Hierarchical Petri nets

prof.dr.ir. Wil van der Aalst
Extensions of the classical Petri net

- **Color** (i.e., tokens have values, places have color sets, arc have inscriptions, transitions have guards, etc.) was introduced in Chapter 5 and formalized in Chapter 6.
- **Time** (i.e., tokens have timestamps, color set may be timed, arcs have delays, etc.) was introduced in Chapter 5 and formalized in Chapter 6.
- **Hierarchy** is introduced and formalized in Chapter 7.
- A colored, timed, hierarchical Petri net is called a **high-level Petri net**.
- **Hierarchical CPNs** (=CPN + hierarchy) provide a concrete language for high-level Petri nets.
The need for hierarchy: Compare with construction drawings of a house

- The top-level: The house as a whole
The second level: The first floor of the house.
The third level: The kitchen
Top-down versus bottom-up
Recall

• Three good reasons for making a process model:
  • gain insight
    for a better understanding of the system
  • analysis
    validation and verification
  • specification
    a blue print for construction
• Like the construction drawing of an architect!
• However, despite the addition of color and time, a hierarchy concept is still missing thus far!
Basic idea

- Transitions correspond to subsystems/subprocesses
- Divide and conquer
- Reuse

Let us formalize this in CPN...
Hierarchical CPN

desk_subprocess
input_pat = wait
output_pat = done

Pat
wait

1`Klaas

HS

done

Pat
desk

Pat

wait

1`Ann

Emp

free

HS
desk_subprocess

start

(p,e)@+d(e)

stop

(p,e)

Emp

busy

EP

Pat

output_pat

Pat

output_pat

Pat

In

input_pat

p

Out

out
Superpage: main

desk

wait 1`Klaas

Pat

socket node

substitution transition

hierarchy inscriptions

desk_subprocess
input_pat = wait
output_pat = done

reference to subpage

port assignment

socket node
Subpage: desk_subprocess
One page can have multiple instances (i.e., reuse)
Semantics: Replace each substitution transition by a copy of the corresponding page.
Thermostat system: 
Add hierarchy

color Temp = string;
color B = unit;
color BT = B timed;
var t:Temp;
var a:B;
var b:BT;
var c,d:Temp;

---

Diagram:

- States and transitions representing the thermostat's behavior.
- Temperatures and time intervals are explicitly marked.
- Activities such as 'switch_off', 'switch_on', and state changes like 'heater', 'warm_up', 'cool_down', and 'updown' are depicted.
- Logical conditions: [t>c], [t<c]

---
Top-level page: *main*

```plaintext
color Temp = string;
color B = unit;
color BT = B timed;
var t: Temp;
var a: B;
var b: BT;
var c: Temp;
```

Diagram:
- `complete_system`
- `temp=temp`
- `HS`
- `socket`
- `1`15`
Subpage: *complete_system*

- Heating process
- Temp
- Day and night program
- Socket
- Port
- In/Out

Flow: heating_process -> [temp] -> socket -> port -> [day_and_night_program] -> [HS]

Symbols:
- HS
- temp
- down
- up
- socket
- port
- In/Out

Values:
- Temp = 18°C
- Temp = 14°C
- day_and_night_program

Notes:
- temp = temp
- down = down
- up = up
Subpage: heating_process
Subpage: day_and_night_program
Overview

```plaintext
color Temp = string;
color B = unit;
color BT = B timed;
var t: Temp;
var a:B;
var b:BT;
var c:Temp;
```

### Complete System

- **heating_process**
  - Temp = temp
  - up = up
  - down = down
- **day_and_night_program**
  - Temp = temp
  - up = up
  - down = down

### Switch to Day

- **heater**
  - B
  - b\+@15
  - warm_up
  - t\+1
  - Temp
  - b
  - a
  - [t< c]

### Switch to Night

- **switch_to_day**
  - B
  - b\+@8*60
  - switch_to_night
- **switch_to_night**
  - B
  - b\+@16*60
Hierarchy

Creating large, intricate nets can be a cumbersome task. But similar to modular programming, the construction of CP-nets can be broken into smaller pieces by utilizing the facilities within CPN Tools for creating substitution transitions. Conceptually, nets with substitution transitions are nets with multiple layers of detail - you can have a somewhat simplified net that gives a broad overview of the system you are modeling, and by substituting transitions of this top-level net with more detailed pages, you can bring more and more detail into the model.

