Validierung des Bereichsdatencompilers für die Linienzugbeeinflussung LZB L72 CE mit Hilfe eines diversitären Ansatzes

Bernd Holzmüller

Informatik Consulting Systems AG
Sonnenbergstraße 13, 70184 Stuttgart
Bernd.Holzmueller@ics-ag.de

1 Zusammenfassung

Bei der Linienzugbeeinflussung (LZB) sind die festen Streckendaten eines LZB-Steuerbereichs (Gleistopologie mit Weichen- und Signalstandorten, Langsamfahrstellen, Neigungen, usw.) in der anlagenspezifischen LZB-Software gespeichert. Für die Zentralen LZB L72 CE werden in textueller, lesbarer Form sog. Bereichsdaten projektiert. Diese Bereichsdaten werden durch den Bereichsdatencompiler (BDC) in mehrere Assemblerdateien und Benutzerlisten transformiert. Vor der Transformation prüft der BDC die Eingabedaten auf syntaktische Korrektheit und Konsistenz. Die generierten Assemblerdateien werden schließlich mit der generischen LZB-Systemsoftware zu einem streckenspezifischen Steuerprogramm zusammengebunden.

Um die sicherheitsrelevante Transformation der Bereichsdaten durch den Bereichsdatencompiler gegen die Vorgaben der Projektierungsrichtlinien und der Anforderungen zu validieren, wurde ein diversitärer Ansatz mit vollständiger Automatisierung entwickelt und erfolgreich umgesetzt. Der Erfolg des diversitären Ansatzes beruht stark auf der Wahl der funktionalen Programmiersprache Haskell zur Entwicklung des diversitären Compilers, die einerseits eine sehr schnelle und damit kostengünstige Realisierung ermöglichte und andererseits einen technologischen Gegenpol zur Implementierung des originären Systems in C in Bezug auf mögliche Fehlerklassen darstellt.

2 Problematik und Lösungsansatz

Eine manuelle Prüfung der BDC-Ausgabedateien wäre wegen der sehr kompakten, bitweisen Codierung der Assembler-Dateien äußerst mühsam, zeitaufwändig und fehleranfällig. Daher wurde ein diversitärer Ansatz erwogen, bei der ein zweiter Bereichsdatencompiler entwickelt werden sollte, der dieselbe Transformation durchführt, um anschließend die Ausgaben beider Compiler automatisiert vergleichen zu können. Dabei müssen allerdings die folgenden Voraussetzungen beachtet werden:

1. Die Entwicklung des diversitären Compilers darf ausschließlich auf Basis der Anforderungen erfolgen.

2. Technologisch bedingte gemeinsame Fehler sollen weitestgehend ausgeschlossen werden können.

3. Die Entwicklung und Anwendung des diversitären Compilers muss gegenüber einer manuellen Prüfung aus wirtschaftlichen Gesichtspunkten vertretbar sein.

Mit dem so erhaltenen diversitären Compiler lassen sich dann sehr viele Tests automatisiert und damit schnell und kostengünstig durchführen, was insbesondere auch bei kleineren Änderungen des BDC vorteilhaft ist, da vollständige Regressionstests mit wenig Aufwand durchgeführt werden können.

3 Haskell als Implementierungssprache

Um die Voraussetzungen 2. und 3. abzudecken wurde zur Realisierung des diversitären Compilers die funktionale Programmiersprache Haskell [2] gewählt. Haskell eignet sich gemäß den Ergebnissen einer Navy-Studie [3] sowie den persönlichen Erfahrungen des Autors hervorragend dazu, funktionsfähige, stabile Software-Werkzeuge mit sehr wenig Aufwand zu erstellen und zu warten. Dies wurde anhand einer Machbarkeitsstudie in Bezug auf den diversitären Bereichsdatencompiler verifiziert: Innerhalb von 2 Wochen entstand ein funktional fast vollständiger Prototyp, der später in kurzer Zeit zu einem vollständigen Werkzeug weiterentwickelt werden konnte.

Im Gegensatz zur Implementierungssprache des ersten Compilers, C, gehört Haskell einer völlig anderen Sprachklasse an, die i.d.R. eine völlig unterschiedliche Denkweise und Algorithmen erfordert. Die strenge Typisierung von Haskell und die übrigen spezifischen Spracheigenschaften schließen eine sehr große Menge typischer Fehlerklassen von C (Speicherzugriffsfehler, typbezogene Fehler, Überläufe, Indexfehler etc., vgl. [4], [5]) von vornherein aus oder machen diese sehr unwahrscheinlich. Gründe hierfür sind u.a.:

- Speicherfehler und Seiteneffekte globaler Variablen sind in Haskell a priori aufgrund der Sprachklasse ausgeschlossen (sofern nicht die primitive Funktion „unsafePerformIO“ verwendet wird).
- Typfehler (z.B. als Folge fehlender Typprüfung) werden in Haskell durch das polymorphe Typsystem samt strenger Typprüfung (auch über Modulgrenzen hinweg) ebenfalls ausgeschlossen.
- Überlaufs- und Rundungsfehler lassen sich durch Verwendung des unbeschränkten Ganzzahltyps Integer und der Verwendung von (exakten) Brüchen (Typ Ratio) für rationale Zahlen vermeiden. Rundungsfehler durch implizite Konversionen sind in Haskell unmöglich.
- Indexfehler sind in Haskell sehr selten, da i.d.R. Listen „als Ganzes“ und mit Hilfe der Standard-Listenoperationen manipuliert werden.

4 Ergebnisse

Obwohl die Laufzeit für die Entwicklung bzw. Weiterentwicklung des BDC bereits mehrere Personenjahre beträgt, konnte der diversitäre Compiler in weniger als vier Wochen vollständig realisiert werden, wobei die Codegröße des diversitären Compilers von ca. 100 KB (2,1 KLOC ohne Leer- und Kommentarzeilen) ein Bruchteil der BDC-Codegröße beträgt. Zwischenzeitliche funktionale Änderungen am BDC konnten stets in wenigen Tagen nachgezogen werden, wodurch Re-Validierungen mit sehr wenig Aufwand möglich waren.

Durch die initiale Validierung konnten 14 Abweichungen des BDC von den Anforderungen sowie 11 Probleme (9 Lücken/Ungenauigkeiten, 2 Inkonsistenzen) in den Anforderungen selbst entdeckt werden. Die Verwendung der funktionalen Programmiersprache Haskell hat dabei wesentlich zum Erfolg des Konzeptes beigetragen.

5 Referenzen

[1] Holzmüller et al. *Validierung des Bereichsdatencompilers (BDC) für die LZB L72 CE*. In Signal+Draht, März 2006.

[2] Simon Peyton Jones et al. *Report on the Programming Language Haskell 98*, Feb. 1999. Verfügbar unter http://www.haskell.org

[3] Paul Hudak and Mark P. Jones. *Haskell vs. Ada vs. C++ vs. Awk vs. ..., An Experiment in Software Prototyping Productivity*. Juli 1994. http://haskell.org/papers/NSWC/jfp.ps

[4] MISRA. *Guidelines For The Use Of The C Language In Vehicle Based Software*. April 1998.

[5] Marc Eisenstadt. *My Hairiest Bug War Stories*, CACM 40(4):30-37, April 1997. http://cpe.njit.edu/dlnotes/CIS/CIS455_CIS679/MyHairest.pdf