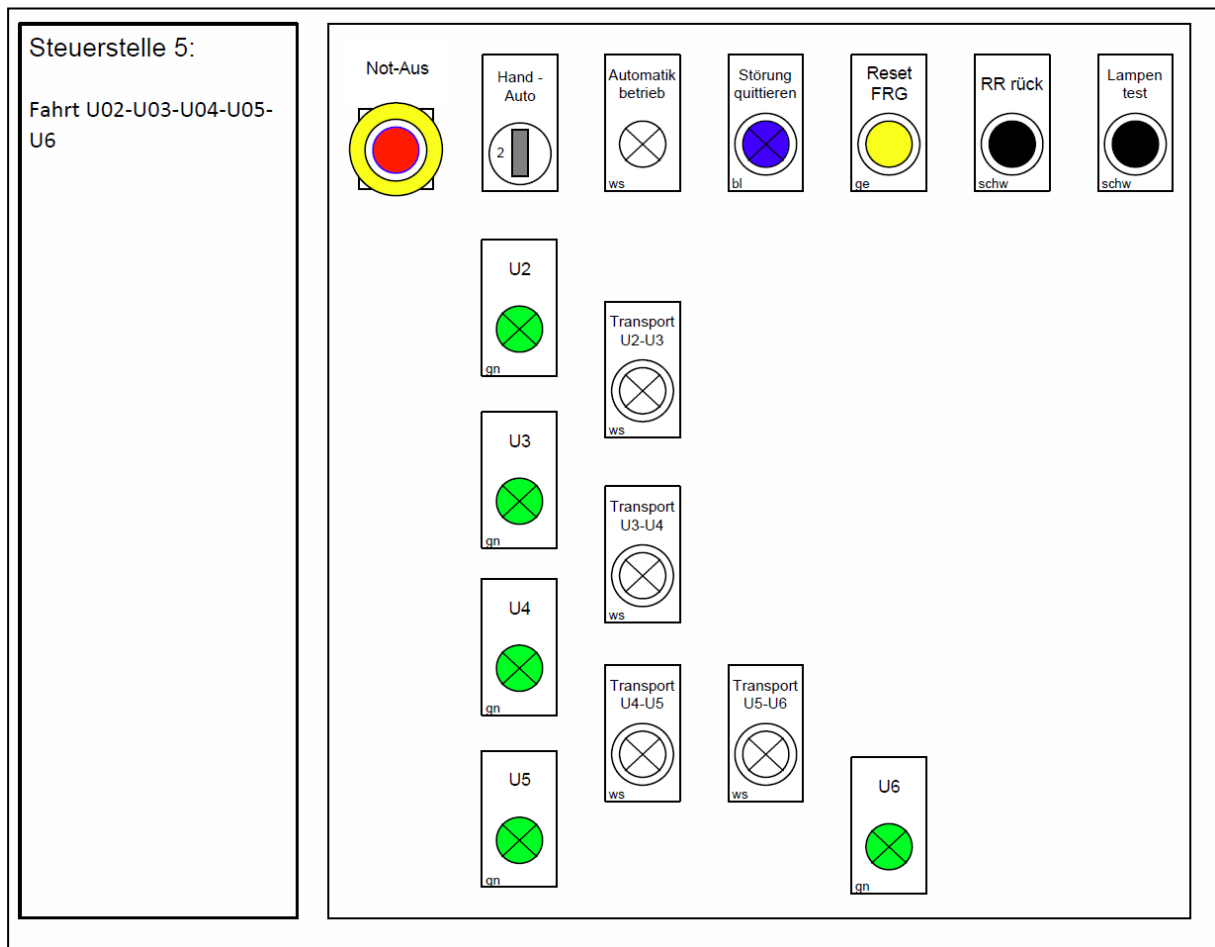


Abbildung 5: Layout Bedienpult (SAA Engineering, 2012)

## 5 Entwicklung des Benutzerinterfaces

Die Entwicklung des Benutzerinterfaces legt großen Wert auf schnelle, effiziente Dateneingabe. Es soll gewährleistet sein, dass die Anzahl an Arbeitsschritten minimiert wird, ohne die Flexibilität der Programmerstellung zu beeinträchtigen.

### 5.1 Lösungsansatz

Die Umsetzung der Aufgabe ist in Prinzipiell in drei Teile zu gliedern; Das Datenmodell, das Userinterface und den Codegenerator, der abschließend Quelltext für die SPS generiert. Hierbei wurde vor allem auf einen möglichst modularen Programmaufbau geachtet, um unter Umständen nachträgliche Änderungen oder Erweiterungen möglichst barrierefrei zu ermöglichen. (Schach, 2010)

Die speicherbaren erstellten Projektdateien sollen möglichst portabel und leicht verarbeitbar sein, deswegen wurde auf den offenen Standard XML gesetzt.

## **5.2 Benutzerfreundlichkeit und Workflow**

Als Hauptaugenmerk der Anwendung ist eine unkomplizierte und schnelle Dateneingabe angestrebt. Es wurden mehrere Eingabemasken entwickelt, die individuell auf eine Daten Art abgestimmt sind, um effiziente Dateneingabe zu gewährleisten.

Auch werden dem Benutzer aus zuvor eingegebenen Daten an diversen Stellen sinnvolle automatische Vorschläge präsentiert, die bei Bedarf nur bestätigt werden müssen, anstatt komplett eingetragen zu werden.

### **5.2.1 Eingabe der Daten**

Als Grundlage für das Benutzerinterface wurden die UI Guidelines von Microsoft verwendet. „Powerful and Simple“ (Microsoft, 2010) ist demnach die Grundlage eines gelungenen Anwendungsbausteines.

Gemeint ist hiermit eine Minimierung der dem Benutzer angezeigten Funktionen, bei gleichzeitiger Maximierung der Funktion.

