Generierung von Systemtestfällen für Car Multimedia Systeme aus domänenspezifischen UML Modellen

Oliver Alt
Robert-Bosch GmbH, CM-DI/ESI2
Daimlerstr. 6, D-71229 Leonberg
oliver.alt2@de.bosch.com

Zur Sicherung der Qualität und Zuverlässigkeit erfordert ein steigender Anteil an Software in Infotainmentsystemen moderner Fahrzeuge im gleichen Maße einen steigenden Anteil an Systemtestfällen. Die Wiederverwendbarkeit solcher Testfälle in verschiedenen Projekten gestaltet sich oft schwierig, da projektspezifische Informationen in den Testfällen enthalten sind. In diesem Papier wird zunächst ein Konzept vorgestellt um Systemtestfälle aus domänenspezifischen Modellen zu gewinnen. Dabei wird Systemverhalten auf verschiedenen Abstraktionsebenen beschrieben. Dadurch erfolgt eine Trennung von projektspezifischen und projektübergreifenden Informationen. Aufgrund der großen Akzeptanz in Industrie und guter Werkzeugunterstützung wird UML2 erweitert durch domänenspezifische Profile als Modellierungssprache eingesetzt. Anhand eines konkreten Beispiels aus der Infotainmentdomäne, dem MOST-Bussystem (Media Oriented System Transport), wird gezeigt, wie man von projektunabhängigen zu projektspezifischen Testfällen gelangt.

Schlüsselwörter: Systemtest, Testfallgenerierung, UML2, SysML, MOST

1 Einleitung

Der Anteil an Software in Infotainmentsystemen moderner Fahrzeuge steigt rapide an. Neue Funktionen wie Internet, Anbindung von Konsumerelektronikprodukten (z.B. Apple I-Pod) und 3D Navigation werden durch Kunden und Automobilhersteller gefordert. Um die Qualität solcher komplexen Systeme sicherzustellen, sind in allen Phasen des Entwicklungsprozesses umfangreiche Softwaretests erforderlich. Die endgültige Qualitätsprüfung der Software während der Entwicklung geschieht durch den System- und/oder Systemintegrationstest. Testfälle für diese Tests werden heute meist per Hand von menschlichen Testentwicklern geschrieben und ausgeführt. Mit steigender Zahl der Systemfunktionen sind auch mehr Testfälle für den Systemtest erforderlich. Dies führt aufgrund der benötigten Resourcen zu erhöhten Kosten und Zeitaufwänden. Darüber hinaus enthalten Testfälle sehr oft projektspezifische Daten, wie z.B. spezielle Nachrichten, die gesendet werden. Die direkte Wiederverwendung der Testfälle in anderen, ähnlichen Projekten ist daher nicht bzw. kaum möglich. Schlechte Wiederverwendbarkeit, monolithische Konstruktion und Einzelanfertigung für ähnliche Aufgaben sind Probleme, die aus der Softwareentwicklung wohl bekannt sind. Die Softwaretechnik versucht die Probleme durch Erhöhung des Ab-

straktionsgrades durch modellbasierte Entwicklung und die Verwendung domänenspezifischen Sprachen (DSM) zu lösen. Der Erfolg dieses Ansatzes wird durch zahlreiche Beispiele untermauert (z.B. [Kel05]). Der hier beschriebene Ansatz wendet diese Konzepte im Hinblick auf die automatische Erzeugung von Systemtestfällen an. Als Modellierungssprache wird dabei UML2 [OMG04] mit domänenspezifischen Erweiterungen, aufgrund ihrer großen Akzeptanz und Werkzeugunterstützung verwendet.
Der Rest des Papiers strukturiert sich wie folgt. Abschnitt 2 beschreibt die Konzeption des Systemmodells, insbesondere die Einteilung in verschiedene Modellebenen. Ein Anwendungsbeispiel in Form eines MOST (Media Oriented System Transport) Systems wird in Abschnitt 3 eingeführt. Abschnitt 4 beschreibt wie die Verhaltensbeschreibungen der verschiedenen Ebenen verknüpft und daraus Testfälle generiert werden können. Verwandte Arbeiten und eine kurze Abgrenzung erfolgt in Abschnitt 5. Eine Zusammenfassung und einen Ausblick gibt Abschnitt 6.

