

SCADE

Safety Critical Application Development Environment

Steven Smyth

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

10. Jun 2010



Gliederung

- 1 SCADE und SCADE-Modellierung
- 2 SCADE und Lustre
- 3 Code-Erzeugung und Validierung
- 4 Beispiel

Motivation

Safety-Critical

- Sicherheitskritische Aufgaben
- Ausfall hat schwerwiegende Folgen
- Systeme werden komplexer

Verbesserungen im Entwicklungsprozess

- Bessere Kommunikation zwischen Ingenieuren und Softwareentwicklern
- Gute Testumgebung
- Kostensenkung

Safety Critical Application Development Environment

SCADE

- Die Sprache SCADE
 - Synchrone Sprache
 - Erweiterung von Lustre
 - Arrays, Datentyp-Konvertierung, Safe State Machines

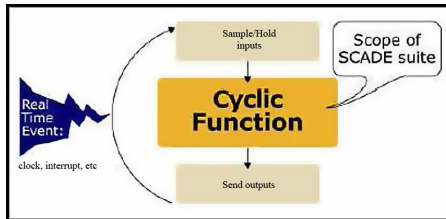


Abbildung: Zyklisches Ausführungsmodell von SCADE ¹

¹aus F. X. Dormoy. Scade 6 - A model based solution for safety critical software development.

Safety Critical Application Development Environment

SCADE

- Die Entwicklungsumgebung SCADE
 - Grafischer Editor
 - Design Verifier
 - Simulationsumgebung
 - Quellcode-Generator

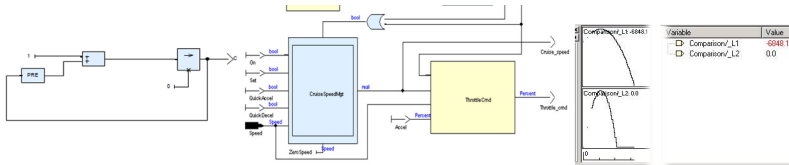


Abbildung: SCADE-Elemente

Modelle in SCADE

SCADE ist eine modellbasierte Entwicklungsumgebung

Correctness-by-Construct

- Code entspricht Modell
- fehlerfrei
- hoher Standard
- Testen auf niedriger Ebene überflüssig

Modelle in SCADE

Blockdiagramme

Modellierung von kontinuierlichen Systemen

- überwachen Umgebung durchgehend
- reagieren auf Veränderungen in der Umgebung

Safe State Machines

Modellierung von diskreten Systemen

- reagieren auf Ereignisse
 - z.B. Interrupts, Benutzereingaben

Blockdiagramme

Darstellung

- Blöcke = Komponenten des Systems
- Verbindungen = Interaktion des Systems

Hierarchie

- hierarchische Schichten
- kompositionaler Ansatz
- übersichtlich
- Design
 - keine komplexen Auswertungen
 - Inhalt geht in übergeordneten Block über

Blockdiagramme - Beispiel 1

Count node

- Erhöht jeden Zyklus um +1

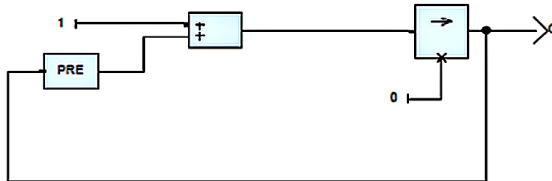


Abbildung: SCADE Blockdiagramm: Count node

Blockdiagramme - Beispiel 2

Resetable count node

- übergeordnet zu Count node
- resetable

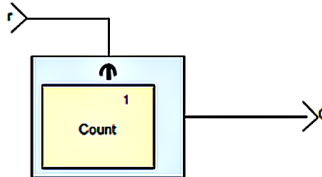


Abbildung: SCADE Blockdiagramm: Resetable count node

Safe State Machines

Ursprung

- Endliche Automaten
- Esterel und SyncCharts

Darstellung

- Box = Zustände
- Verknüpfung = Transition
- Beschriftung = Input / Output

Safe State Machines

Hierarchie

Ein Zustand kann entweder ein Zustand oder ein Macrostate sein

- Macrostates
 - State Machine
 - Sammlung nebenläufiger State Machines

Transitionen

- weak-delayed abortions
 - bricht im nächsten Zyklus ab
- strong abortions
 - bricht sofort ab
- synchronized transitions
 - bricht ab, sobald Zustand `final` erreicht ist

