Hierarchical CPN model-based diagnosis using HAZOP knowledge

E. Németh, R. Lakner, K. M. Hangos, I. T. Cameron

Research Report SCL-009/2003

Contents

List of Figures

Hierarchical CPN model-based diagnosis using HAZOP knowledge

Erzsébet Németh
Department of Computer Science
University of Veszprém
H-8201 Veszprém, P.O. Box 158, Hungary
E-mail: nemethe@scl.sztaki.hu

Rozália Lakner
Department of Computer Science
University of Veszprém
H-8201 Veszprém, P.O. Box 158, Hungary
E-mail: lakner@almos.vein.hu

Katalin M. Hangos
Systems and Control Research Laboratory
Computer and Automation Institute HAS
H-1518 Budapest, P.O. Box 63, Hungary
E-mail: hangos@scl.sztaki.hu

Ian T. Cameron
Department of Chemical Engineering
The University of Queensland
Australia
E-mail: itc@cheque.uq.edu.au

Abstract

The multi-scale modelling of process systems using hierarchical coloured Petri nets for diagnostics is investigated in this report. A case study is used to systematically show the modelling approach using an illustrative example of a tank reactor with cooling jacket. This form of representation shows how CPNs can provide a hierarchically structured framework for advanced fault diagnosis.

1 Introduction

Designing and modelling complex systems in any discipline is a difficult task - such systems are often highly organized and structured, they consist of a large variety of subsystems interacting in a plethora of diverse communication patterns decomposition is a way of stretching and describing such systems.

Since automatic reasoning usually not need all the quantified details of a mathematical model and because sometimes only knowledge about qualitative behavior is available, several qualitative representations have been suggested to replace mathematical models in physics. Common approaches are to use representations based on e.g., logics, constraints or directed graphs. A model of physical structure and component behaviour is used to generate a description of the target system in logic formulae. Together with the simulation results or measurements we compute the possible malfunctions or errors from a given state of the system.

M. Lind is the creator of Multilevel Flow Modelling (MFM) [1] which is a new modelling methodology for representation of goals and functions of complex process plants. The idea of his methodology is to apply functional concepts to represent a plant on several interrelated levels of abstraction. MFM demands that physical components, goals and functions be viewed as separate entities. The assumptions that the functions are separate from components is similar to the "no function in structure" assumption of qualitative physics. MFM also assumes that the goals are not given by separate functions. Instead they must be stated during the model construction.

Several projects [2] have used means-end and functional models for fault diagnosis, which are not pure MFM, but closely related. This means that some representation of goals, functions or both is used. Usually a tree or graph describing a hierarchy of goals or functions.

Process systems are purposeful entities which attempt to fulfil goals set by the designer. Those goals evolve with the design and can be decomposed into a hierarchy of subgoals, leading finally to the top goal of the system. We refer to this structure as a "goal-tree" (GT). They have found extensive use in the reliability engineering area of systems theory [3].

As well as goal hierarchies in process design, we can also establish hierarchies based on functional and behavioural decompositions. Here the hierarchy of function represents the fundamental actions that components or groups of system components perform. Functions address objectives related to each of the system goals. These functional issues are intimately linked to the goal hierarchy. Clearly the functional hierarchy is strongly related to the structural or component hierarchy of the system or model of that system. This is because overall function relates to components and their interconnection. Hence these complementary and related hierarchies provide alternative views of the system and deeper insight into the intent of the design and the specific design developed to address system goals. The combination of the functional and structural hierarchy leads to the concept of the success-tree (ST). This is the description in functional or structural terms by design goals are fulfilled [4].

Other hierarchical representations can also be used depending on the overall purpose of the analysis [5]. However, in the case of fault diagnosis of process systems, goal, function and structural hierarchies provide adequate insight for the analysis. How we effectively represent these hierarchies and then examine and use them is a crucial issue. One approach is the use of Petri nets.

Coloured Petri nets (CPNs) [6, 7] belong to the area of discrete event system methodology. CPN is well known for its capability in modelling discrete event systems in terms of cause–consequence relationships possibly extended by timing information related to the underlying dynamic system variations. CPNs are capable of describing the dynamic behaviour of process systems and handle the hierarchy.

The basic idea behind using hierarchical CPNs is to allow the modeller to construct a large model by using a number of small CPNs which are related to each other in a well-defined way.

This report is organized as follows. First the representation tool of multi-scale process models, coloured Petri net with hierarchical extension is described with its properties. Thereafter the multi-scale models of process systems with modelling and goal hierarchies are discussed for diagnosis. The described representation is shown in a tank reactor with cooling jacket. Finally conclusions are drawn.

2 Coloured Petri nets (CPNs)

In this section, we summarize the basic notations found in the literature about coloured Petri nets and hierarchical coloured Petri nets. For first, the general coloured case will be considered, while the second part concerns with the hierarchical Petri nets.

2.1 Coloured Petri nets

Coloured Petri nets (CPNs) [8] belong to the area of discrete event system methodology. A CPN is well known for its capability in modelling discrete event systems.

The structure of a Petri net is a bipartite directed graph describing the structure of a discrete event system, while the dynamics of the system is described by the execution of the Petri net. A Petri net is coloured if the tokens are distinguishable. According to the formal definition of CPNs [6] a coloured Petri net model is a nine-tuple

$$CPN = (\Sigma, P, T, A, N, C, G, E, IN)$$

satisfying the following requirements:

(i) Σ is a finite set of non-empty types, called *colour sets*

(ii) P is a finite set of *places*

(iii) T is a finite set of *transitions*

(iv) A is a finite set of *arcs* such that $P \cap T = P \cap A = T \cap A = \emptyset$

(v) $N : A \rightarrow P \times T \cup T \times P$ is a *node function*

(vi) $C : P \rightarrow \Sigma$ is a *colour function*

(vii) G is a *guard function*. It is defined from T into expressions such that
$\forall t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \Sigma]$

(viii) E is an *arc function*. It is defined from A into expressions such that
$\forall a \in A : [Type(E(a)) = C(p(s))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$
where $p(a)$ is the place of $N(a)$ and C_{MS} denotes the set of all multi-sets over C

(ix) IN is an *initialization function*. It is defined from P into expressions such that
$\forall p \in P : [Type(IN(p)) = C(p(s))_{MS} \wedge Var(IN(p)) = \emptyset]$

where:

$Type(expr)$ denotes the type of an expression,
$Var(expr)$ denotes the set of variables in an expression,
$C(p)_{MS}$ denotes a multi-set over $C(p)$.

A *binding* of a transition t is a function b defined on $Var(t)$, such that:

(i) $\forall v \in Var(t) : b(v) \in Type(v)$,

(ii) $G(t)\langle b \rangle$ (denotes the evaluation of the guard expression $G(t)$ in the binding b.

A *token element* is a pair (p, c) where $p \in P$ and $c \in C(p)$. A *binding element* is a pair (t, b) where $t \in T$ and $b \in B(t)$. By $B(t)$ denotes the set of all bindings for t. The set of all token elements is denoted by TE while the set of all binding elements is denoted by BE.

A *marking* is a multi-set over TE while a *step* is a non-empty and finite multi-set over BE. The *initial marking* M_0 is the marking which is obtained by evaluating the initialization expressions.

A *transition is enabled* if each of its input places contain the multi-set specified by the input arc inscription (possibly in conjunction with the guard), and the guard evaluates to true. When a transition is enabled it may *occur*, and this means that the tokens are removed from the input places and added to the output places of the occurring transitions. The number and colour of the tokens are determined by the arc expressions, evaluated for the occurring bindings.

A *finite occurrence sequence* is a sequence of markings and steps: $M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots M_n[Y_n\rangle M_{n+1}$, such that $n \in \mathbb{N}$. A marking M'' is *reachable* from a marking M' if and only if exists a finite occurrence sequence having M' as start marking and M'' as end marking, i.e., if and only if for some $n \in \mathbb{N}$ there exists a sequence of steps $Y_1Y_2 \ldots Y_n$ such that: $M'[Y_1Y_2 \ldots Y_n\rangle M''$. We then also say that M'' is reachable from M' in n step. The set of markings which are reachable from M' is denoted by $[M'\rangle$.

