Anforderungen auf Konsistenz überprüft - Formalisierung hilft

Dr. Marc Segelken
MathWorks GmbH, Germany
EMail: marc.segelken@mathworks.de

Abstract: Im Requirements Engineering sind Fragestellungen bzgl. der Eindeutigkeit, Widerspruchsfreiheit und Vollständigkeit Herausforderungen, die in herkömmlichen, manuell textbasierten Vorgehensweisen nur schwer zu beherrschen sind. Dieser Beitrag beschreibt für reaktive Systeme und Anforderungen, wie sie typischerweise in eingebetteten Systemen vorkommen, den Anwendungsfall der automatischen Modell-unabhängigen Überprüfung von Mengen von Requirements hinsichtlich Widerspruchsfreiheit und Vollständigkeit durch Verwendung von in graphischer Darstellung repräsentierter Anforderungen durch formale Methoden.

Das Requirements Engineering in der Spezifikationsphase ist ein wesentlicher Bestandteil eines Software Entwicklungsprozesses, der zumeist textuell oder semi-formal umgesetzt wird, unterstützt durch Requirements-Management Werkzeuge. Auf dieser Basis werden die erforderlichen wiederholten manuellen Reviews durchgeführt, welche Korrektheit, Eindeutigkeit, Widerspruchsfreiheit und Vollständigkeit der Anforderungen gewährleisten sollen. Gerade bei umfangreichen Anforderungskatalogen ist ein solcher Reviewprozess trotz signifikantem Aufwands jedoch überfordert, die genannten Zielstellungen umfassend zu erfüllen.

Abhilfe können hier automatisierbare Analyseverfahren auf Basis vollständig formalisierter Requirements schaffen und manuelle Reviews somit ersetzen. Die notwenigerweise mathematisch präzise Darstellung erzwingt Eindeutigkeit der Anforderungen und erlaubt die klassischen Verifikationsanwendungen zur Testüberwachung und die formale Verifikation selbiger, so dass sich der Mehraufwand für die Formalisierung der Anforderungen bereits hierfür schnell amortisiert. Insbesondere im Umfeld sicherheitskritischer eingebetteter Systeme, welche hier betrachtet werden sollen, sind diese Verfahren zur Absicherung daher weiter auf dem Vormarsch.

Noch weitgehend unbeachtet sind jedoch weitere Anwendungsfälle formalisierter Anforderungen zur automatischen Analyse von Widerspruchsfreiheit von Anforderungen untereinander, sowie der Überprüfung der Vollständigkeit der Spezifikation bzgl. einer praxistauglichen Definition der Vollständigkeit. Diese Analysen sind bereits nach Fertigstellung der Anforderungen möglich und völlig unabhängig von einer Implementierung durchführbar. Hierdurch können neben dem gezielten Aufdecken von Inkonsistenzen und Lücken in der Spezifikation unter dem Einsatz formaler Methoden sogar Konsistenz und Vollständigkeit bewiesen und damit im Entwicklungsprozess diese frühe wichtige Phase abgeschlossen werden ohne in späteren Iterationen teure Korrekturen und Ergänzungen vornehmen zu müssen.

Im Bereich reaktiver Systeme aus dem Embedded Bereich werden in der industriellen Praxis derzeit LTL-basierte Model Checker eingesetzt, bei denen das Requirement letztlich durch einen Observer-Automat repräsentiert ist. Mögliche Umsetzungen und Anwendung entsprechender Beweisdurchführungen zur Konsistenz und Vollständigkeit sind für diesen Bereich noch weitgehend unbekannt. In diesem Beitrag wird gezeigt, wie durch ein graphisches Entwicklungswerkzeug wie Simulink Beweise formalisiert und für automatische Konsistenz- und Vollständigkeitsanalysen genutzt werden können.

Die Formalisierung der Requirements wird graphisch unter Zuhilfenahme von Operatoren zur Spezifikation von temporalen Randbedingungen typischerweise in Form von logischen Wenn-Dann-Aussagen umgesetzt. Für jedes Requirement ensteht so ein einfaches Simulink-Subsystem, welches die zeitlichen Verläufe relevanter Größen der jeweiligen Anforderung beobachtet und bei Verletzung der Anforderung einen Fehler anzeigt. Genau diese werden bereits für anforderungsbasierte Testfallgenerierung und insbesondere für das Beweisen oder Widerlegen der Einhaltung einer Anforderung durch Model Checker verwendet.

Für Konsistenz- und Vollständigkeitsbeweise müssen die formalisierte Anforderungen logisch gruppiert und verbunden werden. Für einen Konsistenz-Beweis werden die Anforderungen hierzu für jedes (Ausgabe-)Signal zusammengefasst, welches sie betreffen, d.h. die zugehörigen Requirements fordern oder verbieten zu bestimmten Zeitpunkten bestimmte Wertverläufe. Durch geeignete logische Kombination wird hieraus ein neues Modell aufgebaut, an dem mögliche Widersprüche durch einfache Erreichbarkeitsanalyse mit Hilfe eines Model-Checkers durchgeführt werden können. Ergebnis hiervon ist die gezielte Ermittlung von Situationen, in denen sich mindestens zwei Requirements bzgl. des von dem später zu erstellendem Systems widersprechen, d.h. die Software-Komponente lässt sich ohne Änderungen wie z.B. Priorisierungen von Anforderungen nicht realisieren. Eine solche Situation kann gezielt aufgedeckt bzw., falls diese Situation nicht eintreten kann, ausgeschlossen werden.

Neben der Widerspruchsfreiheit stellt sich die Frage bzgl. der Vollständigkeit der Anforderungen. Hier ist zunächst eine sinnvolle praxisrelevante Definition von Vollständigkeit zu treffen, da Anforderungen typischerweise einen Spielraum im Wert- und Zeitbereich beinhalten. Hernach kann darauf aufbauend erneut das System hinsichtlich Verletzungen einer solchen Definition durch eine einfache logische Kombination der formalisierten Anforderungen auf Vollständigkeit untersucht werden, wobei ebenfalls automatische Erreichbarkeitsanalysen eingesetzt werden. Als Ergebnis der Analyse werden somit potentielle Unterspezifikationen aufgedeckt, welche durch Präzisierung oder Ergänzung der Requirements auszuschließen wären.

Der Vorteil dieser Vorgehensweise liegt in der frühen Anwendbarkeit bei der Entwicklung eines Software-Systems oder einer Software-Komponente: Lediglich die Schnittstelle muss bekannt sein – es wird keine Implementierung der Komponente benötigt. Beide Analysen lassen sich allein auf Basis der vorliegenden Anforderungen durchführen. Somit kann Widerspruchsfreiheit und Vollständigkeit von Anforderungen unabhängig von einem noch zu entwickelnden Modell gewährleistet werden. Zeitaufwändige Iterationen mit späten Korrekturen der Anforderungen in der Implementierungs- oder Testphase werden somit vermieden.