Gleitender Übergang vom manuellen zum automatisierten Test eingebetteter Software

Sadegh Sadeghipour

ITPower Solutions GmbH

Kolonnenstraße 26
10829 Berlin
sadegh.sadeghipour@itpower.de

Abstract. *Angesichts der zunehmenden Funktionsumfänge und wechselseitigen Abhängigkeiten eingebetteter Systeme wird deren Testprozess immer aufwendiger. Eine erste Voraussetzung zur Beherrschung der Komplexität des Testprozesses ist die Testautomatisierung. Die hohen Kosten und der Aufwand der Einführung und des Betriebs der Testautomatisierung sowie der Bruch zwischen dem manuellen und automatisierten Testprozess sind Aspekte, welche viele Unternehmen davon abhalten, ihren Testprozess zu automatisieren. Im vorliegenden Beitrag wird eine Software-Architektur zur Automatisierung der Testausführung und -auswertung präsentiert, welche den genannten Nachteilen entgegenwirkt. Ferner wird ein Testframework vorgestellt, das die vorgeschlagene Architektur umsetzt.*

1 Einleitung

Entwickler eingebetteter Systeme in verschiedenen Domänen sind mit stets steigenden Anforderungen an die Softwarequalität konfrontiert. Dabei dienen die analytischen Maßnahmen, vor allem der dynamische Test, als Hauptmittel zur Prüfung und Sicherstellung der geforderten Qualitätsmerkmale. Um wettbewerbsfähig zu bleiben und mit Imageverlust und hohen Kosten verbundene, mögliche Rückrufaktionen zu vermeiden, sind deshalb umfangreiche Tests in verschiedenen Testphasen unumgänglich.

Die neuen Industrietrends (Industrie 4.0, Internet of Things, Digitalisierung, autonomes Fahren, …), welche quantitativ und qualitativ neue Ebenen der Automatisierung und Vernetzung einführen, bringen weitere neue Anforderungen an die Qualität und den Test eingebetteter Software mit sich. Die in den Geräten eingebauten elektronischen Systeme müssen in der Lage sein, mit unterschiedlichen Software-Versionen diverser, ggf. zur Zeit der Entwicklung unbekannter, Netzwerkknoten zu kommunizieren. Dies erfordert zum Teil die Anpassung und das Nachladen der Software während der Laufzeit. Ferner ist das typische, zukünftige eingebettete System vielfältigen Variationen der physikalischen und digitalen Umgebung (Druck, Temperatur, Reibung, Kommunikationsbusse, digitale Ein- und Ausgänge, …) ausgesetzt.

Die hochgradige Automatisierung und Vernetzung sowie die neuen Anwendungsfälle implizieren auch neue Qualitätsmerkmale, beispielsweise Aktualisierbarkeit und Diagnostizierbarkeit, welche über die klassischen Qualitätsmerkmale hinaus wie z. B. Korrektheit, Robustheit, Zuverlässigkeit gelten und durch den Test geprüft und gesichert werden müssen.

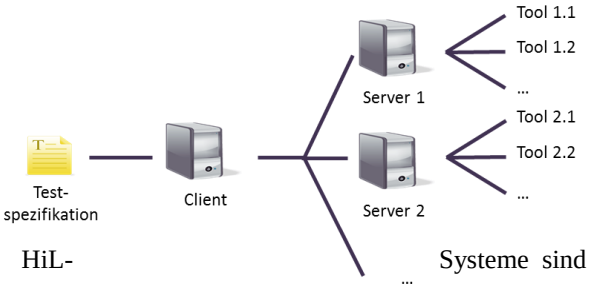
In diesem Kontext nimmt die Bedeutung des Softwaretests zu. Durch den Anstieg des Umfangs und der Variation der Funktionen, der Schnittstellen und der Kommunikationsmöglichkeiten der eingebetteten Software sowie die neu hinzugekommenen Qualitätsmerkmale werden Testumfänge zwangsläufig größer. Gleichzeitig limitieren die Wirtschaftlichkeit und der Wettbewerbsdruck die Kosten und den Aufwand des Tests. Ausweg aus diesem Dilemma können nur effektive und effiziente Tests bieten. Eine Erhöhung der Effizienz ist vor allem durch Automatisierung möglich. Jedoch ist die Einführung und der Betrieb der Automatisierung in der Regel mit hohen Kosten verbunden, u.a. durch notwendige Investitionen zur Anschaffung und laufende Betriebskosten für Wartung der Testumgebung, Schulung der Mitarbeiter und dem Kauf entsprechender Software-Lizenzen.