2.1.1 Reachability/Occurrence graph

The reachability problem addresses the question whether it is possible to reach or avoid a given marking starting from a given initial state. The coverability problem is the generalization of the reachability problem. It answers the question whether there is a marking $M'' \in [M_0\rangle$ such that $M'' \geq M'$, i.e. M'' covers a predefined marking M'. A marking M'' covers a marking M' if $\forall i : M''_i \geq M'_i$, i.e. each components of marking M'' is greater than or equal to the components of marking M'.

An important set of CPN analysis methods uses the occurrence graphs (OGs) of the net [7], which is also called state spaces or reachability graphs. The basic idea behind OGs is to construct a directed graph which has a node for each reachable marking and an arc for each possible state change. Obviously, such a graph may become very large, even for small CPNs. However, it can be constructed and analyzed totally automatically, and there exist techniques which make it possible to work with condensed occurrence graphs without losing analytic power.

The *full occurrence graph* of a CPN is the directed weighted graph $OG = (V, N, A)$ where:

(i) $V = [M_0\rangle$, i.e. the vertex set is the set of all reachable markings from M_0.

(ii) $A = \{(M_1, M_2) \in V \times V \mid \exists (b, M_2) \in BE \times V \ : \ M_1[b\rangle M_2\}$.

(iii) $N(a) = b \ : \ \forall a = (M_1, M_2) \in A \quad \exists (b, M_2) \in BE \times V \ : \ M_1[b\rangle M_2$.

Construction of occurrence graphs The following *algorithm constructs the OG*. The algorithm halts if and only if the OG is finite. Otherwise the algorithm is never terminated.

```
Procedure Consctruct_OG (M_0 :  marking)
      Waiting := ∅
      Node(M_0)
      repeat
        select a node M_1 ∈ Waiting
        for all (b, M_2) ∈ Next(M_1) do
          begin
            Node(M_2)
            Arc(M_1, b, M_2)
          end
        Waiting := Waiting - {M_1}
      until Waiting = ∅
```

Waiting is a set of nodes. It contains those nodes which we have not yet found the successors. *Node(M)* is a procedure that creates a new node M and adds M to *Waiting*, if node M is not in *Waiting*. *Arc*(M_1, b, M_2) is a procedure that creates a new arc (M_1, b, M_2) with source M_1 and destination M_2 if it is not exist. *Next*(M_1) denotes the set of all possible "next moves".

Analysis of occurrence graphs Most of the analysis of dynamic properties is based on a brute-force search where every node or every arc of the OG is visited either to gather some information or to search the corresponding or predefined node(s).

Partial occurrence graphs When an OG is infinite or too big be constructed, it may still be of interest to construct a *partial occurrence graph* (POG), i.e., a subgraph of the OG. This can be conveniently done because the algorithm for constructing OGs supports a set of *stop criteria* and a set of *branching criteria*. Stop criteria allow to specify how large the constructed graph should be. It is also possible to specify a predicate function that checks each node as soon as it has been processed. The branching criteria imply that we don't develop all the successor markings of a given node. It is also possible to specify a predicate function which check each node before any successors are calculated.

When a partial graph has been constructed by using a set of stop criteria or a set of branching criteria, it is later possible to continue the construction (with or without criteria). The construction is always breadth first.

2.2 Hierarchical coloured Petri nets

2.2.1 Introduction to hierarchical coloured Petri nets

CPNs [6, 7] are capable of describing the dynamic behaviour of process systems and handle the hierarchy. The basic idea behind using hierarchical CPNs is to allow the modeller to construct a large model by using a number of small CPNs which are related to each other in a well-defined way. At one level, we want to give a simple description of the modelled activity without having to consider internal details about how it is carried out. Moreover, we want to be able to integrate the detailed specification with more crude descriptions and this integration must be done in such a way that it is meaningful to speak about the behaviour of the *complete* net.

The idea of *substitution transition* is to allow the user to relate a transition (and its surrounding arcs) to a more complex CPN, called *subnet*, which usually gives a more precise and detailed description of the activity represented by the substitution transition. Each subnet has a number of places called *port places* and they constitute the interface with which the subnet communicates with its surroundings. Substitution transition has some input places and some output places called *input* and *output socket places*, respectively. To specify the relationship between a substitution transition and its subnet, we must describe how the port places of the subnet are related to the socket places of the substitution transition. This is done by providing a *port assignment*. When a port place is assigned to a socket place, the two places become identical. Through the input ports the subnet receives tokens from the surroundings. Analogously, the subnet delivers tokens to the surroundings through the output ports.

2.2.2 Formal definition of hierarchical coloured Petri nets

A hierarchical CPN consists of a set of subnets. Each subnet $s \in S$ is a non-hierarchical CPN, i.e., a tuple:

$$(\Sigma_s, P_s, T_s, A_s, N_s, C_s, G_s, E_s, IN_s) \ .$$

We talk about the elements of the entire hierarchical CPN, we use the notation:

$$\Sigma = \bigcup_{s \in S} \Sigma_s \quad P = \bigcup_{s \in S} P_s \quad T = \bigcup_{s \in S} T_s \quad A = \bigcup_{s \in S} A_s$$

where it should be noted that the sets of colour sets usually have common elements, while the sets of net elements are required to be disjoint.

A hierarchical coloured Petri-net is a tuple

$$HCPN = (S, SN, SA, PN, PT, PA, FS, FT, PP)$$

satisfying the following requirements:

(i) S is a finite set of *pages (subnets)* such that:
- Each page $s \in S$ is a non-hierarchical CPN: $(\Sigma_s, P_s, T_s, A_s, N_s, C_s, G_s, E_s, IN_s)$.
- The sets of net elements are pairwise disjoint: $\forall s_1, s_2 \in S : [s_1 \neq s_2 \Rightarrow (P_{s_1} \cup T_{s_1} \cup A_{s_1}) \cap (P_{s_2} \cup T_{s_2} \cup A_{s_2}) = \emptyset]$.

(ii) $SN \subseteq T$ is a set of *substitution nodes.*

(iii) SA is a *page assignment* function. It is defined from SN into S such that:
- No page is a subpage of itself: $\{s_0 s_1 \ldots s_n \in S^* | n \in \mathbb{N}_+ \wedge s_0 = s_n \wedge \forall k \in 1 \ldots n : s_k \in SA(SN_{s_{k-1}})\} = \emptyset$.

(iv) $PN \subseteq P$ is a set of *port nodes.*

(v) PT is a *port type* function. It is defined from PN into $\{in, out, i/o, general\}$.

(vi) PA is a *port assignment* function. It is defined from SN into binary relation such that:
- Socket nodes are related to port nodes: $\forall t \in SN : PA(t) \subseteq X(t) \times PN_{SA(t)}$.
- Socket nodes are of the correct type: $\forall t \in SN : \forall (p_1, p_2) \in PA(t) : [PT(p_2) \neq \text{general} \Rightarrow ST(p_1, t) = PT(p_2)]$.
- Related nodes have identical colour sets and equivalent initialization expressions: $\forall t \in SN : \forall (p_1, p_2) \in PA(t) : [C(p_1) = C(p_2) \wedge I(p_1) <>= I(p_2) <>]$.

(vii) $FS \subseteq P_s$ is a finite set of *fusion sets* such that:
- Members of fusion set have identical colour sets and equivalent initialization expressions: $\forall fs \in FS : \forall p_1, p_2 \in fs : [C(p_1) = C(p_2) \wedge I(p_1) <>= I(p_2) <>]$.

(viii) FT is a *fusion type* function. It is defined from fusion sets into $\{global, page, instance\}$ such that:
- Page and instance fusion sets belong to a single page: $\forall fs \in FS : [FT(fs) \neq global \Rightarrow \exists s \in S : fs \subseteq P_s]$.

(ix) $PP \in S_{MS}$ is a multi-set of *prime pages*.

A page $s \in S$ may have many different page instances. The set of *page instances* of a page $s \in S$ is the set SI_s of all triples $(s^*, n^*, t_1 t_2 \ldots t_m)$ that satisfy the following requirements:

(i) $s^* \in PP \wedge n^* \in 1 \ldots PP(s^*)$.

(ii) $t_1 t_2 \ldots t_m$ is a sequence of substitution nodes, with $m \in \mathbb{N}$, such that:
- $m = 0 \Rightarrow s^* = s$
- $m > 0 \Rightarrow (t_1 \in SN_{s^*} \wedge [k \in 2 \ldots m \Rightarrow t_k \in SN_{SA(t_{k-1})}] \wedge SA(t_m) = s)$.