Safe State Machines - Beispiel

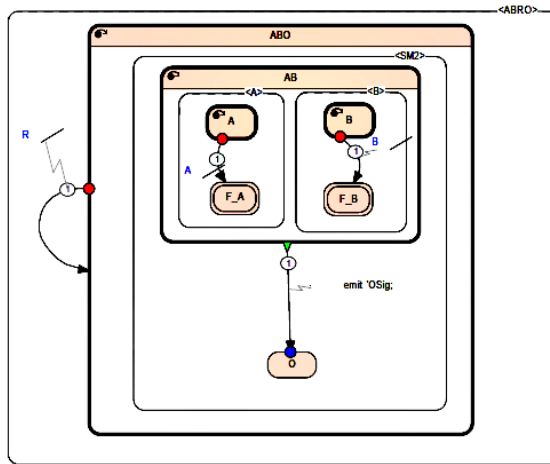


Abbildung: SCADE SSM: ABRO

Modell-Kombination

Kombination

- Safe State Machines in Blockdiagrammen
- Blockdiagramme in Safe State Machines

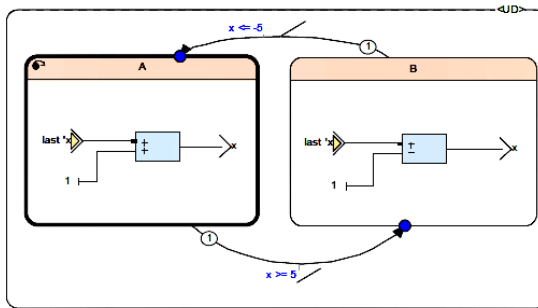


Abbildung: SCADE Blockdiagramme in State Machines

Lustre

Lustre in SCADE

- Synchrone Sprache
- Lustre-Code wird aus Modell erzeugt
- Keine Notwendigkeit direkt mit Lustre zu arbeiten
- sicherer Code

Erweiterungen

- Arrays
- Datentyp-Konvertierung
- Safe State Machines

Lustre - Beispiele

Lustre: Count node

```
1 c = 0 -> (1 + pre c)
```

Lustre: Resetable count node

```
1 c = (restart count every r)()
```

Lustre: upDown node

```
1 node UpDown () returns (x: int last = 0)
2 let
3   automaton UD
4     initial state A
5       x = last 'x + 1;
6       until if x >= 5 restart B
7     state B
8       x = last 'x - 1;
9       until if x <= 5 restart A
10    returns x;
11 tel
```

Codeerzeugung

Zielsprachen

- Ada
- SPARK Ada
- Qualifiable C
- Standard C

Automatische Codeerzeugung

Keine Notwendigkeit direkt in der Zielsprache zu programmieren

Erforderlichen Angaben:

- root node
- Zielsprache

Codeerzeugung

Regeln zur Codeerzeugung

- kein überflüssiger Code
 - Repräsentation des Modells
 - Komplette Neu-Übersetzungen
- kein *dead code*
- keine Zeigerarithmetik
- statische Speicherzuweisung

Erhöhung der Sicherheit

- Wahl der Zielsprache
 - Sprach-Einschränkungen
- keine manuellen Veränderungen am Code
 - werden nicht ins Modell übernommen

Validierung

Probleme beim Testen

- Testfälle oft unvollständig
 - nur bestimmte Bereiche
 - vordefinierte Szenarien
- Wann ist die Testphase abgeschlossen?
- Fehler oft unintuitiv

Validierung

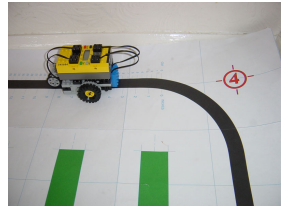
Hilfestellungen

- Code ist *correct-by-construct*
- *bug-chasing mode*
 - Überprüfung mit zufälligen Eingabesequenzen
- *Design Verifier*
 - *model-checking*
 - *Observer nodes*
- Simulation
 - *step-by-step* Simulation
 - Listen und Zeit-Wert-Diagramme

Beispiel - Mindstorms

Mindstorms

- Lego Mindstorms Roboter mit zwei Lichtsensoren
- korrigiert Fahrrichtung
- Motor hat Zustand Fwd oder Stop



Beispiel - Mindstorms

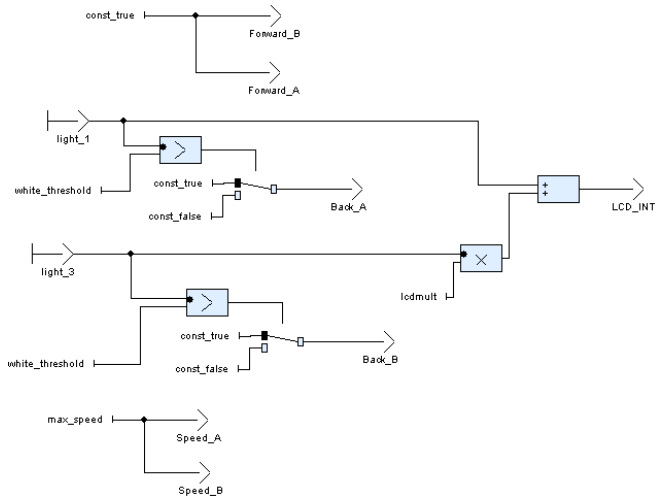


Abbildung: SCADE Blockdiagramm des Pathfinders

Beispiel - Mindstorms

Pathfinder in Lustre

```
1 node followline (touch_1, touch_2, touch_3: bool;}  
2   light_1, light_2, light_3: int;  
3   rotation_1, rotation_2, rotation_3: int)  
4 returns (Forward_A, Back_A: bool;  
5   Speed_A: int;  
6   Forward_B, Back_B: bool;)  
7   Speed_B: int;  
8   LCD_INT: int);  
9 let  
10   Forward_A = true  
11   Forward_B = true  
12   Back_A = if (light_1 > 40) then true else false  
13   Back_B = if (light_3 > 40) then true else false  
14   Speed_A = 255  
15   Speed_B = 255  
16   LCD_INT = (light_1 * 100) + light_3  
17 tel
```

Fazit

Fazit

- Notwendigkeit für Entwicklungshilfen wächst
- *state-of-the-art*
- weitere Entwicklung in Aussicht

Fragen?

- Vielen Dank
- Noch Fragen?