Da der Anwender eine der Abb. 2 ähnliche Übersicht über das Transportsystem, eine der Abb. 5 ähnliche Abbildung der Pulte erhält, und um nicht allzu weit von der Vorhandenen Lösung abzuweichen, wurde die Dateneingabe Großteils mittels Tabellenelementen umgesetzt.

	Index	KurzBz	Bezeichnung	ZwStationLinks	ZwStationRechts	Standardziel	KonstanteMW
►	1	S01	Betonverteiler	501	510	2	2500
	2	S02	Puffer_Einlagern	502	501	3	2502
	3	S03	Auslagern	503	502	4	2504
	4	S04	Abheben	0	503	0	2506
	5	S05	MRP	511	0	0	2508
	6	S06	Handschalen	0	504	7	2510
	7	S07	Durchfahrt T7	504	505	8	2512
	8	S08	Durchfahrt T4	505	506	13	2514
	13	S13	Bewehrung Filzmoser	506	507	14	2524
	14	S14	Bewehrung Progress	507	508	15	2526
	15	S15	Puffer	508	0	0	2528
	16	S16	QTW gehoben Puffer	0	0	0	2530
	17	S17	Puffer	509	0	18	2532
	18	S18	Puffer	510	509	1	2534
	19	S19	MRP S5.1	0	511	5	2536
	20	S20	Virutell RBG	0	0	0	2538
*							

Abbildung 6: Eingabe der Stationen

Die einzelnen Elemente sind via Reiter getrennt, um die Eingabe nach Kategorien getrennt vorzunehmen.

Je mehr Daten der Anwendung bereits bekannt sind, desto konkreter können dem User Vorschläge zur Eingabe angeboten werden.

In den meisten Umlaufsystemen sind automatische Tore vorhanden, welche sich zwischen zwei Bearbeitungsstationen befinden. Da das gesamte Steuerungsprogramm der SPS die Elemente via Indices anspricht, ist die Zuordnung in SPS Code unübersichtlich und Fehleranfällig.

Die Lösung ist hier eine intelligente Eingabemaske, die nur zulässige Orte für diese Tore vorgibt

Stationen	LängsTrans	QTW	Pulte	RBG/Regale	Tore	Maschinen	Eingänge	Ausgänge
	index	KurzBz	Ort		Kommentar	KonstanteMW		
	1	Tor1	ZW503 (S04 <-> S03)	▼		600		
▶	2	Tor2	ZW501 (S02 <-> S01)	▼		602		
*			ZW501 (S02 <-> S01)					
			ZW502 (S03 <-> S02)					
			ZW503 (S04 <-> S03)					
			ZW504 (S06 <-> S07)					
			ZW505 (S07 <-> S08)					
			ZW506 (S08 <-> S13)					
			ZW507 (S13 <-> S14)					

Abbildung 7: Nur zulässige Optionen werden angezeigt

Da sich ein Tor nur zwischen 2 Stationen befinden kann, ist die Ortsauswahl eingeschränkt.

## 5.2.2 Eingangszuweisungen

Bei der Verknüpfung von Ein- und Ausgängen wurde der Workflow grundlegend geändert.

Bei der ursprünglichen Lösung mussten Ein- und Ausgänge manuell benannt und verknüpft werden.

Da die Bezeichnungen standardisiert sind, ist es möglich diese automatisch zu generieren.

Adresse	KurzBz	Kommentar	Typ
E21.0	biS01TrspSelbstLinks		Station / S01
E21.1	biS01LangsamL		Station / S01
E21.2	biS01StopL		Station / S01
E21.3	biS02TrspSelbstLinks		Station / S02
E21.4	biS02LangsamL		Station / S02
E21.5	biS02StopL		Station / S02
E21.6	biS03Folgefahrt1		Allgemein
E21.7	biS03Folgefahrt2		Allgemein
E22.0	biS03Folgefahrt3		Allgemein
E22.1	biS03Folgefahrt4		Allgemein
E22.2	biS03Folgefahrt5		Allgemein
E22.3	biS03LangsamL		Station / S03
E22.4	biS03StopL		Station / S03
E22.5	biP3S03TrspLinks1	3 mal B121 und B180	Allgemein
E22.6	biP3S03TrspLinks2	3 mal B121 und B180	Allgemein
E22.7	biS04TrspSelbstLinks		Station / S04
E23.0	biS04LangsamL		Station / S04
E23.1	biS04StopL		Station / S04
E23.2	biS05LangsamR		Station / S05

**E21.0**

Stat: S01 ▼ Betonverteiler

- ☐ iStopR
- ☒ iStopL
- ☐ iLangsamR
- ☒ iLangsamL
- ☐ iMotorUebertemp[1]
- ☐ iStoerungUmrichter[1]
- ☒ iTrspSelbstLinks
- ☐ iTrspSelbstRechts
- ☐ iMSFehlerRR

Abbildung 8: Eingangszuweisungsdialog

Der Arbeitsverlauf wurde dahingehend verändert, dass ein Benutzer nur den passenden Eingang mit einem Mausklick selektiert, und danach einen Typ (Station/QTW/Pult). Es werden von selbst nur passende Vorschläge angezeigt.

Mit einem weiteren Mausklick wird die Verknüpfung hergestellt, und es kann weiter verfahren werden.

Der Arbeitsaufwand reduziert sich von 5 bis 7 Aktionen (inklusive Tastatureinsatz / Scrollen der Maus) auf 3 (keine Tastatureingabe / Scrollen notwendig).

### **5.2.3 Ausgabe**

Da der Codegenerator viele verschiedene Quellcodeabschnitte generiert, welche an unterschiedlichsten Stellen des fertigen SPS Quelltextes einzügen sind, wurde eine Oberfläche entwickelt, die ein schnelles Einfügen in die STEP7 Umgebung ermöglicht, und gleichzeitig eine übersichtliche Kontrolle des erzeugten Codes zulässt.

Bei Klick auf die entsprechende Kategorie wird automatisch der Textfeldinhalt in die Windows Zwischenablage kopiert.

