Modellierung von Echtzeitsystemen

Reaktive Systeme

Werkzeuge: SCADE, Esterel Studio

Fragen zur letzten Vorlesung

- Was sind Aktoren?
  - Unterscheidung zum Begriff Aktoren aus der Mechatronik?
- Wie zuverlässig ist der generierte Code bei modellbasierten Entwicklungswerkzeugen?
- Wie gut ist der generierte Code zu lesen?

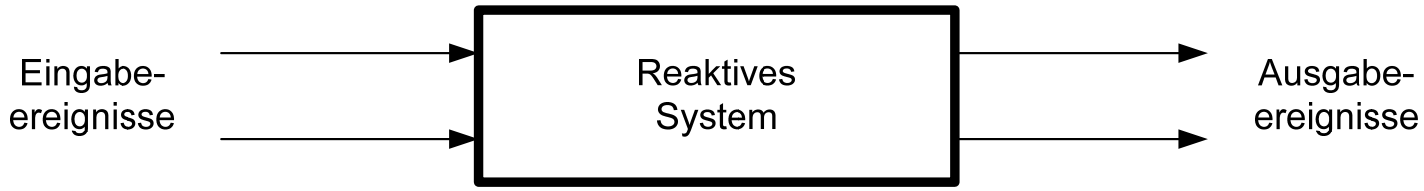
- Frage zu dieser Vorlesung:
  - Wann verwendet man synchrone Sprachen, wann synchronen Datenfluss?
    - Unterscheidung zwischen kontrollorientierten (Fragestellung: wie reagiert das System auf Ereignisse) und datenorientierten (Fragestellung: wie werden eingehende Daten verarbeitet) Anwendungen.

Esterel

- Esterel ist im klassischen Sinne eher eine Programmiersprache, als eine Modellierungssprache
- Esterel wurde von Jean-Paul Marmorat und Jean-Paul Rigault entwickelt um die Anforderungen von Echtzeitsystemen gezielt zu unterstützen:
  - direkte Möglichkeit zum Umgang mit Zeit
  - Parallelismus direkt in der Programmiersprache
- G. Berry entwickelt die formale Semantik für Esterel
- Es existieren Codegeneratoren zur Generierung von u.a. **sequentiellen** C, C++ Code:
  - Aus Esterel-Programmen mit parallelen Berechnungen wird ein Programm mit einem Berechnungsstrang erzeugt ⇒ deterministische Ausführung
  - Technik basiert aus der Erstellung eines endlichen Automaten.
- In der Übung setzen wir die kommerziellen Werkzeuge Esterel Studio / SCADE von Esterel Technology (www.esterel-technologies.com) zum Erlernen von Esterel / Lustre ein.
- SCADE wurde unter anderem zur Entwicklung des Airbus A380 eingesetzt.
- Ein Esterel-Compiler kann unter http://www-sop.inria.fr/meije/esterel/esterel-eng.html umsonst heruntergeladen werden.

