Industrial Automation
(Automação de Processos Industriais)

Discrete Event Systems

http://users.isr.ist.utl.pt/~jag/courses/api1718/api1718.html

Prof. Paulo Jorge Oliveira, original slides
Prof. José Gaspar, rev. 2017/2018
Syllabus:

Chap. 5 – CAD/CAM and CNC [1 week]

...  
Chap. 6 – Discrete Event Systems [2 weeks]  
Discrete event systems modeling. Automata. Petri Nets: state, dynamics, and modeling. Extended and strict models. Subclasses of Petri nets. ...

Chap. 7 – Analysis of Discrete Event Systems [2 weeks]
Some pointers to Discrete Event Systems

History:  
http://prosys.changwon.ac.kr/docs/petrinet/1.htm

Tutorial:  
http://vita.bu.edu/cgc/MIDEDS/  
http://www.daimi.au.dk/PetriNets/

Analyzers,  
and  
http://www.ppgia.pucpr.br/~maziero/petri/arp.html (in Portuguese)

Simulators:  

http://www.informatik.hu-berlin.de/top/pnk/download.html

Bibliography:  
* Discrete Event Systems - Modeling and Performance Analysis,  

* Petri Net Theory and the Modeling of Systems,  

* Petri Nets and GRAFCET: Tools for Modeling Discrete Event Systems  
R. David, H. Alla, Prentice-Hall, 1992
Generic characterization of systems resorting to input / output relations

State equations:

\[
\begin{align*}
\dot{x}(t) &= f(x(t), u(t), t) \\
y(t) &= g(x(t), u(t), t)
\end{align*}
\]

in continuous time (or in discrete time)

Examples?

Figure 1.1. Simple modeling process.
Control: Open loop vs closed-loop (⇔ the use of feedback)

![Diagram of open-loop and closed-loop systems]

**Figure 1.17.** Open-loop and closed-loop systems.

*Advantages of feedback?*

*(to revisit during DES supervision study)*
Example of closed-loop with feedback

Figure 1.18. Flow system of Example 1.11 and closed-loop control model.
Discrete Event Systems: Examples

Set of events:

\[ E = \{N, S, E, W\} \]

Figure 1.20. Random walk on a plane for Example 1.12.

Figure 1.21. Event-driven random walk on a plane.
Discrete Event Systems: Examples

Queueing systems

Set of events:
\( E = \{ \text{arrival, departure} \} \)
Discrete Event Systems: Examples

Computational Systems

Processes Arrival

CPU

Processes Departure

CPU

Processes
Characteristics of systems with continuous variables

1. State space is continuous

2. The state transition mechanism is time-driven

Characteristics of systems with discrete events (DES)

1. State space is discrete

2. The state transition mechanism is event-driven

Intrinsic characteristic of discrete events systems: Polling is avoided!
Taxonomy of Systems

Figure 1.29. Major system classifications.
Levels of abstraction in the study of Discrete Event Systems

Language of a “chocolate selling machine”:

(i) Waiting for a coin.
(ii) Received 1 euro coin.
    Chocolate A given. Go to (i).
(iii) Received 2 euro coin.
    Chocolate B given. Go to (i).

4 sensors:
Received 1 euro coin,
Received 2 euro coin,
Chocolate A given,
Chocolate B given.
Systems Theory Objectives

• Modeling and Analysis
• *Design* and synthesis
• Control / Supervision
• Performance assessment and robustness
• Optimization

Applications of Discrete Event Systems

• Queueing systems
• Operating systems and computers
• Telecommunications networks
• Distributed databases
• Automation
Discrete Event Systems

Typical modeling methodologies

Automata

GRAFCET / SFC

Petri nets

Augmenting in modeling capacity & complexity
Automata Theory and Languages

Genesis of computation theory

**Definition:** A language \( L \), defined over the alphabet \( E \) is a set of *strings* of finite length with events from \( E \).

