UNIVERSITY OF MICHIGAN
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
LECTURE NOTES FOR EECS 661
CHAPTER 1: INTRODUCTION TO DISCRETE EVENT SYSTEMS

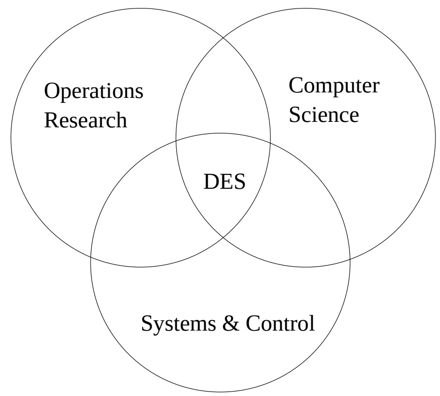
Stéphane Lafortune

September 2004

References for Chapter 1: Textbook, Chapter 1: Section 1.3

Discrete Event Systems

A Multidisciplinary Area:



What:

- Discrete State Space (logical, symbolic variables)
- Event-driven Dynamics

Why:

- Technological Systems, Computer Control

⟶ Large, Complex Systems: they need to be *analyzed*, *diagnosed*, *controlled*, and *optimized*

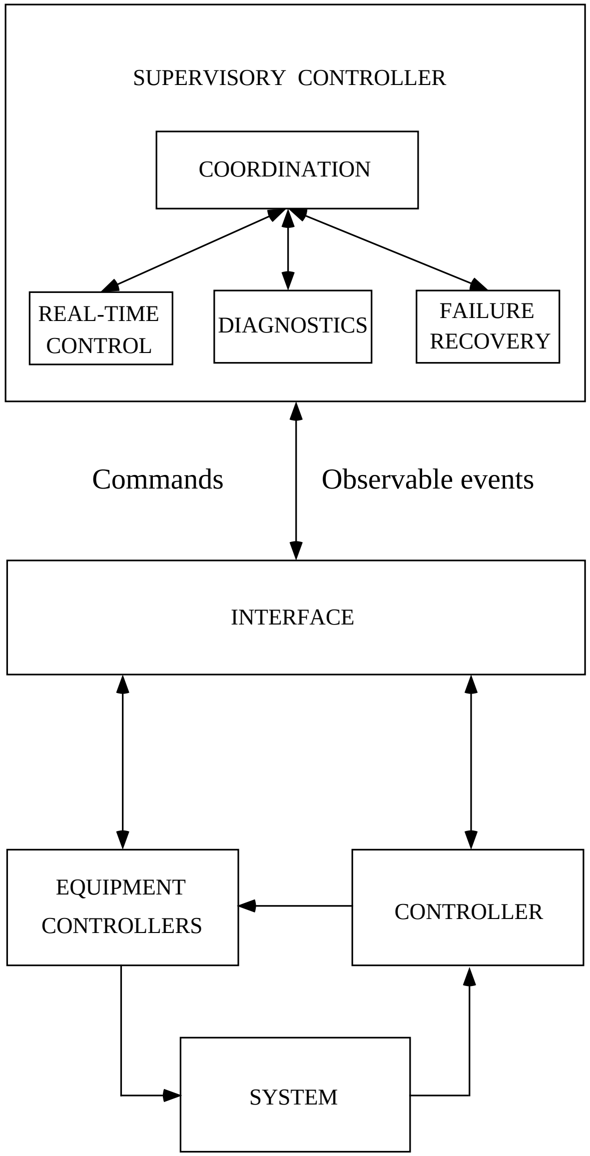
Where:

- Inherently Discrete Systems:
  computer systems, communication networks, automated manufacturing systems (cell and factory levels), software systems.
- Systems with Continuous and Discrete Variables (hybrid systems), modeled as DES at a certain level of abstraction, e.g., for the higher level control logic:
  process control, automated manufacturing systems (machine and cell levels); intelligent transportation systems, air traffic systems.
- *Embedded systems* ; *networked systems*.

How:

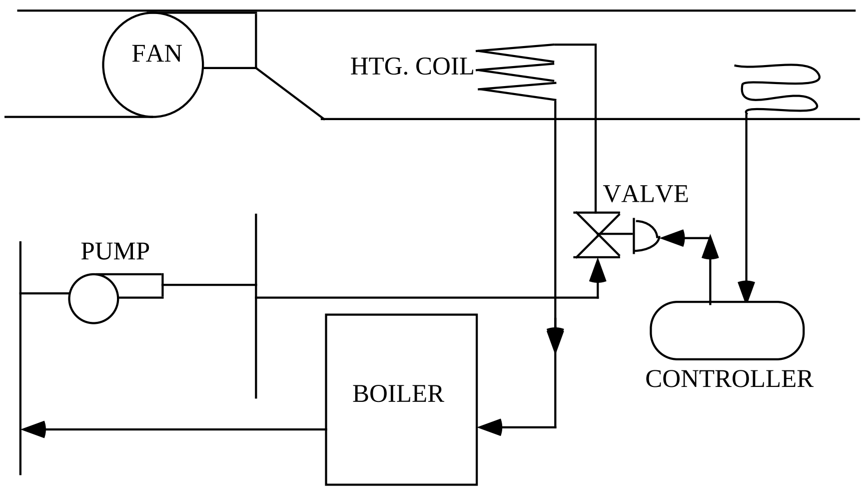
- Mathematical Modeling, Analysis, Verification, Diagnosis, Controller Design, Optimization, Simulation

