Petri nets are a tool for the study of systems. Petri net theory allows a system to be modeled by a Petri net. Petri nets are a formal, graphical, executable technique for the specification and analysis of concurrent, discrete-event dynamic systems, currently undergoing standardization. Analysis should reveal important system properties.

Note – in many fields phenomena cannot be studied directly but only through a model of the system. Astronomy, Nuclear physics, Sociology, Biology, Computer science. Petri nets are used in these fields.

April 2004
Features of Systems

- Made up of components
- Components can be systems
- Each component has a current state
- Components exhibit concurrency (or parallelism)
  - Activities of one component may occur simultaneously with others
- Concurrency creates difficulties
  - Since components interact behavior needs to be synchronized
  - The timing of components may be very complex and so the resulting interactions among components difficult to describe

- Petri nets were invented as a way of modeling systems while making it easy to handle concurrency

Features

- Relatively easy to match Petri nets to events/conditions
- Relatively easy to match Petri nets to flow charts
- Relatively each to match Petri nets to Gant charts

- Widely used in:
  - Systems engineering
  - Electrical engineering
  - Theoretical computer science
A Petri net is composed of 3 parts:
P places
T transitions
I inputs
O outputs
\( \mu \) tokens

A Petri net structure, \( C \), is a four-tuple \( C=(P,T,I,O) \)
A marked Petri net structure is a five-tuple \( M=(P,T,I,O,\mu) \)

Most theoretical work with Petri nets is done mathematically using graph theory
Most applied work is done with the graphical representation
Bipartite directed multi-graphs

In the dual of a Petri net graph transitions are replaced with places and places with transitions
Important from a theory perspective
So far no practical application

Inverse Petri net
Inputs and outputs are reversed
Sometimes used for thinking through issues

 Bundles
When inputs or outputs have high multiplicity they are bundled
Replace simple arrow with large
Simply a graphical drawing technique

A marking \( . \) is an assignment of a token to a place in the graph
Transitions

Transitions can be timed
- Need to execute immediately
- Can wait
- Immediate transitions have priority = that’s because they are primitive

Transitions can be
- Primitive – take 0 time to execute
- Non-primitive – take time to execute

WARNING: Wait – in Petri net theory all transactions take 0 time but in life – transactions = events and events take time ...???

Solution – divide non-primitive transition into two parts
- Non-primitive transition starts
- Non-primitive transition ends
- Add a new place – non-primitive transition is occurring
- Represent non-primitive with a “box”

April 2004

Tokens

Used to denote execution
- Assigned to places
- There can be an unlimited number tokens assigned to any place

\[ \mu = (1,2,0,0,1) \]

April 2004
Execution Rules

- Execution of a Petri net is controlled by the number and distribution of tokens
- Petri nets execute by firing transitions
- A transition fires by removing tokens from its input places and creating new tokens which are distributed at output places
- A transition may fire if it is enabled
  - If each of the transitions input places has at least as many tokens in it as arcs from the place of transition
  - Multiple tokens are needed for multiple input arcs
  - Tokens in the input place that enables a transition are enabling tokens
- If multiple transitions are enabled the choice is made randomly
  - Petri nets are non-deterministic

Execution Rules cont.

- A transition fires by removing all of its enabling tokens and depositing them into each of the output places one token for each arc
- Multiple tokens are produced for multiples arcs
- Imagine t2, I(t2)={p21,p23}, O(t2)={p23,p25,p25}
  - One token is removed from each of p21 and p23
  - One token is deposited at p23 and two at p25
  - Tokens are not conserved
- Firing transitions change the markings of the Petri net
- The number of tokens in a place is always >= 0
- Transition firings continue until none are enabled and then execution halts (quiesence)

April 2004
Concurrency and Control

- Two transitions which are both enabled and do not interact can occur independently
  - Inherent parallelism
  - Transitions are only synchronized if it is required by system being modeled
  - Facilitates modeling distributed control systems

- No inherent measure of time or flow
  - Philosophy – the only important aspect of time is the partial ordering of transitions
  - Transitions can take a variable amount of time
  - Petri net execution defines a sequence of discrete events

Petri Net State Spaces

- The state of a Petri net is defined by its marking
- The firing of a transition generates a state change
- The state space of a Petri net with n places is the set of all markings or \( N^n \)
- \( \delta \) is a change function so \( \delta(\mu, t) \) is undefined if \( t \) is not enabled by \( \mu \)
- Two sequences result from executing a Petri net
  - The sequence of markings
  - The sequence of transitions fired
  - Given the transition sequence the marking sequence can be derived
  - Unless the net is degenerate, given the marking sequence the transition sequence can be derived
  - Either sequence is an execution trace
- Reachability set: A marking is \( \mu'' \) reachable from \( \mu \) iff there is a set of transitions that enable it
Best use is when

- Many independent components
- Trying to model occurrence of events or activities
- Trying to model flow of information or goods/services