Der vorliegende Beitrag beschreibt ein technisches Vorgehen, wie eine existierende, heterogene Toolumgebung des manuellen Tests eingebetteter Software zu einer Umgebung für den automatisierten Test aufgewertet werden kann. Ferner wird ein Testframework namens ContinoProva präsentiert, welches die notwendige Infrastruktur und ein Frontend für eine solche Automatisierung anbietet.

2 Testautomatisierung

Vor dem Hintergrund der oben genannten Herausforderungen des Softwaretests ist die Automatisierung der Testausführung und Testauswertung eingebetteter Software ein dringender und aktueller Bedarf der Industrie. Beim automatisierten Test werden oft Hardware-in-the-Loop-Systeme (kurz: HiL-Systeme) eingesetzt [1]. Ein HiL-System ist eine Nachbildung der realen Umgebung für den Test eines eingebetteten Systems. Es enthält teils reale und teils simulierte Sensoren, Aktuatoren, mechanische und elektrische Komponenten. Der (in der Regel echtzeitfähige) HiL-Rechner, der die Tests steuert, versorgt die analogen

und digitalen Eingänge des Testobjekts mit Werten und protokolliert die Testausgaben.



HiL- Systeme sind

Abbildung 1: Client-Server-Architektur des Testframeworks

zwar in den Industriebranchen, welche eingebettete Systeme entwickeln und einsetzen, weit verbreitet, jedoch stößt deren weitere Verbreitung oft auf Schranken, die im Folgenden kurz geschildert werden:

- Die Einführung von HiL-Systemen ist mit hohen Investitions- und Betriebskosten verbunden. Diese setzen sich u.a. aus Kosten der Hardware, der Softwarelizenzen und Softwarewartung, dem Aufwand für die Konfiguration von Hard- und Software, dem Aufwand zur Erstellung der Umgebungsmodelle sowie der Schulung der Mitarbeiter zusammen.
- Die externe Schnittstelle eines Testobjekts bestimmt die Hardware- und Softwarekonfiguration des entsprechenden HiL-Systems sowie das Verhalten der eingesetzten Umgebungsmodelle. So müssen für den Test verschiedener Hardwarekomponenten eines eingebetteten Systems individuelle HiL-Systeme aufgebaut und in Betrieb genommen werden. Das Gleiche gilt auch für den Test der Produkte verschiedener Entwicklungsprojekte, falls deren Schnittstellen differieren.
- Aus den o.g. Gründen und zur Kostenreduzierung werden HiL-Systeme oft nur beim Systemtest eingesetzt. Beim Integrations- und Komponententest behilft man sich mit anderen Lösungen.
- Ein gleitender Übergang vom manuellen zum automatisierten Test unter Beibehaltung der verwendeten Tools, wie z. B. Oszilloskop, Debugger, Buswerkzeuge, etc., ist im Falle der Verwendung von HiL-Systemen nicht möglich. Folglich können die aktivierten Investitionen und das vorhandene Know-how im Umgang mit Tools nicht weiter verwertet werden.

Mit dem Ziel, die oben genannten Limitierungsfaktoren aufzuheben bzw. zu mindern, wird in diesem Beitrag eine Architektur für ein Testframework vorgestellt, welche die Testausführung und Testauswertung automatisiert und die für den manuellen Test verwendeten Tools integriert. Ein solches Framework ist in allen Testphasen einsetzbar und besitzt das Potential wegen relativ niedrigen Kosten- und Aufwandsschwellen auch durch kleine und mittelständische Unternehmen verwendet zu werden.

3 Architektur eines Frameworks zur Testautomatisierung

In der manuellen Testumgebung eines eingebetteten Systems werden in der Regel unterschiedliche Hardware- und Softwaretools eingesetzt, z. B.:

- Relaisboxen zur Ansteuerung digitaler Eingänge des Testobjektes
- Spezielle Hardwaremodule zur Ansteuerung analoger Eingänge des Testobjektes
- Oszilloskop zur Messung der Ausgangssignale des Testobjektes
- Buswerkzeuge zum Setzen und Lesen von Busbotschaften
- Debugger für den lesenden und schreibenden Zugriff auf interne Parameter und Variablen der Software des Testobjektes
- Spezielle Tools für den lesenden und schreibenden Zugriff auf die Speichermodule des Testobjektes

Das primäre Ziel der hier vorgestellten Architektur ist die Einbindung von bereits im Einsatz befindlichen Tools ins Testframework. Dadurch können die Erfahrungen und das Know-how der Mitarbeiter zum Umgang mit den Tools weiterhin genutzt werden. Ferner profitiert die neue Testautomatisierung von den schon getätigten Investitionen. Die Testautomatisierung mit Integration vorhandener Tools erfordert allerdings, dass von den Benutzer- und Programmierschnittstellen einzelner Tools abstrahiert und ein einheitliches Frontend zur Spezifikation von Tests und Steuerung der Tools bereitgestellt wird. So hat der Benutzer des Testframeworks die Möglichkeit, über Toolgrenzen hinweg Tests spezifizieren und automatisch ausführen zu können.

