METHOD > PARK

Modultest im Embedded Bereich

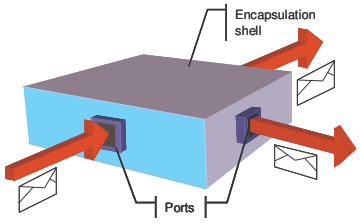
Konzept für einen Testrahmen für Rose Realtime Software

Norbert Fleischmann
Norbert.Fleischmann@methodpark.de

METHOD > PARK

> Projektumfeld

- Rational Rose Realtime Entwicklung
  - Capsules
    - Komplexe, physikalische, eventuell verteilte Architekturobjekte, welche mit ihrer Umgebung über ein oder mehrere signalbasierte Grenzobjekte, sogenannte Ports, interagieren.
  - Ports
    - Schnittstelle einer Capsule zur Außenwelt:
    - Ports realisieren Protokolle.
  - Protokolle
    - Definiert den gültigen Informationsfluss (Signale).
    - „vertragliche Verpflichtungen“ zwischen Capsules



- Code Generierung
- Embedded Steuerung
  - Sensorik, Aktoren
- Zustandsautomaten

METHOD > PARK

Aufgabenstellung: Modultest

- Definieren und Erstellen eines Testkonzepts und einer Testumgebung für den Modultest, sowie Testfallerstellung und Testdurchführung.

- Definition Modultest
  - „Test eines einzelnen Moduls eines modularisierten Softwaresystems.“ [BASIS2003]

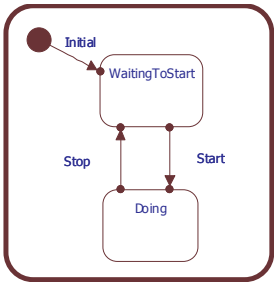
  - Was ist ein Modul in Rose RT ?
    - Package ?
    - Capsule ?
    - Klasse ?
    - Methode ?

METHOD > PARK

Modultest - Zustandsbezogener Test

- Testfokus
  - Innenleben der Capsules
    - Zustandsautomaten
      - Zustände
      - Transitionen
      - Action-Code
    - Datentypen

  - Schnittstellen der Capsules
    - Ports
    - Protokolle
      - Signale

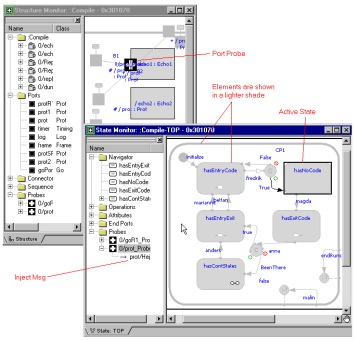


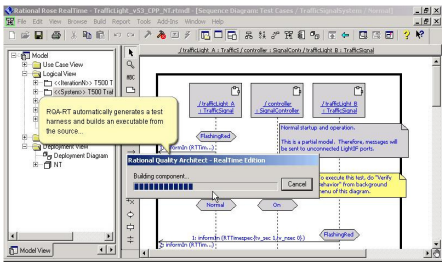
- Überdeckungsmaße
  - Zustandsüberdeckung
  - Transitionsüberdeckung
  - Signalüberdeckung
  - White-Box-Überdeckungsmaße (Tool-Unterstützung notwendig, z.B.: Rose Realtime Test)

METHOD > PARK

Testunterstützung in Rational Rose Realtime

- Quality Architect in RoseRT
  - Verwendung von Sequenz-Diagrammen zur Testfall-Erstellung
  - Nur triviale Testfälle modellierbar (keine Schleifen etc.)
  - Nur explizite Parameter möglich



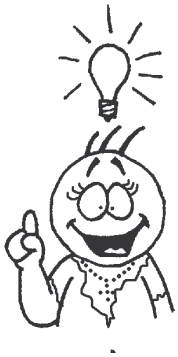


- Rational Rose Debugger
  - Manuelles Injizieren von Signalen
  - Protokollieren von Zustandswechseln

