Embedded Systems - Jian Jia Chen
Zusammenfassung

Maximilian Springenberg
TU-Dortmund WS18/19

0 Teil 1 - ES CPS

0.1 ES

Ein eingebettetes System (ES) ist ein System, das Informationen verabeitet und in ein größeres Produkt eingebettet ist. Ein ES stellt also Software bereit, die durch die Verarbeitung von (Sensor-) Signalen/ Informationen ein Produkt im Features, bzw. Verbesserungen ergänzt.

0.2 CPS

Cyber-Physical-Systems (CPS) sind im Wesentlichen Systeme, die zuzüklich auf Kommunikation ausgelegt sind. (smart-home, smart-...)

0.3 Charakteristika

0.3.1 Dependability

CPS und ES müssen verlässlich sein. Deshalb definieren wir folgende Güten:

$$\begin{aligned}
\text{Reliability } R(t) &= \text{Wahrscheinlichkeit, dass das System zu} t = 0\text{funktioniert} \\
\text{Maintainability } M(d) &= \text{Wahrscheinlichkeit, dass das System} d\text{Zeiteinheiten nach einem Fehler funktioniert} \\
\text{Availability } A(t) &= \text{Wahrscheinlichkeit, dass das System zur Zeit } t\text{funktioniert} \\
\text{Safety} &= \text{Das System verursacht keinen Schaden} \\
\text{Security} &= \text{Das System bedient sich authentischer und vertraulicher Kommunikation}
\end{aligned}$$

0.3.2 Effizienz

Energieeffizienz:
Für ES/ CPS ist die Energieeffizienz von bedeutung, die sie oftmals auf Batteriebetrieb angewiesen sind. Energieeffiziente Prozessoren verlangen jedoch eine aufwendigere Implementierung des ES/ CPS. Hierbei gilt es für Produzenten/ Firmen einen Mittelweg zu finden.

real-time constraints:
Wird bei z.B. einem Auto ein real-time constraint nicht eingehalten können die Folgen verherend sein. Dies wäre auch ein Beispiel für einen harten real-time constraint. Alle nicht in Katastrophen resultierende real-time constraints sind soft.

Weite Faktoren hinsichtlich der Effizienz sind: Gewicht der Hardware, Kosten der Hardware und Entwicklung, Code-Größe/ Länge

0.4 Anwendung

ES/ CPS finden Anwendung in:

1. Transport:
   Die Automobilindustrie verwendet ES mit ABS, ESP, etc., die Flugzeugindustrie benutzt ES für Flugkontrolle, Kollisionsvermeidung, Autopiloten, etc.. Auch Schiffe und Züge verwenden ES für Sicherheits- und Kontroll/ Navigations Features.

2. Logistik:
   Die Logistik verwendet ES für Radio-Frequency-Identification, mobile Kommunikation, etc..

3. Fabriken:
   Fabriken verwenden ES in 'social machines', Maschinen de sich selbst konfigurieren und/ oder distributieren.

4. Structural Safety:
   Darunter fällt Regulation des Wasserstandes eines Damms, Überwachung von Brücken/ Vulkanen, sowie die Neigung von Hochhausern bei Erdbeben.

5. Smart Home:
   Häuser können ES für zero energy buildings, safety/ security, comfort, ambient assited living (selbst regulierende Fenster etc.) benutzen.

1 Teil 2 - Spezifikations- und Modellierungssprachen

1.1 Deadlocks

1. mutual exclusion
2. no preemption
3. holding recources
4. circular waiting

1.2 Imperative von Neumann Modell

Das von Neumann Modell behandelt die Prinzipien von Operationen in Computern.

1.2.1 Probleme

Es gibt zugang zu geteiltem Speicher, dies führt zu Anomalien, die mittels Mutex, Semaphoren umgangen werden müssen. Priority-Inversion kann auch aus dem geteilten Speicher folgen. Timing und Terminierung kann nicht garantiert/ entschieden werden.

Beispiel Java, als instanz einer Sprache nach dem von Neumann Modell:

Pros	Cons
sauber/ sicher	Größe der Runtime Libraries
multi-threading support	Kein direkter Zugang zu spezifischer Hardware
Platform unabhängig	Garbage-Collector nicht absehbar
	non-deterministic dispatcher (mehrere Methoden mit gleichen Namen)
	Performanz Probleme
	Echtzeitanforderungen können nicht überprüft werden

1.3 Undefined Components

1.3.1 Anforderungen an Spezifikations- und Modellierungssprachen/techniken

Meschen sind nicht darauf ausgelegt Systeme zu verstehen, die mehr als 5 Komplexe Objekte enthalten. Die Meisten Systeme fordern jedoch mehr. Hilfe bietet eine Hierarchy für die Spraceh/ Technik.