Examples:

\[ E = \{ \alpha, \beta, \gamma \} \]

\[ L_1 = \{ \varepsilon, \alpha\alpha, \alpha\beta, \gamma\beta\alpha \} \], where \( \varepsilon \) is the null/empty string

\[ L_2 = \{ \text{all strings of length 3} \} \]

How to build a machine that “talks” a given language?

or

What language “talks” a system?
Operations / Properties of languages

$E^* = \text{Kleene-closure of } E$: set of all strings of finite length of $E$, including the null element $\varepsilon$.

Concatenation of $L_a$ and $L_b$:

$$L_a L_b := \left\{ s \in E^* : s = s_a s_b , s_a \in L_a , s_b \in L_b \right\}$$

Prefix-closure of $L \subseteq E^*$:

$$\bar{L} := \left\{ s \in E^* : \exists t \in E^* \; st \in L \right\}$$
Operations / Properties of languages

Example 2.1 (Operations on languages)
Let $E = \{a, b, g\}$, and consider the two languages $L_1 = \{\varepsilon, a, abb\}$ and $L_4 = \{g\}$. Neither $L_1$ nor $L_4$ are prefix-closed, since $ab \notin L_1$ and $\varepsilon \notin L_4$. Then:

\[
\begin{align*}
L_1L_4 &= \{g, ag, abbg\} \\
\overline{L_1} &= \{\varepsilon, a, ab, abb\} \\
\overline{L_4} &= \{\varepsilon, g\} \\
L_1\overline{L_4} &= \{\varepsilon, a, abb, g, ag, abbg\} \\
L_4^* &= \{\varepsilon, g, gg, ggg, \ldots\} \\
L_1^* &= \{\varepsilon, a, abb, aa, aabb, abba, abbb, \ldots\}
\end{align*}
\]

[Cassandras99]
Motivation: An automaton is a device capable of representing a language according to some rules.

Definition: A deterministic automaton is a 5-tuple

$$(E, X, f, x_0, F)$$

where:

- $E$ - finite alphabet (or possible events)
- $X$ - finite set of states
- $f$ - state transition function $f: X \times E \rightarrow X$
- $x_0$ - initial state $x_0 \subseteq X$
- $F$ - set of final states or marked states $F \subseteq E$

Word of caution: the word “state” is used here to mean “step” (Grafcet) or “place” (Petri Nets)
Example 1 of an automaton:

\[(E, X, f, x_0, F)\]

\[E = \{\alpha, \beta, \gamma\}\]

\[X = \{x, y, z\}\]

\[x_0 = x\]

\[F = \{x, z\}\]

\[f(x, \alpha) = x \quad f(x, \beta) = z \quad f(x, \gamma) = z\]

\[f(y, \alpha) = x \quad f(y, \beta) = y \quad f(y, \gamma) = y\]

\[f(z, \alpha) = y \quad f(z, \beta) = z \quad f(z, \gamma) = y\]
Example 2 of a stochastic automaton

$$(E, X, f, x_0, F)$$

$E = \{\alpha, \beta\}$

$X = \{0, 1\}$

$x_0 = 0$

$F = \{0\}$

Figure 2.4. State transition diagram for the nondeterministic automaton of Example 2.7.

$f(0, \alpha) = \{0, 1\}$  
$f(0, \beta) = \{\}$

$f(1, \alpha) = \{\}$  
$f(1, \beta) = 0$
Given an automaton

\[ G = (E, X, f, x_0, F) \]

the **Generated Language** is defined as

\[ L(G) := \{ s \in E^* : f(x_0, s) \text{ is defined} \} \]

*Note: if \( f \) is always defined for all events then \( L(G) = E^* \)*

and the **Marked Language** is defined as

\[ L_m(G) := \{ s \in E^* : f(x_0, s) \in F \} \]
Example 3: marked language of an automaton

\[ L(G) := \{ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, baa, \ldots \} \]

\[ L_m(G) := \{ a, aa, ba, aaa, baa, bba, \ldots \} \]

