Automatische Validierung von Anforderungen

Amalinda Post[1], Andreas Podelski[2]
[1]Robert Bosch GmbH, Stuttgart, Germany
Amalinda.Oertel@de.bosch.com
[2]Universität Freiburg, Institut für Software Engineering,
podelski@informatik.uni-freiburg.de

Motivation

Anforderungen ermöglichen es, ein korrektes System von einem fehlerhaften System zu unterscheiden. Wie aber können wir feststellen, ob die Anforderungen selbst korrekt sind?

Oft werden Anforderungen zur Validierung manuell gereviewt. In diesem Vortrag stellen wir eine automatische Validierungs-Methode vor, welche Verhaltensanforderungen auf drei Eigenschaften überprüft.

Wir haben die Methode prototypisch implementiert und in einer Case Study über 6 BOSCH-Projekte aus dem Automobilbereich angewendet. Die Ergebnisse der Case Study zeigen, dass die Methode Fehler in Anforderungen aufzeigt und einen Requirements Engineer bei deren Behebung unterstützt.

Spezifikationssprache

Um eine automatische Validierung zu ermöglichen verwenden wir zur Anforderungsformalisierung eine eingeschränkte englische Grammatik, definiert von Konrad und Cheng [1], als Spezifikationssprache. Die so formalisierten Anforderungen werden automatisch in Logik übersetzt und von einem Computer validiert.

Die Spezifikationssprache wirkt wie „normales“ Englisch, und ist daher für Stakeholder intuitiv verstehbar und verwendbar. Wie in Bild 1 dargestellt kann sich der Stakeholder aufgrund der automatischen Übersetzung auf die Anforderungen in Spezifikationssprache beschränken. Die semantisch äquivalente Übersetzung in Logik ist nur für den Computer relevant.

Eigenschaften, gegen die validiert wird

Im Rahmen dieser Arbeit prüfen wir Anforderungen gegen drei Eigenschaften: *Inkonsistenz, triviale Konsistenz* und *rt-Inkonsistenz*.

Eine Menge von Anforderungen ist *inkonsistent*, wenn es kein System gibt, welches alle Anforderungen erfüllt. Inkonsistent sind beispielsweise die folgenden zwei Anforderungen „It is always the case that *IRTest* holds“ und „It is never the case that *IRTest* holds“. Das Signal *IRTest* kann nicht gleichzeitig gelten und nicht gelten.

Im Gegensatz dazu sind die beiden folgenden Anforderungen zwar *konsistent*, aber nur *trivial konsistent*: „If *Signal-A* holds then *Signal-B* holds after at most 10ms”, “It is never the case that *Signal-A* holds.” In jedem System, welches beide Anforderungen erfüllt, gilt niemals die Vorbedingung der ersten Anforderung, d.h. die erste Anforderung wird nur *trivial erfüllt*. Wir sagen eine Menge von Anforderungen ist *trivial konsistent*, wenn sie eine Anforderung enthält, die nur trivial erfüllt ist.

Die dritte Validierungseigenschaft prüft für Realzeit-Anforderungen ob Zeitschranken der Anforderungen miteinander in Konflikt stehen können. Die folgenden zwei Anforderungen sind *konsistent* und *nicht trivial konsistent*, trotzdem gibt es Belegungen so dass ein Konflikt auftritt. „If *IRTest* holds, then *IRLamps* holds after at most 10 seconds.”, “If *IRTest* holds, then NOT(*IRLamps*) holds for at least 6 seconds.” Beispielsweise tritt ein Konflikt auf falls *IRTest* ab Zeitpunkt t für 6 Sekunden gilt. Die erste Anforderung fordert dann, dass *IRLamps* spätestens bis zum Zeitpunkt t+10 gelten muss. Die zweite Anforderung fordert aber dass *IRLamps* bis zum Zeitpunkt t+12

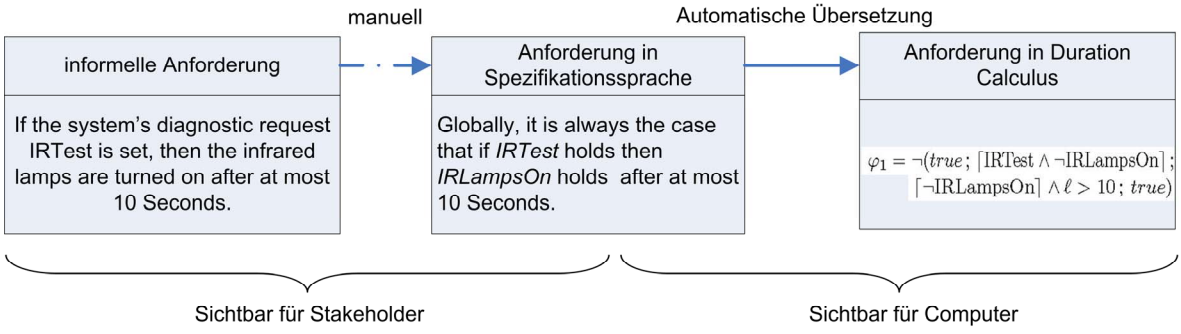

Bild 1: Anforderungen in Spezifikationssprache werden automatisch in Logik übersetzt.

inaktiv sein muss – spätestens zum Zeitpunkt t+10 tritt also ein Konflikt auf. Wir nennen diesen Konflikt rt-Inkonsistenz [2].

