



Abschlussprüfung Sommer 2016

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Parsen eines Schemas in eine Baumstruktur

und zergliedern eines Datenstroms anhand dieses Schemas

Abgabetermin: Nürnberg, den 15.05.2016

Prüfungsbewerber:

René Ederer
Identnummer 4135980
Steinmetzstr. 2
90431 Nürnberg



Ausbildungsbetrieb:

PHOENIX GROUP IT GMBH
Sportplatzstr. 30
90765 Fürth

Ausbildende:

Frau Birgit Günther

Projektbetreuer:

Herr Marco Kemmer

Inhaltsverzeichnis

Inhaltsverzeichnis	I
Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Listings	V
Abkürzungsverzeichnis	VI
1 Projektdefinition	1
1.1 Auftraggeber	1
1.2 Projektumfeld	1
1.2.1 1920Schemas	1
1.2.2 Datenströme	2
1.3 Projektziel	2
1.4 Projektbegründung	2
1.5 Projektschnittstellen	3
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Ressourcenplanung	3
2.3 Entwicklungsprozess	4
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	4
3.2.1 „Make or Buy“-Entscheidung	4
3.2.2 Projektkosten	5
3.2.3 Amortisationsdauer	5
3.3 Anwendungsfälle	6
3.4 Qualitätsanforderungen	6
3.5 Zwischenstand	6
4 Entwurfsphase - Zergliedern des Datenstroms	6
4.1 Zielplattform	6
4.2 Aufbau der Schemadateien	7
4.3 Architekturdesign	7
4.4 Entwurf der Benutzeroberfläche	7
4.5 Geschäftslogik	8
4.6 Maßnahmen zur Qualitätssicherung	8

4.7	Pflichtenheft/Datenverarbeitungskonzept	8
4.8	Zwischenstand	8
5	Implementierungsphase - Zergliedern des Datenstroms	8
5.1	Implementierung der Benutzeroberfläche	9
5.2	Implementierung der Geschäftslogik	9
5.2.1	Parsen des Schemas in eine Baumstruktur	9
5.2.2	Grundschema der Methoden von AbstractNode	10
5.3	Zwischenstand	10
6	Entwurfsphase - Schema speichern und automatisch auswählen	11
6.1	Entwurf eines Bedienkonzeptes	11
6.2	Entwurf des XML-Schemas	11
7	Implementierung - Schema speichern	11
7.1	Erstellung des XSD-Datei	11
8	Abnahmephase	11
8.1	Abnahmetest	11
8.2	Zwischenstand	12
9	Einführungsphase	12
10	Dokumentation	12
10.1	Zwischenstand	12
11	Fazit	13
11.1	Soll-/Ist-Vergleich	13
11.2	Lessons Learned	13
11.3	Ausblick	14
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Beispiel für ein echtes 1920Schema	ii
A.3	Use Case-Diagramm	iii
A.4	Pflichtenheft (Auszug)	iv
A.5	Klassendiagramm	v
A.6	Aktivitätsdiagramm - Parsen eines 1920Schemas	vi
A.7	Objektdiagramm - Parsen des Schemas	vii
A.8	Quelltext Schema.cs	viii
A.9	Quelltext GroupNode.cs	ix
A.10	Quelltext ValueNode.cs	x
A.11	Entwicklerdokumentation	xi

Abbildungsverzeichnis

1	Beispiel für ein echtes 1920Schema	ii
2	Use Case-Diagramm	iii
3	Klassendiagramm	v
4	Aktivitätsdiagramm - Parsen eines 1920Schemas	vi
5	Diese Baumstruktur soll nach dem Parsen entstanden sein	vii
6	Schema.cs	viii
7	GroupNode.cs	ix
8	ValueNode.cs	x
9	Entwicklerdokumentation	xi

Tabellenverzeichnis

1	Beispiel für ein 1920Schema	1
2	Zeitplanung	3
3	Kostenaufstellung	5
4	Zwischenstand nach der Analysephase	6
5	Zwischenstand nach der Entwurfsphase	8
6	Zwischenstand nach der Implementierungsphase	10
7	Zwischenstand nach der Abnahmephase	12
8	Zwischenstand nach der Dokumentation	13
9	Soll-/Ist-Vergleich	13

Listings

Abkürzungsverzeichnis

GUI	Graphical User Interface
Regex	Regular Expression
SVN	Subversion
WPF	Windows Presentation Foundation

1 Projektdefinition

1.1 Auftraggeber

Die Phoenix Pharmahandel GmbH & Co. KG und ihre Tochtergesellschaften sind europaweit unter dem Namen “Phoenix group” mit etwa 30000 Mitarbeitern im Pharmagroßhandel tätig. Ihre Haupttätigkeiten sind der Großhandel mit Pharmaprodukten und die Belieferung von Apotheken mit Medikamenten.

Auftraggeber des Projektes ist die Phoenix group IT GmbH. Sie hat etwa 200 Mitarbeiter und unterstützt die Phoenix group durch die Bereitstellung von IT-Dienstleistungen.

1.2 Projektumfeld

Die Phoenix group IT GmbH ist weiter unterteilt in die Abteilungen Inbound, Outbound und Warehouse. Das Projekt findet in der Abteilung Warehouse statt, in der überwiegend mit COBOL gearbeitet wird.

Phoenix verwendet ein firmeneigenes Dateiformat als Schnittstelle zu verschiedenen Programmen und Diensten. Es wird im Folgenden 1920Schema genannt und spielt eine zentrale Rolle für das Projekt.

1.2.1 1920Schemas

1920Schemas sind Textdateien, die einen Satz hierarchisch gegliederter Variablen beschreiben. Phoenix nutzt diese Schemas unter anderem als Schnittstelle, um Daten von seinem Mainframe zum Lagerrechner zu schicken, als Vorlage für Copybooks¹ und als Schnittstelle zu SSORT².

Beispiel für ein 1920Schema

Level	Name	Typ	Bytes	Wiederholzahl	Kommentar
01	Daten				Level und Name sind Pflichtangaben
03	Personendaten				Level gibt die Hierarchiestufe an
05	Vorname	C	5		fünf Bytes vom Typ char
05	Nachname	C	4		
R03	Gesamter-Name	C	9		redefiniert Personendaten
03	Bestellungen			2	Ein “Array”, Länge 2
05	Artikelnr	N	3		drei Byte lange Artikelnummer

Tabelle 1: Beispiel für ein 1920Schema

Abbildung A.2 ist ein echtes 1920Schema, es ist im Anhang beigelegt.

¹COBOL-Datei, in der eine Variablenstruktur definiert wird

²IBM-Programm, zeigt Copybooks an

1.2.2 Datenströme

Anhand von 1920Schemas werden Datenströme zergliedert und erhalten dadurch eine Bedeutung. Datenströme können prinzipiell beliebige Zeichen³ enthalten, nur die Anzahl muss ausreichen, um jeder Schema-Variablen einen Wert zuzuweisen. Das Schema aus Tabelle 1 beschreibt einen 15 Byte langen Datenstrom (5+4+2*3), der so aussehen könnte:

Beispiel für einen Datenstrom

VN~~~NN~~123456

Datenströme sind fast immer 1920 Bytes lang.

1.3 Projektziel

Es soll ein Programm geschrieben werden, das nach Eingabe von Datenstrom und 1920Schema den Datenstrom dem Schema entsprechend zergliedert und anzeigt. Das Programm wird im Folgenden 1920Parser genannt.

Gewünschte Ausgabe (mit den Beispielwerten der Abschnitte 1.2.1 und 1.2.2)

01 Daten

03 Personendaten

05 Vorname=VN~~~

05 Nachname=NN~~

R03 Gesamter-Name=VN~~~NN~~

03 Bestellungen(1)

05 Artikelnr=1234

03 Bestellungen(2)

05 Artikelnr=5678

1.4 Projektbegründung

Bei Kundenreklamationen, Änderungen an Programmen und Neuentwicklungen stehen die Warehouse-Programmierer oft vor den Problemen:

- Wert einer Schemadatei-Variablen in einem Datenstrom finden.
- Datenstrom-Bytes einer Schemadatei-Variablen zuordnen.

