Modell-gestütztes Framework für das Testen von Mess- und Automatisierungssoftware für Prüfstände der Automobilindustrie

Dr. Udo Oligschläger, FEV GmbH, Aachen

oligschlaeger@fev.com

Zusammenfassung

Die zunehmende Komplexität heutiger Softwaresysteme erfordert eine sorgfältige und systematische Qualitätssicherung. Dies gilt insbesondere für sicherheitskritische Anwendungen wie Mess- und Automatisierungssysteme in Prüfständen der Automobilindustrie. Die Folge ist eine stetig wachsende Anzahl von durchzuführenden Tests verschiedenster Ausprägung in verschiedenartigsten Testumgebungen.

Der Beitrag gibt einen Überblick über ein bei FEV GmbH entwickeltes modell-basiertes Test-Framework, welches mittels einer domänen-spezifischen Spezifikationssprache sowohl die manuelle Definition von Tests erlaubt als auch die einfache Einbindung von Testfall-Generatoren ermöglicht. Dabei stützt sich diese Sprache sowohl auf die spezifizierbare Domäne des zu testenden Systems (SUT) als auch auf die besonderen Konzepte des Systemtests. Das Test-Framework erzeugt, zu einem großen Teil ohne manuelle Eingriffe, ausführbare Test-Suiten bestehend aus der Ablauf-Logik, den Testdaten und Konfigurationen sowohl für den Prüfling selbst als auch für die Umgebungssimulation. Es unterscheidet zwischen Parametern mit und ohne Echtzeit-Verhalten und generiert entsprechende Test-Artefakte.

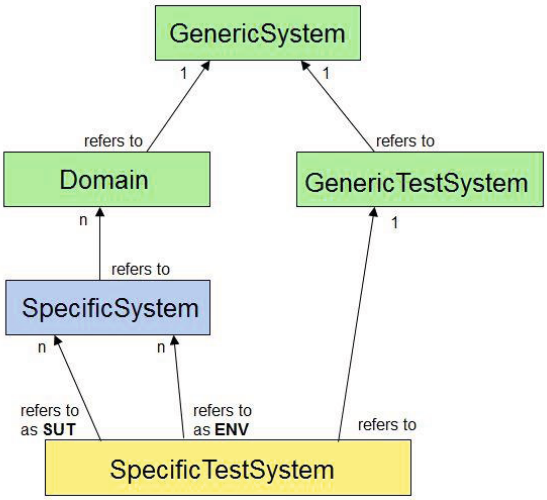
Keywords

Testautomatisierung, DSL, Domänen-spezifische Test-Spezifikationssprache

Unter der Zielsetzung manuell erstellte und automatisch generierte Testfälle gemeinsam und einheitlich behandeln zu können und möglichst auch Kollegen, die nicht unbedingt tiefere Programmiererfahrung haben (z.B. Produktmanager), in die Lage zu versetzen Test-Spezifikationen erstellen zu können und diese anschließend in ausführbare Testumgebungen umzusetzen, wurde im Geschäftsbereich Test-Systems der FEV GmbH ein modell-basiertes Test-Framework entwickelt. Die so entstehende Sprache ist einfach zu erlernen, da sie einen eng umrissenen Umfang hat und in weiten Teilen selbsterklärend ist.

Die angestrebten Vereinfachungen helfen insbesondere bei der Behandlung von Fehlermeldungen aus dem Feld, die typischerweise eine manuelle Erstellung von Testfällen nach sich zieht.

Mit dem Test-Framework ist eine nahtlose Werkzeugkette zur Erstellung, Bearbeitung und Weiterverarbeitung von Test-Spezifikationen entstanden, die als offenes System, u.a. zur Integration von kommerziellen Tools für die Testfall-Generierung und zur Anbindung an Werkzeuge aus benachbarten Prozessgebieten wie z.B. Feature- und Varianten-Management, Continuous-



Integration, Reporting, etc., ausgelegt ist. Für die automatisierte Generierung der ausführbaren Tests ist wichtig, dass das Test-Framework robust gegen Änderungen der Umgebung ist, da sonst manuelle Eingriffe notwendig werden. Dies wird durch Benutzung von Struktur-Informationen aus dem System-Modell bei der Generierung und durch die Einführung einer zusätzlichen Abstraktionsschicht erreicht.

