Modellierung verteilter Systeme

Grundlagen der Programm und Systementwicklung

Sommersemester 2012

Prof. Dr. Dr. h.c. Manfred Broy

Unter Mitarbeit von Dr. M. Spichkova, J. Mund, P. Neubeck

Lehrstuhl Software & Systems Engineering

Synchrone Kommunikation

Formen der Kommunikation

Fragen im Zusammenhang mit Kommunikation

- Adressierung
  - Über Identifikatoren der beteiligten Systeme (siehe OO)
  - Über Kanäle oder Ports
  - Über Vermittler (Konnektoren, Bussysteme, Black Boards, ...)
- Es existieren zwei grundlegend verschiedene Konzepte von Kommunikation (Nachrichtenaustausch)
  - asynchron: Nachricht wird gesendet (unabhängig von der Bereitschaft des Empfängers zum Empfangs): Senden und Empfangen sind eigenständige Aktionen; Senden ist unabhängig vom Empfangen, aber kausal für Empfangen
  - synchron: Nachricht wird gesendet, wenn der Empfänger bereit ist zum Empfang: Nachrichtenaustausch erfolgt als Übergabe in einer gemeinsamen Aktion

Asynchrone Kommunikation

- Senden
  - ist immer möglich, wenn zu sendende Nachricht vorliegt
  - braucht nicht mit dem Empfänger abgestimmt werden
- Übertragen
  - kann über Ketten von Übertragungsschritten erfolgen
  - Kein Deadlock möglich (nur zyklisches Warten auf Nachrichten)
- Gesendete Nachrichten
  - müssen gepuffert werden, bis Empfänger empfangsbereit ist
  - werden u.U. nie empfangen/abgerufen
- Sender
  - kann unabhängig von der Empfangsbereitschaft seine Aktivität fortführen
  - erfährt nichts über den Zustand des Empfängers oder ob dieser die Nachricht abruft
- Empfänger
  - Kann Nachricht empfangen, sobald sie vorliegt

Synchrone Kommunikation

- Senden
  - ist nur möglich, wenn Empfänger empfangsbereit ist
  - muss mit dem Empfänger abgestimmt werden
- Übertragen
  - muss (zumindest logisch) direkt erfolgen
  - Deadlock möglich (Warten auf Sende/Empfangsbereitschaft)
- Gesendete Nachrichten
  - müssen nicht gepuffert werden
  - werden sicher empfangen
- Sender
  - muss ggf. warten - kann nicht unabhängig von der Empfangsbereitschaft seine Aktivität fortführen
  - erfährt etwas über den Zustand des Empfängers und dass der Empfänger die Nachricht abgerufen hat
- Empfänger
  - muss ggf. warten - Kann Nachricht empfangen, sobald Sender sendebereit ist

Synchrone Kommunikation

- Erfordert das Konzept gemeinsamer Aktionen
- Historische Beiträge
  - CSP von Hoare
  - CCS von Milner
  - weitere Prozessalgebren

Zustandsmaschinen mit gemeinsamen Aktionen

- Wir betrachten Zustandsmaschinen mit aktionsmarkierten Übergängen
  - Zustandsmenge Σ,
  - *Aktionsmenge* A
  - Anfangszuständen $\Sigma_0 \subseteq \Sigma$
  - Zustandsübergangsrelation Δ: $\Sigma \times A \times \Sigma \rightarrow IB$
    für $(\sigma, a, \sigma') \in \Delta$ schreiben wir kurz

$$\sigma \xrightarrow{a} \sigma'$$

- Die Aktionsmenge A sei in zwei disjunkte Mengen gegliedert
  - lokale Aktionen
  - gemeinsame Aktionen

Synchrone Automaten

- Basis: Menge von Automaten mit markierten Übergängen

$$\Delta_i : \Sigma_i \times \mathsf{A}_i \rightarrow \wp(\Sigma_i)$$

- Zusätzlich: Menge von Endzuständen

$$\Sigma_i^e \subseteq \Sigma_i = (\sigma \mid \forall a \in A_i : \Delta_i(\sigma, a) = \emptyset)$$

- Komposition von zwei Automaten zu einem synchronen Automaten

$$\Delta = \Delta_1 \,|||\, \Delta_2 : (\Sigma_1 \times \Sigma_2) \times \mathsf{A} \rightarrow \wp(\Sigma_i)$$