Substitution transitions

In hierarchical nets there is a method by which a transition can represent an entire piece of net structure, so that the net containing the transition executes as if the logic that the transition represents were physically present at the location of the transition. Such a transition is a substitution transition.

Substitution transitions add nothing fundamentally new. Everything that can be done with them can also be done by using fusion places. But like fusion places, substitution transitions add so much convenience that they can make the difference between modeling feasibility and total impossibility.

Let us consider a substitution transition named reverse which stands for a net that is used to reverse a list of integers. A small blue tag, called a subpage tag, is associated with the substitution transition reverse.

Subpages and superpages

A page that contains a substitution transition is called a superpage. The page named top in the figure above is a superpage.

When a CP-net uses a substitution transition the logic that the transition represents must be kept somewhere. It is kept on a page called a subpage, and...
CPN Tools (flat model)
With hierarchy
Example: Production system
Kanban

- Kanbans are a means to achieve JIT (Just-In-Time).
- Japanese word for card.
- Attributed to Taiichi Ohno (the father of the Toyota Production System)
Two beer kanban
Data

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

*Processing times*

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

*Resources per work center*

<table>
<thead>
<tr>
<th></th>
<th>Replenishment lead times</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Kanbans in-between work centers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Time in-between subsequent orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
</tr>
</tbody>
</table>
How to configure each work center if they share the same page definition???
Top level page: main

```plaintext
color INT = int;
color Prod = string;
color PT = product Prod * INT;
color PTTimed = PT timed;
color PT = product Prod * INT;
var p:Prod;
var i:INT;
var i:INT;
```

# resources

```
resources X INT
resources Y INT
resources Z INT
```

processing times

```
supplier
io_lead_time PT
processing_time X
work_center X
product1 INT
product2 INT
product3 INT
product4 INT
```

port socket assignments

```
supplier
io_lead_time = io_lead_time
kanban_in = kanban1
product_out = product1
```

```
work_center
processing_time = processing_time X
resources = resources X
kanban_in = kanban1
product_in = product1
product_out = product2
```

```
work_center
processing_time = processing_time Y
resources = resources Y
kanban_in = kanban2
product_in = product2
product_out = product3
```

```
work_center
processing_time = processing_time Z
resources = resources Z
kanban_in = kanban3
product_in = product3
product_out = product4
```

```
customer
processing_time = processing_time Z
c kanban_out = kanban4
product_in = product4
```

```
customer
processing_time = processing_time Z
c kanban_out = kanban4
product_in = product4
```
Sub page: customer

In/Out

place_order

(p,t)

(p,t)@+t

c_ia_time

PTTimed

In

product_in

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time in-between subsequent orders</td>
<td>7</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>
Sub page: work_center

Resources per work center

Processing times

HCPN-30
Overview

One page definition, three page instances!!
CPN tools
Inventory strategies: from (s,Q)-b to (R,s,S)+b

MY COMPANY IS MOVING TO A "JUST IN TIME" INVENTORY STRATEGY. YOU'LL DELIVER WHEN WE NEED IT.

SO...YOUR SUCCESS DEPENDS ON MY COMPANY DOING WHAT IT PROMISES? YOU HAVE MY DEEPEST SYMPATHY.

I FEEL A SHARP, STABBING PAIN IN MY CHEST. AND SO IT BEGINS.
Application: Modeling logistic processes

**8 combinations**

- **Periodic or Instant**: $R,s$ or $s$
- **Fixed or Order-Up-To-Level**: $Q$ or $S$
- **Backorders or Not**: $-b$ or $+b$
External interface of a stock point

orders

- color Product = string;
- color Quantity = int;
- color PQ = product Product * Quantity;

goods

(For simplicity we do not add external configuration places.)
Inventory policy \((s,Q)-b\)

- Complete lost sales, i.e., no backorders.
- Continuous review, i.e., \(R=0\).
- Order point \(s\).
- Fixed order quantity \(Q\).
- \(s\) and \(Q\) determined for each product individually.