Gegenwärtig zählen die Programmierer die passende Anzahl von Bytes in Schema und Datenstrom ab, einige erfahrene kennen die wichtigsten Schemadateien auch teilweise auswendig.

³Ein Byte entspricht einem Zeichen. Die Angaben in 1920Schemas haben die Einheit Byte, im Zusammenhang mit dem Datenstrom ist der Begriff Zeichen aber manchmal anschaulicher. Eine Unterscheidung der beiden Begriffe ist für das Projekt unwesentlich.

1.5 Projektschnittstellen

Benutzer des Programms sind die Programmierer der Phoenix Group IT GmbH, hauptsächlich die aus der Abteilung Warehouse.

1920Parser soll nicht unmittelbar mit anderen Systemen interagieren. Vorgesehen ist, dass die Benutzer die notwendigen Angaben in eine Eingabemaske hineinkopieren.

Projektgenehmigung und die Bereitstellung von Ressourcen erfolgt durch die Ausbildende Frau Birgit Günther, die Projektbetreuung und die Abnahme des Programms durch Herrn Marco Kemmer. Herr Kemmer arbeitet in der Abteilung Warehouse als COBOL-Entwickler, er kommuniziert die Kundenwünsche und will das Programm in Zukunft auch selbst nutzen.

2 Projektplanung

2.1 Projektphasen

Für das Projekt standen 70 Stunden zur Verfügung. Es fand im Zeitraum vom 11.04.2016 - 15.05.2016 statt.

Beispiel Tabelle 2 zeigt die grobe Zeitplanung für das Projekt.

Projektphase	Geplante Zeit
Analysephase	6 h
Entwurfsphase	8 h
Implementierungsphase	44 h
Abnahmetest der Fachabteilung	2 h
Erstellen der Dokumentation	10 h
Gesamt	70 h

Tabelle 2: Zeitplanung

Eine detailliertere Zeitplanung findet sich im Anhang [A.1: Detaillierte Zeitplanung](#) auf Seite i.

2.2 Ressourcenplanung

Zur Projektdurchführung werden folgende Ressourcen eingesetzt: Windows 7, Visual Studio Professional 2010, Microsoft Visio 2010, TexMaker, XMLSpy 2007, OpenText HostExplorer, Büro mit PC mit Verbindung zum Mainframe und Verbindung zum Internet, Strom, Projektbetreuer

2.3 Entwicklungsprozess

Es wurde ein agiler Entwicklungsprozess angewendet, der an Extreme Programming angelehnt war. Der im Projekt angewandte Prozess beinhaltet die Extreme Programming-Praktiken, testgetriebene Entwicklung, häufige Kundeneinbeziehung, häufiges Refaktorisieren, kurze Iterationen und einfaches Design.

3 Analysephase

3.1 Ist-Analyse

Die Ist-Analyse fand durch Befragung von Herrn Kemmer statt. Die Entwickler der Abteilung Warehouse müssen bei Kundenreklamationen, Programmänderungen und Neuentwicklungen die Werte in den beschriebenen Datenströme analysieren. Diese stammen aus den Logdateien des zentralen Lagerrechners, aus der Eingabemaske des Mainframes oder auch aus erhaltenen Email-Anhängen. Um die Bedeutung der Werte im Datenstrom herauszufinden, zählen die Entwickler gegenwärtig die Bytes in Schema und Datenstrom ab, einige erfahrenere kennen die wichtigsten Schemadateien auch zum Teil auswendig. Im Warehouse arbeiten etwa 20 Entwickler, etwa 10 davon haben regelmäßig mit solchen Datenströmen zu tun. Es werden Dutzende verschiedene 1920Schemas für verschiedenste Zwecke verwendet, aber mit nur etwa 10 arbeiten die Entwickler regelmäßig.

3.2 Wirtschaftlichkeitsanalyse

Das Programm soll den Entwicklern das bisher notwendige, fehleranfällige Abzählen von Zeichen in Schema und Datenstrom abnehmen. Das Projekt verspricht dadurch nicht nur, den Entwicklern Zeit zu sparen, sondern auch Zählfehler wirkungsvoll zu verhindern.

3.2.1 „Make or Buy“-Entscheidung

Die Anforderungen sind so speziell, dass fast auszuschließen ist, dass es außerhalb von Phoenix ein Programm gibt, das die Anforderungen erfüllt. Zu bemerken ist aber, dass viele Programme bei Phoenix mit Schemadateien arbeiten. Der Autor fragte deshalb nach, ob Phoenix eventuell schon ein Programm hat, das die Anforderungen erfüllt. Der Auftraggeber antwortete, dass die Datenströme aus so unterschiedlichen Umgebungen stammen (Unix-Lagerrechner, Mainframe, Email), dass die COBOL-Programme, die sie verarbeiten, nur eingeschränkt eingesetzt werden können. Es wurde entschieden, das Programm selbst neu zu schreiben.

3.2.2 Projektkosten

Die Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Der Projektersteller ist Umschüler und erhält deshalb von seinem Ausbildungsbetrieb keine Vergütung.

$$7,7 \text{ h/Tag} \cdot 220 \text{ Tage/Jahr} = 1694 \text{ h/Jahr} \quad (1)$$

$$0 \text{ €/Monat} \cdot 13,3 \text{ Monate/Jahr} = 0 \text{ €/Jahr} \quad (2)$$

$$\frac{0 \text{ €/Jahr}}{1694 \text{ h/Jahr}} = 0,00 \text{ €/h} \quad (3)$$

Dadurch ergibt sich also ein Stundenlohn von 0,00 €. Die Durchführungszeit des Projekts beträgt 70 Stunden. Für die Nutzung von Ressourcen⁴ wird ein pauschaler Stundensatz von 12 € angenommen. Für die anderen Mitarbeiter wird pauschal ein Stundenlohn von 23 € angenommen. Eine Aufstellung der Kosten befindet sich in Tabelle 3 und sie betragen insgesamt 1015,00 €.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	0,00 € + 12 € = 12,00 €	840,00 €
Fachgespräch	3 h	23 € + 12 € = 35 €	105,00 €
Abnahmetest	2 h	23 € + 12 € = 35 €	70,00 €
			1015,00 €

Tabelle 3: Kostenaufstellung

3.2.3 Amortisationsdauer

Es wird davon ausgegangen, dass ausschließlich Entwickler der Abteilung Lager das Programm nutzen werden. Nach Einschätzung von Herrn Kemmer arbeitet die Hälfte der 20 Entwickler regelmäßig mit Schemadateien und dass das Programm jedem täglich 10 Minuten einsparen kann. Bei einer Einsparung von 10 Minuten am Tag für 10 Entwickler an 220 Arbeitstagen im Jahr ergibt sich eine gesamte Zeiteinsparung von

$$10 \cdot 220 \text{ Tage/Jahr} \cdot 10 \text{ min/Tag} = 22000 \text{ min/Jahr} \approx 366,67 \text{ h/Jahr} \quad (4)$$

Dadurch ergibt sich eine jährliche Einsparung von

$$366,67 \text{ h} \cdot (23 + 12) \text{ €/h} = 12833,45 \text{ €} \quad (5)$$

⁴Räumlichkeiten, Arbeitsplatzrechner etc.

Die Amortisationsdauer beträgt also $\frac{1015,00 \text{ €}}{12833,45 \text{ €/Jahr}} \approx 0,08 \text{ Jahre} \approx 1 \text{ Monat}$.

3.3 Anwendungsfälle

Der mit Abstand wichtigste Anwendungsfall ist, dass das Programm nach Angabe von Datenstrom und Schema den Datenstrom entsprechend dem Schema zergliedert anzeigt. Der Kunde nannte noch einige Wünsche zur Funktionalität. Ein Usecase-Diagramm ist im Anhang beigelegt.