Conceptual Control System Architecture:



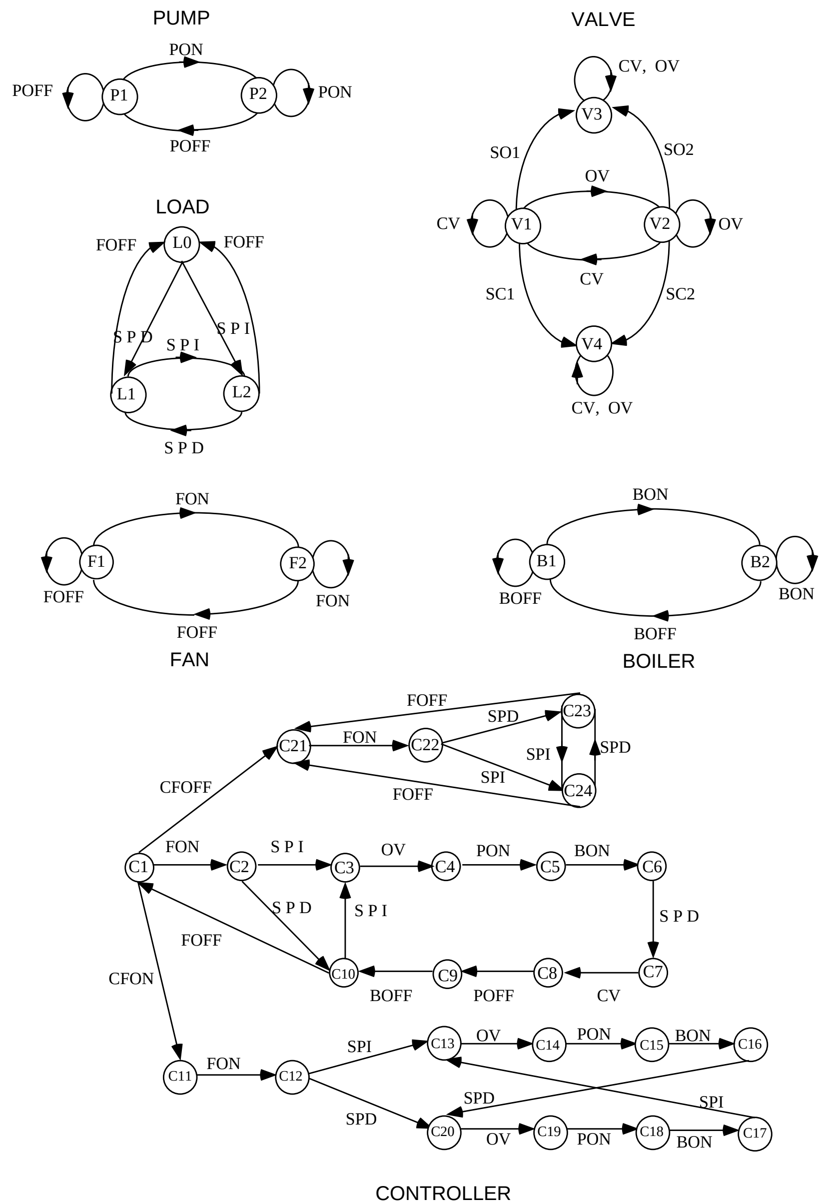
Some Examples

The Heating System of a Heating, Ventilation, and Air Conditioning (HVAC) Unit



- The operation of the unit is monitored by a set of sensors.
- The issue of interest: *Fault Diagnosis*.
- Specifically: diagnose occurrence of "sharp" faults during the on-line operation of the unit.
- Examples of faults: stuck failures of valves, on-off failures of pumps, controllers, sensors, etc.
- Implementation: diagnostics module in the control logic.

Models
of the Components of
the HVAC System:



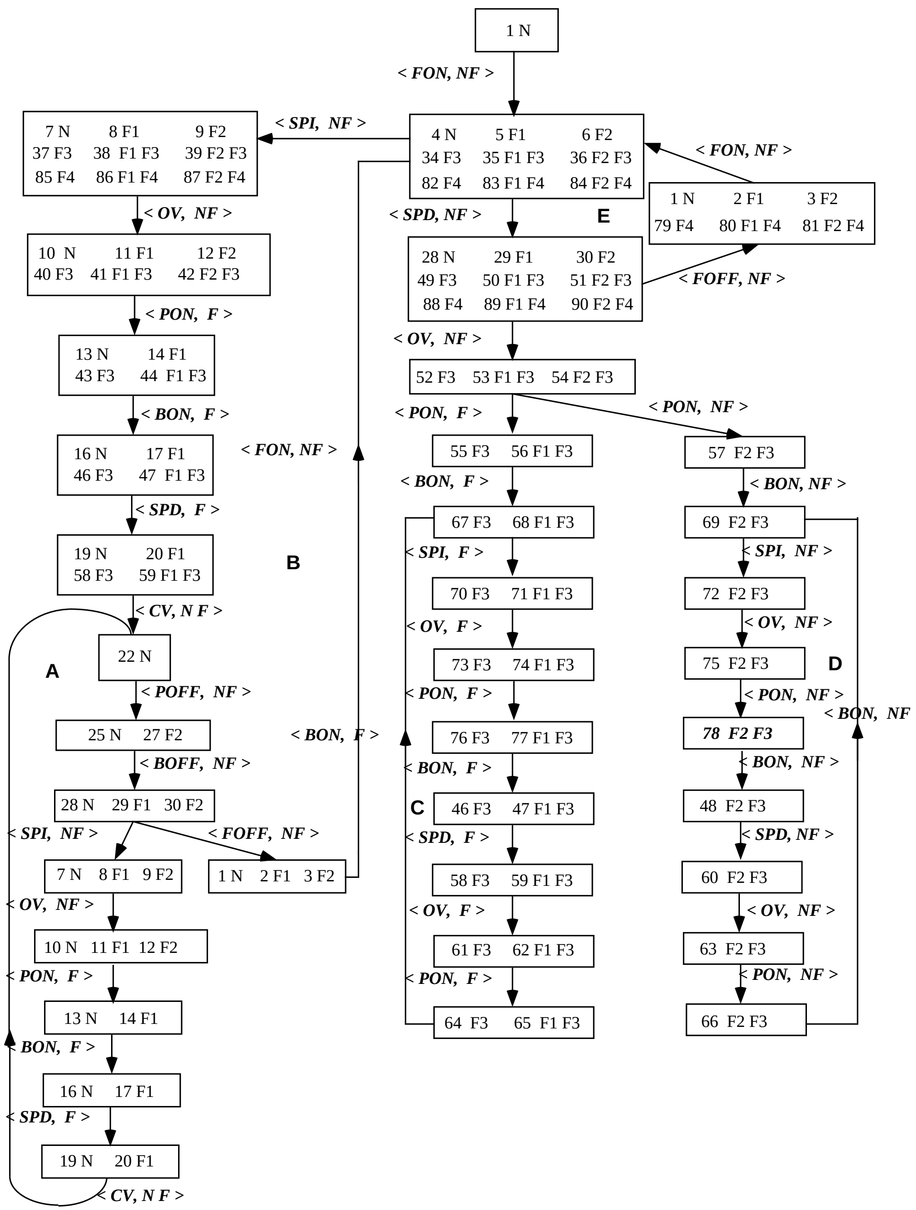
Part of the *Diagnoser* for the Heating System (HVAC Unit)

$F1$: SO

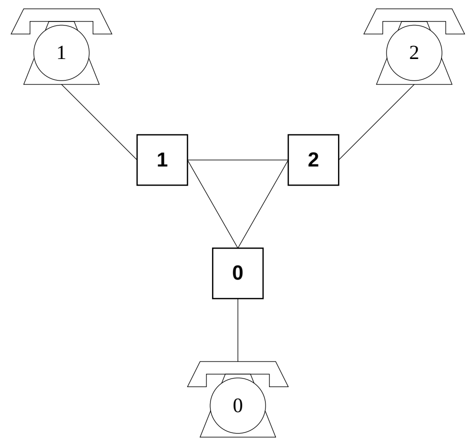
$F2$: SC

$F3$: CFON

$F4$: CFOFF

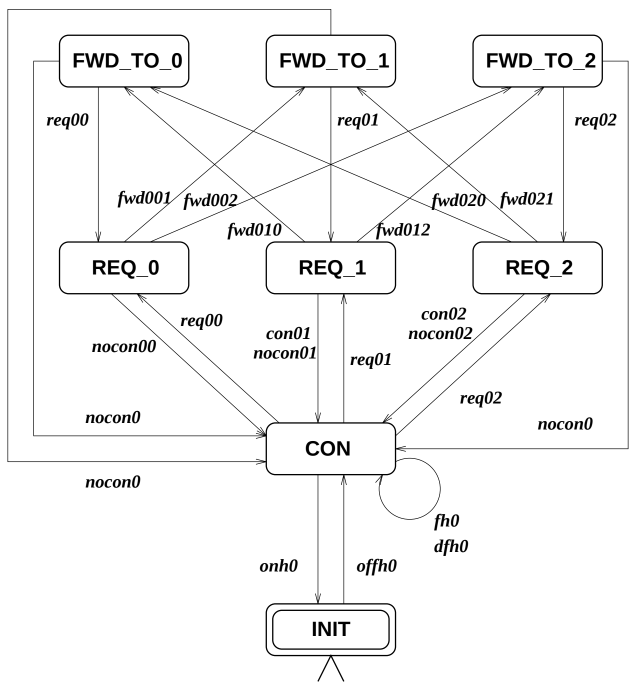


A "Small" Telephone System

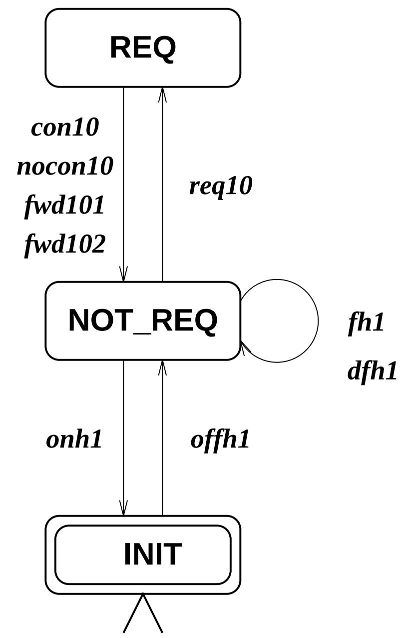


- The network has screening, forwarding, and multi-way calling capabilities.
- The issue of interest: *Feature Interactions*.
- Specifically: detection and resolution of logical conflicts (interactions) between options (features).
- Implementation: correct design of the (modular) software programs that run at the switches.

