Embedded Systems
Petri Nets
Computing changes of markings

- „Firing“ transitions $t$ generate new markings on each of the places $p$ according to the following rules:

\[
M'(p) = \begin{cases} 
M(p) - W(p, t), & \text{if } p \in t \setminus t^* \\
M(p) + W(t, p), & \text{if } p \in t^* \setminus t^*
M(p) - W(p, t) + W(t, p), & \text{if } p \in t \cap t^* \\
M(p) & \text{otherwise}
\end{cases}
\]

When a transition $t$ fires from a marking $M$, $w(p, t)$ tokens are deleted from the incoming places of $t$ (i.e. from places $p \in t$), and $w(t, p)$ tokens are added to the outgoing places of $t$ (i.e. to places $p \in t^*$), and a new marking $M'$ is produced.
Activated transitions

- Transition \( t \) is ”activated“ iff

\[
(\forall p \in \cdot t : M(p) \geq W(p, t)) \land (\forall p \in t^\cdot : M(p) + W(t, p) \leq K(p))
\]

Activated transitions can ”take place“ or ”fire“, but don’t have to. The order in which activated transitions fire is not fixed (it is non-deterministic).
Boundedness

- A place is called **k-safe** or **k-bounded** if it contains in the initial marking $m_0$ and in all other reachable from there markings at most $k$ tokens.

- A net is **bounded** if each place is bounded.

- **Boundedness**: the number of tokens in any place cannot grow indefinitely

- **Application**: places represent buffers and registers (check there is no overflow)

- A Petri net is inherently bounded if and only if all its reachability graphs (i.e. reachability graphs with all possible starting states) all have a **finite number of states**.
Liveness

- A transition $T$ is live if in any marking there exists a firing sequence such that $T$ becomes enabled.
- An entire net is **live if all its transitions are live**.
- Important for checking deadlock.

Live?

![Diagram showing a Petri net example with markings and transitions. The left net is marked with "NO" indicating it is not live, while the right net is marked with "YES" indicating it is live.](image)
Liveness (more precisely)

- A transition $t$ is **dead** at $M$ if no marking $M'$ is reachable from $M$ such that $t$ can fire in $M'$.
- A transition $t$ is **live** at $M$ if there is no marking $M'$ reachable from $M$ where $t$ is dead.
- A marking is **live** if all transitions are live.
- A P/T net is **live** if the initial marking is live.

Observations:
- A live net is deadlock-free.
- No transition is live if the net is not deadlock-free.
Deadlock

- A **dead marking (deadlock)** is a marking where no transition can fire.
- A Petri net is **deadlock-free** if no dead marking is reachable.
Shorthand for changes of markings

Firing transition:

\[ M'(p) = \begin{cases} 
M(p) - W(p,t), & \text{if } p \in \cdot t \setminus t^* \\
M(p) + W(t,p), & \text{if } p \in t^* \setminus \cdot t \\
M(p) - W(p,t) + W(t,p), & \text{if } p \in \cdot t \cap t^* \\
M(p) & \text{otherwise}
\end{cases} \]

Let

\[ t(p) = \begin{cases} 
-W(p,t) & \text{if } p \in \cdot t \setminus t^* \\
+W(t,p) & \text{if } p \in t^* \setminus \cdot t \\
-W(p,t) + W(t,p) & \text{if } p \in \cdot t \cap t^* \\
0 & \text{otherwise}
\end{cases} \]

\[ \forall p \in P: M'(p) = M(p) + t(p) \]

\[ M' = M + t \]

+: vector add
Matrix $N$ describing all changes of markings

$$
t(p) = \begin{cases} 
-W(p,t) & \text{if } p \in t \setminus t^* \\
+W(t, p) & \text{if } p \in t^* \setminus t \\
-W(p,t) + W(t, p) & \text{if } p \in t^* \cap t^* \\
0 & \text{otherwise}
\end{cases}
$$

Def.: Matrix $N$ (incidence matrix ) of net $N$ is a mapping

$$
N: P \times T \rightarrow Z \text{ (integers)}
$$

such that $\forall t \in T$: $N(p,t) = t(p)$

Component in column $t$ and row $p$ indicates the change of the marking of place $p$ if transition $t$ takes place.
incidence matrix $N$ of a pure (no elementary loops) place/transition-net:

$$N_{p,t} := \begin{cases} 
-W(t, p), & \text{arc from } p \text{ to } t \\
+W(t, p), & \text{arc from } t \text{ to } p \\
0, & \text{otherwise}
\end{cases}$$