3.4 Qualitätsanforderungen

Schemas müssen frei angebar sein und Datenströme richtig zergliedert werden. Beim Abnahmetest soll dies anhand der 5 wichtigsten Schemadateien überprüft werden. Die Performance des Programmes ist ziemlich egal, es soll aber flüssig benutzbar sein (Zergliederung von Datenstrom und Anzeige in unter 5 Sekunden). Weil die Benutzer Profis sind, ist es nicht unbedingt notwendig, für alles eine Eingabemaske bereitzustellen. Es genügt auch, wenn eine Einstellung durch Editieren einer Konfigurations-Datei geändert werden kann.

3.5 Zwischenstand

Tabelle 4 zeigt den Zwischenstand nach der Analysephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Analyse des Ist-Zustands	3 h	2 h	-1 h
2. Wirtschaftlichkeitsprüfung und Amortisationsrechnung	1 h	2 h	+1 h
3. Erstellen des Lastenhefts	2 h	2 h	

Tabelle 4: Zwischenstand nach der Analysephase

4 Entwurfsphase - Zergliedern des Datenstroms

4.1 Zielplattform

Das Programm soll auf den Entwicklerrechnern der Phoenix unter Windows 7 laufen. Die Wahl der Programmiersprache wurde zunächst auf die bei Phoenix eingesetzten Sprachen COBOL, C++ und C# eingegrenzt. COBOL schied als Programmiersprache für ein Windows-Tool aus, aber C++ mit Qt und C# waren beide geeignet. Die hohe Performance, die C++ verspricht, wurde für 1920Parser aber nicht wirklich benötigt. Die Wahl fiel auf C# aufgrund von dessen Garbage Collection, **Linq!** (**Linq!**) und gutem GUI-Designer.

4.2 Aufbau der Schemadateien

Zur Analyse des Aufbaus der 1920Schemas fragte der Autor den Auftraggeber nach den Namen der 10 wichtigsten Schemadateien und ludt diese unter Verwendung von OpenText HostExplorer 2014 vom Mainframe herunter.

Level und Name sind Pflichtangaben, alle anderen Felder dürfen leer sein. Typ und Byteanzahl treten nur zusammen auf, solche Variablen sind Wertvariablen (sie definieren Bytes aus dem Datenstrom). Variablen ohne Typ und Bytezahl sind Gruppenvariablen. Zusätzlich zu diesen Variablenzeilen enthalten 1920Schemas Zeilen mit Metainformationen zum Schema. Die Metainformationen sind für 1920Parser nicht relevant und werden ignoriert. Die Angabe zur Byte-Anzahl kann auch die Form "AnzahlGesamtstellen,AnzahlNachkommastellen" haben.

Es ergaben sich unerwartet Schwierigkeiten beim Feld Kommentar. In der Schemadatei IOVK92 gab es eine Variable, deren Kommentar-Feld mit "0" begann. Es musste eine Regel festgelegt werden, damit diese 0 nicht als Wiederholzahl interpretiert wird, sondern als Kommentar-Beginn. Es gab die Möglichkeiten, 0 als Wiederholzahl zu verbieten oder die untereinander-stehende Anordnung der Variablenfelder auszunutzen. Falsche Annahmen hätten zu Fehlfunktion des Programms geführt; es war deshalb nötig, Rücksprache mit dem Auftraggeber zu halten. Auf dessen Vorschlag und nach erneuter Prüfung der Schemadateien wurde festgelegt, dass ein Kommentar mit mindestens 2 Leerzeichen vom vorhergehenden Feld getrennt sein muss.

4.3 Architekturdesign

Während der Projekterstellung wurde auf Kohärenz der Klassen geachtet und gegebenenfalls refaktoriert (zum Beispiel stand die Funktionalität zum Lesen und Schreiben der XML-Config ursprünglich auch in der Klasse Schema, wurde später aber in die neu erschaffene Klasse SchemaManager ausgelagert). Im Projektverlauf entstand eine 3-Schichten-Architektur mit den Klassen 1920ParserView und SaveSchemaView als Teil der Präsentationsschicht, den Klassen AbstractNode, GroupNode, ValueNode und Schema auf Logikschicht und SchemaManager und SchemaConfig als Teil der Datenhaltungsschicht. Klassen der Präsentationsschicht und Klassen Datenhaltungsschicht kennen sich nicht und deren Objekte kommunizieren auch nicht direkt miteinander.

4.4 Entwurf der Benutzeroberfläche

Der Kunde hatte keine Vorgaben bezüglich der Benutzeroberfläche gemacht. Nachdem als Programmiersprache C# feststand, kamen für die Graphical User Interface (GUI) nur Plattformen aus dem .NET-Framework in Betracht. Da Benutzer Schemas speichern können sollen, wäre ein Webinterface (ASP.NET) mit zusätzlichem Aufwand verbunden gewesen (Benutzerverwaltung/Datenbank). Windows Presentation Foundation (WPF) und Winforms waren beide geeignet. Der Autor entschied sich für Winforms, weil er damit mehr Erfahrung hatte.

4.5 Geschäftslogik

Die hierarchische Aufbau von 1920Schemas lässt sich gut mit einer rekursiven Baumstruktur im Programm abbilden. Für die Baumstruktur wurde die abstrakte Klasse `AbstractNode` entworfen. Von dieser Klasse erben die Klassen `GroupNode` und `ValuNode`, die Gruppen- und WerteVariablen darstellen. `GroupNode` erhielt ein Attribut `children` vom Typ `List<AbstractNode>`, mit dem es auf seine Kind-Knoten verweisen kann. `GroupNodes` erlauben durch ihre rekursive Definition eine beliebig tiefe Verschachtelung von `AbstractNodes`.

Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen untereinander darstellt kann im Anhang ?? auf Seite ?? eingesehen werden.

4.6 Maßnahmen zur Qualitätssicherung

Das korrekte Parsen von 1920Schemas und die richtige Zergliederung des Datenstroms sind zentral für 1920Parser. Um sicherzugehen, dass dieser Programmteil funktioniert, wurden die Methoden der Klassen `Schema`, `Abstract-`, `Group-` und `ValueNode` testgetrieben entwickelt.

4.7 Pflichtenheft/Datenverarbeitungskonzept

Ein Auszug aus dem Pflichtenheft ist im Anhang [A.4: Pflichtenheft \(Auszug\)](#) auf Seite iv zu finden.

4.8 Zwischenstand

Tabelle 5 zeigt den Zwischenstand nach der Entwurfsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Analyse des Aufbaus der Schemadateien	2 h	2 h	
2. Entwurf der Klassen für die Baumstruktur	4 h	2 h	-2 h
3. Erstellen des Pflichtenhefts	2 h	2 h	

Tabelle 5: Zwischenstand nach der Entwurfsphase

5 Implementierungsphase - Zergliedern des Datenstroms

Die Implementierung begann mit dem Anlegen eines neuen Subversion ([SVN](#))-Repositories mit der vorgegebenen Verzeichnisstruktur (branch, tag, trunk) und dem Erstellen einer neuen Solution in Microsoft Visual Studio Professional 2010 mit einem C# Winforms Projekt. Der Solution wurde ein NUnit-Testprojekt für die Unit-Tests hinzugefügt.

5.1 Implementierung der Benutzeroberfläche

Visual Studio erstellte mit Neuanlage des Winform-Projekts automatisch eine Form-Klasse. Mit dem GUI-Editor von Winforms wurden dieser Klasse drei Textfelder hinzugefügt, jeweils eines für Datenstrom, Schema und Ergebnis. Es wurde noch ein Button zum Starten der Verarbeitung eingefügt. Dieser wurde im Laufe der Implementierung entfernt und die Verarbeitung bei jeder Eingabe in Schema- oder Datenstrom-Textfeld gestartet.

5.2 Implementierung der Geschäftslogik

5.2.1 Parsen des Schemas in eine Baumstruktur

Parsen einer Variablenzeile zu einer AbstractNode Aus jeder Variablenzeile im Schema wird eine Group- oder ValueNode erstellt. Um dies umzusetzen wurde in der Klasse Schema die Methode ParseLine geschrieben. Die Methode splittet die Angaben einer Variablenzeile mit Hilfe einer Regular Expression (Regex) in ihre Felder auf. Die Regex wurde auf <https://regex101.com> angepasst und getestet, bis sie richtig funktionierte.

