



Modellierung von Echtzeitsystemen

Reaktive Systeme

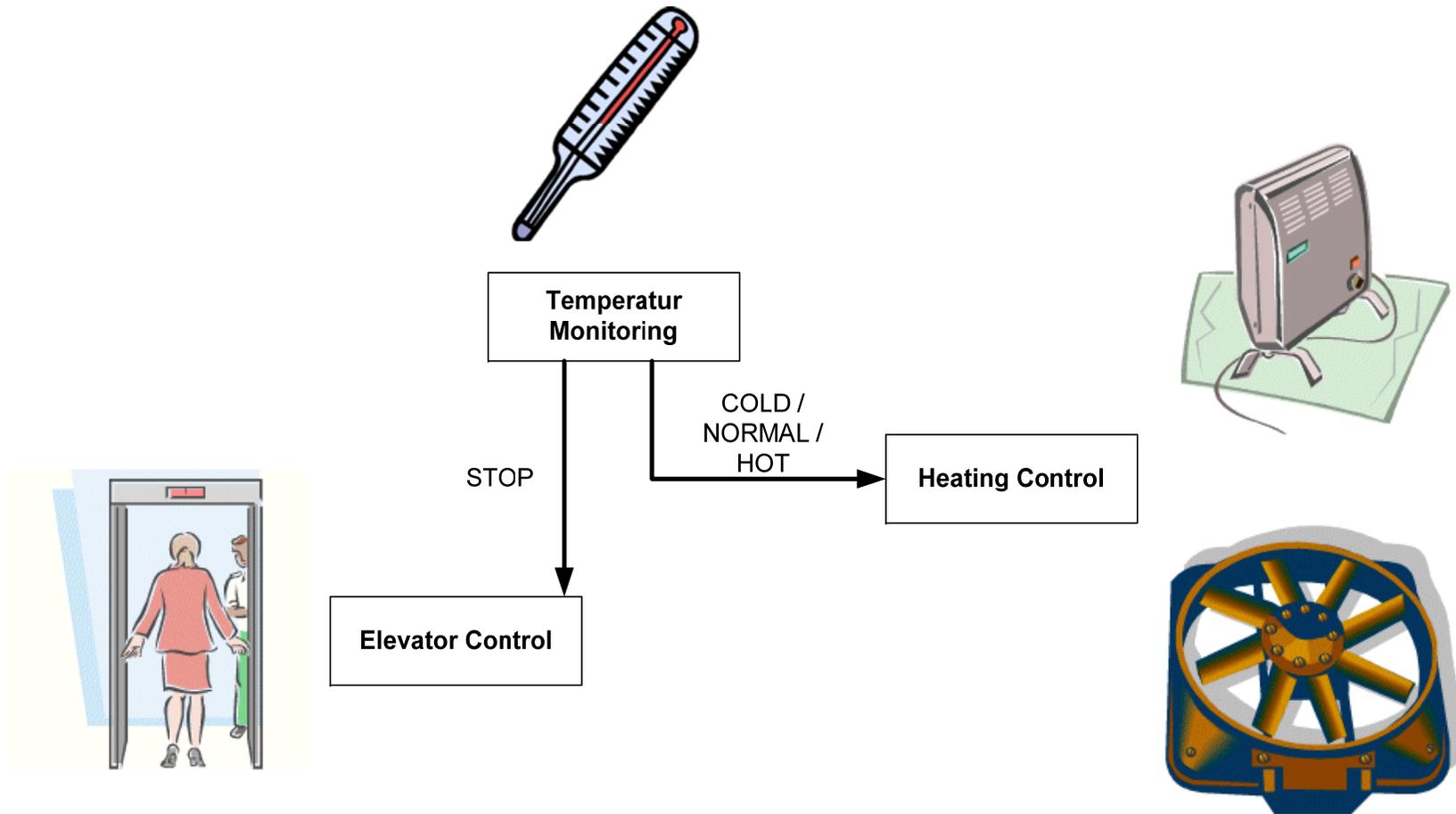
Werkzeuge: SCADE, Esterel Studio



Klausurtermin

- Für Studenten, die einen Schein benötigen, wird am Ende der Vorlesung eine schriftliche Klausur angeboten.
- Stoff der Klausur sind die Inhalte der Vorlesung.
- Termin:
 - 09.02.2008 16:00-18:00 Uhr
 - Ort: CH21010
- Voraussichtlich erlaubte Hilfsmittel: keine

Verfeinertes 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 Abbruchsignal 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
```



Esterel-Konstrukt: Module

- **Module** definieren in Esterel (wiederverwendbaren) Code. Module haben ähnlich wie Unterprogramme 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 Variable 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
```

Esterel-Konstrukt: Parallele Komposition

- Zur parallelen Komposition stellt Esterel den Operator \parallel zur Verfügung. Sind $P1$ und $P2$ zwei Esterel-Programme, so ist auch $P1\parallel P2$ 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 $P1\parallel P2$ 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.

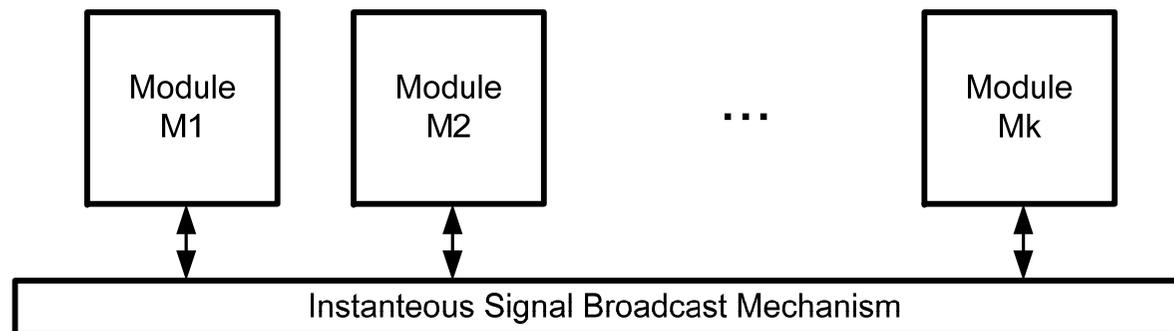


Signale

- Zur Kommunikation zwischen Komponenten (Modulen) werden Signale eingeführt. Signale sind eine logische Einheit zum Informationsaustausch und zur Interaktion.
- 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, das immer verfügbar ist und das nur über den Wert zugreifbar ist)

Esterel-Konstrukt: Broadcast-Mechanismus

- **Versand:** Der Versand von Signalen durch die `emit` Anweisung (terminiert sofort) erfolgt über einen Broadcast-Mechanismus, d.h. Signale sind immer sofort für alle anderen Module verfügbar. Die `sustain` Anweisung erzeugt in jeder Runde das entsprechende Signal und terminiert nicht.
- **Zugriff:** Prozesse können per `await` auf Signale warten oder prüfen, ob ein Signal momentan vorhanden ist (`if`). Auf den Wert eines wertbehafteten Signals kann mittels des Zugriffsoperator `?` zugegriffen werden.





Esterel-Konstrukt: 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 ausgesendet 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.



Esterel-Konstrukt: await Anweisung

```
await
  case Occurrence-1 do Body-1
  case Occurrence-2 do Body-2
  ...
  case Occurrence-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.



Esterel-Konstrukt: 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.



Esterel-Konstrukt: Abort

- 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.



Esterel-Konstrukt: Lokale 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 wertbehafteten Signals kann durch diese Konstruktion spezifiziert werden.



Esterel-Konstrukt: *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;
```



Esterel-Konstrukt: 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.