### Events and Conditions

**Machine Shop Example**

**Conditions**

- a. The machine shop is waiting
- b. An order has arrived and is waiting
- c. The machine shop is processing the order
- d. The order is complete

**Events**

1. Order arrives
2. Machine shop starts processing
3. Machine shop finishes order
4. Order is sent for delivery

<table>
<thead>
<tr>
<th>Event</th>
<th>Pre-conditions</th>
<th>Post-conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</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>None</td>
</tr>
</tbody>
</table>
Now – turn this into a Petri net

- Conditions are places in a Petri net
- Events are transitions
- Inputs of transitions are the pre-conditions
- Outputs of transitions are the post-conditions

Concurrency and Conflict
Job queue

- A job is put in the input queue
- A job is waiting
- A job is processed
- A job is waiting to be output
- A job is output
- A processor is idle

April 2004
**Dining Philosopher’s Problem**

- Classic synchronization problem posed in 1965 by Edsger Dijkstra.
- 5 philosophers are sit down to a Chinese dinner
- There is one chopstick on each side of the plate
- Philosopher needs two chopsticks
- Philosophers alternatively meditate and eat
- Find an algorithm to avoid starvation and deadlock
  - Deadlock occurs if each has one chopstick
  - Starvation occurs if a philosopher cannot get both chopsticks
- Equal number of philosophers and chopsticks

April 2004

**The Philosopher’s Dilemma**

- Each philosopher has two states
  - Meditating M
  - Eating E
- Each chop stick is a place
  - C

April 2004
Solution

- Enable philosophers to declare state as hungry, eating, meditating
- Philosopher cannot eat unless
  - First declares he is hungry
  - Both neighbors are not eating
- Status of the philosophers is kept using a shared data structure (e.g. an array).
- A philosopher may enter the eating state only if both of its neighbors are not in that state.
- To ensure this, the philosopher obtains a mutual exclusion (mutex) lock and then changes his state from thinking to hungry.
- After he changes his state, he tries to obtain both chopsticks and will not do so until both of his neighbors have left the eating state.
- At that point the philosopher changes his state to eating and releases the lock.
- The philosopher then eats.
- After the philosopher is done eating, he again obtains a mutex lock, changes his state to thinking and sees, one at a time, if any of its two neighbors is hungry.
- If at least one is hungry, that philosopher enters the eating state and the cycle continues.

April 2004

Typical flow chart

![Flow chart image]

April 2004
Petri nets can map on to flow charts

Petri net flow chart
The many flavors of Petri nets

- **Original**
  - only one I/O arc per place – 1962 by Petri
- **Standard (that described)** PT-net
- **Colored Petri Nets** - CP-net
- **High Level Petri nets** – HLP-net

Where do I get more????

http://www.daimi.au.dk/PetriNets/

April 2004
Colored Petri Nets

- Ordinary Petri net (PT-net) has **no types and no modules**:
  - All tokens are of the same type
  - The net is "flat"
- Colored Petri Nets (CP-nets)
  - Use **data types**
  - Use complex **data manipulation**:
    - Each token has attached a data value called the **token color**.
    - Token colors can be **investigated** and modified by the occurring transitions.
- CP-nets enable **hierarchical descriptions**:
  - Larger models can be created by combining a set of **submodels**.
  - Well-defined **interfaces** between submodels
  - Well-defined **semantics** of the combined model
  - **Submodels** can be reused.

---

Resource Allocation Example
from Kurt Jensen

- Resource allocation system
- Two kinds of **processes**:
  - Three cyclic q-processes (states A,B,C,D and E).
  - Two cyclic p-processes (states B,C,D and E).
- Three kinds of **resources**:
  - Represented by the places R, S and T.
- During a **cycle** a process reserves some resources and releases them again:
  - Tokens are removed from and added to the resource places R, S and T.
- A **cycle counter** is increased each time a process completes a full cycle.
**Tools**

- PNML – Petri net markup language
  - http://www.informatik.hu-berlin.de/top/pnml/about.html
- XML-based interchange format for Petri nets
- Originally intended to serve as a file format for the Java version of the Petri net kernal –
  - http://www.informatik.hu-berlin.de/top/pnk/index.html
- Petri net kernal
  - an infrastructure for bringing ideas for analyzing, for simulating, or for verifying Petri Nets into action
  - for integrating the idea into a tool
  - there is a python and a java implementation
  - http://www.informatik.hu-berlin.de/top/pnk/

**The Token Game**

- The interactive firing of transitions in subsequent markings is the “token game”
- http://pdv.cs.tu-berlin.de/~azi/click_pn/click0.html
How do you translate places and transitions to entities, relations and rules as used in multi-agent models?

Consider:

**Conditions**
- Hijackers are in city
- Hijackers are in city for critical flight
- Flight has been tested
- Critical flight is occurring
- Flight has been hijacked

**Events**
1. A preflies flight
2. B preflies flight
3. A arrives in city x
4. B arrives in city x
5. A flies critical flight
6. B flies critical flight
7. A and B hijack flight