Page instances where the third component is the empty sequence are said to be *prime* page instances, while all others are *secondary* page instances.

The set of *place instances* of a page $s \in S$ is the set PI_s of all pair (p, id) that satisfy the following requirements:

(i) $p \in P_s$.

(ii) $id \in SI_s$.

The set of *transition instances* of a page $s \in S$ is the set TI_s of all pair (t, id) that satisfy the following requirements:

(i) $t \in T_s \setminus SN_s$.

(ii) $id \in SI_s$.

The set of *arc instances* of a page $s \in S$ is the set AI_s of all pair (a, id) that satisfy the following requirements:

(i) $a \in A_s \setminus A(SN_s)$.

(ii) $id \in SI_s$.

Each place instance, transition instance and arc instance is said to *belong* to the page instance in its second component.

We define token elements, binding elements, markings, steps, initial markings, reachability and occurrence sequences analogously to the corresponding concepts for non-hierarchical CPNs.

2.3 Example for coloured Petri net

This example describes a simple protocol where a sequence of packets is sent from one site to another via a network where packets may be delayed or lost. The CPN model of the protocol system is shown in Fig. 1. It consists of three parts. The *Sender* part has two transitions which can *Send packets* and *Receive acknowledgements*. The *Network* part has two transitions: *Transmit packets* and *Transmit acknowledgements*. The *Receiver* part has a single transition which can *Receive packets* (and send acknowledgements). The interface between the *Sender* and the *Network* consists of places A and D, while the interface between the *Network* and the *Receiver* consists of places B and C.

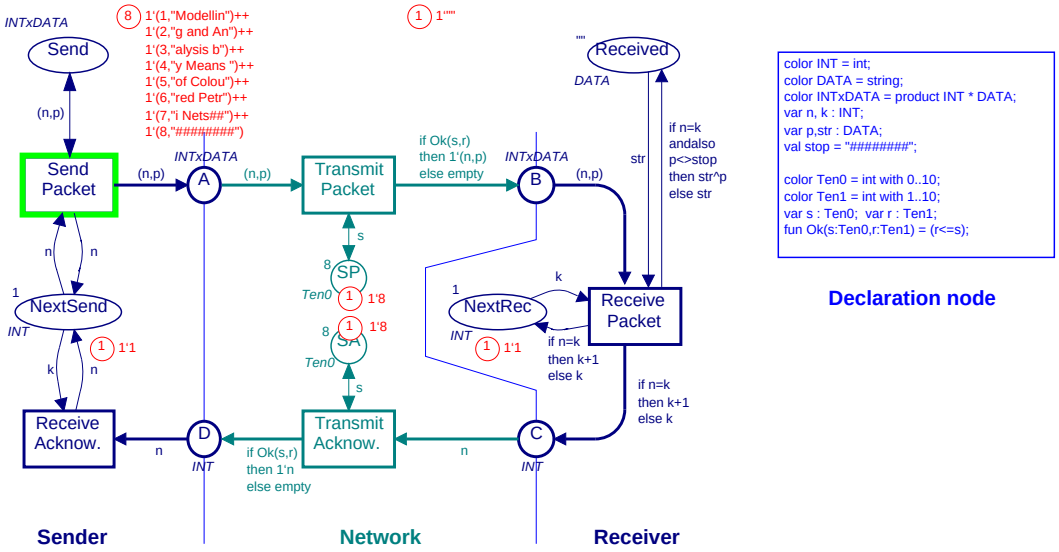


Figure 1: CPN model of the Simple Protocol

The packets to be sent are positioned at the place *Send*. Each token on this place contains a packet number and the data contents of the packet (represented as a text string). The place *Next send* contains the number of the next packet to be sent. Initially this number is 1, and it is updated each time an acknowledgement is received.

The content of the received message is kept at the place *Received*. This place contains a single token with a text string which is the concatenation of the text strings contained in the received packets (ignoring the contents of duplicates and packets received out of order). Initially the text string at *Received* is empty, i.e., "". At the end of the transmission we expect *Received* to contain the text string "Modelling and Analysis by Means of Coloured Petri Nets". The place *Next Rec* contains the number of the next packet to be received. Initially this number is 1, and it is updated each time a packet is successfully received.

We do not model how the *Sender* splits a message into a sequence of packets or how the *Receiver* reassembles the packets into a message. Neither do we model how the tokens at *Send* and *Received* are removed at the end of the transmission or how the packet numbers in *Next Send* and *Next Rec* are reset to 1. Now let us take a closer look at the five different transitions in the protocol system:

- *Send Packet* sends a packet to the *Network* by creating a copy of the packet on place A. The number in *Next Send* specifies which packet to send. It should be noted that the packet is not removed from *Send*. Neither is the counter at *Next Send* increased. The reason is that the packet may be lost and hence need to be retransmitted. Our protocol is pessimistic, in the sense that

it continues to repeat the same packet - until it gets an acknowledgment telling that the packet has been successfully received.

- *Transmit Packet* transmits a packet from the *Sender* site of the *Network* to the *Receiver* site by moving the corresponding token from A to B. The boolean expression $Ok(s, r)$ determines whether the packet is successfully transmitted or lost. The variable r will be bound to an arbitrary value in its colour set (i.e., to any integer between 1 and 10). Design/CPN makes a fair choice between the 10 values (provided that fair simulation is chosen in General Simulation Options). The Ok function returns true if the value of r is less than or equal to the value of s. This means that the probability of successful transmission is determined by the token at place SP. We have given SP a token with value 8. Hence we have 80% chance for successful transmission. However, it is easy to modify the success rate, simply by changing the token value at SP.

- *Receive Packet* receives a packet and checks whether the packet number n is identical to the number k in *Next Rec*. When the two numbers match, the number in *Next Rec* is increased by 1 and the text string in the packet is concatenated to the text string in *Received* - unless it is *stop* = "########", which by convention indicates end-of-message. Otherwise, the packet is ignored and the number in *Next Rec* 4 is left unchanged. In both cases an acknowledgment is sent containing the number of the next packet which the *Sender* should send.

- *Transmit Acknowledgement* transmits an acknowledgment from the *Receiver* site of the *Network* to the *Sender* site by moving the corresponding token from C to D. The transition works in a similar way as *Transmit Packet*. This means that the acknowledgment may be lost, with a probability determined by the token at place SA.

- *Receive Acknowledgment* receives an acknowledgment and updates the number in *Next Send* by replacing the old value with the one contained in the acknowledgment. After a number of steps the CPN may have reached the intermediate marking shown in Fig X.

The CPN from Fig. 1 represented as a many-tuple:

(i) $\Sigma = \{INT, DATA, INT \times DATA, Ten0, Ten1\}$

(ii) $P = \{Send, Next\ Send, Received, Next\ Rec, A, B, C, D, SP, SA\}$

(iii) $T = \{Send\ Packet, Receive\ Acknow., Transmit\ Packet, TransmitAcknow., ReceivePacket\}$

(iv) $A = \{Send\_TO\_Send\ Packet, Next\ Send\_TO\_Send\ Packet,$
$Send\ Packet\_TO\_Send, Send\ Packet\_TO\_A, Send\ Packet\_TO\_Next\ Send,$
$Next\ Send\_TO\_Receive\ Acknow., D\_TO\_Receive\ Acknow.,$
$Receive\ Acknow.\_TO\_Next\ Send,$
$A\_TO\_Transmit\ Packet, SP\_TO\_Transmit\ Packet,$
$Transmit\ Packet\_TO\_SP, Transmit\ Packet\_TO\_B,$
$C\_TO\_Transmit\ Acknow., SA\_TO\_Transmit\ Acknow.,$
$Transmit\ Acknow.\_TO\_SA, Transmit\ Acknow.\_TO\_D,$
$Received\_TO\_Received\ Packet, B\_TO\_Received\ Packet, Next\ Rec\_TO\_Received\ Packet,$
$Received\ Packet\_TO\_Next\ Rec, Received\ Packet\_TO\_C, Received\ Packet\_TO\_Received\}$

(v) $N(a) = (SOURCE, DEST)\ \ if\ a\ is\ in\ the\ form\ SOURCE\_TO\_DEST$

(vi) $C(p) = \begin{cases} INT & if\ p \in \{Next\ Send, D, C, Next\ Rec\} \\ DATA & if\ p = Received \\ INT \times DATA & if\ p \in \{Send, A, B\} \\ Ten0 & if\ p \in \{SA, SP\}. \end{cases}$