Dadurch kann schnell und effizient in die STEP7 Umgebung kopiert werden

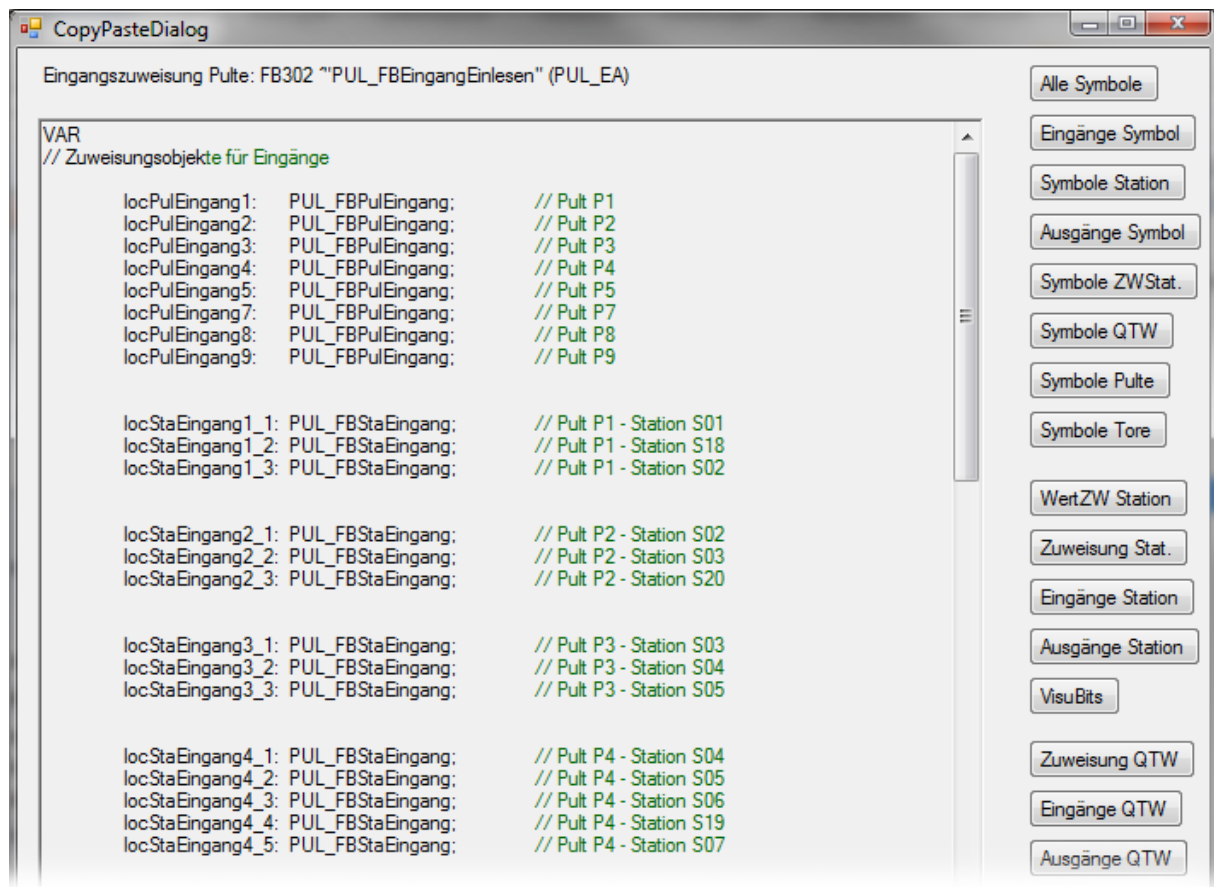


Abbildung 9: Ausgabefenster

## 5.3 Angewandte Software und Technologien

Die komplette Umsetzung erfolgte in C# 4.0 mit dem .NET Framework, Microsofts aktueller Programmiersprache.

### 5.3.1 C# / .net

*"C# is a general-purpose, type-safe, object-oriented programming language. The goal of the language is programmer productivity."*

(Albahari & Albahari, 2010)

C# ist eine von Grund auf objektorientierte Programmiersprache. Im Vergleich zu klassischem C++ leiten sich alle Sprachelemente von einem Mutterobjekt ab, und sind somit auch ableitbar. Zusätzlich ist C# eng in das .NET Framework von Microsoft angebunden. .NET erlaubt es, mittels wenig Programmieraufwand eine umfangreiche Anzahl an Problemen effizient zu lösen.

### 5.3.2 Common Interface Language

Die CIL ist die Zwischensprache von Microsofts .NET Framework.

Eine Zwischensprache erlaubt es, in einer Hochsprache geschriebenen Quellcode in plattformunabhängigen Code zu kompilieren. Als Hauptvorteil sind hier die erhöhte Portabilität und die Optimierung des Zielcodes für eine bestimmte Plattform zu erwähnen.

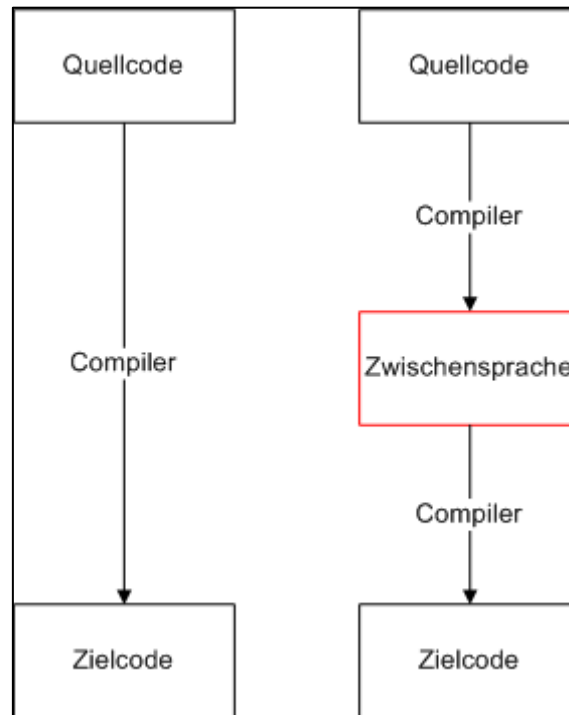


Abbildung 10: Kompilierung ohne und mit einer Zwischensprache (Quelle: (Bertram, 2009))

### 5.3.3 Visual Studio 2010

Visual Studio 2010 von Microsoft ist eine komplette Entwicklungsumgebung für (unter anderem) C#.