Concluding, in this example \( L_m(G) \) means all strings with events \( a \) and \( b \), ended by event \( a \).
Automata equivalence:

The automata $G_1$ and $G_2$ are equivalent if

\[ L(G_1) = L(G_2) \]

and

\[ L_m(G_1) = L_m(G_2) \]
Example 4: two equivalent automata

Objective: To validate a sequence of events

Figure 2.6. State transition diagrams for digit sequence detector in Example 2.9.
Deadlocks (inter-blocagem)

Example 5:

The state 5 is a **deadlock**.

The states 3 and 4 constitute a **livelock**.

How to find the **deadlocks** and the **livelocks**?

Need methodologies for the analysis of **Discrete Event Systems**
Deadlock:

in general the following relations are verified

\[ L_m(G) \subseteq \overline{L}_m(G) \subseteq L(G) \]

An automaton G has a deadlock if

\[ \overline{L}_m(G) \subseteq L(G) \]

and is not blocked when

\[ \overline{L}_m(G) = L(G) \]
Deadlock:

Example:

\[ L_m(G) = \{ab, abgab, abgabgab, \ldots\} \]

\[ L(G) = \{\varepsilon, a, ab, ag, aa, aab, abg, aaba, abga, \ldots\} \]

\[ (L_m(G) \subset L(G)) \]

The state 5 is a deadlock.

The states 3 and 4 constitute a livelock.
Alternative way to detect deadlocks:

Example:

The state 5 is a *deadlock*.

The states 3 and 4 constitute a *livelock*.
Timed Discrete Event Systems

Figure 3.10. The event scheduling scheme.
## Examples of Automata Classes and Applications

<table>
<thead>
<tr>
<th>Automaton Class</th>
<th>Recognizable language</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite state machine (FSM), e.g. Moore machines or Mealy machines</td>
<td>Regular languages</td>
<td>Text processing, compilers, and hardware design</td>
</tr>
<tr>
<td>Pushdown automaton (PDA)</td>
<td>Context-free languages</td>
<td>Programming languages, artificial intelligence, (originally) study of the human languages</td>
</tr>
<tr>
<td>Turing machine (nondeterministic, deterministic, multitape, ...)</td>
<td>Recursively enumerable languages</td>
<td>Theory, complexity</td>
</tr>
</tbody>
</table>
**Petri nets**

Developed by Carl Adam Petri in his PhD thesis in 1962.

**Definition:** A marked Petri net is a 5-tuple

\[(P, T, A, w, x_0)\]

where:

- **P** - set of places
- **T** - set of transitions
- **A** - set of arcs \(A \subseteq (P \times T) \cup (T \times P)\)
- **w** - weight function \(w: A \to \mathbb{N}\)
- **x_0** - initial marking \(x_0: P \to \mathbb{N}\)

[Cassandras93]
Example of a Petri net

\[(P, T, A, w, x_0)\]

\[P=\{p_1, p_2, p_3, p_4, p_5\}\]

\[T=\{t_1, t_2, t_3, t_4\}\]

\[A=\{(p_1, t_1), (t_1, p_2), (t_1, p_3), (p_2, t_2), (p_3, t_3),
(t_2, p_4), (t_3, p_5), (p_4, t_4), (p_5, t_4), (t_4, p_1)\}\]

\[w(p_1, t_1)=1, w(t_1, p_2)=1, w(t_1, p_3)=1, w(p_2, t_2)=1
w(p_3, t_3)=2, w(t_2, p_4)=1, w(t_3, p_5)=1, w(p_4, t_4)=3
w(p_5, t_4)=1, w(t_4, p_1)=1\]

\[x_0 = \{1, 0, 0, 2, 0\}\]
Petri nets

Rules to follow (mandatory):

• **Arcs** (directed connections)
  connect **places** to **transitions** and
  connect **transitions** to **places**

