Modultest im Embedded Bereich

Konzept für einen Testrahmen für Rose Realtime Software

Autor: Norbert Fleischmann

Norbert Fleischmann studierte Elektrotechnik an der Universität Erlangen-Nürnberg. Nach dem Studium war er in der Telekommunikationsindustrie in den Bereichen Design, Entwicklung und Test tätig. Nach der erfolgreichen Durchführung zahlreicher Entwicklungsprojekte im Bereich der Sprachverarbeitung übernahm er die Leitung der Software-Abteilung im Bereich digitaler Signalprozessoren. Nach Rückkehr in die technische Laufbahn arbeitete er im Bereich der objektorientierten Entwicklung von digitaler Signalprozessorsoftware. Sein aktueller Arbeitsschwerpunkt liegt in der Schulung und Beratung objektorientierter Softwareprojekte in den Bereichen Analyse und Design mit UML, Anforderungsmanagement, Implementierung und Test von Embedded Systemen.

Kontakt:

method park Software AG
Wetterkreuz 19a
91058 Erlangen
Tel. 09131-9 72 06-281
Fax 09131-9 72 06-280
info@methodpark.de
www.methodpark.de

1 Überblick

Eingebettete Software findet sich zunehmend in vielfältigen Produktbereichen. Vor allem in der Automobilbranche ist ein deutlicher Trend von immer zahlreicheren und immer komplexeren Softwaresystemen zu verzeichnen. Da jedoch viele Firmen in diesem Bereich traditionell aus der Hardwareentwicklung kommen, sind die Entwicklungsprozesse und -methoden in den Softwareabteilungen nicht sehr stark ausgeprägt. Aus diesem Grund versuchen diese Firmen ihre Prozesse mit Hilfe von Beratungsfirmen zu definieren oder zu optimieren. In diesem speziellen Fall sollte der Softwaretest, welcher schon als Prozess definiert war, zum Leben erweckt werden.

Hier werden Überlegungen zu einer Testumgebung für einen Modultest vorgestellt und durch Resultate validiert.

1.1 Projektumfeld

Die Software wird im beschriebenen Projekt mit Rational Rose Realtime (RoseRT) entwickelt. Das zugrunde liegende Konzept von RoseRT ist die ROOM (Real-Time Object-Oriented Modeling) Methode. Diese Methode erweitert die UML 1.x um folgende Konzepte und Elemente:

- Capsules
  Komplexe, physikalische, eventuell verteilte Architekturobjekte, welche mit ihrer Umgebung über ein oder mehrere signalbasierte Grenz-objekte, sogenannte Ports, inter-agieren.
- Ports
  Schnittstelle einer Capsule zur Außenwelt. Ports realisieren Protokolle.
- Protokolle
  Definition der gültigen Informationsflüsse (Signale).

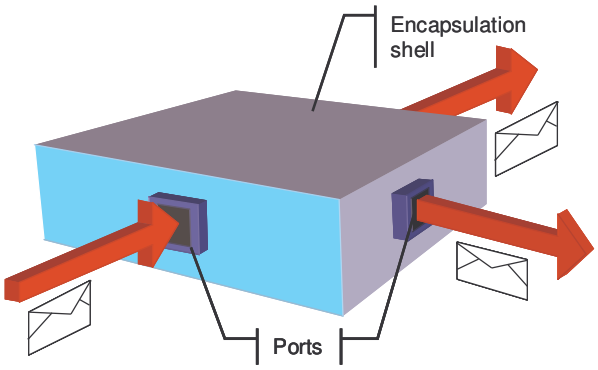

Abbildung 1: Schematische Darstellung einer Capsule

Das dynamische Verhalten einer Capsule wird in Form von hierarchischen Zustandsautomaten spezifiziert. Mit Hilfe einer Codegenerierung wird die gesamte Software aus dem RoseRT-Modell heraus erzeugt. Da es sich in diesem Projekt um eine Steuerung handelt, welche später auf einem kleinen Microcontroller ablaufen soll, wurde hier die Zielsprache C gewählt.

1.2 Aufgabenstellung: Modultest

Da es bei einem Modultest darum geht, einzelne Teile eines Softwaresystems isoliert von einander zu testen, muss man sich erst einmal Gedanken darüber machen, was ein Modul ist. In klassischer objektorientierter Softwareentwicklung würde man sofort auf Begriffe wie Subsysteme, Pakete, Klassen und Methoden treffen. Da jedoch in RoseRT Systemen das dynamische Verhalten in den Zustandsautomaten einer Capsule versteckt ist, bietet es sich an, eine Capsule als ein Modul zu betrachten.