Erstellen einer Baumstruktur aus den Variablenzeilen Dies war die schwierigste Aufgabe des Projektes. Die Schemazeilen mussten durchlaufen werden, aus ihnen AbstractNodes erstellt und diese in eine Baumstruktur gebracht werden. Schwierig war, dass mit jeder neuen Zeile ein Kindknoten, ein Geschwisterknoten, oder ein Geschwisterknoten von einem früheren Vorfahren auftauchen konnte. In jedem dieser Fälle konnte unterhalb des Knotens erneut eine Hierarchie von Knoten sein. Es musste diese Hierarchie, falls vorhanden, erstellt werden, danach musste die Verarbeitung mit der richtigen Schemazeile und dem richtigen Elternknoten weitergehen.

Schwierig war insbesondere, dass eine Schemazeile zu mehreren AbstractNodes werden konnte, abhängig von ihrer Wiederholzahl. In dieser Hinsicht waren besonders GroupNodes problematisch, da sich bei GroupNodes mit der Wiederholzahl der gesamte Baum unterhalb von ihnen wiederholt. Außerdem konnten Wiederholzahlen auch bei mehreren Vorfahren-Knoten vorkommen, so dass sich die Wiederholungen verschachtelten.

Zunächst wurde versucht, innerhalb einer Schleife, die die Schemazeilen durchlief, für jeden Kindknoten die Methode rekursiv erneut aufzurufen und der Methode die Schemazeilenliste mitzugeben (in der ein oder anderen Form, per Value, per Referenz, eine Kopie per Referenz...). Dies führte nicht zum Erfolg, vor allem war es problematisch, dem Lauf des Algorithmus zu folgen, der gleichzeitig iterativ die Schemazeilen durchlief, Levelnummern verglich und in einer weiteren Schleife der Wiederholungszahl entsprechend rekursive aufrufe machte.

Schließlich führte eine Kombination aus zwei Ideen zum Erfolg:

1. statt Rekursion einen Stack zu verwenden

2. nicht zu versuchen eine AbstractNode mehrfach zu erstellen, sondern die Node erst fertigzustellen samt eventuellem Unterbaum und danach eine tiefe Kopie zu erzeugen.

Der Algorithmus zum Parsen des Schemas nutzt einen Stack. Der Stack enthält zu jedem Zeitpunkt die Vorfahren des gerade bearbeiteten Knotens, mit seinem Elternknoten oben und dem Wurzelknoten unten.

5.2.2 Grundschema der Methoden von AbstractNode

Methodenname	Rückgabewert ValueNode	Rückgabewert GroupNode
Equals	bool, ob alle Attribute übereinstimmen	bool, ob alle Attribute übereinstimmen und alle Kinder
ToString	String mit Level, Namen und Wert	String aus Level, Name und den Rückgabewerte der Kinder

Group- und ValueNode erben von AbstractNode Methoden. Die Implementierung all dieser Methoden ähnelt sich, sie folgt für Group- und ValueNodes immer diesem Prinzip: ValueNode macht eine Aktion und gibt danach einen Wert zurück. GroupNode macht eine Aktion und ruft danach für jedes seiner Kinder die gleiche Methode erneut auf (Rekursion). Aus den Rückgabewerten der Kinder wird ein Wert akkumuliert und dieser zurückgegeben. Die von einer GroupNode angestoßenen Rekursionen enden bei ValueNodes und GroupNodes ohne Kinderknoten, die Abbruchbedingung der Rekursion ist polymorph. Es ist kein Zufall, dass alle Methoden rekursiv sind, denn auch die Klassenstruktur ist rekursiv (GroupNodes können auf GroupNodes verweisen).

5.3 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Implementierungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Implementierung der Klassen	1 h	0,5	-0,5 h
2. Schreiben einer Methode zum Vergleichen von Nodes	0 h	2 h	+2 h
3. Erstellen von Tests	5 h	9 h	+4 h
4. Entwickeln einer Regular Expression zum Parsen von Schemazeilen	1 h	3 h	+2 h
5. Schreiben einer Methode zum Kopieren von Nodes	0 h	1 h	+1 h
6. Parsen der Schema-Datei in eine Baumstruktur	14 h	17 h	+3 h
7. Durchlaufen des Baumes und Zuweisen von Werten	12 h	2 h	-10 h
8. Ausgabe des Baumes als String	5 h	1 h	-4 h
9. Erstellen der Benutzeroberfläche	6 h	1 h	-5 h

Tabelle 6: Zwischenstand nach der Implementierungsphase

6 Entwurfsphase - Schema speichern und automatisch auswählen

6.1 Entwurf eines Bedienkonzeptes

Um die GUI übersichtlich zu halten, und weil in Zukunft eventuell noch Auswahlmöglichkeiten für andere Anwendungsfälle notwendig werden könnten, wurde entschieden, ein Menü zu erstellen. Bei Auswahl des Menüpunktes "Schema speichern" soll sich ein Dialog öffnen, in dem der Name des Schemas und ein Textfeld zur Angabe eines Datenstrom-Identifikationstextes⁵ eingegeben werden. Beim Klicken auf den Button "Schema speichern" wird ein neuer Eintrag in die XML-Datei vorgenommen und eine Textdatei mit dem Inhalt des Schema-Textfeldes im Programm-Unterordner "schemas" gespeichert. Neben der Überschrift des Schema-Textfeldes soll eine Combobox eingefügt werden, die die Dateien des schemas-Ordners anzeigt, und über die ein Schema ausgewählt werden kann.

6.2 Entwurf des XML-Schemas

Es gibt ein Microsoft-Tool namens xsd.exe, das aus XSD-Schemas C#-Klassen generieren kann. Um das auszunutzen wurde entschieden, die Einstellungen, die notwendigen Einstellungen als XML-Datei vorzunehmen. Das hatte außerdem den Vorteil, dass die Benutzer die Einstellungen direkt editieren können.

7 Implementierung - Schema speichern

Die Implementierung dieses Anwendungsfalls war eher Routine.

7.1 Erstellung des XSD-Datei

Es wurde eine xml-Datei (schemaConfig.xml) mit den benötigten Angaben und der gewünschten Struktur erstellt. Mit xsd.exe wurde daraus eine XSD-Datei erzeugt, und mit erneutem Aufruf von xsd.exe aus dieser die C#-Klasse SchemaConfig. Diese wird verwendet um die der Xml-Datei zu deserialisieren.

8 Abnahmephase

8.1 Abnahmetest

Der Abnahmetest erfolgte durch den Projektbetreuer Herrn Kemmer. Herr Kemmer ließ sich die erstellten Unit-Tests zeigen überprüfte, dass alle Tests erfolgreich waren. Danach ludt er die 5 am Häufigsten verwendeten Schemas zusammen mit passenden Datenströmen vom Mainframe. 1920Parser zeigte alle

⁵z. B. der Datenstrom enthält an Stelle 3 den Text abc

9 Einführungsphase

getesteten Schemas richtig an und zergliederte die Datenströme in gewünschter Weise. Im Anschluss testete Herr Kemmer die Funktionalität zum Speichern und zur automatischen Auswahl von Schemas. Nach Löschung der XML-Konfigurationsdatei trat bei der Auswahl des Menüpunktes "Config editieren" eine `FileNotFoundException` auf. Diese wurde behoben, danach testete Herr Kemmer noch einmal.

8.2 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Abnahmephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Abnahmetest	2 h	1,5 h	-0,5 h
2. Nachkorrektur und erneute Überprüfung	0 h	0,5 h	+0,5 h

Tabelle 7: Zwischenstand nach der Abnahmephase

9 Einführungsphase

Die Einführung von 1920Parser wurde von Herrn Kemmer übernommen. Diesem wurde per Email ein Link zum [SVN](#)-Repository, das gezippte Programm und die gezippten Dokumentationen zugeschickt. Er will das Programm im Intranet verfügbar zu machen.