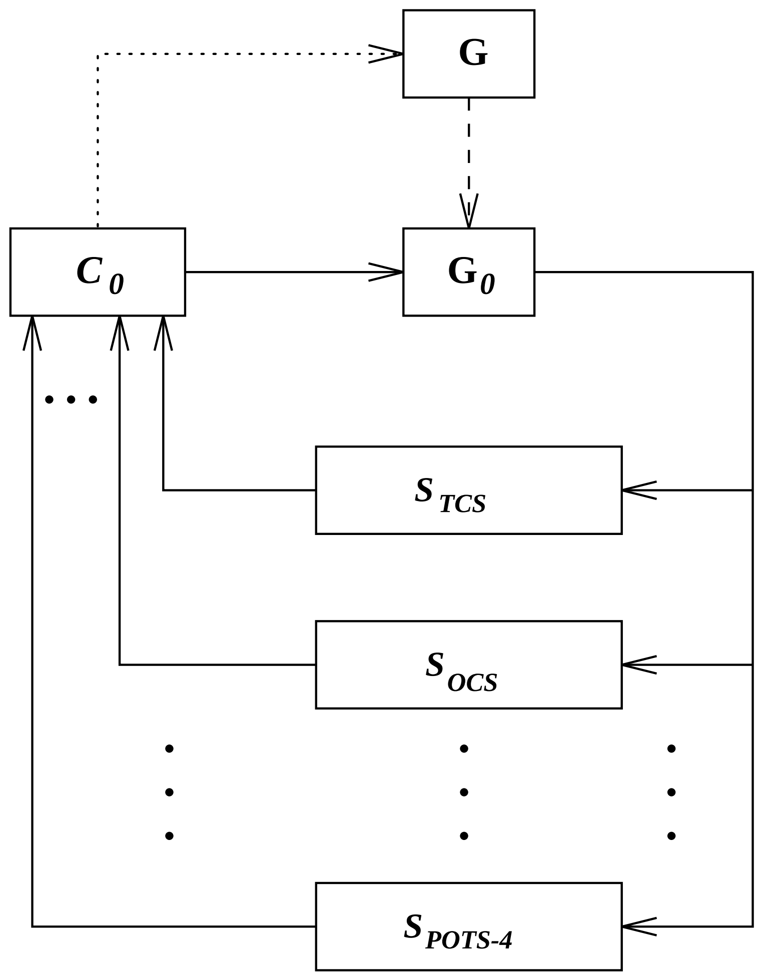
Model of User 0 in a Telephone System:



Model of User 1 at Switch 0 in Telephone System:



A Control Architecture for Approaching this Problem:



Other examples:

Railway Connections and Time Tables[1]

- The network of railway connections is closed and each line has a fixed number of trains. The inter-station travel times are known and deterministic.
- The objective is to design "satisfactory" time tables for the trains.
- Specifications include: certain trains have to wait for one another to allow change overs.
- Constraints: want system to operate fast, but also want perturbations to completely disappear in finite time.
- Issues of interest: how do perturbations to the time table propagate, what limits the minimum operation time, where would it be helpful to add trains, etc.
- Approach: write equations for the departure times of the trains, using "maximum" and "addition."