Evaluationsergebnisse

Um den Nutzen und die Anwendbarkeit der Validierungsmethode zu prüfen führten wir eine Machbarkeitsstudie durch. Wir nahmen Anforderungen aus verschiedenen BOSCH automotive Projekten, z.B. ein Projekt aus dem Fahrerassistenzbereich, eines aus der Motorsteuerung, eines zu einem Human Machine Interface, etc. Um eine repräsentative Stichprobe zu erhalten nutzen wir stratified sampling über die automotive application domains. Aus jedem Stratum wählten wir durch convenience sampling ein Projekt und daraus eine repräsentative Komponente. Wir drückten die Verhaltensanforderungen der jeweiligen Komponente in der Spezifikationssprache aus. Alle Anforderungen waren zuvor manuell gereviewt worden.

Für die Evaluation implementierten wir prototypisch ein Framework in Java welches Anforderungen in Spezifikationssprache automatisch in Logik übersetzt und welches die formalisierten Anforderungen dann automatisch auf Inkonsistenz, rt-Inkonsistenz und triviale Konsistenz prüft. Die Messungen führten wir auf einem PC Windows XP System, mit 2GHz IntelCore 2 Duo Prozessor mit 1GB RAM durch (es wurde nur ein Kern verwendet).

Die Anforderungsmengen pro Komponente variierten zwischen 10 und 82 Anforderungen (siehe Tabelle 1). Alle geprüften Anforderungsmengen waren konsistent. Nur die größte Anforderungsmenge war trivial konsistent. Wir denken, dass dies darauf hindeutet, dass aufgrund des großen Domänenwissens der Reviewer bei kleinen Anforderungsmengen diese Art Fehler bereits in Reviews gefunden wird. Es ist zu prüfen, ob Inkonsistenz und triviale Konsistenz bei größeren Anforderungsmengen öfter auftritt.

Im Gegensatz dazu trat rt-Inkonsistenz bereits auf kleinen Anforderungsmengen auf (Tabelle 1): drei der geprüften sechs Anforderungsmengen waren rt-inkonsistent und mussten überarbeitet werden. Die dritte Komponente benötigte die meisten Änderungen: 7 Anforderungen mussten hinzugenommen werden, 2 Anforderungen wurden gelöscht und 5 Anforderungen geändert. Das Tool half maßgeblich den Grund der rt-Inkonsistenz zu finden.

Zusammenfassung - Lohnt der Aufwand?

Um eine automatische Validierung zu ermöglichen müssen die Anforderungen in der Spezifikationssprache ausgedrückt werden. Das Erlernen dieser Sprache ist geringer Aufwand, da die Sprache nur aus wenigen Bauteilen besteht. Sofern die Entwicklung der Anforderungen mit deren Formulierung in Spezifikationssprache gekoppelt wird, ist auch der Zusatzaufwand gering. Der Berechnungsaufwand für die Validierung kann im schlimmsten Fall jedoch exponentiell wachsen. Für kleine Komponenten hielt sich der Berechnungsaufwand in Grenzen (<2h). Bei größeren Anforderungsmengen erhielten wir bei der Prüfung auf rt-Inkonsistenz aber teilweise Out-of-Memory Fehler. Hierfür müssen effizientere Algorithmen entwickelt werden. Die Prüfung auf Inkonsistenz und triviale Konsistenz skaliert besser, kann im schlimmsten Fall aber auch exponentiell wachsen. Der Gesamt-Aufwand der Methode für kleine Komponenten ist also eher gering, aber es ist zu prüfen wie sich die Berechnungszeit und somit auch der Aufwand für größere Anforderungsmengen verhält.

Die Case Study zeigt, dass der Nutzen der automatischen Validierung hoch ist: es wurden 3 rt-Inkonsistenzen entdeckt, die zuvor in keinem Review gefunden wurden. Die Prüfung auf Konsistenz und triviale Konsistenz scheint nur bei größeren Anforderungsmengen Fehler aufzudecken, die in einem normalen Review nicht gefunden werden. Weitere Vorteile der Methode sind dass der Requirements Engineer bereits in der Anforderungsentwicklung die Validierung durchführen kann, so sofort Feedback erhält falls die Anforderungsmenge Fehler enthält und bei der Fehlersuche unterstützt wird.

Referenzen

[1] Konrad S, Cheng B.H.C: „Real-time specification patterns“, ICSE 2005

[2] Post A, Hoenicke J., Podelski A: „rt-inconsistency: a new property for real-time requirements“, FASE 2011

	#Anf	konsistent?	nicht trivial konsistent?	rt-konsistent?	Korrektur-Aufwand
1	9	ja	ja	nein (1Min)	A:3, C:4
2	10	ja	ja (3Min 40s)	nein (33Min)	A:4, C:4
3	10	ja	ja (3s)	nein (1h 16Min)	A:7, D:2, C:5
4	13	ja	ja (1s)	ja (1s)	—
5	16	ja	ja (1s)	ja (6s)	—
6	17	ja	ja (1Min 32s)	n/a	n/a
7	27	ja	ja (1h 32Min)	n/a	n/a
8	82	ja	nein (1h 5Min)	ja (1h 6Min)	C:1

Tabelle 2: Ergebnis der Case Study. Die Spalten beziehen sich auf die Anzahl der Anforderungen, das Ergebnis der Validierung (in Klammern jeweils die Berechnungszeit, n/a bei Out-of-Memory) sowie den Korrektur-Aufwand, gemessen in Anzahl neuer Anforderungen (A), geänderter Anforderungen (C), gelöschter Anforderungen (D).