(vii) $G(t) = true\ if\ t \in T$

$$\text{(viii)}\quad E(a) = \begin{cases} (n,p) & if\ a \in \{Send\_TO\_Send\ Packet, Send\ Packet\_TO\_Send, \\ & \quad Send\ Packet\_TO\_A, A\_TO\_Transmit\ Packet, B\_TO\_Received\ Packet\} \\ n & if\ a \in \{Next\ Send\_TO\_Send\ Packet, Send\ Packet\_TO\_Next\ Send, \\ & \quad Receive\ Acknow.\_TO\_Next\ Send, D\_TO\_Receive\ Acknow., \\ & \quad C\_TO\_Transmit\ Acknow.\} \\ k & if\ a \in \{Next\ Send\_TO\_Receive\ Acknow., Next\ Rec\_TO\_Received\ Packet\} \\ s & if\ a \in \{SP\_TO\_Transmit\ Packet, Transmit\ Packet\_TO\_SP, \\ & \quad SA\_TO\_Transmit\ Acknow., Transmit\ Acknow.\_TO\_SA\} \\ str & if\ a = Received\_TO\_Received\ Packet \\ if\ Ok(s,r) & \\ \quad then\ 1`n & \\ \quad else\ empty & if\ a = Transmit\ Acknow.\_TO\_D \\ if\ Ok(s,r) & \\ \quad then\ 1`(n,p) & \\ \quad else\ empty & if\ a = Transmit\ Packet\_TO\_B \\ if\ n = k & \\ \quad then\ k+1 & \\ \quad else\ k & if\ a \in \{Received\ Packet\_TO\_Next\ Rec, Received\ Packet\_TO\_C\} \\ if\ n = k & \\ \quad andalso\ p <> stop & \\ \quad then\ str\hat{}p & \\ \quad else\ str & if\ a = Received\ Packet\_TO\_Received. \end{cases}$$

$$\text{(ix)}\quad I(p) = \begin{cases} 1`(1,"Modellin")++1`(2,"g\ and\ An")++ & \\ \quad 1`(3,"alysis\ b")++1`(4,"y\ Means")++ & \\ \quad 1`(5,"of\ Colou")++1`(6,"red\ Petr")++ & \\ \quad 1`(7,"i\ Nets\#\#")++1`(8,"\#\#\#\#\#\#\#\#") & if\ p = Send \\ 1`1 & if\ p = Next\ Send \\ 1`8 & if\ p = SA \\ 1`8 & if\ p = SP \\ 1`1 & if\ p = Next\ Rec \\ 1`"" & if\ p = Received \\ \emptyset & if\ otherwise. \end{cases}$$

The example for partial occurrence graph can be seen in Fig. 2.

2.4 Example for hierarchical coloured Petri net

This example shows how the CPN from "Simple Protocol" can be turned into hierarchical CPN (HCPN) with separate subnets (pages) for the *Sender*, the *Network* and the *Receiver* part. The protocol is modified to accommodate multiple *Receivers*.

The hierarchical page is shown in Fig. 3.a, the top level of the Simple Protocol is shown in Fig. 3.b, and the subnets (subpages) are shown in Fig. 4. Below shoes how the hierarchical CPN in Fig. 3-4 represented as a many-tuple. To save space (and time) we don't give the tuple-definition of the individual pages.

(i) $S = \{Main\#1, Sender\#2, Network\#3, Receiver\#4\}$

(ii) $SN = \{Sender@Main\#1, Network@Main\#1, Rec\_1@Main\#1, Rec\_2@Main\#1\}$

$$\text{(iii)}\quad SA(t) = \begin{cases} Sender\#2 & if\ t = Sender@Main\#1 \\ Network\#3 & if\ t = Network@Main\#1 \\ Receiver\#2 & if\ t = Rec\_1@Main\#1 \\ Receiver\#2 & if\ t = Rec\_2@Main\#1. \end{cases}$$

(iv) $PN = \{A@Sender\#2, D@Sender\#2, A@Network\#3, D@Network\#3, B1@Network\#3, B2@Network\#3, C1@Network\#3, C2@Network\#3, B@Receiver\#4, C@Receiver\#4, Received@Receiver\#4\}$

$$\text{(v)}\quad PT(p) = \begin{cases} in & if\ p \in \{D@Sender\#2, A@Network\#3, C1@Network\#3, C2@Network\#3, B@Receiver\#4\} \\ out & if\ p\{A@Sender\#2, D@Network\#3, B1@Network\#3, B2@Network\#3, C@Receiver\#4\} \\ i/o & if\ p = Received@Receiver\#4. \end{cases}$$

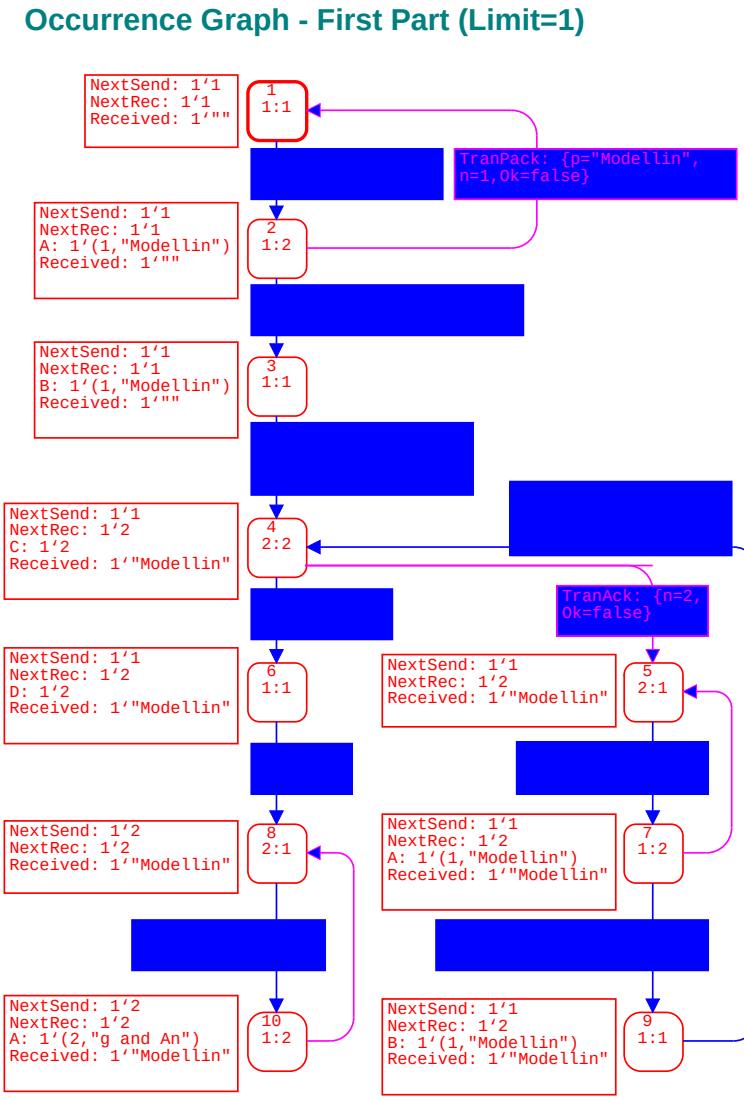
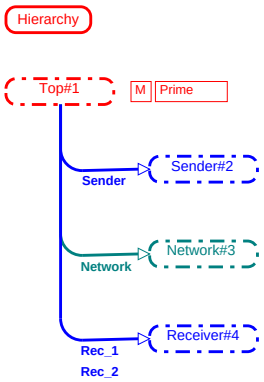
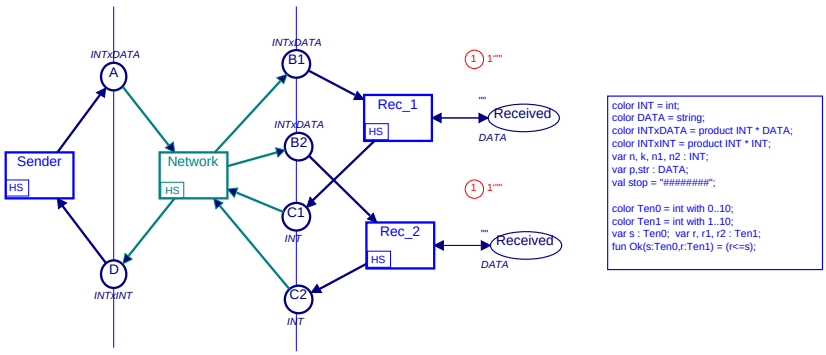