Synchrone Automaten

- Annahmen:
  - $A_1 \cap A_2 = \emptyset$ *keine gemeinsamen Aktionen*
  - $A_i = A_i^s \cup A_i^a$ *Zerlegung der Aktionsmenge in synchrone und asynchrone Aktionen*
  - $h : A_1^s \times A_2^s \rightarrow Bool$ *Prädikat, welche Aktionen gemeinsam (synchron) ausgeführt werden*

- *Aktionsmenge des synchronen Automaten*

$$A = A_1^a \cup A_2^a \cup (A_1^s \times A_2^s)$$

Synchrone Automaten

Die Grundidee ist, dass die synchronen Aktionen durch die Automaten in dem zusammengesetzten Automaten gemeinsam ausgeführt werden (müssen)

$$\Delta((\sigma_1,\sigma_2),a) =$$

$$\begin{array}{l|l} \textit{asynchr. Anteil} & \{(\sigma_1',\sigma_2) : a \in A_1^a \wedge \sigma_1' \in \Delta_1(\sigma_1,a)\} \\ & \cup \{(\sigma_1,\sigma_2') : a \in A_2^a \wedge \sigma_2' \in \Delta_2(\sigma_2,a)\} \\ \hline \textit{synchr. Anteil} & \cup \{(\sigma_1',\sigma_2') : \exists a_1 \in A_1^s, a_2 \in A_2^s : \\ & \quad a = (a_1,a_2) \wedge h(a_1,a_2) \wedge \\ & \quad \sigma_1' \in \Delta_1(\sigma_1,a_1) \wedge \sigma_2' \in \Delta_2(\sigma_2,a_2)\} \end{array}$$

CSP - Syntax

nop	leere Anweisung
abort	nichtterminierende Anweisung
x := E	Zuweisung
c ? v	Eingabeereignis (bei Hoare: *receive*)
c ! E	Ausgabeereignis (bei Hoare: *send*)
S ; S′	Sequenzielle Komposition von Anweisungen
if . . . [] G_i **then** S_i [] . . . **fi**	Bewachte Anweisung (bei Hoare: *Guarded Command*)
S ‖ S′	Parallele Komposition (Nebenläufigkeit, Parallelität)
do . . . [] G_i **then** S_i [] . . . **od**	Wiederholungsanweisung

CSP - Syntax - Kommunikation in Wächtern

Dabei können die Bedingungen G_i von bewachten Anweisungen und Wiederholungsanweisungen folgende syntaktische Form annehmen (C_i sei ein Boolescher Ausdruck und c sei ein Kanalname):

$G_i \equiv C_i$	Wächter (ohne Kommunikation)
$G_i \equiv C_i : c ? v$	bewachte Kommunikation: Eingabe
$G_i \equiv C_i : c ! E$	bewachte Ausgabe

CSP – Beispiel: Puffer

Da in CSP keine Kanäle mit impliziten Puffern (wie bei asynchroner Kommunikation) verwendet werden, müssen Puffer explizit formuliert werden. Ein endlicher Puffer mit maximaler Länge bound wird in CSP wir folgt beschrieben:

```
P (in: Chan Data, out: Chan Data) =
    {q: var Queue Data;
     x: var Data;
     q := ⟨⟩;
     do ¬isempty(q):  out ! first(q)    then q := rest(q)
     [] q < bound:    in ? x            then q := enq(q, x)
     od }
```

CSP – Beispiel: Verkaufsautomat

Kanäle:

wahl, warenaus : Chan Ware

geldein, geldaus : Chan Nat

ausgabe, rückgabe : Chan Signal

```
b: var Bool;  g,y: var Nat;  x: var Ware;  z: var Signal;
do true then
      wahl ? x;
      b, g := true, 0;
      do b ∧ g < preis(x):   geldein ? y      then g := g + y
      []   b ∧ g ≥ preis(x): ausgabe ? z      then warenaus ! x; b := false
      []   b ∧ g < preis(x):  rückgabe ? z    then geldaus ! g; b := false
      od
od
```

