# Konzeptionierung und Implementierung eines statischen Codeanalysewerkzeugs zur Überprüfung von Programmierrichtlinien in SCL

Ole Gramit (ole.gramit@fh-bielefeld.de, ole.gramit@dmgmori.com)
Georgios Lajios (georgios.lajios@fh-bielefeld.de)

## Zusammenfassung

Im Rahmen eines Standardisierungsprojekts ist ein Werkzeug entstanden, welches SCL-Quellcode auf die Einhaltung von Programmierkonventionen überprüft, der *DMG MORI SCL Analyser*. Die Programmiersprache SCL (Structured Control Language) der Firma SIEMENS wird für die Programmierung von Maschinen und Anlagensteuerungen eingesetzt. Auf Basis einer SCL-Grammatik wurde mit dem Parsergenerator ANTLR ein SCL-Parser erzeugt. Dieser Parser dient zum Erzeugen einer Datenstruktur, in der nach Verstößen gegen Programmierrichtlinien gesucht wird.

## 1 Motivation

Bei der Weiterentwicklung und Pflege von Quellcode liegt das Verhältnis der zum Lesen von Code verwendeten Zeit zu der zum Schreiben verwendeten Zeit über zehn zu eins [1]. Quellcode auf Lesbarkeit zu optimieren bedeutet daher auch, Quellcode für das Schreiben zu optimieren. Aus diesem Grund werden in großen Softwareprojekten Programmierkonventionen verwendet, die Fehler begünstigende Konstrukte verhindern, die Portabilität der Software verbessern oder die Verständlichkeit des Quellcodes erhöhen sollen [2].

DMG MORI als Werkzeugmaschinenhersteller im Bereich der spanenden Fertigung produziert CNC (Computerized Numerical Control)-gesteuerte Maschinen, die in der SINUMERIK-basierten Variante eine speicherprogrammierbare Steuerung (Programmable Logic Controller, PLC) der Firma SIEMENS enthalten. Software für diese Steuerungen wird in Programmiersprachen der internationalen Norm IEC61131-3 erstellt. Diese Norm wurde eingeführt, um den wachsenden Anforderungen an speicherprogrammierbare Steuerungen zu begegnen [3]. Sie definiert das grundlegende Softwaremodell und die Programmiersprachen, in der eine SPS programmiert werden kann.

Im Rahmen eines Standardisierungsprojektes untersucht DMG MORI Wege zur vereinheitlichten Prüfung der Einhaltung von internen Programmierkonventionen für die Entwicklung von PLC-Projekten. Zur Pilotierung möglicher Ansätze wurde ein Werkzeug konzipiert und implementiert.

## 2 SCL-Parser

Der SCL-Code liegt in modular aufgebauten Projekten vor, die aus Organisationsbausteinen, Funktionsbausteinen, Funktionen, Datenbausteinen und anwenderdefinierten Datentypen bestehen. Der von ANTLR generierte Parser erstellt einen *Parse Tree*, auf dessen Basis nach Verstößen gegen die Programmierrichtlinien gesucht werden kann. Es wird kein AST (Abstract Syntax Tree) verwendet, da bei der Erzeugung eines solchen Informationen verloren gehen. Für eine Weiterentwicklung des SCL Analysers soll das zusätzliche Generieren eines AST untersucht werden.

Im Parse Tree entspricht jeder Knoten einer Regel aus der für dieses Projekt entwickelten SCL-Grammatik. Angefangen bei dem Wurzelknoten, der der kompletten SCL-Datei entspricht und einen oder mehrere der oben genannten Bausteine als Kinderknoten hat, ist jedes Symbol aus der geparsten SCL-Datei in dem Baum einsortiert. In Abbildung 1 ist ein Ausschnitt aus einem Parse Tree dargestellt, der die Variablenvereinbarung „xsVar : BOOL;“ abbildet.

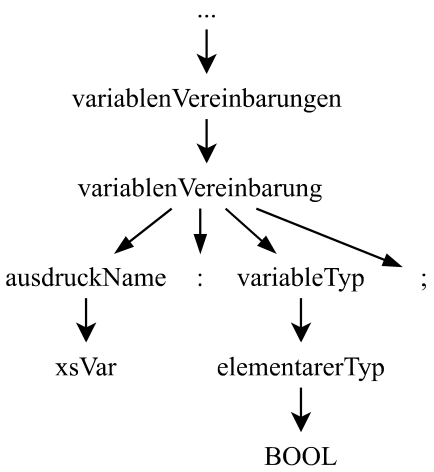


Abbildung 1: Parse Tree zur Variablenvereinbarung „xsVar : BOOL;“

Die SCL-Grammatik ist in Lexer- und Parserregeln eingeteilt. Lexerregeln organisieren die Erkennung von Symbolen in der eingelesenen Zeichenkette, bevor der Parser diese Symbole den entsprechenden Parserregeln zuordnet und den Parse Tree erstellt. Die Grammatik wurde auf der Basis des Projekts [4] und der SCL-Dokumentation von SIEMENS [5] erarbeitet. Ausgehend von den abstrakten Sprachkonstrukten, wie z.B. Organisationsbausteinen, wurden alle spezifischen Konstrukte aus der SCL-Dokumentation abgebildet. Anschließend wurde die Grammatik ausgiebig an vorhandenen SCL-Projekten getestet.

