Introduction to Petri Nets

2006/2007
History

• Petri nets were introduced by C.A. Petri in his Ph.D. Dissertation: "Kommunikation mit Automaten," Institut für Instrumentelle Mathematik, Bonn, 1962.

• They are particularly useful for modeling systems with concurrent and asynchronous processing.
Why concurrency is a problem?

- Concurrency and asynchronous processing is typical in real world.
- It can pose a problem when many entities (people, machines, processing threads) use (share) the same resource (or a limited number of resources).
- A trivial example is that of an elevator – the cabin is single resource that many people want to use. The problem is how to control the elevator to minimise waiting time.
Why concurrency is a problem?

- The elevator "scheduling" control is not crucial, at worst improper algorithm may result in long waiting times
- There are situations where handling concurrency is crucial for correct behaviour of the system
Why concurrency is a problem?

- Unwanted results of concurrency include:
  - Race conditions
  - Resource starvation
  - Deadlocks
Race conditions

- When race conditions occur, the result of the system behaviour may be unexpected and is dependent on the sequence of other events.
- Race conditions were first described in electronics (logic circuits), when parallelism is typical.
- An example of race conditions may be poorly implemented ATM.
ATM case

- Lets imagine the ATM operation is following
  - Authorise client
  - Get account balance
  - Get client request
  - Update account
  - Dispense cash
ATM case

• This can translate to following situation:
  – Time = $t_0$ Authorisation OK
  – Time = $t_0 + 2s$ Balance is 1000
  – Time = $t_0 + 10s$ Client requested 800, 800<1000
  – Time = $t_0 + 12s$ Account updated by -800, 200 left
  – Time = $t_0 + 14s$ Cash dispensed

• Everything worked fine
ATM case

• Now, let's imagine that there are two cards attached to the same account (wife and husband, corporate account etc.)

• Two persons at nearly the same time want to withdraw money. What may happen?
ATM case

Client 1

Time = $t_0$
Authorisation OK

Time = $t_0 + 1s$

Time = $t_0 + 2s$
Balance is 1000

Time = $t_0 + 3s$

Time = $t_0 + 5s$

Time = $t_0 + 7s$
Account updated by -800, 200 left

Time = $t_0 + 9s$

Time = $t_0 + 10s$
Client requested 800, 800<1000

Time = $t_0 + 12s$
Account updated by -800, -600 left

Time = $t_0 + 14s$
Cash dispensed

Client 2

Time = $t_0$
Authorisation OK

Time = $t_0 + 1s$

Balance is 1000

Balance is 1000

Client requested 800, 800<1000

Account updated by -800, 200 left

Cash dispensed
Race conditions

- Race conditions may be resolved by resource locking

<table>
<thead>
<tr>
<th>Time</th>
<th>Client 1</th>
<th>Client 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_0+2s)</td>
<td>Balance is 1000, lock account</td>
<td>account locked - cannot proceed</td>
</tr>
<tr>
<td>(t_0+3s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t_0+10s)</td>
<td>Client requested 800, 800 &lt; 1000</td>
<td></td>
</tr>
<tr>
<td>(t_0+12s)</td>
<td>Account updated by -800, 200 left, unlock account</td>
<td></td>
</tr>
<tr>
<td>(t_0+14s)</td>
<td>Cash dispensed</td>
<td></td>
</tr>
</tbody>
</table>
Resource starvation

- The process (person, system) is denied the resource it needs, because other process is not freeing them
Deadlock

- Deadlock occurs when a number (at least 2) processes are waiting for the other to finish (or release resources) and therefore none progresses

- This can be illustrated by dining philosophers problem (invented by Edsger Dijkstra)
Dining philosophers problem

- The philosopher is either thinking, or eating
- The philosophers do not talk (communicate)
- The philosopher can pick a fork to her/his right or left, one at a time
- The philosopher needs both forks to eat
Graphs

- Petri nets are graphs
- Graph is an ordered pair $G:=(V,E)$, where $V$ is a set of nodes (vertices), $E$ is a set of pairs of distinct vertices – edges (lines)
- If $E$ is a set of unordered vertices, the graph is undirected
- If $E$ is a set of ordered vertices, the graph is directed (digraph)
- Petri nets are digraphs
Graphs
Multigraph

- A multigraph is a graph that may contain more than one edge connecting the same vertices
Bipartite graph

- Petri nets are bipartite graphs.
- In bipartite graphs the set of vertices $V$ can be divided into two disjoint subsets $V_1$ and $V_2$ such that any edge always connects vertices from different subsets.
- The graph is sometimes denoted as $G:=(V_1+V_2', E)$.
Petri Nets as graphs

- In Petri nets nodes of the first subset of vertices are called places, nodes of the second – transitions
- The symbol of a place is a circle or an ellipse
- The symbol of transition is a solid bar or a rectangle
- The edges of the graph are called arcs
Sample simple Petri net
Tokens