10 Dokumentation

Für 1920Parser wurde eine Benutzerdokumentation und eine Entwicklerdokumentation geschrieben.

Beispiel Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang ???: ?? auf Seite ?. Die Entwicklerdokumentation wurde mit Doxygen generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang ???: ?? auf Seite ?.

10.1 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Dokumentation.

Vorgang	Geplant	Tatsächlich	Differenz
1. Erstellen der Benutzerdokumentation	1 h	1 h	
2. Erstellen der Projektdokumentation	7 h	7 h	
3. Erstellen der Entwicklerdokumentation	2 h	2 h	

Tabelle 8: Zwischenstand nach der Dokumentation

11 Fazit

11.1 Soll-/Ist-Vergleich

Wie gewünscht können im Programm Datenstrom und Schema frei angegeben werden. Das Ergebnis wird richtig angezeigt. Insbesondere das Zuweisen der Werte aus dem Datenstrom und die Ausgabe des Baumes als String nahmen deutlich weniger Zeit in Anspruch, als ursprünglich geplant. So konnte noch der Anwendungsfall “Schema speichern” im Rahmen des Projektes umgesetzt werden. Herr Kemmer ist mit dem Programm sehr zufrieden.

Phase	Geplant	Tatsächlich	Differenz
Entwurfsphase	19 h	19 h	
Analysephase	9 h	10 h	+1 h
Implementierungsphase	29 h	28 h	-1 h
Abnahmetest der Fachabteilung	1 h	1 h	
Einführungsphase	1 h	1 h	
Erstellen der Dokumentation	9 h	11 h	+2 h
Pufferzeit	2 h	0 h	-2 h
Gesamt	70 h	70 h	

Tabelle 9: Soll-/Ist-Vergleich

11.2 Lessons Learned

RichTextBox ist besser als TextBox. Interessant war, wie sich der Projektumfang erweitert hatte (ursprünglich war der Vorschlag, dass nur ein bestimmtes Schema zergliedert werden soll) Rekursion ist nicht gut darstellbar mit UML. Für Polymorphie gilt das selbe. Stacks sind super. Rekursion vereinfacht manche Aufgaben enorm. Bemerkenswert, wie sehr sich die Anforderungen ausgeweitet haben. Bemerkenswert war, dass sich der Projektumfang stark erhöhte (ursprünglich wurde mir der Projektvorschlag gemacht, nur die Schemadatei VK60 zu zergliedern).

11.3 Ausblick

Im Rahmen des Projektes wurden nicht alle Wunschkriterien erfüllt. Der Autor hofft aber, dass diese Aufgabe vielleicht an nachfolgende Auszubildende übergeben wird und sich 1920Parser noch weiterentwickelt. Vielleicht wird einmal die Größe der Datenströme geändert, dann könnte der Algorithmus von 1920Parser verwendet werden.

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	9 h
1. Analyse des Ist-Zustands	3 h
1.1. Fachgespräch mit der EDV-Abteilung	1 h
1.2. Prozessanalyse	2 h
2. „Make or buy“-Entscheidung und Wirtschaftlichkeitsanalyse	1 h
3. Erstellen eines „Use-Case“-Diagramms	2 h
4. Erstellen des Lastenhefts mit der EDV-Abteilung	3 h
Entwurfsphase	19 h
1. Prozessentwurf	2 h
2. Datenbankentwurf	3 h
2.1. ER-Modell erstellen	2 h
2.2. Konkretes Tabellenmodell erstellen	1 h
3. Erstellen von Datenverarbeitungskonzepten	4 h
3.1. Verarbeitung der CSV-Daten	1 h
3.2. Verarbeitung der SVN-Daten	1 h
3.3. Verarbeitung der Sourcen der Programme	2 h
4. Benutzeroberflächen entwerfen und abstimmen	2 h
5. Erstellen eines UML-Komponentendiagramms der Anwendung	4 h
6. Erstellen des Pflichtenhefts	4 h
Implementierungsphase	29 h
1. Anlegen der Datenbank	1 h
2. Umsetzung der HTML-Oberflächen und Stylesheets	4 h
3. Programmierung der PHP-Module für die Funktionen	23 h
3.1. Import der Modulinformationen aus CSV-Dateien	2 h
3.2. Parsen der Modulquelltexte	3 h
3.3. Import der SVN-Daten	2 h
3.4. Vergleichen zweier Umgebungen	4 h
3.5. Abrufen der von einem zu wählenden Benutzer geänderten Module	3 h
3.6. Erstellen einer Liste der Module unter unterschiedlichen Aspekten	5 h
3.7. Anzeigen einer Liste mit den Modulen und geparsen Metadaten	3 h
3.8. Erstellen einer Übersichtsseite für ein einzelnes Modul	1 h
4. Nächtlichen Batchjob einrichten	1 h
Abnahmetest der Fachabteilung	1 h
1. Abnahmetest der Fachabteilung	1 h
Einführungsphase	1 h
1. Einführung/Benutzerschulung	1 h
Erstellen der Dokumentation	9 h
1. Erstellen der Benutzerdokumentation	2 h
2. Erstellen der Projektdokumentation	6 h
3. Programmdokumentation	1 h
3.1. Generierung durch PHPdoc	1 h
Pufferzeit	2 h
1. Puffer	2 h
Gesamt	70 h

A.2 Beispiel für ein echtes 1920Schema

```
.fo off
.pa
+-----+-----+-----+
I          I          I          I
I          I          I          I
I P H A R M O S I      Copy-Book-Beschreibung I NACHFUELL I
I          I      Schnittstelle : Host --> PC I          I
I          I          I          I
I          I          I          I
+-----+-----+-----+
I P H A R M O S I      Froh zu sein bedarf es I Ausg.: 1 Wink I
I 8510 Fuerth 2 I      P H A R M O S I Vom :13.12.99I
I 0911-9300-695 I          I Wink :29:10.02I
+-----+-----+-----+
Dateiname: PH326591 OPUS

Satzlaenge:      Bytes
:h6.IOVK91 Schnittstelle Host->PC
.sp 2
.fo off
.tr * 40
.cm FELD-BESCHREIBUNG IOVK91
.pi /Bereich/IOVK91
.bx 2 9          27 31      38 42          &$11
  Stufe  Feldname      Typ Laenge Tab  Kommentar
.bx
****
03 IO91-AREA          Host --> PC
05 IO91-TRANID        C 4      Transaktionscode (VK91)
05 IO91-FUNKID        C 2      Funktionscode
05 IO91-EBENE         N 1      Lagerebene
***
05 IO91-DATEN-AREA    Struktur fuer Wannendaten
***
10 IO91-WANNE          4      Wannendaten
15 IO91-SA             C 1      SATZART 1 = VORAB, 2 = LA
15 IO91-FIL            N 2      Filiale
15 IO91-L1RRN          N 9      Re. Satznr. L1-Satz = LA-Nr
15 IO91-VZEIT          N 11     Vorlaufzeit in SEK
15 IO91-IPUNKT-WNR     N 7      Wannennr vom I-Punkt
15 IO91-WANREST        C 6      Filler
15 IO91-POSTAB         40     Tabelle der Positionen
20 IO91-KANAL          N 8      Kanalnummer
R20 IO91-KANAL-C       C 8      LAGERORT
20 IO91-MENGE          N 2      Menge
20 IO91-S-REST         C 1      Filler
10 IO91-REST           C 9      REST
***
.bx
R05 IO91-FEHLER-AREA   Struktur fuer Fehlermeldung
10 IO91-FEHLMELD       C 80     Meldungstext
R10 IO91-FEHLERMELDR   Struktur fuer Fehlermeldung
15 IO91-FEHLMELD2      C 20     Meldungstext
15 IO91-FEHLNR         N 5      L1-Nummer usw.
15 filler              C 1
15 IO91-FEHLNR2        N 5      L1-Nummer usw.
15 filler              C 49
10 IO91-REST2          C 1831   REST
***
.bx
R05 IO91-einzel         Struktur fuer Fehlermeldung
10 IO91-E-L1RRN        N 5      Taktnummer
10 IO91-E-stpl         N 2      Platznummer im takt
10 IO91-REST3          C 1906   REST
.bx
R05 IO91-STATION-AREA   STRUKTUR FUER FEHLERMELDUNG
10 IO91-STTAB          40     TABELLE DER STATIONEN
15 IO91-VSTATION       N 2      STATION VOR AUTOMAT
***
.bx
```