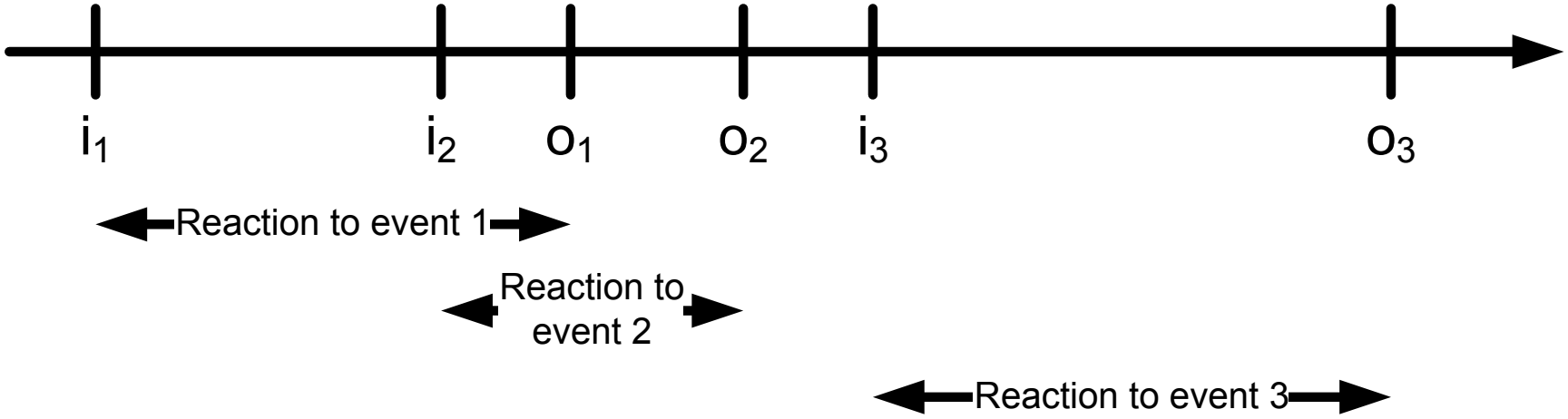
Einführung in Esterel

- Esterel beschreibt reaktive Systemen, das System reagiert auf Eingabeereignisse
- Esterel gehört zu der Familie der synchronen Sprachen, weitere Vertreter: Lustre, Signal, Statecharts
- Synchrone Sprachen zeichnen sich vor allem dadurch aus, dass
  - Interaktionen (Reaktionen) des Systems mit der Umgebung die Basisschritte des Systems darstellen (**reaktives System**).
  - Anstelle von physikalischer Zeit logische Zeit (die Anzahl der Interaktionen) verwendet wird.
  - Interaktionen, oft auch **macro steps** genannt, bestehen aus Einzelschritten (micro steps).



Allgemein: Reaktive Systeme

- Bearbeitung der Ereignisse kann sich überlappen



Synchrony hypothesis

- Die Synchronitätshypothese (**synchrony hypothesis**) nimmt an, dass die zugrunde liegende physikalische Maschine des Systems unendlich schnell ist.

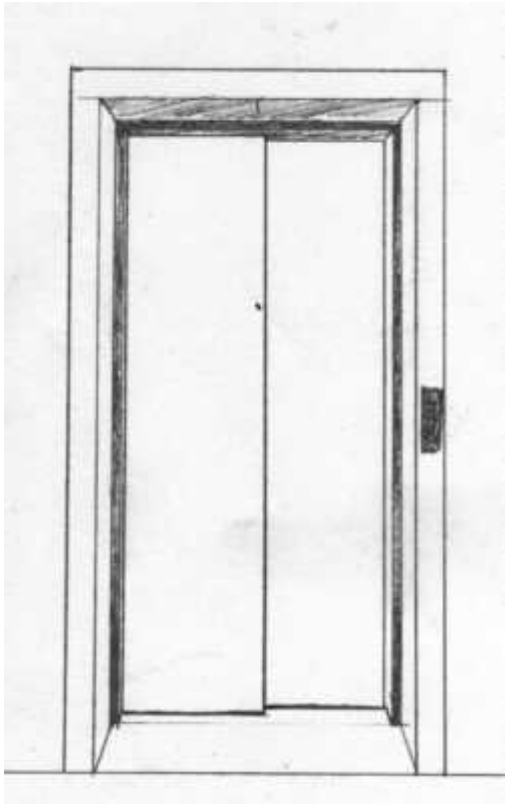
  → Die Reaktion des Systems auf ein Eingabeereignis erfolgt augenblicklich. Reaktionsintervale reduzieren sich zu Reaktionsmomenten (**reaction instants**).
- **Rechtfertigung**: Diese Annahme ist korrekt, wenn die Wahrscheinlichkeit des Eintreffens eines zweiten Ereignisses, während der initialen Reaktion auf das vorangegangene Ereignis, sehr klein ist.
- Esterel erlaubt das gleichzeitige Auftreten von mehreren Eingabeereignissen. Der Reaktionsmoment ist in Esterel dann komplettiert, wenn das System auf alle Ereignisse reagiert hat.

Determinismus

- Esterel setzt den Determinismus der Anwendung voraus: auf eine Sequenz von Ereignissen (auch gleichzeitigen) muss immer dieselbe Sequenz von Ausgabeereignissen folgen.
- Alle Esterel Anweisungen und Konstrukte sind garantiert deterministisch. Die Forderung nach Determinismus wird durch den Esterel Compiler überprüft.
- Durch den Determinismus wird die Verifikation von Anwendungen wesentlich vereinfacht, allerdings birgt es auch die Gefahr, dass Ereignisse „vergessen“ werden, falls sie zeitgleich mit wichtigeren Ereignissen eintreffen.