Verklemmung in CSP

- In CSP stellt jeder Nachrichtenaustausch eine unteilbare Aktion dar.
- Nicht zusammenpassende Aktionen können zu einer **Verklemmung** führen
- Betrachte folgendes Programm:

$$(x\,!\,1;\ y\,?\,v)\ \|\ (y\,!\,2; x\,?\,w)$$

- Der linke Prozess wartet auf einen Empfänger einer Nachricht auf Kanal x, der rechte Prozess wartet auf einen Empfänger einer Nachricht auf Kanal y

- Keine Verklemmung des obigen Programms bei asynchroner Kommunikation

Operationelle Semantik von CSP

- Beschreibung von CSP-Programmen als Zustandsmaschinen
- Zustandsübergangsrelation:

$$\subseteq (\text{CSP} \times \Sigma) \times A \times (\text{CSP} \times \Sigma)$$

- CSP: Menge aller CSP-Programme
- Aktionsmenge:

$$A = \{x!d : x \in Channel \wedge d \in Wert\} \cup \{x?d : x \in Channel \wedge d \in Wert\} \cup \{\epsilon\}$$

- Übergang mit leerer Aktion („stiller Übergang“)

$$t \xrightarrow{\varepsilon} t' \text{ oder auch } t \longrightarrow t'$$

- Auswertungsfunktion Val_σ wertet Ausdrücke im Zustand σ aus

$$Val_\sigma : Exp \longrightarrow Value$$

Operationelle Semantik von CSP: Senden/Empfangen

Operationelle Semantik durch Axiomatisieren der Übergangsfunktion:

Empfangen:

$$(x?v, \sigma) \xrightarrow{x?d} (\mathbf{nop}, \sigma[d/v])$$

Senden:

$$\frac{Val_\sigma[E] = d}{(x!E, \sigma) \xrightarrow{x!d} (\mathbf{nop}, \sigma)}$$

Operationelle Semantik von CSP: Seq. Komposition

Sequentielle Komposition:

$$\frac{(S, \sigma) \xrightarrow{m} (S', \sigma')}{(S; S'', \sigma) \xrightarrow{m} (S'; S'', \sigma')}$$

Sequentielle Komposition mit **nop**:

$$(\textbf{nop}; S, \sigma) \xrightarrow{\varepsilon} (S, \sigma)$$

Operationelle Semantik von CSP: Kommunikation

Bewachte Anweisung mit Kommunikation:

$$\frac{\mathsf{Val}_\sigma[\mathsf{C}_i] = \mathsf{true} \qquad (\mathsf{M}_i, \sigma) \xrightarrow{\mathsf{m}} (\textbf{nop}, \sigma')}{(\textbf{if } \ldots [] \mathsf{C}_i : \mathsf{M}_i \textbf{ then } \mathsf{S}_i [] \ldots \textbf{fi}, \sigma) \xrightarrow{\mathsf{m}} (\mathsf{S}_i, \sigma')}$$

Bewachte Anweisung ohne Kommunikation:

$$\frac{\mathsf{Val}_\sigma[\mathsf{C}_i] = \mathsf{true}}{(\textbf{if } \ldots [] \mathsf{C}_i \textbf{ then } \mathsf{S}_i [] \ldots \textbf{fi}, \sigma) \rightarrow (\mathsf{S}_i, \sigma')}$$

Operationelle Semantik von CSP: Parallelität

Parallele Anweisung:

(a) unabhängige Aktionen:

$$\frac{(S_1, \sigma) \xrightarrow{m} (S_1', \sigma')}{\begin{array}{l}(S_1 \parallel S_2, \sigma) \xrightarrow{m} (S_1' \parallel S_2, \sigma') \\ (S_2 \parallel S_1, \sigma) \xrightarrow{m} (S_2 \parallel S_1', \sigma')\end{array}}$$

(b) interne synchrone Kommunikation:

$$\frac{(S_1, \sigma) \xrightarrow{x!d} (S_1', \sigma), \quad (S_2, \sigma) \xrightarrow{x?d} (S_2', \sigma')}{\begin{array}{l}(S_1 \parallel S_2, \sigma) \xrightarrow{\varepsilon} (S_1' \parallel S_2', \sigma') \\ (S_2 \parallel S_1, \sigma) \xrightarrow{\varepsilon} (S_2' \parallel S_1', \sigma')\end{array}}$$

(c) Terminierung eines Zweiges in der parallelen Komposition

$$(S \parallel \textbf{nop}, \sigma) \xrightarrow{\varepsilon} (S, \sigma)$$

$$(\textbf{nop} \parallel S, \sigma) \xrightarrow{\varepsilon} (S, \sigma)$$

Operationelle Semantik von CSP

Wiederholung und Zuweisung sind wie gehabt für sequentielle Programme definiert.