METHOD > PARK

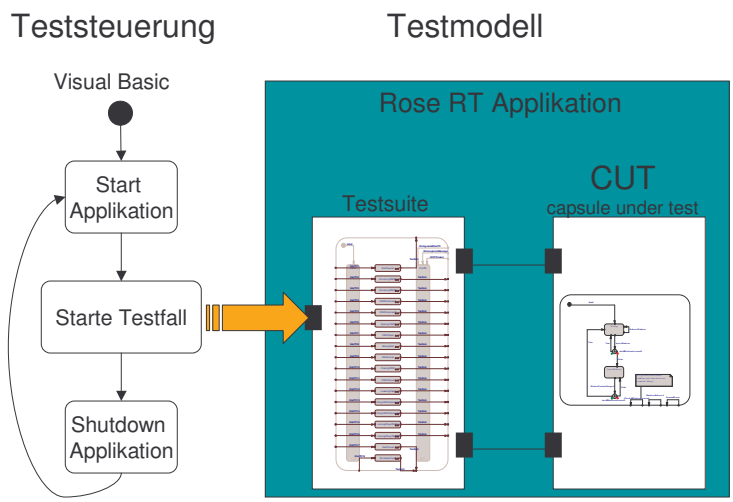
Testumgebung: „Ideen“

- Manuell Testen
  - Verwenden des RoseRT Debuggers zum Erzeugen von Teststimuli
  - Keine Testautomatisierung möglich
- Zugriff auf generierten Quellcode
  - Entwurf einer Testumgebung außerhalb von RoseRT. Test des generierten Quellcodes
  - Hoher Aufwand, die Testumgebung zu erstellen
  - Unübersichtliche und komplexe Testumgebung
  - Änderung im Modell kann komplette Änderung der Testumgebung nach sich ziehen
- Quality Architect von Rose RT
- **RoseRT Testumgebung**
  - Entwurf einer Testumgebung in RoseRT



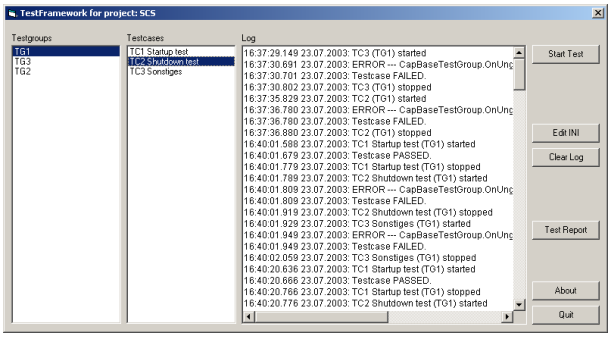
METHOD > PARK

> RTUnit



METHOD > PARK

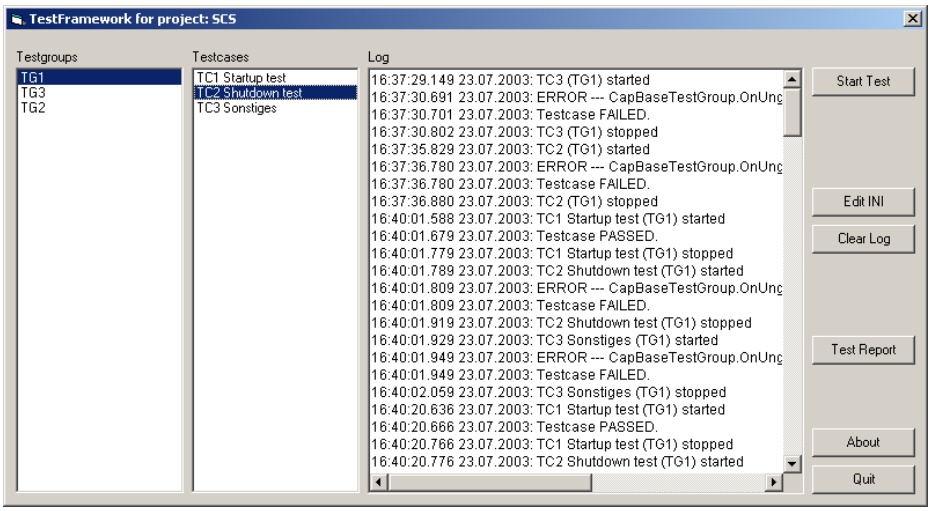
> Teststeuerung



- Sicherstellen eines definierten Anfangszustands für alle Tests
- Ansteuerung von Testgruppen
- Protokollierung von Testabläufen
- Erstellen eines Testreports

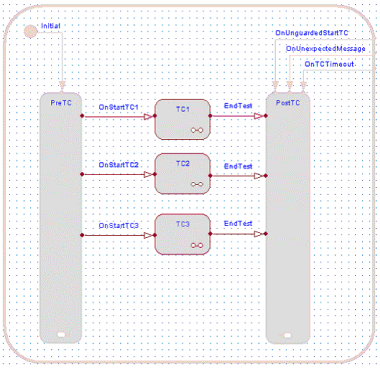
METHOD > PARK

> Teststeuerung



METHOD > PARK

> Testmodell

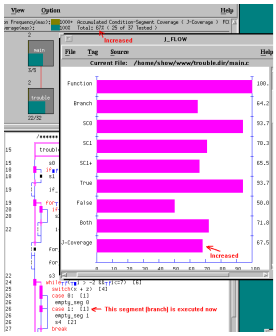


- Eine Testgruppe ist eine Capsule
- Ein Testfall ist ein Zustand in der Testgruppe
- Testfälle können durch hierarchische Zustandsautomaten realisiert werden.
- Vererbung kann genutzt werden