Esterel an einem Beispiel: Aufzugstür

- Aufgabe: Öffnen und Schließen der Aufzugstür
- Sicherheitsfunktion:
  - Tür darf während der Fahrt nicht geöffnet werden
- 1. Schritt: Definition der Module (parallelen Abläufe)
- 2. Schritt: Definition der Eingangssignale
- 3. Schritt: Definition der Ausgangssignale
- 4. Schritt: Definition der Zustände (inkl. Anfangszustand):
- 5. Schritt: Definition der Zustandsübergänge



Module

- **Module** definieren in Esterel (wieder verwendbaren) Code. Module haben ähnlich wie Subroutinen ihre eigenen Daten und ihr eigenes Verhalten.
- Allerdings werden Module nicht aufgerufen, vielmehr findet eine Ersetzung des Aufrufs durch den Modulcode zur Übersetzungszeit statt.
- Globale Daten werden nicht unterstützt. Ebenso sind rekursive Moduldefinitionen nicht erlaubt.
- Syntax:

```
%this is a line comment
module module-name:
declarations and compiler directives
%signals, local variables etc.
body
end module % end of module body
```

Parallelismus

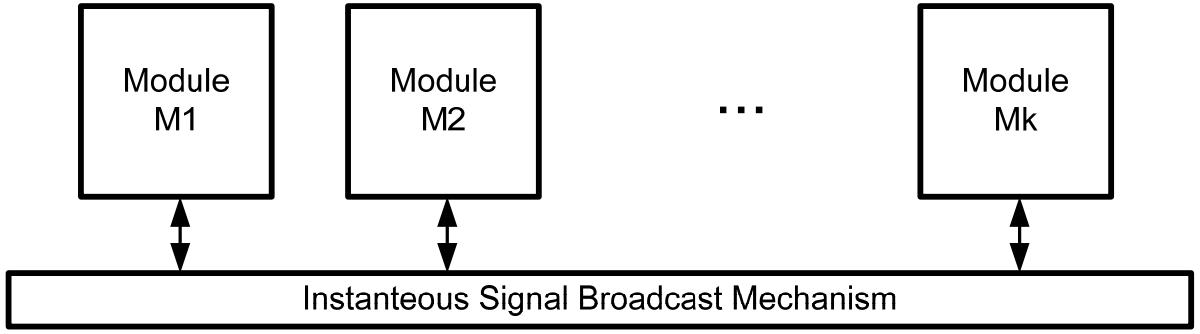
- Zur parallelen Komposition stellt Esterel den Operator || zur Verfügung. Sind P1 und P2 zwei Esterel Programme, so ist auch P1||P2 ein Esterel Programm mit folgenden Eigenschaften:
  - Alle Eingabeereignisse stehen sowohl P1 als auch P2 zur Verfügung.
  - Jede Ausgabe von P1 (oder P2) ist im gleichen Moment für P2 (oder P1) sichtbar.
  - Sowohl P1 als auch P2 werden parallel ausgeführt und die Anweisung P1||P2 endet erst, wenn beide Programme beendet sind.
  - Es können keine Daten oder Variablen von P1 und P2 gemeinsam genutzt werden.
- Zur graphischen Modellierung stehen parallele Teilautomaten zur Verfügung.

Signale

- Zur Modellierung der Kommunikation zwischen Komponenten (Modulen) werden Signale eingeführt. Signale sind eine logische Einheit zum Informationsaustausch und zur Interaktion.
- **Deklaration**: Die Deklaration eines Signals erfolgt am Beginn des Moduls. Der Signalname wird dabei typischerweise in Großbuchstaben geschrieben. Zudem muss der Signaltyp festgelegt werden.
- Esterel stellt verschiedene Signale zur Verfügung. Die Klassifikation erfolgt nach:
  - Sichtbarkeit: Schnittstellen (interface) Signale vs. lokale Signale
  - enthaltener Information: pure Signale vs. wertbehaftete Signale (typisiert)
  - Zugreifbarkeit der Schnittstellensignale: Eingabe (input), Ausgabe(output), Ein- und Ausgabe (inputoutput), Sensor (Signal, dass immer verfügbar ist und das nur über den Wert zugreifbar ist)