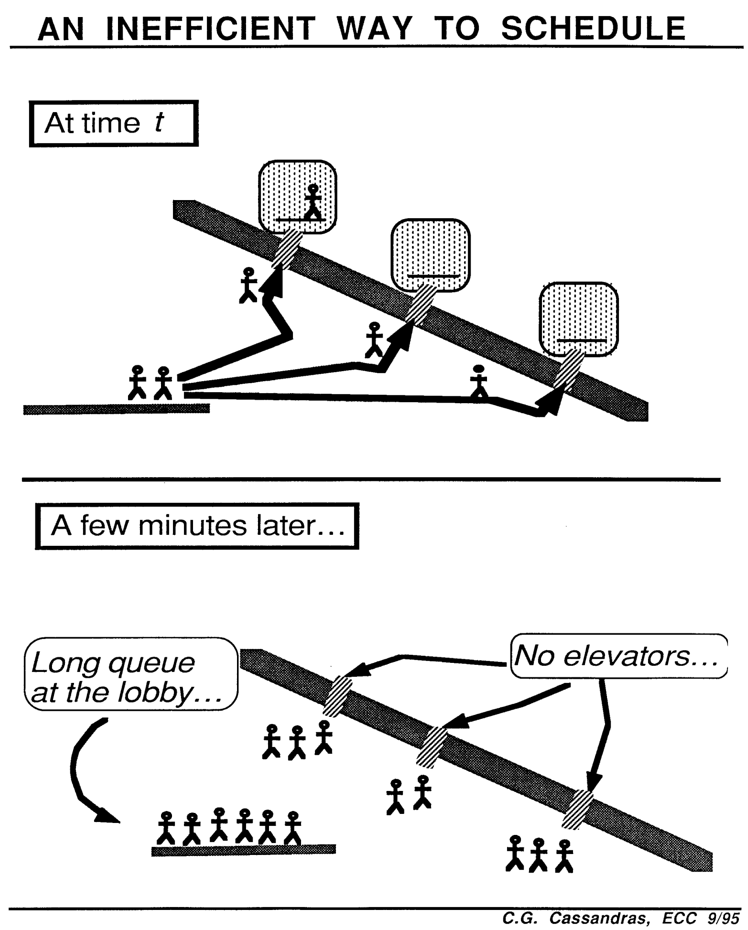
[1]Example due to G. J. Olsder

Dispatching Control in an Elevator System[2]

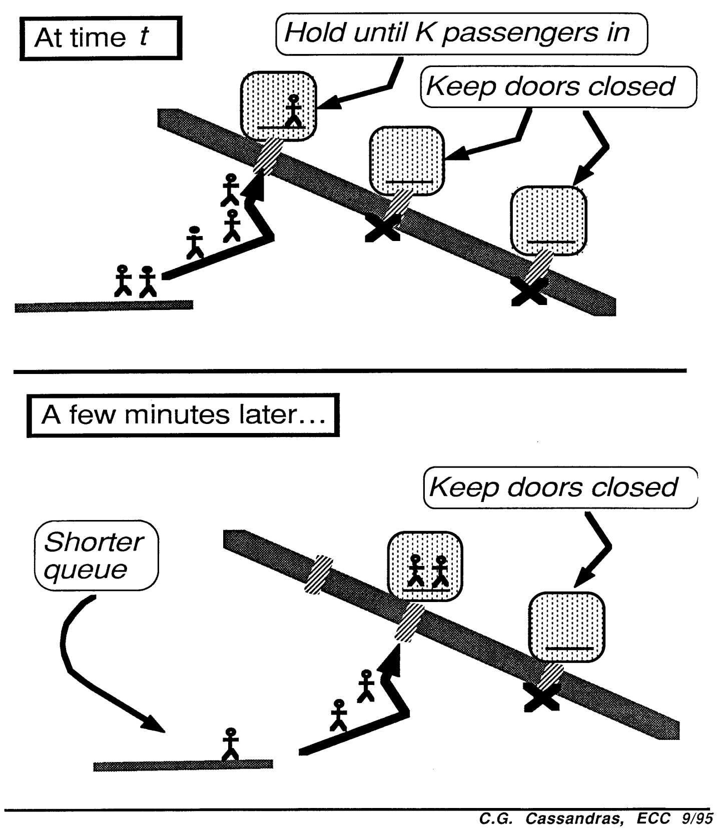
- Events: *hall_call, car_call, car_arrives_at_floor_i*, etc.
- States: position of car k, number of passengers waiting at floor i, etc. (very large state space!)
- Control problem: *which car to send where so as to achieve "satisfactory" performance?*
- Performance measures: *average* waiting time (until car comes), *average* service time (until car delivers to desired floor), fraction of passengers waiting more (on average) than one minute, etc.
- Probabilistic formulation: passenger arrival rates at floors, probability distribution for destination floors, load times and travel times, etc.
- Common solution: threshold-based control, i.e., hold a car until a *threshold* is reached.

  → The issue is then to determine this threshold and "automatically" adjust it in real-time, based on observed passenger arrival rates.

[2]Example due to C. Cassandras

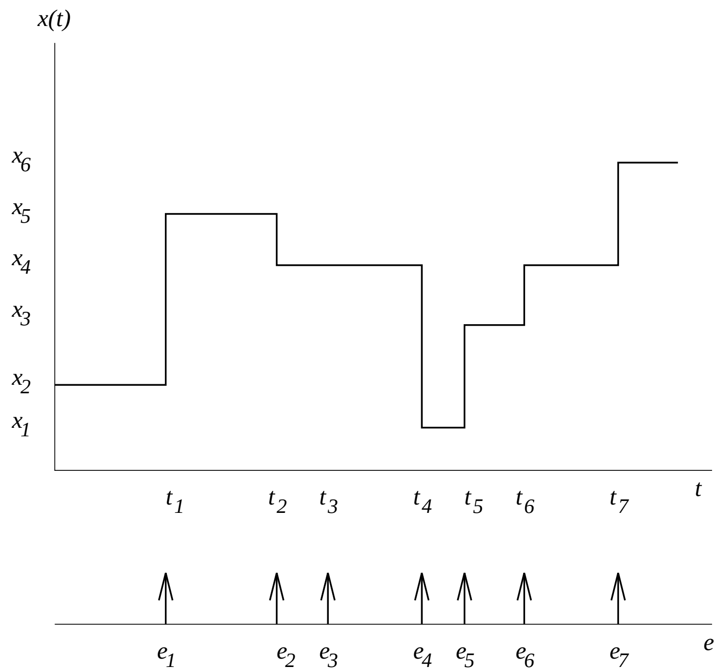

AN INEFFICIENT WAY TO SCHEDULE
At time t
A few minutes later...
Long queue at the lobby...
No elevators...
C.G. Cassandras, ECC 9/95

AN OBVIOUSLY BETTER WAY...