Contribution of $t$ on $p$
Example: $N =$

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
<th>$t_9$</th>
<th>$t_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td></td>
<td></td>
<td></td>
<td>$\varphi$</td>
<td>$\psi$</td>
<td>$\phi$</td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
</tr>
<tr>
<td>$p_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_3$</td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td></td>
<td></td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td></td>
<td></td>
<td>$1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td></td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>$p_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td></td>
<td>$1$</td>
</tr>
<tr>
<td>$p_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td></td>
<td></td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>$p_8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
</tr>
<tr>
<td>$p_9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>$p_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td>$1$</td>
</tr>
<tr>
<td>$p_{12}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>$p_{13}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-1$</td>
</tr>
</tbody>
</table>
State equation

\[ N_{i,t} = N_i - N'_{i}\]

\[ N'_{i} = N_0 + N_{i-1}\]

\[ h' = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}\]

\[ \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}\]
State equation

reachability graph
We are interested in subsets $R$ of places whose number of labels remain invariant under firing of transitions:

- e.g. the number of trains commuting between Amsterdam and Paris (Cologne and Paris) remains constant

Important for correctness proofs
Place - invariants

Standardized technique for proving properties of system models

For any transition \( t_j \in T \) we are looking for sets \( R \subseteq P \) of places for which the accumulated marking is constant:

\[
\sum_{p \in R} t_{-j}(p) = 0
\]

Example:
Characteristic Vector

\[ \sum_{p \in R} t_{-j}(p) = 0 \]

Let:

\[ c_R(p) = \begin{cases} 
1 & \text{if } p \in R \\
0 & \text{if } p \notin R 
\end{cases} \]

\[ \Rightarrow \sum_{p \in R} t_{-j}(p) = t_{-j} \cdot c_R = \sum_{p \in P} t_{-j}(p) c_R(p) = 0 \]

Scalar product
Condition for place invariants

\[ \sum_{p \in R} t_j(p) = t_j \cdot c_R = \sum_{p \in P} t_j(p) c_R(p) = 0 \]

Accumulated marking constant for all transitions if

\[ t_1 \cdot c_R = 0 \]
\[ \ldots \]
\[ t_n \cdot c_R = 0 \]

Equivalent to \( N^T \cdot c_R = 0 \) where \( N^T \) is the transposed of \( N \)
More detailed view of computations

System of linear equations.

Solution vectors must consist of zeros and ones.

Different techniques for solving equation system (Gauss elimination, tools e.g. Matlab, …)
Application to Thalys example

\( N^T \mathbf{c}_R = 0 \), with \( N^T = \)

\[
\begin{array}{cccccccccccc}
 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} & p_{12} & p_{13} \\
\hline
 t_1 & 1 & -1 &  &  &  &  & -1 &  &  &  &  &  & 1 \\
t_2 &  & 1 & -1 &  &  &  &  & -1 &  &  &  &  & 1 \\
t_3 &  &  & 1 & -1 &  &  &  &  & -1 &  &  &  & 1 \\
t_4 &  &  &  & 1 & -1 &  &  &  &  & -1 &  &  & 1 \\
t_5 &  &  &  &  & 1 & -1 &  &  &  &  & -1 &  & 1 \\
t_6 &  &  &  &  &  & 1 &  &  &  &  &  & -1 & -1 \\
t_7 &  &  &  &  &  &  & 1 & -1 &  &  &  &  & -1 \\
t_8 &  &  &  &  &  &  &  & 1 & -1 &  &  &  & 1 \\
t_9 &  &  &  &  &  &  &  &  & 1 & -1 &  &  & 1 \\
t_{10} &  &  &  &  &  &  &  &  &  & 1 & -1 &  & -1 \\
\end{array}
\]

\[ c_{R,1} = \left( 1 1 1 1 1 1 0 0 0 0 0 0 0 0 \right) \]
Solution vectors for Thalys example

We proved that:
• the number of trains serving Amsterdam, Cologne and Paris remains constant.
• the number of train drivers remains constant.

\[
CR_{1} = (1111111000000000),
\]

\[
CR_{2} = (0000000110000100),
\]

\[
CR_{3} = (0000000001100101),
\]

\[
CR_{4} = (1000110011100000).
\]
It follows:

- each place invariant must have at least one label at the beginning, otherwise “dead”
- at least three labels are necessary in the example
$N^T c_R = 0$, with $N^T=$

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>φ</td>
</tr>
<tr>
<td>T2</td>
<td>φ</td>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>T3</td>
<td>-</td>
<td>φ</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>φ</td>
</tr>
<tr>
<td>T4</td>
<td>φ</td>
<td>-</td>
<td>1</td>
<td>φ</td>
<td>1</td>
<td>φ</td>
</tr>
<tr>
<td>T5</td>
<td>φ</td>
<td>φ</td>
<td>-</td>
<td>φ</td>
<td>-</td>
<td>φ</td>
</tr>
</tbody>
</table>

CS - ES
\[ c_1 \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} + c_2 \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} + c_6 \begin{bmatrix} c_7 \\ c_8 \\ c_9 \end{bmatrix} = 0 \]

I: \[ c_1 - c_4 - c_5 = 0 \]

II: \[ c_3 - c_4 - c_6 = 0 \]

III: \[ -c_3 + c_3 + c_5 = 0 \]

IV: \[ -c_2 + c_3 + c_6 = 0 \]

V: \[ -c_3 + c_5 = 0 \] \[ \Rightarrow c_3 = c_5 \]
\[ c_3 = c_5 \]

\[ \overline{III}' - c_2 + c_3 + c_5 = q \]

\[ \overline{III}' = -I \]

\[ \overline{IV}' = -c_2 + c_3 + c_5 = q \]

\[ \overline{IV}' = -II \]
\[ \begin{align*}
I : & \quad C_5 = C_1 - C_4 \\
II : & \quad C_5 = C_2 - C_6 \\
V : & \quad C_5 = C_3
\end{align*} \]

<table>
<thead>
<tr>
<th></th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
<th>C_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>L_2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>L_3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>L_4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>L_5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ P_1 = \{ \rho_1, \rho_4 \}, \quad P_2 = \{ \rho_2, \rho_6 \} \]

\[ P_3 = \{ \rho_1, \rho_2, \rho_3, \rho_5 \} \]
Place - invariants
Predicate/transition nets

- Goal: compact representation of complex systems.
- Key changes:
  - Tokens are becoming individuals;
  - Transitions enabled if functions at incoming edges true;
  - Individuals generated by firing transitions defined through functions
- Changes can be explained by folding and unfolding C/E nets
Example: Dining philosophers problem

- $n > 1$ philosophers sitting at a round table;
- $n$ forks,
- $n$ plates with spaghetti;
- philosophers either thinking or eating spaghetti (using left and right fork).

How to model conflict for forks?

How to guarantee avoiding starvation?

2 forks needed!
Condition/event net model of the dining philosophers problem

- Let $x \in \{1..3\}$
- $t_x$: $x$ is thinking
- $e_x$: $x$ is eating
- $f_x$: fork $x$ is available

Model quite clumsy. Difficult to extend to more philosophers.
Predicate/transition model of the dining philosophers problem (1)

- Let $x$ be one of the philosophers,
- let $l(x)$ be the left spoon of $x$,
- let $r(x)$ be the right spoon of $x$.

- Tokens individuals
- Edges can be labeled with variables and functions
Predicate/transition model of the dining philosophers problem (1)
Predicate/transition model of the dining philosophers problem (2)

- Model can be extended to arbitrary numbers of people.
- No change of the structure.
Time and Petri Nets

- e.g.: Petri nets tell us that "a new request can be issued only after the resource is released"

- Nothing about time

- In literature, time has been added to PNs in many different ways (notion of temporal constraints for: transitions, places, arcs) → TPN
Timed Petri Nets

- **TPN**
  - Each transition is defined precisely based on connectivity and tokens needed for transition
  - Given an initial condition, the exact system state at an arbitrary future time $T$ can be determined

- Timed Petri Nets becomes a 7-tuple system
  - $PN = (P, T, F, W, K, M_0, \tau)$
  - $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a finite set of deterministic time delays to corresponding $t_i$
Time and Petri Nets (TPN)

- adding (quantitative) time to PNs is to introduce temporal constraints on its elements:
  - e.g., a transition must fire after 5 msec
Production system - Top level petri net
magazine/depot
NC axis
Evaluation

- **Pros:**
  - Appropriate for distributed applications,
  - Well-known theory for formally proving properties,

- **Cons:**
  - PN problems with modeling timing (extensions in TPN)
  - no programming elements, no hierarchy (extensions available)

- **Extensions:**
  - Enormous amounts of efforts on removing limitations.