Wiederholung: Sei

$$\mathsf{DO} = \textbf{do} \quad \ldots \;[]\; \mathsf{G}_i \;\textbf{then}\; \mathsf{S}_i \;[]\; \ldots \;\textbf{od}$$

Dann gilt

$$(\mathsf{DO}, \sigma) \rightarrow (\textbf{if} \quad \ldots \;[]\; \mathsf{G}_i \;\textbf{then}\; \mathsf{S}_{ij};\; \mathsf{DO} \;[]\; \ldots [] \; \neg C_1 \wedge \ldots \wedge \neg C_n \;\textbf{then}\; \textbf{nop fi},\; \sigma)$$

wobei C_i jeweils der Boolesche Ausdruck im Wächter G_i sei.

Zuweisung

$$(\mathsf{x} := \mathsf{E}, \sigma) \rightarrow (\textbf{nop}, \sigma[\mathsf{val}_\sigma[\mathsf{E}]/\mathsf{x}])$$

Operationelle Semantik von CSP: Verklemmung

Für einen Zustand $\sigma \in \Sigma^e$ sagen wir, dass der Agent $t \neq \boldsymbol{nop}$ in einer **Verklemmung** ist, falls keine Aktion *m*, Nachfolgeprogramm *t'* und Zustand σ' existieren mit:

$$(\mathrm{t}, \sigma) \xrightarrow{\mathrm{m}} (\mathrm{t}', \sigma')$$

Operationelle Semantik von CSP

- Erweiterung der Übergangsrelation auf **sequenzmarkierte** Übergänge:

$$(t,\sigma) \overset{\epsilon}{\Rightarrow} (t',\sigma') \quad falls\ (t,\sigma) \overset{\epsilon}{\rightarrow} (t',\sigma')$$

$$(t,\sigma) \overset{\langle m \rangle}{\Longrightarrow} (t',\sigma') \quad falls\ (t,\sigma) \overset{m}{\rightarrow} (t',\sigma')$$

$$(t,\sigma) \overset{s \circ s'}{\Longrightarrow} (t'',\sigma'') \quad falls\ (t,\sigma) \overset{s}{\Rightarrow} (t',\sigma')\ und\ (t',\sigma') \overset{s'}{\Rightarrow} (t'',\sigma'')$$

- **Ablaufpfad** s für die Konfiguration (t,σ):

$$\exists\ t',\sigma' :\ (t,\sigma) \overset{s}{\Rightarrow} (t',\sigma')$$

Bereitschaft und Verweigerung

- $R \subseteq M \setminus \{\epsilon\}$ heißt **Bereitschaftsmenge** für einen Ablauf $s \in (M \setminus \{\epsilon\})^*$ und eine Konfiguration (t, σ), wenn ein Konfiguration (t', σ') existiert, so dass gilt:

$$(t,\sigma) \overset{s}{\Rightarrow} (t',\sigma') \wedge \left(\exists t'', \sigma'' : (t',\sigma') \overset{m}{\rightarrow} (t'',\sigma'')\right) \Leftrightarrow m \in R$$

- Eine leere Bereitschaftsmenge charakterisiert die Situation der Verklemmung

- Jede Menge, die mit mindestens einer Bereitschaftsmenge einen leeren Durchschnitt bildet, heißt **Verweigerungsmenge**

Bemerkungen zu CSP und synchroner Kommunikation

- Semantik sehr kompliziert
- Zur Äquivalenz/Kompatibilität von CSP-Programmen siehe nächsten Foliensatz
  - Stichwort Bisimulation
- Kommunikationsmodell entspricht nicht heutigen Systemen
  - Internet
  - Client/Server
  - Bordnetze in Fahr- und Flugzeugen
- Als Abstraktion dennoch manchmal hilfreich