Figure 2: Partial occurrence graph of the of the Simple Protocol

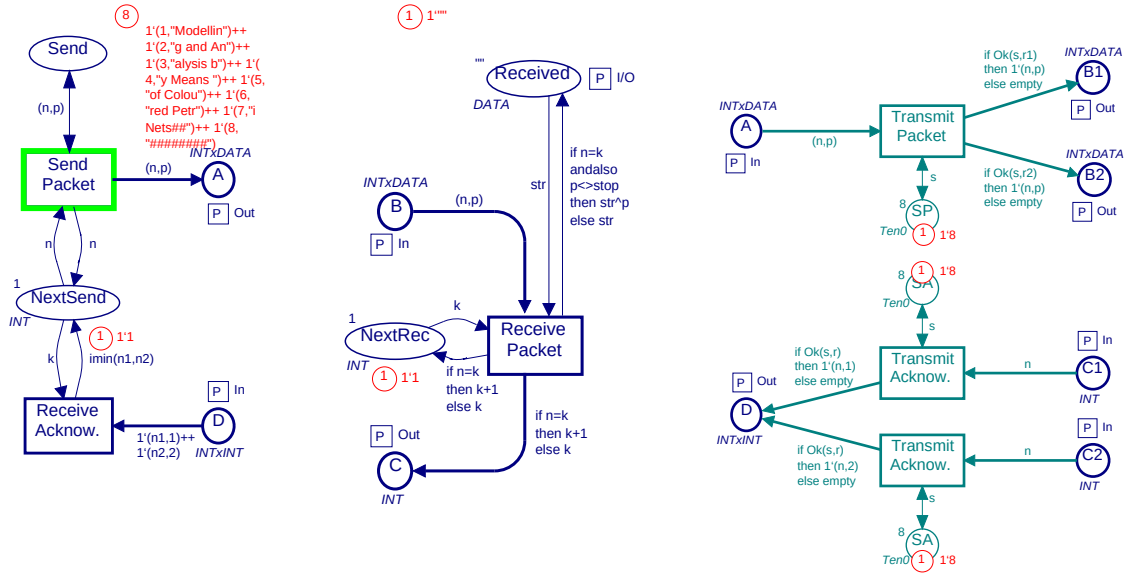


3.a: The hierarchical connections of subnets



3.b: The top level of Simple Protocol

Figure 3: The hierarchy of the Simple Protocol



4.a: The *Sender* subnet 4.b: The *Receiver* subnet 4.c: The *Network* subnet

Figure 4: The subnets of the Simple Protocol

(vi) $$PA(t) = \begin{cases} \{(D@Main\#1, D@Sender\#2), (A@Main\#1, A@Sender\#2)\} & if\ t = Sender@Main\#1 \\ \{(A@Main\#1, A@Network\#3), (C1@Main\#1, C1@Network\#3), \\ \quad (C2@Main\#1, C2@Network\#3), (D@Main\#1, D@Network\#3), \\ \quad (B1@Main\#1, C1@Network\#3), (B2@Main\#1, C2@Network\#3)\} & if\ t = Network@Main\#1 \\ \quad \ldots\ldots\ldots \end{cases}$$

(vii) $FS = \emptyset$

(viii) $FT(fs) = global \quad for\ all\ fs \in FS$

(ix) $PP = 1`Main\#1.$

3 Multi-scale coloured Petri net models of process systems

In this section, we summarize the basic notations in the literature about the process models. For first, general descriptions of process models will be considered, the second part the modelling hierarchy is discussed, while the third part concerns with the multi-scale coloured Petri net models of process models.

3.1 Process models

Dynamic process models originated from first engineering principles are most often too large for process control applications. Therefore they have to be decomposed somehow. One of the possible way of decomposition is to construct a model hierarchy.

The mathematical model of a process system usually is a set of equations which describe the static and/or dynamic behaviours of the system in the space and/or in time. A mathematical model building is an inherently recursive and iterative process which depends on the problem specification and modelling goals.

3.1.1 The structure of process models

The mathematical models of process systems are based on the *conservation balances*. In order to write proper balances, the system must be divided into parts which are described by elementary balances, such parts are called *balance volumes*. The role of balance volumes for mass, energy and momentum is crucial in process modelling. A balance volume is a basic element in a process modelling as it determines the region in which the conserved quantity is contained. How are these regions chosen? There are many possibilities. In general, parts of a process system which are typical candidates for balance volume definition include those regions which:

- contain only one phase or pseudo-phase,
- can be assumed to be perfectly mixed, or
- can be assumed to have a uniform (homogeneous) flow pattern.

The differential equations originate from conservation balances, therefore they can be termed *balance equations*. The application of conservation principles to typical gas-liquid-solid systems normally leads to relationships involving total system mass, component masses, total energy and system momentums. The conservation calls for algebraic equations to make the formulated balance equation complete.

Constitutive relations are usually algebraic equations of mixed origin. The underlying thermodynamic, kinetic, control and design knowledge is formulated and added to the conservation balances as *constitutive equations*. This is because when we write conservation balances for mass, energy and momentum, there will be terms in the equations which will require definition or calculation.

So the mathematical model of a process system is a set of differential and algebraic equations that is divided into four main parts:

- balance equations,
- constitutive (algebraic) equations,
- complementary (algebraic) equations,
- variables.

Since among different variables, equations and members of equations of the model have well-defined relations, the DAE is a structured knowledge which is built from the above elements.

3.2 Modelling hierarchy

Since the usage of process models is an important task and most often complex models are needed and used, it is necessary to develop an efficient unified hierarchical model description. A process system usually has a set of different models for the same purpose that describe the system in different detail. These models are not independent of each other but are related through the process system they describe and through their related modelling goals [9, 10]. The *hierarchy of process models* with different granularity is driven by the level of details based on the fundamental *modelling assumptions*. These assumptions affect the balance volumes, the system boundaries and the basic control mechanisms of the model.

The number of hierarchy levels depends on the complexity of the process system and the modelling goal, and the sequence of models in this hierarchy follows the top-down process engineering design from the process functionality and the flowsheet level to the finest detailed design. It is important that in every lower hierarchy level the union of all the balance volumes is the balance volume of

the top level model and every balance volume can be described by a set of balance equations and constitutive equations. Driven by the role in the process model the variables appearing in these equations can be arranged to a natural hierarchy from the conserved extensive quantities to the parameters.

The levels of a process model hierarchy can be as follows:

- system/plant
- units
- phases
- balance volumes
- balance equations
- constitutive (algebraic) equations
- complementary (algebraic) equations
- variables

On the highest levels the structure of a complex system or plant is represented as a flowsheet (diagram describing the connection between system elements) consisting of separated units or equipment forming the next layer. If a unit contains some phases then we describe all phases as a new level. The lowest level consists of model components described by balance equations, constitutive equations and complementary equations in the form of a differential algebraic equations (DAE) set, where each equation corresponds to a sub-flowsheet (i.e. one element of a (sub)flowsheet is detailed in a sub-flowsheet as flowsheet).

An illustrative example can be seen is Fig. 5. The highest level represents the system with input and output. In the next level (as sub-flowsheet) represents the more detailed system description, in the example the system is divided into two units which are connected. Unit_1 consists of two phases (a gas and a liquid) described by the 3rd layer. In the next layer the liquid phase is detailed by equations.

The elements of a *hierarchical differential algebraic equation (HDAE) process model* are then as follows:

- units which contains some phases on the same hierarchical level, but all of them correspond to a sub-flowsheet.
- balance equations connected to a unit or a phase that is on the same hierarchical level, but all of them correspond to a sub-flowsheet.
- variables used in a balance equation and described by a constitutive algebraic equation are detailed in a sub-flowsheet, i.e. the sub-flowsheet models the corresponding constitutive algebraic equation.
- variables used in an complementary algebraic equation and described by an algebraic equation are also detailed in a sub-flowsheet, i.e. the sub-flowsheet models the corresponding algebraic equation.