Signal Broadcast Mechanismus

- **Versand**: Der Versand von Signalen durch die `emit` Anweisung (terminiert sofort) erfolgt über einen Broadcast Mechanismus. Signale sind immer sofort für alle anderen Module verfügbar. Die `sustain` Anweisung erzeugt in jeder Runde das entsprechende Signal und terminiert nicht. Es muss mit Hilfe von `abort` abgebrochen werden.
- **Verfügbarkeit**: Signale sind nur für den bestimmten Moment verfügbar.
- Nach Ende des aktuellen Moments wird der Bus zurückgesetzt.
- **Zugriff**: Prozesse können per `await` auf Signale warten oder prüfen, ob ein Signal momentan vorhanden ist (`if`). Auf den Wert eines wertbehaftete Signale kann mittels des Zugriffsoperator ? zugegriffen werden.



Ereignisse (Events)

- **Ereignisse** setzen sich zu einem bestimmten Zeitpunkt (**instant**) aus den Eingabesignalen aus der Umwelt und den Signalen, die durch das System als Reaktion ausgesandt werden, zusammen.
- Esterel Programme können nicht direkt auf das ehemalige oder zukünftige Auftreten von Signalen zurückgreifen. Auch kann nicht auf einen ehemaligen oder zukünftigen Moment zugegriffen werden.
- Einzige Ausnahme ist der Zugriff auf den letzten Moment. Durch den Operator `pre` kann das Auftreten in der vorherigen Runde überprüft werden.

Beziehungen (relations)

- Der Esterel-Compiler erzeugt aus der Esterel-Datei einen endlichen Automaten. Hierzu müssen für jeden Zustand (Block) sämtliche Signalkombinationen getestet werden.
- Um bei der automatischen Generierung des endlichen Automaten des Systems die Größe zu reduzieren, können über die `relation` Anweisung Einschränkungen in Bezug auf die Signale spezifiziert werden:
- `relation Master-signal-name => Slave-signal-name;`

  Bei jedem Auftreten des Mastersignals muss auch das Slave-Signal verfügbar sein.
- `relation Signal-name-1 # Signal-name-2 # ... # Signal-name-n;`

  In jedem Moment darf maximal eines der spezifizierten Signale Signal-name-1, Signal-name-2 ,..., Signal-name-n präsent sein.

Zeitdauer

- Die Zeitachse wird in Esterel in diskrete Momente (**instants**) aufgeteilt. Über die Granularität wird dabei in Esterel keine Aussage getroffen.
- Zur deterministischen Vorhersage des zeitlichen Ablaufes von Programmen wird jede Anweisung in Esterel mit einer genauen Definition der Ausführungszeitdauer verknüpft.
- So terminiert beispielsweise `emit` sofort, während `await` so viel Zeit benötigt, bis das assoziierte Signal verfügbar ist.
- Auf den folgenden Folien werden die wichtigsten Konstrukte erläutert.

await Anweisung

```
await
    case Occurence-1 do Body-1
    case Occurence-2 do Body-2
    ...
    case Occurence-n do Body-n
end await;
```

- Mit Hilfe dieser Anweisung wird auf das Eintreten einer Bedingung gewartet. Im Falle eines Auftretens wird der assoziierte Code gestartet. Werden in einem Moment mehrere Bedingungen wahr, entscheidet die textuelle Reihenfolge. So kann eine deterministische Ausführung garantiert werden.

Unendliche Schleife (infinite loop)

```
loop Body end loop;
```

- Mit Hilfe dieser Anweisung wird ein Stück Code Body endlos ausgeführt. Sobald eine Ausführung des Codes beendet wird, wird der Code wieder neu gestartet.
- **Bedingung**: die Ausführung des Codes darf nicht im gleichen Moment, indem sie gestartet wurde, terminieren.

Abort Konstrukt

- Zur einfacheren Modellierung können Abbruchbedingungen nicht nur durch Zustandsübergänge, sondern auch direkt mit Makrozuständen verbunden werden.
- Dabei wird zwischen zwei Arten des Abbruches unterschieden:
  - weak abort: die in der Runde vorhandenen Signale werden noch verarbeitet, danach jedoch der Abbruch vollzogen
  - strong abort: der Abbruch wird sofort vollzogen, eventuell vorhandene Signale ignoriert.
- In der Sprache Esterel wird eine Abbruchbedingung durch das Konstrukt
  `abort Body when Exit_Condition`
  bzw.
  `abort Body when immediate Exit_Condition`
  ausgedrückt.