Während eines Testdurchlaufs sollen gemäß der im Frontend definierten Testspezifikation schrittweise von einzelnen eingebunden Tools Dienste angefordert werden, z. B. Senden von Busbotschaften durch das Buswerkzeug oder Lesen von Signalen am Ausgang des Testobjekts durch das Oszilloskop. Es bietet sich deshalb eine Client-Server-Architektur an, in der der Client vom entsprechenden Server einen Dienst anfordert und der Server, an dem ein externes Tool über eine offene Schnittstelle angebunden ist, diese Anforderung an das Tool weiterleitet und ggf. die Rückgabe der Durchführung des Dienstes an den Client zurückgibt (Abbildung 1). Durch die Client-Server-Architektur ist es möglich, Tools von einem Rechner aus fernzusteuern. Es können nicht nur Signale zwischen den Tools ausgetauscht, sondern auch die Tools gestartet oder angehalten werden. Z. B. kann das Tool zur Buskommunikation zum Testbeginn automatisch initialisiert werden oder der Debugger wird nach Eintreten eines bestimmten Ereignisses aktiviert. Die lokale Verteilung der Tools bietet erhebliche Vorteile hinsichtlich der Performanz bei der Testausführung. Diese Architektur setzt natürlich voraus, dass die angebundenen Tools

über entsprechende Application Program Interfaces (API) verfügen.

Die in Abbildung 1 dargestellte Architektur muss in der Lage sein, die durch die Testspezifikation spezifizierten Testschritte sequenziell zur Ausführung zu bringen und die Ausführungsergebnisse, d.h. die aktuellen Größen des Testobjektes nach der Ausführung des Testschritts wieder in den Testablauf einfließen zu lassen.

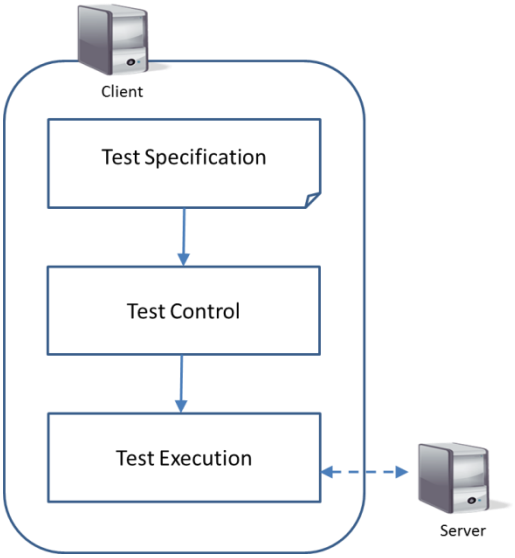


Abbildung 2: Softwaremodule des Clients

Die Architektur des Testframeworks setzt keine bestimmte Form der Testspezifikation voraus. Testspezifikationen können sequenzielle Testabläufe definieren, die z. B. in einem Tabellenkalkulationsprogram oder als Testsequenzen eines Klassifikationsbaums [2] dargestellt sind, oder durch UML-Verhaltensdiagramme repräsentiert werden. In jedem Testschritt wird das Testobjekt mit den in der Testspezifikation definierten Eingabedaten ausgeführt und die Ausgabedaten werden an die Testumgebung zurückgegeben. So muss der Client ein Softwaremodul enthalten, das die Daten des aktuellen Testschritts der Testspezifikation entnimmt und daraus eine ausführbare Code-Einheit generiert. Dieses Softwaremodul nennen wir *Test Control*. Der ausführbare Testschritt wird dann einer Testausführungsmaschine, die wir *Text Execution* nennen, übergeben. Die *Test Execution* stellt fest, welche Tools und dementsprechend welche Server zur Ausführung des Testschritts heranzuziehen sind. Denn ein Testschritt kann aus mehreren Operationen bestehen, welche durch unterschiedliche Tools auszuführen sind, z. B. das Setzen eines digitalen Signals durch eine Relaisbox, das Lesen einer Busbotschaft durch das Buswerkzeug, etc. (Abbildung 2).