- Testsuite stellt den Start des angeforderten Tests sicher
- Testsuite stellt Basisfunktionalität jedem Testfall zur Verfügung
  - Logging
  - Timer

METHOD > PARK

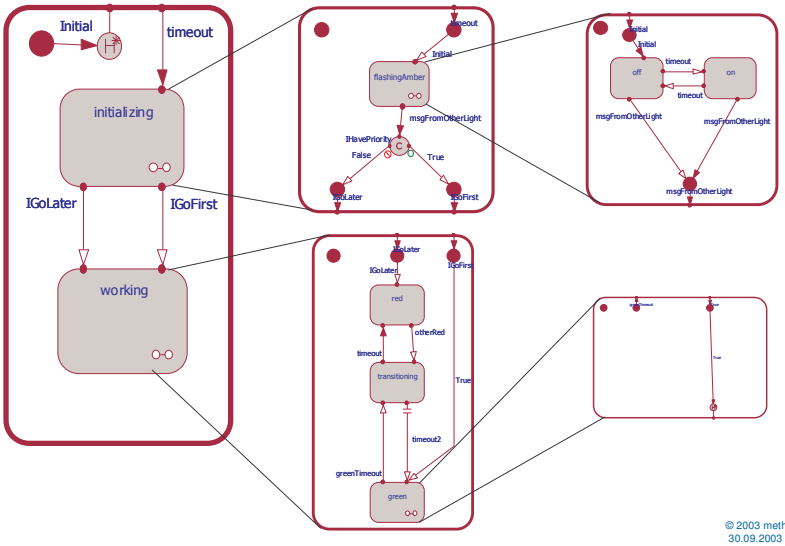
> Testabdeckung



- Zustandsüberdeckung
  - Jeder Zustand wird mindestens einmal eingenommen
- Transitionsüberdeckung
  - Jede Transition wird mindestens einmal gegangen
- Event – Überdeckung
  - Jedes Event, welches zu einer Transition führt, wird getestet
- Test des „Actioncode“
  - Test des von Hand implementierten Actioncodes in einer Transition oder in „Entry/Exit“-Funktionen
- Kombinatorik von Zustandswechseln

METHOD > PARK

> Testmatrix: Beispiel



METHOD > PARK

> Tool zu Erstellung einer Testmatrix

- Rose RT script
  - Visual Basic ähnliche Scripting-Sprache
  - Zugriff auf alle Modell-Elemente
- Erzeugt aus dem Implementierungsmodell für eine ausgewählte Capsule eine Testmatrix
  - Alle Zustände (hierarchisch) der Capsule
  - Alle abgehenden Transitionen
- Gedacht als Hilfe für den Testentwickler, um nicht den Überblick zu verlieren.

METHOD > PARK

> Testmatrix für Transitionsüberdeckung

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 IHavePriority(2)	False	1	1			
C IHavePriority(2)	True	1		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					

METHOD > PARK

Lessons Learned

- Rose RT eignet sich nicht nur für Softwareentwurf, sondern auch für die Testfallerstellung.
- Diese Testumgebung wird auch schon in der Entwickung eingesetzt, um Caspules zu testen, bevor das Gesamtsystem aus implementiert ist.
  - Die Entwickler können ihre gewohnte Entwicklungsumgebung benutzen um Testfälle zu erstellen.
- Reale Software hat eine Komplexität, welche eine Testautomatisierung erfordert.
- Das Testmodell übersteigt die Komplexität des zu testenden Modell um einiges

METHOD > PARK

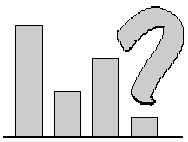
Ausblick

- Integration von White-Box-Testabdeckungswerkzeuge, um die Testabdeckung auf Quellcode-Ebene nachweisen zu können.
  - Z.B.: Rose Realtime Test
- Automatisches Messen der Zustands-und Transitions-Abdeckung beim Testablauf.
- Geeignet für den agilen „test-first“ Ansatz, da die Entwickler in ihrer gewohnten Entwicklungsumgebung arbeiten können.

METHOD > PARK

> Literatur

- [BASIS2003] – Basiswissen Softwaretest – A. Spillner, T. Linz, dpunkt.verlag , 2003



Noch Fragen ?