Order point is compared with inventory position, i.e., on-hand stock + ordered – back orders. (Note that back orders = 0)
Step 1: Port nodes

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
```

customer_order

```
<table>
<thead>
<tr>
<th>In</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQ</td>
</tr>
</tbody>
</table>
```

repl_order

```
<table>
<thead>
<tr>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQ</td>
</tr>
</tbody>
</table>
```

customer_delivery

```
<table>
<thead>
<tr>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQ</td>
</tr>
</tbody>
</table>
```

repl_delivery

```
<table>
<thead>
<tr>
<th>In</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQ</td>
</tr>
</tbody>
</table>
```

orders

goods
In example, we assume two products: "productA" and "productB". Of course the structure of the model stays the same for any set of products.
Step 3: Placing orders

```plaintext
| color Product = string; |
| color Quantity = int; |
| color PQ = product Product * Quantity; |
| var p: Product; |
| var a,b,c,d,e,f: Quantity |
| val sA = ("productA",0); |
| val sB = ("productB",0); |
| val oA = ("productA",0); |
| val oB = ("productB",0); |
| val oqA = ("productA",150); |
| val oqB = ("productB",100); |
| val opA = ("productA",50); |
| val opB = ("productA",60); |
```
order_quantity \( 1'oqA++1'oqB \)

order_point \( 1'opA++1'opB \)

stock \( 1'sA++1'sB \)

\[ (p,b-a) \]

\[ (p,b) \]

\[ (p,b+a) \]

\[ (p,b) \]

\[ (p,b) \]

repl_order

\[ (p,e) \]

\[ (p,f) \]

\[(b+d <= f)\]

repl_delivery

\[ (p,e) \]

\[ (p,d) \]

\[ (p,d) \]

ordered

\[ (p,d+e) \]

\[ (p,b) \]

\[ (p,b) \]

\[ (p,b) \]

\[ (p,b) \]

In

Out
Step 4: Dealing with deliveries
Inventory policy \((R, s, S) + b\)

- **Complete backordering**, i.e., no lost sales.
- **Periodic review**, i.e., \(R\) is review period.
- **Order point** \(s\).
- **Variable order quantity** \(S\), i.e., \(S\) is the order-up-to-level.

Every \(R\) time units, the inventory position is compared with \(s\). If it is below \(s\), the inventory position is raised to \(S\) by placing an order.

Recall: inventory position = on-hand stock + ordered – back orders.
Step 1: Port nodes (as before)

```java
| color Product = string; |
| color Quantity = int; |
| color PQ = product Product * Quantity; |
```

customer_order

![Customer order diagram]

repl_order

![Replacement order diagram]

customer_delivery

![Customer delivery diagram]

repl_delivery

![Replacement delivery diagram]
Step 2: Handle customer order (as before)

```java
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e: Quantity
val sA = ("productA",0); val sB = ("productB",0);
```
Step 3: Adding back orders