Das Test-Framework basiert auf der separaten Spezifikation von Prüflings-bezogenen und Test-bezogenen Aspekten. Durch den Bezug auf eine aus mehreren Einzel-Modellen zusammengesetzte formale Spezifikation des

zu testenden Systems (SUT) und seiner Umgebung (ENV) können ungültige Test-Spezifikationen vermieden und die Effizienz bei der Festlegung der einzelnen Interaktionen verbessert werden. Neben der Spezifikation des SUT werden in einem weiteren Modell die verwendbaren Konzepte aus dem Bereich System-Test festgelegt. Dies gilt sowohl für organisatorische als auch für operative Dinge. So kann das gesamte Test-System, angefangen bei der Struktur und den Elementen einer Test-Suite,

- Test-Suite
- Test-Case
- Test-Phase
- Test-Sequence
- Test-Step

bis hin zum Verhalten, wie Timing der einzelnen Aktionen, Reaktion auf unerwartetes Verhalten des SUT, ebenso konfiguriert werden wie die im Test-Step (kleinstes organisatorisches Element) angebotenen Aktionen

- Stimulation
- Inspektion
- Orakel
- Check

Neben diesen bekannten Aktionen kommt hier eine generalisierte Form des Konzepts Statement zum Einsatz, die nicht nur textuelle und numerische Aussagen über gegebene Muster und Bereiche beinhalten kann, sondern auch Aussagen wie z.B.

- IsAsSetByLastStimuation
- IsAsSeenDuringLastInspection

Diese Art von Statements ist nutzbar für verschiedenste Zwecke, wie Vor-/Nach-Bedingungen, Orakel, Validierung der korrekten Ausführung von Aktionen. Durch die Einführung dieser zusätzlichen Konzepte kann an vielen Stellen auf Programmierarbeit, wie sie beim Einsatz einer General-Purpose-Sprache notwendig gewesen wäre, verzichtet werden.
Durch die Kombination/Verweben der beiden Spezifikationsketten entsteht schließlich die angestrebte Sprache zur Testfall-Spezifikation, die einerseits die Domänenspezifischen Elemente des Anwendungsbereichs als Objekte enthält und andererseits die Funktionalität zum Umgang mit diesen Objekten anbietet, die durch die spezifizierten Test-Konzepte umrissen wurde.
Da die zu testenden Systeme in weiten Teilen Echtzeitanforderungen erfüllen müssen, wurde die System-Beschreibung um die Eigenschaft „TemporalBehavior“ ergänzt, mit der wiederum jede Eigenschaft eines Systemelements näher spezifiziert werden kann. Diese Information wird später vom Generator genutzt, um den Testablauf während der Ausführung entweder direkt auf das jeweilige Systemelement bzw. seine Eigenschaften zugreifen zu lassen oder mit einer vom Haupttestablauf kontrollierbaren Kombination aus einem echtzeitfähigen Player, der die Systemstimulation übernimmt, und einem Logger, der die gesamten verfügbaren Elemente in Echtzeit protokolliert, zu arbeiten. Im zweiten Fall synchronisiert sich der Testablauf auf die Beendigung der im Player linear ausgeführten Echtzeit-Stimulationssequenz und führt anschließend die Inspektion und Bewertung der Systemreaktion durch. Dieser Ansatz wurde gewählt um den Eingriff in den unter Echtzeitbedingungen ausgeführten Systembereich möglichst klein zu halten.
Die erforderlichen Test-Adapter lassen sich in die Kategorien UI-Adapter (Maus, Tastatur, Bildschirm), File-Adapter (Konfigurations- und Log-Files) und Signal-Adapter einteilen. Die UI-Adapter stützen sich auf eine kommerzielle Bibliothek. Da die ordnungsgemäße Funktionsfähigkeit der UI-Adapter auch von äußeren Einflüssen abhängt (z.B. Betriebssystem-Updates), werden die Adapter durch automatisiert erzeugte und automatisiert ablaufende Tests überprüft bevor der eigentliche Test des SUT durchgeführt wird. Die File-Adapter werden zum größten Teil aus SUT-Modell und Test generiert. Für die Signal-Adapter wird, wenn wie beim Prüfstandsautomatisierungssystem (AuSy) möglich, auf Bordmittel zurückgegriffen. Das heißt, Echtzeit-Player und Echtzeit-Logger des AuSy werden zunächst herkömmlich getestet, bevor sie als Teil der Testumgebung für andere zu testende Module des AuSy benutzt werden. Für die Fälle, in denen diese Vorgehensweise nicht anwendbar ist, generiert das Test-Framework eine entsprechende Spezifikation für die Test-Umgebung, die der Tester dann manuell umsetzt.

Zusammenfassung und Ausblick

Das entwickelte Modell-basierte Framework wird für die manuelle Spezifikation von funktionalen Tests (als Folge von Fehlermeldungen aus dem Feld) und die darauffolgende Ableitung der ausführbaren Tests für Mess- und Automatisierungssysteme verwendet. Die Anwender waren dabei bereits nach kurzer Einarbeitung in der Lage, Testfälle effizient zu spezifizieren.
Zur Ergänzung werden derzeit Systeme zur Ableitung von Testfällen aus System-Modellen evaluiert. Nach entsprechender Integration können dann manuell erstellte und aus Verhaltensmodellen generierte Testfälle gemeinsam und einheitlich behandelt werden.
Als weitere Schritte sind die Kopplung mit Feature- und Variantenmodellen sowie einem Modell zur Formulierung und Verwaltung von Anforderungen, und Erweiterungen im Bereich Test-Reporting vorgesehen.