Abbildung 1: Beispiel für ein echtes 1920Schema

A.3 Use Case-Diagramm

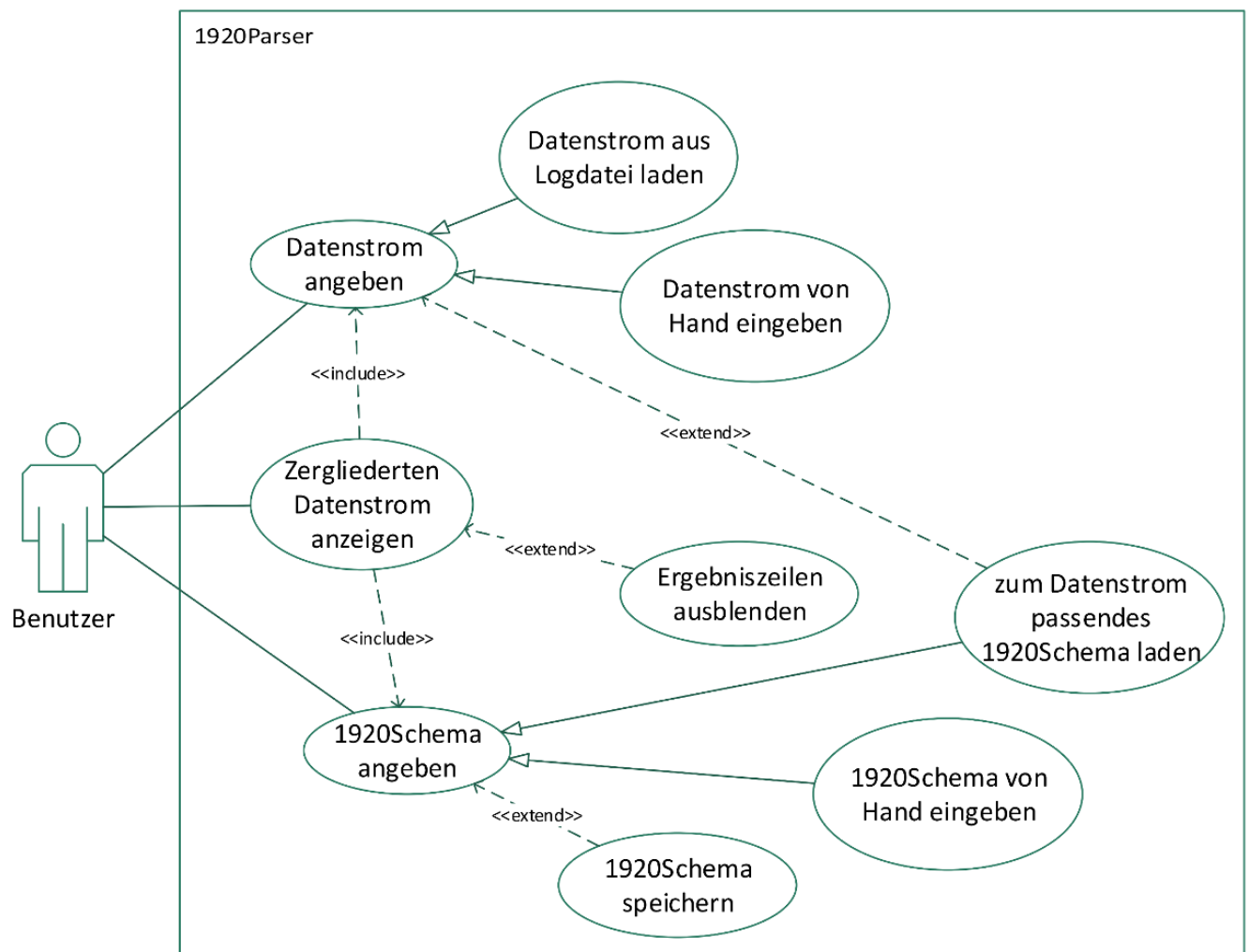


Abbildung 2: Use Case-Diagramm

A.4 Pflichtenheft (Auszug)

Die Anwendung muss folgende Anforderungen erfüllen:

1. Muss-Kriterien:

- 1.1. Datenstrom und Schema müssen frei angegeben werden können.
- 1.2. Das Programm muss den Datenstrom anhand des Schemas zergliedert anzeigen.

2. Soll-Kriterien:

- 2.1. Benutzer sollen ein eingegebenes Schemas speichern können.
- 2.2. Benutzer sollen gespeicherte Schemas laden können.

3. Kann-Kriterien:

- 3.1. Anhand des Transaktionscodes im Datenstrom soll das richtige Schema - falls vorhanden - automatisch ausgewählt werden.
- 3.2. Der Benutzer soll ein Kriterium angeben können, anhand dessen die passenden Datenströme aus der Logdatei ausgewählt werden.
- 3.3. Wenn das Länge-Feld eine Angabe zur Anzahl der Nachkommastellen hat, soll das Programm in der Ergebnis-Anzeige an der richtigen Stelle ein Komma einfügen.
- 3.4. Variablenzeilen sollen ausgeblendet werden können.

4. Ausschluss-Kriterien:

- 4.1. Das Programm soll Schema und Datenstrom nicht auf Korrektheit oder Plausibilität prüfen (z. B. doppelte Variablennamen, Redefines ohne Vorgänger, ungültige Stufennummern, Bytelänge 0, Char mit Nachkommastellen).