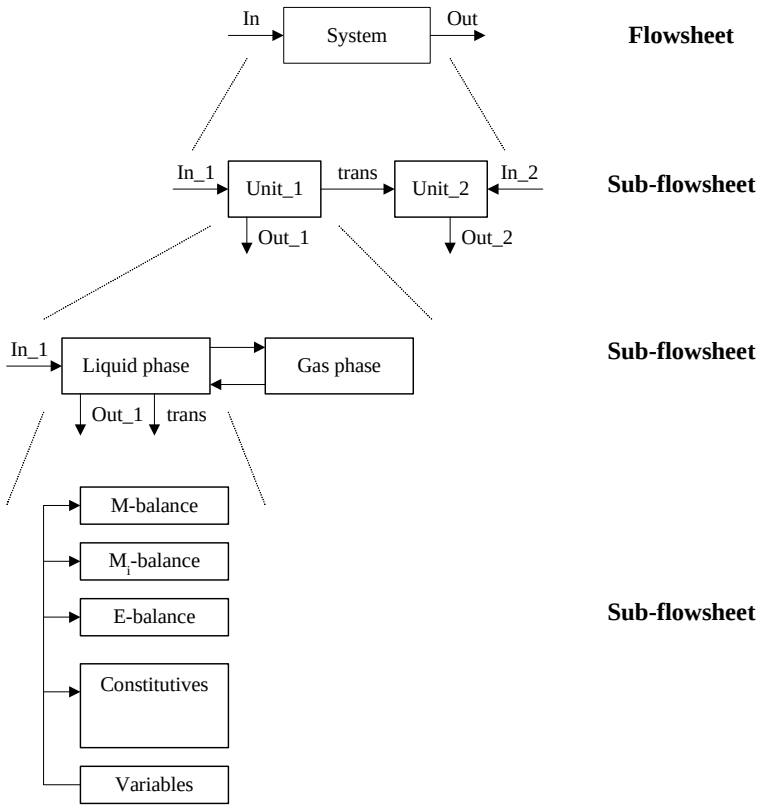


Figure 5: An example for flowsheet and sub-flowsheet

3.3 The conversion method from hierarchical process models to hierarchical CPN

A hierarchical DAE model defined in 3.2 as a process model can be described by a hierarchical CPN. Every sub-flowsheet corresponds to CPN page, i.e. a subnet substituting the corresponding transition. The variables are represented by nodes in the CPN model, and . A transition without any substitution denotes an elementary algebraic function (for example addition or product) or function, and serves as a model extension point. This way *both the cause-consequence and the dynamic relationships are described by the hierarchical CPN model* in a unified framework.

The significance of developing a hierarchically structured CPN model of a process system is explained by the fact that it precursor to developing a structured means of their fault detection and diagnosis.

4 Goal hierarchy

A *goal* describes a subjective motive of a system or its parts. Goals may also be viewed as some internal environmental conditions or criteria that should be attained by the system, so as to allow the system to reach or to maintain balance with its outer environment. The representation of goals is that in order to attain a goal, one needs a collection of functions to be realized. This is done by utilizing the corresponding system parts that achieve these functions. Therefore the goals and functions are closely interlocked.

The goals of a system are the objects to which effort is directed by means of design and operation. Systems are purposeful entities, which carry out specific activities to achieve desired outcomes. It is

possible to develop a hierarchy of goals, starting from the 'top' or overarching goal and then decompose those goal statements down to finer granularity. Thus we develop a *goal-tree* showed in Fig. 6, which can contain numerous subgoals, which combine in specific ways so that the top goal is achieved. It should be noted that the goal-tree structure and the fulfillment of the goal-tree is not unique as there are numerous ways or designs that can meet the top goal. Any process synthesis problem clearly illustrates this point.

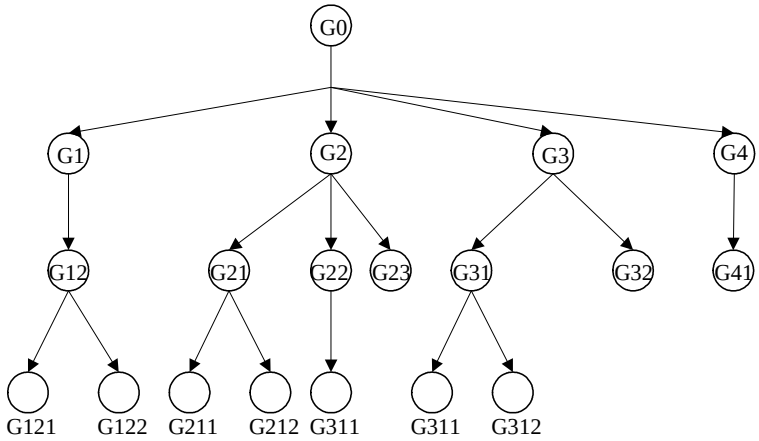


Figure 6: An example for goal tree

4.1 Goal-tree construction

The goal hierarchy can be expressed in natural language, using such expressions as: "Make product B from material A safely and efficiently". Each concept or sub-phrase in this statement can then be analyzed and broken down into finer granularity in order to develop the goal-tree for the application. Hence there are syntactical and semantical aspects to goal-tree development linked to ontologies [11] and data structures [12] that are beyond the scope of this current work.

It should be noted that accompanying the natural language goal-tree specification, clear objectives given in processing or technical terms must be available, so that it is clear when the goals are fulfilled. Hence, each goal statement must be accompanied by an objective or set of objectives to which the design or model relates. In the case study used in this paper, "safely" could be related to objective risk criteria for the plant, personnel and public. Other objectives will relate to specific plant throughput, product quality, acceptable operating temperature ranges and the like. The represent constraints on the design and operation.

Hence the goal hierarchy establishes the desired plant and operating objectives and in turn provides the driving concepts for the design and operation.

It is important to note, however, that there is neither cause–consequence nor dynamic (timing) relationships between goals and sub-goals, therefore this hierarchy cannot be naturally described by CPNs.

4.2 Connection between modelling and goal hierarchies: the functions

There is a natural link between goal hierarchies and the modelling hierarchy of the system which represents the reality under study. The modelling hierarchy can be seen as a series of decompositions into

finer levels of granularity: from the plant-wide model down to the equation and variable representation.

Goal nodes ($G_1, G_2, \ldots$) in the goal tree are addressed by specific functionality in the system components or their combinations. These physico-chemical components can be ultimately expressed in terms of equations and variables at the lowest modelling level. Hence there is a mapping of the model hierarchy through a function/behaviour hierarchy to the goal hierarchy.

Functions represent the roles of a system should have intended in the achievement of the goals. A node of a goal tree represents just an achieve goal, but the fulfilling way isn't described in here. The symptom connected to a goal node contains a function/behaviour like the knowledge, as condition of a goal (i.e. how the connected goal is reached). If the condition in a symptom is fulfilled, the connected goal is reached, otherwise some deviance is occurring. That is a connecting point for fault detection and diagnosis.

The hierarchies between model elements and between subgoals give a natural hierarchy between the symptoms.

The function/behaviour hierarchy is based upon the concept of a "function" which is a functional of the elements in the model hierarchy in an abstract sense. A function serves one or more goals in the goal hierarchy and thus it is a "connector" between the model elements it depends upon and the goal(s) it serves. Furthermore, functions may have "symptoms", i.e. deviations from the normal behavior associated to them, therefore they can be used as abstract "indicators" for fault detection and diagnosis.

So, the modelling hierarchy at the equation-variable level addresses lower subgoals and the composite modelling structures as we proceed up the tree are also logically linked to the upper goals in the goal-tree.

Furthermore, the fault detection and diagnosis, that can be performed using coloured Petri nets, can be connected to the hierarchical model CPN through connecting "function" nodes.

In the following the connection is illustrated in Fig. 7. A symptom is denoted by on the dashed line with hexagon. In the example there is a connection between the *Unit_ 2* inflow with *trans* and the goal *G31* (Get *trans* into *Unit_ 2*) and another connection between the *Unit_ 2* feeds with (*In_ 2*) and the goal *G32* (Get *In_ 2* into *Unit_ 2*) . The function of the element is "Feed *Unit_ 2* with *trans*". The *Symptom_ 1* is represented the goal *G31* achievement, in out case "The feed *trans* is NORMAL". The *Symptom_ 1* contains a condition "feed *trans* is NORMAL" If the condition is TRUE the goal *G31* is fulfilled, otherwise the value of variable *sign* outcoming from *Symptom_ 1* isn't equal "normal" so the guard of the transition connects to *Symptom_ 1* is satisfied, thus the transition fires and causes some tokens in place *Warning* depending on the value of variable *sign* "none", "low" or "high". Similarly working in the case of *Symptom_ 2*.