The Three Levels of Abstraction in Modeling DES

Sample Paths of Discrete Event Systems



Describe this sample path by the *timed sequence of events* that it contains:

$$s_e^t = (e_1, t_1)(e_2, t_2)(e_3, t_3)(e_4, t_4)(e_5, t_5)(e_6, t_6)(e_7, t_7)$$

The behavior of a given DES is described as follows:

- **Timed Language:** set of all timed sequences of events that the DES can generate/execute
- **Stochastic Timed Language:** a timed language with a probability distribution function defined over it
- **Language:** a timed language where the timing information has been deleted, i.e., it is a set of sequences, or *traces*, of events.

$$s_e = e_1 e_2 e_3 e_4 e_5 e_6 e_7$$

Formal language theory:

- Finite set of events $E : \{e_1, e_2, \ldots, e_n\}$
- Set of all finite strings of event in E: E^* - Kleene-closure
- A *language* L is a subset of E^*: $L \subseteq E^*$

This leads to the three complemetary levels of abstraction at which DES are studied.

- **Logical level:** the *language* model is used to study properties that concern event ordering only; e.g., consider the *telephone system* example, as well as the HVAC unit example (diagnosis).

  Priorities, mutual exclusion, deadlock, livelock, occurrence of unobservable events, etc.

- **Temporal level:** the *timed language* model is used to study properties that concern the timing of the events; e.g., consider the *railway network* example.

  Deadlines, cycle times, effect of perturbations, etc.

- **Stochastic level:** the *stochastic timed language* model is used to study properties that concern the expected behavior of the system under the given statistical information; e.g., consider the *elevator* example.

  Average delay, throughput, and other relevant performance measures.

N.B.: *Discrete Event Simulation* usually refers to the stochastic level.
Question: How to *represent* [(stochastic) timed] languages?

Discrete Event Modeling Formalisms

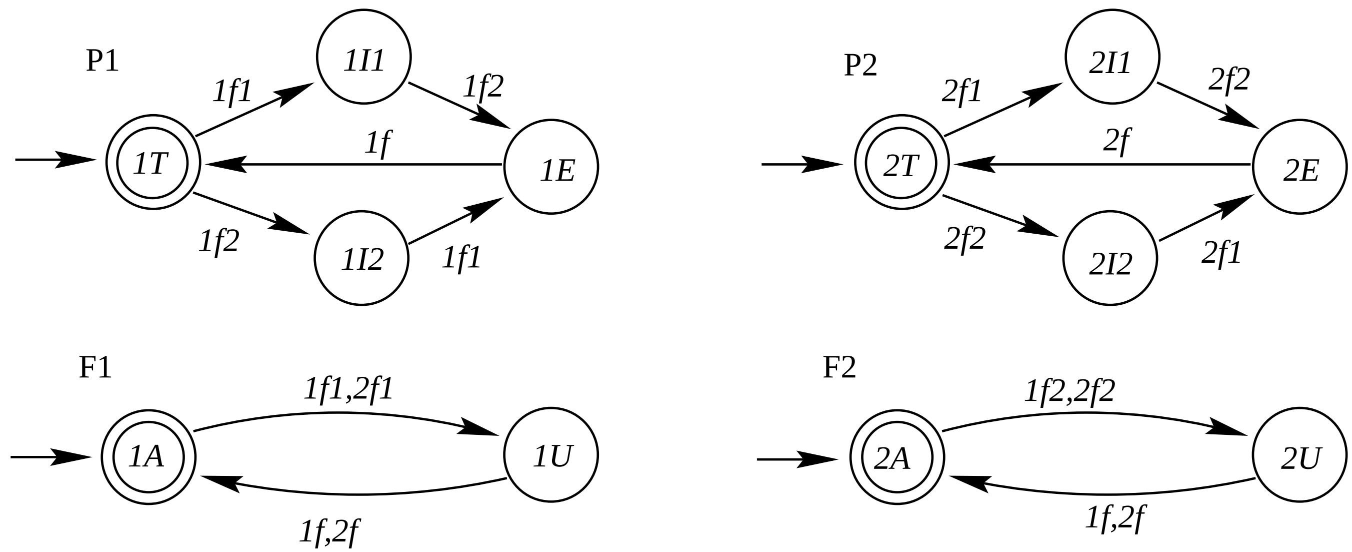
- Formal classes of models that represent [(stochastic) timed] languages
- "State-based" formalisms: define a state space and specify the state transition structure (i.e., *(out_state, event, in_state)* triples) that represents the language.

  Automata (or *State Machines*) and *Petri Nets* are widely used.
- "Trace-based" formalisms: use (recursive) algebraic equations on the events to represent the traces in the language (i.e., no explicit "state"). Often referred to as *Process Algebras*.