- In order to describe dynamics of Petri nets (and being able to “execute” them) another concept is introduced – that of a token.
- The tokens are denoted by a solid dot and can be placed inside the place symbol.
- They indicate presence or absence of, for example, resource.
- Places can hold any number of tokens or only a limited number (capacitated places).
Transitions

- Tokens are used to describe enabling of transitions
- If an arc is drawn from a place to a transition, it indicates that a token in the place is required to enable the transition
- If many arcs are drawn (multigraph!), its number indicates the number of required tokens
- The transition is enabled iff for all arcs coming to the transition the condition of the required tokens are met
Input and output sets

- In order to clarify the description the concept of input and output sets is introduced

- The input set (preset) of a transition $t$, denoted $\bullet t$, is a set of all places for which there are arcs going from these places to the transition $t$

- The output set (postset) of a transition $t$, denoted $t\bullet$, is a set of all places for which there are arcs going from transition $t$ to these places

- Similar definitions apply to input and output sets of a place $p$, denoted by $\bullet p$ and $p\bullet$, respectively
Marking

- Marking of a Petri net is distribution of tokens in this net
- It is a mapping $P \rightarrow \{0, 1, 2, \ldots\}$ that describes the number of tokens present in each place
- The marking of the net at the beginning of an analysis is called initial marking
Firing

- If a transition is enabled, it can fire.
- When a transition fires, tokens are removed from all its input places (taking into account multiple arcs!)
- After that, tokens are inserted into all its output places – again taking into account multiple arcs.
- The number of the tokens removed and inserted may be different!
- Firing sequence is a sequence of transitions firing.
Firing
Firing
Usefulness of Petri nets

- Petri nets can be used to model complex processes
- Petri nets can be simulated (executed) in order to illustrate and test system behaviour, benchmark its speed etc.
- It is possible to perform a formal analysis of Petri net to find possible problems of the systems (for example deadlocks)
- For different applications the places and transitions may have different interpretations
## Interpretations of places and transitions

<table>
<thead>
<tr>
<th>Input places</th>
<th>Transitions</th>
<th>Output places</th>
</tr>
</thead>
<tbody>
<tr>
<td>required resources</td>
<td>task</td>
<td>freed resources</td>
</tr>
<tr>
<td>input data</td>
<td>computations</td>
<td>output data</td>
</tr>
<tr>
<td>input signals</td>
<td>signal processing</td>
<td>output signals</td>
</tr>
<tr>
<td>buffers/registers</td>
<td>processor</td>
<td>buffers/registers</td>
</tr>
</tbody>
</table>
Some sample simulations

In separate files...
Properties of Petri nets

• Petri nets properties are basis for their formal analysis

• The following properties are commonly used:
  – Reachability
  – Liveness
  – Boundedness
Reachability

- Reachability set $R(M_0)$ is a set of all possible markings reachable from the initial marking $M_0$.
- If a marking $M$ is in the reachability set $R(M_0)$, it means that there exists a firing sequence transforming $M_0$ into $M$.
- This is an useful property for analysis of systems. We can ask whether:
  - A desired state can be reached at all?
  - Is it possible for a system to arrive at an undesired (erroneous) state?
Reachability and reachability graph

- Reachability can be analysed by building a reachability graph.
- It is a directed graph where nodes represent markings and edges – transitions between two markings.
- The graph is constructed by finding all possible transitions from the initial marking – this gives a set of markings reachable from the initial one, then all possible transitions from the previously discovered markings, and so on.
- May lead to extremely large graphs.
Liveness

- A marking $M_x$ of network is live if, for any transition $t$ and for every reachable marking $M_y$ there exists a firing sequence from $M_y$ that includes $t$.

- In other words, every transition of the net can fire an infinite number of times.

- A Petri Net PN is structurally live, if any initial marking of PN is live.

- Liveness may be used to model the occurrences of deadlocks.
Boundedness

- A marking $M_x$ is bounded if there exists a positive integer $k$ such that for every reachable marking $M_y$ - element of the reachability set $R(M_x)$ - the number of tokens in each place is bounded by $k$.

- A Petri net is structurally (inherently) bounded if all of its initial markings are bounded.

- In other words, no reachable state can at any place contain more than $k$ tokens.

- If $k$ equals one, the marking is said to be safe.