A.5 Klassendiagramm

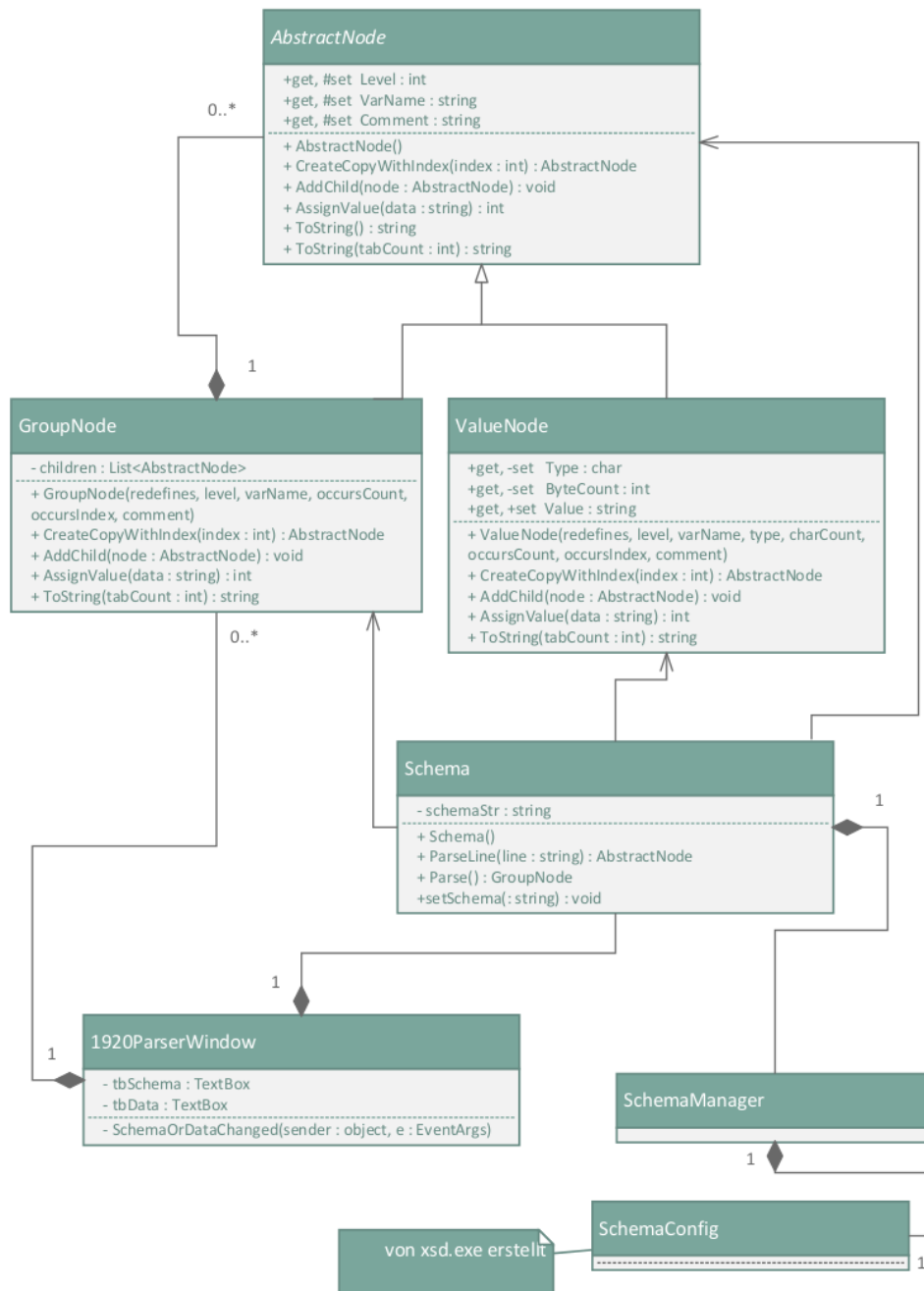


Abbildung 3: Klassendiagramm

A.6 Aktivitätsdiagramm - Parsen eines 1920Schemas

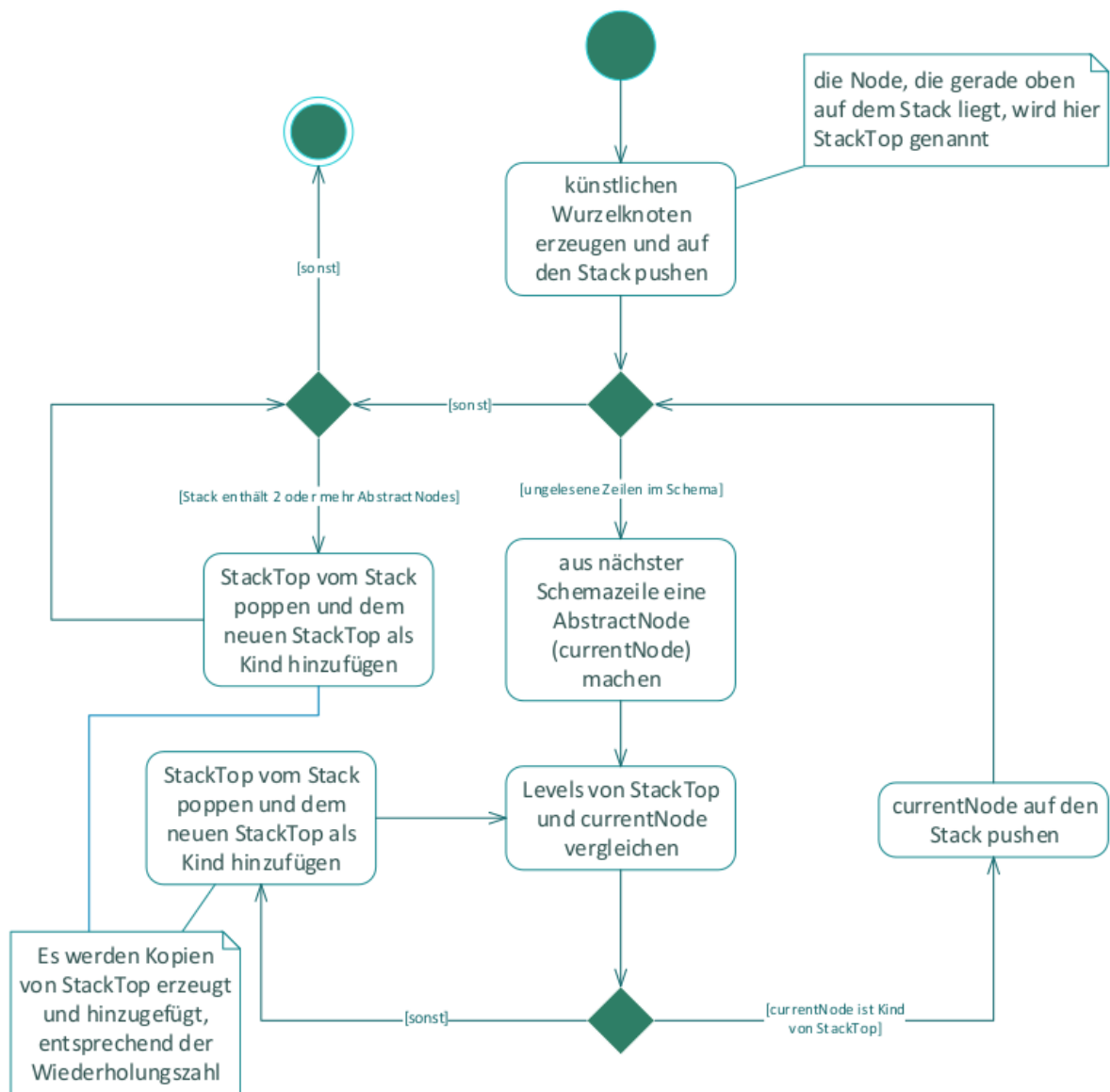


Abbildung 4: Aktivitätsdiagramm - Parsen eines 1920Schemas

A.7 Objektdiagramm - Parsen des Schemas

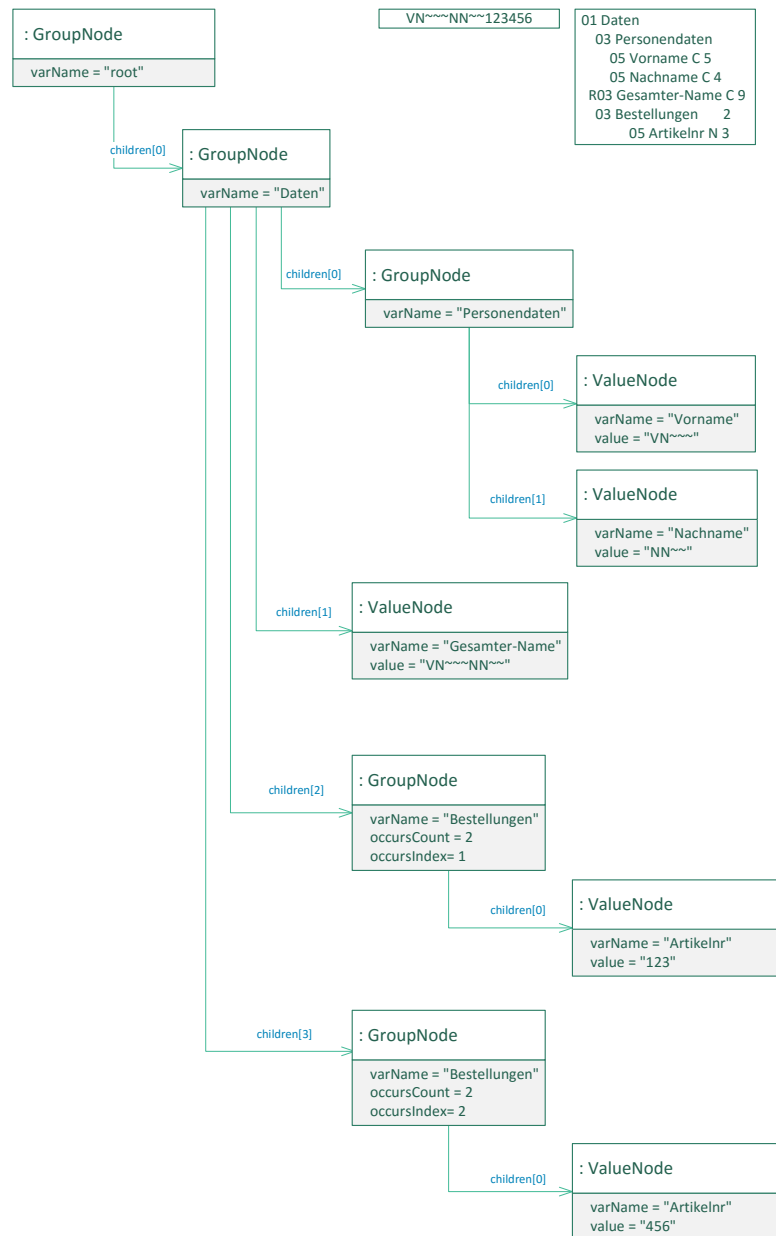


Abbildung 5: Diese Baumstruktur soll nach dem Parsen entstanden sein

A.8 Quelltext Schema.cs

```
1 class Schema
2 {
3     private string pattern = @"^\\s*(?<redefines>R?)(?<level>d+)\\s+(?<varName>P
4     \\S+)((\\s+(?<type>[xcnpXCNP]))\\s+((?<length>d+))\\s+(?<decimalPlaces>d
5     +))?(\\s+(?<repeatCount>d+))?(\\s{2,}(?<comment>.*))?$";
6     private string schemaStr = "";
7
8     public AbstractNode ParseLine(string line)
9     {
10         // gibt eine Value- oder GroupNode zurück, je nachdem ob line
11         // Angaben zu Typ und ByteAnzahl hat oder nicht
12         // wenn pattern nicht matcht wird null zurückgegeben
13     }
14
15     public GroupNode Parse()
16     {
17         var stack = new Stack<AbstractNode>();
18         var root = new GroupNode(false, 0, "root", 1, 1, "");
19         stack.Push(root);
20
21         Action addChildFromStackToParent =
22             () =>
23             {
24                 var child = stack.Pop();
25                 var parent = stack.Peek();
26                 parent.AddChild(child);
27                 for (int currentRepeatIndex = 2; currentRepeatIndex <=
28                     child.RepeatCount; ++currentRepeatIndex)
29                 {
30                     parent.AddChild(child.CreateCopyWithIndex
31                         (currentRepeatIndex));
32                 }
33             };
34
35         var schemaLines = schemaStr.Split(new string[] { "\\r\\n", "\\n" },
36             StringSplitOptions.RemoveEmptyEntries);
37         foreach (var currentLine in schemaLines)
38         {
39             var currentNode = ParseLine(currentLine);
40             if (currentNode == null) { continue; }
41             while (currentNode.Level <= stack.Peek().Level)
42             {
43                 addChildFromStackToParent();
44             }
45             stack.Push(currentNode);
46         }
47         while (stack.Count >= 2)
48         {
49             addChildFromStackToParent();
50         }
51         return root;
52     }
53 }
```

Abbildung 6: Schema.cs

A.9 Quelltext GroupNode.cs

```
1 class GroupNode : AbstractNode
2 {
3     private List<AbstractNode> children = new List<AbstractNode>();
4     public GroupNode(bool redefines, int level, string varName, int
5         repeatCount, int repeatIndex, string comment)
6         : base(redefines, level, varName, repeatCount, repeatIndex, comment)
7     { }
8     public override int AssignValue(string data)
9     {
10         int currentShift = 0;
11         int totalShift = 0;
12         foreach (var child in children)
13         {
14             if (!child.Redefines)
15             {
16                 currentShift = child.AssignValue(data.Substring(totalShift));
17                 totalShift += currentShift;
18             }
19             else
20             {
21                 child.AssignValue(data.Substring(totalShift - currentShift));
22             }
23         }
24         return totalShift;
25     }
26     public override string ToString(int tabCount)
27     {
28         StringBuilder strBuilder = new StringBuilder();
29         if (Level != 0)
30         {
31             strBuilder.Append(string.Format("{0}{1}{2} {3}{4}\r\n",
32                 new string(' ', tabCount * 4 - (Redefines ? 1 : 0)),
33                 Redefines ? "R" : "",
34                 Level.ToString().PadLeft(2, '0'),
35                 VarName,
36                 RepeatCount > 1 ? string.Format("({0})", RepeatIndex) : "");