2 Konzeption des Systemmodells

Um ein Systemmodell des zu testenden Systems zu konzipieren, aus dem Systemtestfälle generiert werden können, muss zunächst klar sein was ein Systemtestfall ist. Ein Systemtestfall besteht mindestens aus den folgenden zwei Schritten:

1. Durchführen von Aktionen aus Sicht des Benutzers des Systems (Trigger).
2. Überprüfen der Reaktion des Systems und beantworten der Frage, ob die Systemreaktion mit dem gewünschten Systemverhalten übereinstimmt (Validierung).

Daher muss das Systemmodell zumindest genügend Informationen enthalten, um die Trigger abzuleiten. Informationen für die Validierung können dann noch zusätzlich modelliert oder aus dem Modell selbst gewonnen werden (Modell als Testorakel).

2.1 Dreiteilige Modellarchitektur

Um Testfälle und Modelle wieder verwenden zu können, muss eine Trennung von projektspezifischen Systeminformationen, welche die konkrete technische Realisierung beschreiben, und projektunabhängigen Systeminformationen erfolgen. Dazu wird das Modell in drei Teile, bzw. Ebenen eingeteilt. Die unterste Ebene bildet die **technische, projektspezifische Ebene** des Systems. Hier wird Systemverhalten durch konkrete Busnachrichten (z.B. CAN- oder MOST-Botschaften) definiert. Es wird beschreiben *wie* das System etwas tun soll (z.B. *Sende MOST Nachricht: AudioDiskPlayer.DeckStatus.Set(Play)*). Auf dieser Ebene werden heutzutage Systemtests typischerweise beschrieben, was eine Wiederverwendung in anderen Projekten fast immer unmöglich macht.

Daher wird eine mittlere Ebene definiert. Auf dieser **funktionalen Ebene** werden domänenspezifische Systemaktionen definiert, die auch projektübergreifend gültig bleiben. Es wird

beschrieben, *was* das System tun soll (z.B. *Wiedergabe starten*) und wie diese Aktionen zueinander in Beziehung stehen.

Darüber kommt die dritte, oberste Ebene, das **HMI** (Human Machine Interface), durch welches das Komplettsystem mit dem Endanwender kommuniziert. Das HMI benutzt die zur Verfügung gestellten Aktionen der funktionalen Ebene um die Systemreaktionen auszulösen und wird dadurch austauschbar. Für die Modell-basierte Entwicklung des HMI stehen heute bereits spezielle Werkzeuge zur Verfügung, die graphische Modellierungsmöglichkeiten für die Benutzerschnittstelle mit Automatenmodellen kombinieren (z.B. [3SO04]). Daher wird diese Ebene im Hinblick auf Testfallgenerierung hier nicht weiter betrachtet.

2.2 Domänenspezifische Modellierung

Durch den Einsatz von domänenspezifischer Modellierung bzw. domänenspezifischen Sprachen können bestimmte Sachverhalte in der Anwendungsdomäne besonders effizient ausgedrückt werden, da eine höhere Abstraktionsebene genutzt wird als mit einer allgemeinengültigeren Sprache (vgl. [LS06]). Aus diesem Grund soll für jede der drei Ebenen eine eigene für den jeweiligen Einsatzzweck passende Sprache verwendet werden.

Da UML2 als Modellierungssprache zum Einsatz kommen soll, müssen die domänenspezifischen Sprachanpassungen mit Hilfe von UML-Profilen vorgenommen werden. Ein relativ neues Profil zur Beschreibung von Systemen ist SysML (System Modeling Language) [Sys06]. Es bietet neue Konzepte, die sich bei der Modellierung von Infotainmentsystemen gut einsetzen lassen (z.B. Parametrische Gleichungen oder Erweiterung der Aktivitätsdiagramme). SysML gestattet es außerdem durch weitere Profildefinitionen die Sprache noch weiter an die eigenen Bedürfnisse anzupassen.

Aus Platzgründen kann in diesem Papier nicht auf die komplette Definition der Modellierung und Techniken zur Testfallgenerierung eingegangen werden. Daher wird im folgenden am Beispiel eines auf MOST basierten Audiosystems gezeigt, wie Testfälle der funktionalen Ebene mit der technischen Ebene verknüpft werden.