Die Trennung von *Test Control* und *Test Execution* dient dazu, eine leichte Anpassung der Implementierung an verschiedene Testspezifikationsarten zu ermöglichen. Dadurch kann für jede Testspezifikationsart ein spezifisches, austauschbares *Test Control* realisiert und eingesetzt werden.

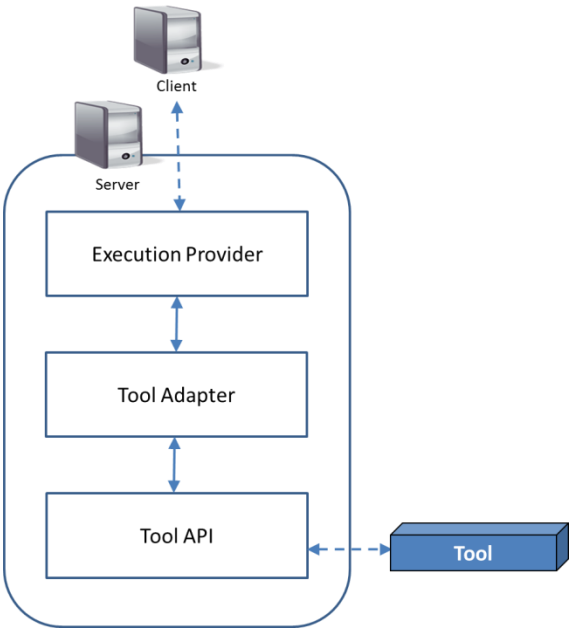


Abbildung 3: Softwaremodule des Servers

Auf der Serverseite wird ein Modul benötigt, welches die Dienstanforderung von *Test Execution* übernimmt und über die Tool-API an das entsprechende Tool übergibt sowie die Rückgabewerte nach der Ausführung des Dienstes empfängt und an *Test Execution* zurückgibt. Dieses Modul nennen wir *Execution Provider*. Um auch hier eine leichte Austauschbarkeit von Tools zu ermöglichen, wird die Aufgabe des Aufrufs und des Empfangs der Rückgabewerte der Funktionen der Tool-API in ein Extramodul namens *Tool Adapter* eingekapselt (Abbildung 3).

Die in Abbildung 2 und Abbildung 3 dargestellten Architekturelemente zeigen den Hauptstrang der Testautomatisierung. Für eine bessere Übersicht wurden hier die Aktivitäten der Testauswertung und Reportgenerierung, welche innerhalb des Clients angesiedelt sind und direkt mit dem Modul *Test Execution* kommunizieren, nicht dargestellt.

4 Testframework ContinoProva

Auf der Basis der im Abschnitt 3 dargestellten Architektur wurde ein Testframework namens ContinoProva entwickelt [3]. Es bietet eine grafische Benutzeroberfläche (GUI) zur sequenziellen Testspezifikation an. Abbildung 4 zeigt einen Screenshot der GUI. Die linke Seite besteht aus einem Baum mit den Strukturelementen der Testspezifikation. Eine Testspezifikation besteht aus einer oder mehreren Testgruppen, die wiederum eine oder mehrere Testsequenzen enthalten können. Eine Testsequenz wird durch eine Folge von Testschritten gebildet, die sequenziell abgearbeitet werden. Testschritte sind die kleinsten Testausführungseinhei-

ten. Die Operationen eines Testschritts, welche in ContinoProva Testtasks genannt werden, werden alle zu einem Zeitpunkt ausgeführt.

Der Inhalt eines Testtasks kann z.B. das Setzen eines Eingangssignalwertes oder die Messung eines Ausgangssignals sein. Wenn ein Testtask im Baum selektiert wird, kann die auszuführende Operation im entsprechenden Formular im rechten Fenster spezifiziert werden (siehe Task 1.1.1.1 in Abbildung 4). Dabei wird zunächst der Service (Services sind Implementierungen von *Tool Adapter* in Abbildung 3) ausgewählt, welcher für das zu steuernde Tool zuständig ist. Kontextbezogen werden die entsprechenden Service-Funktionen zur Auswahl angeboten. Für die Zeitsteuerung stehen Funktionen wie *WaitAfter*, *WaitBefore*, *InWaitCheck* und *WaitUntil* zur Verfügung. Mit Hilfe dieser Funktionen können z. B. Wartezeiten vor der Messung eines Signals definiert, auch Eigenheiten der Testumgebung wie beispielsweise Latenzzeiten berücksichtigt oder Triggerpunkte für folgende Testaktionen definiert werden.