Lokale Signale und wertbehaftete Signale

```
signal Signal-decl-1,Signal-decl-2,...,Signal-
decl-n in
    Body
end;
```

- Durch diese Anweisung werden lokale Signale erzeugt, die nur innerhalb des mit Body bezeichneten Code verfügbar sind.

  `Signal-name: Signal-type`

- Der Typ eines wertbehaften Signals kann durch diese Konstruktion spezifiziert werden.

every Anweisung

- Mit Hilfe der `every` Anweisung kann ein periodisches Wiederstarten implementiert werden.
- Syntax:

```
every Occurence do
    Body
end every
```

- Semantik: Jedes Mal falls die Bedingung `Occurence` erfüllt ist, wird der Code Body gestartet. Falls die nächste Bedingung `Occurence` vor der Beendigung der Ausführung von Body auftritt, wird die aktuelle Ausführung sofort beendet und eine neue Ausführung gestartet.
- Es ist auch möglich eine Aktion in jedem Moment zu starten:

```
every Tick do
    Body
end every;
```

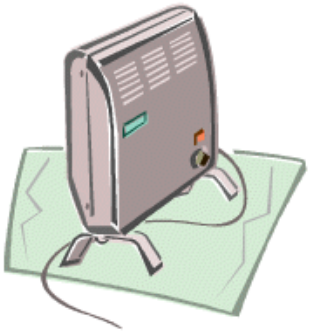
if Anweisung in Bezug auf Signale

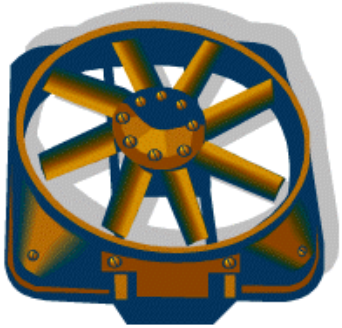
- Durch Verwendung der if- Anweisung kann auch die Existenz eines Signals geprüft werden.
- **Syntax**:

```
if Signal-Name then
    Body-1
else
    Body-2
```

- **Semantik**: Bei Start dieser Anweisung wird geprüft, ob das Signal `Signal-Name` verfügbar ist. Ist es verfügbar, so wird der Code von `Body-1` ausgeführt, anderenfalls von Body-2. Innerhalb der Anweisung `if` kann auch entweder der `then Body-1` oder der `else Body-2` -Teil weggelassen werden.

Beispiel: Temperaturregelung im Aufzug





Beschreibung Beispiel

- Ziel: Regelung der Temperatur (Betriebstemperatur 5-40 Grad Celsius)
- Ansatz: Nähert sich die Temperatur einem der Grenzwerte, so wird der Lüfter bzw. die Heizung (Normalstufe) eingeschaltet. Verbleibt der Wert dennoch im Grenzbereich, so wird auf die höchste Stufe geschaltet.
  Ist der Wert wieder im Normalbereich, so wird (zur Vereinfachung) der Lüfter bzw. die Heizung wieder ausgeschaltet.
  Wird die Betriebstemperatur über- bzw. unterschritten, so wird ein Abbruchssignal an den Aufzug geschickt.

Esterel Code 1. Teil

```
module Temperature:
input TEMP: integer, SAMPLE_TIME, DELTA_T;
output HEATER_ON, HEATER_ON_STRONG,
       HEATER_OFF,VENTILATOR_ON, VENTILATOR_OFF,
       VENTILATOR_ON_STRONG, ABORT;
input relation SAMPLE_TIME => TEMP;
    signal COLD, NORMAL, HOT in
        every SAMPLE_TIME do
            await immediate TEMP;
            if
                case ?TEMP<5 or ?TEMP>40 do emit ABORT
                case ?TEMP>=35 do emit HOT
                case ?TEMP<=10 do emit COLD
                default do emit NORMAL
            end if
        end every
        ||
```

Esterel Code 2. Teil

```
        loop
             await
                 case COLD do
                     emit HEATER_ON;
                     abort
                         await NORMAL;
                         emit HEATER_OFF;
                     when DELTA_T do
                         emit HEATER_ON_STRONG;
                         await NORMAL;
                         emit HEATER_OFF;
                     end abort
                 case HOT do
               ...
             end await
         end loop
    end signal
end module
```