- This property is useful for modelling limited (bounded) resources.
Limitations of Petri nets

• The type of Petri nets described up to this point is called Place/Transition nets

• They have a number of limitations:
  – Inability to model similar (but not identical) processes using one net
  – All tokens are identical
  – No way to represent additional properties – there is no way to associate any additional data with token
Sample net
More complex Petri nets

- In order to overcome these problems, a number of solutions extending the initial approach has been proposed:
  - Coloured Petri Nets
  - Hierarchy Petri Nets
  - Object Petri Nets
  - Prioritised Petri Nets
  - Timed Petri Nets
    - Stochastic Petri Nets
Coloured Petri Nets (CPNs)

- Allow to construct more compact models by sharing similar parts between various processes
- Handling additional data is possible
- This is mainly achieved by transition from identical tokens that represent binary value to typed tokens that can hold any kind of data
Coloured Petri Nets

- The data value attached to a token is called token **colour**
- The data value can be a simple number, a string, a structure consisting of some fields etc.
- The type of the value is called **colour set** – it is identical to type in programming languages
- Each place in CPN has its colour set (type)
Colour sets

- Before any coloured tokens are created, colour sets (types) must be defined.
- Types may be defined in an arbitrary way, but in order to use computer tools, a formal method is required.
- One of such methods is CPN ML language.
Marking and initial marking

- Initial marking in CPNs is determined by evaluation of initialization expressions that are attached to places (by convention an underlined expression next to a place).
- As the tokens may now carry complex information, their representation by presence or absence of dots is not useful.
- Instead, the number of tokens is denoted by a number in a circle next to the place, and the token values are listed below this circle.
Arc expressions

- As the tokens are more capable, transitions should have a way to use the information carried by the tokens
- This is possible thanks to arc expressions
- Arc expressions evaluate to a set of tokens that determine type and number of tokens “passed” through arc
- Arc expressions can contain a number of variables, operations on these variables (e.g. comparison), logical operators etc. Therefore they may evaluate to different values for different tokens
Arc expressions

• Arc expressions can be therefore treated as functions:
  – Tokens (their types and values) are arguments to the expressions
  – Tokens (their types and values) are return values from the expressions and determine token flow
Arc expressions and enabled transitions

- Arc expression going to transition nodes determine if the transition is enabled.

- In CPNs, thanks to arc expressions, enabling of a transition is conditioned not only on the presence of tokens in the input places, but also on their values.
Guards

- Guard is a boolean expression applied to a transition node
- By convention it is placed in square parentheses next to a transition
- Guard is an additional condition that has to be fulfilled in order to enable transition
Sample coloured net

Types (colour sets) definition

Initialisation expression

Tokens in place

Place colour set (type)

Arc expression

Color U = with p | q;
Color I = int;
Color P = product U | I
Color E = with e;
Var x : U;
Var i : I;

if x=q then 1'(q,i+1)
else empty

if x=p then 1'(p,i+1)
else empty
Resource Allocation Example

If \( x = q \) then \( 1^*(q, i+1) \), else empty

If \( x = p \) then \( 1^*(p, i+1) \), else empty

Case \( x \) of
- \( p \Rightarrow 2^*e \)
- \( q \Rightarrow 1^*e \)

if \( x = q \) then \( 1^*(q, 0) \)
else empty

if \( x = p \) then \( 2^*(p, 0)++ \)
else \( 1^*(q, 0) \)
Resource Allocation Example
if \( x = q \) then \( 1 \cdot (q, i+1) \) else empty

if \( x = p \) then \( 1 \cdot (p, i) \) else empty

Resource Allocation Example
Resource Allocation Example
Resource Allocation Example

if x=q then 1^(q,i+1) else empty

1^(q,0)++
1^(q,1)

if x=p then 1^(p,i+1) else empty

2^(p,0)

T1
(x,i) [x=q]

T2
(x,i)

T3
(x,i)

T4
(x,i)

T5
(x,i)

case x of
p => 2^e
| q => 1^e

if x=q then 1^e else empty

1^e

if x=p then 1^e else empty

1^e

3^e

1^e

2^e
Resource Allocation Example
Resource Allocation Example
Coloured version of dining philosophers
Coloured version of dining philosophers

- Declarations
  - val n = 5;
  - colset PH = index ph with 1..n;
  - colset CS = index cs with 1..n;
  - var p: PH;
  - fun Chopsticks(ph(i)) =
    1`cs(i) ++ 1`cs(if i=n then 1 else i+1);
Coloured version of dining philosophers
Coloured version of dining philosophers
Software for Petri nets – CPN Tools

- http://wiki.daimi.au.dk/cpntools/_home.wiki
- Now (May 2007) in version 2.2.0
- Created by CPN Group, University of Aarhus, Denmark
- Runs on Win2K, Win XP, Linux Fedora Core 2, probably other Linux distributions
- Requires hardware OpenGL acceleration, may not work with older graphics cards
CPN Tools interface

- CPN Tools uses a clever and intuitive UI, however, it is quite different from typical GUI – this may pose problems during first contact
- The screen is divided into “Index” (left side) and “Workspace” (the rest)
- Although the Index can be used for some operations, typical user interaction is different
CPN Tools interface

- Many functions are accessible by clicking and holding right mouse button – this brings up a circular context menu.
- Some items can be dragged from the Index into Workspace – they expand into toolboxes.
CPN Tools interface