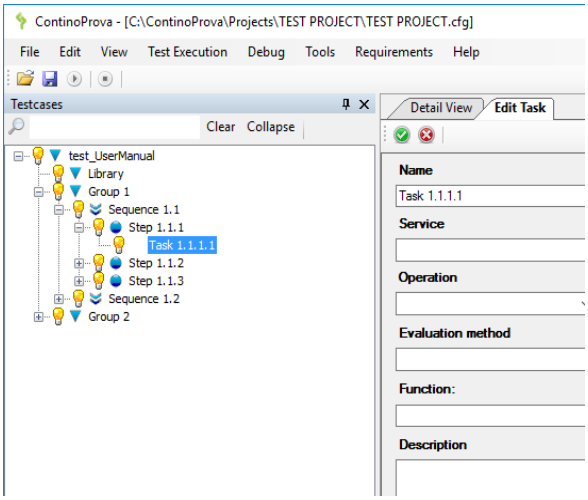


Abbildung 4: ContinoProva

ContinoProva unterstützt ein Bibliothekskonzept. Wiederkehrende Testelemente (Testschritte, Testsequenzen, Testgruppen, oder auch Testauswertungsmethoden) können als Bibliotheksbaustein definiert und in verschiedenen Testspezifikationen verwendet werden. So kann z. B. die Initialisierungsroutine des Steuergerätes oder das Erreichen eines bestimmten Betriebsmodus einmalig spezifiziert und in der Bibliothek abgelegt werden. Die Bibliotheksbausteine werden als gekennzeichnete Referenzen in die Testspezifikation eingefügt.

ContinoProva bietet auch einen Debugger zur Analyse des Testablaufs. Der Benutzer kann durch das Setzen von Unterbrechungspunkten und die schrittweise Ausführung des Tests die Korrektheit einzelner Testschritte sowie der Kommunikation mit den externen Tools prüfen. Auch ist es möglich, beim angehaltenen Testlauf den Zustand eingebundener Tools über ihre Benutzeroberfläche genauer zu analysieren.

5 Fazit

Im vorliegenden Beitrag wurde eine Architektur vorgestellt, welche unter Beibehaltung verschiedener Tools des manuellen Testprozesses die Testausführung und -auswertung eingebetteter Software automatisiert. Diese Architektur ermöglicht eine Abstraktion der spezifischen Schnittstellen der eingebundenen Tools und eine vom Programmcode unabhängige Beschreibung des Testablaufs. Dadurch werden der Austausch der im Testprozess beteiligten Tools und die Wiederverwendung von Testspezifikationen erleichtert. Ferner erlaubt die vorgestellte Architektur den Einsatz verschiedener Testspezifikationsarten, z. B. sequenzielle Testablaufbeschreibungen und Zustandsdiagramme.

Ein auf der Basis der vorgeschlagenen Architektur realisiertes Testframework ist im Vergleich zu Hardware-in-the-Loop-Systemen in verschiedenen Hinsichten kostengünstiger: u.a. Erstinvestition, Schulung der Mitarbeiter und Betriebskosten. Ferner kann ein solches Testframework in verschiedenen Testphasen – Komponententest, Integrationstest und Systemtest – eingesetzt werden.

Das Testframework und die eingebundenen Tools laufen auf nicht echtzeitfähigen Rechnern. Diese Tatsache bestimmt auch die Grenzen seines Einsatzes. Sind harte Echtzeitanforderungen im Testobjekt zu prüfen, ist der Einsatz einer Testumgebung mit Echtzeitrechnern unumgänglich.

Das Testmanagement sollte zwar unabhängig von der Testautomatisierung betrachtet und ggf. durch ein eigenständiges Tool unterstützt werden. Jedoch ist es sinnvoll, in bestimmten Anwendungsfällen, besonders wo es kein Testmanagement-Tool eingesetzt wird, das hier vorgestellte Testframework um grundlegende Features des Testmanagements zu erweitern. Solche Features sind z. B. Nachverfolgbarkeit zwischen Anforderungen und Tests, Versionierung der Testartefakte und Planung der Testausführung. In dem hier vorgestellten Testframework ContinoProva sind einige dieser Features umgesetzt bzw. für die Umsetzung in den zukünftige Toolversionen geplant.

6 Literatur

[1] Schlager, Martin. *Hardware-in-the-Loop Simulation.* Saarbrücken : VDM Verlag Dr. Müller, 2008. ISBN 978-3-8364-6216-7.

[2] Grochtman, Matthias und Grimm, Klaus. Classification trees for partition testing. *Software Testing, Verification & Reliability.* 1993, Vol. 3, 2, pp. 63-92.

[3] ITPower Solutions GmbH. ContinoProva. [Online] http://www.continoprova.de.