• A **transition** can have no **places** directly as inputs (source),
  *i.e. must exist arcs between transitions and places*

• A **transition** can have no **places** directly as outputs (sink),
  *i.e. must exist arcs between transitions and places*

• The same happens with the input and output **transitions** for **places**
Alternative definition of a Petri net

A marked Petri net is a 5-tuple

\[(P, T, I, O, \mu_0)\]

where:
- \(P\) - set of places
- \(T\) - set of transitions
- \(I\) - transition input function \(I : T \rightarrow P^\infty\)
- \(O\) - transition output function \(O : T \rightarrow P^\infty\)
- \(\mu_0\) - initial marking \(\mu_0 : P \rightarrow N\)

Note: \(P^\infty\) = bag of places (is more general than a set of places)
Example of a Petri net and its graphical representation

Alternative definition

\[(P, T, I, O, \mu_0)\]

\[P=\{p_1, p_2, p_3, p_4, p_5\}\]

\[T=\{t_1, t_2, t_3, t_4\}\]

\[I(t_1)=\{p_1\} \quad O(t_1)=\{p_2, p_3\}\]

\[I(t_2)=\{p_2\} \quad O(t_2)=\{p_4\}\]

\[I(t_3)=\{p_3, p_3\} \quad O(t_3)=\{p_5\}\]

\[I(t_4)=\{p_4, p_4, p_4, p_5\} \quad O(t_4)=\{p_1\}\]

\[\mu_0 = \{1, 0, 0, 2, 0\}\]
Petri nets: State, Markings, Weights of Arcs

The state of a Petri net is characterized by the marking of all places.

The set of all possible markings of a Petri net corresponds to its state space.

How does the state of a Petri net evolve?

Nomenclature for the markings and cardinality (weight) of the arcs:

\[ n = \#(p_i, I(t_j)) \]
\[ m = \#(p_i, O(t_j)) \]
Execution Rules for Petri Nets (Dynamics of Petri nets)

A transition \( t_j \in T \) is \textit{enabled} if:

\[
\forall p_i \in P : \quad \mu(p_i) \geq \#(p_i, I(t_j))
\]

A transition \( t_j \in T \) may \textit{fire} whenever enabled, resulting in a new marking given by:

\[
\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))
\]

\#(p_i, I(t_j)) = \text{multiplicity of the arc from } p_i \text{ to } t_j
\#(p_i, O(t_j)) = \text{multiplicity of the arc from } t_j \text{ to } p_i

[Peterson81 §2.3]
Execution Rules for Petri Nets
(Dynamics of Petri nets)

\[ t_j \]

\[ m \rightarrow n \quad 2 \rightarrow 1 \equiv \downarrow \downarrow \]

\[ p_i \]

\[ n = \#(p_i, I(t_j)) \]

\[ m = \#(p_i, O(t_j)) \]

\[ \mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j)) \]

[Peterson81 §2.3]

Later this dynamic equation will be generalized using vector notation \( \mu_{k+1} = \mu_k + (D^+ - D^-)q_k \)
Petri nets

Example of evolution of a Petri net

Initial marking:

\[ \mu_0 = \{1, 0, 1, 2, 0\} \]

This discrete event system can not change state.

It is in a *deadlock*!
Petri nets: Conditions and Events (Places and Transitions)

Example: Machine waits until an order appears and then machines the ordered part and sends it out for delivery.

Conditions:

a) The server is idle.
b) A job arrives and waits to be processed
c) The server is processing the job
d) The job is complete

Events

1) Job arrival
2) Server starts processing
3) Server finishes processing
4) The job is delivered

<table>
<thead>
<tr>
<th>Event</th>
<th>Pre-conditions</th>
<th>Pos-conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a, b</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>d, a</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>-</td>
</tr>
</tbody>
</table>
Discrete Event Systems

Example of a simple automation system modeled using PNs

An automatic soda selling machine accepts 50c and $1 coins and sells 2 types of products: SODA A, that costs $1.50 and SODA B, that costs $2.00.