- **Remark:**
  - A FSM can be represented by a subclass of Petri nets, where each transition has exactly one incoming edge and one outgoing edge.
Summary

- Petri nets: focus on causal dependencies
  - Condition/event nets
    - Single token per place
  - Place/transition nets
    - Multiple tokens per place
  - Predicate/transition nets
    - Tokens become individuals
    - Dining philosophers used as an example
  - Extensions required to get around limitations
SDL - Specification and Description Language
SDL - Specification and Description Language

- Used here as a (prominent) example of a model of computation based on asynchronous message passing communication.
- Appropriate also for distributed systems.
- Language designed for specification of distributed systems.
  - Dates back to early 70s,
  - Formal semantics defined in the late 80s,
  - Defined by ITU (International Telecommunication Union): Z.100 recommendation in 1980
- Another acronym SDL (“System Design Languages”)
SDL - Specification and Description Language

- Provides textual (tool processing) and graphical formats (user interaction)

- Ability to be used as a wide spectrum language from requirements to implementation

- Just like StateCharts, it is based on the CFSM (Communicating FSM) model of computation; each FSM is called a process.

- With SDL the protocol behaviour is completely specified by communicating FSM.

- The formal basis of SDL enables the use of code generation tool chains, which allows an automated implementation of the specification.
SDL - Specification and Description Language

- However, it uses message passing instead of shared memory for communications.

- SDL supports operations on data.

- object-oriented description of components.
Structuring SDL designs

SDL systems can be structured in various means:

- A system consists of a number of blocks connected by channels, each block may contain a substructure of blocks or it may contain process sets connected by signals.

- Processes execute concurrently with other processes and communicate by exchanging signals; or by remote procedure calls.
Specifying behaviour

1. The behaviour of a process is described as an extended FSM: When started, a process executes its start transition and enters the first state. (triggered by signals)

2. In transitions, a process may execute actions.

3. E.g.: Actions can assign values to variable attributes of a process, branch on values of expression, call procedures, create new processes, send signal to other processes.
SDL-representation of FSMs/processes
Communication among SDL-FSMs

- Communication between FSMs (or “processes“) is based on message-passing, assuming a potentially indefinitely large FIFO-queue.

- Each process fetches next entry from FIFO,
- checks if input enables transition,
- if yes: transition takes place,
- if no: input is ignored (exception: SAVE-mechanism).
Determinate?

- Let tokens be arriving at FIFO at the same time:
  - Order in which they are stored, is unknown:

  ![Diagram showing processes](image)

  All orders are legal: simulators can show different behaviors for the same input, all of which are correct.
Operations on data

- Variables can be declared locally for processes.
- Their type can be predefined or defined in SDL itself.
- SDL supports abstract data types (ADTs). Examples:
Interaction between processes can be described in process interaction diagrams (special case of block diagrams).

In addition to processes, these diagrams contain channels and declarations of local signals.

Example:
Designation of recipients

1. **Through process identifiers:**
   Example: OFFSPRING represents identifiers of processes generated dynamically.

2. **Explicitly:**
   By including the channel name.

3. **Implicitly:**
   If signal names imply channel names (B → Sw1)
Hierarchy in SDL

- Process interaction diagrams can be included in **blocks**. The root block is called **system**.

Processes cannot contain other processes, unlike in StateCharts.
Hierarchy of a SDL specification
Timers

- Timers can be declared locally. Elapsed timers put signal into queue (not necessarily processed immediately).
- RESET removes timer (also from FIFO-queue).
SDL application

The semantics of SDL defines the state space of the specification. This state space can be used for various analyses and transformation techniques, e.g.:

- state space exploration, simulation
- checking the SDLSpecification for deadlocks/lifelocks
- deriving test cases automatically
- code generation for an executable prototype or end system
Summary

- MoC: finite state machine components + non-blocking message passing communication
- Representation of processes
- Communication & block diagrams
- Timers and other language elements
- Excellent for distributed applications (e.g., Integrated Services Digital Network (ISDN))
- Commercial tools available from SINTEF, Telelogic, Cinderella (//www.cinderella.dk)