3 Anwendungsbeispiel: Audiosystem mit CD Player und Verstärker

Als konkretes Anwendungsbeispiel dient ein vereinfachtes Audiosystem, bestehend aus CD Player und Verstärker.

3.1 Technische Ebene: MOST

Der Media Oriented System Transport (MOST) [MOS04] ist ein objektorientiertes Bussystem, das speziell zur Vernetzung von Multimediasystemen konzipiert wurde. Neben

den Mediendaten wie Audio- und Videoströme, können durch einen Kontrollkanal auch Steuernachrichten zu den angeschlossenen Geräten gesendet werden.

Bei MOST handelt es sich um ein Master-Slave System mit Ringtopologie, bei dem ein Gerät als Master und alle anderen als Slaves fungieren. Die Funktionen der einzelnen Geräte werden in sog. Funktionsblöcken (*Function Blocks, fb*) organisiert. Funktionen werden durch die Funktionsblöcke klassifiziert und eine Instanz eines Funktionsblockes bildet dann ein virtuelles Gerät. So enthält der Funktionsblock `AudioDiskPlayer` [MOS03] alle nötigen Funktionen um einen CD Player zu realisieren. Ein MOST Gerät (*MOST Device*) kann auch mehrere Funktionsblöcke enthalten, z.B. können Verstärker und CD Player in einem Gerät integriert sein. Es ist aber auch möglich die Funktionsblöcke auf verschiedenen MOST Hardwaregeräten physikalisch zu verteilen. In den Geräten werden dann Instanzen der Funktionsblöcke gebildet. In einem sog. formalen MOST Funktionskatalog wird die Syntax aller MOST Kontrollnachrichten der Funktionsblöcke definiert.

3.2 Funktionale Ebene

Die Definition der Aktivitäten der funktionalen und damit projektunabhängigen Ebene des Beispielsystems zeigt Abbildung 1.

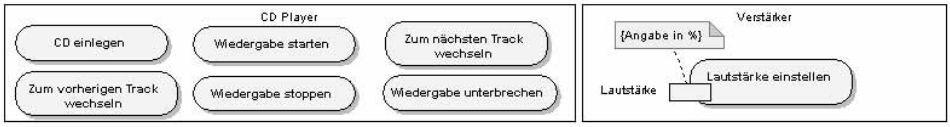

Abbildung 1: Aktivitäten der funktionalen Ebene

Diese Aktivitäten sind domänenspezifisch für das Audiosystem und beschreiben es aus Sicht des Benutzers. Der Benutzer ist dabei entweder der direkte Endanwender oder das HMI. Testfälle, die aus den Modellen der funktionalen Ebene abgeleitet werden, sind daher für alle Geräte einer Domäne, und damit gleichem Funktionsumfang, verwendbar.

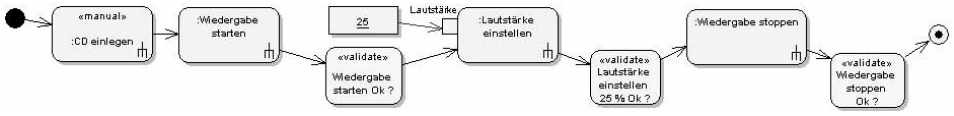

Abbildung 2: Beispiel eines Testfalles

Abbildung 2 zeigt beispielhaft einen solchen Testfall. Dieser Testfall lässt sich als manueller Test auf diverse Audiosysteme bestehend aus CD Player und Verstärker anwenden, unabhängig von der konkreten technischen Realisierung (z.B. Consumer Hifi Anlage oder Car Hifi Anlage).

4 Verknüpfung von funktionaler und technischer Ebene

Um zu einer halb- oder vollautomatischen Ausführung der Testfälle zu gelangen, müssen den Aktivitäten der funktionalen Ebene nun Aktivitäten der technischen Ebene zugewiesen werden, die die Realisierung am konkreten System beschreiben. Dies wird am Beispiel des MOST Audiosystems für zwei Aktivitäten der funktionalen Ebene demonstriert.