- Behavioral: states, processes, procedures
- Structural: processors, racks, printed circuit boards
- Component-based design: Das System muss von Komponenten designed sein, die Synchronisiert argieren können.
- Timing: Erforderliche Spezifikationen
  - Measured elapsed Time: Check nach vergangener Zeit
  - Means for delaying Process: Prozesse können schlafen gelegt werden
  - Possibility to specify timeouts: In einer spezifizierten maximalen Laufzeit for Timeout bleiben
  - Methods for specifying deadlines: Programme sollen vor einer Angegebenen Zeit terminieren
- support for design: kann unterteilt werden in
  - State-oriented behaviour: Verhalten, wie das von Automaten (States)
  - Event-handling: externe oder interne Events lösen Berechnungen aus
  - Exception-oriented behaviour: Imgang mit Fehlern

1.4 SUD

Das system under design (SUD) wird oftmals wie auch in der Software-Konstruktion üblich als Anfragetext vorgelegt. Dabei sind Anforderungen and das SUD:

- Machine-readable
- Version-management
- Dependency analysis

Applikations-Interaktionen können mittels UML modelliert werden. (Sequenz-/Use-Case Diagramme)

1.5 Communicating finite state machines

1.5.1 State Charts

State Charts bieten eine vereinfachte Schreibweise komplexer Automaten.

In einem State Chart wir unteschieden zwischen super- und supstates. Substates werden in einem superstate durchlaufen, der superstate selbst betritt entweder immer seinen ersten substate oder den letzten eingetretenen substate, je nachdem, ob mit einer History gearbeitet wird oder nicht.

Superstates heißen OR-superstate wenn genau ein substate aktiv ist und AND-superstate, wenn substates conkurrierend aktuv sind.

Kanten werden mit der Konvetion 'event[condition]/reaction' (z.B. 'service-off[not in lProc]/service:=0') beschriftet. Die Evaluierung dieser Labels erfolgt mithilfe der State-Mate Semantik. In dieser existieren 3 Phasen.

1. Effekte externer Veränderungne und Konditionen evaluieren
2. Die Menge an Traisitionen während des aktuellen Schrittes und rechte Seiten (Zuweisungen) werden berechnet
3. Transitionen werden aktiv, die Variablen bekommen ihre neuen Werte zugewiesen

Hierbei erhalten wieder ein bestimmtes Verfahren durch die Trennung der Phasen 2 und 3.

1.5.2 SDL

SDL wird benutzt um asynchrone Nachrichtenvermittlung (asynchronous message passing communication) zu modellieren

Kommunikation zwischen SDLs basiert auf die Nachrichtenverteilung von Signalen(=input+output) mittels einer FIFO-queue. Wenn Signale gleichzeitig ankommen ist deren Reihenfolge in der FIFO-queue nicht deterministisch/ unbekannt.

1.6 Datenfluss

Das Modellieren eines Datenflusses umfasst wie Daten in einem Informationssystem bewegt werden. Dazu werden unter anderem:

- Przesse (Aktivitäten, die Daten transformieren)
- Datenbanken/ data stores
- Externe Entitäten (was/ wer sendet/ empfängt daten ins/vom System)
- Datenflusse (Routen des Datenflusses)

betrachtet.

1.6.1 Kahn process networks (KPN)

Kahn Netzwerke haben jeweils mindestens einen Input- und Output-Kanal. Einschränkungen:

- Es kann nicht für verfügbare Daten for einer versuchten Leseoperation gecheckt werden

- Ein Prozess kann immer nur die Ports nacheinander lesen

Daraus folgt: Die Reihenfolge der Leseoperationen erfolgt nicht über die Ankunft von Daten. Also sind KPNs deterministisch. Ferner sind KPNs Turingvollständig, aber nicht Deadlock-frei.

1.6.2 Synchronous data flow (SDF)

Weniger stark als KPN, aber besser zu analysieren sind SDFs. SDFs besieren auf einer globalen Uhr, diese entscheidet wann Token ‘gefeuert‘ werden. In einem SDF Graphen/Netzwerk werden die knoten als Akteure bezeichnet. Ein Akteur ist bereit, wenn genügend Tokens für ihn vorhanden sind. Ein akteur verbrauch und produziert Token (Input/ Output).

Gibt es Deadlocks in SDFs?
Nein SDFs sind Deadlock-frei. Dafür sind SDFs aber nicht Turingvollständig.

1.7 Petri Netze

Petri Netze modellieren kausale Abhängigkeiten. Petri Netze haben auch gewichtete Ein- und Ausgangskanten. Ähnlich wie beim KPN wird auch ‘gefeuert‘, jedoch nicht in Abhängigkeit von Zeit. Damit sind Petri Netze nichtdeterministisch.