- 

```
color Product = string;
color Quantity = int;
color PQ = Product * Quantity;
var p: Product;
var a,b,c,d,e: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val bA = ("productA",0); val bB = ("productB",0);
```

Diagram:

- **t1**
  - (p,a) to PQ
  - (p,b) to PQ
  - (p,b-a) to PQ
  - (p,b) to PQ

- **t2**
  - (p,a) to PQ
  - (p,b) to PQ
  - (p,b-a) to PQ

- **t3**
  - (p,a) to PQ
  - (p,b) to PQ
  - (p,b) to PQ
  - (p,b-a) to PQ

- **Customer Order**
  - In
  - PQ

- **Customer Delivery**
  - Out
  - PQ

- **Back Order**
  - (p,a) to PQ
  - (p,c) to PQ
  - (p,b) to PQ
  - (p,b) to PQ
  - (p,c+a) to PQ
  - (p,b) to PQ

- **Back Ordered**
  - (p,a) to PQ
  - (p,c) to PQ
  - (p,b) to PQ
  - (p,b-a) to PQ
  - (p,c-a) to PQ

- **Stock**
  - (p,a) to PQ
  - (p,b) to PQ
  - (p,b) to PQ

- **Replenishment Order**
  - Out
  - PQ

- **Replenishment Delivery**
  - In
  - PQ

- **Back Orders**
  - Sum of backorders
Step 4: Ordering

- **customer_order**: \( (p,a) \)
- **customer_delivery**: \( (p,a) \)
- **back_order**: \( (p,a) \)
- **back_ordered**: \( (p,a) \)
- **stock**: \( (p,b-a) \)
- **order_up_to_level**: \( (p,b) \)
- **order_point**: \( (p,c) \)
- **check**: \( (p,c+a) \)
- **repl_order**: \( (p,b-a) \)
- **repl_delivery**: \( (p,c-a) \)

Product:
- **check**: \( (p,e) \)
- **order_point**: \( (p,f) \)
- **repl_order**: \( (p,d) \)
- **repl_delivery**: \( (p,e) \)

1. **Step 4: Ordering**

- **review**: \( 1\'rA++1\'rB \)
- **order_point**: \( 1\'opA++1\'opB \)
- **check**: \( 1\'ouA++1\'ouB \)
- **repl_order**: \( 1\'oA++1\'oB \)

- Conditions:
  - \([c=0] \text{ andalso } (b>=a)\]
  - \([b<a] \)
  - \([b>=a] \)

- Rules:
  - if \((b+d-c <= f)\) then 1\'(p,e-(b+d-c)) else empty
  - if \((b+d-c <= f)\) then (p,e+c-b) else (p,d)

- Additional Notes:
  - 1\'sA++1\'sB
  - 1\'bA++1\'bB
  - 1\'oA++1\'oB
  - 1\'rA++1\'rB
  - 1\'ouA++1\'ouB
  - 1\'opA++1\'opB

- Diagram Elements:
  - In
  - Out
  - Review
  - Check
  - Repl_Order
  - Repl_Delivery
  - Order Point
  - Order Up To Level
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
color Period = int;
color PP = product Product * Period timed;
var p: Product;
var t: Period;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val oA = ("productA",0); val oB = ("productB",0);
val bA = ("productA",0); val bB = ("productB",0);
val ouA = ("productA",150); val ouB = ("productB",100);
val opA = ("productA",50); val opB = ("productA",60);
val rA = ("productA",3);
val rB = ("productB",4);
$[(c=0) \text{ andalso } (b>=a)]$
Step 6: Dealing with deliveries

- **customer_order**: \( (p,a) \)
- **back_order**: \( (p,a) \)
- **customer_delivery**: \( (p,a) \)
- **stock**: \( (p,b-a) \)
- **back_ordered**: \( (p,c) \)
- **order_point**: \( (p,d) \)
- **review**: \( 1^rA+1^rB \)
- **repl_order**: \( 1^oA+1^oB \)
- **repl_delivery**: \( (p,a) \)

- **order_up_to_level**: \( 1^ouA+1^ouB \)
- **check**: \( p \)

- **if (b+d-c <= f)**
  - then \( 1^oA-(b+d-c) \)
  - else empty

- **if (b+d-c <= f)**
  - then \( (p,e+c-b) \)
  - else \( (p,d) \)

- **Step 6: Dealing with deliveries**

In

Out

In

Out

...
Six remaining inventory policies

- $(s, S) - b$
- $(R, s, S) - b$
- $(s, Q) + b$
- $(R, S) - b$
- $(s, Q) - b$
- $(s, S) + b$
- $(R, s, S) + b$
- $(R, S) + b$
Question: Which inventory policy?
color Product = string;
color Quantity = int;
| color PQ = product Product * Quantity;
| color Period = int;
| color PP = product Product * Period timed;
| var p: Product;
| var t: Period;
| var a,b,c,d,e: Quantity
| val sA = ("productA",0); val sB = ("productB",0);
| val oA = ("productA",0); val oB = ("productB",0);
| val ouA = ("productA",150); val ouB = ("productB",100);
| val rA = ("productA",3);
| val rB = ("productB",4);

In
Out
Out
In

Question: Which inventory policy?

(R,S)-b
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
color Period = int;
color PP = product Product * Period timed;
var p: Product;
var t: Period;
var a,b,c,d,e: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val oA = ("productA",0); val oB = ("productB",0);
val bA = ("productA",0); val bB = ("productB",0);
val ouA = ("productA",150); val ouB = ("productB",100);
val rA = ("productA",3);
val rB = ("productB",4);

Question: Which inventory policy?
Question: Which inventory policy?