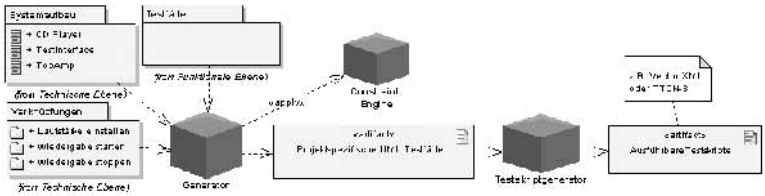

Abbildung 3: Ablauf der Testfallgenerierung

Abbildung 3 zeigt den Ablauf einer Generierung von ausführbaren Testfällen aus den Modellen. Ein Generator erzeugt aus den Testfällen der funktionalen Ebene und den Modellinformationen der technischen Ebene zunächst projektspezifische UML-Testfälle. Dazu werden die Aktionen der funktionalen Ebene durch projektspezifische Aktionen ersetzt. Als Grundlage dienen die modellierten Verknüpfungen zwischen der funktionalen und technischen Ebene in Form von Aktivitätsdiagrammen.

Abbildung 4: Verknüpfung Wiedergabe starten mit entsprechender MOST Funktion

In Abbildung 4 ist beispielhaft die Verknüpfung der Aktivität `Wiedergabe starten` der funktionalen Ebene mit der entsprechenden MOST Funktion auf der technischen Ebene zu sehen. Die Realisierung der konkreten Abläufe der funktionalen Aktivitäten wird mit Aktionen der technischen Ebene als Aktivitätsdiagramm modelliert. Hier im Beispiel wird im Funktionsblock `AudioDiskPlayer` der `DeckStatus` auf den Wert `Play` gesetzt, was das Abspielen der Disk startet. Die mit dem Stereotyp `MOSTAction` versehene Aktion ist Teil der Sprache der technischen Ebene und verwendet die in [AS06] beschriebene Nomenklatur. In einem zweiten Generierungsschritt kann daraus ein ablauffähiges MOST Testskript in Form von TTCN-3 oder ein XML Testmodul für das MOST Testwerkzeug CANoe.MOST der Fa. Vector Informatik [Vec05] erzeugt werden.

Aus Platzgründen können hier nur einfache Beispiele von Verknüpfungen zwischen funktionaler und technischer Ebene dargestellt werden. Da der MOST selbst ein domänenspezifisches Bussystem ist, fällt die Zuordnung hier oft aber auch besonders einfach aus.
Dynamisches MOST Verhalten wird üblicherweise mit Message Sequence Charts beschrieben[1]. Durch die Verwendung von Aktivitätsdiagrammen ergeben sich im Gegensatz

[1] Ein Telekommunikationsstandard (ITU-T Z.120), nahezu identisch mit UML2 Sequenzdiagrammen.

zu Sequezdiagrammen jedoch einige Vorteile. Bereits auf der funktionalen Ebene werden Aktivitäten und Aktivitätsdiagramme zur System- und Testbeschreibung eingesetzt. Durch die Verwendung auf verschiedenen Modellebenen wird die Erlernbarkeit und Beherrschbarkeit der Modellierungssprache erleichtert. Weiterhin werden auch Aktivitätsdiagramme zur Verknüpfung zwischen funktionaler und technischer Ebene verwendet. Auch Objektflüsse lassen sich in Aktivitätsdiagrammen explizit beschreiben. Bei MOST Systemen wird davon häufig Gebrauch gemacht, um z.B. Werte einer Antwort auf eine `Get` Anfrage als Parameter einer anderen MOST Funktion zu verwenden. Sequenzdiagramme bieten hierzu keine direkte Unterstützung. Objektflüsse zwischen verschiedenen Kommunikationspartnern und Nachrichten werden daher nur schwer erkennbar, bzw. werden durch Prosa Annotationen beschrieben.

4.1 Parameteranpassung

Häufig kommt es vor, dass auf der funktionalen Ebene Parameter von Aktivitäten andere Einheiten und Darstellungen haben, als auf der technischen. Dies geschieht oft schon aufgrund von technisch bedingten Implementierungsvorgaben. Beispielsweise wird eine Variable auf der technischen Ebene als ein Byte realisiert, was einen Wertebereich von 0-255 bei vorzeichenloser Darstellung ergibt. Die Definition auf funktionaler Ebene kann jedoch, z.B. wegen besserer Handhabbarkeit, eine Andere sein. Daher wird ein Mechanismus benötigt, der zwischen Parametern der funktionalen und technischen Ebene umrechnet. Mit der SysML wurde mit den parametrischen Gleichungen und Constraint Blocks etwas definiert, dass sich für die Lösung des Problems verwenden lässt. Durch parametrische Diagramme und Constraints lassen sich in SysML Systemeigenschaften und Abhängigkeiten von Parametern untereinander beschreiben. Dieses, eigentlich mit dem Ziel kritische Systemparameter zu identifizieren eingeführte Konzept, wird nun hier adaptiert.