37         }
38         foreach (var child in children)
39         {
40             strBuilder.Append(child.ToString(tabCount + (Level == 0 ? 0 :
41                 1)));
42         }
43         return strBuilder.ToString();
44     }
45     public override AbstractNode CreateCopyWithIndex(int index)
46     {
47         GroupNode g = new GroupNode(Redefines, Level, VarName, RepeatCount,
48             index, Comment);
49         foreach (var child in children)
50         {
51             g.AddChild(child.CreateCopyWithIndex(child.RepeatIndex));
52         }
53         return g;
54     }
55 }
```

Abbildung 7: GroupNode.cs

A.10 Quelltext ValueNode.cs

```
1 class ValueNode : AbstractNode
2 {
3     public string Type { get; private set; }
4     public int Length { get; private set; }
5     public string Value { get; private set; }
6
7     public ValueNode(bool redefines, int level, string varName, string type,
8         int length, int repeatCount, int repeatIndex, string comment)
9         : base(redefines, level, varName, repeatCount, repeatIndex, comment)
10    {
11        Type = type;
12        Length = length;
13    }
14
15    public override AbstractNode CreateCopyWithIndex(int index)
16    {
17        return new ValueNode(Redefines, Level, VarName, Type, Length,
18            RepeatCount, index, Comment);
19    }
20
21    public override int AssignValue(string data)
22    {
23        Value = data.Substring(0, Length);
24        return Length;
25    }
26
27    public override string ToString(int tabCount)
28    {
29        if (Level == 0)
30        {
31            return "";
32        }
33        return string.Format("{0}{1}{2} {3}{4}={5}\r\n",
34            new string(' ', tabCount * 4 - (Redefines ? 1 : 0)),
35            Redefines ? "R" : "",
36            Level.ToString().PadLeft(2, '0'),
37            VarName,
38            (RepeatCount > 1) ? "(" + RepeatIndex + ")" : "",
39            Value);
40    }
41
42    public override void AddChild(AbstractNode child)
43    {
44        throw new InvalidOperationException(this.VarName + ": " + "Werteknoten
45            können keine Kindknoten haben!");
46    }
47 }
```

Abbildung 8: ValueNode.cs

A.11 Entwicklerdokumentation

1920Parser

Main Page	Namespaces	Classes	
Class List	Class Index	Class Hierarchy	Class Members
_1920Parser	GroupNode		

[_1920Parser.GroupNode](#) Class Reference

[GroupNode](#) represents an internal node. [More...](#)

Inherits [_1920Parser.AbstractNode](#).

Public Member Functions

	GroupNode (bool redefines, int level, string varName, int repeatCount, int repeatIndex, string comment)
override int	AssignValue (string data) Assigns substrings of data to its children. The begin of a substring shifts with each assignment to a child. More...
override string	ToString (int tabCount) Returns a string representing this GroupNode and its children. More...
override AbstractNode	CreateCopyWithIndex (int index) Creates a deep copy of this GroupNode and sets its RepeatIndex to the value of the parameter. More...
override void	AddChild (AbstractNode child) Adds a child node. The child could be another GroupNode . More...
override int	GetHashCode () Returns the Hashcode of this class. This method is not supposed to be called directly, it was overwritten in conjunction with Equals() . More...
override bool	Equals (object obj) Compares two GroupNodes. GroupNodes are equal if all their attributes are equal and the attributes of all their children are equal. More...

► Public Member Functions inherited from [_1920Parser.AbstractNode](#)

Additional Inherited Members

► Properties inherited from [_1920Parser.AbstractNode](#)

Detailed Description

[GroupNode](#) represents an internal node.

Member Function Documentation

Abbildung 9: Entwicklerdokumentation