Es ist des Weiteren die empfohlene Entwicklungsumgebung für diese Hochsprache, und verfügt über spezielle Funktionen (z.B. den WinForms Designer) welche es ermöglichen schnell und einfach Anwendungen für die Windows Plattform zu entwickeln.

Im Besonderen sei hier „Intellisense“ zu erwähnen, mit dessen Textvervollständigungsfunktionalität eine komfortable und schnelle Codeeingabe ermöglicht wird. (Microsoft, 2012)

Mit diesen Eigenschaften eignet sich diese Entwicklungsumgebung optimal für die gegebene Problemstellung.

### **5.3.4 XML**

Die Extensible Markup Language, im weiteren „XML“ genannt, ist ein standardisiertes Datenformat zur Darstellung hierarchischer Daten. Dieses ist in menschenlesbarer Form ausgeführt, und durch das World Wide Web Consortium (W3C) spezifiziert.

Das XML Format wurde als Persistenzformat gewählt, da es erweiterbar und menschenlesbar ist, und dadurch auch von externen Programmen oder, wenn gewünscht, von einem Bediener auch ohne jegliche proprietärer Software eingesehen und bearbeitet werden kann. Die Datenspeicherung sowie diverse Fehlertext Importe und Exporte implementieren XML.

### **5.3.5 Extention Methods**

.NET erlaubt seit dem neuesten Framework (4.0) sogenannte „Extention Methods“ mittels denen es möglich ist, vorhandene Klassen um Funktionen zu erweitern, ohne eine neue Ableitung dieser zu erstellen.

Es ist somit Erreichbar, wenn das Programm erweitert wird, einfach und flexibel Anpassungen an vorhandenen Objekten durchzuführen.

## **5.4 Fehlertextgeneration**

Zusätzlich zur Codegeneration für die Umlaufsteuerung wurde auch eine Anbindung an eine vorhandene Fehlertextdatenbank implementiert, um den Bedienern der Umlaufanlage konkrete Fehlermeldungen anzeigen zu können.

Diese Fehlertexte sind im Umlaufsystem auch nur via Indizes vermerkt, bzw. für jeden Fehlerzustand gibt es einen definierten Text. Als Beispiel sei hier eine Fehlermeldung „Station belegt“ angeführt.

Dieser Fehler kann auf jeder Station auftreten, jedoch enthält dieser Text keine Referenz auf welcher Station er vorgefallen ist.

Auch das Editieren und Verwalten dieser Texte war umständlich, da hier direkt in SQL Datenbanktabellen zu arbeiten war.

Da das erstellte Programm bereits die komplette Umlauftopologie beinhaltet, war es naheliegend auch die Fehlertexterstellung und Verwaltung als Feature zu implementieren.

Des Weiteren ist es möglich, die Meldungen mit Eindeutigen Präfixen zu versehen, so das bei der Fehleranzeige an der Leitstelle sofort erkennbar ist, welche Station/Maschine/etc. betroffen ist.