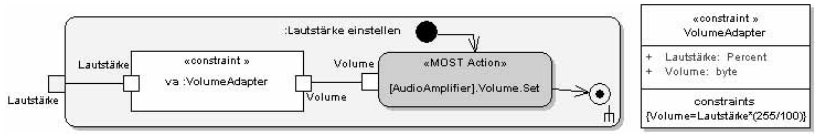

Abbildung 5: Lautstärke einstellen mit nötiger Parameteranpassung und zugehörigem Constraint

Abbildung 5 zeigt die Verknüpfung der Aktivität `Lautstärke einstellen` der funktionalen Ebene mit den entsprechenden MOST Aktionen der technischen Ebene. Auf der funktionalen Ebene wird die Lautstärke in Prozent angegeben. Der entsprechende Wert im MOST System ist allerdings ein Byte mit dem Wertebereich 0-255. Daher ist eine entsprechende Umrechnung zwischen beiden Einheiten nötig. Unter Verwendung der in SysML definierten Konzepte ergibt sich der dargestellte ConstraintBlock und dessen Verwendung innerhalb des Aktivitätsdiagrammes als ConstraintProperty. Der Constraint Block definiert die notwendige Formel um zwischen dem Parameter `Volume` und `Lautstärke` umzurechnen. Eine entsprechende *Constraint Engine* (Abbildung 3), löst dann bei der Erzeugung der spezifischen Testfälle die Constraints auf.

5 Verwandte Arbeiten

Briand und Labiche [BL02] beschreiben ein Verfahren zur Gewinnung von Testfällen aus UML1.x Modellen mit Aktivitäts- und Sequenzdiagrammen. Im hier beschriebenen Ansatz werden durchgängig UML2 Aktivitätsdiagramme für die Modellierung verwendet, insbesondere auch für die Verhaltensmodellierung und Beschreibung der Testfälle. Eine Generierung von Systemtestfällen aus Aktivitätsdiagrammen beschreiben Hartmann et. al. in [HVFR01]. Dort werden speziell auf PC Applikationen zugeschnittene Tests generiert. Mit der hier angewandten Trennung von funktionalem und spezifischem Verhaltensmodell können mit ein und dem selben Generator Testfälle für alle Systeme erzeugt werden, für die entsprechende Verknüpfungen zwischen beiden Modellteilen vorhanden sind. Eine Fallstudie des TT-Medal Consortium [SHB05] untersucht und demonstriert die Automatisierbarkeit von System- und Systemintegrationstests mit Hilfe der Testbeschreibungssprache TTCN-3 am Beispiel eines MOST Car Multimedia Systems. Die dort evaluierten Konzepte zur Testautomation können auch mit dem hier beschriebenen Ansatz angewandt werden. In der Studie werden die Testfälle per Hand implementiert, während hier die Testfälle durchgängig aus UML und SysML Modellen für die MOST Geräte generiert werden.