```
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
color Period = int;
color PP = product Product * Period timed;
var p: Product;
var t: Period;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0); val sB = ("productB",0);
val oA = ("productA",0); val oB = ("productB",0);
val ouA = ("productA",150); val ouB = ("productB",100);
val opA = ("productA",50); val opB = ("productA",60);
val rA = ("productA",3);
val rB = ("productB",4);
```

(R,s,S)-b
Question: Which inventory policy?
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val ouA = ("productA",150);
val ouB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);

Question: Which inventory policy?

In
Out
In
Out
(s,S)-b
Modeling a supply chain

(For simplicity we do not add external configuration places. Direction and circular structure may be confusing.)
Exercise: Modify to allow for partial shipments

Part of the order may be lost.
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productA",60);

customer_order
PQ
(p,a) (p,a)
[b>=a]  

stock
PQ
(p,b) (p,b)
(p,b-a) (p,b)

if b>0
then 1\((p,b)\)
else empty
PQ
(p,a) (p,a)
(b<a)

order_quantity
1\(\text{oqA}++1\text{oqB}\)
(p,e) (p,e)

ordered
PQ
(p,b) (p,b)
(p,b+a) (p,b)

[b+d <= f]

1\(\text{oA}++1\text{oB}\)
(p,d) (p,d)
(p,d+e) (p,d+e)

repl_order
PQ
(p,d) (p,d)
(p,e) (p,e)

repl_delivery
PQ
(p,d) (p,d)
(p,a) (p,a)

In
Out
In
Out

In
Out

In
Out

In
Out

In
Out

In
Out

In
Out
Exercise: Modify to allow for partial shipments
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
val sA = ("productA",0);
val sB = ("productB",0);
val oA = ("productA",0);
val oB = ("productB",0);
val bA = ("productA",0);
val bB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);

In

Out

In

Out

if b>0 then 1*(p,b) else empty

[(c=0) andalso (b>=a)]

[(b+d-c <= f)]

Solution
val oB = ("productB",0);
val bA = ("productA",0);
val bB = ("productB",0);
val oqA = ("productA",150);
val oqB = ("productB",100);
val opA = ("productA",50);
val opB = ("productB",60);

How to model that backorders can also be partially delivered?
color Product = string;
color Quantity = int;
color PQ = product Product * Quantity;
var p: Product;
var a,b,c,d,e,f: Quantity
fun min(x:int,y:int) = if x<y the x else y;

Solution
var a,b,c,d,e,f: Quantity
fun min(x:int,y:int) = if x<y then x else y;

[(c=0) andalso (b>=a)]

if b>0 then 1 `(p,b) else empty

if a>b then 1 `(p,a-b) else empty
After studying Chapter 6 one should be able to:

- Flatten a hierarchical CPN model.
- Design a hierarchical CPN model for scratch.
- Modify a hierarchical CPN model.
Exercise: Five Chinese philosophers

• Make a hierarchical CPN model of five Chinese philosophers alternating between states thinking and eating. To eat two chopsticks are needed. In total there are five chopsticks. The philosophers are sitting in a circle, and need to complete for chopsticks with their direct neighbors (left and right). Assume that both chopsticks need to be taken at the same time. Model this using a hierarchical CPN model. Make sure to model the behavior of a philosopher only once and just use the color set BlackToken of type unit.

• Change the model such that philosophers can take one chopstick at a time but avoid deadlocks and a fixed ordering of philosophers.

• Flatten the hierarchical CPN model.
color BlackToken = unit;
var b:BackToken

philosopher
left = CS5
right = CS4

philosopher
left = CS4
right = CS3

philosopher
left = CS3
right = CS2

philosopher
left = CS2
right = CS1

philosopher
left = CS1
right = CS5
Page philosopher

Diagram:
- Think
- Eat
- Take chopsticks
- Put down chopsticks
- Left
- Right
- BlackToken
- In/Out

Transitions:
- think -> ()
- eat -> BlackToken
- take_chopsticks -> BlackToken
- put_down_chopsticks -> BlackToken
- left -> BlackToken
- right -> BlackToken
Flat model is obtained by replacing substitution transitions by subpages

```plaintext
color BlackToken = unit;
var b:BackToken

philosopher
left = CS5
right = CS4
HS
philosopher
left = CS4
right = CS3
HS
philosopher
left = CS3
right = CS2
HS
philosopher
left = CS2
right = CS1
HS
philosopher
left = CS1
right = CS5
HS

take_chopsticks

RS

think

RS

eat

RS

put_down_chopsticks

left

RS

right

Repeat 5 times...
```
In CPN Tools