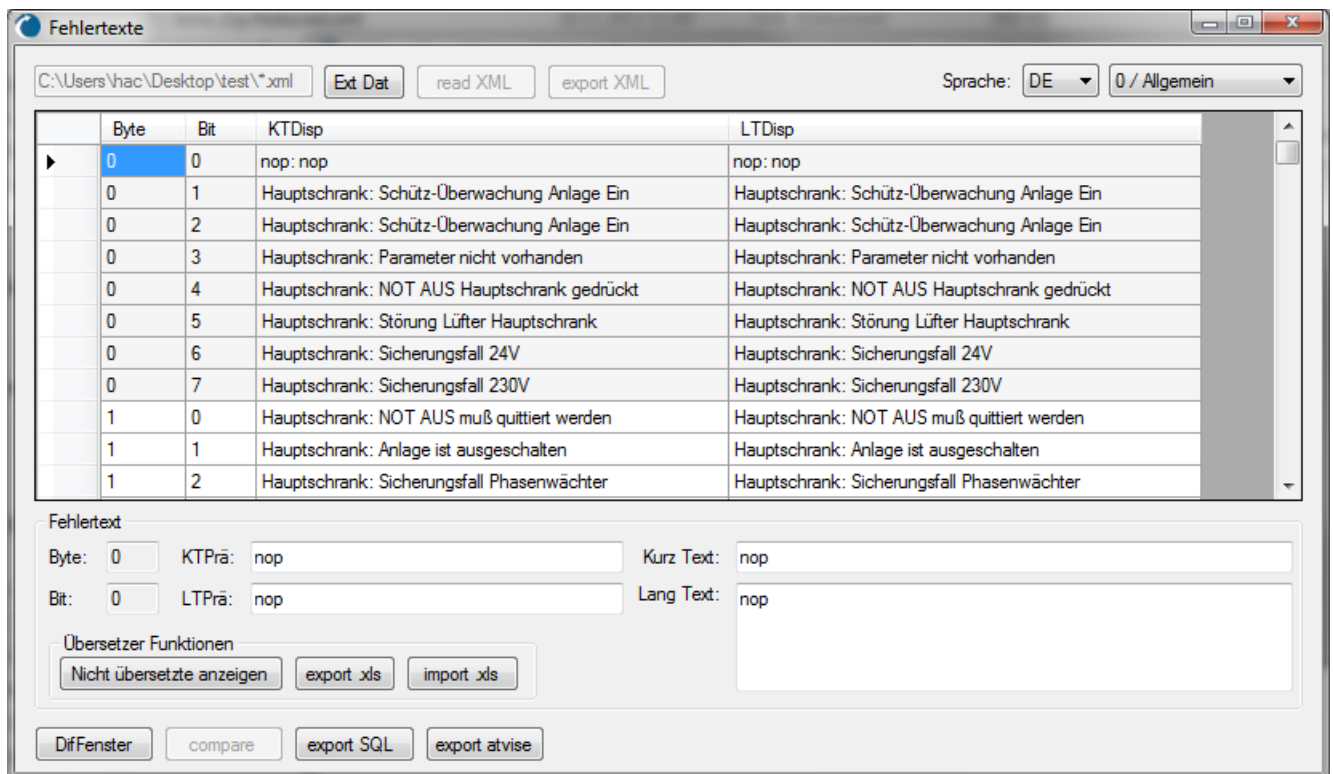


Abbildung 11: Fehlertext Editor / Manager

Das Programm wurde um ein Managementmodul erweitert, mittels dem es möglich ist, Fehlertexte benutzerfreundlich zu erstellen und zu bearbeiten.

### 5.4.1 Übersetzermodul

Da diese Fehlertexte zwar für diverse Anlagen hauptsächlich gleich sind, aber diese Anlagen in unterschiedlichen Regionen der Welt sind, ist Mehrsprachigkeit eine Voraussetzung.

Mit der bisherigen Lösung, war es nicht leicht möglich, die diversen Fehlertextdateien mehrsprachig synchron zu halten.

Zur besseren Verwaltung von mehrsprachigen Fehlertexten wurden Funktionen Implementiert, die Fremdsprachige Fehlertextdateien automatisch aktuell halten, indem vorhandene Sprachdateien miteinander verglichen werden.

Eine Spezielle Export Funktion ermöglicht es, noch nicht Übersetzte Texte in eine Datei zu exportieren, die von einem Übersetzerunternehmen verarbeitet werden kann.

Diese Datei ist dann einfach wieder importier bar.

### 5.4.2 SQL Export

SQL ist eine Datenbanksprache, die zur Verwaltung von Datenbanken angewendet wird.

Da die Fehlertexte für den Umlauf in einer Solchen vermerkt sind, ist es mit Hilfe des Fehlertextmanagers möglich, die Korrekten Texte einfach zu exportieren.

```
delete from fehlertexte where fehlerquellenid = 4472 and sprachid = 1;
insert into fehlertexte (fehlernummer, fehlerquellenid, sprachid, kurztext,langtext) Values
(000000,4472,1,'nop: nop','nop: nop');
insert into fehlertexte (fehlernummer, fehlerquellenid, sprachid, kurztext,langtext) Values
(000001,4472,1,'Hauptschrank: Schütz-Überwachung Anlage Ein','Hauptschrank: Der Anlage-Ein Schütz schaltet
nicht gleich wie der SPS-Ausgang');
insert into fehlertexte (fehlernummer, fehlerquellenid, sprachid, kurztext,langtext) Values
(000002,4472,1,'Hauptschrank: Schütz-Überwachung Anlage Ein','Hauptschrank: Die Hauptschütze der
Anlagenteile schalten nicht gleich wie das NOT-Aus Relais');
insert into fehlertexte (fehlernummer, fehlerquellenid, sprachid, kurztext,langtext) Values
(000003,4472,1,'Hauptschrank: Parameter nicht vorhanden','Hauptschrank: Parameter nicht vorhanden,
kopieren Sie die Daten vom Leit2000 in die SPS');
insert into fehlertexte (fehlernummer, fehlerquellenid, sprachid, kurztext,langtext) Values
(000004,4472,1,'Hauptschrank: NOT AUS Hauptschrank gedrückt','Hauptschrank: NOT AUS Hauptschrank
gedrückt');
```

Abbildung 12: Beispiel für SQL Syntax

Der SQL Syntax ist eher unübersichtlich, und schwer per Hand zu erstellen.

### 5.4.3 atvise SCADA

In den Betonanlagen wird zusätzlich oft das atvise Framework für die Statusvisualisierung verwendet.

Mit dem Fehlertextmodul ist es zusätzlich möglich, die Fehlertexte direkt in eine atvise Datenbank zu exportieren, um auch hier passende Texte anzuzeigen.

Da diese Funktionalität nicht direkt von atvise unterstützt wird, wurde mittels reverse-engineering das atvise-interne Dateiformat analysiert, und nachgebildet.

## 5.5 Routenplanung

Die ursprüngliche Methode der Routendefinition war eine einfache Textdatei, in die hintereinander die anzufahrenden Stationen per Index eingetragen waren.

Fehler in dieser Konfiguration wurden oft erst im Betrieb festgestellt und korreliert.

Um die Fehlerhäufigkeit zu minimieren, wurde die Eingabemaske so ausgeführt, das nur gültige Routen eingetragen werden können