Since the construction of goal-tree the fulfillment of all subgoal is gives the truth of the implementing of the parent goal. In case one of subgoal isn't true, realized by the connection warning place isn't empty, the parent goal of the subgoal also false. This construction gives the natural connection between the warning places, but these places gives a similar tree as goal-tree, but the relation between children warnings is "OR". In our case, the place *Warning* of the goal's *G3* symptom is the parent of the places *Warning* of the *Symptom_ 1* and *Symptom_ 2*.

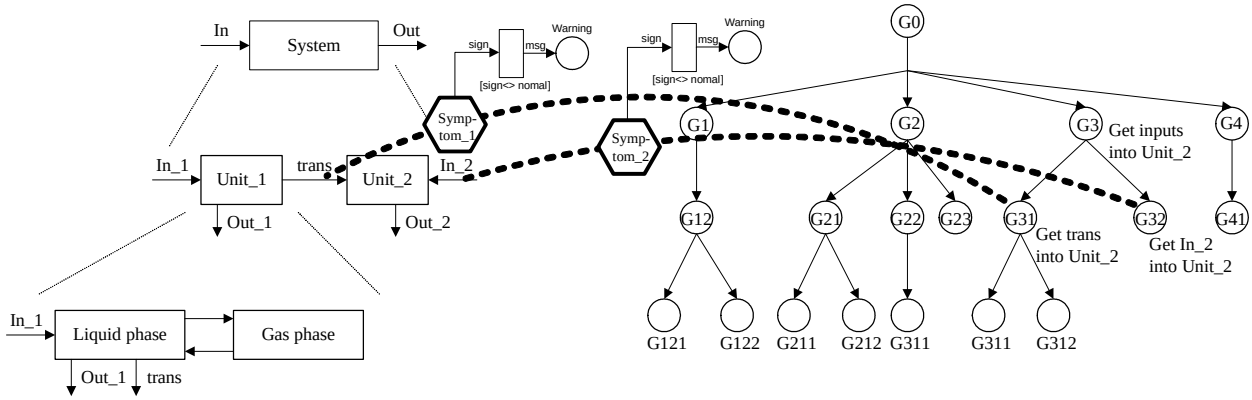


Figure 7: An example for the connection between the modelling and goal hierarchies with symptoms

5 Case study: A tank reactor with cooling jacket

In the following example simple multi-scale process models will be used to demonstrate the integrated development of the modelling and goal hierarchies for fault prediction-based fault-detection and diagnosis purposes.

5.1 The description of the tank reactor with cooling jacket

The examined system shows in Fig. 8 is a continuous, stirred tank reactor with cooling jacket. The reactor has constant volume and a single first order exothermic $A \rightarrow B$ reaction is taking place. The liquid inlet is fed into the reactor with volume rate v, concentration c_{a0} and temperature T_0, and the out flow has volume rate v, concentrations c_a, c_b and temperature T. The reactor is cooled by a liquid in a jacket with inlet volume rate v_c, inlet temperature T_{c0} and outlet volume rate v_c, outlet temperature T_c.

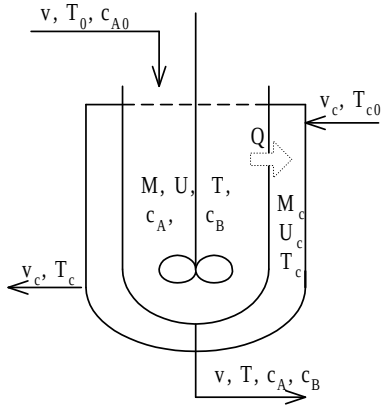


Figure 8: The diagram of the reactor

5.1.1 Dynamic model equations

The dynamic model equations of the reactor are derived from the conservation balance equations for the overall mass and energy of the system equipped by suitable algebraic constitutive equations. In order to have a relatively simple dynamic model for hierarchical description and diagnostic purposes, simplifications are needed.

Modelling assumptions

(i) The liquid in the tank reactor and in the cooling jacket is perfectly stirred.

(ii) There is only cooling liquid (water) in the jacket.

(iii) Balances are only set up for liquid phase in the tank reactor (the gas phase is neglected).

(iv) Physico-chemical properties are constant.

(v) The liquid phase in the tank has a constant volume V.

(vi) There are 3 components in the tank: A, B and inert (water).

Mathematical model Two principal balance volumes were identified for the system: the liquid phase of the tank and of the cooling jacket. The model equations can be seen in below.
Tank balances:

- Mass balance:

$$M = const$$

- Component balances:

$$\begin{aligned} \frac{dm_A}{dt} &= v * c_{A0} - v * c_A - V * r \\ \frac{dm_B}{dt} &= -v * c_B + V * r \end{aligned}$$

- Energy balance:

$$\frac{dU}{dt} = v * \rho * c_p * T_0 - v * \rho * c_p * T - Q + V * r * (-\Delta H)$$

Cooling jacket balances:

- Mass balance:

$$M_c = const$$

- Energy balance

$$\frac{dU_c}{dt} = v_c * \rho_c * c_{p_c} * T_{c0} - v_c * \rho_c * c_{p_c} * T_c + Q$$

The constitutive algebraic equations:

$$\begin{aligned} Q &= K * A * (T - T_c) \\ r &= k * c_A \end{aligned}$$

The complementary algebraic equations:

$$\begin{aligned} k &= k_0 * e^{-\frac{E}{R*T}} \\ m_A &= V * c_A \\ m_B &= V * c_B \\ U &= M * c_p * T \\ U_c &= M_c * c_{p_c} * T_c \end{aligned}$$

where the variables are:

M	–	overall mass in the reactor $[kg]$	M_c	–	overall mass in the jacket $[kg]$
U	–	total internal energy in the reactor $[kJ]$	U_c	–	total internal energy in the jacket $[kJ]$
V	–	volume of reactor $[m^3]$	V_c	–	volume of cooling jacket $[m^3]$
T	–	temperature of mixture $[K]$	T_c	–	temperature of coolant $[K]$
v	–	feed volume rate $[m^3/s]$	v_c	–	coolant volume rate $[m^3/s]$
ρ	–	density of mixture $[kg/m^3]$	ρ_c	–	density of coolant $[kg/m^3]$
c_p	–	heat capacity of mixture $[kJ/kgK]$	c_{p_c}	–	heat capacity of coolant $[kJ/kgK]$
m_A	–	mass of component A $[kg]$	c_A	–	concentration of component A $[kg/m^3]$
m_B	–	mass of component B $[kg]$	c_B	–	concentration of component B $[kg/m^3]$
m_w	–	mass of catalyst $[kg]$	c_{A_0}	–	initial feed concentration $[kg/m^3]$
T_0	–	initial feed temperature $[K]$	T_{c_0}	–	initial coolant feed temperature $[K]$
A	–	heat transfer area $[m^2]$	K	–	heat transfer coefficient (constant) $[kJ/m^2Ks]$
r	–	reaction rate $[kg/m^3s]$	k	–	reaction rate coefficient $[1/s]$
k_0	–	reaction rate constant $[1/s]$	R	–	gas constant $[kJ/kmolK]$
E	–	activation energy (constant) $[kJ/kmol]$	ΔH	–	heat of reaction (constant) $[kJ/kmol]$

5.1.2 Modelling hierarchy

Figure 9 shows a particular system decomposition for a tank reactor with cooling jacket. The hierarchical CPN model of the above system described by hierarchical CPN is showed in Fig. 10.