1.3 Zustandsbezogener Test

Da sich das Verhalten einer Capsule als Zustandsautomat beschreibt, wird man als Testmethode sofort an zustandsbezogenen Test denken. Das heißt, man muss eine Kombination von Zuständen und Zustandsübergängen testen. Als Erweiterung wird man in RoseRT zusätzlich noch die Schnittstellen einer Capsule, die Ports und Protokolle abtesten. Weiterhin darf man die sogenannten Action-Codes nicht vergessen. Als Action-Code bezeichnet man Zielsprachenanweisungen, welche z.B. hinter

Zustandsübergängen hinterlegt sind. Diese Anweisungen veranlassen unter anderem das Senden von Signalen, welche wiederum zu Zustandsübergängen führen können.

In diesem Zuge wird man sich auch Gedanken machen, in welchem Umfang man die Software testen möchte, das heißt, über Überdeckungsmaße, In diesem Kontext tauchen folgende Maße auf:

- Zustandsüberdeckung
- Transitionsüberdeckung
- Signalüberdeckung
- White-Box-Überdeckungsmaße (Tool-Unterstützung notwendig, z.B.: Rose Realtime Test)

In diesem Projekt wird im ersten Ansatz eine vollständige Zustands-/Transitionsüberdeckung angestrebt.

2 Testumgebung

Um die Testfälle zu implementieren und ausführen zu lassen, benötigt man eine Testumgebung, welche es gestattet entsprechende Eingangssignale zu erzeugen und die Ausgänge einer Capsule zu überwachen.

Da RoseRT den Anspruch erhebt, auch für dieses Problem eine Lösung zu haben, wurde zuerst untersucht, in wie weit diese Möglichkeiten ausreichen, um Testfälle zu implementieren. Eine Möglichkeit in RoseRT ist der sogenannte „Quality Architect“. Er erlaubt es, Testfälle in Form von UML-Sequenzdiagrammen zu erstellen. Diese Sequenzdiagramme werden dann von RoseRT in Testfälle umgewandelt. Jedoch stößt man sehr schnell an die Grenzen des Systems, da man keine Schleifenkonstrukte zur Verfügung hat. Weiterhin ist es nicht möglich, einen einmal definierten Testablauf wiederholt für verschiedene Eingangsparameter ablaufen zu lassen. Eine weitere Möglichkeit ist die Verwendung des RoseRT Debuggers. Jedoch ist hier eine Testautomatisierung nicht vorgesehen, so dass auch diese Lösung ausscheidet.

Im weiteren Verlauf der Untersuchung hat sich eine Lösung herauskristallisiert, welche nun näher dargestellt werden soll.

3 RTUnit

Die Grundidee besteht darin, RoseRT zu benutzen, um Testfälle zu formulieren und ablaufen zu lassen, so dass die Einarbeitungszeit für RoseRT Entwickler so gering wie möglich ist. Die Testumgebung besteht aus zwei Teilen.

- Teststeuerung
- Testmodell

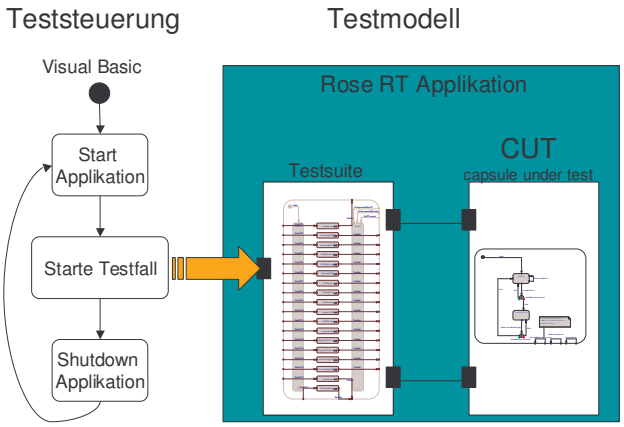

Abbildung 2: RTUnit

3.1 Teststeuerung

Die Teststeuerung hat die Aufgabe sicherzustellen, dass jeder Testfall dieselben Anfangsbedingungen hat. Dazu wird die erzeugte RoseRT-Applikation, welche das gesamte Testmodell enthält, für jeden Testfall einzeln aufgerufen und ein Testfall gestartet.

