Schwachpunkte und Grenzen gegenwärtiger Methoden

Theodor Tempelmeier

Fakultät für Informatik
Hochschule Rosenheim
Hochschulstraße 1
83024 Rosenheim
tempelmeier@fh-rosenheim.de

Abstract: In der Entwicklung eingebetteter, sicherheitskritischer Systeme werden vor allem in der industriellen Praxis zum Teil Methoden eingesetzt, die nach Meinung des Autors Schwachpunkte aufweisen. Ziel dieses Beitrags ist die Überwindung dieser Schwachpunkte, oder wenigstens eine Schärfung des Bewusstseins bezüglich dieser Schwachpunkte, um eine Verbesserung der Situation zu erreichen.

1 Gegenwärtige Situation

Aktuell ist die Entwicklungs- und Nachweissituation computergestützter sicherheitskritischer Systeme stark geprägt durch Ansätze wie die modellgetriebene Entwicklung und die automatische Code-Generierung sowie durch kontinuierliche Verbesserungen im Bereich der mathematischen, auch automatischen, Korrektheitsbeweise. Selbstverständlich sind diese Ideen, ohne dass weitere Argumente nötig wären, auf das entschiedenste zu bejahen. Aber dennoch zeigt die industrielle Praxis Schwachpunkte, deren Überwindung relativ kurzfristig eine Verbesserung der Situation ermöglichen könnte. Und bezüglich der mathematischen Korrektheitsbeweise sind deren inhärente Grenzen zu beachten.

2 Modellgetriebene Entwicklung und automatische Code-Erzeugung

Sofern aus Modellen sicherheitsrelevanter Code generiert werden soll, ist für die Modellierungssprache eine *klar definierte Semantik* zu fordern; dies ist bedauerlicherweise nicht immer gegeben. Zur automatischen Code-Erzeugung muss deutlich gemacht werden, dass der Grundsatz „Von nichts kommt nichts“ auch hier gilt. Im Grunde wird nichts generiert, sondern nur der (grafische) Ausgangscode in Zielcode transformiert. Dabei fließen die womöglich unzureichende Semantik des Modells sowie eine Vielzahl weiterer Informationen ein. Diese *Transformation* muss deshalb ebenfalls *präzise definiert* sein. Es erscheint abwegig, wenn eine Reihe von Modellierungselementen vermieden werden muss, damit korrekter Zielcode entsteht [Te07], oder wenn die Code-Erzeugung durch ca. 50 Schalter (entsprechend 2^{50} Möglichkeiten) gesteuert werden kann [Te07]. Schließlich ist noch die Modellierung selbst korrekt anzuwenden. Die Mo-

delle müssen einen höheren *Abstraktionsgrad* als die Implementierung aufweisen, ansonsten wäre nichts gewonnen. Beispiele für eine *Modellierungsinversion* (d.h. das Modell ist auf demselben oder einem schlechteren Abstraktionsniveau als die Implementierung) sind Oder-Gatter im grafischen Modell oder auch UML-Packages für Zielcode mit einem höherwertigen Paketkonzept wie z.B. in der Programmiersprache Ada.

3 Programmiersprachen

Trotz der vorherrschenden Codeerzeugungseuphorie soll das Thema Programmiersprache nicht unbeachtet bleiben. Die Sprachen *C und C++* können für sicherheitskritische Anwendungen nur als *hoffnungslos ungeeignet* eingestuft werden. Die geringfügigen Verbesserungen in C++ (gelegentlich als „objektorientierter Assembler" verspottet) reichen nicht aus. *Java* ist aufgrund seiner dynamischen Laufzeitaspekte *ebenfalls ungeeignet*. Man kann wohl Java, vielleicht auch C++, so weit abändern, dass sie für sicherheitskritische Anwendungen geeignet werden. Damit würde aber nur eine längst geleistete Arbeit nachvollzogen. Denn die Programmiersprache *Ada* ist für sicherheitskritische Anwendungen von vornherein *hervorragend geeignet*, auch wenn sie außerhalb der Luft- und Raumfahrtindustrie weitgehend ignoriert wird. Die Stärken von Ada sind ein überlegener Synchronisationsmechanismus (CSP-ähnlich vs. Monitor-ähnlich in Java), ein überlegenes Package-Konzept (wichtig für „teile und herrsche"), das sog. Ravenscar-Profil für sicheres Tasking, integrierte Festpunkttypen sowie eine strenge Typisierung (wie in der IEC 61508 vorgegeben). Gerade vor dem letztgenannten Punkt ist der weitverbreitete Einsatz von C als erschreckend und beunruhigend abzulehnen.

4 Korrektheitsbeweise

Strenge Korrektheitsbeweise von Programmen sind als zusätzliche Absicherung absolut zu bejahen. Man muss jedoch auch deren Grenzen erkennen. *Die Beweise gelten nur in Bezug auf die getroffenen Annahmen.* Als Beispiel könnten Veränderungen in der Umwelt, z.B. verschüttete Schmierseife, den Bremsweg eines autonomen Roboters vergrößern, so dass die „bewiesene" Systemsicherheit gefährdet wäre. Oder eine Flugzeugsteuerung könnte nach der Landung den Umkehrschub verweigern, weil sich die Räder des Fahrwerks wegen Aquaplaning nicht drehen; der Korrektheitsbeweis unter der falschen Annahme „Räder drehen sich nicht → kein Bodenkontakt" ist dann nutzlos. Abhilfe bringt hier nur eine extreme Sorgfalt in der Berücksichtigung und Beurteilung aller Voraussetzungen und Annahmen des Gesamtsystems. Leider hat nämlich *Albert Einstein* recht: *„Insofern sich die Sätze der Mathematik auf die Wirklichkeit beziehen, sind sie nicht sicher, und insofern sie sicher sind, beziehen sie sich nicht auf die Wirklichkeit."*

Literaturverzeichnis

[Te07] Tempelmeier, T.: Untersuchung verschiedener gängiger Modellierungswerkzeuge und Modellierungsmethoden, Interner Bericht, Hochschule Rosenheim, 2007.