Die Grammatikbausteine für das Erstellen des Baums aus Abbildung 1 sind in Abbildung 2 dargestellt. Ab Zeile neun sind Lexerregeln abgebildet. Lexerregeln regeln das Erkennen von Symbolen in der eingelesenen Zeichenkette, bevor der Parser diese Symbole den entsprechenden Parserregeln zuordnet und den Parse Tree erstellt.

```
1 //Parser Regeln
2 variablenVereinbarungen:
  (variablenVereinbarung ,;‘)* ;
3 variablenVereinbarung:
  ausdruckName ,:‘ variablenTyp
  ,;‘ ;
4 ausdruckName:
  BEZEICHNER ;
5 variablenTyp:
  elementarerTyp |
  anwenderdefinierterTyp;
6 elementarerTyp:
  BOOL |
  INT ;
7 anwenderdefinierterTyp:
  BEZEICHNER ;
8 //Lexer Regeln
9 BOOL:
(’B’|’b’)(’O’|’o’)(’O’|’o’)(’L’|’l’);
10 INT:
  (’I’|’i’)(’N’|’n’)(’T’|’t’)
11 BEZEICHNER:
  [a-zA-Z$_][a-zA-ZA0-9$_]* ;
12 LEERZEICHEN:
  , ‘ -> skip ;
```

Abbildung 2: Vereinfachter Auszug aus der SCL-Grammatik. Es ist zu erkennen, wie die Definition von Regeln an Regular Expressions angelehnt ist und die Parserregeln die Lexerregeln Gruppieren. Parserregeln sind kleingeschrieben und Lexerregeln groß.

## 3 Konzept

Der DMG MORI SCL Analyser bietet dem Benutzer die Möglichkeit, eine SCL-Datei einzulesen und diese auf Verstöße gegen festgelegte Programmierrichtlinien zu prüfen. Die Ergebnisse werden dem Benutzer ausgegeben. Für das Überprüfen der geparsten SCL-Datei wird ein Traversieren ausgelöst, welches jeden Knoten des Parse Trees besucht. Besucht dieses Haupttraversieren einen Knoten, in dem eine Programmierrichtlinie überprüft werden soll, wird der Richtlinie ein Parse Tree mit dem aktuellen Knoten als Wurzel übergeben. Dieser Baum wird dann durch die Richtlinienklasse traversiert und durch eine richtlinienspezifische Logik auf Verstöße geprüft.

Als Beispiel sei eine Programmierrichtlinie genannt, die überprüft, ob der Präfix eines Variablennamens zu dem angegebenen Datentyp passt. Diese Richtlinie wird durch das Haupttraversieren im Knoten „variablenVereinbarung“ (Abbildung 1) aufgerufen. Die Richtlinie erhält diesen Knoten dann als Wurzelknoten, von dem aus ein weiteres Traversieren ausgelöst wird. Werden ein Variablenname und ein Variablentyp gefunden, werden diese abgeglichen und gegebenenfalls ein Verstoß ausgegeben.

## 4 Fazit

Da der DMG MORI SCL Analyser im Rahmen eines Pilotprojekts entstanden ist lag der Fokus zunächst darauf, eine Anwendung zu entwickeln, die SCL-Dateien einlesen, parsen und beispielhafte Programmierrichtlinien überprüfen kann. Die entwickelte Anwendung kann als Grundlage für eine in die Entwicklungsumgebung integrierte statische Codeanalyseanwendung verwendet werden, da während der Entwicklung auf die Modularität und Wiederverwendbarkeit der Software geachtet wurde.

## Referenzen


[1] R. C. Martin et al.: *Clean code: A handbook of agile software craftsmanship.* Upper Saddle River, NJ, Prentice Hall, 2009.

[2] P. Liggesmeyer, Software-Qualität: *Testen, Analysieren und Verifizieren von Software*, 2. Auflage, Spektrum Akademischer Verlag, 2009.

[3] M. Tiegelkamp and K. H. John.: *SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen*, 4. Auflage, Springer-Verlag, 2009.

[4] GitHub Benutzer jc211: *Siemens SCL Antlr-Grammar.* [Link:] https://github.com/jc211/Siemens-SCL-Antlr-Grammar (abgerufen: 30. April 2020).

[5] SIEMENS AG: *S7-SCL V5.3 for S7-300/400.* [Link:] https://support.industry.siemens.com/cs/document/5581793/s7-scl-v5-3-f%C3%BCr-s7-300-400 (abgerufen: 30. April 2020).