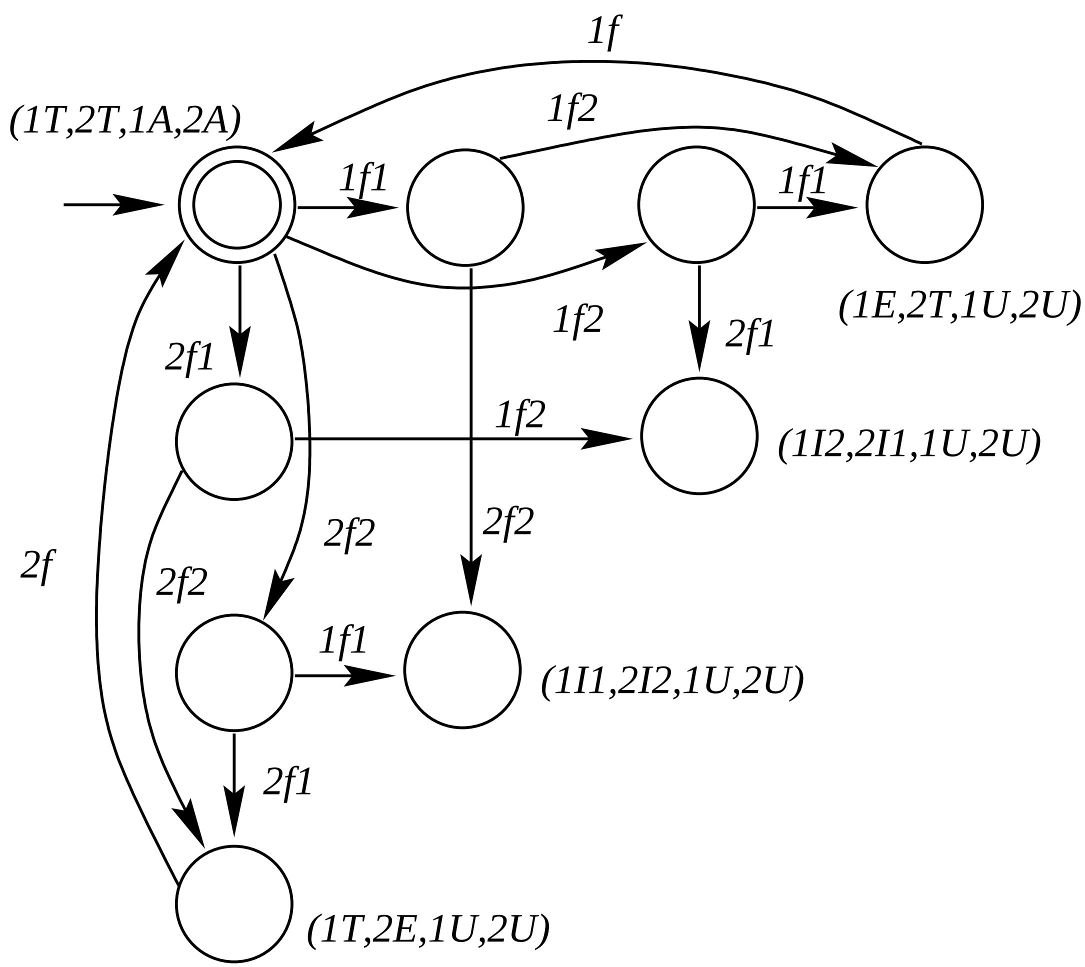
  Communicating Sequential Processes (CSP) is a well-know formalism in this category.
- We will study:
  - (untimed and timed) automata [modeling, analysis, diagnosis, supervisory control]
  - (untimed and timed) Petri nets [modeling, analysis, some control]
  - timed event graphs, a special case of timed Petri nets [analysis using *max-plus algebra*]

→ We illustrate the above modeling formalisms for the (familiar) example of the dining philosophers.

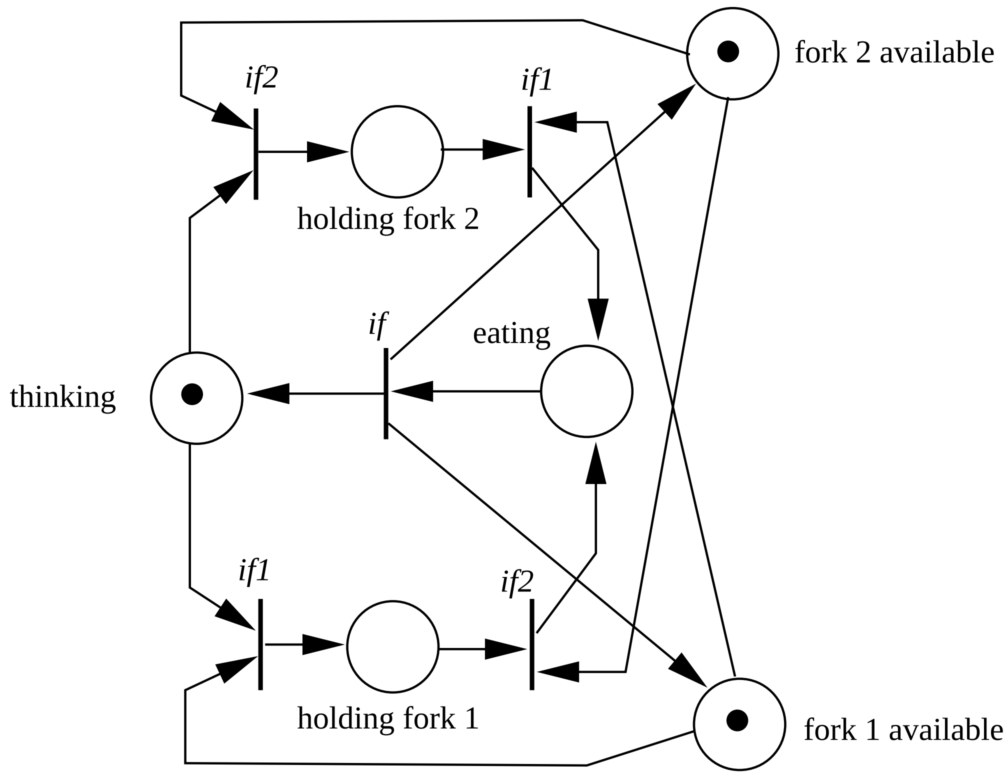
Automaton models of two philosophers ($P1$, $P2$) and two forks ($F1$, $F2$)