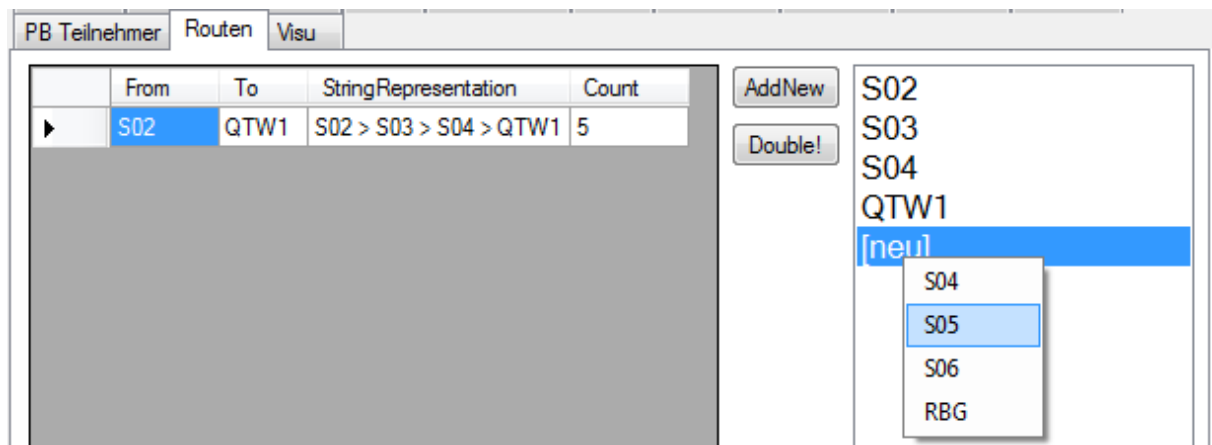


Abbildung 13: Routenkonfiguration

Der Anwender kann nun, ohne sich auf die Indizes konzentrieren zu müssen, mittels einfacher Mausbedienung nur gültige Routen anlegen.

## 5.6 Überblick über das Daten / Objektmodell

Aus den so gesammelten Daten, werden diese in das Objektorientierte Datenmodell eingetragen.

Die Folgenden Diagramme stellen einen Überblick über die Hauptobjekte dar.

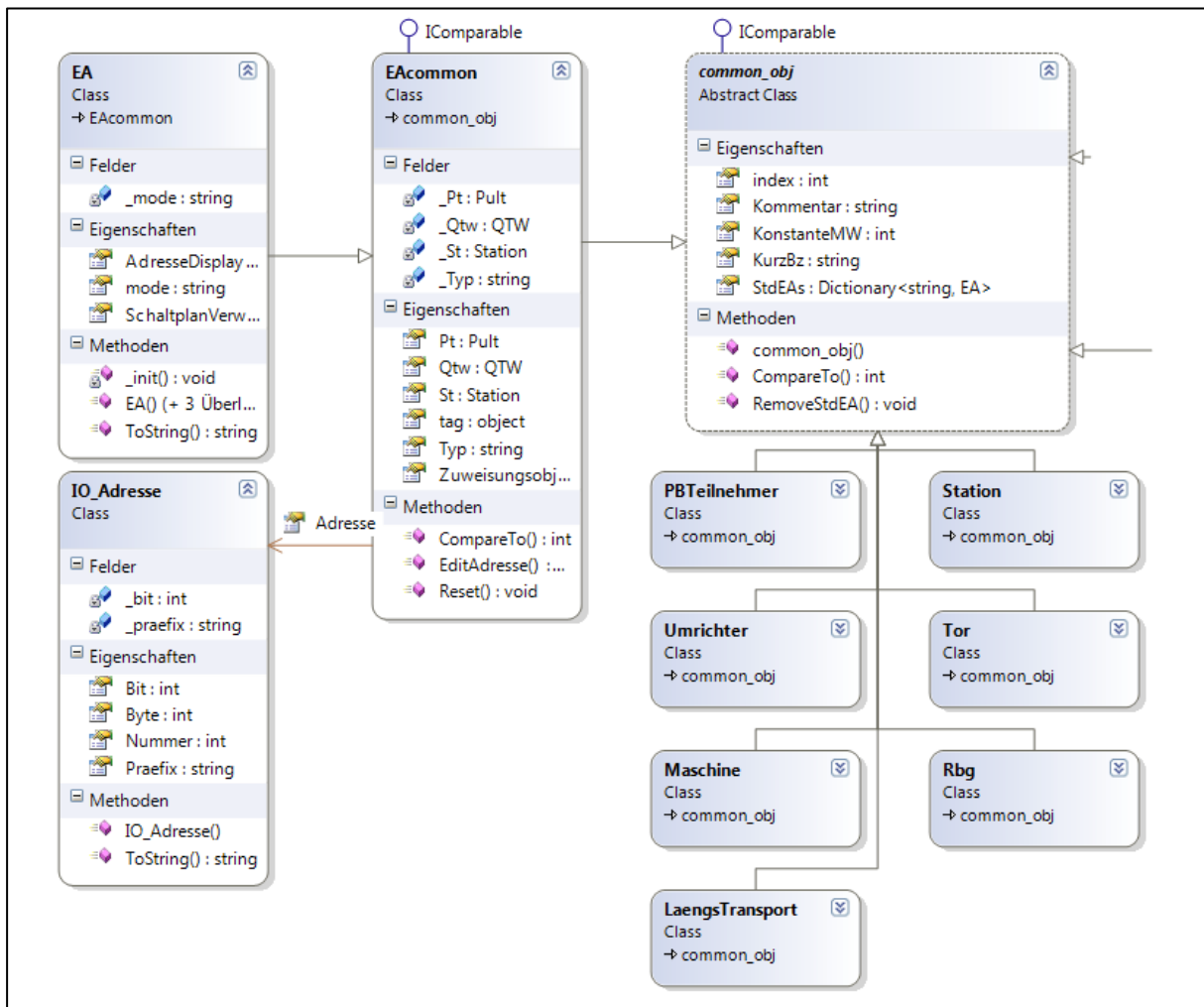


Abbildung 14: Objektmodell (Teil1)