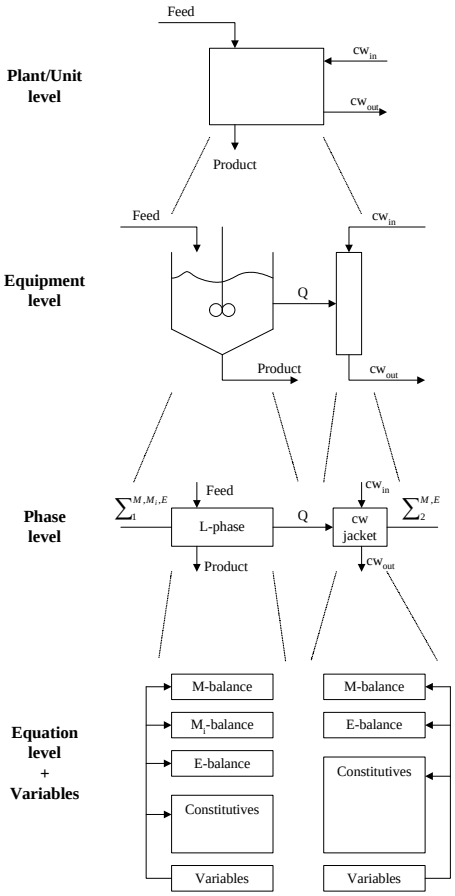


Figure 9: The modelling hierarchy of the reactor

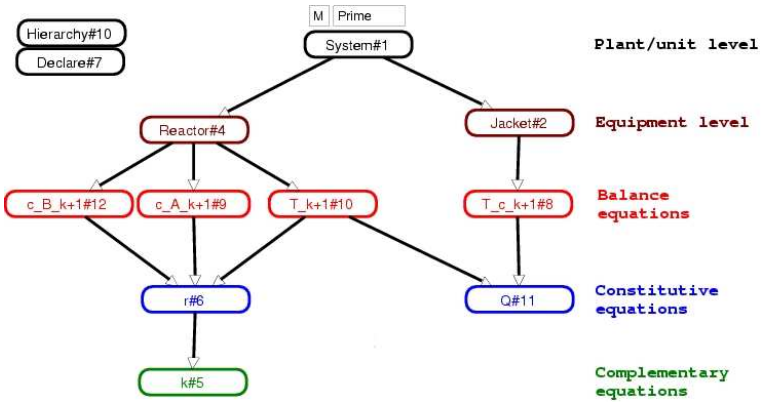


Figure 10: The modelling hierarchy represented by hierarchical CPN of the reactor

5.1.3 Goal hierarchy

The goal hierarchy of the tank reactor is showed in Fig. 11. For example, the goal G_3 requires temperature control of the reactor contents. This in turn requires an external cooling medium, G_7 and a means to control its flow G_8. Subgoals G_{10} and G_{11} demand a heat transfer surface and a cooling supply. The actual CSTR system as modelled by components would involve a heat transfer area A associated with the vessel-jacket and a utility flow v_c and temperature T_c.

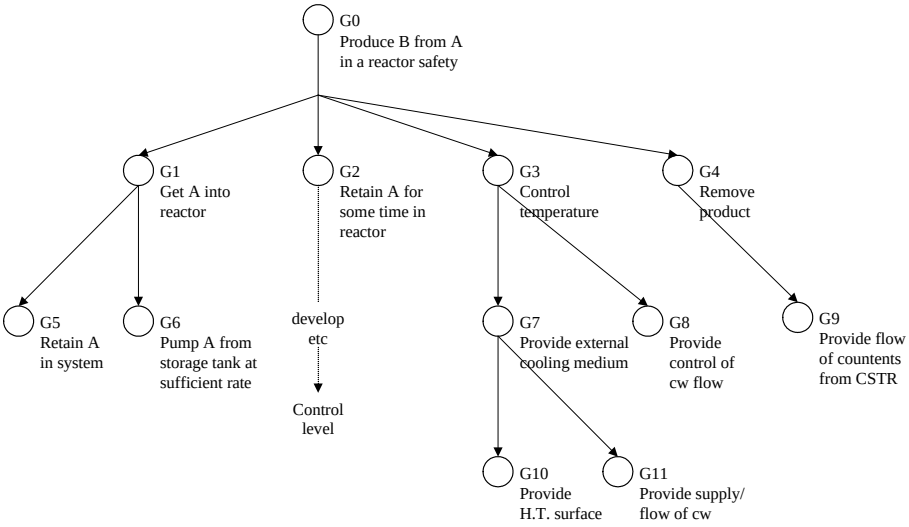


Figure 11: The goal hierarchy of the reactor

5.1.4 Connection between the modelling and goal hierarchies

The connection is described by functions and symptoms associated to them, and they can be seen in Fig. 12.

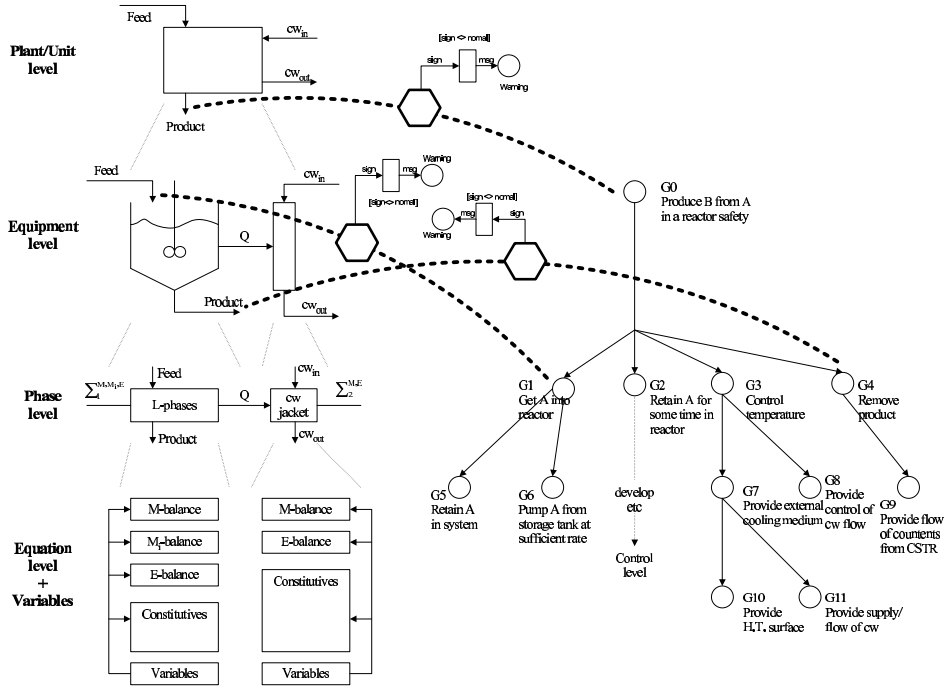


Figure 12: The connection between the modelling and goal hierarchies with symptoms in the reactor model

6 Conclusion and future work

A multi-scale goal-driven CPN model framework is proposed in this report for fault detection and diagnosis. The hierarchy levels, as well as the connections between the modelling and goal hierarchies are described together with the connecting point for the diagnosis.

A simple CSTR system is used to illustrate the concepts.

Acknowledgement

I. T. Cameron acknowledges support funding from the Australian Research Council Linkage Grant Scheme (LP0214142). Part of the research is sponsored by the Hungarian National Research Fund through grant no. T042710.

References

[1] M. Lind. Modelling goals and functions of complex plant. *Journal of Applied Artificial Intelligence*, 8(2):259–283, 1994.

[2] T. Cadman Modarres, M. A method of alarm system analysis for process plants. *Comput. Chemical Engineering*, 10:557–565, 1986.

[3] M. Lind. Plant modeling for human supervisory control. *Transactions of the Institute of Measurement and Control*, 21:171–180, 1999.

[4] A. Jalashgar. Goal-oriented systems modelling: justification of the approach and overview of methods. *Reliability Engineering and System Safety*, 64:271–278, 1999.

[5] S. W. Cheon Modarres, M. Function-centered modeling of engineering systems using the goal tree-success tree technique and functional primitives. *Reliability Engineering and System Safety*, 64:181–200, 1999.

[6] Kurt Jensen, editor. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use Volume 1.* Springer-Verlag, 1992.

[7] Kurt Jensen, editor. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use Volume 2.* Springer-Verlag, 1995.

[8] Kurt Jensen and Grzegorz Rosenberg, editors. *High-level Petri nets: Theory and Application.* Springer-Verlag, 1991.

[9] K.M. Hangos and I.T. Cameron, editors. *Process modelling and model analysis.* Academic Press, 2001.

[10] R. Lakner, K.M. Hangos, and I.A. Cameron. Construction of minimal models for control purposes. *Computer-Aided Chemical Engineering*, 11:49–57, 2003.

[11] W3 Consortium. Owl web ontology language semantics and abstract syntax. *http://www.w3.org/TR/2003/CR-owl-semantics-20030818/*, 2003.

[12] B. Bayer, C. Krobb, and W. Marquardt. A data model for design data in chemical engineering - information models. Technical Report LPT-2001-15, RWTH, Aachen, 2001.