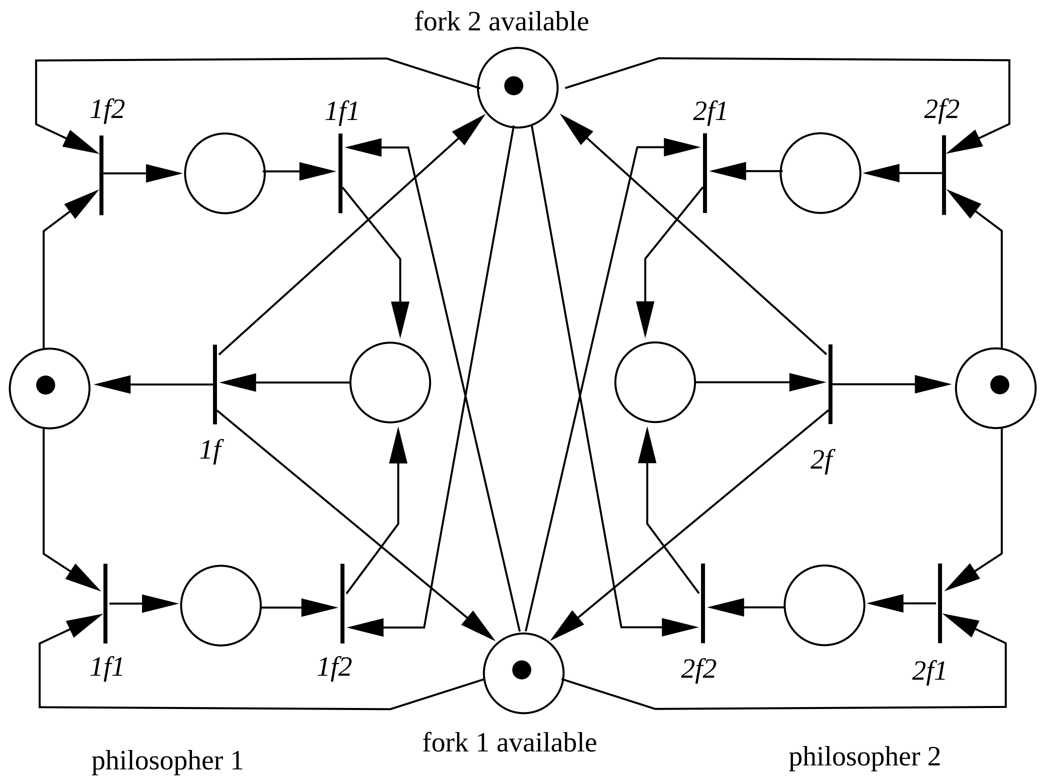
Composition of the four automata: $P1||P2||F1||F2$



Petri net model of one philosopher and two forks



Petri net model of two philosophers and two forks



Recursive equation model of two philosophers and two forks

$$
\begin{aligned}
P1 &= (1f1 \to 1f2 \to E1 \mid 1f2 \to 1f1 \to E1) \\
E1 &= (1f \to P1) \\
P2 &= (2f1 \to 2f2 \to E2 \mid 2f2 \to 2f1 \to E2) \\
E2 &= (2f \to P2) \\
F1 &= (1f1 \to 1f \to F1 \mid 2f1 \to 2f \to F1) \\
F2 &= (1f2 \to 1f \to F2 \mid 2f2 \to 2f \to F2) \\
SYSTEM &= P1||P2||F1||F2
\end{aligned}
$$

In general, we get a set of equations of the form:

$$
\begin{aligned}
X &= f(X) \\
Y &= g(X)
\end{aligned}
$$

where X is a vector of processes and f must contain $\to$.

How to Compare Modeling Formalisms?

Descriptive Power: Language complexity or class of languages that a (finite) model can represent.

- Finite-state automata: Regular Languages $\mathcal{R}$
- Labeled Petri Nets: $\mathcal{PNL} \supset \mathcal{R}$.

Algebraic Structure: Formal operations that permit to build complex systems by interconnecting simple systems and that allow to "manipulate" a model for analysis and synthesis purposes.

- $\mathcal{R}$ has nice properties: closed under union, concatenation, intersection, parallel composition, complementation w.r.t. E^*.

  These operations can be "implemented" using finite-state automata.
- $\mathcal{PNL}$ does not enjoy such nice properties.

  However, Petri nets have intrinsically modular structure: e.g., system decomposition by means of *place-bordered* Petri nets.