Assume that the money return operation is omitted.

$p_1$: machine with $0.00;
$t_1, t_3, t_5, t_7$: coin of 50c introduced;
$t_2, t_4, t_6$: coin of $1 introduced;
$t_9$: SODA A sold, $t_8$: SODA B sold.
Petri nets: **Modeling mechanisms**

**Concurrence**

```
  t_1

          /
         /
        /
      t_2
```

**Conflict**

```
  t_1

          /
         /
        /
      t_2
```
Petri nets: **Modeling mechanisms**

**Mutual Exclusion**

Place $m$ represents the permission to enter the critical section

**Producer / Consumer**

$B =$ buffer holding produced parts
Petri nets: Modeling mechanisms

Producer / Consumer with finite capacity

s Readers / t Writers
Extensions to Petri nets

Switches [Baer 1973]

Possible to be implemented with restricted Petri nets.
Extensions to Petri nets

Inhibitor Arcs

Equivalent to

nets with priorities

Can be implemented with restricted Petri nets?

Zero tests...

Infinity tests...
Extensions to Petri nets

P-Timed nets

Job arrival

Start of processing

End of processing

Job is delivered

Jobs waits processing

Job is being processed

Job is complete

Server is idle
Extensions to Petri nets

T-Timed nets
Extensions to Petri nets

Stochastic nets

Stochastic switches

\[ q_0 + q_1 + q_2 = 1 \]

Transitions with stochastic timings described by a stochastic variable with known pdf

\[ \mu_0 \]
Discrete Event Systems

Sub-classes of Petri nets

State Machine:

Petri nets where each transition has exactly one input arc and one output arc.

Marked Graphs:

Petri nets where each place has lesser than or equal to one input arc and one output arc.
Discrete Event Systems

Example of DES:

Manufacturing system composed by 2 machines (M₁ and M₂) and a robotic manipulator (R). This takes the finished parts from machine M₁ and transports them to M₂.

No buffers available on the machines. If R arrives near M₁ and the machine is busy, the part is rejected.

If R arrives near M₂ and the machine is busy, the manipulator must wait.

Machining time: M₁=0.5s; M₂=1.5s; R_{M₁→M₂}=0.2s; R_{M₂→M₁}=0.1s;
Definition of places

- $M_1$ is characterized by places $x_1$
- $M_2$ is characterized by places $x_2$
- $R$ is characterized by places $x_3$

$x_1 = \{\text{Idle, Busy, Waiting}\}$
$x_2 = \{\text{Idle, Busy}\}$
$x_3 = \{\text{Idle, Carrying, Returning}\}$

Example of arrival of parts:

$$a(t) = \begin{cases} 1 & \text{in } \{0.1, 0.7, 1.1, 1.6, 2.5\} \\ 0 & \text{in } \text{other time stamps} \end{cases}$$
Discrete Event Systems

Example of DES:

Definition of events:

\[ a_1 \] - loads part in \( M_1 \)
\[ d_1 \] - ends part processing in \( M_1 \)
\[ r_1 \] - loads manipulator
\[ r_2 \] - unloads manipulator and loads \( M_2 \)
\[ d_2 \] - ends part processing in \( M_2 \)
\[ r_3 \] - manipulator at base
Chap. 6 – Discrete Event Systems

Discrete Event Systems

\[ x_1 = \{\text{Idle, Busy, Waiting}\} \]
\[ x_2 = \{\text{Idle, Busy}\} \]
\[ x_3 = \{\text{Idle, Carrying, Returning}\} \]
Discrete Event Systems

Example of DES:

Events:

- $a_1$ - loads part in $M_1$
- $d_1$ - ends part processing in $M_1$
- $r_1$ - loads manipulator
- $r_2$ - unloads manipulator and loads $M_2$
- $d_2$ - ends part processing in $M_2$
- $r_3$ - manipulator at base