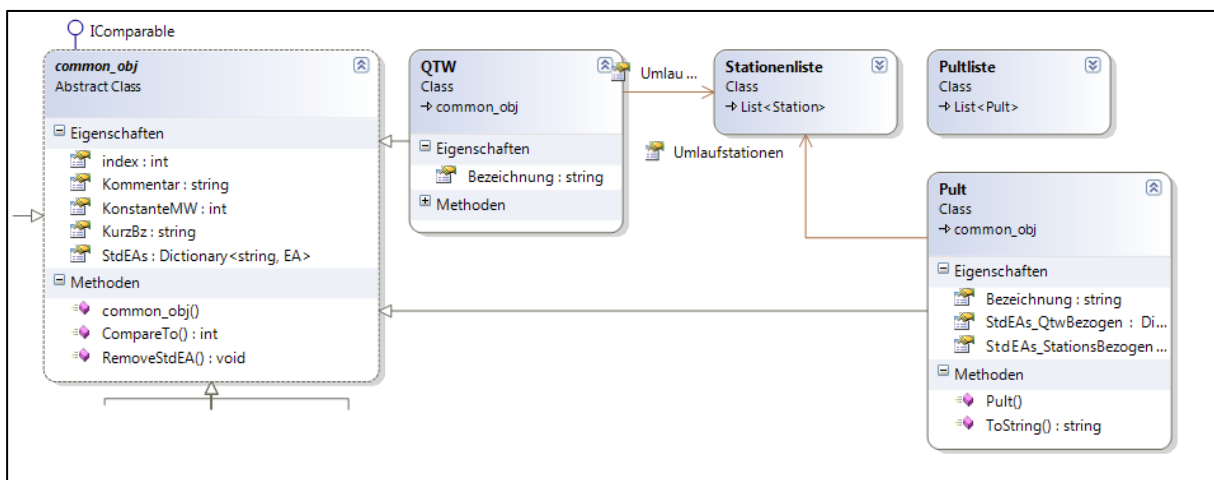


Abbildung 15: Objektmodell (Teil2)

## **6 Codegenerator für Persistenten Zielcode**

Durch konsequente Kapselung der Daten, ist es möglich den Codegenerator als eigenständiges Programmmodul auszuführen, welches eine niedrige Komplexität aufweist. Dies ist vor allem für die Wartbarkeit ausschlaggebend, da bei einer etwaigen Syntaxänderung der Zielplattform dieser schnell und effizient angepasst, oder sogar durch ein anderes Codegenerierungsmodul ersetzt werden kann.

### **6.1 Separation von Generator und Interface**

In der Ursprünglichen Excel-Lösung waren Elemente der Datenverarbeitung, Eingabe und Codegeneration eng miteinander vernetzt.

Ohne klare Trennung ist es kaum möglich, grundlegende Änderungen des Verhaltens ohne erheblichen Arbeitsaufwand umzusetzen.

Die Angewandte Trennung des Codegenerators von den Verarbeitungsalgorithmen und der Dateneingabe vermeidet diese Komplikationen.

Durch diese Trennung ist gewährleistet, dass auch nach mehreren Jahren eine Anpassung des Programms einfach möglich ist.

### **6.2 Wartungs- und Erweiterungsmöglichkeiten**

Durch die objektorientierte Umsetzung ist es für nachfolgende Entwickler äußerst einfach, bereits beschriebene Objekte um neue Funktionen und Eigenschaften zu erweitern.

Diese Änderungen sind dann auch im Benutzerinterface und im generierten Zielcode sofort erkennbar, was zu einem deutlich reduzierten Entwicklungsaufwand führt.

Es ist dadurch höchstmögliche Flexibilität gegeben.

### **6.3 STEP7 Syntax**

Der SPS Zielcode ist in STEP7/Strukturierter Text ausgeführt.

```
//----- P1 -----
locPulEingang1.iPultIndex :=1;
locPulEingang1(iqDatenEntry := PUL_DBPultDaten.Pulte[1].Input);
locPulEingang1.iNotAus :=      biP1NotAus;
locPulEingang1.iAutomatik :=    biP1Automatik;
locPulEingang1.iQuittError :=   biP1QuittError;
locPulEingang1.iLampentest :=   biP1Lampentest;
```

Abbildung 16: Beispiel für Zuweisungen in STEP7

STEP7 erlaubt mehrere Arten der Programmierung, wovon die Variante ST („Strukturierter Text“) in den meisten industriellen Anwendungen gebrauch findet, da sie bekannten Hochsprachen am ähnlichsten ist. (Berger, 2007 )

Da selbst diese Fortgeschrittene Methode prinzipiell sehr systemnahe ausgeführt ist, sind „Komfortfunktionen“ wie dynamische Arrays oder Listen nicht von Grund aus verfügbar.

Aus diesem Grund gelten mehrere Einschränkungen bei der Codeerstellung, wobei oftmals Viele Codezeilen für relativ simple Operationen von Nöten sind.

Bei der Manuellen Erstellung passieren hier oft Fehler, die durch automatische Codegeneration vermieden werden können.

Auch die derzeit erhältliche Entwicklungsumgebung ist im Punkt Fehlerprävention einer Automatischen Lösung unterlegen, da nur Syntaxfehler, nicht jedoch logische Fehler direkt erkannt werden können

## 7 Diskussion

Eine modellgetriebene Softwareerstellung ist in jedem Fall, der dieser Problemstellung ähnelt, klassischen Erstellungsmethoden (vor allem das manuelle erstellen) bezüglich Zeiteffizienz und Fehlerquote deutlich voraus.

### 7.1 Zusammenfassung

Zusammenfassend ist Festzustellen, dass eine direkte objektorientierte Umsetzung dieser Problemstellung einer linearen Codegeneration in Effizienz und Wartbarkeit wesentlich überlegen ist. Durch eine erheblich vereinfachte Bedienung, und Implementierung von Benutzerunterstützung, sind erwünschte Ergebnisse schneller und mit weniger Bearbeitungsschritten erreichbar.

Durch die eine konsequente Kapselung und Strukturierung sind auch zukünftige Erweiterungen des Programms nicht nur möglich, sondern auch ohne enormen Mehraufwand umzusetzen.

Durch die Umsetzung konnte die Vorbereitungszeit für Projekte von durchschnittlich drei Arbeitstagen auf ca. 8 Mannstunden reduziert werden. Zurückzuführen ist das einerseits auf einer stark reduzierten Fehleranfälligkeit der Quelltexte, als auch auf einen stark erhöhten Automatisierungsgrad bei deren Erstellung.

Die Syntaktische Fehlerrate konnte auf null gesenkt werden.