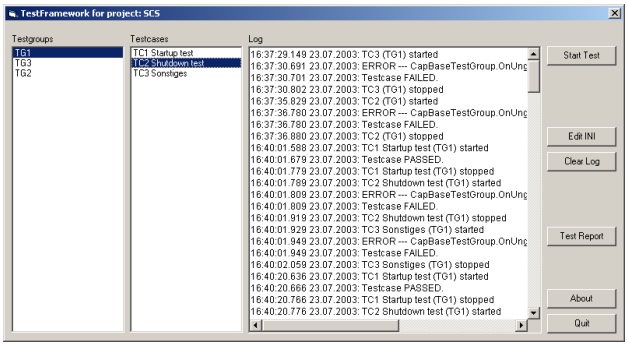

Abbildung 3: Die Teststeuerung

Eine weitere Aufgabe ist das Verwalten und Protokollieren von Testabläufen, welche in Testgruppen geordnet sind.

3.2 Testmodell

Das Testmodell beinhaltet die eigentlichen Testfälle. Es besteht aus einer hierarchischen Capsule, welche die Testfälle als Zustands-automaten realisiert.

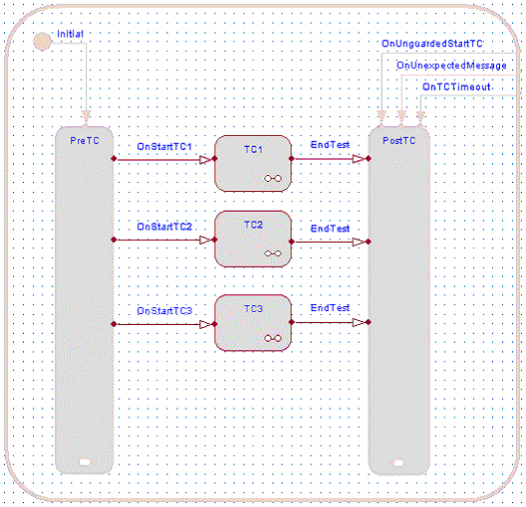

Abbildung 4: Das Testmodell

Diese Capsule wird dann mit dem Testling verbunden. Jeder Testfall ist als Zustand in dieser Capsule implementiert. Jeder Testfall selbst kann natürlich beliebig viele Hierarchieebenen besitzen, so dass auch sehr komplexe Testfälle realisiert werden können.

4 Testmatrix

Um den Testern ein Hilfsmittel an Hand zu geben, das ihnen gestattet einen Testfall so zu entwerfen, dass er eine möglichst hohe Testabdeckung erreicht, wurde ein Skript für RoseRT erstellt, das eine Testmatrix erzeugt. Dieses Skript erzeugt aus den Modell-informationen eine Excel-Tabelle für alle Zustände und deren abgehende Zustands-übergänge.

Model: C:\Programme\Rational\Rose RealTime\Examples\Models\C++\TrafficLights\TrafficLights.rtmdl
Capsule: TrafficLightAustrian
No of States: 14
No of Transitons: 20
Total Coverage: 65,00%

State	Transition	Coverage	TC1	TC2	TC3	
initializing(1)	IGoFirst	1	1			
initializing(1)	IGoLater	2		1	1	
C:HavePriority(2)	False	1	1			
C:HavePriority(2)	True	1				
flashingAmber(2)	msgFromOtherLight	1	1			
on(3)	timeout	1		1		
on(3)	msgFromOtherLight					
off(3)	timeout	1	1			
off(3)	msgFromOtherLight					
green(2)	greenTimeout	1	1			
transitioning(2)	timeout					
transitioning(2)	timeout2					
amber(3)	timeout	1	1			
flashingGreen(3)	timeout	1			1	
lightOn(4)	timeout	1			1	
lightOff(4)	timeout	1			1	
C:flashedEnough(4)	False	1			1	
C:flashedEnough(4)	True					
redAndAmber(3)	timeout					
red(2)	otherRed					

Abbildung 5: Beispiel Testmatrix

Durch manuelles Eintragen, der in einem Testfall abgetesteten Zustände und Zu-standsübergänge, berechnet die Tabelle die entsprechende Testabdeckung, in diesem Beispiel die Testabdeckung für Transitions-überdeckung

5 Fazit und Ausblick

Es hat sich in diesem Projekt gezeigt, dass es mit einem geeigneten Konzept möglich ist, RoseRT als Testumgebung für einen Modultest zu verwenden. Dieses Konzept hat selbst unter den Entwicklern eine hohe Akzeptanz erreicht, da sie ihre gewohnte Arbeitsumgebung nicht verlassen müssen, um Testfälle zu erstellen. Dadurch eignet sich dieses Konzept auch hervorragend für den agilen „test-first“ Ansatz. Eine Erweiterung des Konzepts um einen Nachweis von White-Box-Testabdeckungsmaße wäre eine mögliche Weiterentwicklung, wozu die Hilfe von entsprechenden Werkzeugen zurate gezogen werden sollte.