6 Zusammenfassung und Ausblick

Um die Qualität moderner Systeme im Car Multimedia Bereich zu sichern, ist ein erhöhter Test und damit nach den bisherigen Arbeitsweisen einhergehender Zeit-, Personal- und Kostenaufwand notwendig. In diesem Papier wurden daher zunächst Konzepte vorgestellt, die diese Situation durch Generierung von Testfällen aus einem Systemmodell verbessern sollen. Wichtig dabei war Systemtestfälle in verschiedenen Projekten wieder verwenden zu können. Eine zentrale Rolle spielt dabei die vorgestellte 3-teilige Architektur des Systemmodells mit HMI, einer projektunabhängigen funktionalen und einer projektabhängigen technischen Ebene. Die Modellierungssprachen der verschiedenen Ebenen sind dabei durch UML2-Profile auf die Anwendungsdomäne zugeschnitten. Neben eigenen Profilen wurden dabei insbesondere Konzepte von SysML übernommen. Anhand eines Beispiels aus dem Telematik Bereich wurde demonstriert, wie sich die funktionale und die technisch-projektspezifische Ebene verknüpfen lassen um zu halb- oder vollautomatischen Testfällen zu kommen. Durchgängige Verwendung fanden dabei UML2 Aktivitätsdiagramme.
Zur Zeit wird an der Implementierung eines Plug-In für das UML Werkzeug Enterprise Architect 6.x [Spa06] gearbeitet, welches die Generierung und Constraint-Auflösung vornimmt. Damit soll die beschriebene Umsetzung zwischen funktionaler und technischer Ebene an einem konkreten MOST Systembeispiel, insbesondere auch die Parameteranpassung, evaluiert und ablauffähige XML Testmodule für CANoe.MOST aus den Modellen generiert werden. Mit steigender Werkzeugunterstützung des sehr neuen SysML Standards wird sich zukünftig zeigen, inwieweit die vorgestellten Konzepte auch ohne ein solches Plug-In mit den Bordmitteln eines Werkzeuges realisierbar sind.
Zukünftig ist angedacht auch Alternativpfade auf technischer Ebene explizit zu modellie-

ren, die eine Aktivität der funktionalen Ebene semantisch gleich umsetzen. Der Testfallgenerator kann dann entscheiden welchen der alternative Kontrollflüsse er in einen Testfall einbezieht. Dadurch lassen sich mehr Testfälle gewinnen und die Testabdeckung auf der technischen Ebene erhöhen.
Im einem nächsten Schritt soll dann die automatische Generierung von Testfällen der funktionalen Ebene umgesetzt werden, mit dem Ziel einen durchgängigen automatisierten (Systemtest-) Prozess für die Domäne Car Multimedia zu erreichen.

Literatur

[3SO04] 3SOFT. Komplexe HMIs generieren mit GUIDE, Oktober 2004.

[AS06] Oliver Alt und Markus Schmidt, Hrsg. *Derivation of System Integration Tests from Design Models in UML*, Bilbao, Juli 2006. ECMDA workshop IMDT 06.

[BL02] Lionel Briand und Yvan Labiche, Hrsg. *A UML-Based Approach to System Testing*. Software Quality Engineering Laboratory Carleton University, Carleton University, June 2002.

[HVFR01] Jean Hartmann, Marlon Vieira, Herb Foster und Axel Ruder, Hrsg. *UML-based Test Generation and Execution*. Siemens Corporate Research, Inc., 2001.

[Kel05] Steven Kelly. Software-Modellierung ohne Kunstgriffe: Mit domönenspezifischer Modellierung zu hundertprozentiger Code-Generierung. *Elektronik*, 1:64–69, Januar 2005.

[LS06] Friedemann Ludwig und Frank Salger. Werkzeuge zur domänenspezifischen Modellierung. *OBJEKTspektrum*, 3(3):16–20, Mai/Juni 2006.

[MOS03] MOST Cooperation. *MOST FunctionBlock AudioDiskPlayer*. MOST Cooperation, rev. 2.4. Auflage, 9 2003.

[MOS04] MOST Cooperation. *MOST Specification Rev. 2.3 08/2004*. MOST Cooperation, rev. 2.3. Auflage, August 2004.

[OMG04] OMG, Hrsg. *Unified Modeling Language: Superstructure version 2.0*. OMG, OMG, Oktober 2004.

[SHB05] Michael Schmidt, Jens Herrmann und André Baresel, Hrsg. *Domain Specific Test Systems Automotive Case Study Description, 3rd release*. TT-MEDAL Consortium, TT-MEDAL Consortium, Mai 2005.

[Spa06] Sparx Systems Pty Ltd. *Enterprise Architect 6.x*, 2006. `http://www.sparxsystems.com`.

[Sys06] SysML Partners. *Systems Modeling Language (SysML) Specification*. SysML Partners, 1.0 draft. Auflage, April 2006.

[Vec05] Vector Informatik. CANoe.MOST Version 5.2 - Simulation- and testtool for CAN and MOST systems, `www.vector-informatik.de`, 2005.