Die Codegeneration (Rechenzeit) konnte von ca. 50 auf 0.5 Sekunden (Standard Office-PC) reduziert werden. Durch einen modularen Aufbau ist es nun möglich, das Programm schnell zu erweitern.

Alle befragten Mitarbeiter ziehen die neue Methode der Umlaufkonfiguration der alten, linearen Arbeitsweise vor.

## **7.2 Vor- und Nachteile**

Als größter Vorteil ist auf jeden Fall die deutlich gesteigerte Zeiteffizienz im Vergleich zu der manuellen Codegeneration, als auch zu der Excel-basierten Lösung zu sehen.

Als weitere positive Entwicklung ist auch die Komplexität für den Bediener deutlich geringer, und somit die Einarbeitungszeit für einen neuen Mitarbeiter enorm verkürzt.

Durch Familiäre Benutzersteuerelemente und einen logischen Eingabeablauf ist es möglich, innerhalb kürzester Zeit mit dem Programm produktiv zu arbeiten.

Auch die gesteigerte Applikationsportabilität ermöglicht es nun, unabhängig von Microsoft-Office Paketen (insbesondere Excel) auf verschiedensten Rechnern auszuführen. Von Windows XP bis Windows 8 wird nun jede Plattform unterstützt.

Als Vernachlässigbarer Nachteil ist hier die „Umgewöhnungsphase“ der bereits bestehenden Anwender erwähnt, da die Dateneingabe deutlich von der gewohnten Arbeitsweise abweicht. Erste Testläufe mit Mitarbeitern lieferten jedoch durchwegs positives Feedback, woraus zu schließen ist, dass die Umstellung reibungslos verlaufen wird.

## **7.3 Ausblick**

Mit der Fertigstellung des Codegenerators ist ein Grundstein zur effizienten Arbeitsvorbereitung für Mitarbeiter der Firma SAA Software Engineering gelegt.

Die Optimierung des Arbeitsprozesses verspricht längerfristig deutliche Zeitersparnis bei der Inbetriebnahme von Betonwerkpalettenumlaufsystemen.

Nicht nur die schnellere Erstellung von SPS Initialisierungsroutinen, sondern auch die Fehlerreduktion ist ein deutlicher Schritt zur effizienteren Softwareentwicklung auf diesem Gebiet.

Durch strikte Trennung von Benutzerinterface und Generator sind Erweiterungen nicht nur möglich, sondern vorgesehen.

Erweiterungsmöglichkeiten wie z.B. Verwertung von bereits vorhandenem SPS Code zu Projektdateien sind angedacht.

Des Weiteren ist es in Zukunft möglich, auch andere Steuerungssysteme mit dem gleichen Interface zu bedienen, in dem eine Codegeneratorschicht mit anderer Syntax entwickelt werden kann

Die hier dargestellte Art von Codegeneration ist sicherlich die Methode der Zukunft, wenn es um effiziente Parametercodeerstellung geht.

## 8 Literaturverzeichnis

Albahari, J. & Albahari, B., 2010. *C# 4.0 in a Nutshell*. New York: O'REILLY.

Berger, H., 2007. *Automating with STEP 7 in STL and SCL: SIMATIC S7-300/400 Programmable Controllers*. Erlangen: Publicis Publishing.

Bertram, A., 2009. *Common Intermediate Language*. [Online]

Verfügbar unter:

<http://www.fh-wedel.de/~si/seminare/ws07/Ausarbeitung/08.cil/cil01.htm>

[Zugriff am 5.11.2013].

Karsai, G., 2004. *Generative Programming and Component Engineering*. Berlin Heidelberg: Springer-Verlag.

Meyer, B., 1986. *Genericity versus inheritance*, University of California, Santa Barbara: s.n.

Microsoft, 2010. *Windows User Experience Interaction Guidelines*. [Online]

Verfügbar unter:

<http://www.microsoft.com/en-us/download/confirmation.aspx?id=2695>

[Zugriff am 11.11.2012].

Microsoft, 2012. *Visual C#-IntelliSense*. [Online]

Verfügbar unter: <http://msdn.microsoft.com/de-de/library/vstudio/43f44291.aspx>

[Zugriff am 2.10.2012].

Schach, S. R., 2010. *Object-Oriented and Classical Software Engineering*. 8. Hrsg. New York: McGraw-Hill.

Sharp, J., 2010. *Microsoft Visual C# 2010 Step by Step*. Redmond: Microsoft Press.

## 9 Abbildungsverzeichnis

Abbildung 1: Screenshot (SAA Engineering, 2012) .....	8
Abbildung 2: Layout einer Umlaufanlage (teilweise) (SAA Engineering, 2012) .....	11
Abbildung 3: Betonverteilerstation (SAA Engineering, 2012).....	12
Abbildung 4: Längstransportbahnen (Blau) und Quertransportwagen (Gelb) (SAA Engineering, 2012) .....	13
Abbildung 5: Layout Bedienpult (SAA Engineering, 2012).....	14
Abbildung 7: Eingabe der Stationen.....	16
Abbildung 8: Nur zulässige Optionen werden angezeigt.....	17
Abbildung 9: Eingangszuweisungsdialog .....	17
Abbildung 10: Ausgabefenster .....	19
Abbildung 11: Kompilierung ohne und mit einer Zwischensprache (Quelle: (Bertram, 2009)) .....	20
Abbildung 12: Fehlertext Editor / Manager .....	22
Abbildung 13: Beispiel für SQL Syntax .....	23
Abbildung 14: Routenkonfiguration .....	24
Abbildung 15: Objektmodell (Teil1) .....	25
Abbildung 16: Objektmodell (Teil2) .....	25
Abbildung 17: Beispiel für Zuweisungen in STEP7 .....	27

# Abkürzungsverzeichnis

UI	User Interface
SQL	Structured Query Language
QTW	Quertransportwagen
XML	Extensible Markup Language